HW 5: Predicting Taxi Ride Duration

Due Date: 2021.6.10 (Fri) 11:59PM

Collaboration Policy

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

Collaborators: *list collaborators here*

This Assignment

In this homework, you will use what you've learned in class to create a regression model that predicts the travel time of a taxi ride in New York.

After this project, you should feel comfortable with the following:

- The data science lifecycle: data selection and cleaning, EDA, feature engineering, and model selection.
- Using `sklearn` to process data and fit linear regression models.
- Embedding linear regression as a component in a more complex model.

First, let's import:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

class bcolor:
   BLACK = '₩033[40m'
   YELLOW = '₩033[93m'
   RED = '₩033[91m'
   BOLD = '₩033[1m'
   END = '₩033[0m'

def print_passed(str_in):
  print(bcolor.BLACK + bcolor.YELLOW + bcolor.BOLD + str_in + bcolor.END)
```

Mount your Google Drive

When you run a code cell, Colab executes it on a temporary cloud instance. Every time you open the notebook, you will be assigned a different machine. All compute state and files saved on the previous machine will be lost. Therefore, you may need to re-download datasets or rerun code after a reset. Here, you can mount your Google drive to the temporary cloud instance's local filesystem using the following code snippet and save files under the specified directory (note that you will have to provide permission every time you run this).

```
# mount Google drive
from google.colab import drive
drive.mount('/content/drive')

# now you can see files
!echo -e "₩nNumber of Google drive files in /content/drive/My Drive/:"
!ls -l "/content/drive/My Drive/" | wc -l
# by the way, you can run any linux command by putting a ! at the start of the line

# by default everything gets executed and saved in /content/
!echo -e "₩nCurrent directory:"
!pwd
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c

Number of Google drive files in /content/drive/My Drive/:
172

Current directory:
/content
```

```
workspace_path = '/content/drive/MyDrive/hw5/'  # Change this path!
print(f'Current Workspace: {workspace_path}')
```

```
Current Workspace: /content/drive/MyDrive/hw5/
```

The Data

Run the following cell to load the cleaned Manhattan data.

```
manhattan_taxi = pd.read_csv(workspace_path+'manhattan_taxi.csv')
```

Attributes of all yellow taxi trips in January 2016 are published by the NYC Taxi and Limosine Commission.

Columns of the `manhattan_taxi` table include:

- `pickup_datetime`: date and time when the meter was engaged

- `dropoff_datetime`: date and time when the meter was disengaged
- `pickup_lon`: the longitude where the meter was engaged
- `pickup_lat`: the latitude where the meter was engaged
- `dropoff_lon`: the longitude where the meter was disengaged
- `dropoff_lat`: the latitude where the meter was disengaged
- `passengers`: the number of passengers in the vehicle (driver entered value)
- `distance`: trip distance
- `duration`: duration of the trip in seconds

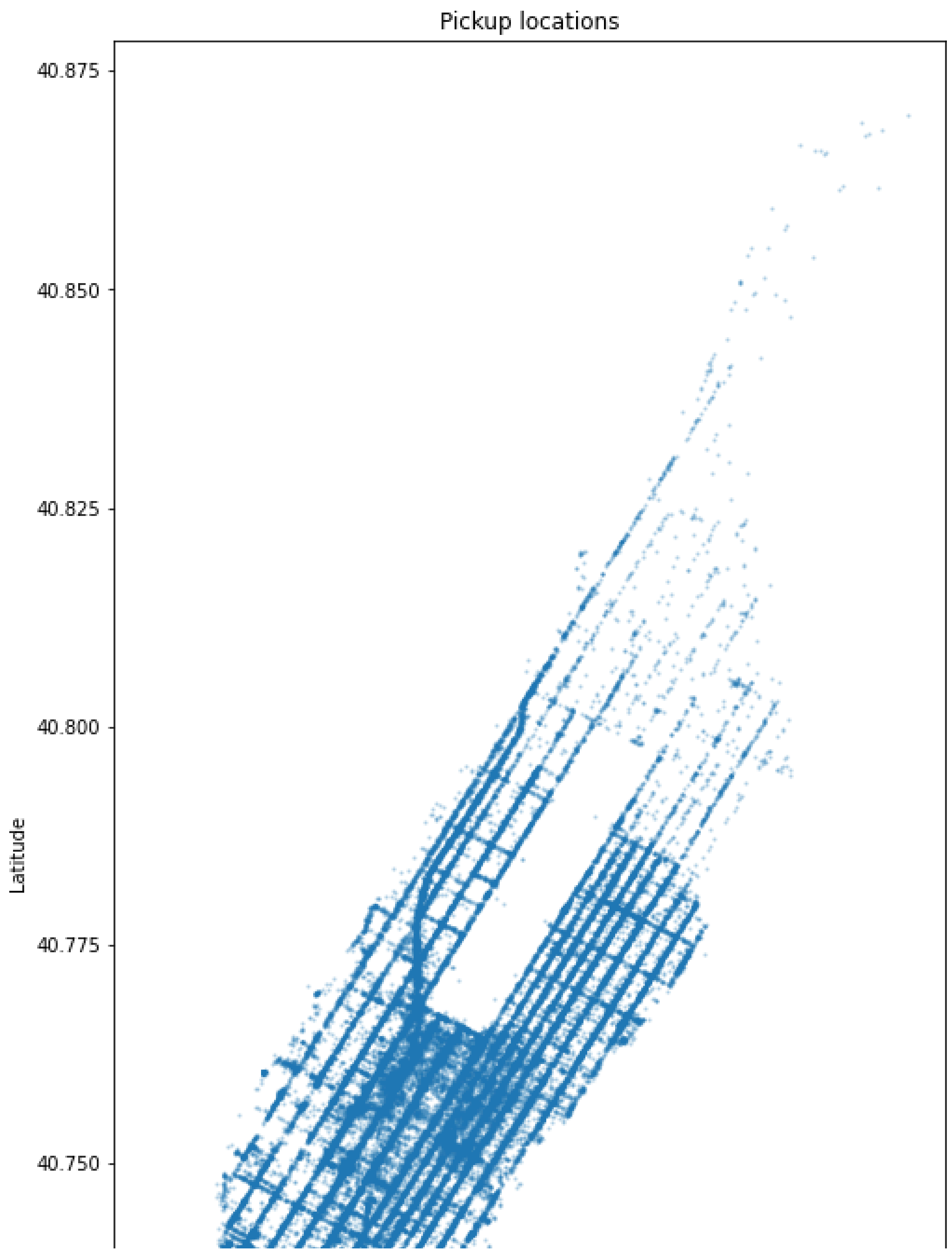
Your goal will be to predict `duration` from the pick-up time, pick-up and drop-off locations, and distance.

