proj3

September 10, 2019

```
[2]: # 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: Thursday 5/2/19, 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	3
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:

[3]:
```
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 disengaged - `dropoff_datetime`: date and time when the meter was engaged - `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 40.63 and 40.85 (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.

[4]:
```python
import sqlite3

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

all_taxi = pd.read_sql('''SELECT * from taxi
WHERE pickup_lon BETWEEN -74.03 AND -73.75
AND dropoff_lon BETWEEN -74.03 AND -73.75
AND pickup_lat BETWEEN 40.6 AND 40.88
AND dropoff_lat BETWEEN 40.6 and 40.88
''', conn)
all_taxi.head()
```

[4]:
```
       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
```

[5]:
```
ok.grade("q1a");
```

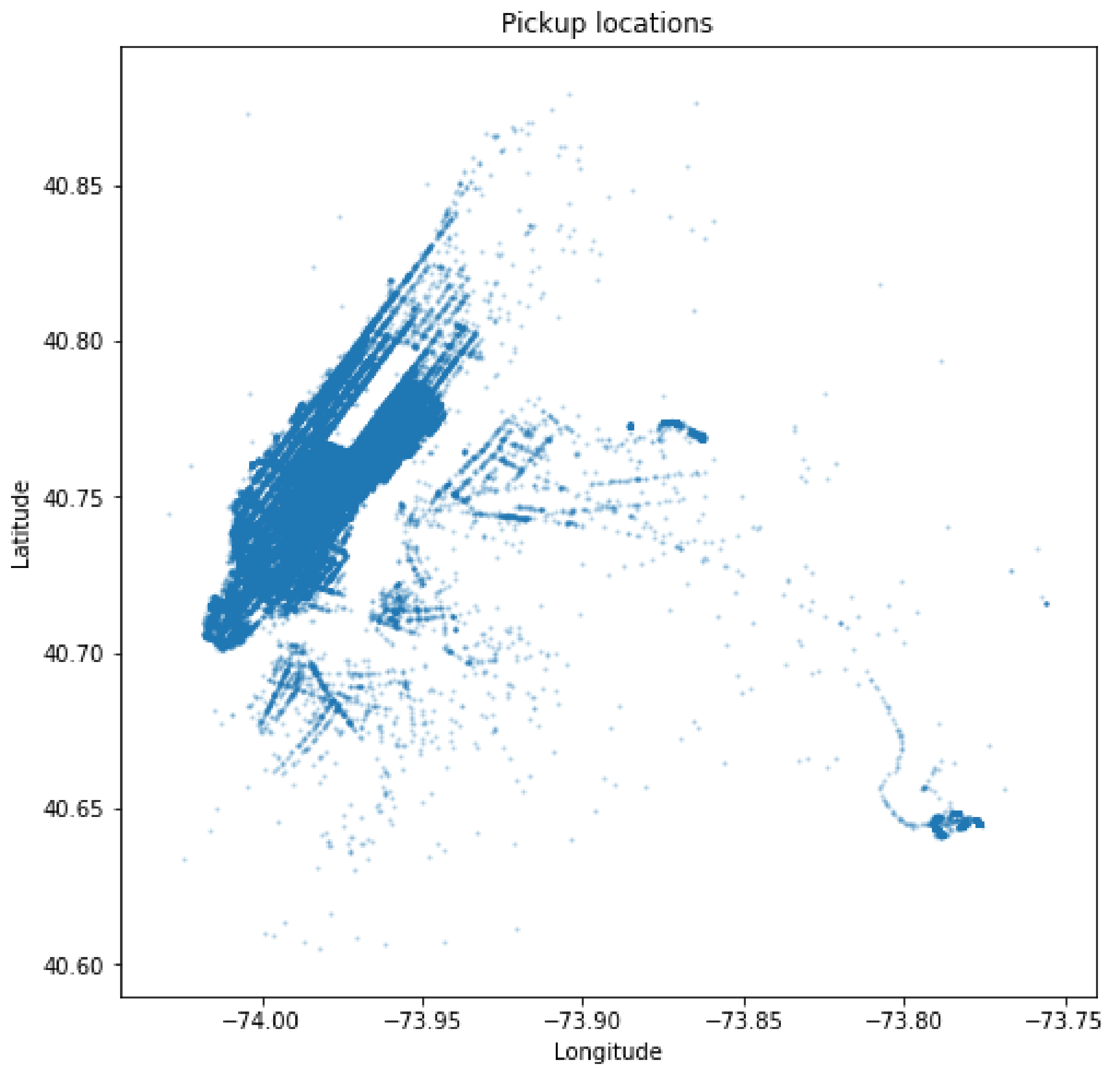
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

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

[6]:
```
lon_bounds[0]
```

[6]: -74.03

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.

[7]:
```
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.

[8]:
```
clean_taxi = all_taxi[(all_taxi["passengers"] > 0) & (all_taxi["duration"] >=
↪60) & (all_taxi["duration"] <= 3600) &
```

```
                        (all_taxi["distance"]/(all_taxi["duration"]/3600) <= 100)␣
↪& (all_taxi["distance"] > 0)]
```

[9]:
```
ok.grade("q1b");
```

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

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

[10]:
```
all_taxi.head()
len(clean_taxi)
```

[10]: 96445

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.

[11]:
```
def pointInPolygon(x, y, polyCorners, polyX, polyY):
    j = polyCorners-1
    inPoly = False

    for i in range(polyCorners):
        if (polyY[i]<y and polyY[j]>=y or polyY[j]<y and polyY[i]>=y):
            if (polyX[i] + (y - polyY[i]) / (polyY[j] - polyY[i]) * (polyX[j] -␣
↪polyX[i]) < x):
                inPoly = not inPoly
```

```
        j=i;

    return inPoly

#y is latitude
#polyX[i] + ((y - polyY[i]) / (polyY[j] - polyY[i])) * (polyX[j] - polyX[i])
```

[12]:
```
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."""
    polyCorners = len(polygon)
    return pointInPolygon(x, y, polyCorners, polygon["lon"].values,␣
 ↪polygon["lat"].values)

def in_manhattan_trip(row):
    pickupIn = in_manhattan(row["pickup_lon"], row["pickup_lat"])
    dropoffIn = in_manhattan(row["dropoff_lon"], row["dropoff_lat"])
    return pickupIn and dropoffIn

#create another fnc and return if in_manhattan(row) true for both pickup and␣
 ↪dropoff

# Recommended: Then, apply this function to every trip to filter clean_taxi.
manhattan_taxi = clean_taxi[clean_taxi.apply(lambda row:␣
 ↪in_manhattan_trip(row), axis=1)]
#Lambda x: in Manhattan(x[pickup lat], x[pickup lon])
```

[13]:
```
ok.grade("q1c");
```

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

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

[14]:
```
#type(polygon)
#len(polygon)
#polygon["lat"].values
#clean_taxi
```

