In [17]:

```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('proj2.ok')
```

```
=====================================================================
Assignment: proj2
OK, version v1.18.1
=====================================================================
```

Project 2: Predicting Taxi Ride Duration

Due Date: 2021.6.11 (Fri) 11:59PM

Collaboration Policy

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

Collaborators: *list collaborators here*

Score Breakdown

Question	Points
1a	1
1b	2
2a	2
2b	1
2c	2
2d	2
3a	2
3b	2
3c	2
3d	2
3e	2
3f	2
3g	4
Total	26

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:

In [18]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

The Data

Run the following cell to load the cleaned Manhattan data.

In [19]:

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

Attributes of all yellow taxi (https://en.wikipedia.org/wiki/Taxicabs_of_New_York_City) trips in January 2016 are published by the NYC Taxi and Limosine Commission (https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page).

Columns of the `manhattan_taxi` table include:

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

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

```
In [20]: manhattan_taxi.head()
```

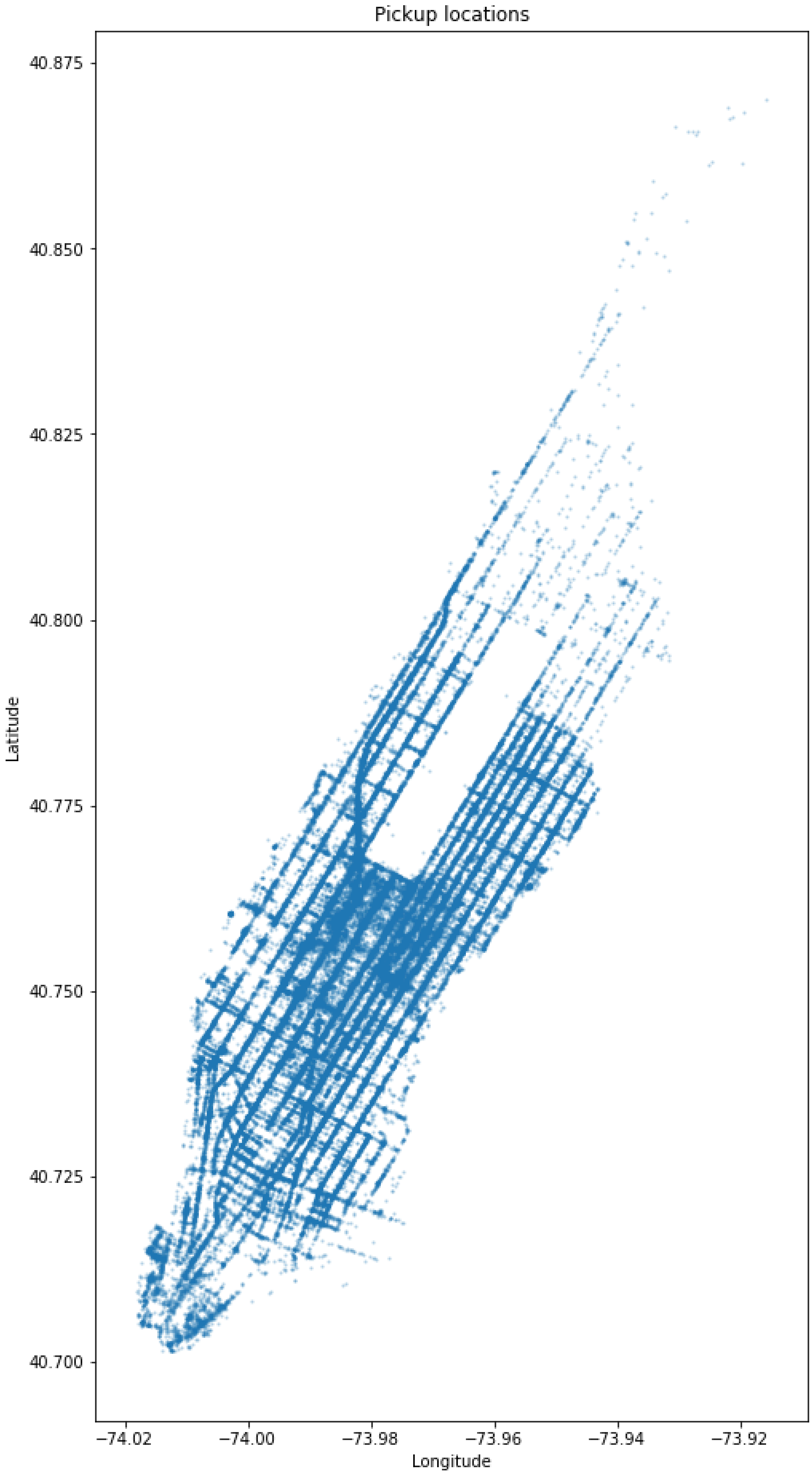
Out[20]:

	pickup_datetime	dropoff_datetime	pickup_lon	pickup_lat	dropoff_lon	dropoff_
0	2016-01-30 22:47:32	2016-01-30 23:03:53	-73.988251	40.743542	-74.015251	40.70980
1	2016-01-04 04:30:48	2016-01-04 04:36:08	-73.995888	40.760010	-73.975388	40.78220
2	2016-01-07 21:52:24	2016-01-07 21:57:23	-73.990440	40.730469	-73.985542	40.73851
3	2016-01-08 18:46:10	2016-01-08 18:54:00	-74.004494	40.706989	-74.010155	40.71675
4	2016-01-02 12:39:57	2016-01-02 12:53:29	-73.958214	40.760525	-73.983360	40.76040

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

In [21]:

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

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


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

Part 1: Exploratory Data Analysis

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

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

Note that January 2016 had some atypical days.

- New Years Day (January 1) fell on a Friday.
- Martin Luther King Jr. Day was on Monday, January 18.
- A historic blizzard (https://en.wikipedia.org/wiki/January_2016_United_States_blizzard) passed through New York that month.

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

Question 1a

Add a column labeled `date` to `manhattan_taxi` that contains the date (but not the time) of pickup, formatted as a `datetime.date` value (docs (https://docs.python.org/3/library/datetime.html#date-objects)).

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

In [22]:

```
# BEGIN YOUR CODE
# -----------------------
manhattan_taxi.loc[:, 'date'] = pd.to_datetime(manhattan_taxi['pickup_datetime']).a
pply(lambda x: x.date())
# -----------------------
# END YOUR CODE
manhattan_taxi.head()
```

Out[22]:

	pickup_datetime	dropoff_datetime	pickup_lon	pickup_lat	dropoff_lon	dropoff_
0	2016-01-30 22:47:32	2016-01-30 23:03:53	-73.988251	40.743542	-74.015251	40.70980
1	2016-01-04 04:30:48	2016-01-04 04:36:08	-73.995888	40.760010	-73.975388	40.78220
2	2016-01-07 21:52:24	2016-01-07 21:57:23	-73.990440	40.730469	-73.985542	40.73851
3	2016-01-08 18:46:10	2016-01-08 18:54:00	-74.004494	40.706989	-74.010155	40.71675
4	2016-01-02 12:39:57	2016-01-02 12:53:29	-73.958214	40.760525	-73.983360	40.76040

In [23]:

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

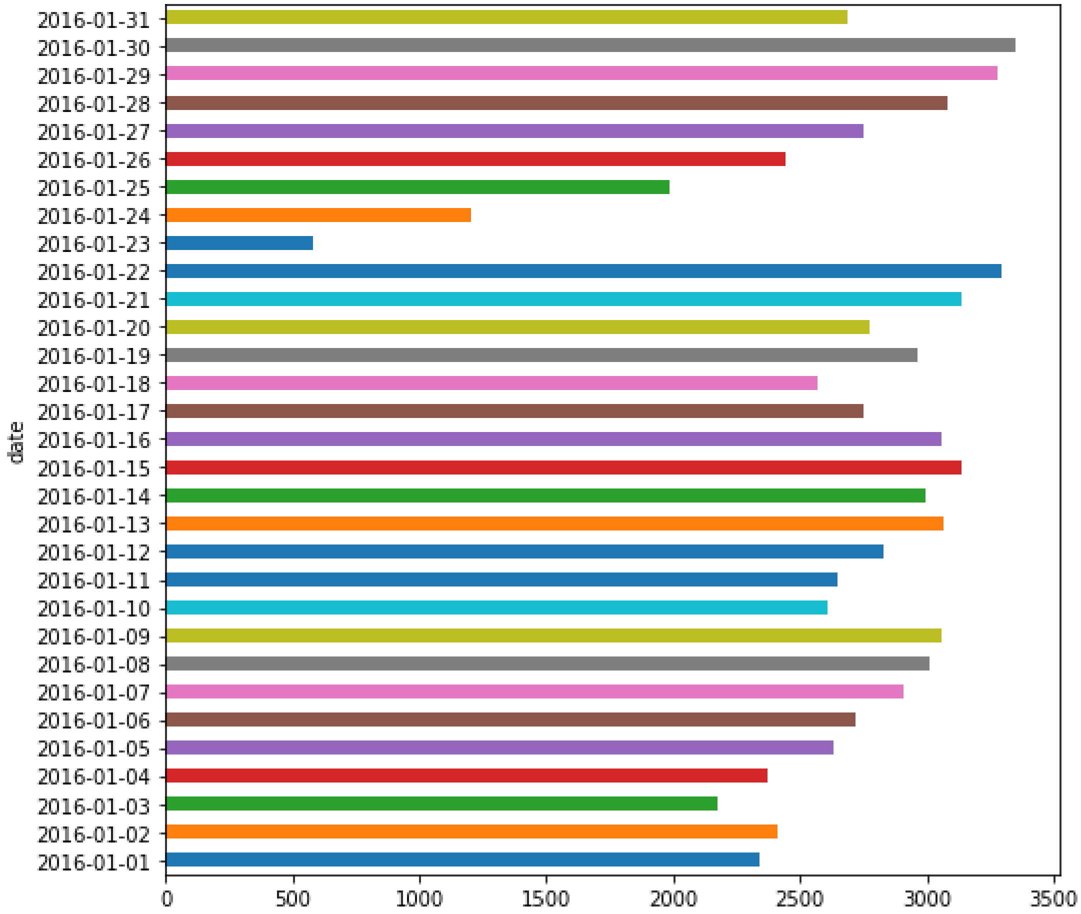
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