```
manhattan_taxi.head()
```

	pickup_datetime	dropoff_datetime	pickup_lon	pickup_lat	dropoff_lon	drop
0	2016-01-30 22:47:32	2016-01-30 23:03:53	-73.988251	40.743542	-74.015251	4
1	2016-01-04 04:30:48	2016-01-04 04:36:08	-73.995888	40.760010	-73.975388	4
2	2016-01-07 21:52:24	2016-01-07 21:57:23	-73.990440	40.730469	-73.985542	4

A scatter diagram of only Manhattan taxi rides has the familiar shape of Manhattan Island.

```
def pickup_scatter(t):
    plt.scatter(t['pickup_lon'], t['pickup_lat'], s=2, alpha=0.2)
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.title('Pickup locations')

plt.figure(figsize=(8, 16))
pickup_scatter(manhattan_taxi)
```



Part 1: Exploratory Data Analysis

In this part, you'll choose which days to include as training data in your regression model.

Your goal is to develop a general model that could potentially be used for future taxi rides. There is no guarantee that future distributions will resemble observed distributions, but some effort to limit training data to typical examples can help ensure that the training data are representative of future observations.

Note that January 2016 had some atypical days.

- New Years Day (January 1) fell on a Friday.
- Martin Luther King Jr. Day was on Monday, January 18.
- A historic blizzard passed through New York that month.

Using this dataset to train a general regression model for taxi trip times must account for these unusual phenomena, and one way to account for them is to remove atypical days from the training data

Question 1a

Add a column labeled `date` to `manhattan_taxi` that contains the date (but not the time) of pickup, formatted as a `datetime.date` value (docs).

hint: use pandas.to_datetime

```
# BEGIN YOUR CODE
# -----------------------
date_time = pd.to_datetime(manhattan_taxi['pickup_datetime'])
manhattan_taxi['date'] = date_time.dt.date
# -----------------------
# END YOUR CODE
manhattan_taxi.head()
```

	pickup_datetime	dropoff_datetime	pickup_lon	pickup_lat	dropoff_lon	drop
0	2016-01-30 22:47:32	2016-01-30 23:03:53	-73.988251	40.743542	-74.015251	4(
1	2016-01-04 04:30:48	2016-01-04 04:36:08	-73.995888	40.760010	-73.975388	4(
2	2016-01-07 21:52:24	2016-01-07 21:57:23	-73.990440	40.730469	-73.985542	4(

Explanation: Used 'to_datetime' to convert the argument into a date time and named that into date_time.

Then created a new column with the name 'date' and have that be formatted as *YYYY-MM-DD by the dt.date*

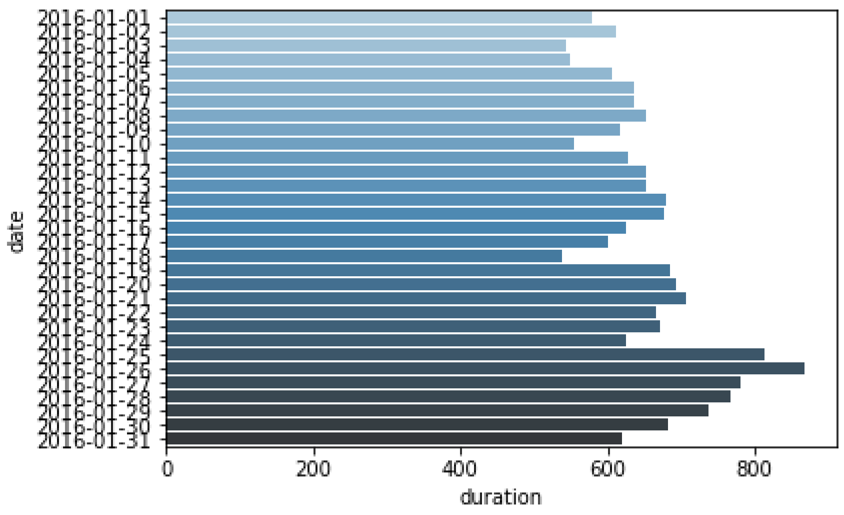
```
assert manhattan_taxi.shape == (82800, 10)
assert list(manhattan_taxi.groupby('date').size())[:8] == [2337, 2411, 2177, 2368, 2630, 2721, 2