```
#clean_taxi.head()
#clean_taxi.apply(lambda row: in_manhattan_trip(row), axis=1)
```

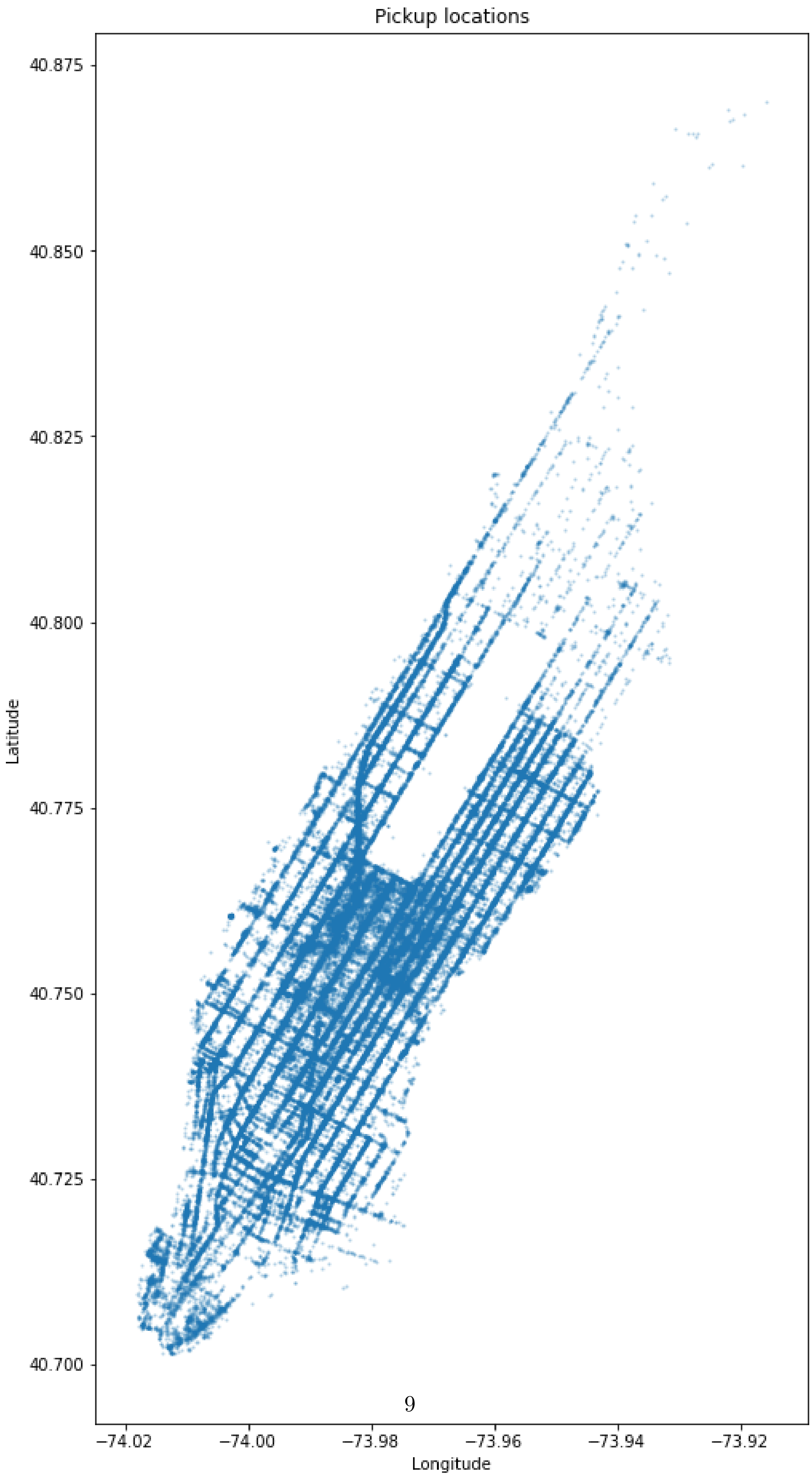
If you are unable to solve the problem above, have trouble with the tests, or want to work on the rest of the project before solving it, run the following cell to load the cleaned Manhattan data directly. (Note that you may not solve the previous problem just by loading this data file; you have to actually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

[15]:
```
manhattan_taxi = pd.read_csv('manhattan_taxi.csv')
len(manhattan_taxi)
```

[15]: 82800

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

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



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

1.5.4 Question 1d

Print a summary of the data selection and cleaning you performed. Your Python code should not include any number literals, but instead should refer to the shape of `all_taxi`, `clean_taxi`, and `manhattan_taxi`.

E.g., 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 the example above are not accurate.)

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

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.

[17]:
```
aT = len(all_taxi)
cT = len(clean_taxi)
mT = len(manhattan_taxi)

print("Of the original %s trips, %s anomolous trips were removed through data␣
↪cleaning, and then the %s trips within " %(aT, aT-cT, mT) +
     "Manhattan were selected for further analysis.")

#'%s and %s' %(a,b)
#print(aT, cT, mT)
```

```
Of the original 97692 trips, 1247 anomolous trips were removed through data
cleaning, and 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.*

[18]:
```
from datetime import datetime

manhattan_taxi["date"] = pd.to_datetime(manhattan_taxi["pickup_datetime"]).
 ↪apply(datetime.date)
#manhattan_taxi.head()
```

[19]:
```
ok.grade("q2a");
```

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

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

[20]:
```
manhattan_taxi["date"].value_counts()

#Filtered (clean_taxi):
#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.
#22, 23, 24
```

[20]:
```
2016-01-30    3352
2016-01-22    3291
2016-01-29    3280
2016-01-15    3139
2016-01-21    3133
2016-01-28    3083
2016-01-13    3066
2016-01-16    3059
2016-01-09    3058
2016-01-08    3010
2016-01-14    2992
2016-01-19    2963
2016-01-07    2908
2016-01-12    2829
```

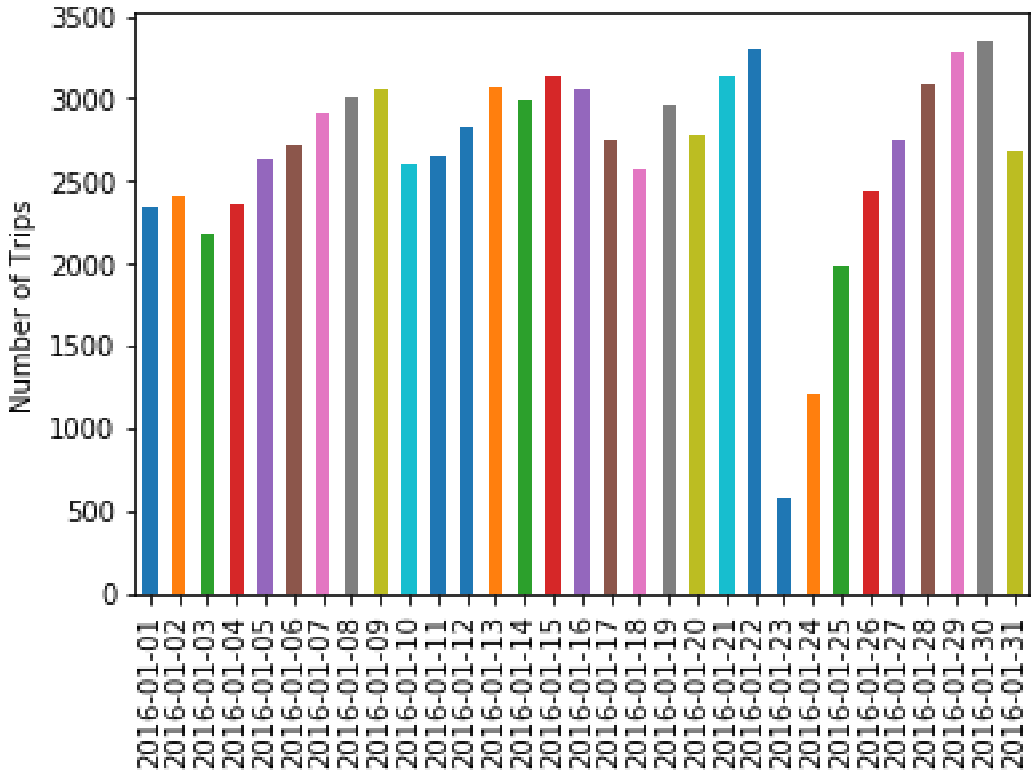
```
2016-01-20    2776
2016-01-17    2753
2016-01-27    2750
2016-01-06    2721
2016-01-31    2690
2016-01-11    2645
2016-01-05    2630
2016-01-10    2605
2016-01-18    2566
2016-01-26    2445
2016-01-02    2411
2016-01-04    2368
2016-01-01    2337
2016-01-03    2177
2016-01-25    1982
2016-01-24    1203
2016-01-23     578
Name: date, dtype: int64
```

