Before you turn in the homework, make sure everything runs as expected. To do so, select **Kernel→Restart & Run All** in the toolbar above. Remember to submit both on **DataHub** and **Gradescope**.

Please fill in your name and include a list of your collaborators below.

In [1]:
```
NAME = "Junsheng Pei"
COLLABORATORS = ""
```

Project 2: NYC Taxi Rides

Part 4: Feature Engineering and Model Fitting

In this final part of the project, you will finally build a regression model that attempts to predict the duration of a taxi ride from all other available information.

You will build this model using a processing pipeline and submit your results to Kaggle. We will first walk you through a generic example using the data we saved from Part 1. Please carefully follow these steps as you will need to repeat this for your final model. After, we give you free reign and let you decide how you want to define your final model.

In [2]:
```
import os
import pandas as pd
import numpy as np
import sklearn.linear_model as lm
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from sqlalchemy import create_engine
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV

sns.set(style="whitegrid", palette="muted")

plt.rcParams['figure.figsize'] = (12, 9)
plt.rcParams['font.size'] = 12

%matplotlib inline
```

Training and Validation

The following code loads the training and validation data from part 1 into a Pandas DataFrame.

In [3]:
```
# Run this cell to load the data.
data_file = Path("./", "cleaned_data.hdf")
train_df = pd.read_hdf(data_file, "train")
val_df = pd.read_hdf(data_file, "val")
```

In [4]:
```
train_df.head()
```

Out[4]:

	record_id	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag	...	dropoff_latitude	payment_type	fare_amount	ex
13242	5711100	1	2016-01-17 17:48:41	2016-01-17 17:55:53	1	1.00	-74.006470	40.738766	1	N	...	40.735664	1	6.5	
12723	4989400	1	2016-01-17 01:18:39	2016-01-17 01:21:15	1	0.40	-73.989365	40.763000	1	N	...	40.766121	2	4.0	
8508	2436400	2	2016-01-12 09:07:00	2016-01-12 09:41:17	1	11.40	-73.984108	40.774509	1	N	...	40.770458	1	37.0	
21304	10899100	2	2016-01-29 09:07:54	2016-01-29 09:18:25	1	1.42	-74.002907	40.760262	1	N	...	40.742764	1	8.5	
3817	1319400	1	2016-01-06 11:44:54	2016-01-06 11:49:55	1	0.80	-73.969742	40.760273	1	N	...	40.751129	2	5.0	

5 rows × 21 columns

Testing

Here we load our testing data on which we will evaluate your model.

In [5]:
```
test_df = pd.read_csv("./proj2_test_data.csv")
test_df['tpep_pickup_datetime'] = pd.to_datetime(test_df['tpep_pickup_datetime'])
test_df.head()
```

Out[5]:

	record_id	VendorID	tpep_pickup_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag	dropoff_longitude	dropoff_latitude	payment_type	fare_amount	extra	mta_tax
0	10000	1	2016-01-02 01:45:37	1	1.20	-73.982224	40.768620	1	N	-73.983765	40.779598	1	6.0	0.5	0.5
1	19000	2	2016-01-02 03:05:16	1	10.90	-73.999977	40.738121	1	N	-73.888657	40.824364	1	31.5	0.5	0.5
2	21000	1	2016-01-02 03:24:36	1	1.80	-73.986618	40.747379	1	N	-73.978508	40.729622	1	8.5	0.5	0.5
3	23000	2	2016-01-02 03:47:38	1	5.95	-74.002922	40.744572	1	N	-73.942413	40.786419	1	20.5	0.5	0.5
4	27000	1	2016-01-02 04:36:44	1	1.60	-73.986366	40.759464	1	N	-73.963081	40.760353	2	8.0	0.5	0.5

