Traffic

July 2, 2023

[1]:
```
# !pip install otter-grader
# !pip install --upgrade pip
# !pip install contextily==1.2.0
# !pip install pandas fiona shapely pyproj rtree
# !pip install geopandas
```

[2]:
```
# Initialize Otter
import otter
grader = otter.Notebook("Traffic.ipynb")
```

1 Final Project: Traffic

1.1 Due Date: Monday, December 13th, 11:59 PM

1.2 Collaboration Policy

Data science is a collaborative activity. While you may talk with other groups about the project, we ask that you **write your solutions within your own groups**. If you do discuss the assignments with others outside of your group please **include their names** at the top of your notebook.

2 Data 100 Final Project: Traffic in a post-lockdown world

Scenario: You're a data scientist at Uber – sitting in a war room on March 16, 2020, 1 day after California-wide COVID lockdown measures began and the day shelter-in-place measures are announced in the bay area. The entire data science department is on fire: All of your existing traffic models have regressed *significantly*. Given the sudden change in traffic patterns (i.e., no traffic at all), the company's traffic estimates are wildly incorrect. This is a top priority for the company. Since traffic estimates are used directly for pricing strategies, this is actively costing the company millions every hour. You are tasked with fixing these models.

Takeaways: How do you "fix" models that have learned biases from pre-lockdown traffic? How do you train new ones, with just 24 hours of data? What sorts of data do you examine, to better understand the situation? In the midst of company-wide panic, you'll need a strong inferential acumen to lead a robust data science response. In this project, we'll walk you through a simulated war room data science effort, culminating in some strategies to fix models online, which are experiencing large distributional shifts in data.

For this project, we'll explore traffic data provided by the **Uber Movement** dataset, specifically around the start of COVID shutdowns in March 2020. Your project is structured around the following ideas:

```
1. Guided data cleaning: Clustering data spatially
    a. Load Uber traffic speeds dataset
    b. Map traffic speeds to Google Plus Codes (spatially uniform)
        i. Load node-to-gps-coordinates data
        ii. Map traffic speed to GPS coordinates
        iii. Convert GPS coordinates to plus code regions
        iv. Sanity check number of plus code regions in San Francisco
        v. Plot a histogram of the standard deviation in speed, per plus code region.
    c. Map traffic speeds to census tracts (spatially non-uniform)
        i. Download census tracts geojson
        ii. Map traffic speed to census tracts
        iii. Sanity check number of census tracts in San Francisco with data.
        iv. Plot a histogram of the standard deviation in speed, per census tract.
    d. What defines a "good" or "bad" spatial clustering?
2. Guided EDA: Understanding COVID lockdown impact on traffic
    a. How did lockdown affect average traffic speeds?
        i. Sort census tracts by average speed, pre-lockdown.
        ii. Sort census tracts by average speed, post-lockdown.
        iii. Sort census tracts by change in average speed, from pre to post lockdown.
        iv. Quantify the impact of lockdown on average speeds.
        v. Quantify the impact of pre-lockdown average speed on change in speed.
    b. What traffic areas were impacted by lockdown?
        i. Visualize heatmap of average traffic speed per census tract, pre-lockdown.
        ii. Visualize change in average daily speeds pre vs. post lockdown.
        iii. Quantify the impact of lockdown on daily speeds, spatially.
3. Open-Ended EDA: Understanding lockdown impact on traffic times
    a. Download Uber Movement (Travel Times) dataset
4. Guided Modeling: Predict traffic speed post-lockdown
    a. Predict daily traffic speed on pre-lockdown data
        i. Assemble dataset to predict daily traffic speed.
        ii. Train and evaluate linear model on pre-lockdown data.
    b. Understand failures on post-lockdown data
        i. Evaluate on post-lockdown data
        ii. Report model performance temporally
    c. "Fix" model on post-lockdown data
        i. Learn delta off of a moving bias
        ii. Does it "solve itself"? Does the pre-lockdown model predict, after the change poin
        iii. Naively retrain model with post-lockdown data
        iv. What if you just ignore the change point?
5. Open-Ended Modeling: Predicting travel times post-lockdown
```

Concepts tested: regex, pivot, join, grouping, inferential thinking

[3]:
```
import pandas as pd
import geopandas as gpd
import numpy as np
```

```
import csv
import json
import os
import contextily as cx
from collections import defaultdict
import re
from typing import Callable

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

from zipfile import ZipFile
zf = ZipFile('data.zip', 'r')
zf.extractall('.')

# more readable exceptions
%pip install --quiet iwut
%load_ext iwut
%wut on
```

```
Note: you may need to restart the kernel to use updated packages.
```

[4]:
```
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score, mean_squared_error
import seaborn as sns
from sklearn.linear_model import LogisticRegression
```

3 Step 1 - Guided Data Cleaning: Partitioning Data Spatially

Our hope is answer: How do we group information spatially? We'll specifically look at 2 ways of partitioning data spatially, to understand the impact of spatial partitioning strategies on our analyses:

1. Dividing the world uniformly into slices, like Google's plus codes.

2. Dividing the world according to population, using census tracts.

In this step, we'll load the following datasets that we'll need for this project:

- Daily travel times from Uber Movement data in March 2020 from San Francisco, by census tract
- Daily traffic speeds from Uber Movement data in Q1 2020 from San Francisco, between OSM nodes
- Census tracts dividing San Francisco by GPS coordinates
- Mapping from OSM nodes to GPS coordinates

There are several terms and concepts to get familiar with upfront:

- **Open Street Maps (OSM)** provides nodes (points in space, wiki) and ways (segments between nodes wiki). These IDs are used in the Uber Movement dataset to identify streets in the traffic speeds dataset.
- **Census Tracts** provided by the county of San Francisco geographically divides space according to the US 2010 Census. This is used in the Uber Movement dataset to identify regions of differing travel times.

3.1 1.a. Load Uber traffic speeds dataset

The dataset is located at `data/movement-speeds-daily-san-francisco-2020-3.csv`. **Load this dataset into a dataframe.**

The original dataset from Uber was provided hourly and took up 2.1 GB on disk, which means it couldn't fit into your 1GB of RAM. You can find the dataset preparation script at `data/PrepareTrafficDataset.ipynb` *which aggregated within each day, reducing the dataset to just 55MB on disk.*

This was originally going to be question in this project, but it takes 22 minutes to run. Better yet, if you mess up, your kernel dies and you start over. We deemed it too frustrating and preprocessed the dataset to spare you the pain... but just know that this is a real-world issue!

[5]:
```
# Load Uber Movement (Movement Speeds) dataset into dataframe
speeds_to_nodes = pd.read_csv('data/movement-speeds-daily-san-francisco-2020-3.
 ↪csv')

speeds_to_nodes.shape
```

[5]:
```
(1586652, 4)
```

[6]:
```
grader.check("q1a")
```

[6]:
```
q1a results: All test cases passed!
```

3.2 1.b. Map traffic speed to Google Plus Codes

Google Plus Codes divide up the world uniformly into rectangular slices (link). Let's use this to segment traffic speeds spatially. Take a moment to answer: **Is this spatial structure effective for summarizing traffic speed?** Before completing this section, substantiate your answer with examples of your expectations (e.g., we expect A to be separated from B). After completing this section, substantiate your answer with observations you've made.

Type your answer here, replacing this text.

3.2.1 1.b.i. Load Node-to-GPS-Coordinate Data

In this substep, we'll load a mapping from OSM nodes to GPS coordinates. The dataset is provided in a gzip'ed XML file from OpenStreetMaps (OSM). The mapping from OSM nodes to GPS coordinates was downloaded from https://download.bbbike.org/osm/bbbike/SanFrancisco/SanFrancisco.osm.gz. We've downloaded this for you, to avoid any issues with OSM updates.

If you try to load the provided `.osm` (an `.xml` in disguise) using Python's built-in XML utilities **(by uncommenting the last 2 lines in the below cell)**, you will hit an out-of-memory error, as your kernel is forced to restart.

[7]:
```
# [OSM] - Read the OSM XML and extract mapping from node ID to GPS coordinates
PATH_OSM = os.path.expanduser('data/SanFrancisco.osm')

# Runs out of memory! File itself is 430 MB, even when filtering out
# irrelevant rows, and remaining 3M rows are too expensive to parse,
# resulting in OOM

# import xml.etree.ElementTree as ET
# _tree = ET.parse(PATH_OSM)
```

Your above code hits a memory error, so instead, we will use our handy-dandy tool–regex–from earlier in the semester to load just the parts of the file that we need. **Given the XML snippet below, write a regex pattern to extract OSM node ID, latitude, and longitude.** (The first capture group should be node ID. The second should be latitude, and the third should be longitude.) A snippet of the XML is included below (screenshot):

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="osmconvert 0.8.3">
    <bounds minlat="42.4543" minlon="-2.4761999" maxlat="42.4..."/>
    <node id="26861066" lat="42.471111" lon="-2.454722" version="..."/>
        <tag k="name" v="Camping La Playa"/>
        <tag k="tourism" v="camp_site"/>
        <tag k="operator" v="private"/>
        ...
    </node>
    <node id="34793287" lat="42.4713587" lon="-2.4510783" version="..."/>
        <tag k="created_by" v="JOSM"/>
    </node>
    <node id="34793294" lat="42.4610836" lon="-2.4303622" version="..."/>
    <node id="34793297" lat="42.4548363" lon="-2.4287657" version="..."/>
    ...
</osm>
```

[8]:
```python
# [OSM] - Read the OSM XML using a regex operation instead.
def read_node_lat_lon(path: str, pattern: str, line_condition: Callable):
    """
    Read the provided path line at a line. If the provided regex pattern
    has a match, return the grouped matches as items in a generator.

    :param path: Path to read data from
    :param pattern: Regex pattern to test against each line
    :param line_condition: function that returns if we should check regex
        against current line
    """
    with open(path) as f:
        for line in f:
            result = re.search(pattern, line)
            if result is not None and line_condition(result):
                yield int(result.group(1)), float(result.group(2)),␣
 ↪float(result.group(3))
```

[9]:
```python
node_ids = set(speeds_to_nodes.osm_start_node_id) | set(speeds_to_nodes.
 ↪osm_end_node_id)

NODE_PATTERN = r"id=.(\d+).\slat=.(\d+.\d+).\slon=.(.\d+.\d+)"

node_to_gps = pd.DataFrame(read_node_lat_lon(
    PATH_OSM,
    pattern=NODE_PATTERN,
    line_condition=lambda result: int(result.group(1)) in node_ids
), columns=['osm_node_id', 'Latitude', 'Longitude'])
node_to_gps
```

[9]:

	osm_node_id	Latitude	Longitude
0	26118026	37.675280	-122.389194
1	29891973	37.674935	-122.389130
2	29892598	37.716892	-122.398893
3	30033679	37.599877	-122.376497
4	30033686	37.642167	-122.405946
…	…	…	…
19139	6522255428	37.760543	-122.443563
19140	6522255492	37.759317	-122.444996
19141	6522764204	37.762163	-122.436143
19142	6522764212	37.756061	-122.436761
19143	6522764213	37.761187	-122.440089

[19144 rows x 3 columns]

[10]:
```python
grader.check("q1bi")
```

[10]: q1bi results: All test cases passed!

3.2.2 1.b.ii. Map traffic speed to GPS coordinates.

Traffic speeds are currently connected to OSM nodes. You will then use the mapping from OSM nodes to GPS coordinates, to map traffic speeds to GPS coordinates. **Link each traffic speed measurement to the GPS coordinate of its starting node.**

Note: For simplicity, assume each segment is associated with the node it *starts* with.

Hint: Not all nodes are included in the OSM node mapping. Make sure to ignore any nodes without valid GPS coordinates.

[11]:
```
# Find mapping from traffic speeds to GPS coordinates
speeds_to_gps = speeds_to_nodes.merge(node_to_gps, left_on='osm_start_node_id',␣
↪right_on='osm_node_id')
speeds_to_gps
```

[11]:
```
        osm_start_node_id  osm_end_node_id  day  speed_mph_mean  osm_node_id  \
0                26118026        259458979    1       64.478000     26118026
1                26118026        259458979    2       62.868208     26118026
2                26118026        259458979    3       62.211750     26118026
3                26118026        259458979    4       62.192458     26118026
4                26118026        259458979    5       61.913292     26118026
…                       …                …    …               …            …
417634         4069109544        615120176   30       38.956000   4069109544
417635         5448539901         65446993   16       25.627000   5448539901
417636          302964668       4069109544   19       40.802000    302964668
417637          302964668       4069109544   20       36.076000    302964668
417638         5022068066        302964668   19       39.592000   5022068066

         Latitude   Longitude
0       37.675280 -122.389194
1       37.675280 -122.389194
2       37.675280 -122.389194
3       37.675280 -122.389194
4       37.675280 -122.389194
…             …           …
417634  37.732039 -122.507126
417635  37.622476 -122.413763
417636  37.732418 -122.507206
417637  37.732418 -122.507206
417638  37.733635 -122.507100

[417639 rows x 7 columns]
```

[12]:
```
grader.check("q1bii")
```

[12]: q1bii results: All test cases passed!

3.2.3 1.b.iii. Convert GPS coordinates to plus code regions.

Plus code regions divide up the world into uniformly-sized rectangles, which we will assume is 0.012 degrees latitudiunally and longitudinally. **For each traffic speed row, compute the plus code region it belongs to**, based on its GPS coordinates.

To do this, we suggest computing a latitudinal index `plus_latitude_idx` and a longitudinal index `plus_longitude_idx` for the plus code region each row belongs to. *Make sure these columns are integer-valued.*

Hint: If you're running into nans, you did 1.b.ii. incorrectly!

[13]:
```
lat_min = np.min(speeds_to_gps['Latitude'])
lat_max = np.max(speeds_to_gps['Latitude'])

lon_min = np.min(speeds_to_gps['Longitude'])
lon_max = np.max(speeds_to_gps['Longitude'])

lat_bins = np.arange(-180, 180, 0.012)
lon_bins = np.arange(-180, 180, 0.012)

lat_indices = np.digitize(speeds_to_gps['Latitude'], lat_bins)
lon_indices = np.digitize(speeds_to_gps['Longitude'], lon_bins)

speeds_to_gps['plus_latitude_idx'] = lat_indices
speeds_to_gps['plus_longitude_idx'] = lon_indices

plus_code = [str(m) + "_" + str(n) for m,n in␣
 ↪zip(speeds_to_gps['plus_latitude_idx'],speeds_to_gps['plus_longitude_idx'])]

speeds_to_gps['plus_code'] = plus_code # do this however you like

speeds_to_gps
```

[13]:
```
        osm_start_node_id  osm_end_node_id  day  speed_mph_mean  osm_node_id  \
0                26118026        259458979    1       64.478000     26118026
1                26118026        259458979    2       62.868208     26118026
2                26118026        259458979    3       62.211750     26118026
3                26118026        259458979    4       62.192458     26118026
4                26118026        259458979    5       61.913292     26118026
…                       …                …  …               …            …
417634         4069109544        615120176   30       38.956000   4069109544
417635         5448539901         65446993   16       25.627000   5448539901
417636          302964668       4069109544   19       40.802000    302964668
417637          302964668       4069109544   20       36.076000    302964668
417638         5022068066        302964668   19       39.592000   5022068066
```

```
         Latitude   Longitude  plus_latitude_idx  plus_longitude_idx  \
0       37.675280 -122.389194              18140                4801
1       37.675280 -122.389194              18140                4801
2       37.675280 -122.389194              18140                4801
3       37.675280 -122.389194              18140                4801
4       37.675280 -122.389194              18140                4801
…             …           …                 …                  …
417634  37.732039 -122.507126              18145                4792
417635  37.622476 -122.413763              18136                4799
417636  37.732418 -122.507206              18145                4792
417637  37.732418 -122.507206              18145                4792
417638  37.733635 -122.507100              18145                4792

         plus_code
0       18140_4801
1       18140_4801
2       18140_4801
3       18140_4801
4       18140_4801
…              …
417634  18145_4792
417635  18136_4799
417636  18145_4792
417637  18145_4792
417638  18145_4792

[417639 rows x 10 columns]
```

[14]:
```python
grader.check("q1biii")
```

[14]:
```
q1biii results: All test cases passed!
```

3.2.4 1.b.iv. Sanity check number of plus code regions in San Francisco.

Compute the number of unique plus codes found in your dataset. You're checking that the number isn't ridiculous, like 1, or 100,000 (SF is 231 sq mi, so 100k tracts would average 12 sq ft per tract).

If you followed the suggestion above, this is the number of unique (`plus_latitude_idx`, `plus_longitude_idx`) pairs.

[15]:
```python
# You're expecting 276 plus codes here. Don't just type "276"
# below to pass the autograder. The goal is to sanity check your
# dataframe!
num_pluscode_regions = len(np.unique(plus_code))
num_pluscode_regions
```

[15]: 276

[16]:
```
grader.check("q1biv")
```

[16]: q1biv results: All test cases passed!

3.2.5 1.b.v. How well do plus code regions summarize movement speeds?

The following will give us an idea of how well the average represents traffic speed per plus code region. For these questions, we'll refer to a "plus code region" as a "cluster":

1. **Plot a histogram of the within-cluster standard deviation**.
2. **Compute across-cluster average of within-cluster standard deviation**.
3. **Compute across-cluster standard deviation of within-cluster average speeds**.
4. **Is this average variance reasonable?** To assess what "reasonable" means, consider these questions and how to answer them: (1) Do plus codes capture meaningful subpopulations? (2) Do differences between subpopulations outweigh differences within a subpopulation? Use the statistics above to answer these questions, and compute any additional statistics you need. Additionally explain *why these questions are important to assessing the quality of a spatial clustering*.

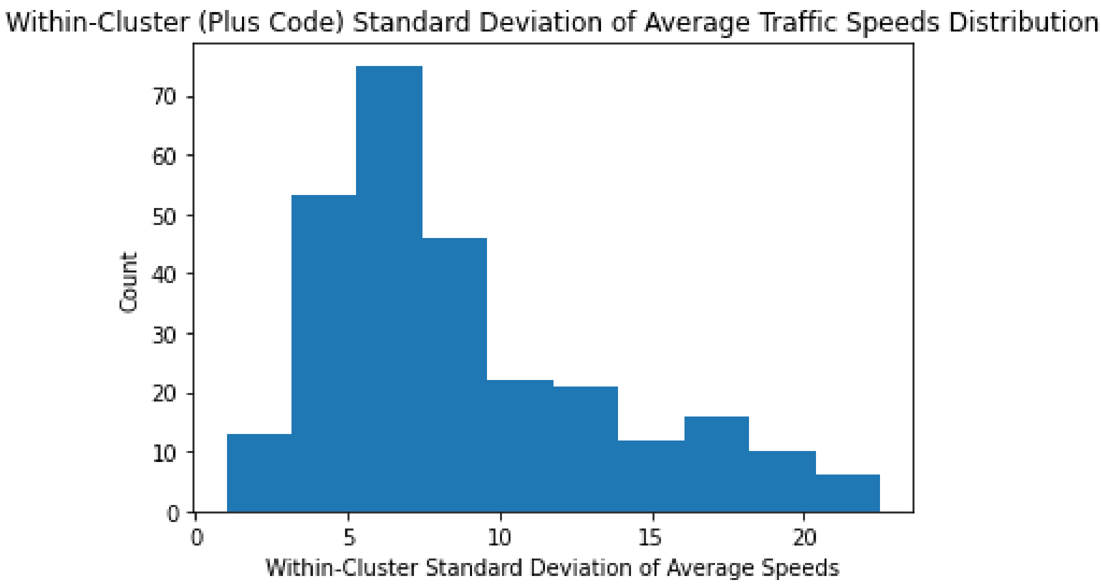
Hint: Run the autograder first to ensure your variance average and average variance are correct, before starting to draw conclusions.

In the first cell, write your written answers. In the second cell, complete the code.

Type your answer here, replacing this text.

[17]:
```
speed_variance_by_pluscode = speeds_to_gps.
 ↪groupby('plus_code')['speed_mph_mean'].agg(np.std)
plt.title('Within-Cluster (Plus Code) Standard Deviation of Average Traffic␣
 ↪Speeds Distribution')
plt.xlabel('Within-Cluster Standard Deviation of Average Speeds')
plt.ylabel('Count')
plt.hist(speed_variance_by_pluscode)
average_variance_by_pluscode = (speed_variance_by_pluscode).mean()
variance_average_by_pluscode = (speeds_to_gps.
 ↪groupby('plus_code')['speed_mph_mean'].agg(np.mean)).std()
average_variance_by_pluscode, variance_average_by_pluscode
```

[17]: (8.684748294968637, 10.13573858675904)



[18]:
```
speed_variance_by_pluscode
```

[18]:
```
plus_code
18129_4791     3.650232
18129_4803    20.704487
18129_4807    12.342234
18130_4791     4.083821
18130_4802    21.390912
                 …
18161_4803    15.293632
18161_4804    17.915911
18161_4805    14.243592
18161_4806    12.237229
18161_4807    17.590817
Name: speed_mph_mean, Length: 276, dtype: float64
```

[19]:
```
grader.check("q1bv3")
```

[19]:
```
q1bv3 results: All test cases passed!
```

3.3 1.c. Map traffic speed to census tract.

Census tracts divide the space much less uniformly, subdividing regions that we were interested in into smaller zones. This suggests promise in providing informative spatial segments. Note that the daily traffic speeds are provided between OpenStreetMap (OSM) nodes, so we'll need to map nodes to census tracts somehow.

Above, we've mapped traffic speeds to GPS coordinates. Below, we'll then link GPS coordinates to census tracts, to complete the mapping from traffic speeds to census tracts.

3.3.1 1.c.i. Download Census Tracts Geojson

Load the census tracts geojson. Make sure to see the relevant geopandas io documentation to see how to load a geojson.

Hint: It should take you just one line to load.

[20]:
```
PATH_TRACTS = os.path.expanduser('data/san_francisco_censustracts.json')
tract_to_gps = gpd.read_file(PATH_TRACTS)
tract_to_gps['MOVEMENT_ID'] = tract_to_gps['MOVEMENT_ID'].astype(int)
tract_to_gps.head(10)
```

[20]:
```
   MOVEMENT_ID                                       DISPLAY_NAME  \
0            1                              Sargent Creek, San Ardo
1            2  400 Northumberland Avenue, Redwood Oaks, Redwo…
2            3                     18300 Sutter Boulevard, Morgan Hill
3            4                2700 Stoughton Way, Sheffield, Sacramento
4            5             3200 Huntsman Drive, Rosemont Park, Sacramento
5            6                            100 Carlsbad Circle, Vacaville
6            7                                Unnamed Road, Vacaville
7            8                              700 Carlsbad Court, Petaluma
8            9              500 Hyde Street, Tenderloin, San Francisco
9           10                          3200 Nightingale Drive, Modesto

                                            geometry
0  MULTIPOLYGON (((-121.59511 36.11126, -121.5401…
1  MULTIPOLYGON (((-122.22463 37.46507, -122.2236…
2  MULTIPOLYGON (((-121.67978 37.15859, -121.6719…
3  MULTIPOLYGON (((-121.35921 38.57175, -121.3462…
4  MULTIPOLYGON (((-121.37512 38.55309, -121.3715…
5  MULTIPOLYGON (((-121.96392 38.36476, -121.9575…
6  MULTIPOLYGON (((-122.02388 38.32508, -122.0170…
7  MULTIPOLYGON (((-122.65880 38.26414, -122.6579…
8  MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
9  MULTIPOLYGON (((-121.07441 37.69998, -121.0690…
```

[21]:
```
grader.check("q1ci")
```

[21]:
```
q1ci results: All test cases passed!
```

3.3.2 1.c.ii Map traffic speed to census tracts.

You will need to *spatially join* the (1) mapping from traffic speed to GPS coordinates `speed_to_gps` and (2) the mapping from GPS coordinates to boundaries of census tracts `tract_to_gps` to group all traffic speeds by census tract. This "spatial join" is an advanced feature recently released (as of time of writing, in Oct 2021) in geopandas, which allows us to connect single points to their enclosing polygons. You will do this question in 3 parts:

1. Convert the last dataframe `speeds_to_gps` into a geopandas dataframe `speeds_to_points`, where GPS coordinates are now geopandas points. See this tutorial: https://geopandas.org/gallery/create_geopandas_from_pandas.html#From-longitudes-and-latitudes
2. Set the coordinate-system for the new geopandas dataframe to the "world geodesic system" link, or in other words, the coordinate system that GPS coordinates are reported in.
3. Compute a spatial join between census tracts `tract_to_gps` and the geopandas traffic speeds `speeds_to_points`

[22]:
```
# !pip install pygeos
# !pip install rtree
# !pip install --upgrade pip
```

[23]:
```
type(speeds_to_gps)
type(tract_to_gps)
```

[23]:
```
geopandas.geodataframe.GeoDataFrame
```

[24]:
```
speeds_to_points = gpd.GeoDataFrame(speeds_to_gps, geometry=gpd.
 ↪points_from_xy(speeds_to_gps.Longitude, speeds_to_gps.Latitude))
speeds_to_points.set_crs(epsg=4326, inplace=True)
speeds_to_tract = speeds_to_points.sjoin(tract_to_gps, how='left')

speeds_to_tract
```

[24]:
```
        osm_start_node_id  osm_end_node_id  day  speed_mph_mean  osm_node_id  \
0                26118026        259458979    1       64.478000     26118026
1                26118026        259458979    2       62.868208     26118026
2                26118026        259458979    3       62.211750     26118026
3                26118026        259458979    4       62.192458     26118026
4                26118026        259458979    5       61.913292     26118026
…                     …              …  …             …           …
417634         4069109544        615120176   30       38.956000   4069109544
417635         5448539901         65446993   16       25.627000   5448539901
417636          302964668       4069109544   19       40.802000    302964668
417637          302964668       4069109544   20       36.076000    302964668
417638         5022068066        302964668   19       39.592000   5022068066