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.

[21]:

```
import matplotlib.pyplot as plt
sorted_taxi = (manhattan_taxi["date"].value_counts()).sort_index()
sorted_taxi.plot.bar()
plt.ylabel("Number of Trips");

#The blizzard dates are listed as Jan 22-24. It makes sense that there are less␣
↪taxi rides during a blizzard
#because people wouldn't be really be going outside. Notice how the 23rd and␣
↪24th have the min number of rides.
#The 22nd has a high number of trips, perhaps because people were rushing to␣
↪get home?
```



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.)

[22]:
```
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 the project is already pretty long.

[23]:
```
# 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.

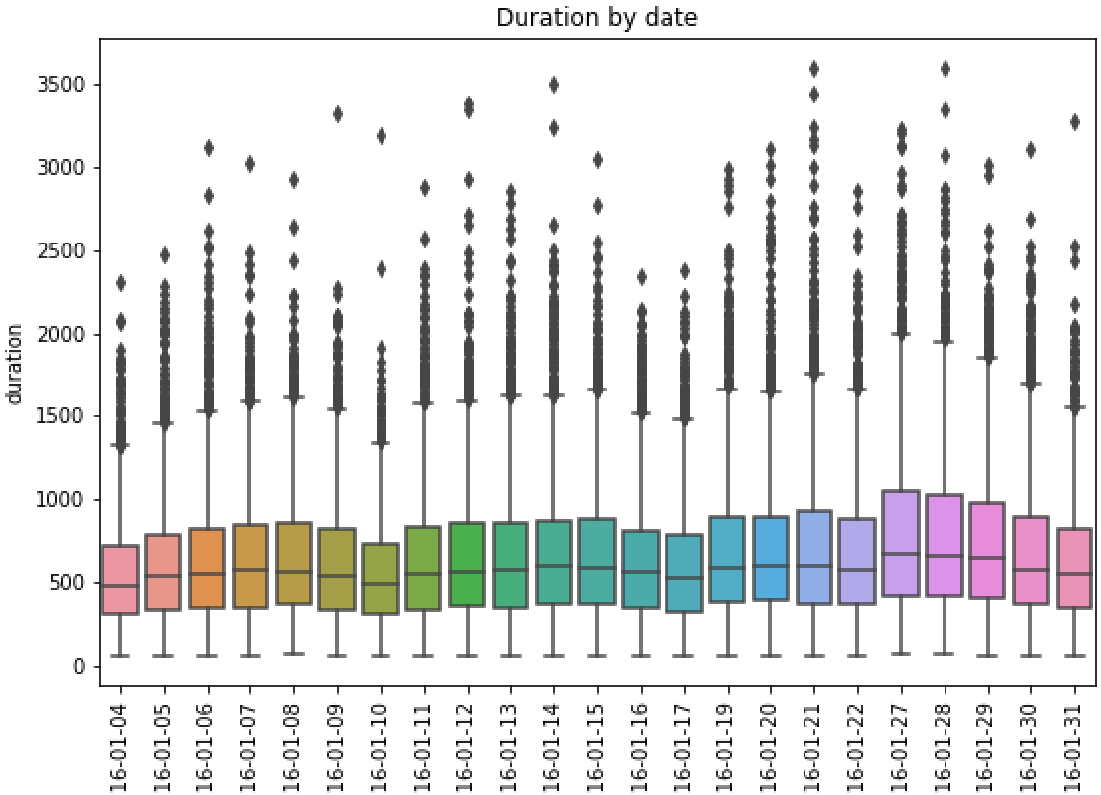
Because you are going to look at the data in detail in order to define features, it's best to split the data into training and test sets now, then only inspect the training set.

[24]:
```
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)
```

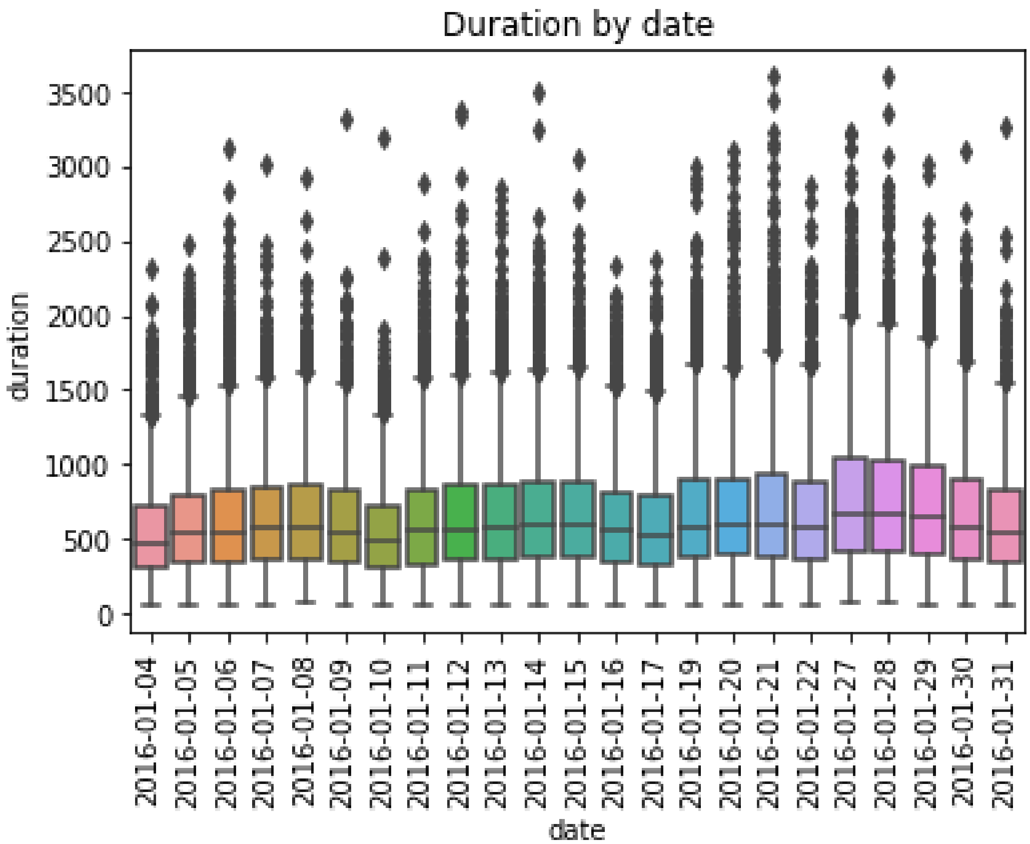
1.7.1 Question 3a

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 this:



[25]:
```python
bp = sns.boxplot(x="date", y="duration", data=train.sort_values("date"));
bp.set_xticklabels(bp.get_xticklabels(),rotation=90);
plt.title("Duration by date")
```

[25]: Text(0.5, 1.0, 'Duration by date')



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.