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

Question 1b

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

In [31]:

```
# BEGIN YOUR CODE
# -----------------------
manhattan_taxi.groupby('date').size().plot(kind='barh', figsize=(8, 8));
# -----------------------
# END YOUR CODE
```



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.)

In [32]:

```
import calendar
import re

from datetime import date

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

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

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

```
Typical dates:

    January 2016
Mo Tu We Th Fr Sa Su

 4  5  6  7  8  9 10
11 12 13 14 15 16 17
   19 20 21 22
      27 28 29 30 31
```

Part 2: Feature Engineering

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

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

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

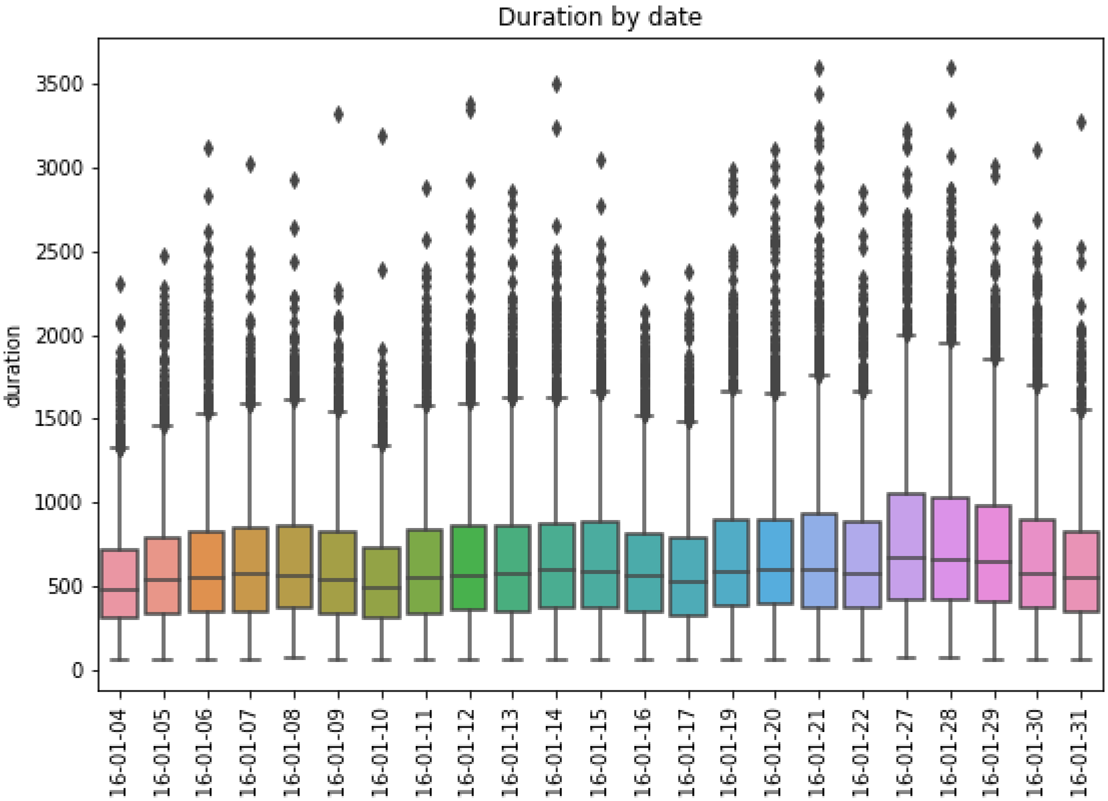
In [33]:

```
import sklearn.model_selection

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

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

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

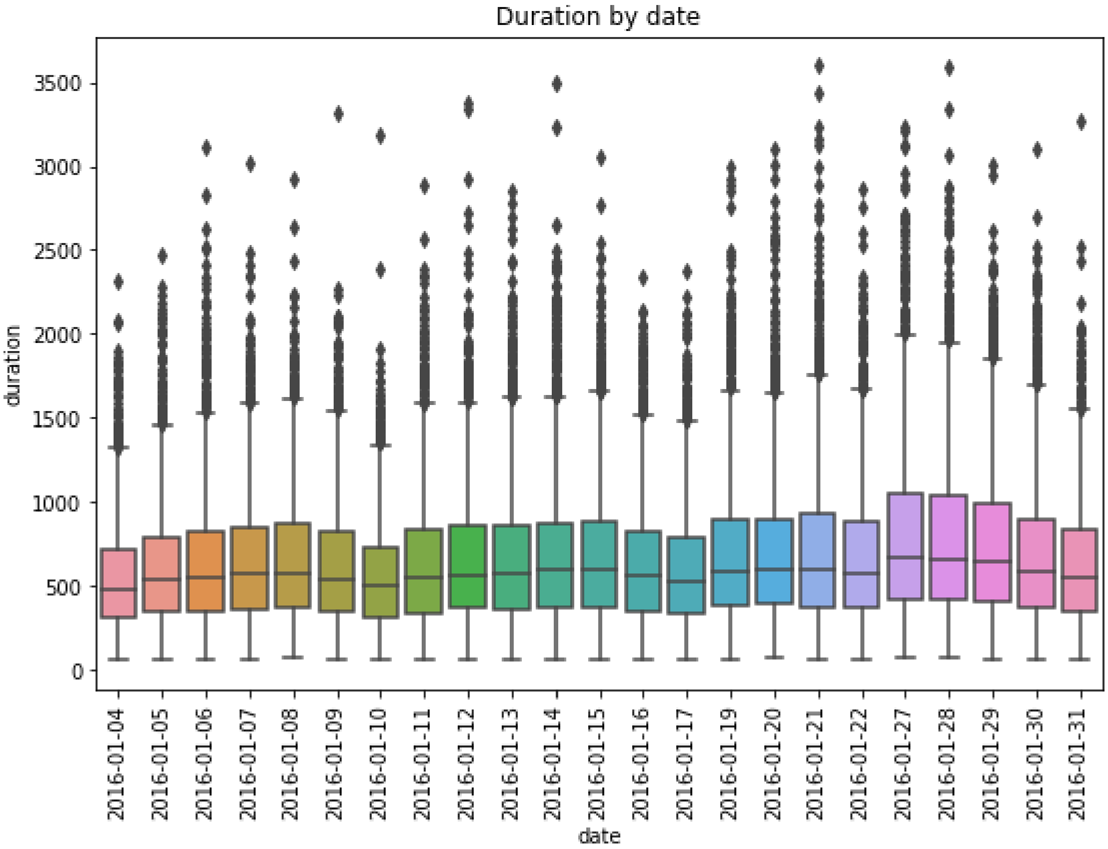
Question 2a

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



```
In [34]: plt.figure(figsize=(9, 6))
         # BEGIN YOUR CODE
         # -----------------------
         data = train.sort_values('date')
         plt.figure(figsize=(9, 6))
         sns.boxplot('date', 'duration', data=data);
         plt.xticks(rotation=90);
         plt.title('Duration by date');
         # -----------------------
         # END YOUR CODE
```

```
<matplotlib.figure.Figure at 0x25f8b198d68>
```



Question 2b

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

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

Answer: The mean of the duration of taxi trips in weekdays is usually higher than the mean of the duration o ftaxi trips in the weekends.