In [6]:
```
test_df.describe()
```

Out[6]:

	record_id	VendorID	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	dropoff_longitude	dropoff_latitude	payment_type	fare_amount	extra	mta_tax	tip_amount
count	1.377400e+04	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000
mean	3.465950e+07	1.536082	1.663642	2.954688	-72.953619	40.187999	1.043778	-73.055577	40.245056	1.340061	12.836930	0.333091	0.497985	1.805420
std	2.015133e+07	0.498714	1.311739	3.704427	8.628431	4.753186	0.877637	8.191366	4.512564	0.490019	10.707619	0.429590	0.036632	2.416784
min	1.000000e+04	1.000000	0.000000	0.000000	-77.039436	0.000000	1.000000	-77.039436	0.000000	1.000000	-93.300000	-0.500000	-0.500000	0.000000
25%	1.719975e+07	1.000000	1.000000	1.000000	-73.992058	40.735166	1.000000	-73.991318	40.734002	1.000000	6.500000	0.000000	0.500000	0.000000
50%	3.457400e+07	2.000000	1.000000	1.700000	-73.981846	40.752432	1.000000	-73.979897	40.753263	1.000000	9.500000	0.000000	0.500000	1.350000
75%	5.216875e+07	2.000000	2.000000	3.157500	-73.967119	40.767264	1.000000	-73.962749	40.768455	2.000000	14.500000	0.500000	0.500000	2.360000
max	6.940400e+07	2.000000	6.000000	104.800000	0.000000	40.868210	99.000000	0.000000	41.540859	4.000000	156.040000	4.500000	1.740000	40.000000

Modeling

We've finally gotten to a point where we can specify a simple model. Remember that we will be fitting our model on the training set we created in part 1. We will use our validation set to evaluate how well our model might perform on future data.

Reusable Pipeline

Throughout this assignment, you should notice that your data flows through a single processing pipeline several times. From a software engineering perspective, this should be sufficient motivation to abstract parts of our code into reusable functions/methods. We will now encapsulate our entire pipeline into a single function `process_data_gm` . gm is shorthand for "guided model".

In [7]:
```python
# Copied from part 2
def haversine(lat1, lng1, lat2, lng2):
    """
    Compute haversine distance
    """
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    average_earth_radius = 6371
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * average_earth_radius * np.arcsin(np.sqrt(d))
    return h

# Copied from part 2
def manhattan_distance(lat1, lng1, lat2, lng2):
    """
    Compute Manhattan distance
    """
    a = haversine(lat1, lng1, lat1, lng2)
    b = haversine(lat1, lng1, lat2, lng1)
    return a + b

# Copied from part 2
def bearing(lat1, lng1, lat2, lng2):
    """
    Compute the bearing, or angle, from (lat1, lng1) to (lat2, lng2).
    A bearing of 0 refers to a NORTH orientation.
    """
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))

# Copied from part 2
def add_time_columns(df):
    """
    Add temporal features to df
    """
    df.is_copy = False # propogate write to original dataframe
    df.loc[:, 'month'] = df['tpep_pickup_datetime'].dt.month
    df.loc[:, 'week_of_year'] = df['tpep_pickup_datetime'].dt.weekofyear
    df.loc[:, 'day_of_month'] = df['tpep_pickup_datetime'].dt.day
    df.loc[:, 'day_of_week'] = df['tpep_pickup_datetime'].dt.dayofweek
    df.loc[:, 'hour'] = df['tpep_pickup_datetime'].dt.hour
    df.loc[:, 'week_hour'] = df['tpep_pickup_datetime'].dt.weekday * 24 + df['hour']
    return df

# Copied from part 2
def add_distance_columns(df):
    """
    Add distance features to df
    """
    df.is_copy = False # propogate write to original dataframe
    df.loc[:, 'manhattan'] = manhattan_distance(lat1=df['pickup_latitude'],
                                                lng1=df['pickup_longitude'],
                                                lat2=df['dropoff_latitude'],
                                                lng2=df['dropoff_longitude'])

    df.loc[:, 'bearing'] = bearing(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    df.loc[:, 'haversine'] = haversine(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    return df

def select_columns(data, *columns):
    return data.loc[:, columns]
```

In [8]:
```python
def process_data_gm1(data, test=False):
    X = (
        data

        # Transform data
        .pipe(add_time_columns)
        .pipe(add_distance_columns)

        .pipe(select_columns,
              'pickup_longitude',
              'pickup_latitude',
              'dropoff_longitude',
              'dropoff_latitude',
              'manhattan',
             )
    )
    if test:
        y = None
    else:
        y = data['duration']

    return X, y
```

We will use our pipeline defined above to pre-process our training and test data in exactly the same way. Our functions make this relatively easy to do!