print_passed('Q1a: Passed all unit tests!')
```

```
Q1a: Passed all unit tests!
```

Question 1b

Create a data visualization that allows you to identify which dates were affected by the historic blizzard of January 2016. Make sure that the visualization type is appropriate for the visualized data.

```
# BEGIN YOUR CODE
# -----------------------
groupby_date = manhattan_taxi.groupby(manhattan_taxi['date']).mean().reset_index()
visualization = sns.barplot(x = 'duration', y = 'date', data = groupby_date, palette = "Blues_d"
plt.show()
# -----------------------
# END YOUR CODE
```



Explanation: First, I had to make a graph grouped by the newly created column: 'date'. Then I created the data visualization using sns barplot with the data being the grouped DataFrame. Finally, I used plt.show() to display the bar plot

Finally, we have generated a list of dates that should have a fairly typical distribution of taxi rides, which excludes holidays and blizzards. The cell below assigns `final_taxi` to the subset of `manhattan_taxi` that is on these days. (No changes are needed; just run this cell.)

```
import calendar
import re

from datetime import date

atypical = [1, 2, 3, 18, 23, 24, 25, 26]
typical_dates = [date(2016, 1, n) for n in range(1, 32) if n not in atypical]
typical_dates

print('Typical dates:₩n')
pat = '  [1-3]|18 | 23| 24|25 |26 '
print(re.sub(pat, '   ', calendar.month(2016, 1)))

final_taxi = manhattan_taxi[manhattan_taxi['date'].isin(typical_dates)]
```

```
Typical dates:

    January 2016
Mo Tu We Th Fr Sa Su

 4  5  6  7  8  9 10
```

```
        11 12 13 14 15 16 17
           19 20 21 22
              27 28 29 30 31
```

Part 2: Feature Engineering

In this part, you'll create a design matrix (i.e., feature matrix) for your linear regression model. You decide to predict trip duration from the following inputs: start location, end location, trip distance, time of day, and day of the week (*Monday, Tuesday, etc.*).

You will ensure that the process of transforming observations into a design matrix is expressed as a Python function called `design_matrix`, so that it's easy to make predictions for different samples in later parts of the project.

Because you are going to look at the data in detail in order to define features, it's best to split the data into training and test sets now, then only inspect the training set.

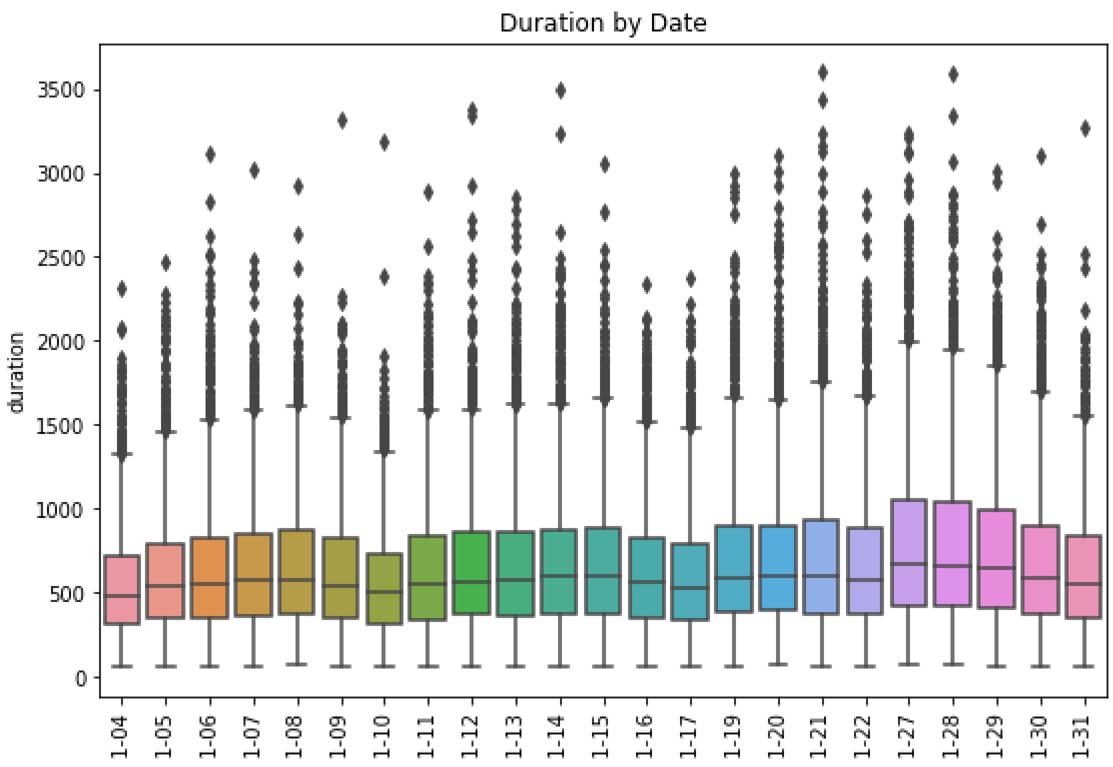
```python
import sklearn.model_selection