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

- hour: The integer hour of the pickup time. E.g., a 3:45pm taxi ride would have 15 as the hour. A 12:20am ride would have 0.
- day: The day of the week with Monday=0, Sunday=6.
- weekend: 1 if and only if the day is Saturday or Sunday.
- period: 1 for early morning (12am-6am), 2 for daytime (6am-6pm), and 3 for night (6pm-12pm).
- speed: Average speed in miles per hour.

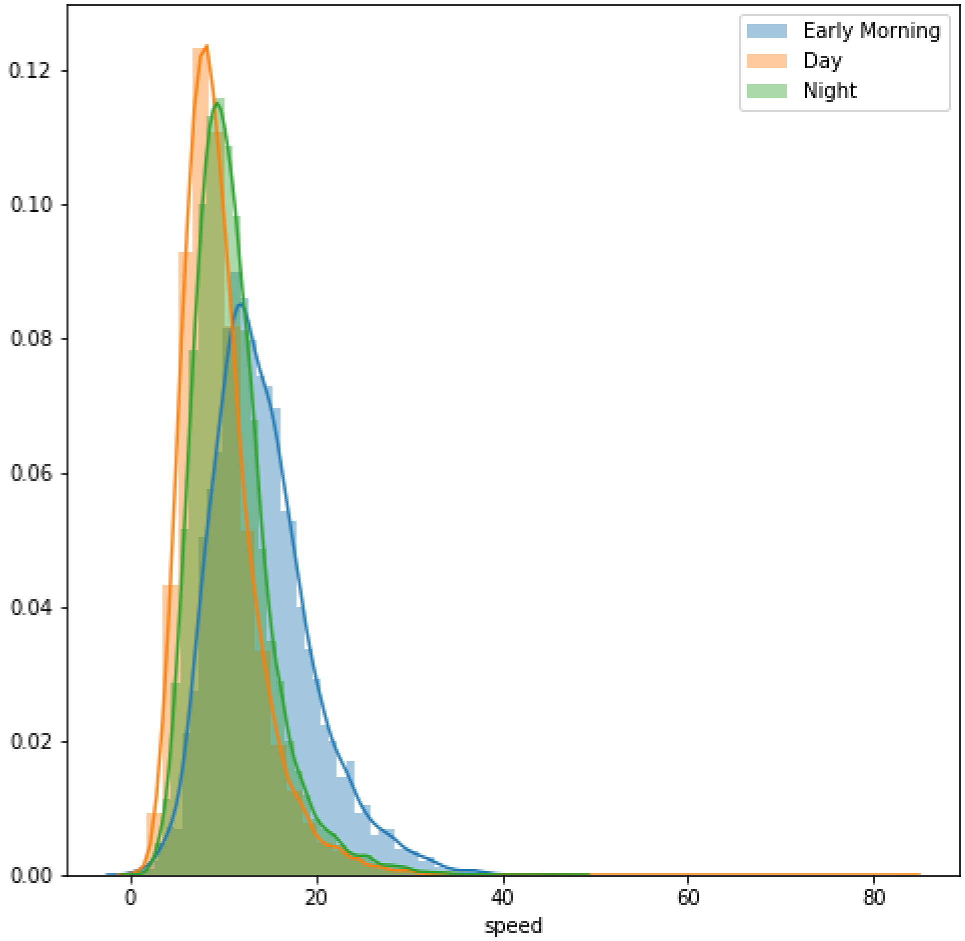
No changes are required; just run this cell.

```
In [35]: 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
```

```
Out[35]: 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
```