In [9]:
```python
# Train
X_train, y_train = process_data_gm1(train_df)
X_val, y_val = process_data_gm1(val_df)
guided_model_1 = lm.LinearRegression(fit_intercept=True)
guided_model_1.fit(X_train, y_train)

# Predict
y_train_pred = guided_model_1.predict(X_train)
y_val_pred = guided_model_1.predict(X_val)
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

Here, `y_val` are the correct durations for each ride, and `y_val_pred` are the predicted durations based on the 7 features above (`vendorID` , `passenger_count` , `pickup_longitude` , `pickup_latitude` , `dropoff_longitude` , `dropoff_latitude` , `manhattan`).

In [10]:
```python
assert 600 <= np.median(y_train_pred) <= 700
assert 600 <= np.median(y_val_pred) <= 700
```

The resulting model really is a linear model just like we saw in class, i.e. the predictions are simply generated by the product $\Phi\theta$. For example, the line of code below generates a prediction for x_1 by computing $\phi_1^T\theta$. Here `guided_model_1.coef_` is θ and `X_train.iloc[0, :]` is ϕ_1.

Note that unlike in class, here the dummy intercept term is not included in Φ.

In [11]:
```
X_train.iloc[0, :].dot(guided_model_1.coef_) + guided_model_1.intercept_
```

Out[11]: 558.751330511368

We see that this prediction is exactly the same (except for possible floating point error) as generated by the `predict` function, which simply computes the product $\Phi\theta$, yielding predictions for every input.

In [12]:
```
y_train_pred[0]
```

Out[12]: 558.75133051135344

In this assignment, we will use Mean Absolute Error (MAE), a.k.a. mean L1 loss, to measure the quality of our models. As a reminder, this quantity is defined as:

$$MAE = \frac{1}{n} \sum_i |y_i - \hat{y}_i|$$

Why may we want to use the MAE as a metric, as opposed to Mean Squared Error (MSE)? Using our domain knowledge that most rides are short in duration (median is roughly 600 seconds), we know that MSE is susceptible to outliers. Given that some of the outliers in our dataset are quite extreme, it is probably better to optimize for the majority of rides rather than for the outliers. You may want to remove some of these outliers later on.

In [13]:
```
def mae(actual, predicted):
    """
    Calculates MAE from actual and predicted values
    Input:
      actual (1D array-like): vector of actual values
      predicted (1D array-like): vector of predicted/fitted values
    Output:
      a float, the MAE
    """

    mae = np.mean(np.abs(actual - predicted))
    return mae
```

In [14]:
```
assert 200 <= mae(y_val_pred, y_val) <= 300
print("Validation Error: ", mae(y_val_pred, y_val))
```