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

The weekdays consistently have higher medians than the days of the weekend. Thus, we can expect for durations of taxi trips during the weekdays to be longer.

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.

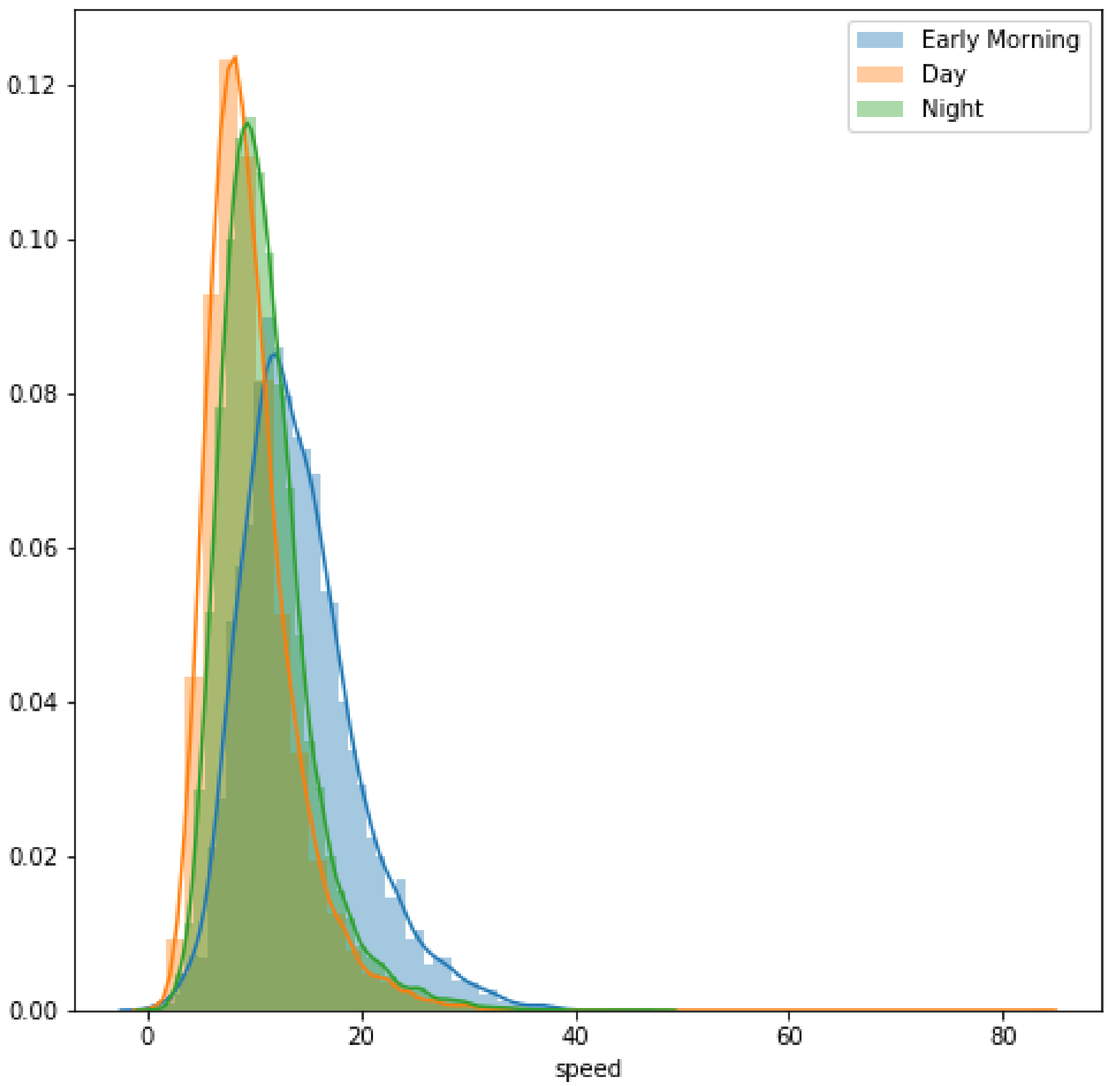
No changes are required; just run this cell.

[26]:
```python
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
```

[26]:
```
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
hour                                 18
day                                   3
weekend                               0
period                                3
speed                           6.50065
Name: 14043, dtype: object
```

1.7.3 Question 3c

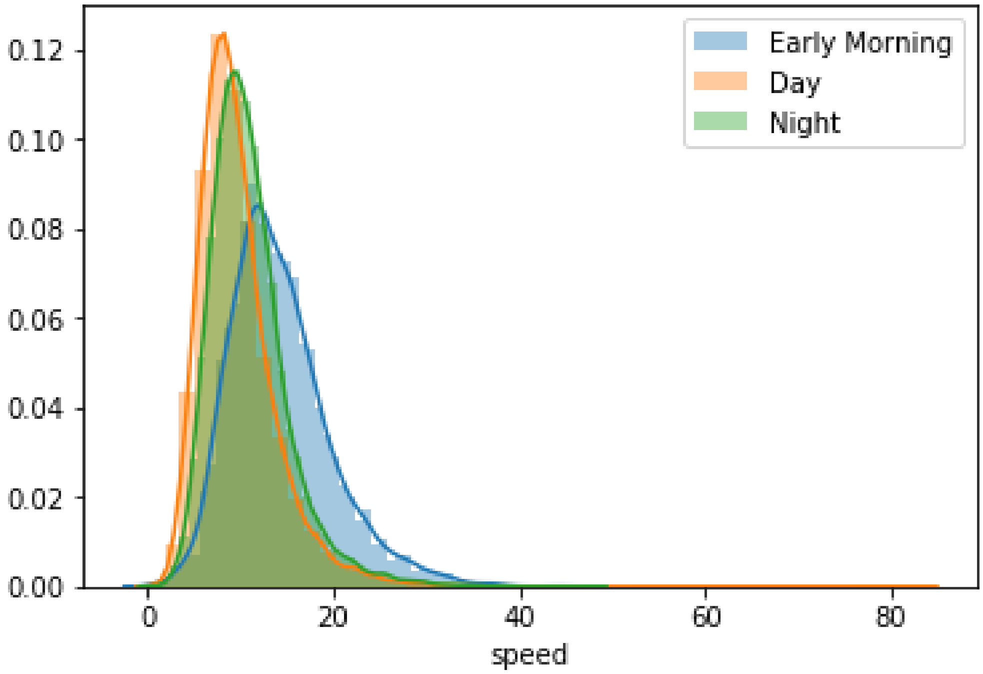
Use `sns.distplot` to create an overlaid histogram comparing the distribution of average speeds for taxi rides that start in the early morning (12am-6am), day (6am-6pm; 12 hours), and night (6pm-12am; 6 hours). Your plot should look like this:



[27]:
```
#split into 3 dfs
#plot each df["speed"]
#group = train.sort_values(by=['period'])
df1 = train[train["period"] == 1]
df2 = train[train["period"] == 2]
df3 = train[train["period"] == 3]

sns.distplot(df1["speed"], label = 'Early Morning')
sns.distplot(df2["speed"], label = 'Day')
sns.distplot(df3["speed"], label = 'Night')
plt.legend()
plt.show()
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
```

```
be interpreted as an array index, `arr[np.array(seq)]`, which will result either
in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



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).

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.

[28]:
```
# Find the first principle component
D = train[["pickup_lon", "pickup_lat"]] #...
```

```
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"]] #...
    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)
```

[29]:
```
ok.grade("q3d");
```

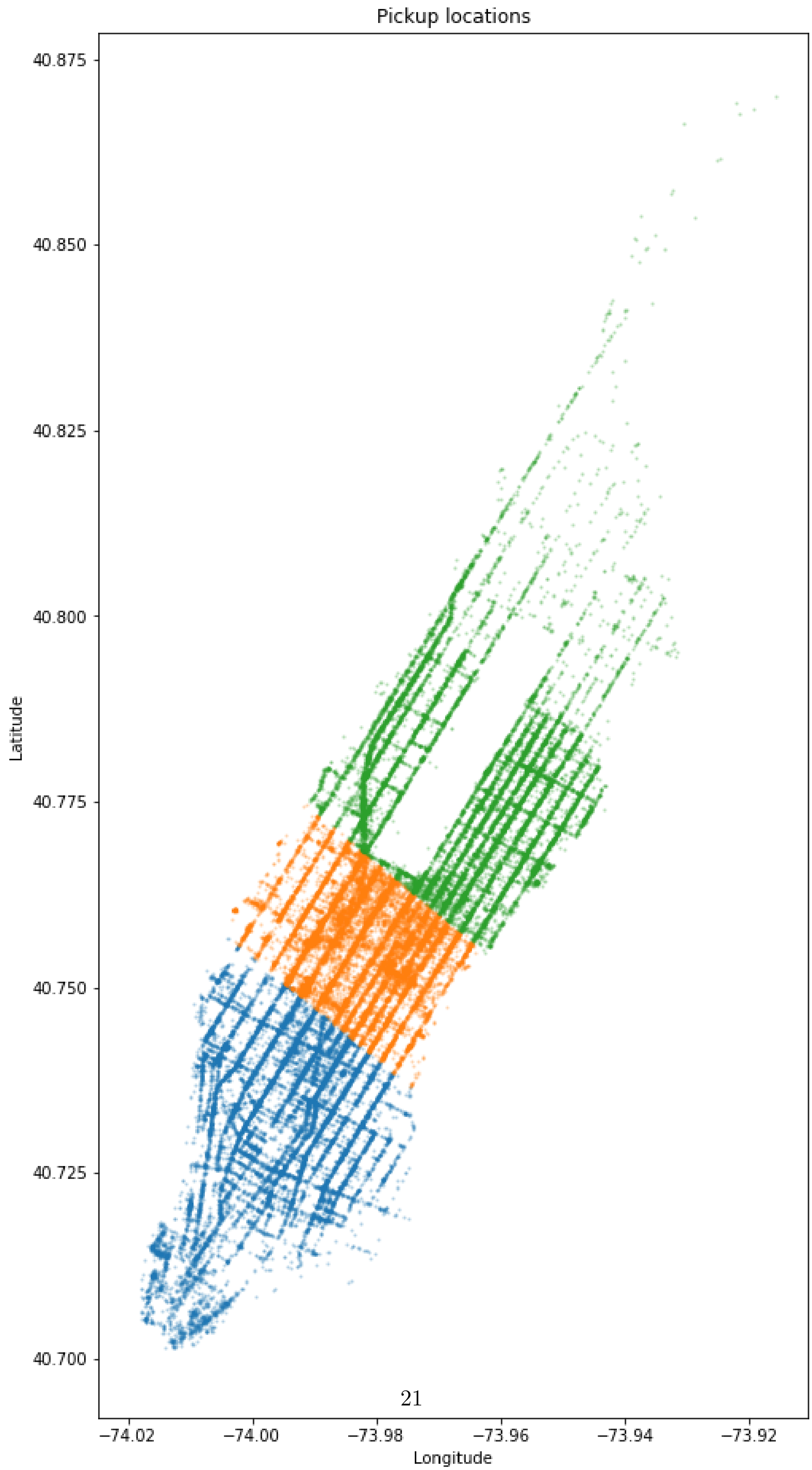
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

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

[30]:
```
train.shape[0]
train["pickup_lon"].shape
```

[30]:
```
(53680,)
```

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!

[31]:
```
plt.figure(figsize=(8, 16))
for i in [0, 1, 2]:
    pickup_scatter(train[train['region'] == i])
```



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

1.7.5 Questoin 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?

[32]:
```
...
```

[32]: Ellipsis

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.

[33]:
```
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,:]
```

[33]:
```
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.

[34]:
```
def rmse(errors):
    """Return the root mean squared error."""
    return np.sqrt(np.mean(errors ** 2))
```

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

[34]: 399.1437572352666

[35]:
```
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 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.

[75]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(train[["distance"]], train["duration"]) #model creates a line of best␣
↪fit (an equation)
predictions = model.predict(test[["distance"]]) #this outputs the predictions␣
↪based on test's distance
#you put in the test's distance values and predict
simple_rmse = rmse(predictions - test["duration"])
simple_rmse
```

[75]: 276.7841105000342

[76]:
```
ok.grade("q4b");
```

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

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

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

[ ]:

```

```

[77]:

```
train.head()
test.head()
```

[77]:

```
             pickup_datetime     dropoff_datetime  pickup_lon  pickup_lat  \
70160  2016-01-06 12:31:00  2016-01-06 12:54:00  -73.969139   40.763515
13029  2016-01-27 16:54:27  2016-01-27 17:13:15  -73.990288   40.771641
79736  2016-01-22 16:39:48  2016-01-22 16:46:00  -73.937447   40.797520
74616  2016-01-30 08:44:02  2016-01-30 08:53:48  -73.976097   40.719196
7187   2016-01-17 19:49:01  2016-01-17 20:06:14  -73.991562   40.750031

       dropoff_lon  dropoff_lat  passengers  distance  duration        date  \
70160   -73.969139    40.763515           3      3.04      1380  2016-01-06
13029   -73.978989    40.752441           1      2.00      1128  2016-01-27
79736   -73.948822    40.801723           1      0.60       372  2016-01-22
74616   -73.996437    40.725433           1      1.10       586  2016-01-30
7187    -73.951561    40.766403           1      3.50      1033  2016-01-17

       hour  day  weekend  period      speed region
70160    12    2        0       2   7.930435      2
13029    16    2        0       2   6.382979      1
79736    16    4        0       2   5.806452      2
74616     8    5        1       2   6.757679      0
7187     19    6        1       3  12.197483      1
```

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.