train, test = sklearn.model_selection.train_test_split(
    final_taxi, train_size=0.8, test_size=0.2, random_state=42)

print('Train:', train.shape, 'Test:', test.shape)
```

```
Train: (53680, 10) Test: (13421, 10)
```

Question 2a

Use `sns.boxplot` to create a box plot that compares the distributions of taxi trip durations for each day **using** `train` **only**. Individual dates shoud appear on the horizontal axis, and duration values should appear on the vertical axis. Your plot should look like this:

```python
plt.figure(figsize=(9, 6))
# BEGIN YOUR CODE
# -----------------------
train_data = train.sort_values(by= 'date')
visual = sns.boxplot(x = 'date', y= 'duration', data = train_data)
plt.title("Duration by Date")
plt.xticks(rotation = 'vertical')
plt.show()
# -----------------------
# END YOUR CODE
```



Explanation: Very similar to the previous question, but the difference is now we sort the values(by sort_values) of date, and instead of barplot we use boxplot.

I labeled the x-axis and y-axis as such but since the x-axis, displayed horizontally, was very chaotic, I used plt.xticks to rotate the labels to 'vertical'.

Finally, I gave the box plot a title and showed the visualization by plt.show()

Question 2b

In one or two sentences, describe the assocation between the day of the week and the duration of a taxi trip.

Note: The end of Part 2 showed a calendar for these dates and their corresponding days of the week.

Answer: Most of the box plot shows that the distribution is a bit more right skewed than left skewed, so we can know that the average taxi trips were longer than the medium in all days of the week. Also there seemed to be longer taxi rides during the weekdays than there were in the weekends.

Below, the provided `augment` function adds various columns to a taxi ride dataframe.

- `hour` : The integer hour of the pickup time. E.g., a 3:45pm taxi ride would have `15` as the hour. A 12:20am ride would have `0`.
- `day` : The day of the week with Monday=0, Sunday=6.
- `weekend` : 1 if and only if the `day` is Saturday or Sunday.

- `period`: 1 for early morning (12am-6am), 2 for daytime (6am-6pm), and 3 for night (6pm-12pm).
- `speed`: Average speed in miles per hour.

No changes are required; just run this cell.

```
def speed(t):
    """Return a column of speeds in miles per hour."""
    return t['distance'] / t['duration'] * 60 * 60

def augment(t):
    """Augment a dataframe t with additional columns."""
    u = t.copy()
    pickup_time = pd.to_datetime(t['pickup_datetime'])
    u.loc[:, 'hour'] = pickup_time.dt.hour
    u.loc[:, 'day'] = pickup_time.dt.weekday
    u.loc[:, 'weekend'] = (pickup_time.dt.weekday >= 5).astype(int)
    u.loc[:, 'period'] = np.digitize(pickup_time.dt.hour, [0, 6, 18])
    u.loc[:, 'speed'] = speed(t)
    return u