         Latitude   Longitude  plus_latitude_idx  plus_longitude_idx  \
0       37.675280 -122.389194              18140                4801
1       37.675280 -122.389194              18140                4801
2       37.675280 -122.389194              18140                4801
3       37.675280 -122.389194              18140                4801
4       37.675280 -122.389194              18140                4801
…             …          …                  …                   …
417634  37.732039 -122.507126              18145                4792
417635  37.622476 -122.413763              18136                4799
```

```
417636  37.732418 -122.507206            18145               4792
417637  37.732418 -122.507206            18145               4792
417638  37.733635 -122.507100            18145               4792

          plus_code                      geometry  index_right  MOVEMENT_ID  \
0        18140_4801  POINT (-122.38919 37.67528)         1729         1730
1        18140_4801  POINT (-122.38919 37.67528)         1729         1730
2        18140_4801  POINT (-122.38919 37.67528)         1729         1730
3        18140_4801  POINT (-122.38919 37.67528)         1729         1730
4        18140_4801  POINT (-122.38919 37.67528)         1729         1730
…               …                           …           …          …
417634   18145_4792  POINT (-122.50713 37.73204)         1778         1779
417635   18136_4799  POINT (-122.41376 37.62248)         1456         1457
417636   18145_4792  POINT (-122.50721 37.73242)         1778         1779
417637   18145_4792  POINT (-122.50721 37.73242)         1778         1779
417638   18145_4792  POINT (-122.50710 37.73363)         1778         1779

                                          DISPLAY_NAME
0                                0 Park Lane, Brisbane
1                                0 Park Lane, Brisbane
2                                0 Park Lane, Brisbane
3                                0 Park Lane, Brisbane
4                                0 Park Lane, Brisbane
…                                                  …
417634   500 John Muir Drive, Lakeshore, San Francisco
417635                    1500 Donner Avenue, San Bruno
417636   500 John Muir Drive, Lakeshore, San Francisco
417637   500 John Muir Drive, Lakeshore, San Francisco
417638   500 John Muir Drive, Lakeshore, San Francisco

[418848 rows x 14 columns]
```

[25]:
```python
grader.check("q1cii")
```

[25]: q1cii results: All test cases passed!

3.3.3 1.c.iii. Aggregate movement speeds by census tract.

- Create a new dataframe `speeds_by_tract` to group movement speeds by census tract. See the outputted dataframe from 1.c.i. to check how census tracts are identified.
- Always double-check your numbers. **Report the number of census tracts** in your dataset.

[26]:
```python
speeds_by_tract = speeds_to_tract.groupby('MOVEMENT_ID')
num_census_tracts = len(speeds_by_tract)
num_census_tracts
```

[26]: 295

[27]:
```
grader.check("q1ciii")
```

[27]: q1ciii results: All test cases passed!

3.3.4 1.c.iv. How well do census tracts summarize movement speeds?

The following will give us an idea of how well the average represents traffic speed per plus code region. For these questions, we'll refer to a "census tract" as a "cluster":

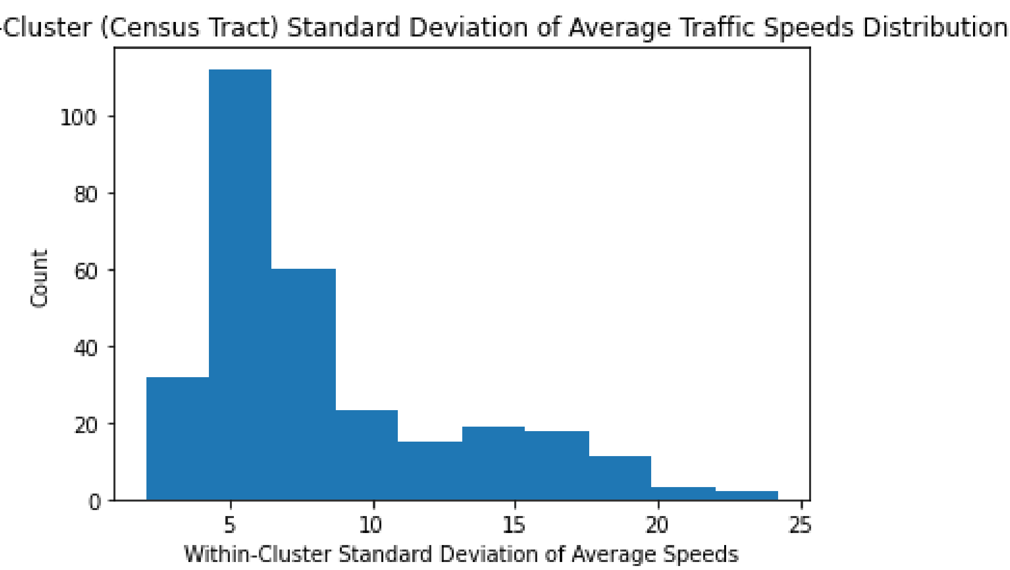
1. **Plot a histogram of the within-cluster standard deviation**.
2. **Compute across-cluster average of within-cluster standard deviation**.
3. **Compute across-cluster standard deviation of within-cluster average speeds**.
4. **Is this average variance reasonable?** To assess what "reasonable" means, consider these questions and how to answer them: (1) Do plus codes capture meaningful subpopulations? (2) Do differences between subpopulations outweigh differences within a subpopulation? Use these ideas to assess whether the average standard deviation is high or not.

Note: We are using the speed metric of miles per hour here.

Just like before, please written answers in the first cell and coding answers in the second cell.

Type your answer here, replacing this text.

[28]:
```
speed_variance_by_tract = speeds_by_tract['speed_mph_mean'].std()
average_variance_by_tract = speed_variance_by_tract.mean()
variance_average_by_tract = speeds_by_tract['speed_mph_mean'].agg(np.mean).std()
plt.title('Within-Cluster (Census Tract) Standard Deviation of Average Traffic␣
↪Speeds Distribution')
plt.xlabel('Within-Cluster Standard Deviation of Average Speeds')
plt.ylabel('Count')
plt.hist(speed_variance_by_tract);
```



[29]:
```
speed_variance_by_tract
```

[29]:
```
MOVEMENT_ID
9        3.821144
20       5.522853
21       3.640453
44       6.634154
78       3.838873
           …
2691     3.379664
2694     5.787065
2695     4.617596
2700    13.191079
2708     7.136608
Name: speed_mph_mean, Length: 295, dtype: float64
```

[30]:
```
grader.check("q1civ3")
```

[30]:
```
q1civ3 results: All test cases passed!
```

3.4 1.d. What would be the ideal spatial clustering?

This is an active research problem in many spatiotemporal modeling communities, and there is no single agreed-upon answer. Answer both of the following specifically knowing that you'll need to analyze traffic patterns according to this spatial clustering:

1. **What is a good metric for a spatial structure?** How do we define good? Bad? What information do we expect a spatial structure to yield? Use the above parts and questions to help answer this.
2. **What would you do to optimize your own metric for success in a spatial structure?**

See related articles:

- Uber's H3 link, which divides the world into hexagons
- Traffic Analysis Zones (TAZ) link, which takes census data and additionally accounts for vehicles per household when dividing space

Type your answer here, replacing this text.

4 Step 2 - Guided EDA: Understanding COVID Lockdown Impact on Traffic

In this step, we'll examine the impact of COVID on traffic. In particular, we'll study 3 different questions:

- How did lockdown affect traffic speed? What factors dictate how much lockdown affected traffic speed?

- What areas of traffic were most impacted by lockdown?

4.1 2.a. How did lockdown affect traffic speed?

4.1.1 2.a.i. Sort census tracts by average speed, pre-lockdown.

Consider the pre-lockdown period to be March 1 - 13, before the first COVID-related restrictions (travel bans) were announced on March 14, 2020.

1. **Report a DataFrame which includes the *names* of the 10 census tracts with the lowest average speed**, along with the average speed for each tract.
2. **Report a DataFrame which includes the *names* of the 10 census tracts with the highest average speed**, along with the average speed for each tract.
3. Do these names match your expectations for low speed or high speed traffic pre-lockdown? What relationships do you notice? (What do the low-speed areas have in common? The high-speed areas?) For this specific question, answer qualitatively. No need to quantify. **Hint**: Look up some of the names on a map, to understand where they are.
4. **Plot a histogram for all average speeds, pre-lockdown**.
5. You will notice a long tail distribution of high speed traffic. What do you think this corresponds to in San Francisco? Write down your hypothesis.

Hint: To start off, think about what joins may be useful to get the desired DataFrame.

Type your answer here, replacing this text.

Answer the following question:

[31]:
```
# compute the average speed per census tract (will use this later),
# BEFORE the shelter-in-place was announced on March 14, 2020.
# Autograder expects this to be a series
averages_pre1 = speeds_to_tract[speeds_to_tract['day'] < 14].
↪groupby('MOVEMENT_ID', as_index=False).agg(np.mean)
averages_pre = averages_pre1.set_index('MOVEMENT_ID')['speed_mph_mean']
# Autograder expects this to be a dataframe with name of census tract,
# polygon for census tract, and average speed per census tract
averages_pre_named = averages_pre1.merge(tract_to_gps)[['DISPLAY_NAME',␣
↪'speed_mph_mean', 'geometry']]
averages_pre_named
```

[31]:
```
                                          DISPLAY_NAME  speed_mph_mean  \
0            500 Hyde Street, Tenderloin, San Francisco       14.585102
1        900 Sutter Street, Lower Nob Hill, San Francisco       15.679922
2      3400 Pierce Street, Marina District, San Franc…      14.292445
3            1700 Egbert Avenue, Bayview, San Francisco       23.353083
4            1400 Thomas Avenue, Bayview, San Francisco       16.213552
..                                                 …             …
290                           800 Hacienda Way, Millbrae       20.746333
291  1900 Buchanan Street, Western Addition, San Fr…      17.042386
292    2200 Rivera Street, Sunset District, San Franc…      20.029011
293    300 Ponderosa Road, Avalon, South San Francisco       32.184422
```

```
294  200 Westview Drive, Sunshine Gardens, South Sa…       24.822808

                                              geometry
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
1    MULTIPOLYGON (((-122.42208 37.78847, -122.4153…
2    MULTIPOLYGON (((-122.44191 37.80374, -122.4371…
3    MULTIPOLYGON (((-122.40211 37.72779, -122.3998…
4    MULTIPOLYGON (((-122.39270 37.72928, -122.3918…
..                                                 …
290  MULTIPOLYGON (((-122.42288 37.60714, -122.4187…
291  MULTIPOLYGON (((-122.43549 37.78870, -122.4338…
292  MULTIPOLYGON (((-122.49505 37.74968, -122.4858…
293  MULTIPOLYGON (((-122.44834 37.64598, -122.4460…
294  MULTIPOLYGON (((-122.45179 37.66912, -122.4506…

[295 rows x 3 columns]
```

[32]:
```
grader.check("q2ai2")
```

[32]:
```
q2ai2 results: All test cases passed!
```

Report the lowest 10 census tracts with the lowest average speed Remember we want the NAME of each census tract too. For the autograder, please keep the name of the speed field, `speed_mph_mean`.

[33]:
```
bottom10_averages_pre = averages_pre_named.sort_values('speed_mph_mean',␣
↪ascending = True).head(10)
bottom10_averages_pre
```

[33]:
```
                                          DISPLAY_NAME  speed_mph_mean  \
166    200 O'Farrell Street, Tenderloin, San Francisco       12.417079
249          0 Mason Street, Tenderloin, San Francisco       12.595120
163        1100 Taylor Street, Nob Hill, San Francisco       12.945291
59   2900 22nd Street, Mission District, San Francisco       13.195865
51         200 Myrtle Street, Tenderloin, San Francisco       13.490311
164  200 Sutter Street, Financial District, San Fra…       13.502505
99         800 Jackson Street, Chinatown, San Francisco       13.549474
100         500 Geary Street, Tenderloin, San Francisco       13.570625
52          200 Jones Street, Tenderloin, San Francisco       13.626251
158          200 Hyde Street, Tenderloin, San Francisco       13.944773

                                              geometry
166  MULTIPOLYGON (((-122.41462 37.78558, -122.4129…
249  MULTIPOLYGON (((-122.41405 37.78279, -122.4107…
163  MULTIPOLYGON (((-122.41629 37.79389, -122.4152…
59   MULTIPOLYGON (((-122.41672 37.75717, -122.4123…
51   MULTIPOLYGON (((-122.42146 37.78663, -122.4182…
164  MULTIPOLYGON (((-122.40879 37.79016, -122.4071…
```

```
99   MULTIPOLYGON (((-122.41172 37.79629, -122.4084…
100  MULTIPOLYGON (((-122.41500 37.78745, -122.4133…
52   MULTIPOLYGON (((-122.41443 37.78466, -122.4127…
158  MULTIPOLYGON (((-122.41771 37.78424, -122.4160…
```

[34]:
```
grader.check("q2ai3")
```

[34]:
```
q2ai3 results: All test cases passed!
```

Report the highest 10 census tracts with the highest average speed.

[35]:
```
top10_averages_pre = averages_pre_named.sort_values('speed_mph_mean', ascending␣
↪= True).tail(10)
top10_averages_pre
```

[35]:
```
                                        DISPLAY_NAME  speed_mph_mean  \
198              600 San Bruno Avenue East, San Bruno       38.944079
191             0 Longview Drive, Westlake, Daly City       40.587037
222                       Liccicitos Road, Moss Beach       42.784267
288                        0 Burgess Court, Sausalito       43.848188
231  0 Crystal Springs Terrace, Hillsborough Park, …      44.304919
199                        1200 Helen Drive, Millbrae       45.492292
248              Frenchmans Creek Road, Half Moon Bay       47.225137
155                        Petrolite Street, Richmond       47.318340
36                4200 Shelter Creek Lane, San Bruno       53.867847
23                     1600 Maritime Street, Oakland       59.498552

                                            geometry
198  MULTIPOLYGON (((-122.41676 37.63935, -122.4115…
191  MULTIPOLYGON (((-122.50053 37.70083, -122.4961…
222  MULTIPOLYGON (((-122.52036 37.57534, -122.5180…
288  MULTIPOLYGON (((-122.52032 37.87046, -122.5193…
231  MULTIPOLYGON (((-122.37189 37.54776, -122.3710…
199  MULTIPOLYGON (((-122.42820 37.60497, -122.4263…
248  MULTIPOLYGON (((-122.46816 37.56079, -122.4605…
155  MULTIPOLYGON (((-122.42976 37.96540, -122.4185…
36   MULTIPOLYGON (((-122.43101 37.61999, -122.4300…
23   MULTIPOLYGON (((-122.33037 37.82058, -122.3161…
```

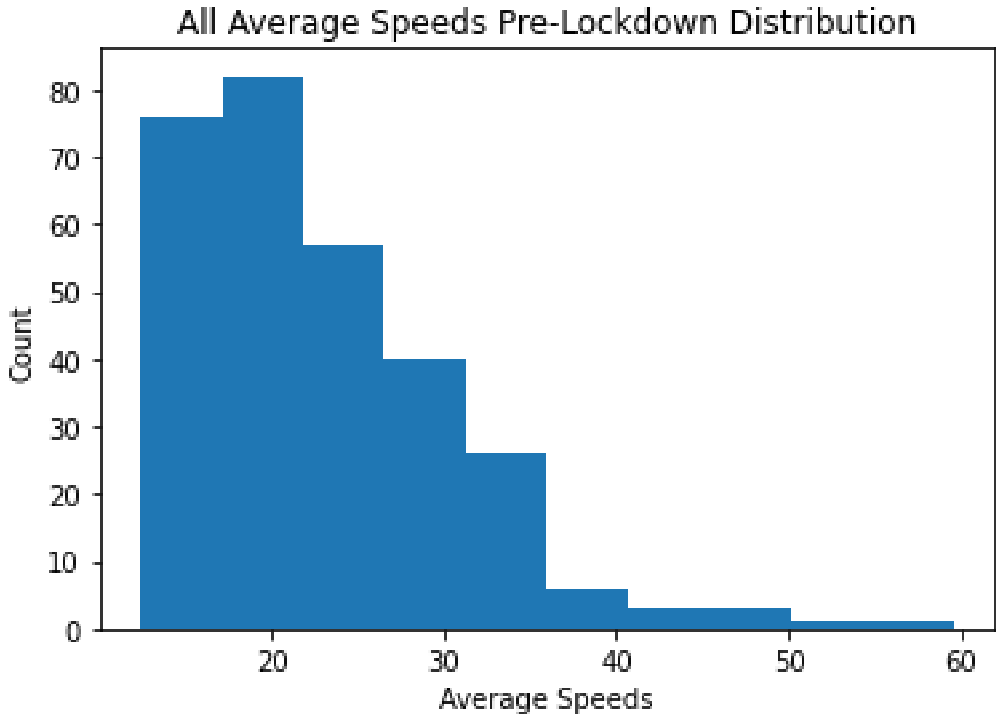
[36]:
```
grader.check("q2ai4")
```

[36]:
```
q2ai4 results: All test cases passed!
```

Plot the histogram

[37]:
```
plt.title('All Average Speeds Pre-Lockdown Distribution')
plt.xlabel('Average Speeds')
plt.ylabel('Count')
```

```
plt.hist(averages_pre_named['speed_mph_mean']);
```



4.1.2 2.a.ii. Sort census tracts by average speed, post-lockdown.

I suggest checking the top 10 and bottom 10 tracts by average speed, post-lockdown. Consider the post-lockdown period to be March 14 - 31, after the first COVID restrictions were established on March 14, 2020. It's a healthy sanity check. For this question, you should report:

- **Plot a histogram for all average speeds, post-lockdown.**
- **What are the major differences between this post-lockdown histogram relative to the pre-lockdown histogram above**? Anything surprising? What did you expect, and what did you find?

Write the written answers in the cell below, and the coding answers in the cells after that.

Type your answer here, replacing this text.

[38]:
```
# compute the average speed per census tract (will use this later),
# AFTER (and including) the first COVID restrictions were put into effect.
# Autograder expects this to be a series
averages_post1 = speeds_to_tract[speeds_to_tract['day'] >= 14].
 ↪groupby('MOVEMENT_ID', as_index = False).agg(np.mean)
averages_post = averages_post1.set_index('MOVEMENT_ID')['speed_mph_mean']
# Autograder expects this to be a dataframe with name of census tract,
# polygon for census tract, and average speed per census tract
```

```
averages_post_named = averages_post1.merge(tract_to_gps)[['DISPLAY_NAME',␣
↪'speed_mph_mean', 'geometry']]
averages_post_named
```

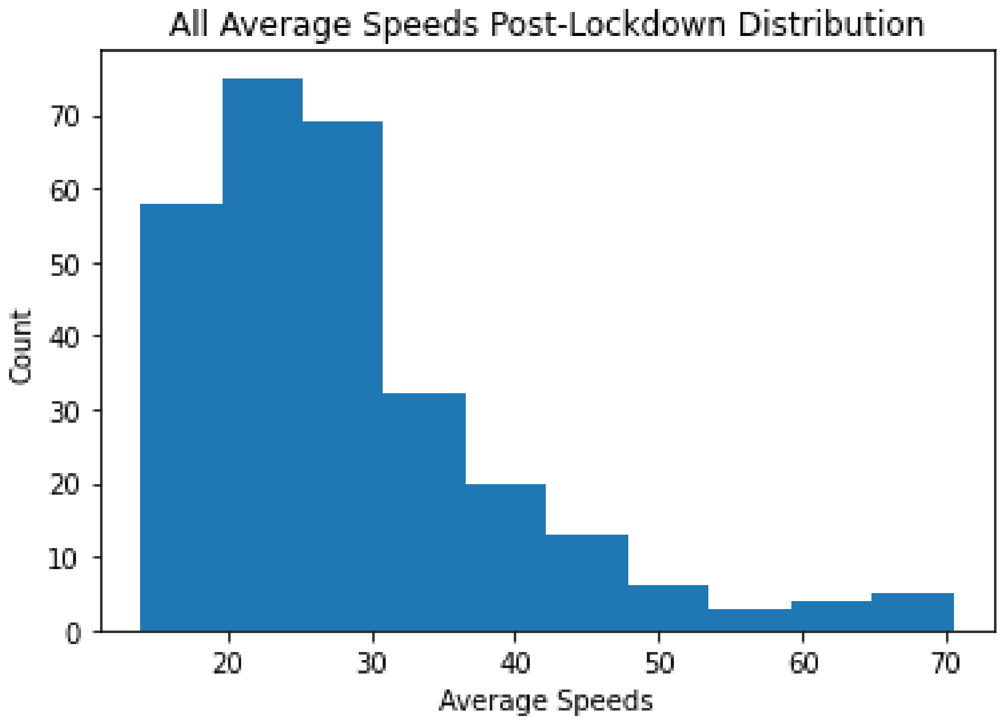
[38]:
```
                                          DISPLAY_NAME  speed_mph_mean  \
0            500 Hyde Street, Tenderloin, San Francisco       16.143154
1      900 Sutter Street, Lower Nob Hill, San Francisco       16.871488
2     3400 Pierce Street, Marina District, San Franc…       15.754795
3            1700 Egbert Avenue, Bayview, San Francisco       25.956602
4            1400 Thomas Avenue, Bayview, San Francisco       16.476000
..                                                 …               …
280                           800 Hacienda Way, Millbrae       17.917000
281  1900 Buchanan Street, Western Addition, San Fr…       22.128519
282   2200 Rivera Street, Sunset District, San Franc…       23.440404
283    300 Ponderosa Road, Avalon, South San Francisco       38.807594
284  200 Westview Drive, Sunshine Gardens, South Sa…       26.171347

                                              geometry
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
1    MULTIPOLYGON (((-122.42208 37.78847, -122.4153…
2    MULTIPOLYGON (((-122.44191 37.80374, -122.4371…
3    MULTIPOLYGON (((-122.40211 37.72779, -122.3998…
4    MULTIPOLYGON (((-122.39270 37.72928, -122.3918…
..                                                 …
280  MULTIPOLYGON (((-122.42288 37.60714, -122.4187…
281  MULTIPOLYGON (((-122.43549 37.78870, -122.4338…
282  MULTIPOLYGON (((-122.49505 37.74968, -122.4858…
283  MULTIPOLYGON (((-122.44834 37.64598, -122.4460…
284  MULTIPOLYGON (((-122.45179 37.66912, -122.4506…