```
Validation Error:  266.136130855
```

Side note: scikit-learn also has tools to compute mean absolute error (`sklearn.metrics.mean_absolute_error`). In fact, most metrics that we have discussed in this class can be found as part of the [`sklearn.metrics` module (https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics)](https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics). Some of these may come in handy as part of your feature engineering!

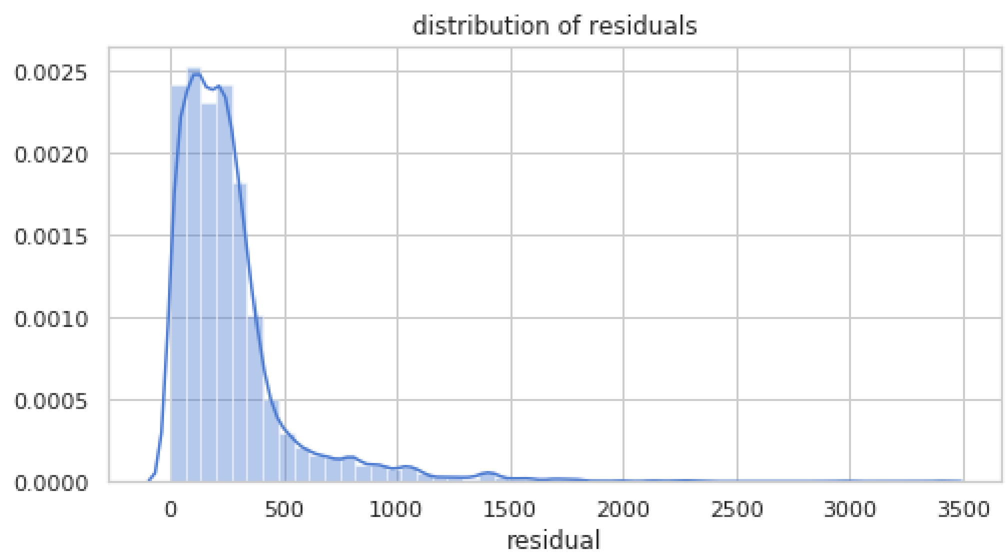
Visualizing Error

You should be getting between 200 and 300 MAE, which means your model was off by roughly 3-5 minutes on trips of average length 12 minutes. This is fairly decent performance given that our basic model uses only using the pickup/dropoff latitude and manhattan distance of the trip. 3-5 minutes may seem like a lot for a trip of 12 minutes, but keep in mind that this is the *average* error. This metric is susceptible to extreme outliers, which exist in our dataset.

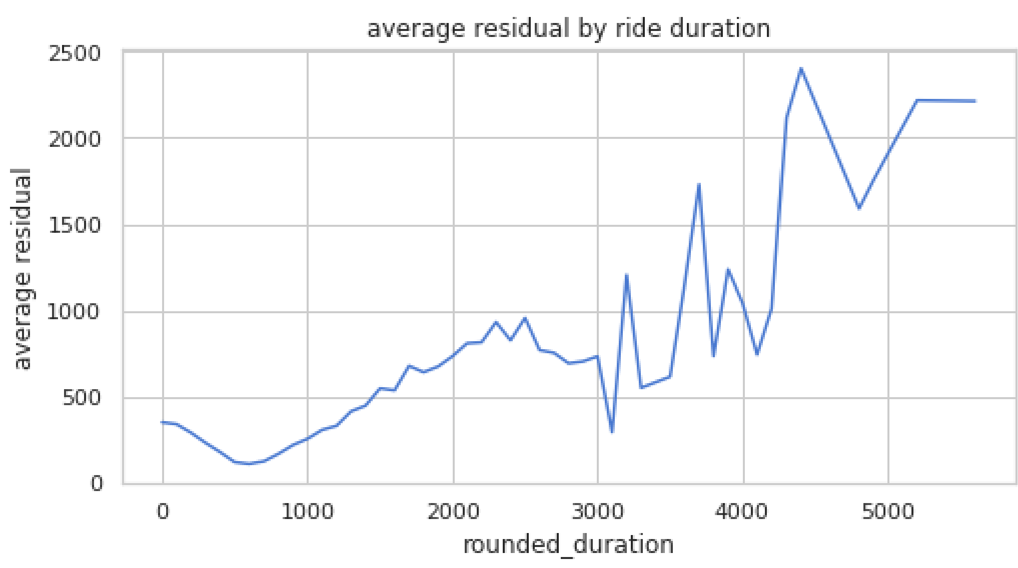
Now we will visualize the residual for the validation set. We will plot the following:

1. Distribution of residuals
2. Average residual grouping by ride duration

In [15]:
```
# Distribution of residuals
plt.figure(figsize=(8,4))
sns.distplot(np.abs(y_val - y_val_pred))
plt.xlabel('residual')
plt.title('distribution of residuals');
```



In [16]:
```
# Average residual grouping by ride duration
val_residual = X_val.copy()
val_residual['duration'] = y_val
val_residual['rounded_duration'] = np.around(y_val, -2)
val_residual['residual'] = np.abs(y_val - y_val_pred)
tmp = val_residual.groupby('rounded_duration').mean()
plt.figure(figsize=(8,4))
tmp['residual'].plot()
plt.ylabel('average residual')
plt.title('average residual by ride duration');
```



In the first visualization, we see that most of the residuals are centered around 250 seconds ~ 4 minutes. There is a minor right tail, suggesting that we are still unable to accurately fit some outliers in our data. The second visualization also suggests this, as we see the average residual increasing as a somewhat linear function of duration. But given that our average ride duration is roughly 600-700 seconds, it seems that we are indeed optimizing for the average ride because the residuals are smallest around 600-700.

Keep this in mind when creating your final model! Visualizing the error is a powerful tool and may help diagnose shortcomings of your model. Let's go ahead and submit to kaggle, although your error on the test set may be higher than 300.

Submission to Kaggle

The following code will write your predictions on the test dataset to a CSV, which you can submit to Kaggle. You may need to modify it to suit your needs, but we recommend you make a copy and preserve the original function.

Remember that if you've performed transformations or featurization on the training data, you must also perform the same transformations on the test data in order to make predictions. For example, if you've created features for the columns `pickup_datetime` or `pickup_latitude` on the training data, you must also extract the same features in order to use scikit-learn's `.predict(...)` method.

In [17]:
```
from datetime import datetime
def generate_submission(test, predictions, force=False):
    if force:
        if not os.path.isdir("submissions"):
            os.mkdir("submissions")
        submission_df = pd.DataFrame({
            "id": test_df.index.values,
            "duration": predictions,
        },
            columns=['id', 'duration'])