train = augment(train)
test = augment(test)
train.iloc[0,:] # An example row
```

```
pickup_datetime     2016-01-21 18:02:20
dropoff_datetime    2016-01-21 18:27:54
pickup_lon                   -73.994202
pickup_lat                    40.751019
dropoff_lon                  -73.963692
dropoff_lat                   40.771069
passengers                            1
distance                           2.77
duration                           1534
date                         2016-01-21
hour                                 18
day                                   3
weekend                               0
period                                3
speed                          6.500652
Name: 14043, dtype: object
```

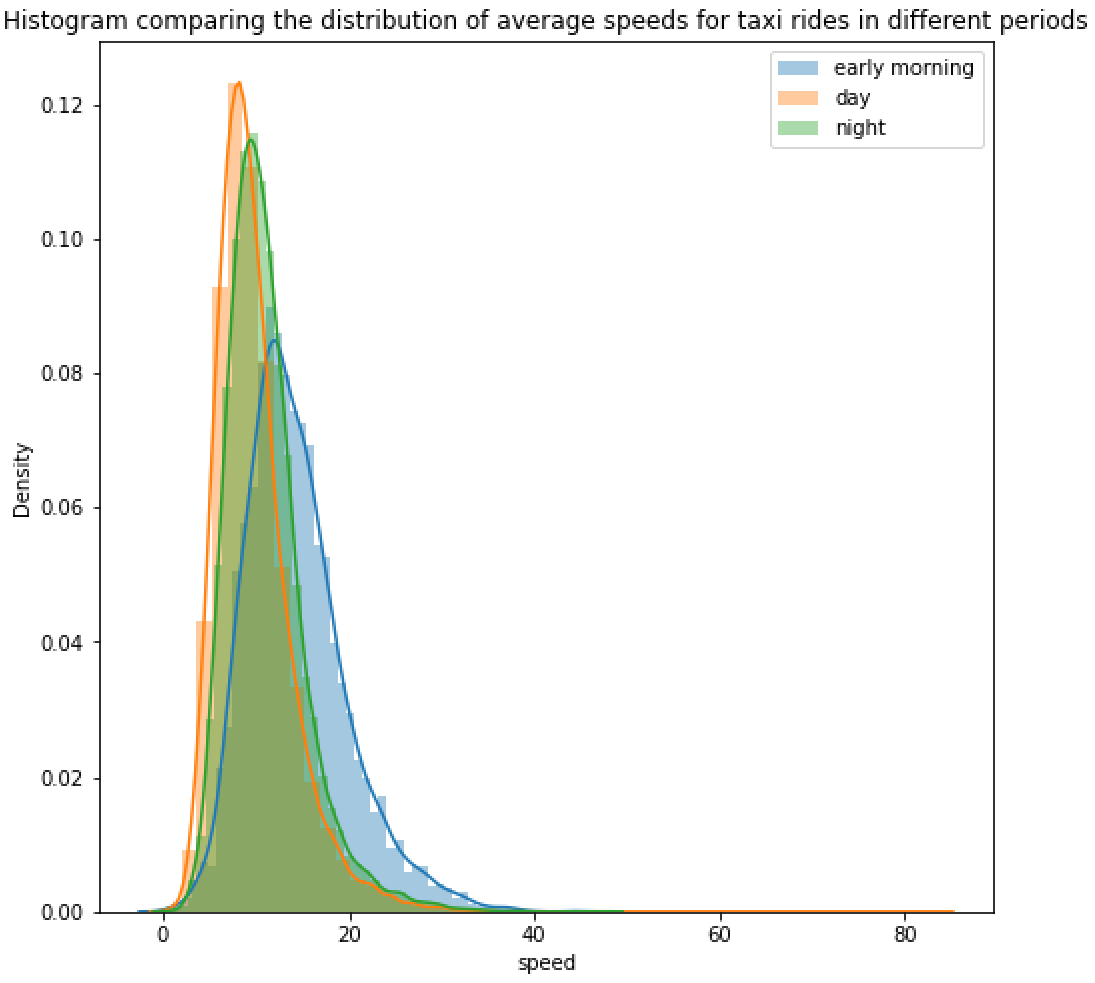
Question 2c

Use `sns.distplot` to create an overlaid histogram comparing the distribution of average speeds for taxi rides that start in the early morning (12am-6am), day (6am-6pm; 12 hours), and night (6pm-12am; 6 hours). Your plot should look like this:

```
plt.figure(figsize=(8, 8))
```

```
# BEGIN YOUR CODE
# ------------------------
sns.distplot(train[train['period'] == 1]['speed'], label ='early morning')
sns.distplot(train[train['period'] == 2]['speed'], label = 'day')
sns.distplot(train[train['period'] == 3]['speed'], label = 'night')
plt.legend()
plt.title("Histogram comparing the distribution of average speeds for taxi rides in different pe
plt.show()
# ------------------------
# END YOUR CODE
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `dist
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `dist
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `dist
  warnings.warn(msg, FutureWarning)
```



Explanation: In question 2b, it showed that period 1 is early morning, 2 is day and 3 is night. So all I had to do was create a dataframe with the appropriate constraints.

Next, I placed the sns.displot with the dataframe and had it labeled according to the right descriptions.

Finally, had to include the legend and the title.

It looks like the time of day is associated with the average speed of a taxi ride.

Question 2d (PCA)

Manhattan can roughly be divided into Lower, Midtown, and Upper regions. Instead of studying a map, let's approximate by finding the first principal component of the pick-up location (latitude and longitude).

- Add a `region` column to `train` that categorizes each pick-up location as 0, 1, or 2 based on the value of each point's first principal component, such that an equal number of points fall into each region.
- Read the documentation of `pd.qcut`, which categorizes points in a distribution into equal-frequency bins.
- You don't need to add any lines to this solution. Just fill in the assignment statements to complete the implementation.

The provided tests ensure that you have answered the question correctly.