[285 rows x 3 columns]
```

[39]:
```
grader.check("q2aii2")
```

[39]: q2aii2 results: All test cases passed!

Plot the histogram

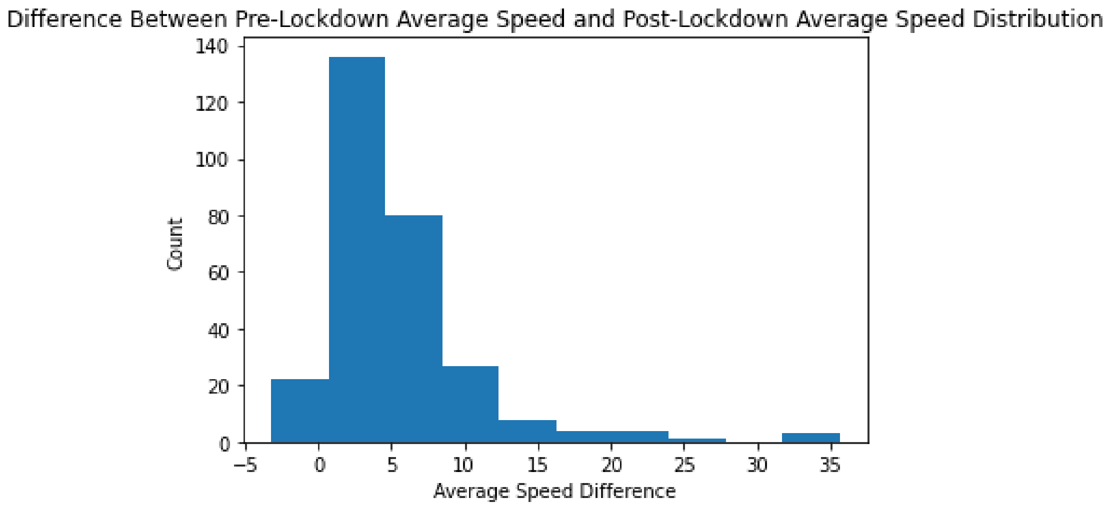
[40]:
```
plt.title('All Average Speeds Post-Lockdown Distribution')
plt.xlabel('Average Speeds')
plt.ylabel('Count')
plt.hist(averages_post_named['speed_mph_mean']);
```



4.1.3 2.a.iii. Sort census tracts by change in traffic speed from pre to post lockdown.

For each segment, compute the difference between the pre-lockdown average speed (March 1 - 13) and the post-lockdown average speed (March 14 - 31). **Plot a histogram of all differences.** Sanity check that the below histogram matches your observations of the histograms above, on your own.

[41]:
```
# The autograder expects differences to be a series object with index
# MOVEMENT_ID.
differences = averages_post - averages_pre
# plot the differences
plt.title('Difference Between Pre-Lockdown Average Speed and Post-Lockdown␣
 ↪Average Speed Distribution')
plt.xlabel('Average Speed Difference')
plt.ylabel('Count')
plt.hist(differences);
```



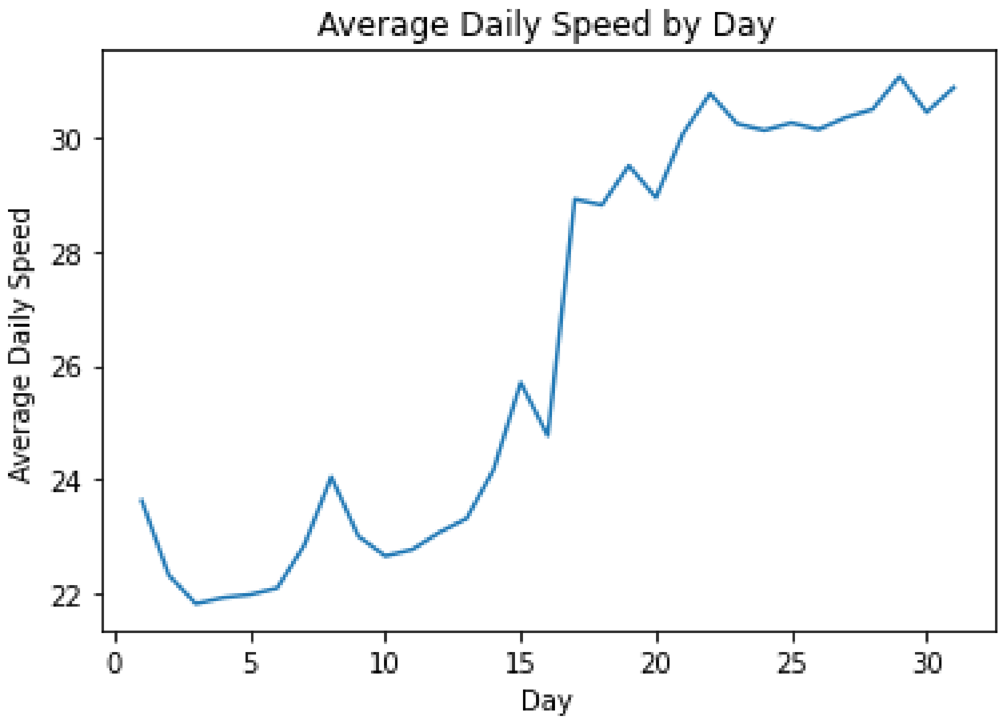
[42]:
```
grader.check("q2aiii")
```

[42]: q2aiii results: All test cases passed!

4.1.4 2.a.iv. Quantify the impact of lockdown on average speeds.

1. **Plot the average speed by day, across all segments**. Be careful not to plot the average of census tract averages instead. Recall the definition of segments from Q1.
2. Is the change in speed smooth and gradually increasing? Or increasing sharply? Why? Use your real-world knowledge of announcements and measures during that time, in your explanation. You can use this list of bay area COVID-related dataes: https://abc7news.com/timeline-of-coronavirus-us-covid-19-bay-area-sf/6047519/

[43]:
```
# Autograder expects this to be a series object containing the
# data for your line plot -- average speeds per day.
speeds_daily = speeds_to_tract.groupby('day')['speed_mph_mean'].agg(np.mean)
speeds_daily

plt.title('Average Daily Speed by Day')
plt.xlabel('Day')
plt.ylabel('Average Daily Speed')
plt.plot(speeds_daily);
```



Write your written answer in the cell below

Type your answer here, replacing this text.

Ignore the empty cell below, just run the autograder to test the code above is correct.

[ ]:

```

```

[44]:

```
grader.check("q2aiv3")
```

[44]: q2aiv3 results: All test cases passed!

4.1.5 2.a.v. Quantify the impact of pre-lockdown average speed on change in speed.

1. Compute the correlation between change in speed and the *pre*-lockdown average speeds. Do we expect a positive or negative correlation, given our analysis above?
2. Compute the correlation between change in speed and the post-lockdown average speeds.
3. **How does the correlation in Q1 compare with the correlation in Q2?** You should expect a significant change in correlation value. What insight does this provide about traffic?

Written answers in the first cell, coding answerts in the following cell.

Type your answer here, replacing this text.

[45]:

```
corr_pre_diff = differences.corr(averages_pre)
corr_post_diff = differences.corr(averages_post)
```

```
corr_pre_diff, corr_post_diff
```

[45]: (0.4633006380580185, 0.7926799984780658)

[46]:
```
grader.check("q2av2")
```

[46]: q2av2 results: All test cases passed!

4.2 2.b. What traffic areas were impacted by lockdown?

4.2.1 2.b.i. Visualize spatial heatmap of average traffic speed per census tract, pre-lockdown.

Visualize a spatial heatmap of the grouped average daily speeds per census tract, which you computed in previous parts. Use the geopandas chloropleth maps. **Write your observations, using your visualization, noting down at least 2 areas or patterns of interest**. These may be a local extrema, or a region that is strangely all similar.

Hint: Use `to_crs` and make sure the `epsg` is using the Pseudo-Mercator projection.

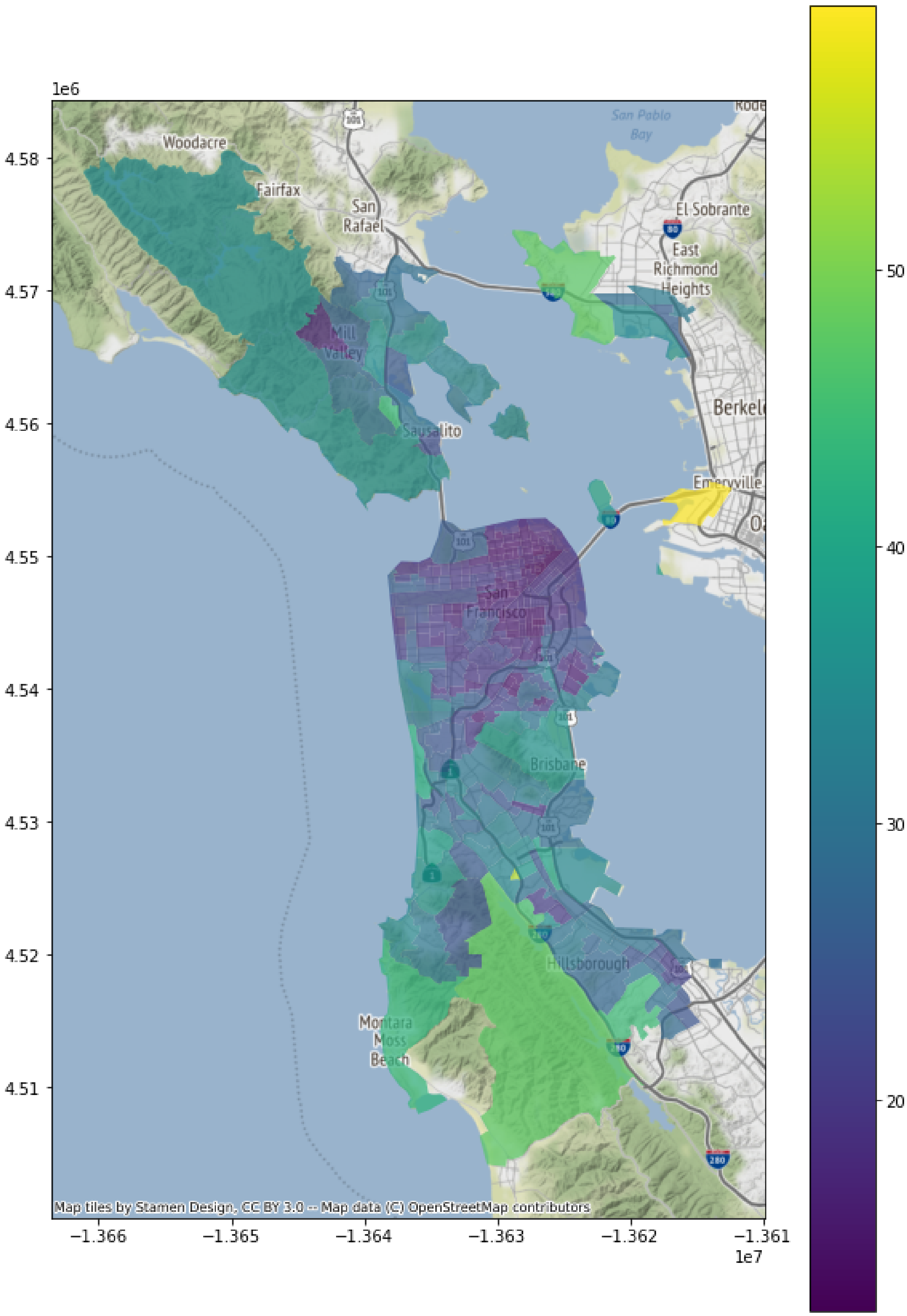
Hint: You can use `contextily` to superimpose your chloropleth map on a real geographic map.

Hint You can set a lower opacity for your chloropleth map, to see what's underneath, but be aware that if you plot with too low of an opacity, the map underneath will perturb your chloropleth and meddle with your conclusions.

Written answers in the first cell, coding answers in the second cell.

Type your answer here, replacing this text.

[47]:
```
graph = averages_pre_named.to_crs(epsg = 3857).plot(column = 'speed_mph_mean',␣
 ↪legend = True, figsize = (10, 15), alpha=0.7)
cx.add_basemap(graph)
```



1e6
4.58
4.57
4.56
4.55
4.54
4.53
4.52
4.51
−1.366
−1.365
−1.364
−1.363
−1.362
−1.361
1e7
50
40
30
20
Woodacre
Fairfax
San Rafael
El Sobrante
East Richmond Heights
San Pablo Bay
Mill Valley
Sausalito
Berkeley
Emeryville
San Francisco
Brisbane
Hillsborough
Montara
Moss Beach
Map tiles by Stamen Design, CC BY 3.0 -- Map data (C) OpenStreetMap contributors

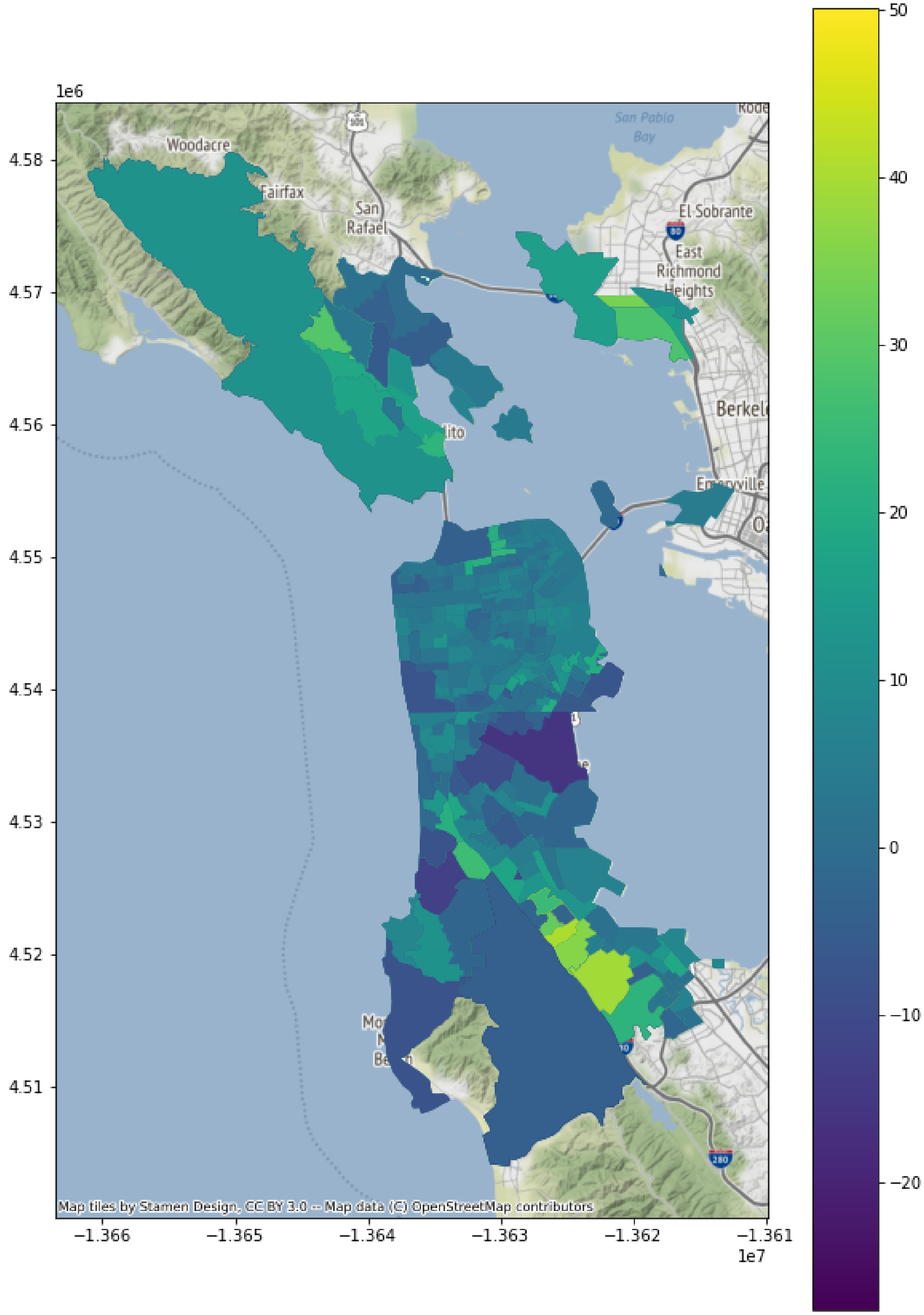
4.2.2 2.b.ii. Visualize change in average daily speeds pre vs. post lockdown.

Visualize a spatial heatmap of the census tract differences in average speeds, that we computed in a previous part. **Write your observations, using your visualization, noting down at least 2 areas or patterns of interest.** Some possible ideas for interesting notes: Which areas saw the most change in average speed? Which areas weren't affected? Why did some areas see *reduced* average speed?

First cell is for the written answers, second cell is for the coding answers.

Type your answer here, replacing this text.

[48]:
```
pre_post_avg = averages_pre_named.sjoin(averages_post_named, how = 'left')
pre_post_avg['differences'] = pre_post_avg['speed_mph_mean_right'] -␣
 ↪pre_post_avg['speed_mph_mean_left']
pre_post_avg
#type(averages_pre_named)
graph_2 = pre_post_avg.to_crs(epsg = 3857).plot(column = 'differences', legend␣
 ↪= True, figsize = (10, 15) )
cx.add_basemap(graph_2)
```



1e6
4.58
4.57
4.56
4.55
4.54
4.53
4.52
4.51
−1.366
−1.365
−1.364
−1.363
−1.362
−1.361
1e7
Woodacre
Fairfax
San Rafael
San Pablo Bay
El Sobrante
East Richmond Heights
Berkel
Map tiles by Stamen Design, CC BY 3.0 -- Map data (C) OpenStreetMap contributors
50
40
30
20
10
0
−10
−20

[49]:

```
pre_post_avg
```

[49]:

```
                               DISPLAY_NAME_left  speed_mph_mean_left  \
0          500 Hyde Street, Tenderloin, San Francisco            14.585102
0          500 Hyde Street, Tenderloin, San Francisco            14.585102
0          500 Hyde Street, Tenderloin, San Francisco            14.585102
0          500 Hyde Street, Tenderloin, San Francisco            14.585102
0          500 Hyde Street, Tenderloin, San Francisco            14.585102
..                                               …                  …
294  200 Westview Drive, Sunshine Gardens, South Sa…           24.822808
294  200 Westview Drive, Sunshine Gardens, South Sa…           24.822808
294  200 Westview Drive, Sunshine Gardens, South Sa…           24.822808
294  200 Westview Drive, Sunshine Gardens, South Sa…           24.822808
294  200 Westview Drive, Sunshine Gardens, South Sa…           24.822808

                                          geometry  index_right  \
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…           97
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…           48
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…            0
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…            1
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…          154
..                                               …            …
294  MULTIPOLYGON (((-122.45179 37.66912, -122.4506…          283
294  MULTIPOLYGON (((-122.45179 37.66912, -122.4506…           26
294  MULTIPOLYGON (((-122.45179 37.66912, -122.4506…          284
294  MULTIPOLYGON (((-122.45179 37.66912, -122.4506…           84
294  MULTIPOLYGON (((-122.45179 37.66912, -122.4506…           27

                              DISPLAY_NAME_right  speed_mph_mean_right  \
0    300 McAllister Street, Civic Center, San Franc…            19.342905
0        200 Myrtle Street, Tenderloin, San Francisco            16.392457
0          500 Hyde Street, Tenderloin, San Francisco            16.143154
0      900 Sutter Street, Lower Nob Hill, San Francisco            16.871488
0          200 Hyde Street, Tenderloin, San Francisco            15.665158
..                                               …                   …
294    300 Ponderosa Road, Avalon, South San Francisco            38.807594
294  200 Alta Mesa Drive, Serra Highlands, South Sa…            34.512982
294  200 Westview Drive, Sunshine Gardens, South Sa…            26.171347
294  300 Dolores Way, Sunshine Gardens, South San F…            32.027427
294                    1800 Hillside Boulevard, Colma            32.228029

     differences
0       4.757802
0       1.807355
0       1.558052
0       2.286386
0       1.080056
```

```
..           …
294    13.984786
294     9.690174
294     1.348539
294     7.204619
294     7.405221

[1950 rows x 7 columns]
```

5 Step 3 - Open-Ended EDA: Understanding lockdown impact on travel times

Explore daily travel times from Hayes Valley to other destinations both before and throughout lockdown. Use the following questions as suggestions for what to explore, temporally and spatially:

- How did lockdown affect travel times? Are there any meaningful factors that determined how travel time would be impacted? How was travel time affected over time?
- Travel to which destinations were affected by lockdown? Are there surprisingly disproportionate amounts of impact in certain areas?

5.1 3.a. Load Datasets

In this step, we will load two datasets:

- Daily travel times from Hayes Valley to all other census tracts around San Francisco.
- Daily travel times from 300 Hayes St to Golden Gate Park in San Francisco.

For this specific set of data, we can ask several more questions; which questions you pursue are up to you, including any that you come up that are not on this list:

- Which routes from Hayes Valley had similar impact on travel time? Did they share any factors in common? Traveling through the same place – e.g., a freway? Traveling in similar areas e.g., residential areas?
- Were clusters of routes impacted more severely than others over time? What determined the degree of impact?

[50]:
```
PATH_TIMES = 'data/travel-times-daily-san-francisco-2020-3.csv'
times_to_tract = pd.read_csv(PATH_TIMES)
times_to_tract
```

[50]:
```
       Origin Movement ID                              Origin Display Name  \
0                    1277  300 Hayes Street, Civic Center, San Francisco
1                    1277  300 Hayes Street, Civic Center, San Francisco
2                    1277  300 Hayes Street, Civic Center, San Francisco
3                    1277  300 Hayes Street, Civic Center, San Francisco
4                    1277  300 Hayes Street, Civic Center, San Francisco
…                     …                                           …
10333                1277  300 Hayes Street, Civic Center, San Francisco
10334                1277  300 Hayes Street, Civic Center, San Francisco
```

```
10335                1277  300 Hayes Street, Civic Center, San Francisco
10336                1277  300 Hayes Street, Civic Center, San Francisco
10337                1277  300 Hayes Street, Civic Center, San Francisco

       Destination Movement ID  \
0                            9
1                           20
2                           21
3                           44
4                           46
…                          …
10333                     2624
10334                     2643
10335                     2673
10336                     2694
10337                     2695

                               Destination Display Name  \
0             500 Hyde Street, Tenderloin, San Francisco
1       900 Sutter Street, Lower Nob Hill, San Francisco
2      3400 Pierce Street, Marina District, San Franc…
3             1700 Egbert Avenue, Bayview, San Francisco
4              500 Chester Street, West Oakland, Oakland
…                                                    …
10333      1300 16th Avenue, Inner Sunset, San Francisco
10334         1300 Egbert Avenue, Bayview, San Francisco
10335  100 Rutledge Street, Bernal Heights, San Franc…
10336  1900 Buchanan Street, Western Addition, San Fr…
10337  2200 Rivera Street, Sunset District, San Franc…

                                            Date Range  \
0        3/1/2020 - 3/1/2020, Every day, Daily Average
1        3/1/2020 - 3/1/2020, Every day, Daily Average
2        3/1/2020 - 3/1/2020, Every day, Daily Average
3        3/1/2020 - 3/1/2020, Every day, Daily Average
4        3/1/2020 - 3/1/2020, Every day, Daily Average
…                                                  …
10333  3/31/2020 - 3/31/2020, Every day, Daily Average
10334  3/31/2020 - 3/31/2020, Every day, Daily Average
10335  3/31/2020 - 3/31/2020, Every day, Daily Average
10336  3/31/2020 - 3/31/2020, Every day, Daily Average
10337  3/31/2020 - 3/31/2020, Every day, Daily Average

       Mean Travel Time (Seconds)  Range - Lower Bound Travel Time (Seconds)  \
0                             322                                         211
1                             291                                         179
2                             635                                         438
```

```
3                           786                                      566
4                           891                                      682
…                         …                                        …
10333                       502                                      411
10334                       571                                      475
10335                       367                                      265
10336                       222                                      167
10337                       917                                      778

       Range - Upper Bound Travel Time (Seconds)  day
0                                            489    1
1                                            470    1
2                                            920    1
3                                           1090    1
4                                           1162    1
…                                            …   …
10333                                        611   31
10334                                        685   31
10335                                        507   31
10336                                        294   31
10337                                       1080   31

[10338 rows x 9 columns]
```

[51]:
```python
#'Pre_time' dataframe contains rows for the days before the COVID lockdown was␣
↪imposed
pre_time = times_to_tract[times_to_tract['day']<14]
#'Post_time' dataframe contains rows for the days when the COVID lockdown was␣
↪imposed
post_time = times_to_tract[times_to_tract['day']>=15]
```

[52]:
```python
#pivoted = pd.pivot_table(times_to_tract)
#plt.boxplot(pre_time['Mean Travel Time (Seconds)'])
#plt.title("Travel Time (seconds) before COVID Lockdown");
#plt.ylabel("Travel Time (seconds)");
#WHAT SHOULD BE THE X AXIS?
```

[53]:
```python
#plt.boxplot(post_time['Mean Travel Time (Seconds)']);
#plt.title("Travel Time (seconds) after COVID Lockdown");
#plt.ylabel("Travel Time (seconds)");
#WHAT SHOULD BE THE X AXIS?
```

6 Step 4 - Guided Modeling: Predict traffic speed post-lockdown

In this step, you'll train a model to predict traffic speed. In particular, you'll learn how to provide implicit supervision and correction to your model, when you know there's been a distribution shift in its data, leading to a large gap between train and test sets. You'll follow the following outline:

- Build a model to predict daily traffic speed in San Francisco. Train and evaluate on *pre*-lockdown traffic speeds around the city.
- Evaluate your model on post-lockdown traffic speeds. Where is your model most mistaken, and why?
- Using this knowledge, how would you correct your model for a more accurate post-lockdown traffic predictor?

The technical term for a phenomenon like the lockdown, which caused major distributional shifts in the data, is *change point*. A large body of work studies "change point detection," but you'll be harder pressed to find a "handling change point" paper.

6.1 4.a. Predict daily traffic speed on pre-lockdown data

For your model, you will predict daily traffic speed per census tract, given the previous $k = 5$ daily traffic speeds for that census tract. In particular, say a matrix A is $n \times d$, where n is the number of census tracts and d is the number of days. We define the following inputs and labels:

$$X_{(i,t)} = [A_{(i,t-5)}, A_{(i,t-4)}, A_{(i,t-3)}, A_{(i,t-2)}, A_{(i,t-1)}]$$
$$y_{(i,t)} = [A_{(i,t)}]$$

This just means that each sample X_i includes speed averages from the previous 5 days for the ith census track.

6.1.1 4.a.i. Assemble dataset to predict daily traffic speed.

Below, we've included skeletons for the helper functions we defined, to complete the problem. We highly recommend following this skeleton code, else we cannot guarantee staff support for debugging your work.

Hint: What's wrong with collecting all samples, then randomly selecting some percentage to hold out? See the answer in the expandable below.

[Click to expand] How to do train-validation split correctly, on time series

For a *time series* in particular, this random split would be cheating, because data within each day is highly correlated. Instead, you should hold out entire days from the dataset. In this case, you should hold out the last 2 days for your validation set.

[54]:
```
def dataframe_to_time_series(df: pd.DataFrame):
    """Convert your dataframe into a 'time series'.