        timestamp = datetime.isoformat(datetime.now()).split(".")[0]

        submission_df.to_csv(f'submissions/submission_{timestamp}.csv', index=False)

        print(f'Created a CSV file: submission_{timestamp}.csv')
        print('You may now upload this CSV file to Kaggle for scoring.')
```

In [18]:
```
X_test, _ = process_data_gm1(test_df, True)
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

In [19]:
```
assert list(X_train.columns) == list(X_test.columns), "Different columns or different column ordering"
submission_predictions = (guided_model_1
                          .fit(X_train, y_train)
                          .predict(X_test))
submission_predictions = submission_predictions.astype(int)
submission_predictions[submission_predictions < 0] = 0
generate_submission(test_df, submission_predictions, True)
```

```
Created a CSV file: submission_2018-12-04T22:28:34.csv
You may now upload this CSV file to Kaggle for scoring.
```

In [20]:
```
# Check your submission
assert isinstance(submission_predictions, np.ndarray), "Submission not an array"
assert all(submission_predictions >= 0), "Duration must be non-negative"
assert issubclass(submission_predictions.dtype.type, np.integer), "Seconds must be integers"
```

Your Turn!

Now it's your turn! Draw upon everything you have learned this semester to find the best features to help your model accurately predict the duration of a taxi ride.

You may use whatever method you prefer in order to create features. You may use features that we created and features that you discovered yourself from any of the 2 datasets. However, we want to make it fair to students who are seeing these techniques for the first time. As such, you are only allowed regression models and their regularized forms. This means no random forest, k-nearest-neighbors, neural nets, etc.

Here are some ideas to improve your model:

- **Data selection**: January 2016 was an odd month for taxi rides due to the blizzard. Would it help to select training data differently?
- **Data cleaning**: Try cleaning your data in different ways. In particular, consider how to handle outliers.
- **Better features**: Explore the 2 datasets and find what features are most helpful. Utilize external datasets to improve your accuracy.
- **Regularization**: Try different forms of regularization to avoid fitting to the training set. Recall that `Ridge` and `Lasso` are the names of the classes in `sklearn.linear_model` that combine `LinearRegression` with regularization techniques.
- **Model selection**: You can adjust parameters of your model (e.g., the regularization parameter) to achieve higher accuracy. [GridSearchCV (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) may be helpful.
- **Validation**: Recall that you should use cross-validation to do feature and model selection properly! Otherwise, you will likely overfit to your training data.

There's many things you could try that could help your model. We have only suggested a few. Be creative and innovative! Please use `proj2_extras.ipynb` for all of your extraneous work. Note that you will be submitting `proj2_extras.ipynb` and we will be grading it. Please properly comment and format this notebook!