```
# Find the first principle component
D = train[['pickup_lon', 'pickup_lat']].values
pca_n = D.shape[0]
pca_means = np.mean(D, axis=0)
X = (D - pca_means) / np.sqrt(pca_n)
u, s, vt = np.linalg.svd(X, full_matrices=False)

def add_region(t):
    """Add a region column to t based on vt above."""

    D = t[['pickup_lon', 'pickup_lat']].values
    X = (D - pca_means) / (np.sqrt(pca_n))
    assert D.shape[0] == t.shape[0], 'You set D using the incorrect table'

    # BEGIN YOUR CODE
    # -----------------------
    first_pc = X @ vt.T[0:,0]
    # -----------------------
    # END YOUR CODE
    t.loc[:,'region'] = pd.qcut(first_pc, 3, labels=[0, 1, 2])

add_region(train)
add_region(test)
```

Explanation: Using the given data transformation of X, I used that to compute the vt. Also, because it must find the very first principal component and must be a 1D array, I had to highlight that in the T[0: ,0]

```
assert np.isclose(s[0], 0.02514825, 1e-3)
assert train.shape == (53680, 16)
assert test.shape == (13421, 16)
assert list(train['region'][:8]) == [1, 1, 0, 1, 2, 1, 1, 0]
assert list(test['region'][:8]) == [2, 1, 2, 0, 1, 0, 1, 2]
assert sum(train[train['region']==1]['duration']) == 11666210
assert sum(test[test['region']==1]['duration']) == 2897696

print_passed('Q2d: Passed all unit tests!')
```

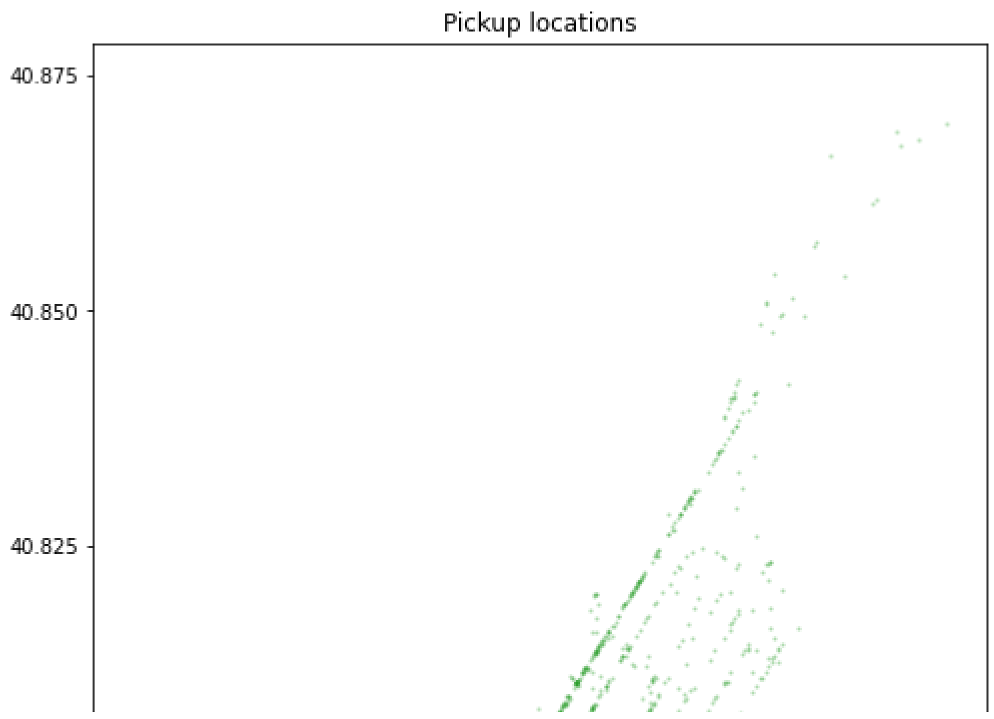
```
Q2d: Passed all unit tests!
```

Let's see how PCA divided the trips into three groups. These regions do roughly correspond to

- Lower Manhattan (below 14th street)
- Midtown Manhattan (between 14th and the park)
- Upper Manhattan (bordering Central Park).

No prior knowledge of New York geography was required!

```
plt.figure(figsize=(8, 16))
for i in [0, 1, 2]:
    pickup_scatter(train[train['region'] == i])
```



Finally, we create a design matrix that includes many of these features.

- Quantitative features are converted to standard units
- Categorical features are converted to dummy variables using one-hot encoding.

Note that,

- The `period` is not included because it is a linear combination of the `hour`.
- The `weekend` variable is not included because it is a linear combination of the `day`.
- The `speed` is not included because it was computed from the `duration` (it's impossible to know the speed without knowing the duration, given that you know the distance).