Question 2c

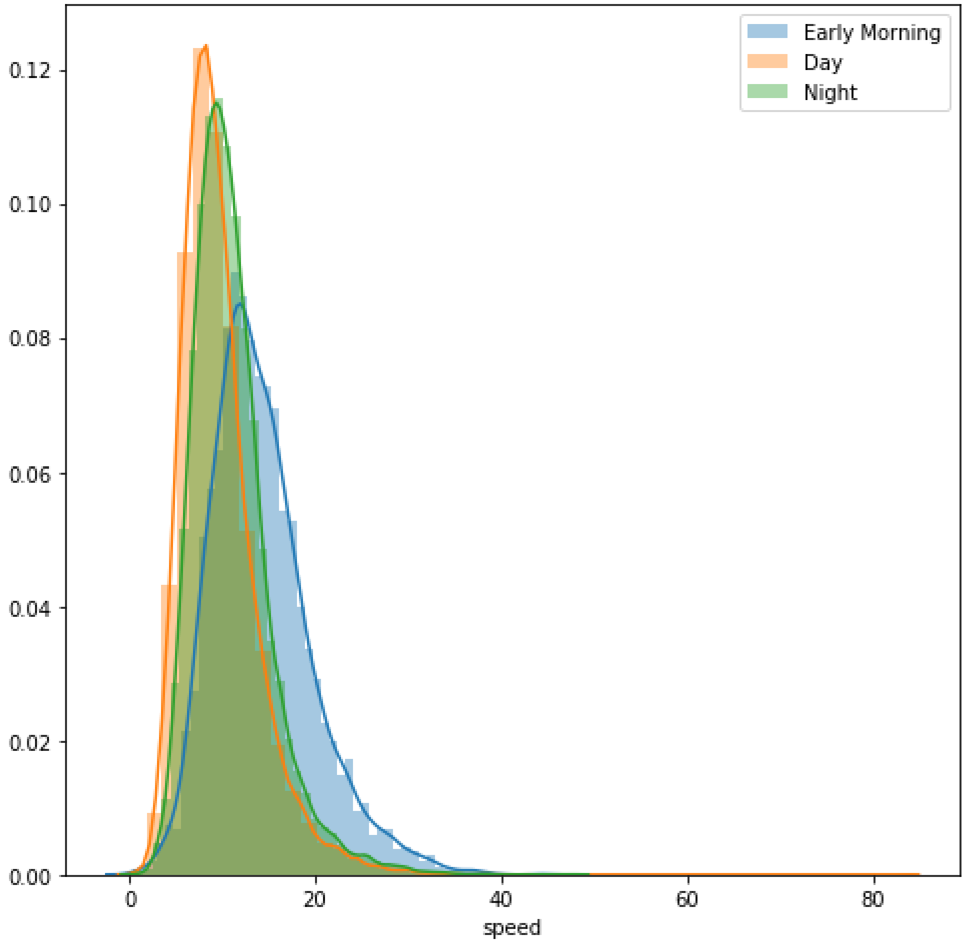
Use `sns.distplot` to create an overlaid histogram comparing the distribution of average speeds for taxi rides that start in the early morning (12am-6am), day (6am-6pm; 12 hours), and night (6pm-12am; 6 hours). Your plot should look like this:



In [36]:

```
plt.figure(figsize=(8, 8))
# BEGIN YOUR CODE
# -----------------------
plt.figure(figsize=(8, 8))
for i, s in enumerate(['Early Morning', 'Day', 'Night']):
    sns.distplot(train[train['period'] == i+1]['speed'], label=s)
plt.legend();
# -----------------------
# END YOUR CODE
```

```
<matplotlib.figure.Figure at 0x25f8a2b52e8>
```



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

Question 2d (PCA)

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

- Add a `region` column to `train` that categorizes each pick-up location as 0, 1, or 2 based on the value of each point's first principal component, such that an equal number of points fall into each region.
- Read the documentation of `pd.qcut` (https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.qcut.html), 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.

In [37]:

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

def add_region(t):
    """Add a region column to t based on vt above."""
    # BEGIN YOUR CODE
    # -----------------------
    D = t[['pickup_lon', 'pickup_lat']].values
    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] # SOLUTION
    # -----------------------
    # END YOUR CODE
    t.loc[:,'region'] = pd.qcut(first_pc, 3, labels=[0, 1, 2])

