proj3

December 30, 2019

[1]:

```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('proj3.ok')
```

```
=====================================================================
Assignment: proj3
OK, version v1.13.11
=====================================================================
```

1 Project 3: Predicting Taxi Ride Duration

1.1 Due Date: Sunday 12/08/19, 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*

1.2 Score Breakdown

Question	Points
1a	2
1b	2
1c	2
1d	2
2a	1
2b	2
3a	2
3b	1
3c	2
3d	2
4a	2

Question	Points
4b	2
4c	2
4d	2
4e	2
4f	2
4g	4
Total	35

1.3 This Assignment

In this project, 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. Some questions in this project are more substantial than those of past projects.

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:

[2]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

1.4 The Data

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

The full data set takes a long time to download directly, so we've placed a simple random sample of the data into `taxi.db`, a SQLite database. You can view the code used to generate this sample in the `taxi_sample.ipynb` file included with this project (not required).

Columns of the `taxi` table in `taxi.db` 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.

1.5 Part 1: Data Selection and Cleaning

In this part, you will limit the data to trips that began and ended on Manhattan Island (map).

1.5.1 Question 1a

Use a SQL query to load the `taxi` table from `taxi.db` into a Pandas DataFrame called `all_taxi`.

Only include trips that have **both** pick-up and drop-off locations within the boundaries of New York City:

- Longitude is between -74.03 and -73.75 (inclusive of both boundaries)
- Latitude is between 40.6 and 40.88 (inclusive of both boundaries)

Hint: Your solution will be shorter if you write Python code to generate the SQL query string. Try not to copy and paste code.

The provided tests check that you have constructed `all_taxi` correctly.

[3]:

```
import sqlite3

conn = sqlite3.connect('taxi.db')
lon_bounds = [-74.03, -73.75]
lat_bounds = [40.6, 40.88]

pickup_bound_lon = "pickup_lon >= " + str(lon_bounds[0]) + " AND pickup_lon <=␣
 ↪" + str(lon_bounds[1])
pickup_bound_lat = "pickup_lat >= " + str(lat_bounds[0]) + " AND pickup_lat <=␣
 ↪" + str(lat_bounds[1])
dropoff_bound_lon = "dropoff_lon >= " + str(lon_bounds[0]) + " AND dropoff_lon␣
 ↪<= " + str(lon_bounds[1])
dropoff_bound_lat = "dropoff_lat >= " + str(lat_bounds[0]) + " AND dropoff_lat␣
 ↪<= " + str(lat_bounds[1])

query = "SELECT * FROM taxi WHERE " + pickup_bound_lon + " AND " +␣
 ↪pickup_bound_lat + " AND " + dropoff_bound_lon + " AND " + dropoff_bound_lat

all_taxi = pd.read_sql(query, conn)
all_taxi.head()
```

[3]:

```
       pickup_datetime     dropoff_datetime  pickup_lon  pickup_lat  \
0  2016-01-30 22:47:32  2016-01-30 23:03:53  -73.988251   40.743542
1  2016-01-04 04:30:48  2016-01-04 04:36:08  -73.995888   40.760010
2  2016-01-07 21:52:24  2016-01-07 21:57:23  -73.990440   40.730469
3  2016-01-01 04:13:41  2016-01-01 04:19:24  -73.944725   40.714539
```

```
4  2016-01-08 18:46:10  2016-01-08 18:54:00  -74.004494   40.706989

   dropoff_lon  dropoff_lat  passengers  distance  duration
0   -74.015251    40.709808           1      3.99       981
1   -73.975388    40.782200           1      2.03       320
2   -73.985542    40.738510           1      0.70       299
3   -73.955421    40.719173           1      0.80       343
4   -74.010155    40.716751           5      0.97       470
```

[4]:

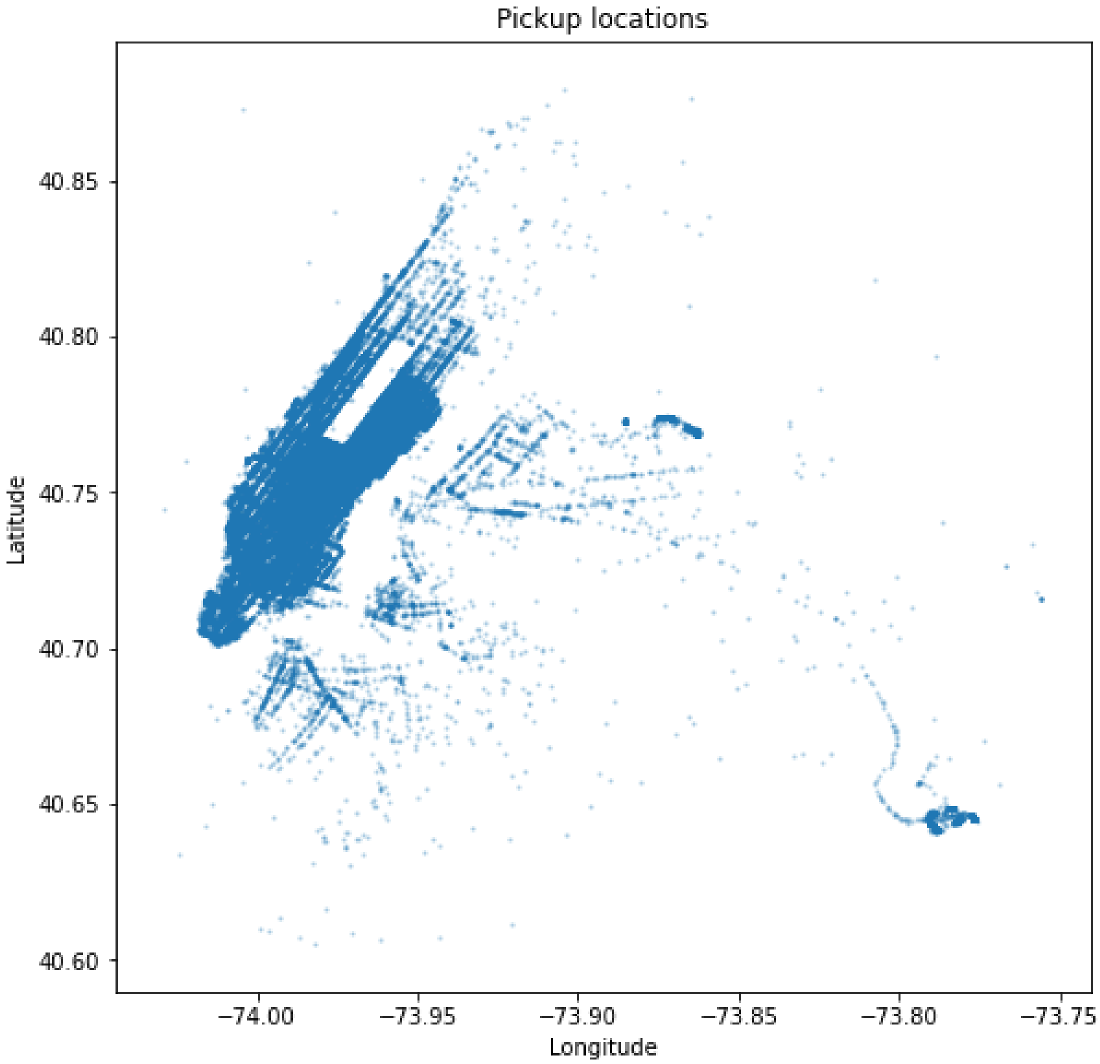
```
ok.grade("q1a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

A scatter plot of pickup locations shows that most of them are on the island of Manhattan. The empty white rectangle is Central Park; cars are not allowed there.

[5]:

```
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, 8))
pickup_scatter(all_taxi)
```



The two small blobs outside of Manhattan with very high concentrations of taxi pick-ups are airports.

1.5.2 Question 1b

Create a DataFrame called `clean_taxi` that only includes trips with a positive passenger count, a positive distance, a duration of at least 1 minute and at most 1 hour, and an average speed of at most 100 miles per hour. Inequalities should not be strict (e.g., `<=` instead of `<`) unless comparing to 0.

The provided tests check that you have constructed `clean_taxi` correctly.

[6]:
```
#pos_pass = " passengers > 0 "
#pos_dist = " distance > 0 "
```

```
#duration_bw = " duration >= 60 AND duration <= 3600 "
#avg_speed = " distance / duration <= 100"

#query = "SELECT * from taxi WHERE" + pos_pass + "AND" + pos_dist + "AND" +␣
 ↪duration_bw + "AND" + avg_speed
duration_bw = (60 <= all_taxi['duration']) & (all_taxi['duration'] <= 3600)
avg_speed = (all_taxi['distance'] / all_taxi['duration'] <= 100/3600)
clean_taxi = all_taxi[(all_taxi['passengers'] > 0) & (all_taxi['distance'] > 0)␣
 ↪& duration_bw & avg_speed]
clean_taxi
```

[6]:

```
           pickup_datetime     dropoff_datetime  pickup_lon  pickup_lat  \
0      2016-01-30 22:47:32  2016-01-30 23:03:53  -73.988251   40.743542
1      2016-01-04 04:30:48  2016-01-04 04:36:08  -73.995888   40.760010
2      2016-01-07 21:52:24  2016-01-07 21:57:23  -73.990440   40.730469
3      2016-01-01 04:13:41  2016-01-01 04:19:24  -73.944725   40.714539
4      2016-01-08 18:46:10  2016-01-08 18:54:00  -74.004494   40.706989
…                      …                    …           …         …
97687  2016-01-31 02:59:16  2016-01-31 03:09:23  -73.997391   40.721027
97688  2016-01-14 22:48:10  2016-01-14 22:51:27  -73.988037   40.718761
97689  2016-01-08 04:46:37  2016-01-08 04:50:12  -73.984390   40.754978
97690  2016-01-31 12:55:54  2016-01-31 13:01:07  -74.008675   40.725979
97691  2016-01-05 08:28:16  2016-01-05 08:54:04  -73.968086   40.799915

       dropoff_lon  dropoff_lat  passengers  distance  duration
0       -74.015251    40.709808           1      3.99       981
1       -73.975388    40.782200           1      2.03       320
2       -73.985542    40.738510           1      0.70       299
3       -73.955421    40.719173           1      0.80       343
4       -74.010155    40.716751           5      0.97       470
…              …            …          …         …        …
97687   -73.978447    40.745277           1      2.17       607
97688   -73.983337    40.726162           1      0.60       197
97689   -73.985909    40.751820           4      0.79       215
97690   -74.009598    40.716003           1      0.85       313
97691   -73.972290    40.765533           5      3.30      1548

[96445 rows x 9 columns]
```

[7]:

```
ok.grade("q1b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
```

```
    Failed: 0
[ooooooooook] 100.0% passed
```

1.5.3 Question 1c (challenging)

Create a DataFrame called `manhattan_taxi` that only includes trips from `clean_taxi` that start and end within a polygon that defines the boundaries of Manhattan Island.

The vertices of this polygon are defined in `manhattan.csv` as (latitude, longitude) pairs, which are published here.

An efficient way to test if a point is contained within a polygon is described on this page. There are even implementations on that page (though not in Python). Even with an efficient approach, the process of checking each point can take several minutes. It's best to test your work on a small sample of `clean_taxi` before processing the whole thing. (To check if your code is working, draw a scatter diagram of the (lon, lat) pairs of the result; the scatter diagram should have the shape of Manhattan.)

The provided tests check that you have constructed `manhattan_taxi` correctly. It's not required that you implement the `in_manhattan` helper function, but that's recommended. ***If you cannot solve this problem, you can still continue with the project; see the instructions below the answer cell.***

[8]:
```
polygon = pd.read_csv('manhattan.csv')

# Recommended: First develop and test a function that takes a position
#              and returns whether it's in Manhattan.
def in_manhattan(x, y):
    """Whether a longitude-latitude (x, y) pair is in the Manhattan polygon."""
    num_corners = len(polygon)
    poly_x = polygon['lon']
    poly_y = polygon['lat']
    j = num_corners-1
    odd_nodes = False
    for i in range(num_corners):
        if (poly_y[i] < y and poly_y[j] >= y) or (poly_y[j] < y and poly_y[i]␣
 ↪>= y):
            if poly_x[i]+(y-poly_y[i])/
 ↪(poly_y[j]-poly_y[i])*(poly_x[j]-poly_x[i]) < x:
                odd_nodes = not odd_nodes
        j = i
    return odd_nodes

# Recommended: Then, apply this function to every trip to filter clean_taxi.
pickup_in_manhattan = np.array([in_manhattan(x, y) for x,y in␣
 ↪clean_taxi[['pickup_lon', 'pickup_lat']].values])
```

```
dropoff_in_manhattan = np.array([in_manhattan(x, y) for x,y in␣
 ↪clean_taxi[['dropoff_lon', 'dropoff_lat']].values])
manhattan_taxi = clean_taxi[pickup_in_manhattan & dropoff_in_manhattan]
manhattan_taxi
```

```
[8]:            pickup_datetime     dropoff_datetime  pickup_lon  pickup_lat  \
     0      2016-01-30 22:47:32  2016-01-30 23:03:53  -73.988251   40.743542
     1      2016-01-04 04:30:48  2016-01-04 04:36:08  -73.995888   40.760010
     2      2016-01-07 21:52:24  2016-01-07 21:57:23  -73.990440   40.730469
     4      2016-01-08 18:46:10  2016-01-08 18:54:00  -74.004494   40.706989
     5      2016-01-02 12:39:57  2016-01-02 12:53:29  -73.958214   40.760525
     …                      …                    …         …          …
     97687  2016-01-31 02:59:16  2016-01-31 03:09:23  -73.997391   40.721027
     97688  2016-01-14 22:48:10  2016-01-14 22:51:27  -73.988037   40.718761
     97689  2016-01-08 04:46:37  2016-01-08 04:50:12  -73.984390   40.754978
     97690  2016-01-31 12:55:54  2016-01-31 13:01:07  -74.008675   40.725979
     97691  2016-01-05 08:28:16  2016-01-05 08:54:04  -73.968086   40.799915

            dropoff_lon  dropoff_lat  passengers  distance  duration
     0       -74.015251    40.709808           1      3.99       981
     1       -73.975388    40.782200           1      2.03       320
     2       -73.985542    40.738510           1      0.70       299
     4       -74.010155    40.716751           5      0.97       470
     5       -73.983360    40.760406           1      1.70       812
     …             …          …         …          …        …
     97687   -73.978447    40.745277           1      2.17       607
     97688   -73.983337    40.726162           1      0.60       197
     97689   -73.985909    40.751820           4      0.79       215
     97690   -74.009598    40.716003           1      0.85       313
     97691   -73.972290    40.765533           5      3.30      1548