```
from sklearn.preprocessing import StandardScaler

num_vars = ['pickup_lon', 'pickup_lat', 'dropoff_lon', 'dropoff_lat', 'distance']
cat_vars = ['hour', 'day', 'region']

scaler = StandardScaler()
scaler.fit(train[num_vars])

def design_matrix(t):
    """Create a design matrix from taxi ride dataframe t."""
    scaled = t[num_vars].copy()
    scaled.iloc[:,:] = scaler.transform(scaled) # Convert to standard units
    categoricals = [pd.get_dummies(t[s], prefix=s, drop_first=True) for s in cat_vars]
    return pd.concat([scaled] + categoricals, axis=1)

design_matrix(train).iloc[0,:]
```

```
pickup_lon     -0.805821
pickup_lat     -0.171761
dropoff_lon     0.954062
dropoff_lat     0.624203
distance        0.626326
```

```
hour_1        0.000000
hour_2        0.000000
hour_3        0.000000
hour_4        0.000000
hour_5        0.000000
hour_6        0.000000
hour_7        0.000000
hour_8        0.000000
hour_9        0.000000
hour_10       0.000000
hour_11       0.000000
hour_12       0.000000
hour_13       0.000000
hour_14       0.000000
hour_15       0.000000
hour_16       0.000000
hour_17       0.000000
hour_18       1.000000
hour_19       0.000000
hour_20       0.000000
hour_21       0.000000
hour_22       0.000000
hour_23       0.000000
day_1         0.000000
day_2         0.000000
day_3         1.000000
day_4         0.000000
day_5         0.000000
day_6         0.000000
region_1      1.000000
region_2      0.000000
Name: 14043, dtype: float64
```

Part 3: Model Selection

In this part, you will select a regression model to predict the duration of a taxi ride.

Important: *Tests in this part do not confirm that you have answered correctly. Instead, they check that you're somewhat close in order to detect major errors. It is up to you to calculate the results correctly based on the question descriptions.*

Question 3a

Assign `constant_rmse` to the root mean squared error on the test set for a constant model that always predicts the mean duration of all training set taxi rides.

```
def rmse(errors):
    """Return the root mean squared error."""
    return np.sqrt(np.mean(errors ** 2))

# BEGIN YOUR CODE
# -----------------------
predicted_value = test['duration'].mean()
```

```
observed_value = test['duration']
constant_rmse = rmse(observed_value - predicted_value )
# -----------------------
# END YOUR CODE
constant_rmse
```

```
399.0372310626764
```

Explanation: To find the mean duration of all training set taxi rides, first I must retrieve the 'mean duration' or the predicted value from the dataset and the observed value of the duration.

Next, put that in the root mean squared error.

```
assert np.isclose(constant_rmse, 399.04376, 1e-4)

print_passed('Q3a: Passed all unit tests!')
```

```
Q3a: Passed all unit tests!
```

Question 3b

Assign `simple_rmse` to the root mean squared error on the test set for a simple linear regression model that uses only the distance of the taxi ride as a feature (and includes an intercept).

Terminology Note: Simple linear regression means that there is only one covariate. Multiple linear regression means that there is more than one. In either case, you can use the `LinearRegression` model from `sklearn` to fit the parameters to data.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
# BEGIN YOUR CODE
# -----------------------
x = train[['distance']]
y = train['duration']
model.fit(x,y)
duration_predict_fromDistance = model.predict(test[['distance']]) #Now to use this model to make
# -----------------------
# END YOUR CODE
errors = test['duration'] - duration_predict_fromDistance
simple_rmse = rmse(errors)
simple_rmse
```

```
276.7841105000336
```

Explanation: Using model.fit, I created the linear regression model and fitted(placed) the training data. Now the x array had to be a 2-dimensional array so I placed another [] to make it a dataframe.

Next, I had to use the ".predict" so that it can properly predict the distance feature from the model. So my **duration_predict_fromDistance** is given the distance value of the test data, it will predict the duration.

Finally, the root mean squared error(taken from the previous question) is observed - predicted. **So observed is test['duration'] and predicted is duration_predict_fromDistance**

```
assert np.isclose(simple_rmse, 276.78411, 1e-4)
print_passed('Q3b: Passed all unit tests!')
```

```
Q3b: Passed all unit tests!
```

Question 3c

Assign `linear_rmse` to the root mean squared error on the test set for a linear regression model fitted to the training set without regularization, using the design matrix defined by the `design_matrix` function from Part 3.

The provided tests check that you have answered the question correctly and that your `design_matrix` *function is working as intended.*

```
model = LinearRegression()
# BEGIN YOUR CODE
# -----------------------
X = design_matrix(train)
Y = train['duration']
model.fit(X,Y)
predict_fromDesignMatrix = model.predict(design_matrix(test)) #test set for model
# -----------------------
# END YOUR CODE
errors = test['duration'] - predict_fromDesignMatrix
linear_rmse = rmse(errors)
linear_rmse
```

```
255.19146631882776
```

Explanation: This is very similar to Question 3b, but the only difference is that the value used for predicting the duration is obtained by the design matrix, which adds more features to train the dataframe.