add_region(train)
add_region(test)
```

```
In [38]: ok.grade("q2d");
```

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

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

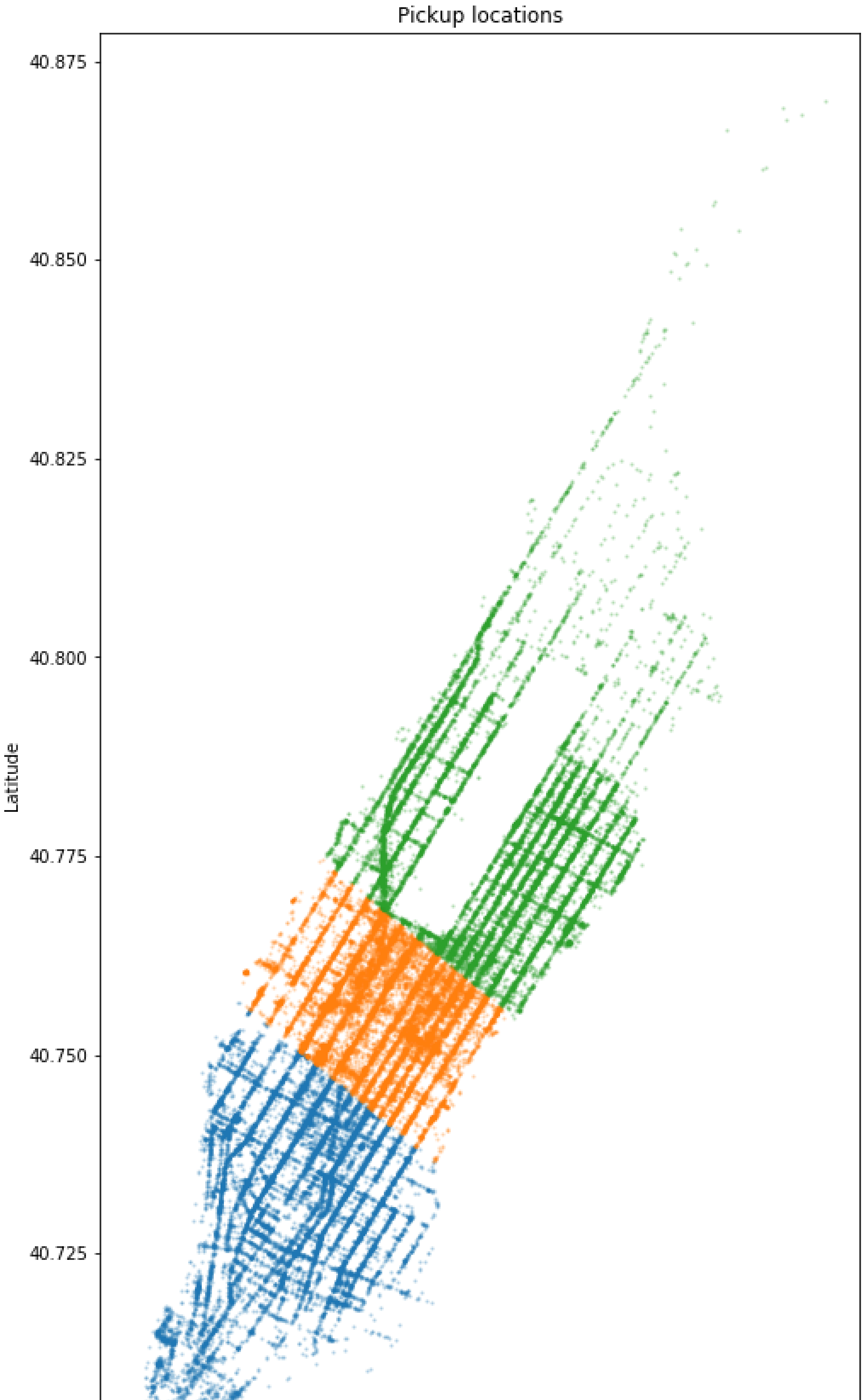
Let's see how PCA divided the trips into three groups. These regions do roughly correspond to

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

No prior knowledge of New York geography was required!

In [39]:

```
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

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

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

Note that,

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

In [40]:

```
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_v
ars]
    return pd.concat([scaled] + categoricals, axis=1)

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

Out[40]:

```
pickup_lon    -0.805821
pickup_lat    -0.171761
dropoff_lon    0.954062
dropoff_lat    0.624203
distance       0.626326
hour_1         0.000000
hour_2         0.000000
hour_3         0.000000
hour_4         0.000000
hour_5         0.000000
hour_6         0.000000
hour_7         0.000000
hour_8         0.000000
hour_9         0.000000
hour_10        0.000000
hour_11        0.000000
hour_12        0.000000
hour_13        0.000000
hour_14        0.000000
hour_15        0.000000
hour_16        0.000000
hour_17        0.000000
hour_18        1.000000
hour_19        0.000000
hour_20        0.000000
hour_21        0.000000
hour_22        0.000000
hour_23        0.000000
day_1          0.000000
day_2          0.000000
day_3          1.000000
day_4          0.000000
day_5          0.000000
day_6          0.000000
region_1       1.000000
region_2       0.000000
Name: 14043, dtype: float64
```

Part 3: Model Selection

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

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

Question 3a

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

In [46]:

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

# BEGIN YOUR CODE
# -----------------------
constant_rmse = rmse(np.mean(train['duration']) - test['duration'])
# -----------------------
# END YOUR CODE
constant_rmse
```