Once you are satisfied with your results, answer the questions in the Deliverables section. You may want to read this section in advance so you have an idea of what we're looking for.

Deliverables

Feature/Model Selection Process

Let's first look at selection of better features. In this following cell, describe the process of choosing good features to improve your model. You should use at least 3-4 sentences each to address the follow questions. Backup your responses with graphs supporting your claim (you can save figures and load them, no need to add the plotting code here). Use these questions to concisely summarize all of your extra work!

Question 1a

How did you find better features for your model?

In [21]:
```
q1a_answer = r"""
I testet more columns in the given dataset by considering each coloumn as a feature and visualizing the
relationship between such feature and duration.

If the values of the feature are discrete, I use barplot. And if the duration distincts from each value, I add
this feature to my feature matirx.

If the values of the feature are continuous, I use scatter plot(we can also find outliers with scatter plot).
And if the duration have a linear relationship with such feature, I add this feature to my feature matirx.

If the feature helps improve the accuray, it's a good feature.

"""
# YOUR CODE HERE
#raise NotImplementedError()
```

Question 1b

What did you try that worked / didn't work?

In [22]:
```
q1b_answer = r"""

I found that 'tip_amount', 'haversine', 'trip_distance', 'ifdaytime', 'ifweekday', 'total_amount' and
'fare_amount' worked.

However, record_id', 'VendorID', passenger_count','RatecodeID','store_and_fwd_flag','payment_type',
'improvement_surcharge', 'total_amount', 'month','week_of_year' and 'day_of_month' didn't work.

"""
# YOUR CODE HERE
#raise NotImplementedError()
```

Question 1c

What was surprising in your search for good features?

In [23]:
```
q1c_answer = r"""
I found that 'total_amount' is extremely helpful, it is proportional to the duration.
And I found replacing the outliers with average value can make the feature more helpful, like pickup_latitude,
pickup_longtitute have many values as 0.

"""
# YOUR CODE HERE
#raise NotImplementedError()
```

Question 2

Just as in the guided model above, you should encapsulate as much of your workflow into functions as possible. Define `process_data_fm` and `final model` in the cell below. In order to calculate your final model's MAE, we will run the code in the cell after that.

Note: You *MUST* name the model you wish to be evaluated on `final_model`. This is what we will be using to generate your predictions. We will take the state of `final_model` right after executing the cell below and run the following code:

```
# Load in test_df, solutions
X_test, _ = process_data_fm(test_df, True)
submission_predictions = final_model.predict(X_test)
# Generate score for autograding
```

We encourage you to conduct all of your exploratory work in `proj2_extras.ipynb`, which will be graded for 10 points.

In [24]:
```
data_file_fm = Path("./", "cleaned_data_2016.hdf")
train_df_fm = pd.read_hdf(data_file_fm, "train")
val_df_fm = pd.read_hdf(data_file_fm, "val")
```

In [25]:
```
def add_ifdaytime(data):
    data['ifdaytime'] = (data['hour'] >= 8) & (data['hour'] <= 18)
    return data

def add_ifweekday(data):
    data['ifweekday'] = data['day_of_week'] > 4
    return data

def drop_outlier(data, col, _filter):
    return data.loc[data[col][lambda x: _filter(x)].index]

def replace_outlier(data, col, _filter):
    mean = data[col][lambda x : _filter(x)].mean()
    data[col] = data[col].apply(lambda x : x if _filter(x) else mean)
    return data