So we are *predicting the duration by using the model that uses design_matrix*

```
assert np.isclose(linear_rmse, 255.19147, 1e-4)
assert list(design_matrix(test).sum())[10:15] == [290.0, 511.0, 699.0, 687.0, 683.0]

print_passed('Q3c: Passed all unit tests!')
```

```
Q3c: Passed all unit tests!
```

Question 3d

For each possible value of `period`, fit an unregularized linear regression model to the subset of the training set in that `period`. Assign `period_rmse` to the root mean squared error on the test set for a model that first chooses linear regression parameters based on the observed period of the taxi ride, then predicts the duration using those parameters. Again, fit to the training set and use the `design_matrix` function for features.

```
model = LinearRegression()
errors = []

for v in np.unique(train['period']):
    # BEGIN YOUR CODE
    # -----------------------
    X = design_matrix(train[train['period'] == v])
    Y = train[train['period'] == v]['duration'] #Training set in that period
    model.fit(X,Y)
    predict_fromNewDesignMatrix = model.predict(design_matrix(test[test['period'] == v])) #test
    # -----------------------
    # END YOUR CODE
    errors.extend(test[test['period'] == v]['duration'] - predict_fromNewDesignMatrix) #test set
period_rmse = rmse(np.array(errors))
period_rmse
```

```
246.6286883116517
```

Explanation: Since it is looking for the unregularized linear regression model for each possible value of period we have to make 'period' equal v(which is the duration parameter).

So the new X value must be the design_matrix when the values of the period column equal v and the Y value should also follow that logic.

After fitting the proper x and y values into the linear regression model, now we have to make a prediction. The prediction uses the design_matrix function for features and retrieves the data from the test set where the value equals v. Now similar to the previous questions, the root mean squared error is observed - predicted so **test[test['period'] == v]['duration'] - predict_fromNewDesignMatrix**

Important thing is that the result is an array so to initialize the array, np.array is used

```
assert np.isclose(period_rmse, 246.628688, 1e-4)

print_passed('Q3d: Passed all unit tests!')
```

```
Q3d: Passed all unit tests!
```

This approach is a simple form of decision tree regression, where a different regression function is estimated for each possible choice among a collection of choices. In this case, the depth of the tree is only 1.

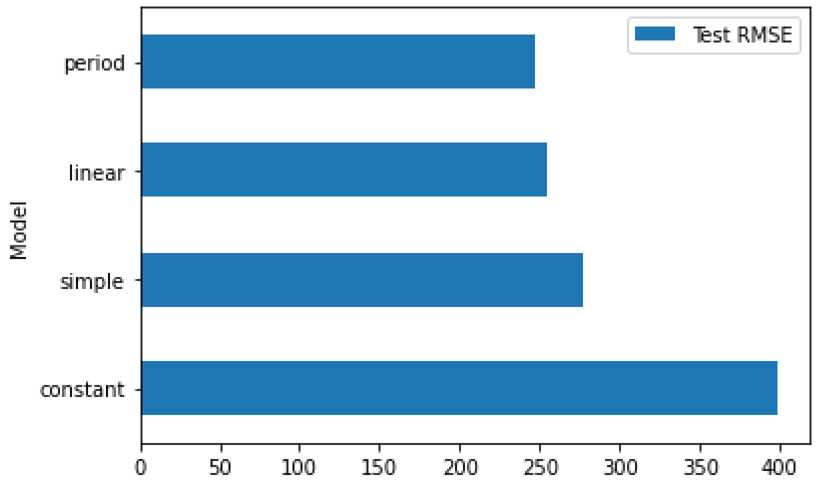
Question 3e

In one or two sentences, explain how the `period` regression model could possibly outperform linear regression when the design matrix for linear regression already includes one feature for each possible hour, which can be combined linearly to determine the `period` value.

Answer: The period regression model can outperform linear regression because the period regression model focuses on a certain feature for a duration of time (v) and the linear regression tries to fit the model with as much features as possible. However, including many features might lead to overfitting(too much variance) in the data. As a result, the small variance can help the period regression model perform better than linear regression model.

Here's a summary of your results:

```
models = ['constant', 'simple', 'linear', 'period']
pd.DataFrame.from_dict({
    'Model': models,
    'Test RMSE': [eval(m + '_rmse') for m in models]
}).set_index('Model').plot(kind='barh');
```



Congratulations!

You've carried out the entire data science lifecycle for a challenging regression problem.

- In Part 1 on `EDA`, you used the data to assess the impact of a historical event---the 2016 blizzard---and filtered the data accordingly.

- In Part 2 on `feature engineering`, you used PCA to divide up the map of Manhattan into regions that roughly corresponded to the standard geographic description of the island.
- In Part 3 on `model selection`, you found that using linear regression in practice can involve more than just choosing a design matrix. Tree regression made better use of categorical variables than linear regression. The domain knowledge that duration is a simple function of distance and speed allowed you to predict duration more accurately by first predicting speed.

Hopefully, it is apparent that all of these steps are required to reach a reliable conclusion about what inputs and model structure are helpful in predicting the duration of a taxi ride in Manhattan.

Congratulations! You have completed the last homework.

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.,

Please save before submitting!

Please generate pdf as follows and submit it to Gradescope.

File > Print Preview > Print > Save as pdf

✓ 0초 오후 5:28에 완료됨 ● ×