Out[46]:

```
399.14375723526661
```

In [47]:

```
ok.grade("q3a");
```

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

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

Question 3b

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

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

In [50]:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
# BEGIN YOUR CODE
# -----------------------
model.fit(train[['distance']],train['duration'])
predictions =  model.predict(test[['distance']])
# -----------------------
# END YOUR CODE
errors = predictions - test['duration']
simple_rmse = rmse(errors)
simple_rmse
```

Out[50]:

```
276.78411050003422
```

In [51]:

```
ok.grade("q3b");
```

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

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

Question 3c

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

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

In [52]:

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

Out[52]:

```
255.19146631882754
```

In [53]:

```
ok.grade("q3c");
```

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

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

Question 3d

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

In [55]:

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

for v in np.unique(train['period']):
    # BEGIN YOUR CODE
    # -----------------------
    v_train = train[train['period'] == v]
    v_test = test[test['period'] == v]
    model.fit(design_matrix(v_train), v_train['duration'])
    predictions = model.predict(design_matrix(v_test))
    # -----------------------
    # END YOUR CODE
    errors.extend(predictions - v_test['duration'])
period_rmse = rmse(np.array(errors))
period_rmse
```

Out[55]:

```
246.62868831165173
```

In [56]:

```
ok.grade("q3d");
```

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

---------------------------------------------------------------------
Test summary
    Passed: 2
    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.

Question 3e

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

Answer: `The period regression model outperforms the linear regression model due to the fact that it divides the data up into "clusters" which have distinct patterns from each other. The ordinary regression model is less accurate because it is essentially averaging over all of these three clusters while the period model creates an individual model for each of the three different clusterings and thus can predict more accurately for each individual cluster.`

Question 3f

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.

In [60]:

```
model = LinearRegression()
# BEGIN YOUR CODE
# -----------------------
model.fit(design_matrix(train), train['speed'])
speed_predictions = model.predict(design_matrix(test))
duration_predictions = test['distance']/ speed_predictions * 60 * 60
# -----------------------
# END YOUR CODE
errors = duration_predictions - test['duration']
speed_rmse = rmse(errors)
speed_rmse
```

Out[60]:

```
243.01798368514949
```

In [61]:

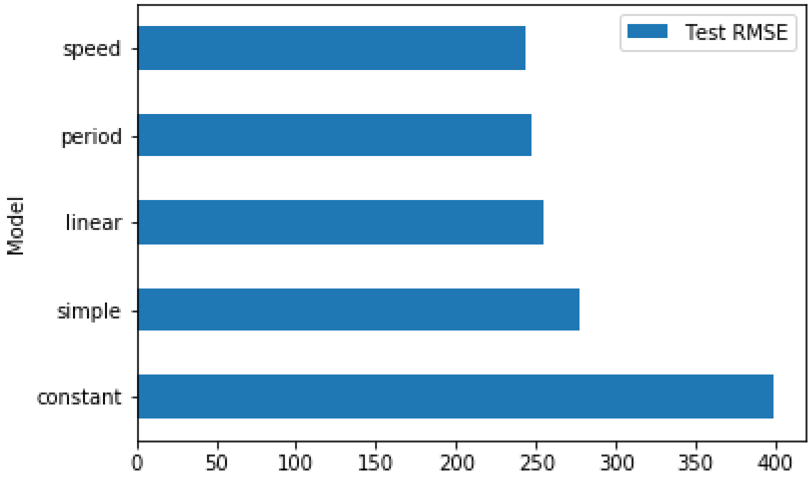
```
ok.grade("q3f");
```

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

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

Here's a summary of your results:

In [62]:

```
models = ['constant', 'simple', 'linear', 'period', '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 `EDA`, you used the data to assess the impact of a historical event---the 2016 blizzard---and filtered the data accordingly.
- In Part 2 on `feature engineering`, you used PCA to divide up the map of Manhattan into regions that roughly corresponded to the standard geographic description of the island.
- In Part 3 on `model selection`, you found that using linear regression in practice can involve more than just choosing a design matrix. Tree regression made better use of categorical variables than linear regression. The domain knowledge that duration is a simple function of distance and speed allowed you to predict duration more accurately by first predicting speed.

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

Congratulations! You have completed Project 2.

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output.,

Please save before submitting!

Please generate pdf as follows and submit it to Gradescope.

File > Print Preview > Print > Save as pdf