Project: Predicting Taxi Ride Duration

Due Date: 2024.6.19 (Wed) 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.

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

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
Total	22

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:

```
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.

```
manhattan_taxi = pd.read_csv('/content/drive/MyDrive/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://www.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.

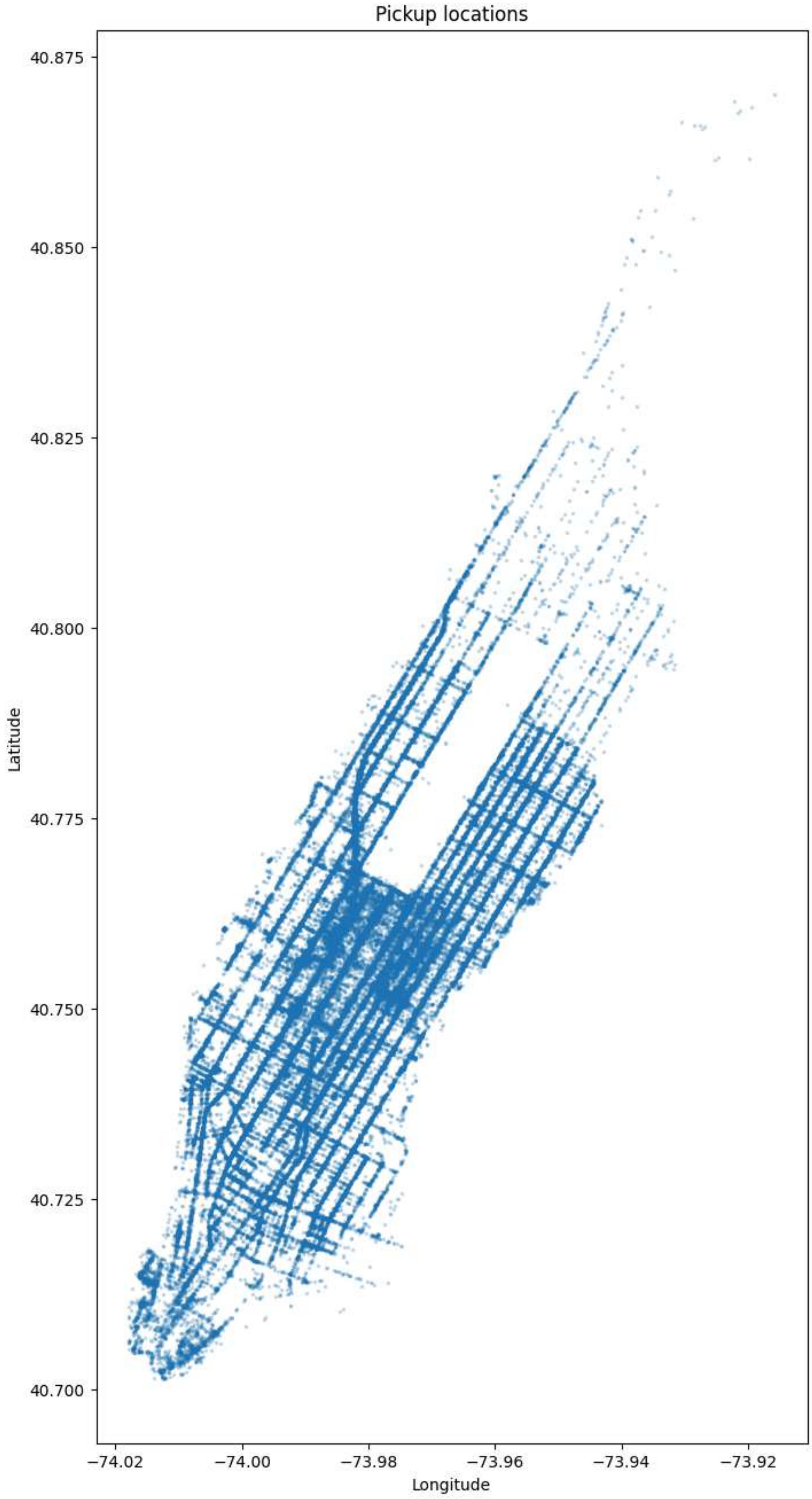
```
manhattan_taxi.head()
```

	pickup_datetime	dropoff_datetime	pickup_lon	pickup_lat	dropoff_lon	dropoff_lat	passengers	distance	duration
0	2016-01-30 22:47:32	2016-01-30 23:03:53	-73.988251	40.743542	-74.015251	40.709808	2	3.99	981
1	2016-01-04 04:30:48	2016-01-04 04:36:08	-73.995888	40.760010	-73.975388	40.782200	1	2.03	320
2	2016-01-07 21:52:24	2016-01-07 21:57:23	-73.990440	40.730469	-73.985542	40.738510	1	0.70	299
3	2016-01-08 18:46:10	2016-01-08 18:54:00	-74.004494	40.706989	-74.010155	40.716751	5	0.97	470
4	2016-01-02 12:39:57	2016-01-02 12:53:29	-73.958214	40.760525	-73.983360	40.760406	1	1.70	812

Next steps: Generate code with `manhattan_taxi` View recommended plots

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

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

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



Part 1: Exploratory Data Analysis

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