     [82800 rows x 9 columns]
```

```
[9]: ok.grade("q1c");
```

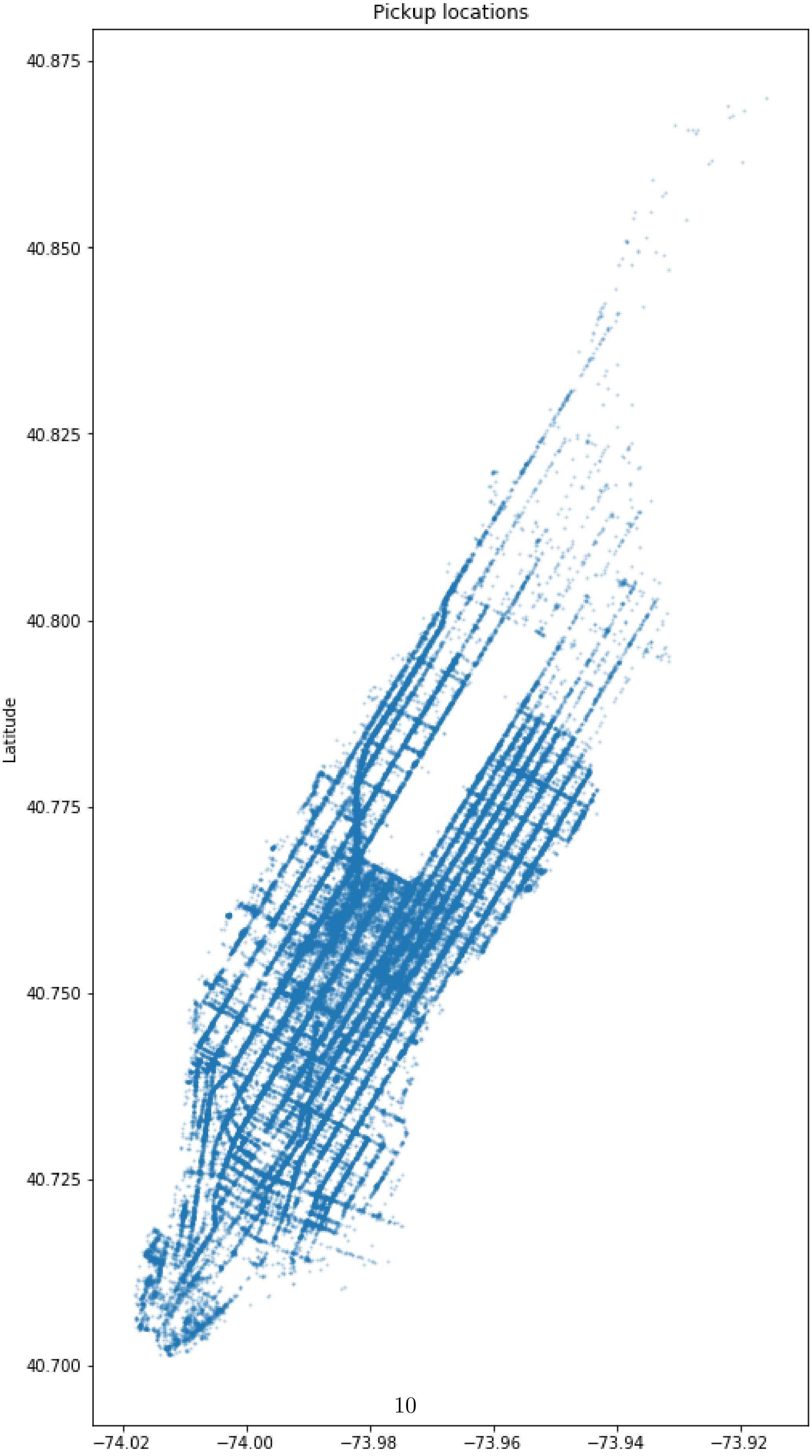
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

If you are unable to solve the problem above, have trouble with the tests, or want to work on the rest of the project before solving it, run the following cell to load the cleaned Manhattan data directly. (Note that you may not solve the previous problem just by loading this data file; you have to actually write the code.)

```
[10]: manhattan_taxi = pd.read_csv('manhattan_taxi.csv')
```

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

```
[11]: plt.figure(figsize=(8, 16))
      pickup_scatter(manhattan_taxi)
```


Pickup locations
40.875
40.850
40.825
40.800
40.775
40.750
40.725
40.700
Latitude
−74.02
−74.00
−73.98
−73.96
−73.94
−73.92
Longitude

1.5.4 Question 1d

In the following cell, print a summary of the data selection and cleaning you performed. For example, you should print something like: ”Of the original 1000 trips, 21 anomolous trips (2.1%) were removed through data cleaning, and then the 600 trips within Manhattan were selected for further analysis.” (Note that the numbers in this example are not accurate.)

Your Python code should not include any number literals, but instead should refer to the shape of `all_taxi`, `clean_taxi`, and `manhattan_taxi`. Your response will be scored based on whether you generate an accurate description and do not include any number literals in your Python expression, but instead refer to the dataframes you have created.

One way to do this is with Python’s f-strings. For instance,

```
name = "Joshua"
print(f"Hi {name}, how are you?")
```

prints `Hi Joshua, how are you?`.

Please ensure that your Python code does not contain any very long lines, or we can’t grade it.

[12]:

```
print(f"Of the original {len(all_taxi)} trips in the taxi dataset, we cleaned␣
↪out {len(all_taxi)-len(clean_taxi)} ({(len(all_taxi)-len(clean_taxi)) /␣
↪len(all_taxi)}) anomolous trips. Then the {len(manhattan_taxi)} trips within␣
↪Manhattan were selected for further analysis.")
```

```
Of the original 97692 trips in the taxi dataset, we cleaned out 1247
(0.012764607132620890) anomolous trips. Then the 82800 trips within Manhattan
were selected for further analysis.
```

1.6 Part 2: 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.

January 2016 had some atypical days. New Years Day (January 1) fell on a Friday. MLK 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.

1.6.1 Question 2a

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).

The provided tests check that you have extended `manhattan_taxi` *correctly.*

```
[13]: import datetime
      manhattan_taxi['date'] = manhattan_taxi['pickup_datetime'].str[:10].
       ↪apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%d"))
      manhattan_taxi.head()
```

```
[13]:      pickup_datetime     dropoff_datetime  pickup_lon  pickup_lat  \
      0  2016-01-30 22:47:32  2016-01-30 23:03:53  -73.988251   40.743542
      1  2016-01-04 04:30:48  2016-01-04 04:36:08  -73.995888   40.760010
      2  2016-01-07 21:52:24  2016-01-07 21:57:23  -73.990440   40.730469
      3  2016-01-08 18:46:10  2016-01-08 18:54:00  -74.004494   40.706989
      4  2016-01-02 12:39:57  2016-01-02 12:53:29  -73.958214   40.760525

         dropoff_lon  dropoff_lat  passengers  distance  duration        date
      0   -74.015251    40.709808           2      3.99       981  2016-01-30
      1   -73.975388    40.782200           1      2.03       320  2016-01-04
      2   -73.985542    40.738510           1      0.70       299  2016-01-07
      3   -74.010155    40.716751           5      0.97       470  2016-01-08
      4   -73.983360    40.760406           1      1.70       812  2016-01-02
```