    :param df: the original dataframe, mapping speeds to census tracts.
        This dataframe should contain the `MOVEMENT_ID` (census tract id),
        `day`, and average speed for that day `speed_mph_mean`
    :return: a new dataframe that is formatted as n x d, where
        n is the number of samples (census tracts) and d is the number of
        dimensions (days). The values are the speeds.
    """
```

```
    new_df = pd.pivot_table( df,
                             values = 'speed_mph_mean',
                             index = ['MOVEMENT_ID'],
                             columns = ['day']
    )
    return new_df
time_series = dataframe_to_time_series(speeds_to_tract)
time_series_pre = time_series.iloc[:, list(range(13))]
```

[55]:
```
grader.check("q4ai1")
```

[55]: q4ai1 results: All test cases passed!

[56]:
```
# X = np.array( [ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
#                 [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
#                 [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
#                 [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
#                 [ 2, 3, 4, 5, 6, 7, 8, 9,10, 11] ] )

# X.shape
```

[57]:
```
# 295*19 == 5605
# time_series.iloc[:, 0:10]
# time_series.iloc[:, 1:11]
# time_series.iloc[:, 2:12]
# time_series.iloc[:, 3:13]
# ...
# time_series.iloc[:, 19:28]
# time_series
```

[58]:
```
# # np.arange(0, 20)
# # np.arange(0, 295)

# np.arange(0, 19)
```

[59]:
```
# result = []
# for i in np.arange(0, 19):
#     for j in np.arange(0, 295):
#         result.append(time_series.iloc[j, i:i+10])
```

[60]:
```
# # 5605
# # pd.DataFrame(np.asarray(result))

# np.asarray(result).shape
```

[61]:
```
# pd.DataFrame(np.asarray(result).iloc[292:298, :])
```

[62]:
```
# a = pd.DataFrame([[5, 6]])
# b = pd.DataFrame([[7, 9]])

# a.append(b)
```

[63]:
```
# foo = []

# foo.append(time_series.iloc[:, 0])
# foo.append(time_series.iloc[:, 1])
# np.asarray(foo).shape
# foo
```

[64]:
```
# T, n_val = 10, 2
# # answer = time_series_to_numpy(time_series, 10, 2)
# # time_series.iloc[]

# result = []

# rows = time_series.shape[1] - n_val - T

# for row in np.arange(rows):
#     temp = []
#     for i in np.arange(row, row+T):
#         temp.append(time_series.iloc[:, i])
#     result.append(temp)
#     # time_series.iloc[:, row:row+10]

# # result
# np.asarray(result).shape
```

[65]:
```
# T, n_val = 10, 2
# # answer = time_series_to_numpy(time_series, 10, 2)
# # time_series.iloc[]

# result = pd.DataFrame()

# rows = time_series.shape[1] - n_val - T

# for row in np.arange(rows):
#     temp = []
#     for i in np.arange(row, row+T):
#         temp.append(time_series.iloc[:, i])
#     df_t = pd.DataFrame(temp)
#     result.append(df_t)
#     # time_series.iloc[:, row:row+10]

# result
```

[66]:
```python
# for i in np.arange(0,
```

[67]:
```python
# def time_series_to_numpy(df: pd.DataFrame, T: int, n_val: int):
#     """Convert your 'time series' into train-validate splits, in numpy

#     You can assume your dataframe contains a `day` column where days
#     start from 1 and are consecutive.

#     :param df: the dataframe formatted as n x d, where
#         n is the number of samples (census tracts) and d is the number of
#         dimensions (days). The values are the speeds.
#     :param T: number of days to include in each training sample
#     :param n_val: number of days to hold out for the validation set.
#         Say we have 5 total days in our dataset, T=2, n_val=2. This means
#         during training, we have samples that pull averages from days 1 and
#         2 to predict day 3: x=(1, 2), y=(3,) For validation, we have samples
#         like x=(2, 3), y=(4,) and x=(3, 4), y=5,). This way, the model sees
#         data from days 4 and 5 only during validation.
#     :return: Set of 4 numpy arrays - X_train, y_train, X_val, y_val - where
#         X_* arrays are (n, T) and y_* arrays are (n,).
#     """
#     X_train = []
#     for i in np.arange(0, df.shape[1]-T-n_val):
#         for j in np.arange(0, df.shape[0]):
#             result.append(df.iloc[j, i:i+T])





#     return np.asarray(result)


#     # answer = time_series_to_numpy(time_series, 10, 2)
#     # time_series.iloc[]

# #     result = []

# #     rows = time_series.shape[1] - n_val - T

# #     for row in np.arange(rows):
# #         temp = []
# #         for i in np.arange(row, row+T):
# #             temp.append(time_series.iloc[:, i])
# #         result.append(temp)
# #         # time_series.iloc[:, row:row+10]
```

```
#     # X_train = df.loc[:,:T]
#     # #y_train = df.loc[T+1]
#     # return np.array(X_train, y_train, X_val, y_val)
#     # return result
#     # return np.array([np.array(result), [], [], []])

# def remove_nans(X: np.array, y: np.array):
#     """Remove all nans from the provided (X, y) pair.

#     Note: A nan in X means that sample must be removed from *both X and y.
#         Likewise, a nan in y means that sample must be removed from *both
#         X and y.

#     :param X: (n, T) array of model inputs
#     :param y: (n,) array of labels
#     :return: (X, y)
#     """
#     if not len(X):
#         return X, y
#     new = np.concatenate((X, y.T), axis=1)
#     result = new[~np.isnan(new).any(axis=1), :]

#     x = []
#     y = []

#     for arr in result:
#         x.append(arr[:-1])
#         y.append(arr[-1:])
#     return np.asarray(x), np.asarray(y)

# answer = time_series_to_numpy(time_series, 10, 2)
# # answer2 = remove_nans(answer[0], answer[1])
```

[68]:
```
# def time_series_to_numpy(df: pd.DataFrame, T: int, n_val: int):
#     """Convert your 'time series' into train-validate splits, in numpy

#     You can assume your dataframe contains a `day` column where days
#     start from 1 and are consecutive.

#     :param df: the dataframe formatted as n x d, where
#         n is the number of samples (census tracts) and d is the number of
#         dimensions (days). The values are the speeds.
#     :param T: number of days to include in each training sample
#     :param n_val: number of days to hold out for the validation set.
#         Say we have 5 total days in our dataset, T=2, n_val=2. This means
#         during training, we have samples that pull averages from days 1 and
#         2 to predict day 3: x=(1, 2), y=(3,) For validation, we have samples
```

```
#         like x=(2, 3), y=(4,) and x=(3, 4), y=5,). This way, the model sees
#         data from days 4 and 5 only during validation.
#     :return: Set of 4 numpy arrays - X_train, y_train, X_val, y_val - where
#         X_* arrays are (n, T) and y_* arrays are (n,).
#     """
#     X_train = []
#     y_train = []
#     X_val = []
#     y_val = []
#     for i in np.arange(0, df.shape[1]-T-n_val):
#         for j in np.arange(0, df.shape[0]):
#             X_train.append(df.iloc[j, i:i+T])

#     for i in np.arange(T, df.shape[1]-n_val):
#         for j in np.arange(0, df.shape[0]):
#             y_train.append(df.iloc[j, i])

#     for i in np.arange(df.shape[1]-T-n_val, df.shape[1]-T):
#         for j in np.arange(0, df.shape[0]):
#             X_val.append(df.iloc[j, i:i+T])

#     for i in np.arange(df.shape[1]-n_val, df.shape[1]):
#         for j in np.arange(0, df.shape[0]):
#             y_val.append(df.iloc[j, i])
#     return np.asarray(X_train), np.asarray(y_train),np.asarray(X_val), np.
↪asarray(y_val)


# def remove_nans(X: np.array, y: np.array):
#     """Remove all nans from the provided (X, y) pair.

#     Note: A nan in X means that sample must be removed from *both X and y.
#         Likewise, a nan in y means that sample must be removed from *both
#         X and y.

#     :param X: (n, T) array of model inputs
#     :param y: (n,) array of labels
#     :return: (X, y)
#     """
#     if not len(X):
#         return X, y
#     # new = np.concatenate((X, np.reshape(y, (y.shape[0], 1)).shape), axis=1)
#     new = np.concatenate((X, y[:,None]), axis=1)
#     result = new[~np.isnan(new).any(axis=1), :]

#     x = []
#     y = []
```

```
#     for arr in result:
#         x.append(arr[:-1])
#         y.append(arr[-1:])
#     return np.asarray(x), np.asarray(y)

# answer = time_series_to_numpy(time_series, 10, 2)
# answer[1].shape
# answer2 = remove_nans(answer[0], answer[1])
# answer2
```

[69]:
```
def time_series_to_numpy(df: pd.DataFrame, T: int, n_val: int):
    """Convert your 'time series' into train-validate splits, in numpy

    You can assume your dataframe contains a `day` column where days
    start from 1 and are consecutive.

    :param df: the dataframe formatted as n x d, where
        n is the number of samples (census tracts) and d is the number of
        dimensions (days). The values are the speeds.
    :param T: number of days to include in each training sample
    :param n_val: number of days to hold out for the validation set.
        Say we have 5 total days in our dataset, T=2, n_val=2. This means
        during training, we have samples that pull averages from days 1 and
        2 to predict day 3: x=(1, 2), y=(3,) For validation, we have samples
        like x=(2, 3), y=(4,) and x=(3, 4), y=5,). This way, the model sees
        data from days 4 and 5 only during validation.
    :return: Set of 4 numpy arrays - X_train, y_train, X_val, y_val - where
        X_* arrays are (n, T) and y_* arrays are (n,).
    """
    total_days = df.shape[0]
    X_train = []
    y_train = []
    X_val = []
    y_val = []
    for i in np.arange(0, df.shape[1]-T-n_val):
        for j in np.arange(0, df.shape[0]):
            X_train.append(df.iloc[j, i:i+T])

    for i in np.arange(T, df.shape[1]-n_val):
        for j in np.arange(0, df.shape[0]):
            y_train.append(df.iloc[j, i])

    for i in np.arange(df.shape[1]-T-n_val, df.shape[1]-T):
        for j in np.arange(0, df.shape[0]):
            X_val.append(df.iloc[j, i:i+T])
```

```
    for i in np.arange(df.shape[1]-n_val, df.shape[1]):
        for j in np.arange(0, df.shape[0]):
            y_val.append(df.iloc[j, i])

    return np.asarray(X_train), np.asarray(y_train),np.asarray(X_val), np.
↪asarray(y_val)





def remove_nans(X: np.array, y: np.array):
    """Remove all nans from the provided (X, y) pair.

    Note: A nan in X means that sample must be removed from *both X and y.
        Likewise, a nan in y means that sample must be removed from *both
        X and y.

    :param X: (n, T) array of model inputs
    :param y: (n,) array of labels
    :return: (X, y)
    """
    if not len(X):
        return X, y
    # new = np.concatenate((X, np.reshape(y, (y.shape[0], 1)).shape), axis=1)
    new = np.concatenate((X, y[:,None]), axis=1)
    result = new[~np.isnan(new).any(axis=1), :]

    x = []
    y = []

    for arr in result:
        x.append(arr[:-1])
        y.append(arr[-1:])
    return np.asarray(x), np.asarray(y).reshape((np.asarray(y).shape[0],))

answer = time_series_to_numpy(time_series, 10, 2)
answer2 = remove_nans(answer[0], answer[1])
```

[70]:
```
answer[0].shape, answer[1].shape, answer[2].shape, answer[3].shape
```

[70]: ((5605, 10), (5605,), (590, 10), (590,))

[71]:
```
answer2[1].shape
```

[71]: (4355,)

[72]:

```
# X = answer[0]
# answer[1].shape
# np.reshape(answer[1], (5605, 1)).shape
```

[73]:

```
# # answer = time_series_to_numpy(time_series, 10, 2)
# # answer2 = remove_nans(answer[0], answer[1])

# # numpy.empty((3,3,))

# # j = 0
# # for i in np.arange(len(a)):
# #     a[i].append(b[j])
# #     j += 1

# # foo = np.array([1, 2, 3])
# # bar = np.array([2, 4, 5])

# # any(np.isnan(new[0]))

# # for i in result.size:

# # x = result[:-1]
# # y = result[-1:]

# a = np.array([[1, 2, 3],
#               [4, 5, 6],
#               [7, float("NAN"), 9],
#               [3, 2, 1]])

# b = np.array([[float("NAN"), 7, 4, 9]]).T

# new = np.concatenate((a, b), axis=1)

# result = new[~np.isnan(new).any(axis=1), :]


# x = []
# y = []

# for arr in result:
#     x.append(arr[:-1])
#     y.append(arr[-1:])
```

[74]:

```
grader.check("q4ai2")
```

[74]: q4ai2 results: All test cases passed!

[75]:
```
def time_series_to_dataset(time_series: pd.DataFrame, T: int, n_val: int):
    """Convert 'time series' dataframe to a numpy dataset.

    Uses utilites above `time_series_to_numpy` and `remove_nans`

    For description of arguments, see `time_series_to_numpy` docstring.
    """
    time_series_nans = time_series_to_numpy(time_series, T, n_val)
    t_s_no_nans_train = remove_nans(time_series_nans[0], time_series_nans[1])
    t_s_no_nans_vals = remove_nans(time_series_nans[2], time_series_nans[3])
    return t_s_no_nans_train[0], t_s_no_nans_train[1], t_s_no_nans_vals[0],␣
 ↪t_s_no_nans_vals[1]

X_train, y_train, X_val, y_val = time_series_to_dataset(time_series_pre, 5, 2)
y_val.shape
# time_series_to_dataset(time_series_pre, 5, 2)
```

[75]: (562,)

[76]:
```
grader.check("q4ai3")
```

[76]: q4ai3 results: All test cases passed!

6.1.2 4.a.ii. Train and evaluate linear model on pre-lockdown data.

1. **Train a linear model that forecasts the next day's speed average** using your training dataset `X_train`, `y_train`. Specifically, predict $y_{(i,t)}$ from $X_{(i,t)}$, where

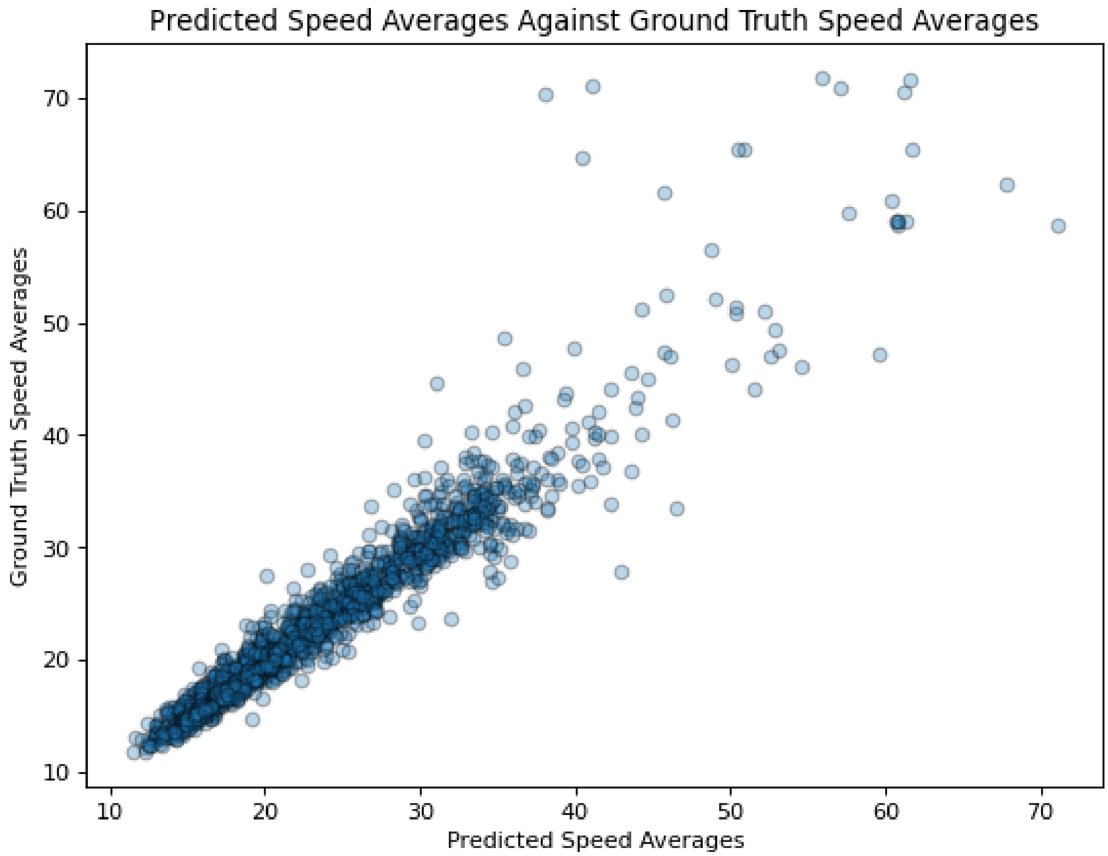
- $y_{(i,t)}$ is the daily speed average for day t and census tract i
- $X_{(i,t)}$ is a vector of daily speed averages for days $t-5, t-4, t-3, t-2, t-1$ for census tract i

2. **Evaluate your model** on your validation dataset `X_val`, `y_val`.
3. **Make a scatter plot**, plotting predicted averages against ground truth averages. Note the perfect model would line up all points along the line $y = x$.

Our model is quantitatively and qualitatively pretty accurate at this point, training and evaluating on pre-lockdown data.

[77]:
```
from matplotlib.pyplot import figure

figure(figsize=(8, 6), dpi=80)

model = LinearRegression(fit_intercept = True)
reg = model.fit(X_train, y_train) # set to trained linear model
score = reg.score(X_val, y_val) # report r^2 score
score
# create the scatter plot below
plt.scatter(reg.predict(X_train), y_train, edgecolors='black', alpha=0.3)
plt.title('Predicted Speed Averages Against Ground Truth Speed Averages')
```

```
plt.xlabel('Predicted Speed Averages')
plt.ylabel('Ground Truth Speed Averages');
```



[78]:
```
grader.check("q4aii2")
```

[78]: q4aii2 results: All test cases passed!

6.2 4.b. Understand failures on post-lockdown data

Your dataset is distributed spatially and temporally. As a result, the most intuitive spaces to visualize your model error or performance along is both spatially and temporally. In this step, we focus on understanding *where* your model fails.

6.2.1 4.b.i. Evaluate on post-lockdown data

1. Using your previously trained linear regression model `reg`, **evaluate on post-lockdown data**, meaning daily speed averages on March 14, 2020. Evaluate on all census tracts.
2. **Make a scatter plot**, plotting predicted averages against ground truth averages. Note the perfect model would line up all points along the line $y = x$.

[79]:
```
time_series_x_pre = time_series_to_dataset(time_series.iloc[:, 8:15], 5, 1)[0]␣
 ↪# get 'time series' dataframe for days 8, 10, 11, 12, 13
time_series_y_post = time_series_to_dataset(time_series.iloc[:, 8:15], 5, 1)[1]␣
 ↪# get 'time series' dataframe for 14th
score_pre_14th = reg.score(time_series_x_pre, time_series_y_post)
score_pre_14th
```

[79]: 0.9337122097376677

[80]:
```
time_series.iloc[:, 0:2]
# time_series_to_dataset(time_series.iloc[:, 0:2], 1, 1)[0]
```

[80]:
```
day                  1          2
MOVEMENT_ID
9            16.196918  14.395121
20           17.418045  15.460956
21           15.141171  13.176998
44           25.079544  23.492586
78           16.174464  16.755496
…                  …          …
2691               NaN        NaN
2694         17.809761  16.725889
2695         20.106061  20.228850
2700         34.586890  31.372308
2708         25.176235  24.725863

[295 rows x 2 columns]
```

[81]:
```
# # x = time_series_to_dataset(time_series.iloc[:, 8:15], 5, 1)[0]
# # y = time_series_to_dataset(time_series.iloc[:, 8:15], 5, 1)[1]
# # reg.score(x, y)

# x = time_series_to_dataset(time_series.iloc[:, 0:7], 5, 1)[0]
# y = time_series_to_dataset(time_series.iloc[:, 0:7], 5, 1)[1]
# reg.score(x, y)