```

In [26]:
```
def process_data_fm(data, test=False):
    # Put your final pipeline here

    # data cleaning
    if(test):
        clean_data = replace_outlier
    else:
        clean_data = drop_outlier
        data =  clean_data(data,'duration', lambda x : (x < 8000) & x > 0)

    filter_latitude = lambda x : (x >= 40.63) & (x <= 40.85)

    data = clean_data(data,'pickup_latitude', filter_latitude )
    data = clean_data(data,'dropoff_latitude', filter_latitude )

    filter_longitude  = lambda x : (x >= -74.03) & (x <= -73.75)

    data = clean_data(data,'pickup_longitude', filter_longitude)
    data = clean_data(data,'dropoff_longitude', filter_longitude )

    data = clean_data(data,'total_amount', lambda x: (x>0) & (x <= 90))
    data = clean_data(data,'fare_amount', lambda x: (x>0) & (x <= 80))
    data = clean_data(data,'tip_amount', lambda x :(x>0) & (x <= 20) )

    data = clean_data(data, 'trip_distance', lambda x : x < 50)

    X = (
        data
        # Transform data
        .pipe(add_time_columns)
        .pipe(add_distance_columns)
        .pipe(add_ifdaytime)
        .pipe(add_ifweekday)
        .pipe(select_columns,
              'pickup_longitude',
              'pickup_latitude',
              'dropoff_longitude',
              'dropoff_latitude',
              'manhattan',
              'tip_amount',
              'haversine',
              'trip_distance',
               'ifdaytime',
              'ifweekday',
              'total_amount',
              'fare_amount',
             )
    )
    if test:
        y = None
    else:
        y = data['duration']

    return X, y

# YOUR CODE HERE
#raise NotImplementedError()
```

In [27]:

```
X_train_fm, y_train_fm = process_data_fm(train_df_fm)
X_val_fm, y_val_fm = process_data_fm(val_df_fm)

#final_model = lm.LinearRegression(fit_intercept=True)
final_model = lm.Ridge(alpha = 3, fit_intercept=True)

# Define your final model here, feel free to try other forms of regression
final_model.fit(X_train_fm, y_train_fm)

y_train_pred_fm = final_model.predict(X_train_fm)
y_val_pred_fm = final_model.predict(X_val_fm)

print(mae(y_train_pred_fm,y_train_fm))
print(mae(y_val_pred_fm,y_val_fm))
```

```
122.552163418
122.027044081
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

In [28]:

```
# Feel free to change this cell
X_test, _ = process_data_fm(test_df, True)
final_predictions = final_model.predict(X_test)
final_predictions = final_predictions.astype(int)
generate_submission(test_df, final_predictions, True) # Change to true to generate prediction
```

```
Created a CSV file: submission_2018-12-04T22:28:35.csv
You may now upload this CSV file to Kaggle for scoring.
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

Question 3

The following hidden cells will test your model on the test set. Please do not delete any of them if you want credit!

In [29]:
```
# NO TOUCH
```

In [30]:
```
# NOH
```

In [31]:
```
# STAHP
```

In [32]:
```
# NO MOLESTE
```

In [33]:
```
# VA-T'EN
```

In [34]:
```
# NEIN
```

In [35]:
```
# PLSNO
```

In [36]:
```
# THIS SPACE IS NOT YOURS
```

In [37]:
```
# TAWDEETAW
```

In [38]:
```
# MAU LEN
```

In [39]:
```
# ALMOST
```

In [40]:
```
# TO
```

In [41]:
```
# THE
```

In [42]:
```
# END
```

In [43]:
```
# Hmph
```

In [44]:
```
# Good riddance
```

In [45]:
```
generate_submission(test_df, submission_predictions, True)
```

```
Created a CSV file: submission_2018-12-04T22:28:35.csv
You may now upload this CSV file to Kaggle for scoring.
```

This should be the format of your CSV file.
Unix-users can verify it running `!head submission_{datetime}.csv` in a jupyter notebook cell.

```
id,duration
id3004672,965.3950873305439
id3505355,1375.0665915134596
id1217141,963.2285454171943
id2150126,1134.7680929570924
id1598245,878.5495792656438
id0668992,831.6700312449248
id1765014,993.1692116960185
id0898117,1091.1171629594755
id3905224,887.9037911118357
```

Kaggle link: https://www.kaggle.com/t/f8b3c6acc3a045cab152060a5bc79670 (https://www.kaggle.com/t/f8b3c6acc3a045cab152060a5bc79670)

Submission

You're almost done!

Before submitting this assignment, ensure that you have:

1. Restarted the Kernel (in the menubar, select Kernel→Restart & Run All)
2. Validated the notebook by clicking the "Validate" button.

Then,

1. **Submit** the assignment via the Assignments tab in **Datahub**

2. **Upload and tag** the manually reviewed portions of the assignment on **Gradescope**