```
[14]: ok.grade("q2a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

1.6.2 Question 2b

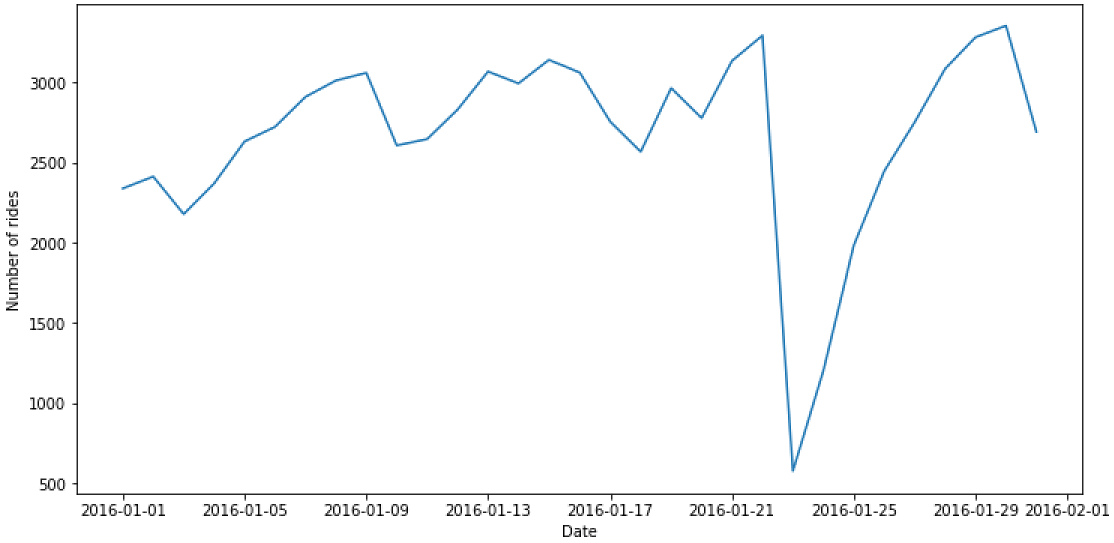
Create a data visualization that allows you to identify which dates were affected by the historic blizzard of January 2016. Make sure that the visualization type is appropriate for the visualized data.

Hint: How do you expect taxi usage to differ on blizzard days?

```
[15]: # Blizzards might cause taxi usage to drop due to severe road/traffic␣
       ↪conditions and closures.
```

```
plt.figure(figsize=(12,6))
sns.lineplot(data=manhattan_taxi.groupby('date').count()['pickup_datetime']);
plt.xlabel('Date')
plt.ylabel('Number of rides');
```

```
/srv/conda/envs/data100/lib/python3.6/site-
packages/pandas/plotting/_matplotlib/converter.py:103: FutureWarning: Using an
implicitly registered datetime converter for a matplotlib plotting method. The
converter was registered by pandas on import. Future versions of pandas will
require you to explicitly register matplotlib converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```



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.)

[16]:

```
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
```

You are welcome to perform more exploratory data analysis, but your work will not be scored. Here's a blank cell to use if you wish. In practice, further exploration would be warranted at this point, but we won't require that of you.

```
[17]: # Optional: More EDA here
```

1.7 Part 3: 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.

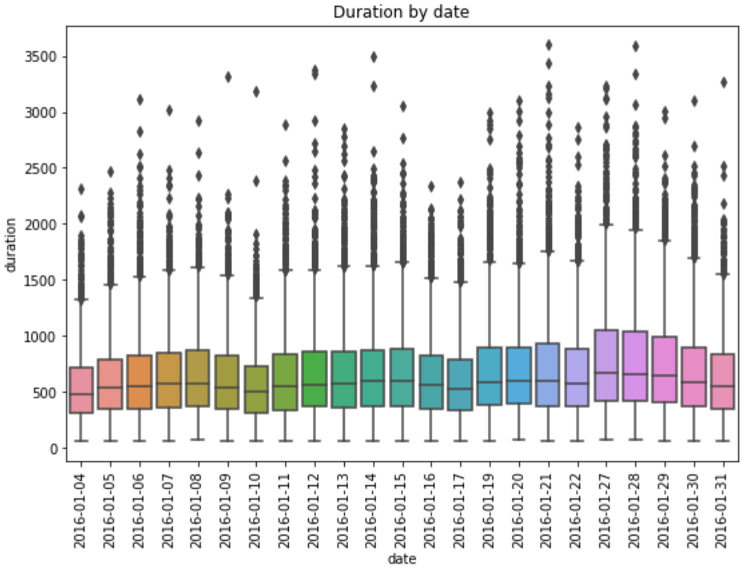
```
[18]: import sklearn.model_selection