# x = time_series_to_dataset(time_series.iloc[:, 24:31], 5, 1)[0]
# y = time_series_to_dataset(time_series.iloc[:, 24:31], 5, 1)[1]
# reg.score(x, y)
```

[82]:
```
result = []

for i in np.arange(0, 25):
    x = time_series_to_dataset(time_series.iloc[:, i:i+7], 5, 1)[0]
    y = time_series_to_dataset(time_series.iloc[:, i:i+7], 5, 1)[1]
    result.append(reg.score(x, y))
```

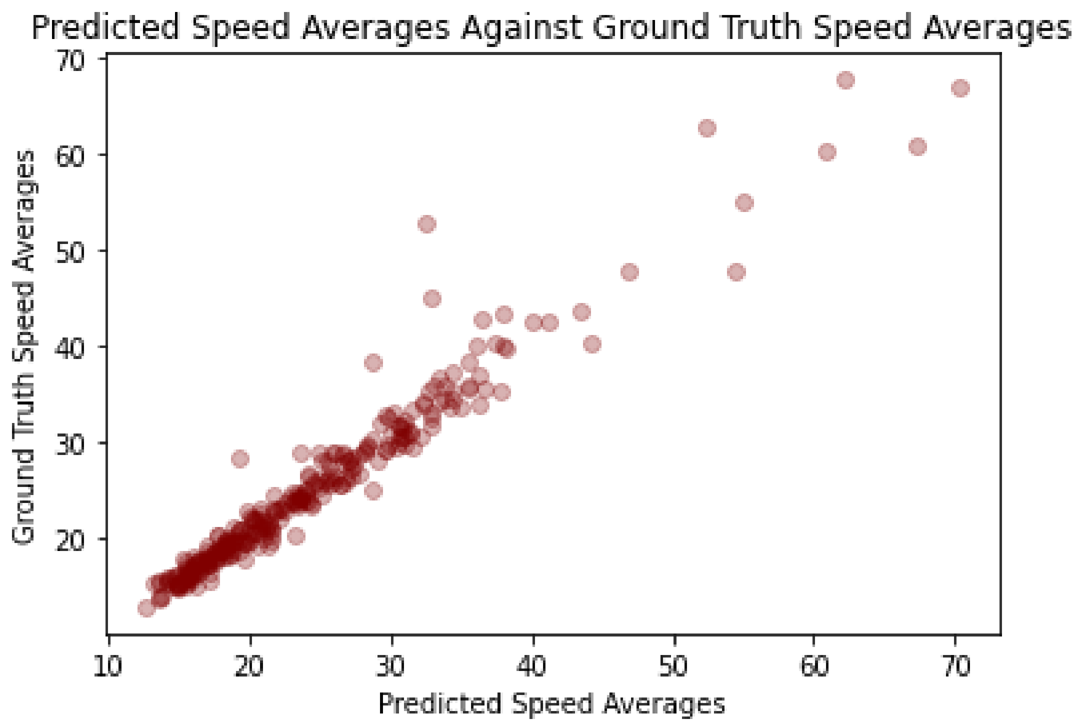
```
new_model = LinearRegression(fit_intercept = True)
new_reg = model.fit(X_train, y_train)
score = reg.score(X_val, y_val)
```

[83]:
```
grader.check("q4bi1")
```

[83]: q4bi1 results: All test cases passed!

Make scatter plot below.

[84]:
```
plt.title('Predicted Speed Averages Against Ground Truth Speed Averages')
plt.xlabel('Predicted Speed Averages')
plt.ylabel('Ground Truth Speed Averages')
plt.scatter(reg.predict(time_series_x_pre), time_series_y_post, alpha = 0.3,␣
 ↪color = 'maroon');
```



6.2.2 4.b.ii. Report model performance temporally

1. **Make a line plot** showing performance of the original model throughout all of March 2020.
2. **Report the lowest point on the line plot**, reflecting the lowest model performance.
3. **Why is model performance the worst on the 17th?** Why does it begin to worsen on march 15th? And continue to worsen? Use what you know about covid measures on those dates. You may find this webpage useful: https://abc7news.com/timeline-of-coronavirus-us-covid-19-bay-area-sf/6047519/

4. **Is the dip in performance on the 9th foreshadowed** by any of our EDA?
5. **How does the model miraculously recover on its own?**
6. **Make a scatter plot**, plotting predicted averages against ground truth averages *for model predictions on March 17th*. Note the perfect model would line up all points along the line $y = x$. When compared against previous plots of this nature, this plot looks substantially worse, with points straying far from $y = x$.

Note: Answer questions 2-5 in the Markdown cell below. Q1 and Q6 are answered in the two code cells below.

The lowest point on the Line graph was day 17. On day 17, or on the 16th at midnight, a shelter in place order was announced in the Bay Area which closed most of the businesses/schools. As a result, most of the Bay was brought to a standstill, which shows why the day 17th line graph takes a serious dip. For the dip in day 9, there was constant news throughout the week before of the Princess Cruise ship among first deaths in the state of California. Once the first death took place in Santa Clara County (in the Bay), then there was probably much more immediate action on that day regarding staying indoors as people realized how close this virus is hitting home. In addition, the Princess Cruise with many positive covid persons which had been on the news for the past week, was now disembarking causing more cautious behavior to Bay residents and staying off the roads to not go anywhere - translating to a dip in the graph at around day 9. The model miraculously recovers on its own through natural ways. This is because after the strict lockdown, things started opening up again and people started going a bit more outside the house to which the line graph shot up again and stayed at a fluctuating top position. While most things did not open up, there were certain things such as grocery stores or appliance stores that people needed to get things from to store in their house while they were in lockdown.

Generate line plot.

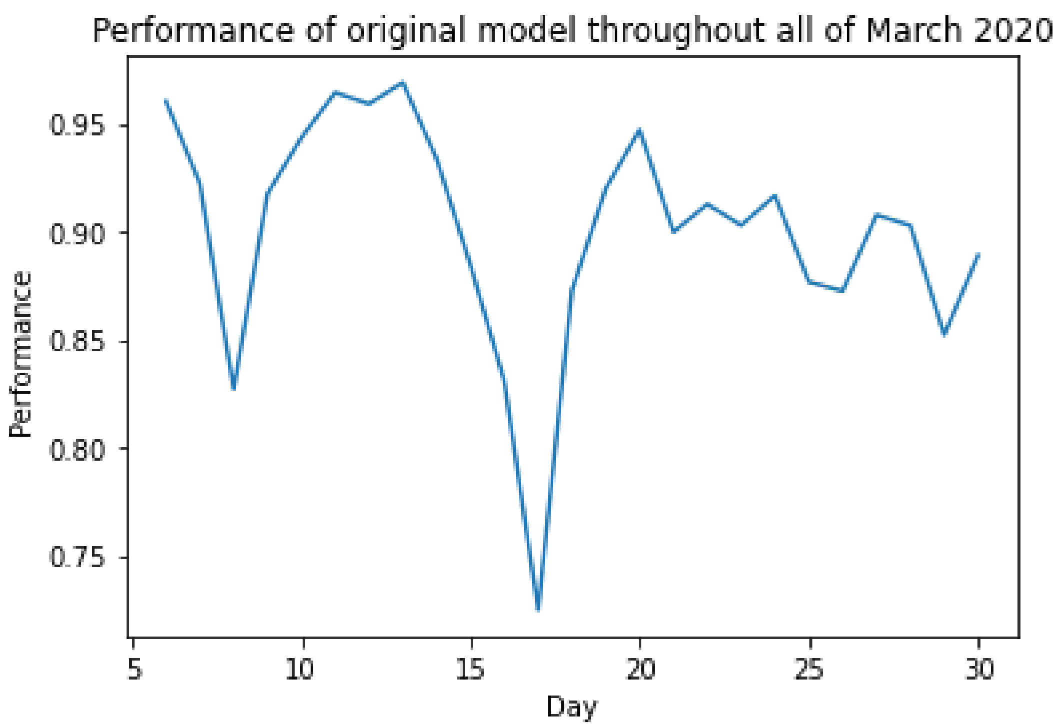
[85]:
```
# 1, 2, 3, 4, 5 -> 6
# 2, 3, 4, 5, 6 -> 7
```

[86]:
```
# time_series_x_pre = time_series_to_dataset(time_series.iloc[:, 16:23], 5,␣
↪1)[0] # get 'time series' dataframe for days 9, 10, 11, 12, 13
# time_series_y_post = time_series_to_dataset(time_series.iloc[:, 16:23], 5,␣
↪1)[1] # get 'time series' dataframe for 14th
# score_pre_14th = reg.score(time_series_x_pre, time_series_y_post)
# score_pre_14th
```

[87]:
```
# plt.title('Performance of original model throughout all of March 2020')
# plt.xlabel('Day')
# plt.ylabel('Speed')
# plt.plot(time_series.columns, [reg.score(time_series_to_dataset(time_series,␣
↪i, 1)[0], time_series_to_dataset(time_series, i, 1)[1]) for i in␣
↪range(1,time_series.shape[0])])
```

[88]:
```
y = []
for i in range(25):
    time_series_x_pre = time_series_to_dataset(time_series.iloc[:, i:i+7], 5,␣
↪1)[0] # for day 6 get 1,2,3,4,5
```

```
    #time_series.iloc[:, 8:15], 5, 1)[1] get 'time series' dataframe for days␣
↪9, 10, 11, 12, 13
    time_series_y_post = time_series_to_dataset(time_series.iloc[:, i:i+7], 5,␣
↪1)[1] # get 'time series' dataframe for 14th
    score_ith = reg.score(time_series_x_pre, time_series_y_post)
    y.append(score_ith)

plt.title('Performance of original model throughout all of March 2020')
plt.xlabel('Day')
plt.ylabel('Performance')
plt.plot(time_series.columns[5:30], y)

march_performance = pd.Series(y, time_series.columns[5:30])
```



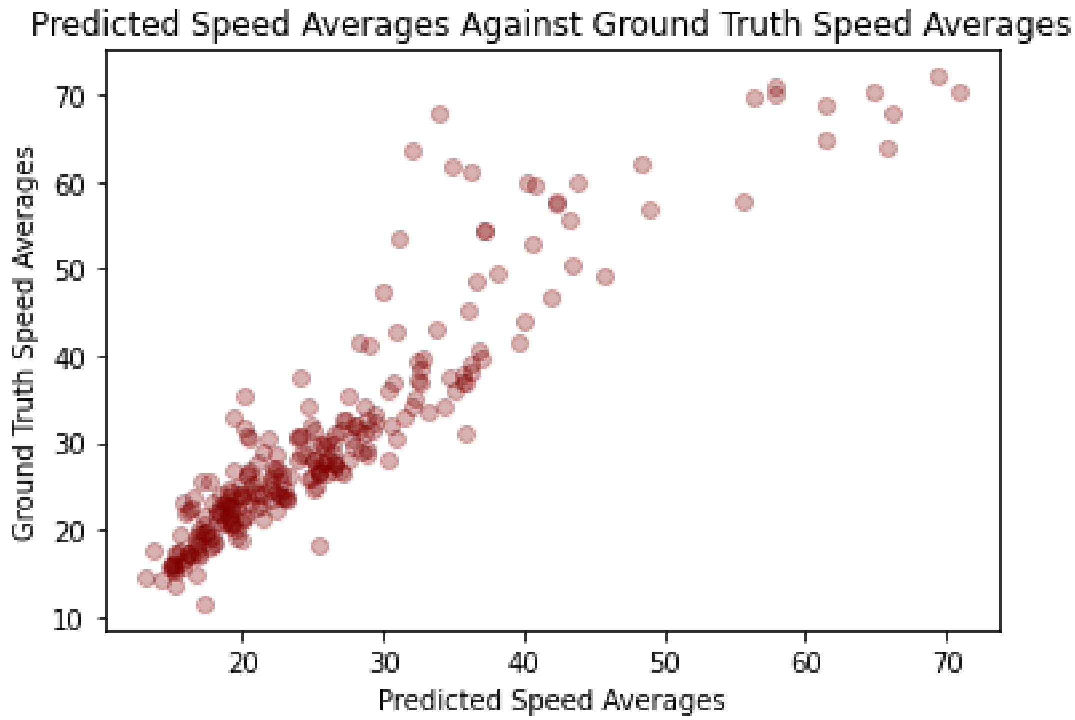

```
[89]: print("Day", march_performance.idxmin(), "had the lowest performance of",␣
↪march_performance.min());
```

```
Day 17 had the lowest performance of 0.7248438392437919
```

Generate a scatter plot.

```
[90]: time_series_x_17th = time_series_to_dataset(time_series.iloc[:, 11:18], 5,␣
↪1)[0] # get 'time series' dataframe for days 9, 10, 11, 12, 13
```

```
time_series_y_17th = time_series_to_dataset(time_series.iloc[:, 11:18], 5,␣
    ↪1)[1] # get 'time series' dataframe for 14th
score_pre_17th = reg.score(time_series_x_17th, time_series_y_17th)
plt.title('Predicted Speed Averages Against Ground Truth Speed Averages')
plt.xlabel('Predicted Speed Averages')
plt.ylabel('Ground Truth Speed Averages')
plt.scatter(reg.predict(time_series_x_17th), time_series_y_17th, alpha = 0.3,␣
    ↪color = 'maroon');
```



6.3 4.c. "Fix" model on post-lockdown data

Per this survey https://pure.tue.nl/ws/files/3488790/740215.pdf, there are 4 categories of fixes for change points: - Forgetting mechanisms - Explicit change detection - Ensemble techniques - Context-aware approaches

In this part, we'll combine insights in model errors with previous EDA insights to produce a fix.

6.3.1 4.c.i. Learn delta off of a moving bias

According to our previous work in EDA, the average speed shoots upwards sharply. As a result, our trick to learn delta the around the average and to naively assume that the average of day t is the average for day $t + 1$. We will do this in 4 steps:

1. **Create a dataset for your delta model**.
2. **Train your delta model** on pre-lockdown data.

3. **Evaluate your model on pre-lockdown data**, to ensure that the model has learned to a satisfactory degree, in the nominal case. Remember the naive model achieved 0.97 r^2 on pre-lockdown data.
4. **Evaluate your model on the 17th**, to compare against the naive model also evaluated on that day. Notice that your r^2 score has improved by 10%+. Why is your delta model so effective for the 17th?
5. **Evaluate your model on the 14th**, to compare against the naive model also evaluated on that day. Notice that your r^2 score is now complete garbage. Why is your delta so ineffective for the 14th?

Hint: As you build your datasets, always check to make sure you're using the right days! It's easy to have a one-off error that throws off your results.

Write your written questions in the next cell, then write the code in the following cells.

Type your answer here, replacing this text.

[91]:
```
# time_series.iloc[:, 1:14]
```

[92]:
```
# np.arange(1, 14)
```

[93]:
```
# np.arange(1, 14)
```

[94]:
```
# time_series.iloc[:, 1:14].mean().index
```

[95]:
```
# speeds_daily[0:13]
```

[96]:
```
time_series_delta = time_series_pre - speeds_daily[0:13] # subtract daily␣
↪average from pre-lockdown 'time series' dataframe `time_series_pre`
time_series_delta
```

[96]:
```
day                  1         2         3         4         5         6   \
MOVEMENT_ID
9            -7.436210 -7.929735 -7.957726 -7.706332 -7.673488 -8.288859
20           -6.215082 -6.863900 -6.363125 -6.511982 -6.693731 -6.873540
21           -8.491956 -9.147858 -8.044932 -7.879632 -7.907553 -8.464493
44            1.446417  1.167730  0.901289  0.342397 -0.110985  0.981564
78           -7.458664 -5.569360 -5.787301 -5.813345 -6.320619 -6.303126
…                 …         …         …         …         …         …
2691               NaN       NaN       NaN       NaN -3.385687 -4.313850
2694         -5.823366 -5.598968 -5.295639 -5.485229 -5.482634 -5.448377
2695         -3.527067 -2.096006 -1.977287 -1.785869  0.371917 -2.850371
2700         10.953762  9.047451  9.039961  9.069252  9.650690  8.113142
2708          1.543108  2.401007  2.303618  2.355356  2.378369  2.989213

day                  7          8         9          10         11         12  \
MOVEMENT_ID
9            -7.342673  -7.838718 -8.163539  -7.950124 -9.037052 -8.687048
20           -5.762984  -5.960594 -6.830754  -7.366163 -8.139844 -8.795882
```

```
21           -8.893040  -9.407162 -7.392644  -7.811186 -8.986853 -8.505135
44            0.024509   1.604391  0.492527   0.078211  0.979607  1.910786
78           -7.548693  -4.765377 -6.755502  -6.530367 -7.003837 -6.987069
…                    …          …         …          …         …         …
2691          0.621691        NaN -3.970859        NaN       NaN -0.971484
2694         -5.423980  -5.992169 -5.698587  -5.662927 -5.885166 -6.070161
2695         -3.627592  -4.578175 -2.511783  -2.260573  1.093583 -2.863087
2700         10.730575  11.394160  9.171008  10.290198  8.797572  9.094474
2708          2.700848   2.246655  2.503544   2.224406  1.125645  2.184237

day                 13
MOVEMENT_ID
9           -9.364599
20          -8.883074
21          -8.143181
44          -0.144157
78          -7.112274
…                   …
2691              NaN
2694        -5.761289
2695        -2.977301
2700         8.013932
2708         1.129593

[295 rows x 13 columns]
```

[97]:
```python
grader.check("q4ci2")
```

[97]: q4ci2 results: All test cases passed!

[98]:
```python
X_delta_train, y_delta_train, X_delta_val, y_delta_val = 
 ↪time_series_to_dataset(time_series_delta, 5, 2)
model_delta = LinearRegression(fit_intercept = True)
reg_delta = model_delta.fit(X_delta_train, y_delta_train)
res_4ci3 = reg_delta.score(X_delta_val, y_delta_val) # learning delta as easy 
 ↪as learning original dataset!
res_4ci3
```

[98]: 0.9645254590172871

[99]:
```python
grader.check("q4ci3")
```

[99]: q4ci3 results: All test cases passed!

[100]:
```python
# speeds_daily[11:18]
```

[101]:
```
# test = speeds_daily[7:14]
# test.index = [9, 10, 11, 12, 13, 14, 15]
# foobar = time_series.iloc[:, 8:15] - test

# x_5, y_5, x_5_val, y_5_val = time_series_to_dataset(foobar, 5, 1)

# model_5 = LinearRegression(fit_intercept = True)
# reg_5 = model_5.fit(x_5, y_5)

# res_4ci5 = reg_5.score(x_5_val, y_5_val)
# res_4ci5
```

[102]:
```
# ### BACKUP for q4ci4
# test = speeds_daily[12:19]
# test.index = [12, 13, 14, 15, 16, 17, 18]
# foobar = time_series.iloc[:, 11:18] - test

# x, y, x_val, y_val = time_series_to_dataset(foobar, 5, 2)

# model = LinearRegression(fit_intercept = True)
# reg = model.fit(x, y)

# res_4ci4 = reg.score(x_val, y_val)
# res_4ci4
```

[103]:
```
# Evaluate your model on the 17th, to compare against the naive
# model also evaluated on that day. Notice that your r^2 score
# has improved by 10%+. Why is your delta model so effective for the 17th?
```

[104]:
```
# time_series.iloc[:, 11:18]
```

[105]:
```
# test = speeds_daily[12:19]
# test.index = [12, 13, 14, 15, 16, 17, 18]

# x_4, y_4, x_4_val, y_4_val = time_series_to_dataset(time_series.iloc[:, 11:
 ↪18] - test, 5, 1)

# model_4 = LinearRegression(fit_intercept = True)
# reg_4 = model_4.fit(x_4, y_4)

# res_4ci4 = reg_4.score(x_4_val, y_4_val)
# res_4ci4
```

[106]:
```
# np.isclose(res_4ci4, 0.8616633417528182, rtol=1e-4, atol=1e-4)
```

[107]:
```
# time_series.iloc[:, 11:18]
```

[108]:
```
# test = speeds_daily[12:19]
# test.index = [12, 13, 14, 15, 16, 17, 18]

# x_4, y_4, x_4_val, y_4_val = time_series_to_dataset(time_series.iloc[:, 11:
↪18] - test, 5, 1)

# model = LinearRegression(fit_intercept = True)
# reg = model.fit(x_4, y_4)

# res_4ci4 = reg.score(x_4_val, y_4_val)
# res_4ci4
```

[109]:
```
test = speeds_daily[9:16]
test.index = [9, 10, 11, 12, 13, 14, 15]

x_4, y_4, x_4_val, y_4_val = time_series_to_dataset(time_series.iloc[:, 8:15] -␣
↪test, 5, 1)

model_4 = LinearRegression(fit_intercept = True)
reg_4 = model_4.fit(x_4, y_4)

res_4ci4 = reg_4.score(x_4_val, y_4_val)
res_4ci4
```

[109]:
```
0.8426975156332025
```

[110]:
```
grader.check("q4ci4")
```

[110]:
```
q4ci4 results:
    q4ci4 - 1 result:
        Trying:
            np.isclose(res_4ci4, 0.8616633417528182, rtol=1e-4, atol=1e-4)
        Expecting:
            True
        **********************************************************************
        Line 1, in q4ci4 0
        Failed example:
            np.isclose(res_4ci4, 0.8616633417528182, rtol=1e-4, atol=1e-4)
        Expected:
            True
        Got:
            False
```

[111]:
```
# test = speeds_daily[9:16]
# test.index = [9, 10, 11, 12, 13, 14, 15]
# time_series.iloc[:, 8:15] - test
```

[112]:
```
# test = speeds_daily[9:16]
# test.index = [9, 10, 11, 12, 13, 14, 15]
# foobar = time_series.iloc[:, 8:15] - test

# x_5, y_5, x_5_val, y_5_val = time_series_to_dataset(foobar, 5, 1)

# model_5 = LinearRegression(fit_intercept = True)
# reg_5 = model_5.fit(x_5, y_5)

# res_4ci5 = reg_5.score(x_5_val, y_5_val)
# res_4ci5
```

[113]:
```
grader.check("q4ci5")
```

[113]:
```
q4ci5 results:
    q4ci5 - 1 result:
        Trying:
            np.isclose(res_4ci5, 0.11611253470677951, rtol=1e-4, atol=1e-4)
        Expecting:
            True
        **********************************************************************
        Line 1, in q4ci5 0
        Failed example:
            np.isclose(res_4ci5, 0.11611253470677951, rtol=1e-4, atol=1e-4)
        Exception raised:
            Traceback (most recent call last):
              File "/opt/conda/lib/python3.9/doctest.py", line 1336, in __run
                exec(compile(example.source, filename, "single",
              File "<doctest q4ci5 0[0]>", line 1, in <module>
                np.isclose(res_4ci5, 0.11611253470677951, rtol=1e-4, atol=1e-4)
            NameError: name 'res_4ci5' is not defined
```

6.3.2 4.c.ii. Does it "solve itself"? Does the pre-lockdown model predict, after the change point?

Had we ignored the problem, would we have been okay? The temporal plot above showing performance over time suggests a partial recovery. **Evaluate the original, naive model on all post-lockdown data** to see. If your final r^2 score does not match the autograder's: - Double check you have selected daily average speeds for the right days, by printing your dataframe. - Double check you're using the right model (a brand new trained model) - Check you're using `T=5`, `n_val=2`

[114]:
```
time_series_x_pre = time_series_to_dataset(time_series.iloc[:,13:31], 5, 0)[0]␣
↪# for day 6 get 1,2,3,4,5
# time_series.iloc[:, 8:15], 5, 1)[1] get 'time series' dataframe for days 9,␣
↪10, 11, 12, 13
```

```
time_series_y_post = time_series_to_dataset(time_series.iloc[:,13:31], 5, 0)[1]␣
↪# get 'time series' dataframe for 14th
score_ith = reg.score(time_series_x_pre, time_series_y_post)
# y_post.append(score_ith)

score_og_post = reg.score(time_series_x_pre, time_series_y_post)
score_og_post
```

[114]: 0.9014738674628208

[115]:
```
grader.check("q4cii")
```

[115]: q4cii results: All test cases passed!

6.3.3 4.c.iii. Naively retrain model with post-lockdown data

Can we use the same tactics–that we used to train the original model on pre-lockdown data–to train on the post-lockdown data? **Retrain a linear model and evaluate on post-lockdown data only**. You should construct a new dataset using `time_series_to_dataset` using only time series from March 14 to March 31. If your final r^2 score does not match the autograder's: - Double check you have selected daily average speeds for the right days, by printing your dataframe. - Double check you're using the right model (a brand new trained model) - Check you're using `T=5, n_val=2`

[116]:
```
X_train_post, y_train_post, X_val_post, y_val_post =␣
↪time_series_to_dataset(time_series.iloc[:,13:31],5,2)

model_post = LinearRegression(fit_intercept = True)
reg = model_post.fit(X_train_post, y_train_post) # set to trained linear model
score_post = reg.score(X_val_post, y_val_post) # report r^2 score
score_post
```

[116]: 0.8993687576351703

[117]:
```
grader.check("q4ciii")
```

[117]: q4ciii results: All test cases passed!

6.3.4 4.c.iv. What if you just ignore the change point?

Turns out, this is no good. Even acknowledging the change point and training *either* before *or* after is better. Being ignorant and training on *both* is the worst option, producing a lower r^2.

[118]:
```
X_train_all, y_train_all, X_val_all, y_val_all =␣
↪time_series_to_dataset(time_series,5,5)
model_last = LinearRegression(fit_intercept = True)
reg_all = model_last.fit(X_train_all, y_train_all) # set to trained linear model
```

```
res_4civ = reg_all.score(X_val_all, y_val_all)
res_4civ
```

[118]: 0.8843433608623491

[119]:
```
grader.check("q4civ")
```

[119]: q4civ results: All test cases passed!

7 Step 5 - Open-Ended Modeling: Predicting travel time post-lockdown

This is the real deal and ultimately what Uber cares about. Traffic speeds is a proxy task, but the bottom line and moneymaking machine relies on this travel time estimation. Focus on designing experiments instead of focusing on experimental, quantitative results. Your experiments are successful if they inform a decision, even despite a lower-performing model.

7.1 Question 5a

Train a baseline model of your choice using any supervised learning approach we have studied; you are not limited to a linear model.

Example

Given the data for this question, you could build a model to predict travel time from Cheesecake Factory to UC Berkeley.

7.2 5a. Loading and Cleaning Data

7.2.1 5a.i. Loading the DataFrame for our model

The `pre_post` DataFrame contains labeled data that we will use to train our model. It contains the following columns:

Post-lockdown Columns: 1. `MOVEMENT_ID_left`: Unique ID of each movement data 1. `DISPLAY_NAME_left`: Destination address 1. `geometry`: Mapping from GPS coordinates to boundaries of census tracts 1. `Origin Movement ID_left`: Movement ID of the starting point, 300 Hayes St 1. `Origin Display Name_left`: Adress of the starting point, 300 Hayes St 1. `Destination Movement ID_left`: Movement ID of the destination 1. `Destination Display Name_left`: Adress of the destination 1. `Date Range_left`: Date of movment data 1. `Mean Travel Time (Seconds)_left`: Mean travel time of each movement data 1. `Range - Lower Bound Travel Time (Seconds)_left`: 1. `Range - Upper Bound Travel Time (Seconds)_left`: 1. `day_left`: Day of movement data on March

Pre-lockdown Columns: 1. `MOVEMENT_ID_right`: Unique ID of each movement data 1. `DISPLAY_NAME_right`: Destination address 1. `Origin Movement ID_right`: Movement ID of the starting point, 300 Hayes St 1. `Origin Display Name_right`: Adress of the starting point, 300 Hayes St 1. `Destination Movement ID_right`: Movement ID of the destination 1. `Destination Display Name_right`: Adress of the destination 1. `Date Range_right`: Date of movment data 1. `Mean Travel Time (Seconds)_right`: Mean travel time of each movement data 1. `Range - Lower Bound Travel Time (Seconds)_right`: 1. `Range - Upper Bound Travel Time (Seconds)_right`: 1. `day_right`: Day of movement data on March 1. `proportion after/before`: Proportion of post/pre-lockdown average traffic speeds

[120]:
```
pd.set_option('display.max_columns', None) # setting to show all columns on a␣
↪dataframe
```

[121]:
```
# 'pre_named': Dataframe merged with tract_to_gps on Movement ID(from Guided␣
↪EDA) to get the geometry for Destination Movement ID
pre_named = tract_to_gps.merge(pre_time, how='right', left_on='MOVEMENT_ID',␣
↪right_on='Destination Movement ID')
# 'post_named': Dataframe merged with tract_to_gps on Movement ID(from Guided␣
↪EDA) to get the geometry for Destination Movement ID
post_named = tract_to_gps.merge(post_time, how='right', left_on='MOVEMENT_ID',␣
↪right_on='Destination Movement ID')
```

[122]:
```
# 'pre_post': spatially joining pre and post data ( _left: pre data / _right:␣
↪post data )
pre_post = gpd.sjoin(pre_named, post_named, predicate='within')