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

Note that January 2016 had some atypical days.

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

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

Question 1a

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

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

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

	pickup_datetime	dropoff_datetime	pickup_lon	pickup_lat	dropoff_lon	dropoff_la
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

Next steps: Generate code with `manhattan_taxi` View recommended plots

```
assert(list(manhattan_taxi.groupby('date').size())[:8]==[2337, 2411, 2177, 2368, 2630, 2721, 2908, 3010])

print('Passed all unit tests!')
```

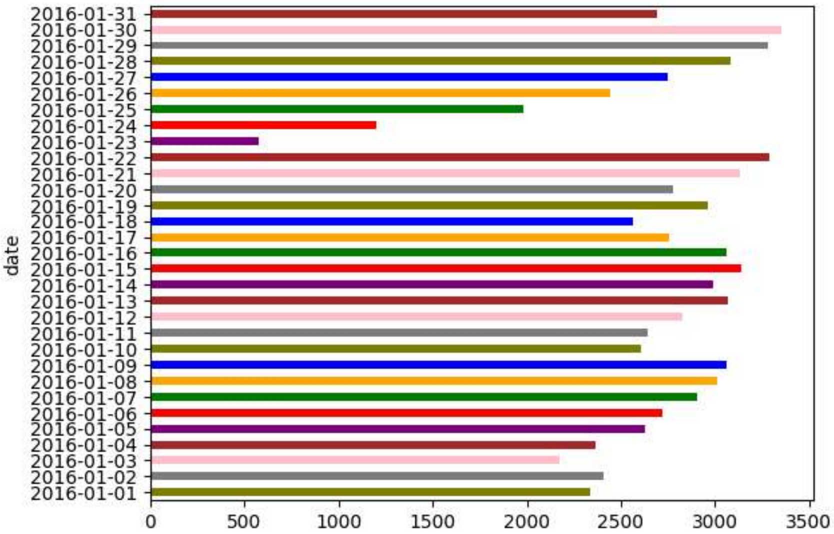
```
Passed all unit tests!
```

Question 1b

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

```
from itertools import cycle, islice
date = manhattan_taxi.groupby('date').size()
my_colors = list(islice(cycle(['olive', 'grey', 'pink', 'brown', 'purple', 'r', 'g', 'orange', 'b']), None, len(manhattan_taxi)))
date.plot.barh(color=my_colors)
```

```
<Axes: ylabel='date'>
```



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

```
import calendar
import re

from datetime import date

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

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

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

```
Typical dates:

    January 2016
Mo Tu We Th Fr Sa Su

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

Part 2: Feature Engineering

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

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

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

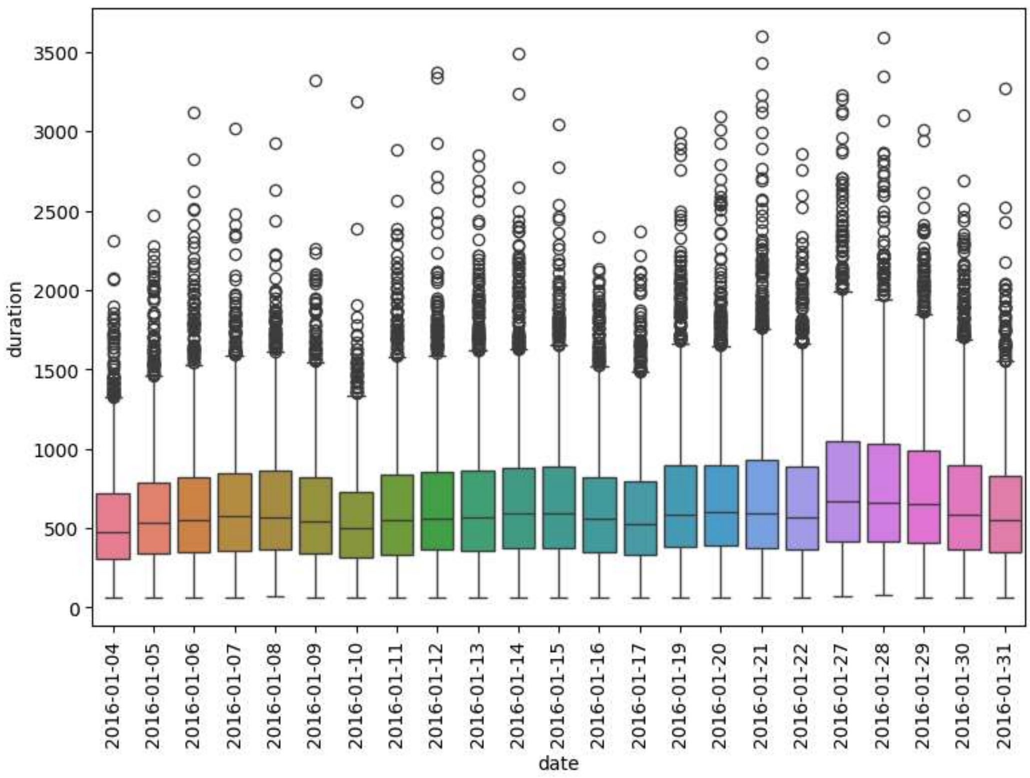
```
import sklearn.model_selection

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:

```
plt.figure(figsize=(9, 6))
# BEGIN YOUR CODE
sns.boxplot(data=train.sort_values('date'), x=train['date'].sort_values(), y='duration', hue='date', palette="husl", legend=False).tick_
# END YOUR CODE
```



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: I think the association from my perspective, is that if it is the weekends there is less traffic, since not that many people work on sundays and saturdays therefore duration is less on weekends compared to weekdays (working days).

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

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

No changes are required; just run this cell.

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

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

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

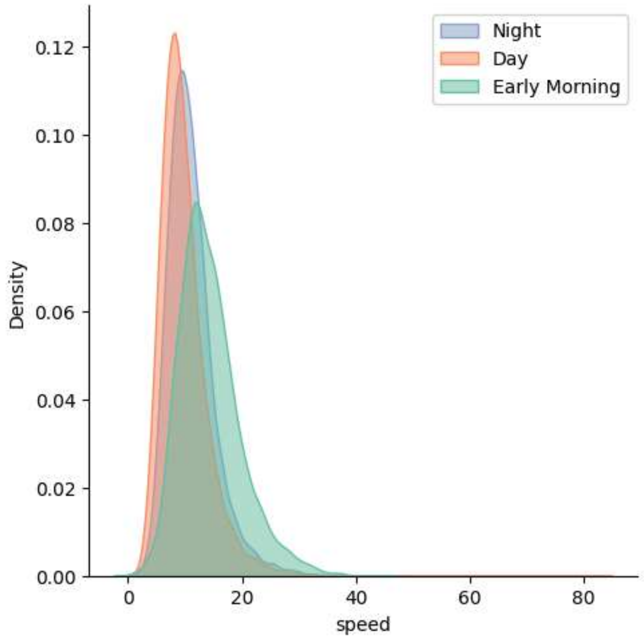
```
pickup_datetime     2016-01-21 18:02:20
dropoff_datetime    2016-01-21 18:27:54
pickup_lon                   -73.994202
pickup_lat                    40.751019
dropoff_lon                  -73.963692
dropoff_lat                   40.771069
passengers                            1
distance        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