      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)
```

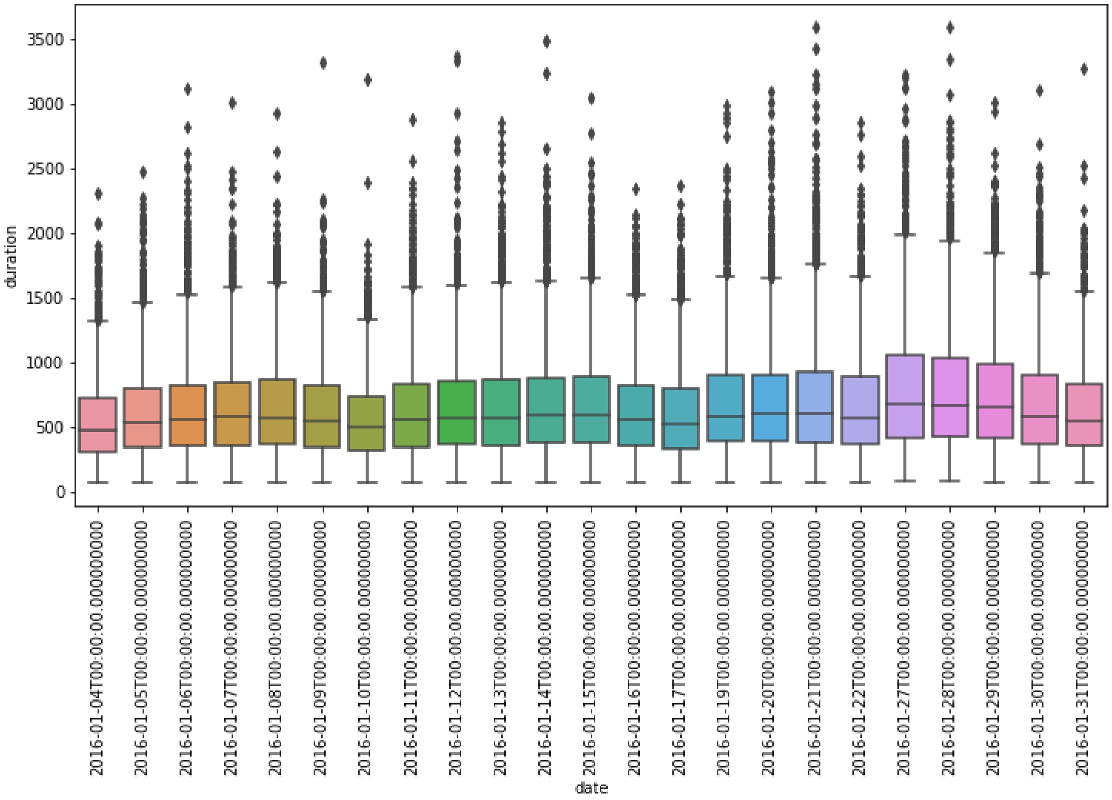
1.7.1 Question 3a

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 the following.

Hint: Use `sns.boxplot`.



[19]:
```
plt.figure(figsize=(12,6))
sns.boxplot(train['date'], train['duration'])
plt.xticks(rotation=90);
```



1.7.2 Question 3b

In one or two sentences, describe the assocation between the day of the week and the duration of a taxi trip. This question will be graded on whether your answer is justified by your boxplot and if it is at least somewhat meaningful.

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

It seems that during the weekdays, the durations of taxi trips are longer than the durations of taxi trips on weekends. In fact, in some weeks it looks like the durations increase slightly in duration, then decrease as the week closes, with the the lowest duration happening on Sunday.

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.

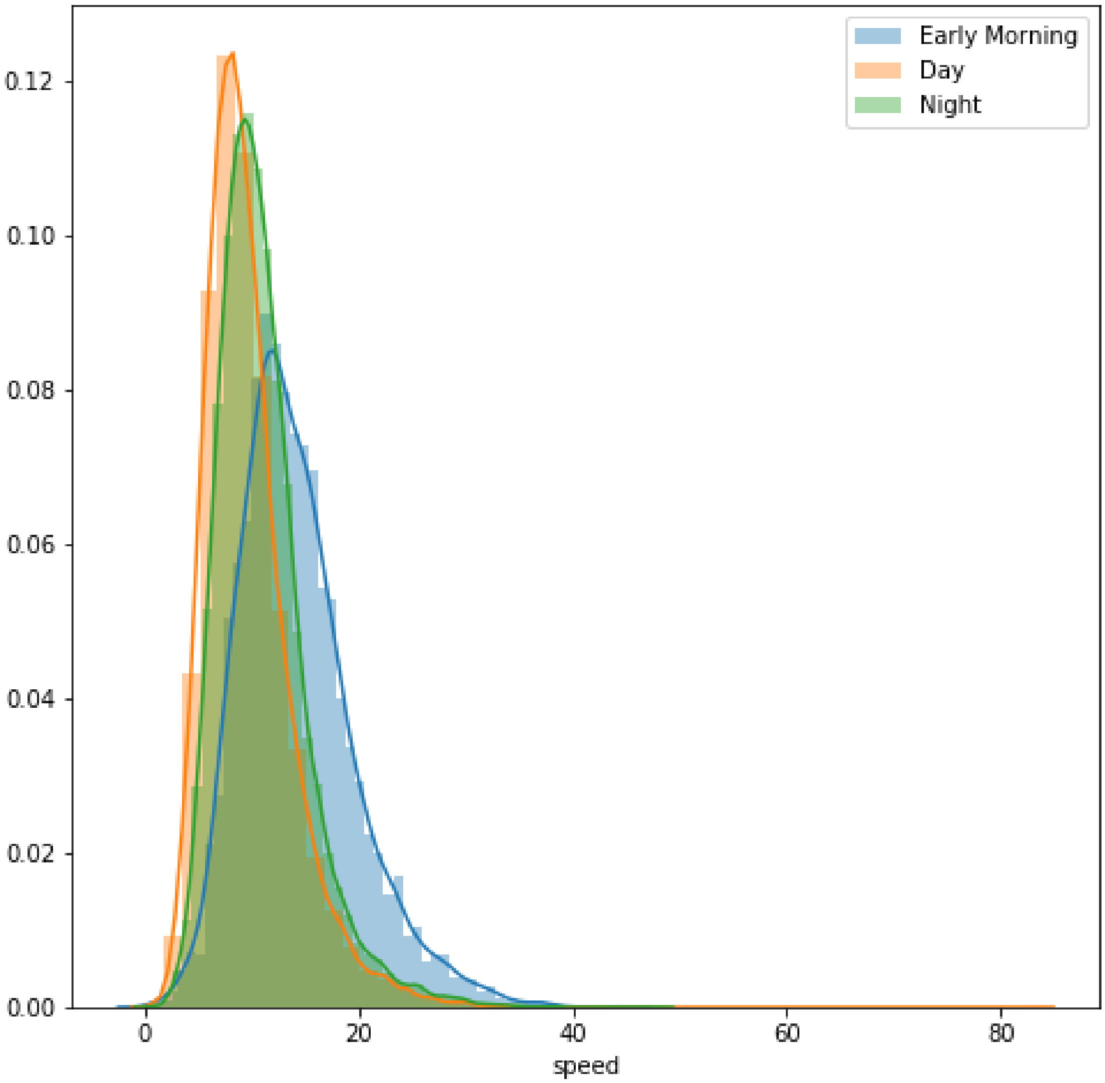
[20]:
```
def speed(t):
    """Return a column of speeds in miles per hour."""
    return t['distance'] / t['duration'] * 60 * 60