# creating 'differences' column to calculate the difference between pre COVID␣
↪lockdown and post COVID lockdown travel time
pre_post['proportion before/after'] = pre_post['Mean Travel Time␣
↪(Seconds)_left'] / pre_post['Mean Travel Time (Seconds)_right']
pre_post.drop('index_right', axis = 1, inplace = True)
pre_post.head(3)
```

[122]:
```
     MOVEMENT_ID_left                     DISPLAY_NAME_left  \
0                   9  500 Hyde Street, Tenderloin, San Francisco
512                 9  500 Hyde Street, Tenderloin, San Francisco
958                 9  500 Hyde Street, Tenderloin, San Francisco

                                              geometry  \
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
512  MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
958  MULTIPOLYGON (((-122.41827 37.78704, -122.4150…

     Origin Movement ID_left                    Origin Display Name_left  \
0                       1277  300 Hayes Street, Civic Center, San Francisco
512                     1277  300 Hayes Street, Civic Center, San Francisco
958                     1277  300 Hayes Street, Civic Center, San Francisco

     Destination Movement ID_left            Destination Display Name_left  \
0                               9  500 Hyde Street, Tenderloin, San Francisco
512                             9  500 Hyde Street, Tenderloin, San Francisco
958                             9  500 Hyde Street, Tenderloin, San Francisco
```

```
                                 Date Range_left  \
0    3/1/2020 - 3/1/2020, Every day, Daily Average
512  3/2/2020 - 3/2/2020, Every day, Daily Average
958  3/3/2020 - 3/3/2020, Every day, Daily Average

     Mean Travel Time (Seconds)_left  \
0                                322
512                              355
958                              369

     Range - Lower Bound Travel Time (Seconds)_left  \
0                                               211
512                                             220
958                                             233

     Range - Upper Bound Travel Time (Seconds)_left  day_left  \
0                                               489         1
512                                             570         2
958                                             583         3

     MOVEMENT_ID_right                           DISPLAY_NAME_right  \
0                    9  500 Hyde Street, Tenderloin, San Francisco
512                  9  500 Hyde Street, Tenderloin, San Francisco
958                  9  500 Hyde Street, Tenderloin, San Francisco

     Origin Movement ID_right                      Origin Display Name_right  \
0                        1277  300 Hayes Street, Civic Center, San Francisco
512                      1277  300 Hayes Street, Civic Center, San Francisco
958                      1277  300 Hayes Street, Civic Center, San Francisco

     Destination Movement ID_right  \
0                                9
512                              9
958                              9

                 Destination Display Name_right  \
0    500 Hyde Street, Tenderloin, San Francisco
512  500 Hyde Street, Tenderloin, San Francisco
958  500 Hyde Street, Tenderloin, San Francisco

                                    Date Range_right  \
0    3/15/2020 - 3/15/2020, Every day, Daily Average
512  3/15/2020 - 3/15/2020, Every day, Daily Average
958  3/15/2020 - 3/15/2020, Every day, Daily Average

     Mean Travel Time (Seconds)_right  \
```

```
0                                  262
512                                262
958                                262

     Range - Lower Bound Travel Time (Seconds)_right  \
0                                                178
512                                              178
958                                              178

     Range - Upper Bound Travel Time (Seconds)_right  day_right  \
0                                                385         15
512                                              385         15
958                                              385         15

     proportion before/after
0                   1.229008
512                 1.354962
958                 1.408397
```

[123]:
```
# pre_post['differences'] = pre_post['Mean Travel Time (Seconds)_left'] -␣
 ↪pre_post['Mean Travel Time (Seconds)_right']
# # we are trying to standardize our difference to accomadate the differences␣
 ↪in the distance which directly affects the time travelled between two␣
 ↪different locations
# pre_post_mean = pre_post['differences'].mean()
# pre_post_std = pre_post['differences'].std()
# pre_post_stand = (pre_post['differences'] - pre_post_mean)/pre_post_std
# # creating 'standard_value' column which calculates the standardized travel␣
 ↪time value
# pre_post['standard_value'] = pre_post_stand
```

[ ]:
```

```

[ ]:
```

```

[124]:
```
#############################################
#############################################
#############################################
#############################################
#############################################
#############################################
#############################################
#############################################


### Drop the year columns!!!!
```

```
##############################################
##############################################
##############################################
##############################################
##############################################
##############################################
##############################################
##############################################
```

[ ]:

```

```

[ ]:

```

```

7.2.2 5.a.ii. Loading and Cleaning Income Data

The `census_tract_income_median` DataFrame contains labeled data that we will use to get the income data for our model. It contains the following columns:

1. `ID Year`:
2. `Year`:
3. `ID Race`:
4. `Race`:
5. `Household Income by Race`: Household income by race
6. `Household Income by Race Moe`: Household income by race margin of error (Moe)
7. `Geography`: Census tract location
8. `ID Geography`: ID of each census tract

[125]:

```
census_tract_income_median = pd.read_csv('data/Income by Location.csv') #=␣
↪census_tract_income_median[census_tract_income_median['Year' == '2019']]
cleaned_value = [ census_tract_income_median['Geography'].str.split(',')[i][0]␣
↪for i in range(census_tract_income_median['Geography'].str.split(',').
↪shape[0])]
census_tract_income_median['NAMELSAD10'] = cleaned_value
```

[126]:

```
census_tract_income_median.head(3)
```

[126]:

```
   ID Year  Year  ID Race   Race  Household Income by Race  \
0     2019  2019        0  Total                     62414
1     2019  2019        0  Total                    151453
2     2019  2019        0  Total                    150972

   Household Income by Race Moe                                  Geography  \
0                       26676.0  Census Tract 101, San Francisco County, CA
1                       19040.0  Census Tract 102, San Francisco County, CA
2                       20529.0  Census Tract 103, San Francisco County, CA

         ID Geography       NAMELSAD10
```

```
0  14000US06075010100  Census Tract 101
1  14000US06075010200  Census Tract 102
2  14000US06075010300  Census Tract 103
```

7.2.3 5.a.iii. Loading and Cleaning location data

The `sf_geo` DataFrame contains labeled data that we will use to get the location data for our model. It contains the following columns:

1. `statefp10`:
2. `mtfcc10`:
3. `name10`:
4. `intptlat10`:
5. `awater10`:
6. `namelsad10`:
7. `funcstat10`:
8. `aland10`:
9. `geoid10`:
10. `tractce10`:
11. `intptlon10`:
12. `countyfp10`:
13. `geometry`:

[127]:
```
P = os.path.expanduser('data/Census 2010_ Tracts for San Francisco.geojson')
sf_geo = gpd.read_file(P)
```

[128]:
```
sf_geo.head(3)
```

[128]:
```
  statefp10 mtfcc10 name10   intptlat10 awater10        namelsad10 funcstat10  \
0        06   G5020    165  +37.7741958        0  Census Tract 165          S
1        06   G5020    164  +37.7750995        0  Census Tract 164          S
2        06   G5020    163  +37.7760456        0  Census Tract 163          S

  aland10      geoid10 tractce10    intptlon10 countyfp10  \
0  370459  06075016500    016500  -122.4477884        075
1  309097  06075016400    016400  -122.4369729        075
2  245867  06075016300    016300  -122.4295509        075

                                            geometry
0  MULTIPOLYGON (((-122.44647 37.77580, -122.4447…
1  MULTIPOLYGON (((-122.44034 37.77658, -122.4398…
2  MULTIPOLYGON (((-122.42915 37.77801, -122.4289…
```

7.2.4 5.a.iv. Merging income and location data

Slicing `census_tract_income_median` dataframe to merge with the `sf_geo` dataframe

[129]:
```
census_ready_to_merge = census_tract_income_median.loc[:, ['Household Income by␣
↪Race', 'ID Geography', 'NAMELSAD10','Household Income by Race Moe']]
```

[130]:
```
census_ready_to_merge.head(3)
```

[130]:
```
   Household Income by Race        ID Geography        NAMELSAD10  \
0                     62414  14000US06075010100  Census Tract 101
1                    151453  14000US06075010200  Census Tract 102
2                    150972  14000US06075010300  Census Tract 103

   Household Income by Race Moe
0                       26676.0
1                       19040.0
2                       20529.0
```

Merge `sf_geo` with `census_ready_to_merge` to include household income

[131]:
```
# merge_data = census_tract_points.merge(census_ready_to_merge, how = 'outer',␣
↪on='NAMELSAD10')
merge_data = sf_geo.merge(census_ready_to_merge, how = 'inner',␣
↪left_on='namelsad10', right_on='NAMELSAD10')
merge_data = merge_data.loc[:, ['geometry', 'Household Income by Race',␣
↪'NAMELSAD10', 'Household Income by Race Moe']]
```

[132]:
```
merge_data.head(3)
```

[132]:
```
                                            geometry  \
0  MULTIPOLYGON (((-122.44647 37.77580, -122.4447…
1  MULTIPOLYGON (((-122.44647 37.77580, -122.4447…
2  MULTIPOLYGON (((-122.44647 37.77580, -122.4447…

   Household Income by Race        NAMELSAD10  Household Income by Race Moe
0                    168536  Census Tract 165                       31297.0
1                    132750  Census Tract 165                       34187.0
2                    117156  Census Tract 165                       12445.0
```

Rename `the_geom` column to `geometry`

[133]:
```
merge_data.rename(columns={"the_geom": "geometry"}, inplace=True)
```

Select columns that will be used for modeling in the `pre_post` dataframe

[134]:
```
pre_post = pre_post.loc[:, ['geometry', 'Destination Movement ID_left',␣
↪'Destination Display Name_left', 'Mean Travel Time (Seconds)_left', 'Range -␣
↪Lower Bound Travel Time (Seconds)_left', 'Range - Upper Bound Travel Time␣
↪(Seconds)_left',
```

```
                                 'Mean Travel Time (Seconds)_right', 'Range - Lower␣
↪Bound Travel Time (Seconds)_right', 'Range - Upper Bound Travel Time␣
↪(Seconds)_right', 'proportion before/after']]
```

[135]:
```
pre_post.head(3)
```

[135]:
```
                                              geometry  \
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
512  MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
958  MULTIPOLYGON (((-122.41827 37.78704, -122.4150…

     Destination Movement ID_left               Destination Display Name_left  \
0                               9  500 Hyde Street, Tenderloin, San Francisco
512                             9  500 Hyde Street, Tenderloin, San Francisco
958                             9  500 Hyde Street, Tenderloin, San Francisco

     Mean Travel Time (Seconds)_left  \
0                                322
512                              355
958                              369

     Range - Lower Bound Travel Time (Seconds)_left  \
0                                               211
512                                             220
958                                             233

     Range - Upper Bound Travel Time (Seconds)_left  \
0                                               489
512                                             570
958                                             583

     Mean Travel Time (Seconds)_right  \
0                                 262
512                               262
958                               262

     Range - Lower Bound Travel Time (Seconds)_right  \
0                                                178
512                                              178
958                                              178

     Range - Upper Bound Travel Time (Seconds)_right  proportion before/after
0                                                385                 1.229008
512                                              385                 1.354962
958                                              385                 1.408397
```

[136]:
```
# Columns: incomes, proportion after / before time travel, lat, long,␣
↪proportion range,
```

Spatially join `pre_post` dataframe with `merge_data` dataframe to create a new dataframe with household income, speed, and location data

[137]:
```
match_census_pre_post = pre_post.sjoin(merge_data, how="inner")
```

[138]:
```
match_census_pre_post.head(3)
```

[138]:
```
                                              geometry  \
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
512  MULTIPOLYGON (((-122.41827 37.78704, -122.4150…
958  MULTIPOLYGON (((-122.41827 37.78704, -122.4150…

     Destination Movement ID_left              Destination Display Name_left  \
0                               9  500 Hyde Street, Tenderloin, San Francisco
512                             9  500 Hyde Street, Tenderloin, San Francisco
958                             9  500 Hyde Street, Tenderloin, San Francisco

     Mean Travel Time (Seconds)_left  \
0                                322
512                              355
958                              369

     Range - Lower Bound Travel Time (Seconds)_left  \
0                                               211
512                                             220
958                                             233

     Range - Upper Bound Travel Time (Seconds)_left  \
0                                               489
512                                             570
958                                             583

     Mean Travel Time (Seconds)_right  \
0                                 262
512                               262
958                               262

     Range - Lower Bound Travel Time (Seconds)_right  \
0                                                178
512                                              178
958                                              178

     Range - Upper Bound Travel Time (Seconds)_right  proportion before/after  \
0                                                385                 1.229008
```

```
512                                              385               1.354962
958                                              385               1.408397

     index_right  Household Income by Race          NAMELSAD10  \
0            227                     15272  Census Tract 123.01
512          227                     15272  Census Tract 123.01
958          227                     15272  Census Tract 123.01

     Household Income by Race Moe
0                          2872.0
512                        2872.0
958                        2872.0
```

Create `new_tract` dataframe to merge with `match_census_pre_post dataframe` to include the Latitude, Longitude, and MOVEMENT_ID

[139]:
```
new_tract = speeds_to_tract.loc[:, ['Latitude', 'Longitude', 'MOVEMENT_ID']]
new_tract = new_tract.groupby('MOVEMENT_ID').agg(lambda x: list(x)[0])
```

[140]:
```
new_tract.head(3)
```

[140]:
```
              Latitude   Longitude
MOVEMENT_ID
9            37.785467 -122.415677
20           37.787866 -122.416782
21           37.800611 -122.437908
```

Merge `match_census_pre_post` dataframe to `new_tract` dataframe to include 'Latitude' and 'Longitude'

[141]:
```
# # ( _left columns: pre / _right columns: post )
match_census_pre_post = match_census_pre_post.merge(new_tract,␣
 ↪left_on='Destination Movement ID_left', right_on='MOVEMENT_ID', how='inner')
```

Include 'proportion lower' and 'proportion upper' columns into `match_census_pre_post` dataframe

[142]:
```
match_census_pre_post['proportion lower'] = match_census_pre_post['Range -␣
 ↪Lower Bound Travel Time (Seconds)_right'] / match_census_pre_post['Range -␣
 ↪Lower Bound Travel Time (Seconds)_left']
match_census_pre_post['proportion upper'] = match_census_pre_post['Range -␣
 ↪Upper Bound Travel Time (Seconds)_right'] / match_census_pre_post['Range -␣
 ↪Upper Bound Travel Time (Seconds)_left']
```

[143]:
```
match_census_pre_post = match_census_pre_post.loc[:, ['Destination Movement␣
 ↪ID_left', 'Destination Display Name_left', 'proportion lower', 'proportion␣
 ↪upper', 'Latitude', 'Longitude', 'Household Income by Race', 'proportion␣
 ↪before/after']]
match_census_pre_post
```

[143]:
```
         Destination Movement ID_left  \
0                                   9
1                                   9
2                                   9
3                                   9
4                                   9
…                                 …
1617116                          1778
1617117                          1778
1617118                          1778
1617119                          1778
1617120                          1778

                             Destination Display Name_left  proportion lower  \
0                 500 Hyde Street, Tenderloin, San Francisco          0.843602
1                 500 Hyde Street, Tenderloin, San Francisco          0.809091
2                 500 Hyde Street, Tenderloin, San Francisco          0.763948
3                 500 Hyde Street, Tenderloin, San Francisco          0.773913
4                 500 Hyde Street, Tenderloin, San Francisco          0.723577
…                                                        …                …
1617116  7700 Geary Boulevard, Richmond District, San F…          0.955882
1617117  7700 Geary Boulevard, Richmond District, San F…          0.929678
1617118  7700 Geary Boulevard, Richmond District, San F…          0.915493
1617119  7700 Geary Boulevard, Richmond District, San F…          0.912281
1617120  7700 Geary Boulevard, Richmond District, San F…          1.002571

         proportion upper   Latitude   Longitude  Household Income by Race  \
0                0.787321  37.785467 -122.415677                     15272
1                0.675439  37.785467 -122.415677                     15272
2                0.660377  37.785467 -122.415677                     15272
3                0.652542  37.785467 -122.415677                     15272
4                0.670732  37.785467 -122.415677                     15272
…                     …          …          …                    …
1617116          0.645137  37.777343 -122.502824                     99917
1617117          0.786425  37.777343 -122.502824                     99917
1617118          0.629710  37.777343 -122.502824                     99917
1617119          0.666922  37.777343 -122.502824                     99917
1617120          0.692982  37.777343 -122.502824                     99917

         proportion before/after
0                       1.229008
1                       1.354962
2                       1.408397
3                       1.408397
4                       1.435115
…                            …
1617116                 1.273058
```

```
1617117                 1.168689
1617118                 1.316748
1617119                 1.281553
1617120                 1.199029

[1617121 rows x 8 columns]
```

7.2.5 5.a.v. Create a model to predict average speed proportion using income

We've created a model to predict proportion of average speed (post-lockdown average traffic speed / pre-lockdown average speed) and calculated its accuracy.

[144]:
```
X_1 = match_census_pre_post.loc[:, ['Household Income by Race']].values  #␣
↪features or predictor
y_1 = match_census_pre_post.loc[:, ['proportion before/after']].values   #␣
↪target
# X, y = remove_nans(X_regre,y_regre)

X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_1, y_1,␣
↪test_size=0.3, random_state=0)

model_1 = LinearRegression().fit(X_train_1, y_train_1)
# y_pred_1 = model_1.predict(X_test_1)

model_1.score(X_test_1, y_test_1), model_1.score(X_train_1, y_train_1)
```

[144]:
```
(0.005109961565790688, 0.005326041743364929)
```

[145]:
```
model_lasso = Lasso(alpha=0.01)
model_lasso.fit(X_train_1, y_train_1)
pred_train_lasso= model_lasso.predict(X_train_1)
print(np.sqrt(mean_squared_error(y_train_1,pred_train_lasso)))
print(r2_score(y_train_1, pred_train_lasso))

pred_test_lasso= model_lasso.predict(X_test_1)
print(np.sqrt(mean_squared_error(y_test_1,pred_test_lasso)))
print(r2_score(y_test_1, pred_test_lasso))
```

```
0.20183142281550306
0.005326041741898102
0.2022607761326694
0.005109964832327618
```

7.2.6 5.a.vi. Visualization of model prediction

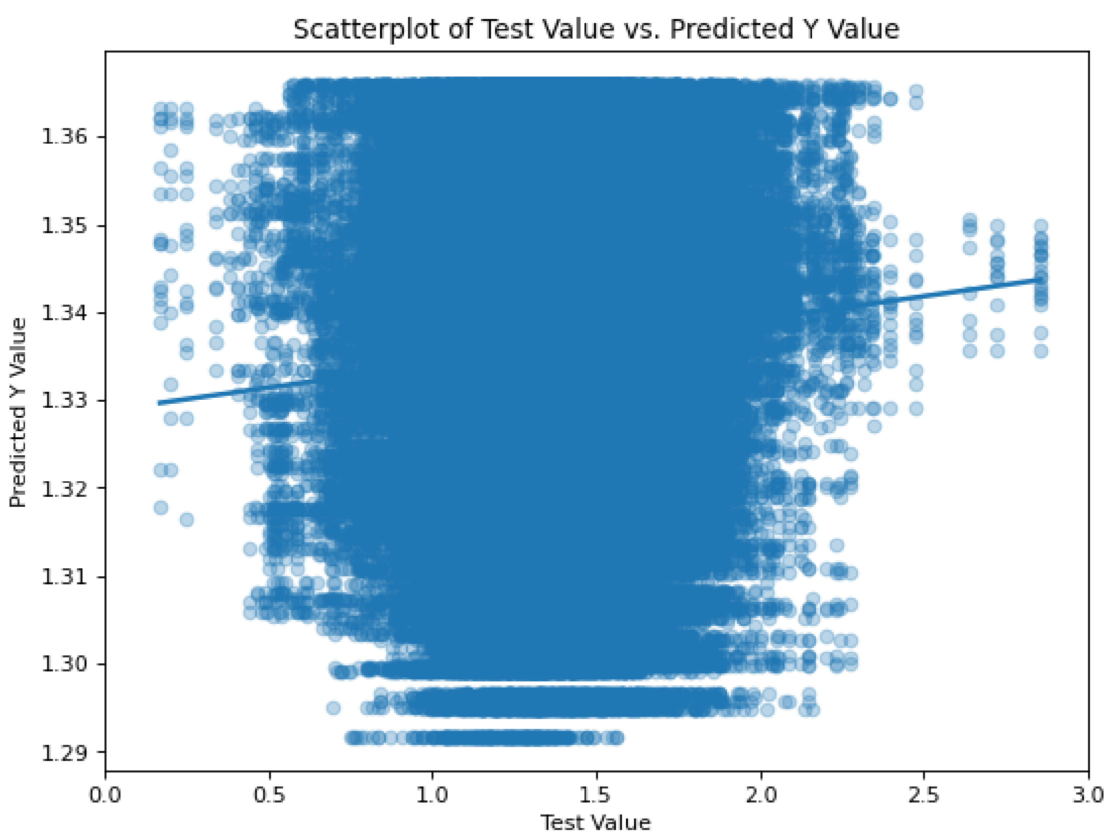
[146]:
```
y_pred_1 = model_1.predict(X_test_1)
```

[147]:
```
mean_squared_error(y_test_1, y_pred_1), r2_score(y_test_1, y_pred_1)
```

[147]: (0.040909421696108304, 0.005109961565790688)

[148]:
```
from matplotlib.pyplot import figure

figure(figsize=(8, 6), dpi=80)

aa = sns.regplot(y_test_1, y_pred_1, scatter_kws={'alpha':0.3})
aa.set_xlim(0,3)
plt.xlabel("Test Value")
plt.ylabel("Predicted Y Value")
plt.title("Scatterplot of Test Value vs. Predicted Y Value");
```

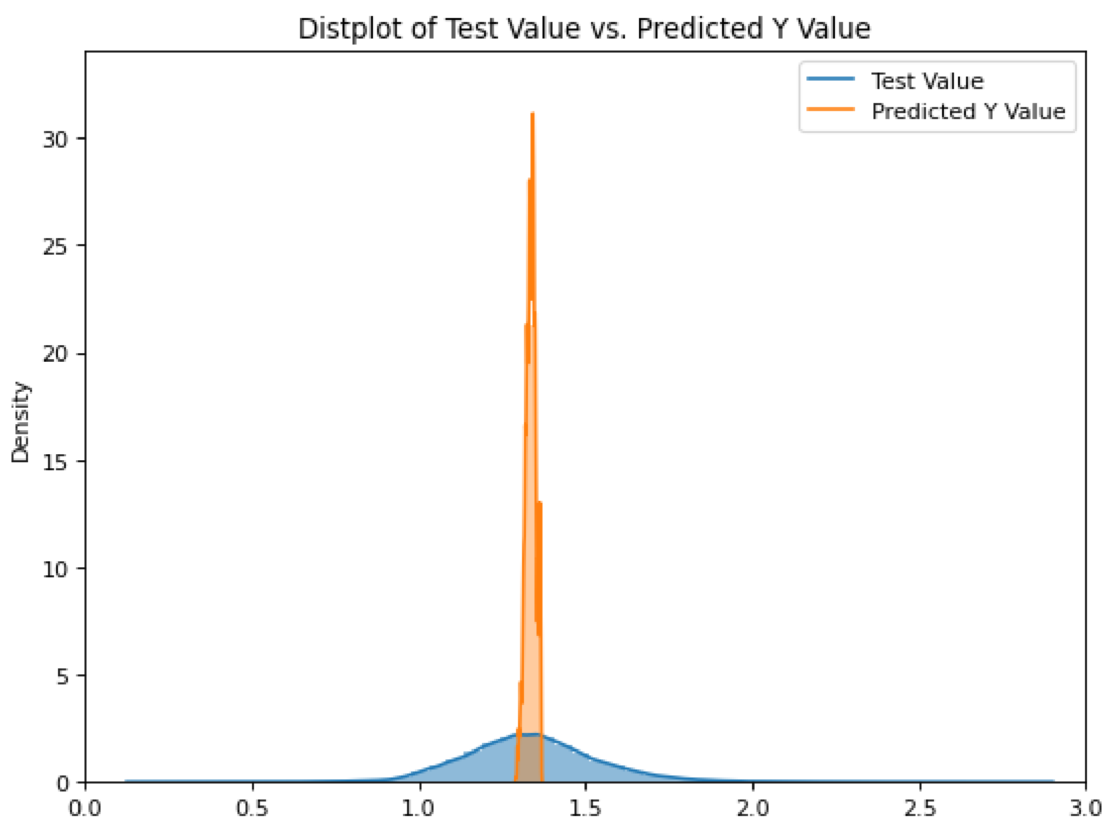
```
/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