[78]:

```
model2 = LinearRegression()
model2.fit(design_matrix(train), train["duration"])
predictions2 = model2.predict(design_matrix(test))
linear_rmse = rmse(predictions2 - test["duration"])
linear_rmse
```

```python
#model = LinearRegression()
#model.fit(train[["distance"]], train["duration"]) #model creates a line of
 ↪best fit
#predictions = model.predict(test[["distance"]]) #this outputs the predictions
 ↪based on test's distance

#simple_rmse = rmse(predictions - test["duration"])
#simple_rmse
```

[78]: 255.19146631882757

[79]:
```python
ok.grade("q4c");
```

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

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

[80]:
```python
design_matrix(train);
#predictions
```

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.

[81]:
```python
model3 = LinearRegression()
errors = []

for v in np.unique(train['period']):
    tr = train[train["period"] == v]
    te = test[test["period"] == v]

    model3.fit(design_matrix(tr), tr["duration"])
    predictions3 = model3.predict(design_matrix(te))
    error = predictions3 - te["duration"]
    errors.extend(error)
```

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

#period == v. train[train["period"] == v] and same for test
#fit a model
#predict the values
#add to errors

#model2 = LinearRegression()
#model2.fit(design_matrix(train), train["duration"])
#predictions2 = model2.predict(design_matrix(test))
#linear_rmse = rmse(predictions2 - test["duration"])
#linear_rmse
```

[81]: 246.62868831165173

[82]:
```python
ok.grade("q4d");
```

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

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

[83]:
```python
train[train["period"] == 1]
```

[83]:
```
            pickup_datetime     dropoff_datetime  pickup_lon  pickup_lat  \
66957  2016-01-30 00:22:55  2016-01-30 00:27:29  -74.014717   40.713631
74803  2016-01-31 00:37:01  2016-01-31 00:45:31  -73.983780   40.762581
81159  2016-01-10 01:22:28  2016-01-10 01:29:17  -73.990349   40.718899
29703  2016-01-30 00:37:30  2016-01-30 00:41:19  -73.970947   40.752380
7319   2016-01-16 02:08:06  2016-01-16 02:14:02  -73.988586   40.748924
59375  2016-01-12 04:20:19  2016-01-12 04:37:05  -73.990585   40.731644
54218  2016-01-08 01:13:11  2016-01-08 01:26:42  -74.004318   40.742374
8193   2016-01-16 00:52:53  2016-01-16 00:56:13  -73.976601   40.759682
29582  2016-01-16 00:44:00  2016-01-16 00:59:36  -74.007317   40.740883
59235  2016-01-05 00:39:01  2016-01-05 00:54:00  -74.007759   40.740536
82795  2016-01-31 02:59:16  2016-01-31 03:09:23  -73.997391   40.721027
2174   2016-01-22 04:07:06  2016-01-22 04:11:01  -74.006783   40.744049
45971  2016-01-30 00:40:51  2016-01-30 00:52:23  -73.987076   40.725018
31714  2016-01-16 01:58:48  2016-01-16 02:06:17  -74.003418   40.732567
47024  2016-01-31 02:23:56  2016-01-31 02:36:20  -73.978851   40.741119
46056  2016-01-13 01:18:52  2016-01-13 01:25:01  -73.981018   40.760643
```

```
1101    2016-01-12 00:08:24  2016-01-12 00:10:15  -73.968178   40.755562
49172   2016-01-04 00:39:43  2016-01-04 00:45:12  -73.985535   40.763157
81476   2016-01-13 02:07:12  2016-01-13 02:12:41  -74.001923   40.739376
47137   2016-01-10 00:00:57  2016-01-10 00:09:18  -73.972115   40.765522
22358   2016-01-29 02:23:17  2016-01-29 02:30:07  -73.989639   40.762321
58953   2016-01-22 05:58:00  2016-01-22 06:02:05  -73.976181   40.751610
35723   2016-01-17 00:29:18  2016-01-17 00:44:27  -73.987396   40.719852
40138   2016-01-31 01:29:39  2016-01-31 01:33:19  -74.005310   40.719513
69740   2016-01-14 05:57:11  2016-01-14 06:06:07  -73.981781   40.779461
43341   2016-01-09 01:25:00  2016-01-09 01:27:28  -73.949104   40.777252
44336   2016-01-10 02:08:59  2016-01-10 02:11:24  -73.987457   40.732922
61835   2016-01-19 00:17:37  2016-01-19 00:24:34  -74.002052   40.724609
37135   2016-01-17 01:05:06  2016-01-17 01:12:45  -73.984154   40.760815
62122   2016-01-16 01:34:01  2016-01-16 01:44:25  -73.956367   40.771511
…                      …                    …          …          …
40307   2016-01-30 04:24:28  2016-01-30 04:31:54  -73.983543   40.738152
58223   2016-01-27 00:34:20  2016-01-27 00:39:15  -73.998665   40.730656
53637   2016-01-09 02:12:46  2016-01-09 02:26:50  -73.987160   40.720539
21580   2016-01-30 02:19:17  2016-01-30 02:26:48  -73.987000   40.720863
56995   2016-01-10 02:04:27  2016-01-10 02:19:21  -73.984238   40.725128
2717    2016-01-14 05:29:31  2016-01-14 05:36:02  -73.957512   40.769932
3143    2016-01-27 00:44:37  2016-01-27 00:55:49  -73.968376   40.799683
75566   2016-01-29 00:42:38  2016-01-29 00:44:55  -73.976723   40.762516
40396   2016-01-16 00:08:46  2016-01-16 00:20:37  -73.997490   40.721310
39076   2016-01-16 00:29:26  2016-01-16 00:44:23  -73.991455   40.735180
57790   2016-01-22 00:09:32  2016-01-22 00:15:18  -73.951729   40.790428
27152   2016-01-09 01:08:20  2016-01-09 01:13:57  -73.982056   40.763912
2236    2016-01-21 03:45:31  2016-01-21 03:52:37  -74.002495   40.750111
3598    2016-01-17 01:12:30  2016-01-17 01:18:09  -73.959320   40.763424
11660   2016-01-16 05:08:12  2016-01-16 05:17:08  -73.949631   40.796356
60563   2016-01-15 03:55:01  2016-01-15 04:07:36  -73.989799   40.725742
77241   2016-01-11 05:32:14  2016-01-11 05:38:13  -73.985580   40.731682
8508    2016-01-10 00:22:25  2016-01-10 00:45:34  -73.990265   40.754257
33706   2016-01-13 01:52:57  2016-01-13 01:58:03  -73.970993   40.761288
33188   2016-01-22 05:13:01  2016-01-22 05:15:39  -73.960114   40.774136
76290   2016-01-17 04:58:13  2016-01-17 05:09:20  -73.999184   40.728142
78599   2016-01-31 02:48:26  2016-01-31 03:04:21  -73.987602   40.732635
28800   2016-01-11 02:45:55  2016-01-11 02:49:26  -73.977058   40.754578
65400   2016-01-10 00:47:28  2016-01-10 00:58:30  -73.965378   40.759220
44203   2016-01-06 04:27:27  2016-01-06 04:34:53  -73.990395   40.731434
72968   2016-01-16 02:14:14  2016-01-16 02:38:53  -73.987411   40.721142
31734   2016-01-15 05:00:38  2016-01-15 05:07:05  -73.961464   40.764427
6550    2016-01-31 01:30:53  2016-01-31 01:43:41  -73.980026   40.743118
79967   2016-01-17 00:32:58  2016-01-17 00:39:18  -73.954033   40.787281
1076    2016-01-04 05:46:00  2016-01-04 05:52:29  -73.984055   40.725250