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
```

[20]:
```
pickup_datetime     2016-01-21 18:02:20
dropoff_datetime    2016-01-21 18:27:54
```

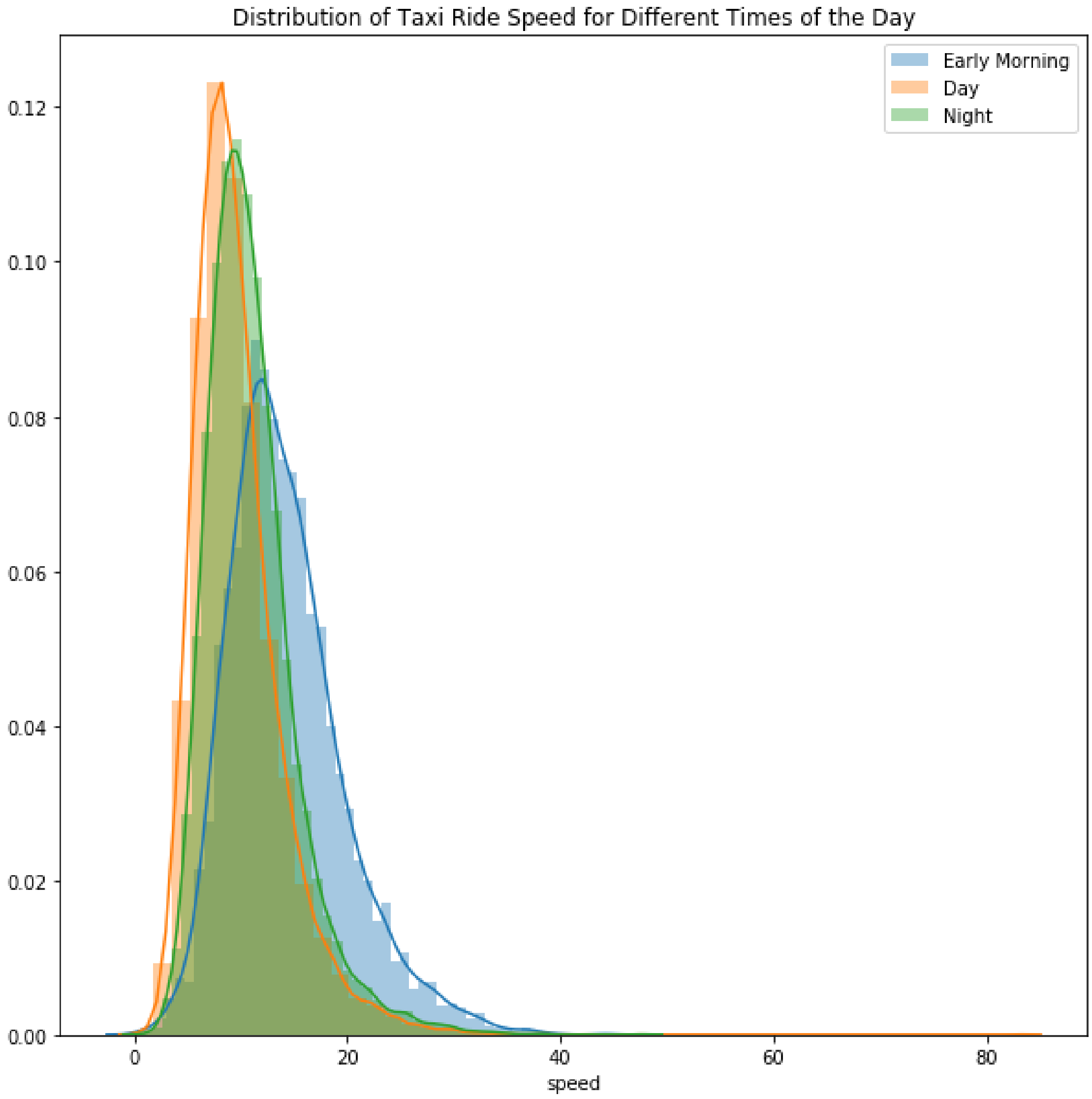
```
pickup_lon                   -73.9942
pickup_lat                     40.751
dropoff_lon                  -73.9637
dropoff_lat                   40.7711
passengers                          1
distance                         2.77
duration                         1534
date              2016-01-21 00:00:00
hour                               18
day                                 3
weekend                             0
period                              3
speed                         6.50065
Name: 14043, dtype: object
```

1.7.3 Question 3c

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:



[21]:
```
plt.figure(figsize=(10,10))
sns.distplot(train[train['hour'] < 6]['speed'], label='Early Morning')
sns.distplot(train[(train['hour'] >= 6) & (train['hour'] < 18)]['speed'],␣
 ↪label='Day')
sns.distplot(train[(train['hour'] >= 18) & (train['hour'] < 24)]['speed'],␣
 ↪label='Night')
plt.title("Distribution of Taxi Ride Speed for Different Times of the Day")
plt.legend();
```

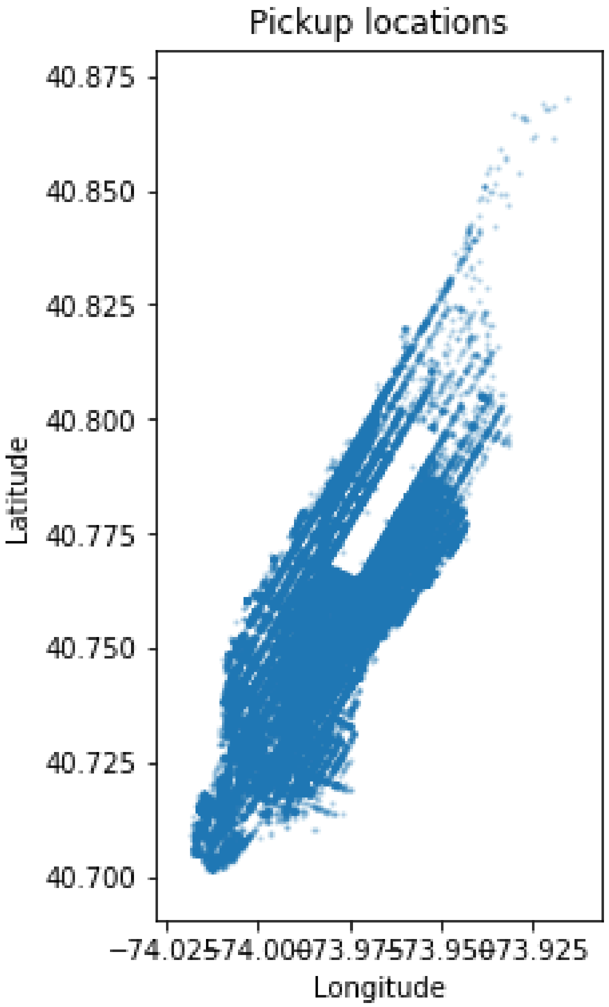


It looks like the time of day is associated with the average speed of a taxi ride.

1.7.4 Question 3d

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). Before doing that, let's once again take a look at a scatterplot of trips in Manhattan:

[22]:
```
plt.figure(figsize=(3, 6))
pickup_scatter(manhattan_taxi)
```



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.

Hint: If we have a matrix of latitudes and longitudes of each trip's pickup location, roughly in which direction should PC1 be, based on the above scatterplot?

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

[23]:
```
# Find the first principal component
D = train[['pickup_lon', 'pickup_lat']]
pca_n = len(train)
```

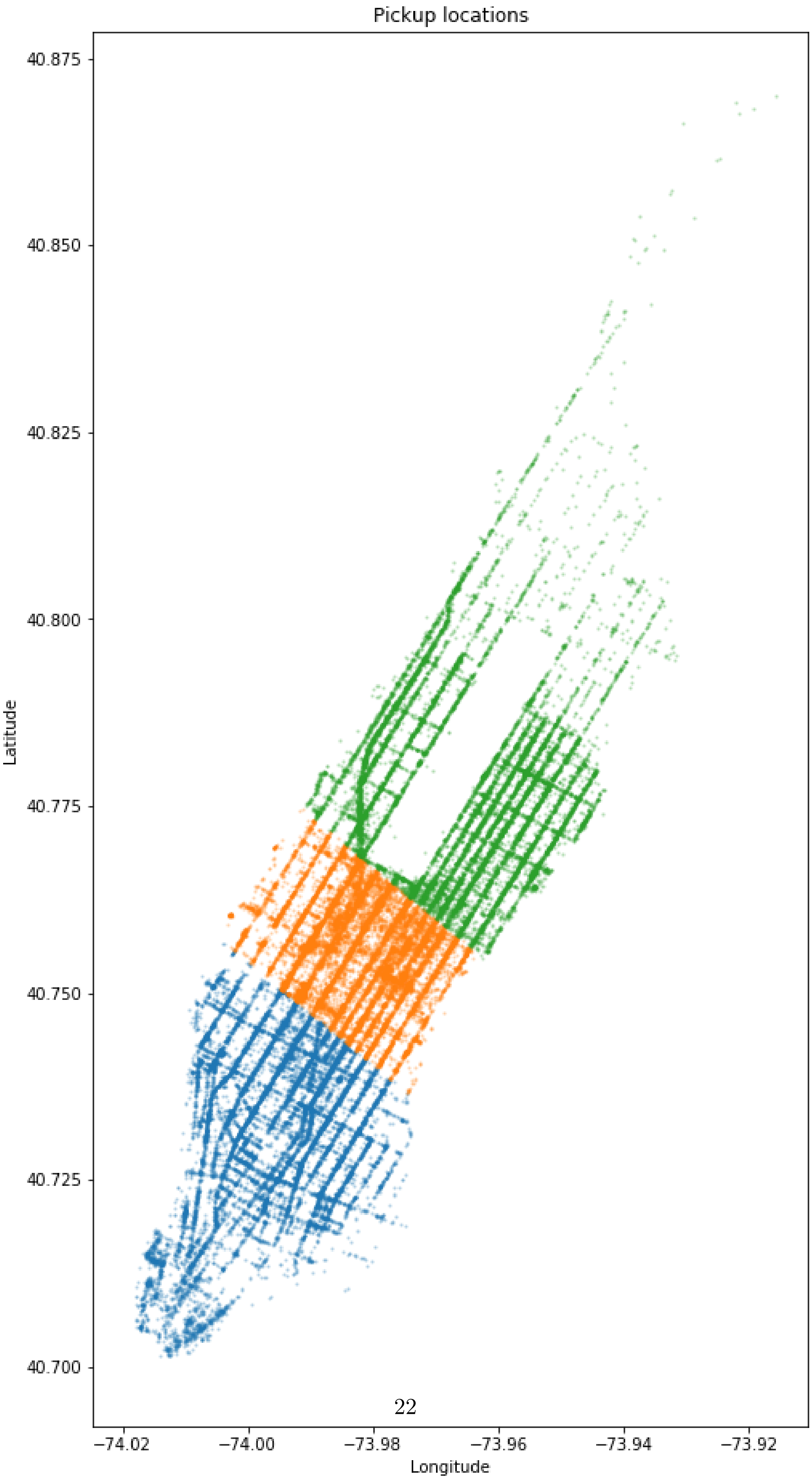
```
pca_means = D.apply(np.mean, axis=0)
X = (D - pca_means) / np.sqrt(pca_n)
u, s, vt = np.linalg.svd(X, full_matrices=False)