[149]:

```
figure(figsize=(8, 6), dpi=80)

ax = sns.distplot(y_test_1, kde=True, hist_kws={"alpha": 0.5},)
sns.distplot(y_pred_1, kde=True, hist_kws={"alpha": 0.4}, ax=ax)
plt.legend(labels=['Test Value', 'Predicted Y Value'])
ax.set_xlim(0,3)
plt.title("Distplot of Test Value vs. Predicted Y Value");
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



[150]:
```python
import statsmodels.api as sm

#est = sm.OLS(y.astype(float), X.astype(float)).fit()

y_1c = match_census_pre_post.loc[:, ['proportion before/after']]
x_1c = sm.add_constant(match_census_pre_post.loc[:, ['Household Income by␣
↪Race']])
model_1c = sm.OLS(y_1c.astype(float), x_1c.astype(float))
results_1c = model_1c.fit(cov_type = 'HC1')
results_1c.summary()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

[150]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               OLS Regression Results
================================================================================
===
Dep. Variable:     proportion before/after   R-squared:
0.005
Model:                                 OLS   Adj. R-squared:
0.005
Method:                      Least Squares   F-statistic:
7637.
Date:                     Mon, 13 Dec 2021   Prob (F-statistic):
0.00
Time:                             23:13:59   Log-Likelihood:
2.9229e+05
No. Observations:                  1617121   AIC:
-5.846e+05
Df Residuals:                      1617119   BIC:
-5.845e+05
Df Model:                                1
Covariance Type:                       HC1
================================================================================
============
                               coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------------
const                        1.3700      0.000   3114.707      0.000       1.369
1.371
Household Income by Race   -3.6e-07   4.12e-09    -87.391      0.000   -3.68e-07
-3.52e-07
```

```
==============================================================================
Omnibus:                    62769.470   Durbin-Watson:                   0.467
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           141703.291
Skew:                           0.243   Prob(JB):                         0.00
Kurtosis:                       4.367   Cond. No.                     2.64e+05
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
[2] The condition number is large, 2.64e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[151]:
```python
# from sklearn.preprocessing import StandardScaler

# sc_y = StandardScaler()
# sc_x = StandardScaler()

# y_std = sc_y.fit_transform(y_train_1[:, np.newaxis].reshape(y_train_1.
 ↪shape[0], y_train_1.shape[1])).flatten()
# X_train_std = sc_x.fit_transform(X_train_1)
# X_test_std = sc_x.transform(X_test_1)
# y_train_std = sc_y.fit_transform(y_train_1[:, np.newaxis].reshape(y_train_1.
 ↪shape[0], y_train_1.shape[1])).flatten()
```

[152]:
```python
# from sklearn.linear_model import Lasso
# from sklearn.metrics import r2_score

# alpha = np.linspace(0.01,0.4,10)

# r2_train =[]
# r2_test =[]
# norm = []

# alpha = np.linspace(0.01,0.4,10)

# for i in range(10):
#     lasso = Lasso(alpha = alpha[i])
#     lasso.fit(X_train_std,y_train_std)
#     y_train_std = lasso.predict(X_train_std)
#     y_test_std = lasso.predict(X_test_std)
#     r2_train = np.append(r2_train, r2_score(y_train,sc_y.
 ↪inverse_transform(y_train_std)))
#     r2_test = np.append(r2_test, r2_score(y_test,sc_y.
 ↪inverse_transform(y_test_std)))
#     norm = np.append(norm,np.linalg.norm(lasso.coef_))
```

[153]:
```
# plt.figure(figsize=(8,6))
# plt.scatter(alpha,r2_train,label='r2_train')
# plt.plot(alpha,r2_train)
# plt.scatter(alpha,r2_test,label='r2_test')
# plt.plot(alpha,r2_test)
# plt.scatter(alpha,norm,label = 'norm')
# plt.plot(alpha,norm)
# plt.ylim(-0.1,1)
# plt.xlim(0,.43)
# plt.xlabel('alpha', size = 14)
# plt.ylabel('R2_score',size = 14)
# plt.legend()
# plt.show()
```

[154]:
```
# from sklearn.linear_model import Lasso, LassoCV
# from sklearn.preprocessing import StandardScaler
# from sklearn.model_selection import train_test_split
# from sklearn.pipeline import Pipeline
# from sklearn.compose import ColumnTransformer
# from sklearn.preprocessing import FunctionTransformer
# from sklearn.impute import SimpleImputer
# from sklearn.linear_model import LinearRegression
# from sklearn.model_selection import cross_val_score
```

[155]:
```
# quantitative_features = ['Household Income by Race']
```

[156]:
```
# lasso_model = Pipeline([
#     ("SelectColumns", ColumnTransformer([
#         ("keep", StandardScaler(), quantitative_features),
#     ])),
#     ("Imputation", SimpleImputer()),
#     ("LinearModel", LassoCV(cv=3))
# ])
```

[157]:
```
# lasso_model.fit(X_train_1, y_train_1)
# # models["LassoCV"] = lasso_model
# # compare_models(models)
```

7.3 Question 5b

Improve on your baseline model. Specify the model you designed and its input features. Justify why you chose these features and their relevance to your model's predictions.

Example

Here are potential questions to consider for this part: How does the other variant of your travel times dataset, aggregated across time but reported for all routes, useful? What additional data from the Uber Movement website can you export to better your model?

7.4 5b. Improve our model from 5a using Multiple Linear Regression

To further improve the baseline model we've created on 5a, we will add the following extra features to our model: `Latitude`, `Longitude`, `proportion lower`

[158]:
```
X_2 = match_census_pre_post.loc[:, ['Household Income by Race', 'Latitude',␣
↪'Longitude', 'proportion lower']].values  # features or predictor
y_2 = match_census_pre_post.loc[:, ['proportion before/after']].values   #␣
↪target
# X, y = remove_nans(X_regre,y_regre)

X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_2, y_2,␣
↪test_size=0.3, random_state=0)

model_2 = LinearRegression().fit(X_train_2, y_train_2)
# y_pred_1 = model_1.predict(X_test_1)

model_2.score(X_test_2, y_test_2), model_2.score(X_train_2, y_train_2)
```

[158]: (0.5641835625657587, 0.5667251597547145)

[232]:
```
mean_squared_error(y_test_2, y_pred_2)
```

[232]: 0.0179205718545065

[159]:
```
model_lasso = Lasso(alpha=0.01)
model_lasso.fit(X_train_2, y_train_2)
pred_train_lasso= model_lasso.predict(X_train_2)
print(np.sqrt(mean_squared_error(y_train_2,pred_train_lasso)))
print(r2_score(y_train_2, pred_train_lasso))

pred_test_lasso= model_lasso.predict(X_test_2)
print(np.sqrt(mean_squared_error(y_test_2, pred_test_lasso)))
print(r2_score(y_test_2, pred_test_lasso))
```

```
0.16475283065355398
0.33722067219096297
0.16510718517474884
0.33704555174455486
```

[160]:
```
y_5b_1 = match_census_pre_post.loc[:, ['proportion before/after']]
x_5b_1 = sm.add_constant(match_census_pre_post.loc[:, ['Household Income by␣
↪Race', 'Latitude', 'Longitude','proportion lower']])
model_ols_5b_1 = sm.OLS(y_5b_1.astype(float), x_5b_1.astype(float))#.fit()
results_5b_1 = model_ols_5b_1.fit(cov_type = 'HC1')
results_5b_1.summary()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142:
```

```
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

[160]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               OLS Regression Results
================================================================================
===
Dep. Variable:     proportion before/after   R-squared:
0.566
Model:                                 OLS   Adj. R-squared:
0.566
Method:                      Least Squares   F-statistic:
1.208e+05
Date:                     Mon, 13 Dec 2021   Prob (F-statistic):
0.00
Time:                             23:14:02   Log-Likelihood:
9.6287e+05
No. Observations:                  1617121   AIC:
-1.926e+06
Df Residuals:                      1617116   BIC:
-1.926e+06
Df Model:                                4
Covariance Type:                       HC1
================================================================================
============
                               coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------------
const                       21.4063      0.533     40.138      0.000      20.361
22.452
Household Income by Race -1.657e-07    2.7e-09    -61.314      0.000   -1.71e-07
-1.6e-07
Latitude                    -0.2513      0.005    -54.055      0.000      -0.260
-0.242
Longitude                    0.0766      0.004     18.045      0.000       0.068
0.085
proportion lower            -1.4621      0.003   -555.216      0.000      -1.467
-1.457
==============================================================================
Omnibus:                   196084.489   Durbin-Watson:                   0.498
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1820500.437
Skew:                          -0.234   Prob(JB):                         0.00
Kurtosis:                       8.177   Cond. No.                     5.06e+08
==============================================================================
```

```
Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
[2] The condition number is large, 5.06e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

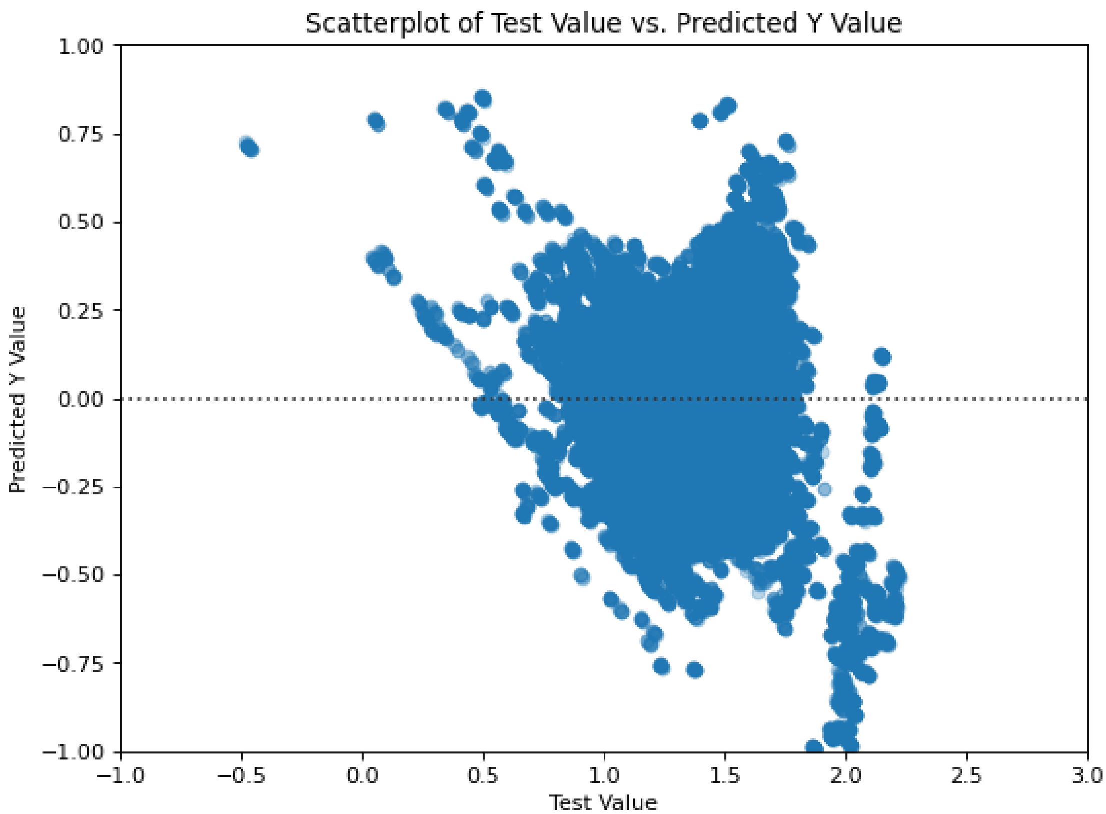
7.4.1 5.b.i. Visualization of model prediction

[161]:
```
y_pred_2 = model_2.predict(X_test_2)
```

[162]:
```
mean_squared_error(y_test_2, y_pred_2), r2_score(y_test_2, y_pred_2)
```

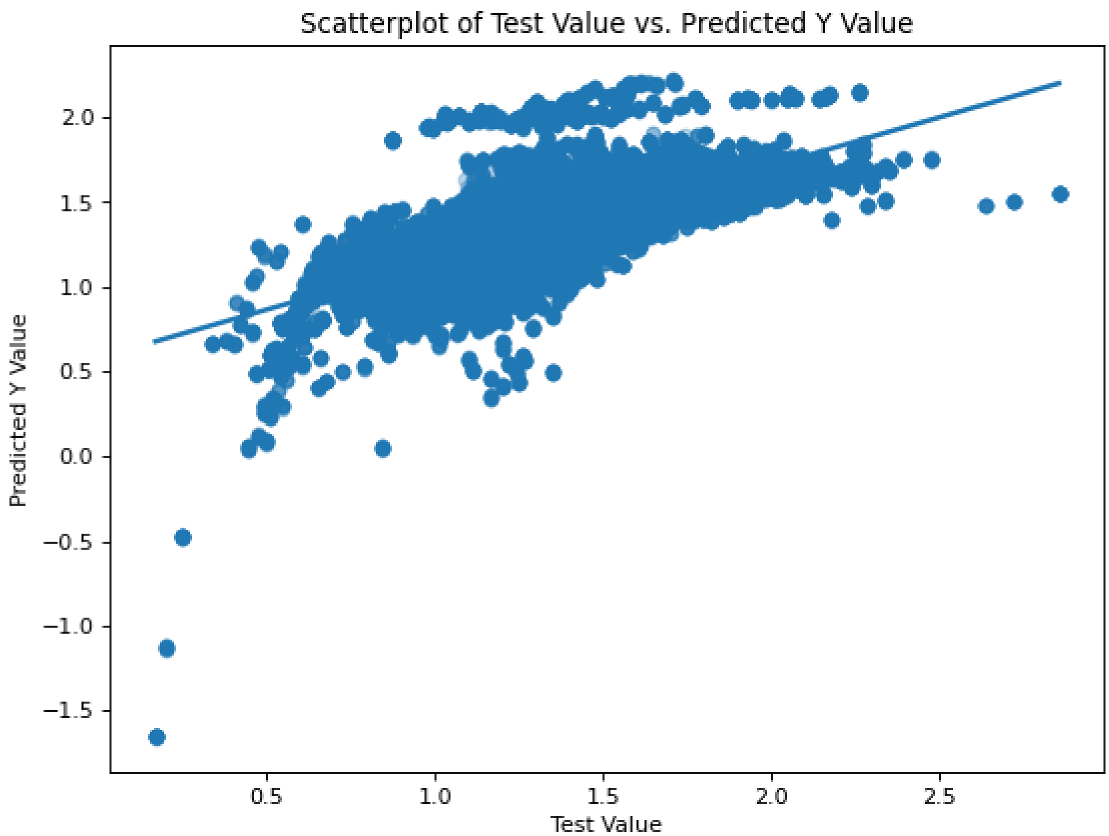
[162]:
```
(0.0179205718545065, 0.5641835625657587)
```

[235]:
```
figure(figsize=(8, 6), dpi=80)

red = sns.residplot(y_pred_2, y_test_2, scatter_kws={'alpha':0.3})
red.set_ylim(-1,1)
red.set_xlim(-1,3)

plt.xlabel("Test Value")
plt.ylabel("Predicted Y Value")
plt.title("Scatterplot of Test Value vs. Predicted Y Value");
```



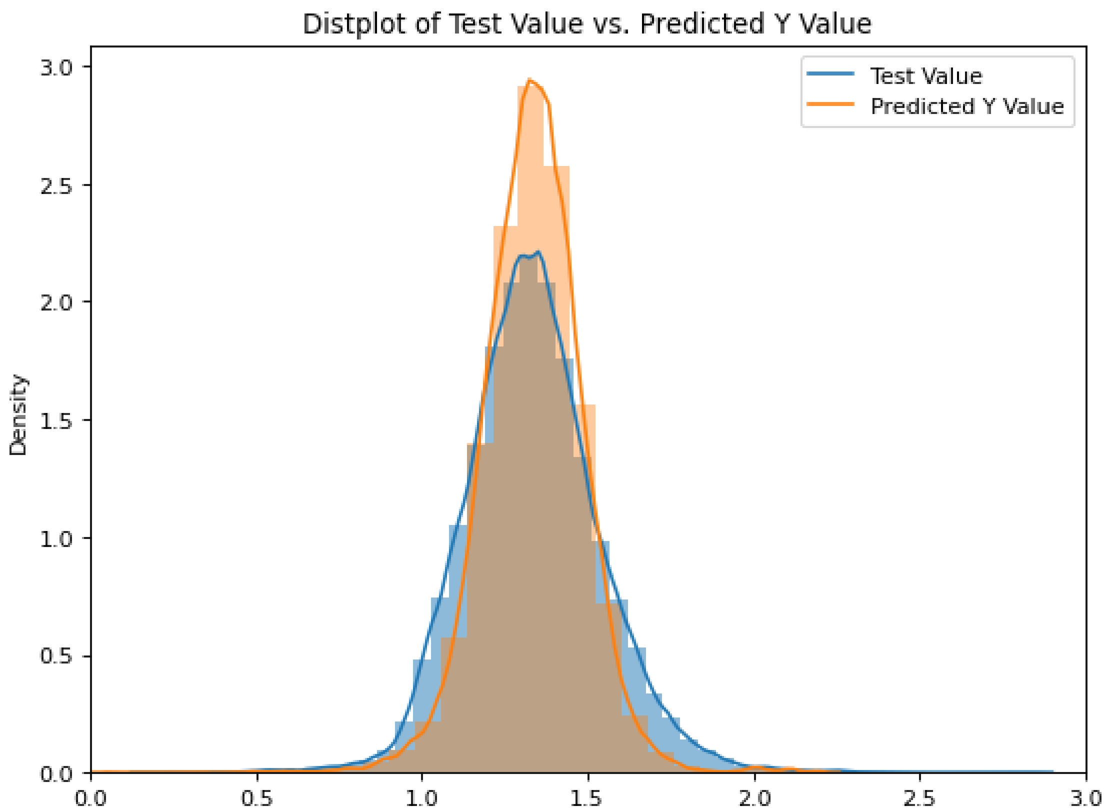
[163]:
```
figure(figsize=(8, 6), dpi=80)

sns.regplot(y_test_2, y_pred_2, scatter_kws={'alpha':0.3})
plt.xlabel("Test Value")
plt.ylabel("Predicted Y Value")
plt.title("Scatterplot of Test Value vs. Predicted Y Value");
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



[164]:
```
figure(figsize=(8, 6), dpi=80)

ax = sns.distplot(y_test_2, kde=True, hist_kws={"alpha": 0.5})
sns.distplot(y_pred_2, kde=True, hist_kws={"alpha": 0.4}, ax=ax)
plt.legend(labels=['Test Value', 'Predicted Y Value'])
ax.set_xlim(0,3)
plt.title("Distplot of Test Value vs. Predicted Y Value");
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



[165]:
```
####################################
# OLS summary, RMSE, LASSO?, 2 plots (KDE plot, scatterplot of predicted values␣
↪and residuals)
```

[166]:
```
# from sklearn.metrics import r2_score, mean_squared_error

# mean_squared_error(y_test_1, y_pred), r2_score(y_test_1, y_pred)
```

[167]:
```
# model_regre.intercept_ , model_regre.coef_
```

[168]:
```
# import seaborn as sns
# y_pred, y_test
```

[169]:
```
# sns.distplot(y_pred, color='green')
# plt.show()
```

[170]:
```
# fig, ax = plt.subplots()
# seaborn.kdeplot(y_test, ax=ax)
# seaborn.kdeplot(y_pred, ax=ax)
```

[171]:
```
# np.var(y_train), np.var(y_pred)
```

[172]:
```python
# sns.regplot(y_test, y_pred, ci=None)
```

[173]:
```python
# fig = plt.figure(figsize=(12,8))
# fig = sm.graphics.plot_regress_exog(model_2, 'proportion after/before',␣
↪fig=fig)
```

7.5 Question 5c

Explore other modeling aspects and/or temporal information. You are free to relate this to your hypothesis or not. Please expand into multiple parts that logically separate and break down your modeling work!

Example

For example, explore change across time, before and after the lockdown: (a) train and evaluate on *pre*-lockdown traffic travel times for that route; and (b) evaluate your model on *post*-lockdown traffic patterns. How would you correct your model for a more accurate post-lockdown traffic predictor? *The above is just a suggestion. You may pick any topic you find interesting.*

[174]:
```python
# One-hot encoding for temporal split (weekend / weekday)
```

[175]:
```python
pre_named.head(3)
```

[175]:
```
   MOVEMENT_ID                                      DISPLAY_NAME  \
0            9          500 Hyde Street, Tenderloin, San Francisco
1           20    900 Sutter Street, Lower Nob Hill, San Francisco
2           21  3400 Pierce Street, Marina District, San Franc…

                                            geometry  Origin Movement ID  \
0  MULTIPOLYGON (((-122.41827 37.78704, -122.4150…                1277
1  MULTIPOLYGON (((-122.42208 37.78847, -122.4153…                1277
2  MULTIPOLYGON (((-122.44191 37.80374, -122.4371…                1277

                          Origin Display Name  Destination Movement ID  \
0  300 Hayes Street, Civic Center, San Francisco                        9
1  300 Hayes Street, Civic Center, San Francisco                       20
2  300 Hayes Street, Civic Center, San Francisco                       21

                          Destination Display Name  \
0         500 Hyde Street, Tenderloin, San Francisco
1   900 Sutter Street, Lower Nob Hill, San Francisco
2  3400 Pierce Street, Marina District, San Franc…

                                        Date Range  Mean Travel Time (Seconds)  \
0  3/1/2020 - 3/1/2020, Every day, Daily Average                          322
1  3/1/2020 - 3/1/2020, Every day, Daily Average                          291
2  3/1/2020 - 3/1/2020, Every day, Daily Average                          635
```

```
   Range - Lower Bound Travel Time (Seconds)  \
0                                        211
1                                        179
2                                        438

   Range - Upper Bound Travel Time (Seconds)  day
0                                        489    1
1                                        470    1
2                                        920    1
```

[176]:
```
match_census_pre= pre_named.sjoin(merge_data, how="inner")
```

[177]:
```
match_census_pre['is_weekend'] = match_census_pre['day'].isin([1, 7, 8, 14, 15,␣
 ↪21, 22, 28, 29]).astype(int)
match_census_pre.head(3)
```

[177]:
```
     MOVEMENT_ID                                DISPLAY_NAME  \
0              9   500 Hyde Street, Tenderloin, San Francisco
121          644  200 Jones Street, Tenderloin, San Francisco
229         1245  500 Geary Street, Tenderloin, San Francisco

                                              geometry  Origin Movement ID  \
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…                1277
121  MULTIPOLYGON (((-122.41443 37.78466, -122.4127…                1277
229  MULTIPOLYGON (((-122.41500 37.78745, -122.4133…                1277

                              Origin Display Name  Destination Movement ID  \
0    300 Hayes Street, Civic Center, San Francisco                        9
121  300 Hayes Street, Civic Center, San Francisco                      644
229  300 Hayes Street, Civic Center, San Francisco                     1245

                        Destination Display Name  \
0     500 Hyde Street, Tenderloin, San Francisco
121  200 Jones Street, Tenderloin, San Francisco
229  500 Geary Street, Tenderloin, San Francisco