Question 2c

Use `sns.distplot` (can use other functions like `sns.displot` or `sns.histplot` for seaborn v0.14.0 or greater) to create an overlaid histogram comparing the distribution of average speeds for taxi rides that start in the early morning (12am-6am), day (6am-6pm; 12 hours), and night (6pm-12am; 6 hours). Your plot should look like this:

```
plt.figure(figsize=(8, 8))
# BEGIN YOUR CODE
# common_norm if True normalization will apply over the full dataset
# density normalizes such that the total area of the histogram equals 1
sns.displot(data=train, hue='period', x='speed', kind='kde', common_norm=False, palette="Set2", alpha=0.5, fill=True, legend=False)
plt.legend(['Night', 'Day', 'Early Morning'])
# END YOUR CODE
```

```
<matplotlib.legend.Legend at 0x7cedd92173a0>
<Figure size 800x800 with 0 Axes>
```



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

Question 2d (PCA)

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

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

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

```
# Find the first principle component
D = train[['pickup_lon', 'pickup_lat']].values
pca_n = D.shape[0]
pca_means = np.mean(D, axis=0)    # used to center the data
X = (D - pca_means) / np.sqrt(pca_n)  # standardization
# decomposes X into u, s, vt
u, s, vt = np.linalg.svd(X, full_matrices=False)   # s is here sigma

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
    # variance = (ith singular value)^2 / N
    X = (D - pca_means) / np.sqrt(pca_n)
    first_pc = np.matmul(X, vt.T[:,0])
    # -----------------------
    # END YOUR CODE
    t.loc[:,'region'] = pd.qcut(first_pc, 3, labels=[0, 1, 2])

add_region(train)
add_region(test)
```

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

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

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

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

No prior knowledge of New York geography was required!

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

...

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

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

Note that,

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

```
from sklearn.preprocessing import StandardScaler

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

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

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

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

...

Part 3: Model Selection

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

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

Question 3a

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

```
def rmse(errors):
  # rmse: how similar, on average, are the numbers in list1 to list2
    """Return the root mean squared error."""
    return np.sqrt(np.mean(errors ** 2))

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

...

```
assert(np.isclose(constant_rmse, 399.14376, 1e-4))
assert(350 <= constant_rmse)
assert(constant_rmse <= 450)

print('Passed all unit tests!')
```

...

Question 3b

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

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

```
from sklearn.linear_model import LinearRegression

# fitting is equal to training
model = LinearRegression()
# BEGIN YOUR CODE
# -----------------------
model.fit(train[['distance']], train.loc[:, 'duration'])
predictions = model.predict(test[['distance']])
# -----------------------
# END YOUR CODE
errors = predictions - test['duration']
simple_rmse = rmse(errors)
simple_rmse
```

...

```
assert(260 <= simple_rmse)
assert(simple_rmse <= 300)
assert(np.isclose(simple_rmse, 276.78411, 1e-4))

print('Passed all unit tests!')
```

...

Question 3c

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

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

```
model = LinearRegression()
# BEGIN YOUR CODE
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
```

...

```
assert(list(design_matrix(test).sum())[10:15]==[290.0, 511.0, 699.0, 687.0, 683.0])
assert(250 <= linear_rmse)
assert(linear_rmse <= 260)
assert(np.isclose(linear_rmse, 255.19147, 1e-4))

print('Passed all unit tests!')
```

...

Question 3d

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

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

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

...

```
assert(240 <= period_rmse)
assert(period_rmse <= 255)
assert(np.isclose(period_rmse, 246.628688, 1e-4))

print('Passed all unit tests!')
```

...

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

Question 3e

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

Answer: As the question itself states design matrix for linear regression already includes one feature for each possible hour, I think this means that linear regression having more "data" while training won't be able to perform well on unseen data.

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.

```
model = LinearRegression()
# BEGIN YOUR CODE
# -----------------------
model.fit(design_matrix(train), train['speed'])
speed_predictions = model.predict(design_matrix(test))
duration_predictions = (test['distance'] * 3600) / speed_predictions
# -----------------------
# END YOUR CODE
errors = duration_predictions - test['duration']
speed_rmse = rmse(errors)
speed_rmse
```

...

```
assert(240 <= speed_rmse)
assert(speed_rmse <= 255)
assert(np.isclose(speed_rmse, 243.01798, 1e-4))

print('Passed all unit tests!')
```

...

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.