def add_region(t):
    """Add a region column to t based on vt above."""
    D = t[['pickup_lon', 'pickup_lat']]
    assert D.shape[0] == t.shape[0], 'You set D using the incorrect table'
    # Always use the same data transformation used to compute vt
    X = (D - pca_means) / np.sqrt(pca_n)
    first_pc = (X @ vt.T)[0]
    t.loc[:,'region'] = pd.qcut(first_pc, 3, labels=[0, 1, 2])

add_region(train)
add_region(test)
```

[24]:
```
ok.grade("q3d");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 7
    Failed: 0
[ooooooooook] 100.0% passed
```

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), and Upper Manhattan (bordering Central Park). No prior knowledge of New York geography was required!

[25]:
```
plt.figure(figsize=(8, 16))
for i in [0, 1, 2]:
    pickup_scatter(train[train['region'] == i])
```

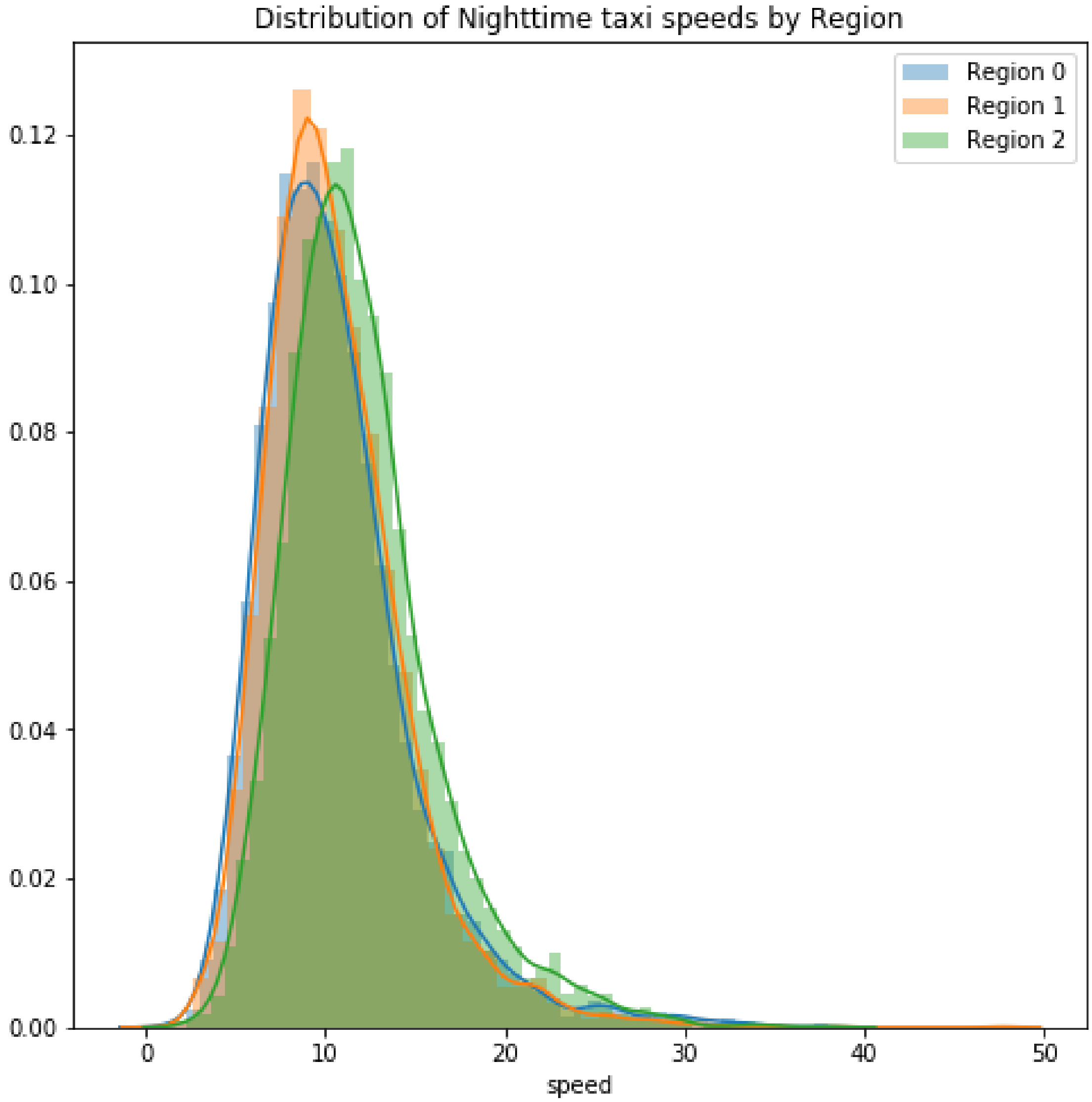

Pickup locations
Latitude
Longitude
40.875
40.850
40.825
40.800
40.775
40.750
40.725
40.700
−74.02
−74.00
−73.98
−73.96
−73.94
−73.92

1.7.5 Question 3e (ungraded)

Use `sns.distplot` to create an overlaid histogram comparing the distribution of speeds for night-time taxi rides (6pm-12am) in the three different regions defined above. Does it appear that there is an association between region and average speed during the night?

[26]:

```
plt.figure(figsize=(8,8))
for i in range(3):
    sns.distplot(train[(train['hour'] >= 18) & (train['region'] ==␣
 ↪i)]['speed'], label='Region '+str(i))
plt.legend()
plt.title("Distribution of Nighttime taxi speeds by Region");
```



Finally, we create a design matrix that includes many of these features. Quantitative features are converted to standard units, while categorical features are converted to dummy variables using one-hot encoding. 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.

[27]:

```
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,:]
```

[27]:

```
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
```

1.8 Part 4: 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.*

1.8.1 Question 4a

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.

[28]:

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

constant_rmse = rmse(test['duration'] - np.mean(train['duration']))
constant_rmse
```

[28]:

```
399.1437572352666
```

[29]:

```
ok.grade("q4a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests
```

```
---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

1.8.2 Question 4b

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 feature. 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.

[30]:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(train['distance'].values.reshape(-1,1), train['duration'])
simple_rmse = rmse(test['duration'] - model.predict(test['distance'].values.
 ↪reshape(-1,1)))
simple_rmse
```

[30]:

```
276.7841105000342
```

[31]:

```
ok.grade("q4b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

1.8.3 Question 4c

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.

[32]:

```
model = LinearRegression()
model.fit(design_matrix(train), train['duration'])
linear_rmse = rmse(test['duration'] - model.predict(design_matrix(test)))
linear_rmse
```

[32]:

```
255.19146631882754
```

[33]:

```
ok.grade("q4c");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

1.8.4 Question 4d

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.

[34]:

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

for v in np.unique(train['period']):
    model.fit(design_matrix(train[train['period'] == v]), train[train['period']␣
 ↪== v]['duration'])
    errors.extend(test[test['period']==v]['duration'] - model.
 ↪predict(design_matrix(test[test['period'] == v])))

period_rmse = rmse(np.array(errors))
period_rmse
```

[34]:

```
246.62868831165173
```

[35]:

```
ok.grade("q4d");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
```

```
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

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.

1.8.5 Question 4e

In one or two sentences, explain how the `period` regression model could possibly outperform linear regression model, even when the design matrix of the latter includes one feature for each possible hour.