        dropoff_lon  dropoff_lat  passengers  distance  duration        date  \
```

66957	-74.009247	40.713051		2	0.43	274	2016-01-30
74803	-73.969017	40.755753		4	1.00	510	2016-01-31
81159	-73.982758	40.731277		1	0.94	409	2016-01-10
29703	-73.972351	40.761211		5	0.76	229	2016-01-30
7319	-73.985931	40.768112		2	1.60	356	2016-01-16
59375	-73.968369	40.787445		2	4.75	1006	2016-01-12
54218	-73.987701	40.750294		5	1.81	811	2016-01-08
8193	-73.977676	40.753201		3	0.56	200	2016-01-16
29582	-73.956520	40.766895		1	4.56	936	2016-01-16
59235	-73.976158	40.775970		2	3.20	899	2016-01-05
82795	-73.978447	40.745277		1	2.17	607	2016-01-31
2174	-73.992287	40.743679		1	0.93	235	2016-01-22
45971	-73.963860	40.757046		1	2.60	692	2016-01-30
31714	-74.004166	40.720901		2	0.87	449	2016-01-16
47024	-74.001404	40.729317		1	2.30	744	2016-01-31
46056	-73.995811	40.753120		3	1.37	369	2016-01-13
1101	-73.975014	40.746189		6	0.73	111	2016-01-12
49172	-73.973434	40.784492		2	1.78	329	2016-01-04
81476	-73.991234	40.747185		1	1.10	329	2016-01-13
47137	-73.955696	40.770920		1	1.50	501	2016-01-10
22358	-73.992958	40.740856		1	2.00	410	2016-01-29
58953	-73.972038	40.763592		1	0.93	245	2016-01-22
35723	-73.977783	40.752323		1	2.80	909	2016-01-17
40138	-73.999573	40.718597		1	0.40	220	2016-01-31
69740	-73.977409	40.755241		6	2.06	536	2016-01-14
43341	-73.943100	40.786781		1	0.78	148	2016-01-09
44336	-73.981201	40.737156		1	0.50	145	2016-01-10
61835	-73.986122	40.730831		2	1.31	417	2016-01-19
37135	-73.980385	40.780422		1	1.90	459	2016-01-17
62122	-73.980019	40.742458		1	2.45	624	2016-01-16
…	…	…	…	…	…	…	
40307	-73.963448	40.768181		1	2.45	446	2016-01-30
58223	-73.984200	40.724033		1	0.80	295	2016-01-27
53637	-73.986809	40.750061		2	2.74	844	2016-01-09
21580	-73.990547	40.714512		3	0.90	451	2016-01-30
56995	-73.946892	40.781708		2	4.60	894	2016-01-10
2717	-73.981499	40.760399		1	1.95	391	2016-01-14
3143	-73.990959	40.761093		1	4.10	672	2016-01-27
75566	-73.982162	40.766907		1	0.35	137	2016-01-29
40396	-74.005150	40.741802		1	2.00	711	2016-01-16
39076	-73.971451	40.762814		1	2.79	897	2016-01-16
57790	-73.938408	40.817348		1	2.25	346	2016-01-22
27152	-73.991051	40.750496		1	1.00	337	2016-01-09
2236	-73.987801	40.749222		1	1.40	426	2016-01-21
3598	-73.948067	40.784554		1	1.70	339	2016-01-17
11660	-73.993439	40.736046		1	4.81	536	2016-01-16
60563	-73.986092	40.761463		1	3.30	755	2016-01-15

77241	-73.999290	40.713772	2	1.46	359	2016-01-11
8508	-73.983902	40.715111	1	4.03	1389	2016-01-10
33706	-73.970261	40.752312	1	0.80	306	2016-01-13
33188	-73.952026	40.771080	1	0.50	158	2016-01-22
76290	-73.986214	40.761490	1	2.87	667	2016-01-17
78599	-74.016365	40.705254	1	3.40	955	2016-01-31
28800	-73.991432	40.749794	1	1.15	211	2016-01-11
65400	-73.992188	40.749458	1	2.18	662	2016-01-10
44203	-73.979012	40.758553	1	2.20	446	2016-01-06
72968	-73.953598	40.788139	1	6.90	1479	2016-01-16
31734	-73.976463	40.761257	2	1.24	387	2016-01-15
6550	-73.994949	40.721371	2	2.20	768	2016-01-31
79967	-73.974098	40.794121	2	1.30	380	2016-01-17
1076	-74.001221	40.731049	1	1.00	389	2016-01-04

	hour	day	weekend	period	speed	region
66957	0	5	1	1	5.649635	0
74803	0	6	1	1	7.058824	1
81159	1	6	1	1	8.273839	0
29703	0	5	1	1	11.947598	1
7319	2	5	1	1	16.179775	1
59375	4	1	0	1	16.998012	0
54218	1	4	0	1	8.034525	0
8193	0	5	1	1	10.080000	1
29582	0	5	1	1	17.538462	0
59235	0	1	0	1	12.814238	0
82795	2	6	1	1	12.869852	0
2174	4	4	0	1	14.246809	0
45971	0	5	1	1	13.526012	0
31714	1	5	1	1	6.975501	0
47024	2	6	1	1	11.129032	1
46056	1	2	0	1	13.365854	1
1101	0	1	0	1	23.675676	1
49172	0	0	0	1	19.477204	1
81476	2	2	0	1	12.036474	0
47137	0	6	1	1	10.778443	2
22358	2	4	0	1	17.560976	1
58953	5	4	0	1	13.665306	1
35723	0	6	1	1	11.089109	0
40138	1	6	1	1	6.545455	0
69740	5	3	0	1	13.835821	2
43341	1	5	1	1	18.972973	2
44336	2	6	1	1	12.413793	0
61835	0	1	0	1	11.309353	0
37135	1	6	1	1	14.901961	1
62122	1	5	1	1	14.134615	2
…	…	…	…	…	…	…