                                      Date Range  \
0    3/1/2020 - 3/1/2020, Every day, Daily Average
121  3/1/2020 - 3/1/2020, Every day, Daily Average
229  3/1/2020 - 3/1/2020, Every day, Daily Average

     Mean Travel Time (Seconds)  Range - Lower Bound Travel Time (Seconds)  \
0                           322                                        211
121                         356                                        221
229                         368                                        255

     Range - Upper Bound Travel Time (Seconds)  day  index_right  \
```

```
0                                                   489     1          227
121                                                 571     1          227
229                                                 529     1          227

     Household Income by Race           NAMELSAD10  \
0                       15272  Census Tract 123.01
121                     15272  Census Tract 123.01
229                     15272  Census Tract 123.01

     Household Income by Race Moe  is_weekend
0                          2872.0           1
121                        2872.0           1
229                        2872.0           1
```

[178]:
```
# from sklearn.cross_validation import train_test_split
```

[179]:
```
X_5c = match_census_pre.loc[:, ['Mean Travel Time (Seconds)']]
y_5c = match_census_pre.loc[:, ['is_weekend']]
```

[180]:
```
X_train_5c, X_test_5c, y_train_5c, y_test_5c = train_test_split(X_5c,y_5c,␣
 ↪test_size=0.25,random_state=0)

# instantiate the model (using the default parameters)
reg_5c = LogisticRegression()

# fit the model with data
reg_5c.fit(X_train_5c, y_train_5c)

y_pred_5c = reg_5c.predict(X_test_5c)
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

[181]:
```
# y_pred_5c[y_pred_5c[:] == 1]
sum(y_pred_5c)
```

[181]: 0

[182]:
```
reg_5c.score(X_test_5c, y_test_5c), reg_5c.score(X_train_5c, y_train_5c)
```

[182]: (0.7672268037679859, 0.7700138026224983)

[183]:
```
model_lasso = Lasso(alpha=0.01)
model_lasso.fit(X_train_5c, y_train_5c)
```

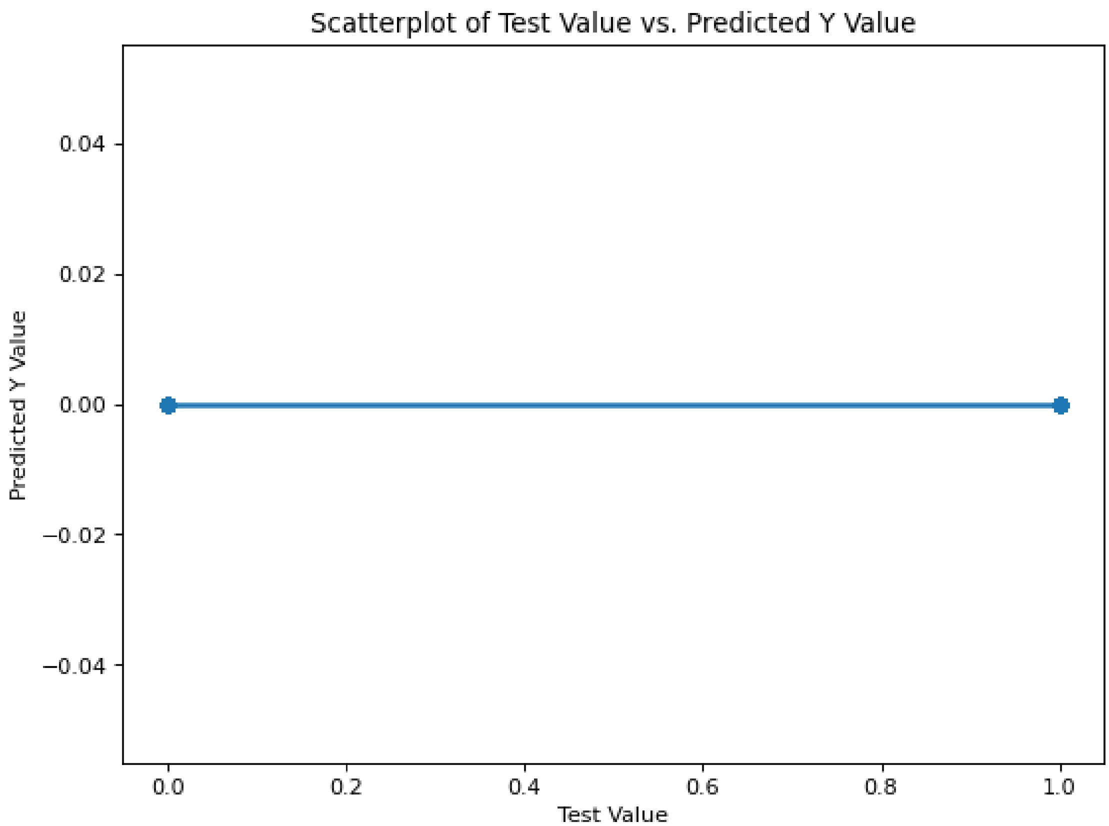
```python
pred_train_lasso= model_lasso.predict(X_train_5c)
print(np.sqrt(mean_squared_error(y_train_5c,pred_train_lasso)))
print(r2_score(y_train_5c, pred_train_lasso))

pred_test_lasso= model_lasso.predict(X_test_5c)
print(np.sqrt(mean_squared_error(y_test_5c, pred_test_lasso)))
print(r2_score(y_test_5c, pred_test_lasso))
```

```
0.4192692362393678
0.007373850357853873
0.4207529103228416
0.008717314732789694
```

[184]:

```python
figure(figsize=(8, 6), dpi=80)

sns.regplot(y_test_5c, y_pred_5c, scatter_kws={'alpha':0.3})
plt.xlabel("Test Value")
plt.ylabel("Predicted Y Value")
plt.title("Scatterplot of Test Value vs. Predicted Y Value");
```

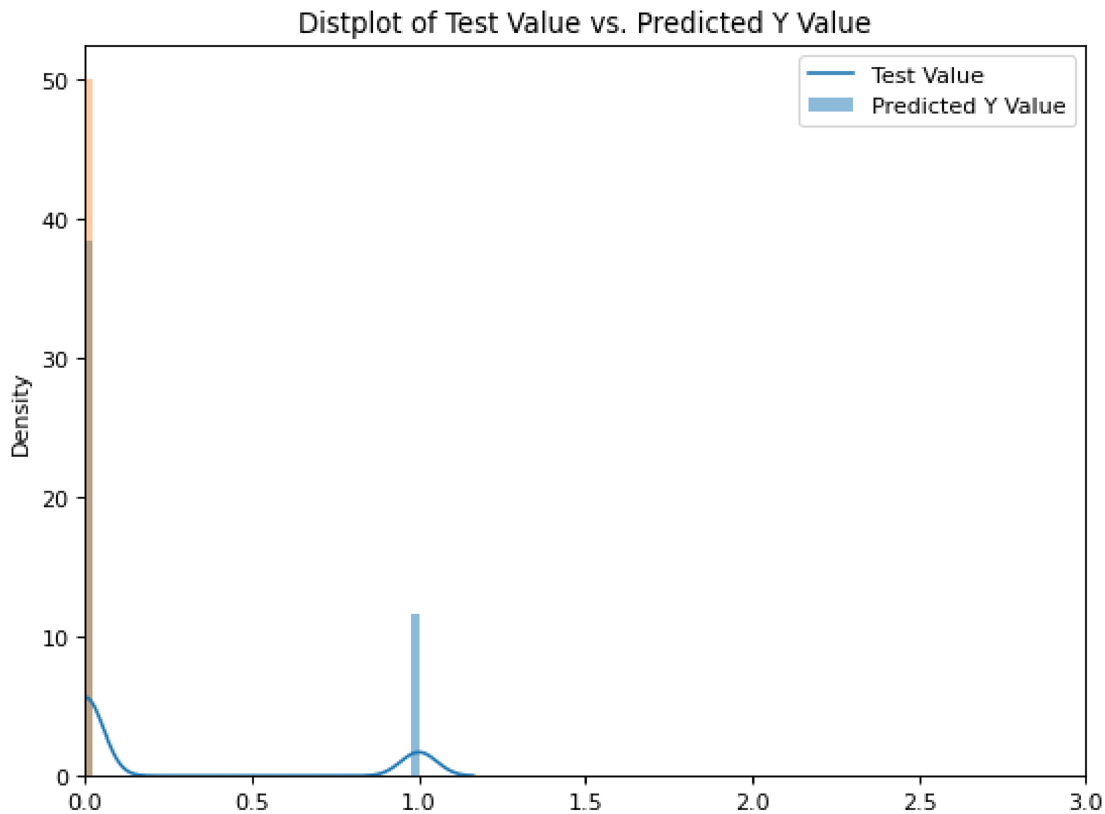
```
/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



[185]:
```
figure(figsize=(8, 6), dpi=80)

ax = sns.distplot(y_test_5c, kde=True, hist_kws={"alpha": 0.5})
sns.distplot(y_pred_5c, kde=True, hist_kws={"alpha": 0.4}, ax=ax)
plt.legend(labels=['Test Value', 'Predicted Y Value'])
ax.set_xlim(0,3)
plt.title("Distplot of Test Value vs. Predicted Y Value");
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:316:
```

```
UserWarning: Dataset has 0 variance; skipping density estimate. Pass
`warn_singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
```



[186]:
```
# ax = sns.distplot(y_test_5c, kde=True, hist_kws={"alpha": 0.5},)
# sns.distplot(y_pred_5c, kde=True, hist_kws={"alpha": 0.4}, ax=ax)
# ax.set_xlim(0,1.5)
```

[187]:
```
# plt.scatter(X_5c, y_5c)
```

[188]:
```
# import seaborn as sns
# sns.distplot(X_5c)
```

[189]:
```
# sns.distplot(y_5c)
```

[190]:
```
match_census_post = post_named.sjoin(merge_data, how="inner")
```

[191]:
```
match_census_post['is_weekend'] = match_census_post['day'].isin([1, 7, 8, 14,␣
↪15, 21, 22, 28, 29]).astype(int)
match_census_post.head(3)
```

[191]:

```
     MOVEMENT_ID                                  DISPLAY_NAME  \
0              9   500 Hyde Street, Tenderloin, San Francisco
88           644  200 Jones Street, Tenderloin, San Francisco
170         1245  500 Geary Street, Tenderloin, San Francisco

                                              geometry  Origin Movement ID  \
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…                1277
88   MULTIPOLYGON (((-122.41443 37.78466, -122.4127…                1277
170  MULTIPOLYGON (((-122.41500 37.78745, -122.4133…                1277

                          Origin Display Name  Destination Movement ID  \
0    300 Hayes Street, Civic Center, San Francisco                        9
88   300 Hayes Street, Civic Center, San Francisco                      644
170  300 Hayes Street, Civic Center, San Francisco                     1245

                        Destination Display Name  \
0     500 Hyde Street, Tenderloin, San Francisco
88   200 Jones Street, Tenderloin, San Francisco
170  500 Geary Street, Tenderloin, San Francisco

                                          Date Range  \
0    3/15/2020 - 3/15/2020, Every day, Daily Average
88   3/15/2020 - 3/15/2020, Every day, Daily Average
170  3/15/2020 - 3/15/2020, Every day, Daily Average

     Mean Travel Time (Seconds)  Range - Lower Bound Travel Time (Seconds)  \
0                           262                                        178
88                          326                                        228
170                         310                                        222

     Range - Upper Bound Travel Time (Seconds)  day  index_right  \
0                                          385   15          227
88                                         465   15          227
170                                        432   15          227

     Household Income by Race           NAMELSAD10  \
0                       15272  Census Tract 123.01
88                      15272  Census Tract 123.01
170                     15272  Census Tract 123.01

     Household Income by Race Moe  is_weekend
0                          2872.0           1
88                         2872.0           1
170                        2872.0           1
```

[192]:

```
X_5c2 = match_census_post.loc[:, ['Mean Travel Time (Seconds)']]
y_5c2 = match_census_post.loc[:, ['is_weekend']]
```

[193]:
```
X_train_5c2, X_test_5c2, y_train_5c2, y_test_5c2 =␣
  ↪train_test_split(X_5c2,y_5c2, test_size=0.25,random_state=0)

# instantiate the model (using the default parameters)
reg_5c2 = LogisticRegression()

# fit the model with data
reg_5c2.fit(X_train_5c2, y_train_5c2)

y_pred_5c2 = reg_5c2.predict(X_test_5c2)
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/utils/validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

[194]:
```
reg_5c.score(X_test_5c2, y_test_5c2), reg_5c.score(X_train_5c2, y_train_5c2)
```

[194]:
```
(0.7114723551900279, 0.7138359391732705)
```

[195]:
```
model_lasso = Lasso(alpha=0.01)
model_lasso.fit(X_train_5c2, y_train_5c2)
pred_train_lasso= model_lasso.predict(X_train_5c2)
print(np.sqrt(mean_squared_error(y_train_5c2,pred_train_lasso)))
print(r2_score(y_train_5c2, pred_train_lasso))

pred_test_lasso= model_lasso.predict(X_test_5c2)
print(np.sqrt(mean_squared_error(y_test_5c2, pred_test_lasso)))
print(r2_score(y_test_5c2, pred_test_lasso))
```

```
0.45171994691587125
0.0010930439865454167
0.45296436603357054
0.0005004207566660791
```

[196]:
```
final_match = pd.concat([match_census_pre, match_census_post])#.loc[:,␣
  ↪['MOVEMENT_ID', 'DISPLAY_NAME', 'geometry', 'Origin Movement ID', 'Mean␣
  ↪Travel Time (Seconds)', 'day','namelsad10','Household Income by␣
  ↪Race','Household Income by Race Moe','ID Geography']]
final_match['Range'] = final_match['Range - Upper Bound Travel Time (Seconds)']␣
  ↪- final_match['Range - Lower Bound Travel Time (Seconds)']
final_match.head()
```

[196]:
```
     MOVEMENT_ID                                       DISPLAY_NAME  \
0              9          500 Hyde Street, Tenderloin, San Francisco
121          644        200 Jones Street, Tenderloin, San Francisco
```

```
229        1245        500 Geary Street, Tenderloin, San Francisco
320        1674         200 Hyde Street, Tenderloin, San Francisco
326        1689  200 Sutter Street, Financial District, San Fra…

                                              geometry  Origin Movement ID  \
0    MULTIPOLYGON (((-122.41827 37.78704, -122.4150…                1277
121  MULTIPOLYGON (((-122.41443 37.78466, -122.4127…                1277
229  MULTIPOLYGON (((-122.41500 37.78745, -122.4133…                1277
320  MULTIPOLYGON (((-122.41771 37.78424, -122.4160…                1277
326  MULTIPOLYGON (((-122.40879 37.79016, -122.4071…                1277

                                Origin Display Name  Destination Movement ID  \
0    300 Hayes Street, Civic Center, San Francisco                        9
121  300 Hayes Street, Civic Center, San Francisco                      644
229  300 Hayes Street, Civic Center, San Francisco                     1245
320  300 Hayes Street, Civic Center, San Francisco                     1674
326  300 Hayes Street, Civic Center, San Francisco                     1689

                                Destination Display Name  \
0           500 Hyde Street, Tenderloin, San Francisco
121        200 Jones Street, Tenderloin, San Francisco
229        500 Geary Street, Tenderloin, San Francisco
320         200 Hyde Street, Tenderloin, San Francisco
326  200 Sutter Street, Financial District, San Fra…

                                          Date Range  \
0    3/1/2020 - 3/1/2020, Every day, Daily Average
121  3/1/2020 - 3/1/2020, Every day, Daily Average
229  3/1/2020 - 3/1/2020, Every day, Daily Average
320  3/1/2020 - 3/1/2020, Every day, Daily Average
326  3/1/2020 - 3/1/2020, Every day, Daily Average

     Mean Travel Time (Seconds)  Range - Lower Bound Travel Time (Seconds)  \
0                           322                                        211
121                         356                                        221
229                         368                                        255
320                         287                                        184
326                         546                                        365

     Range - Upper Bound Travel Time (Seconds)  day  index_right  \
0                                          489    1          227
121                                        571    1          227
229                                        529    1          227
320                                        447    1          227
326                                        816    1          227

     Household Income by Race           NAMELSAD10  \
```

```
0                       15272  Census Tract 123.01
121                     15272  Census Tract 123.01
229                     15272  Census Tract 123.01
320                     15272  Census Tract 123.01
326                     15272  Census Tract 123.01

     Household Income by Race Moe  is_weekend  Range
0                          2872.0           1    278
121                        2872.0           1    350
229                        2872.0           1    274
320                        2872.0           1    263
326                        2872.0           1    451
```

[197]:
```
stt = speeds_to_tract.loc[:, ['Latitude', 'Longitude', 'DISPLAY_NAME']]
stt.groupby('DISPLAY_NAME').agg(lambda x: list(x)[0])
```

[197]:
```
                                                     Latitude   Longitude
DISPLAY_NAME
0 16th Avenue, Hayward Park, San Mateo              37.549912 -122.325959
0 Avoca Alley, West of Twin Peaks, San Francisco    37.734415 -122.438411
0 Bass Court, Bayview, San Francisco                37.730582 -122.381439
0 Berkeley Way, Diamond Heights, San Francisco      37.735978 -122.441906
0 Bernard Street, Nob Hill, San Francisco           37.798171 -122.412108
…                                                         …          …
Old Guadalupe Trail, Daly City                      37.703571 -122.423852
Old Sled Trail, Bolinas                             37.833773 -122.483632
Petrolite Street, Richmond                          37.927252 -122.376739
Regatta Boulevard, Marina Bay, Richmond             37.925186 -122.348573
West Point Road, Bayview, San Francisco             37.733599 -122.377000

[295 rows x 2 columns]
```

[198]:
```
# final_match.merge(stt, on = 'DISPLAY_NAME', how = 'inner')
```

[199]:
```
# y_table_1 = final_match.loc[:, ['Household Income by Race', 'Range - Upper␣
↪Bound Travel Time (Seconds)','Range - Lower Bound Travel Time (Seconds)',␣
↪'intptlat10', 'intptlon10', 'day', 'Mean Travel Time (Seconds)']]
# y_table_1
```

[200]:
```
# y_table = match_census_post.loc[:, ['Household Income by Race', 'Mean Travel␣
↪Time (Seconds)']]
# y_table
```

[201]:
```
# import statsmodels.api as sm

# #est = sm.OLS(y.astype(float), X.astype(float)).fit()
```

```
# y_1c = match_census_pre['Mean Travel Time (Seconds)']
# x_1c = sm.add_constant(match_census_pre[['Household Income by Race', 'Range -␣
↪Upper Bound Travel Time (Seconds)','Range - Lower Bound Travel Time␣
↪(Seconds)', 'intptlat10', 'intptlon10', 'day', 'Mean Travel Time␣
↪(Seconds)']])
# model_1c = sm.OLS(y_1c.astype(float), x_1c.astype(float))
# results_1c = model_1c.fit(cov_type = 'HC1')
# results_1c.summary()
```

[202]:
```
# X_regre = y_table.iloc[:, :-1].values  # features or predictor
# y_regre = y_table.iloc[:, -1].values   # target
# #X, y = remove_nans(X_regre,y_regre)
```

[203]:
```
# X_train, X_test, y_train, y_test = train_test_split(X_regre, y_regre,␣
↪test_size=0.2, random_state=0)
# y_test.shape
```

[204]:
```
# model_regre = LinearRegression().fit(X_train, y_train)
# y_pred = model_regre.predict(X_test)
# y_pred
```

[205]:
```
# model_regre.score(X_test, y_test)
```

[206]:
```
# model_regre.score(X_train, y_train)
```

[207]:
```
#model_regre.summary()
```

[208]:
```
# from sklearn.metrics import r2_score, mean_squared_error

# mean_squared_error(y_test, y_pred), r2_score(y_test, y_pred)
```

[209]:
```
# model_regre.intercept_
```

[210]:
```
# model_regre.coef_
```

[211]:
```
# import seaborn as sns
# y_pred, y_test
```

[212]:
```
# sns.distplot(y_pred, color='green')
# plt.show()
```

[213]:
```
# sns.distplot(y_test, color='green')
# plt.show()
```

[214]:
```
# np.var(y_train)
```

[215]:
```
# np.var(y_pred)
```

[216]:
```
# y_table_post = match_census_post.loc[:, ['Household Income by Race', 'Range -␣
↪Upper Bound Travel Time (Seconds)','Range - Lower Bound Travel Time␣
↪(Seconds)', 'intptlat10', 'intptlon10', 'day', 'Mean Travel Time (Seconds)']]
# y_table_post
```

[217]:
```
# X_regre_1 = y_table.iloc[:, :-1].values  # features or predictor
# y_regre_1 = y_table.iloc[:, -1].values   # target
# #X, y = remove_nans(X_regre,y_regre)
```

[218]:
```
# X_train_1, X_test_1, y_train, y_test = train_test_split(X_regre_1, y_regre_1,␣
↪test_size=0.2, random_state=0)
# y_test.shape
```

[219]:
```
# final_match['thresh'] = (final_match['Household Income by Race'] > 52677).
↪astype(int)
```

[220]:
```
# new_df = pd.pivot_table(final_match, values = 'Mean Travel Time (Seconds)',␣
↪index = ['namelsad10'], columns = ['day'])
```

[221]:
```
# new_df
```

[222]:
```
# X_train_model, y_train_model, X_val_model, y_val_model =␣
↪time_series_to_dataset(new_df, 5, 5)



# model_1 = LinearRegression(fit_intercept = True)
# reg_1 = model_1.fit(X_train_model, y_train_model) # set to trained linear␣
↪model
# score_1 = reg_1.score(X_val_model, y_val_model)
# score_1
```

[223]:
```
# reg_1.accuracy_score(X_val_model, y_val_model)
```

[224]:
```
# from sklearn.model_selection import train_test_split

# X = new_df.iloc[:, :-1].values  # features or predictor
# y = new_df.iloc[:, -1].values   # target

# X, y = remove_nans(X, y)

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=0, shuffle=False)
```

[225]:
```
# model = LinearRegression().fit(X_train, y_train)
```

[226]:
```python
# y_pred = model.predict(X_test)
```

[227]:
```python
# model.score(X_test, y_test)
```

[228]:
```python
# model.score(X_train, y_train)
```

[229]:
```python
# Add more features to model, RMSE, Summary
```

To double-check your work, the cell below will rerun all of the autograder tests.

[230]:
```python
grader.check_all()
```

[230]:
```
q1a results: All test cases passed!

q1bi results: All test cases passed!

q1bii results: All test cases passed!

q1biii results: All test cases passed!

q1biv results: All test cases passed!

q1bv3 results: All test cases passed!

q1ci results: All test cases passed!

q1cii results: All test cases passed!

q1ciii results: All test cases passed!

q1civ3 results: All test cases passed!

q2ai2 results: All test cases passed!

q2ai3 results: All test cases passed!

q2ai4 results: All test cases passed!

q2aii2 results: All test cases passed!

q2aiii results: All test cases passed!

q2aiv3 results: All test cases passed!

q2av2 results: All test cases passed!
```

```
q4ai1 results: All test cases passed!

q4ai2 results: All test cases passed!

q4ai3 results: All test cases passed!

q4aii2 results: All test cases passed!

q4bi1 results: All test cases passed!

q4ci2 results: All test cases passed!

q4ci3 results: All test cases passed!

q4ci4 results:
    q4ci4 - 1 result:
        Trying:
            np.isclose(res_4ci4, 0.8616633417528182, rtol=1e-4, atol=1e-4)
        Expecting:
            True
        **********************************************************************
        Line 1, in q4ci4 0
        Failed example:
            np.isclose(res_4ci4, 0.8616633417528182, rtol=1e-4, atol=1e-4)
        Expected:
            True
        Got:
            False

q4ci5 results:
    q4ci5 - 1 result:
        Trying:
            np.isclose(res_4ci5, 0.11611253470677951, rtol=1e-4, atol=1e-4)
        Expecting:
            True
        **********************************************************************
        Line 1, in q4ci5 0
        Failed example:
            np.isclose(res_4ci5, 0.11611253470677951, rtol=1e-4, atol=1e-4)
        Exception raised:
            Traceback (most recent call last):
              File "/opt/conda/lib/python3.9/doctest.py", line 1336, in __run
                exec(compile(example.source, filename, "single",
              File "<doctest q4ci5 0[0]>", line 1, in <module>
                np.isclose(res_4ci5, 0.11611253470677951, rtol=1e-4, atol=1e-4)
            NameError: name 'res_4ci5' is not defined
```

```
q4cii results: All test cases passed!

q4ciii results: All test cases passed!

q4civ results: All test cases passed!
```

7.6 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

[236]:
```python
# Save your notebook first, then run this cell to export your submission.
grader.export()
```

```
<IPython.core.display.HTML object>
```