Similar to the idea of question 6e on the midterm, it could be possible that the linear model is simply different, i.e., different slopes, for different values of the period.

1.8.6 Question 4f

Instead of predicting duration directly, an alternative is to predict the average *speed* of the taxi ride using linear regression, then compute an estimate of the duration from the predicted speed and observed distance for each ride.

Assign `speed_rmse` to the root mean squared error in the **duration** predicted by a model that first predicts speed as a linear combination of features from the `design_matrix` function, fitted on the training set, then predicts duration from the predicted speed and observed distance.

Hint: Speed is in miles per hour, but duration is measured in seconds. You'll need the fact that there are 60 * 60 = 3,600 seconds in an hour.

[44]:
```
model = LinearRegression()
model.fit(design_matrix(train), train['speed'])
test_design_matrix = design_matrix(test)
pred_duration = (1 / model.predict(test_design_matrix)) * test['distance'] * 3600
speed_rmse = rmse(pred_duration - test['duration'])
speed_rmse
```

[44]:
```
243.0179836851495
```

[45]:
```
ok.grade("q4f");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
```

```
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

At this point, think about why predicting speed leads to a more accurate regression model than predicting duration directly. You don't need to write your answer anywhere.

1.8.7 Question 4g

Finally, complete the function `tree_regression_errors` (and helper function `speed_error`) that combines the ideas from the two previous models and generalizes to multiple categorical variables.

The `tree_regression_errors` should: - Find a different linear regression model for each possible combination of the variables in `choices`; - Fit to the specified `outcome` (on train) and predict that `outcome` (on test) for each combination (`outcome` will be `'duration'` or `'speed'`); - Use the specified `error_fn` (either `duration_error` or `speed_error`) to compute the error in predicted duration using the predicted outcome; - Aggregate those errors over the whole test set and return them.

You should find that including each of `period`, `region`, and `weekend` improves prediction accuracy, and that predicting speed rather than duration leads to more accurate duration predictions.

Hint: Use `print` statements to try and figure out what the skeleton code does, if you're stuck.

[58]:

```python
model = LinearRegression()
choices = ['period', 'region', 'weekend']

def duration_error(predictions, observations):
    """Error between predictions (array) and observations (data frame)"""
    return predictions - observations['duration']

def speed_error(predictions, observations):
    """Duration error between speed predictions and duration observations"""
    return (1 / predictions) * observations['distance'] * 3600 -␣
 ↪observations['duration']

def tree_regression_errors(outcome='duration', error_fn=duration_error):
    """Return errors for all examples in test using a tree regression model."""
    errors = []
    for vs in train.groupby(choices).size().index:
        v_train, v_test = train, test
        for v, c in zip(vs, choices):
            v_train = v_train[v_train[c] == v]
            v_test = v_test[v_test[c] == v]
            #print(v_train.shape, v_test.shape)
        model.fit(design_matrix(v_train), v_train[outcome])
        errors.extend(error_fn(model.predict(design_matrix(v_test)), v_test))
    return errors
```

```
errors = tree_regression_errors()
errors_via_speed = tree_regression_errors('speed', speed_error)
tree_rmse = rmse(np.array(errors))
tree_speed_rmse = rmse(np.array(errors_via_speed))
print('Duration:', tree_rmse, '\nSpeed:', tree_speed_rmse)
```

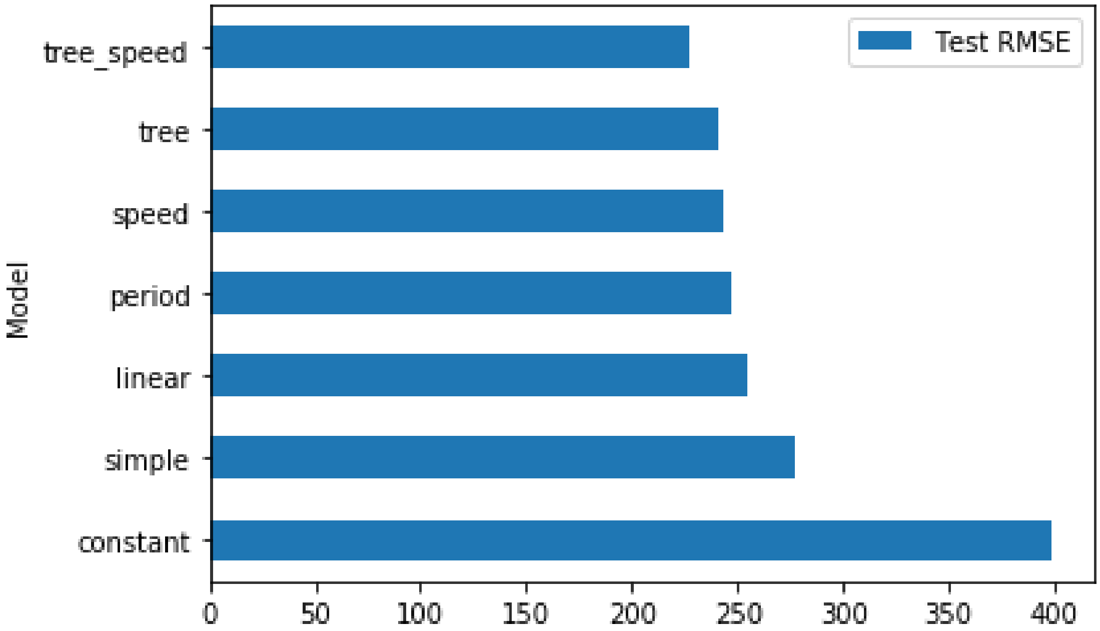
```
Duration: 240.3395219270353
Speed: 226.90793945018314
```

[59]:
```
ok.grade("q4g");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

Here's a summary of your results:

[60]:
```
models = ['constant', 'simple', 'linear', 'period', 'speed', 'tree',␣
 ↪'tree_speed']
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 data selection, you solved a domain-specific programming problem relevant to the analysis when choosing only those taxi rides that started and ended in Manhattan.

In Part 2 on EDA, you used the data to assess the impact of a historical event---the 2016 blizzard---and filtered the data accordingly.

In Part 3 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 4 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.

1.9 Future Work

Here are some questions to ponder:

- The regression model would have been more accurate if we had used the date itself as a feature instead of just the day of the week. Why didn't we do that?
- Does collecting this information about every taxi ride introduce a privacy risk? The original data also included the total fare; how could someone use this information combined with an individual's credit card records to determine their location?

- Why did we treat `hour` as a categorical variable instead of a quantitative variable? Would a similar treatment be beneficial for latitude and longitude?
- Why are Google Maps estimates of ride time much more accurate than our estimates?

Here are some possible extensions to the project:

- An alternative to throwing out atypical days is to condition on a feature that makes them atypical, such as the weather or holiday calendar. How would you do that?
- Training a different linear regression model for every possible combination of categorical variables can overfit. How would you select which variables to include in a decision tree instead of just using them all?
- Your models use the observed distance as an input, but the distance is only observed after the ride is over. How could you estimate the distance from the pick-up and drop-off locations?
- How would you incorporate traffic data into the model?

2 Submit

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!**

[62]:

```
# Save your notebook first, then run this cell to submit.
import jassign.to_pdf
jassign.to_pdf.generate_pdf('proj3.ipynb', 'proj3.pdf')
ok.submit()
```

```
Generating PDF…
Saved proj3.pdf

<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


Saving notebook… Saved 'proj3.ipynb'.
Submit… 100% complete
Submission successful for user: jcpagadora@berkeley.edu
URL: https://okpy.org/cal/data100/fa19/proj3/submissions/vQ6LV0
```

[ ]:

```
```