```
40307     4   5        1        1  19.775785      0
58223     0   2        0        1   9.762712      0
53637     2   5        1        1  11.687204      0
21580     2   5        1        1   7.184035      0
56995     2   6        1        1  18.523490      0
2717      5   3        0        1  17.953964      2
3143      0   2        0        1  21.964286      2
75566     0   4        0        1   9.197080      1
40396     0   5        1        1  10.126582      0
39076     0   5        1        1  11.197324      0
57790     0   4        0        1  23.410405      2
27152     1   5        1        1  10.682493      1
2236      3   3        0        1  11.830986      0
3598      1   6        1        1  18.053097      2
11660     5   5        1        1  32.305970      2
60563     3   4        0        1  15.735099      0
77241     5   0        0        1  14.640669      0
8508      0   6        1        1  10.444924      1
33706     1   2        0        1   9.411765      2
33188     5   4        0        1  11.392405      2
76290     4   6        1        1  15.490255      0
78599     2   6        1        1  12.816754      0
28800     2   0        0        1  19.620853      1
65400     0   6        1        1  11.854985      2
44203     4   2        0        1  17.757848      0
72968     2   5        1        1  16.795132      0
31734     5   4        0        1  11.534884      2
6550      1   6        1        1  10.312500      1
79967     0   6        1        1  12.315789      2
1076      5   0        0        1   9.254499      0

[4868 rows x 16 columns]
```

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 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.

The design matrix can include features that are not linearly associated with duration, which means that the predicted values can be thrown off.

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.

[84]:
```
type(test["duration"])
```

[84]: pandas.core.series.Series

[85]:
```
model4 = LinearRegression()
model4.fit(design_matrix(train), train["speed"])
predictions4 = model4.predict(design_matrix(test)) #predicted speeds
predicted_duration = (test["distance"]/predictions4) * 3600
speed_rmse = rmse(test["duration"].values - predicted_duration)
speed_rmse

#model2 = LinearRegression()
#model2.fit(design_matrix(train), train["duration"])
#predictions2 = model2.predict(design_matrix(test))
#convert to duration (duration = speed * distance)
#linear_rmse = rmse(predictions2 - test["duration"])
#linear_rmse


#test["duration"]
#m/h * 1/m and then reciprocal to get h and then divide by 3600?
#m * h/m
```

[85]: 243.01798368514952

[86]:
```
ok.grade("q4f");
```

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

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

[87]:
```
train.head()
```

[87]:
```
            pickup_datetime     dropoff_datetime  pickup_lon  pickup_lat  \
14043  2016-01-21 18:02:20  2016-01-21 18:27:54  -73.994202   40.751019
9122   2016-01-29 06:18:36  2016-01-29 06:21:32  -73.990402   40.756344
9291   2016-01-04 20:34:21  2016-01-04 20:42:33  -74.006554   40.732922
76214  2016-01-09 12:12:58  2016-01-09 12:20:26  -73.992065   40.750313
46314  2016-01-13 10:57:45  2016-01-13 11:02:06  -73.959358   40.771824

       dropoff_lon  dropoff_lat  passengers  distance  duration        date  \
14043   -73.963692    40.771069           1      2.77      1534  2016-01-21
9122    -73.984161    40.761757           3      0.69       176  2016-01-29
9291    -74.001175    40.751366           1      1.60       492  2016-01-04
76214   -73.982803    40.755829           1      0.90       448  2016-01-09
46314   -73.964661    40.770443           1      0.40       261  2016-01-13

       hour  day  weekend  period      speed region
14043    18    3        0       3   6.500652      1
9122      6    4        0       2  14.113636      1
9291     20    0        0       3  11.707317      0
76214    12    5        1       2   7.232143      1
46314    10    2        0       2   5.517241      2
```

Optional: Explain why predicting speed leads to a more accurate regression model than predicting duration directly.

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.

[88]:
```
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"""
    #...
    convertedPredictions = (observations["distance"]/predictions) * 3600 #m * h/
↪m * 3600
    return convertedPredictions - 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: #each vs is a combination
        #print(vs)
        v_train, v_test = train, test #reset
        for v, c in zip(vs, choices): #this for loop always iterates 3 times␣
↪(look at print)
            #print(v, c)
            #filter v_trian and v_test based on v,c
            v_train = v_train[v_train[c] == v]
            v_test = v_test[v_test[c] == v]
            #...
        model.fit(design_matrix(v_train), v_train[outcome])
        predictions = model.predict(design_matrix(v_test))
        error = error_fn(predictions, v_test) #error_fn(predictions,␣
↪v_test[outcome]) not this b/c needs to be df?
        errors.extend(error)
        #...
    return errors

#use design matrix
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)

#model2 = LinearRegression()
#model2.fit(design_matrix(train), train[outcome])
#predictions2 = model2.predict(design_matrix(test))
##linear_rmse = error_fn(predictions2 - test[outcome])
##linear_rmse
```

```
Duration: 240.33952192703526
Speed: 226.90793945018308
```

[89]:
```
ok.grade("q4g");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

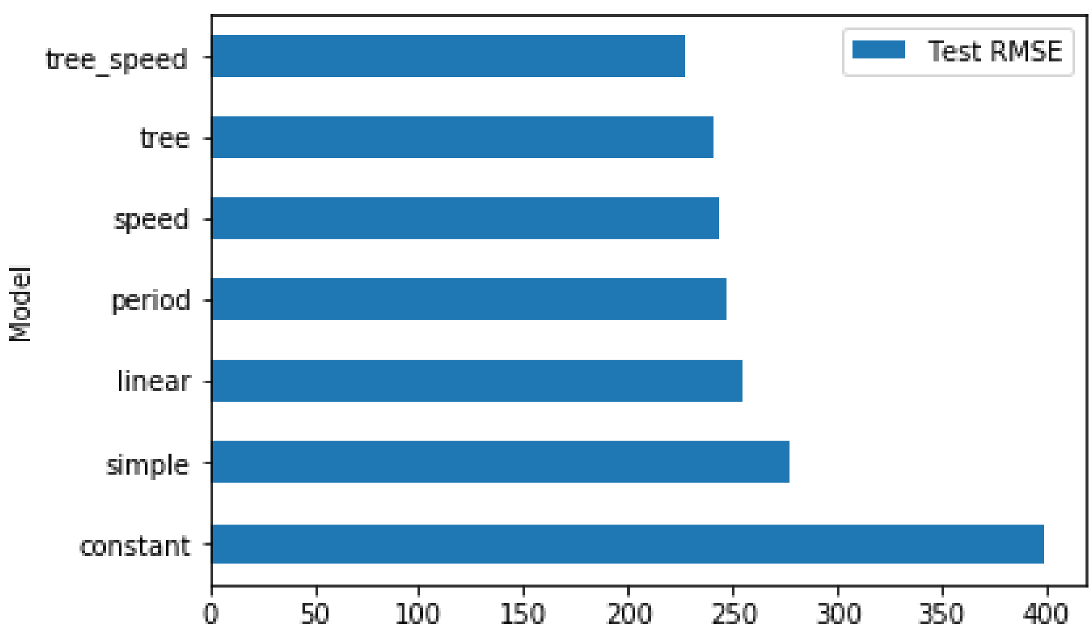
```
Running tests

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

[90]:
```
choices = ['period', 'region', 'weekend']
train.groupby(choices).size().index
```

[90]:
```
MultiIndex(levels=[[1, 2, 3], [0, 1, 2], [0, 1]],
           labels=[[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2], [0,
0, 1, 1, 2, 2, 0, 0, 1, 1, 2, 2, 0, 0, 1, 1, 2, 2], [0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1]],
           names=['period', 'region', 'weekend'])
```

Here's a summary of your results:

[91]:
```
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!**

[ ]:
```
# 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>
```

[ ]:
```

```