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 [33]:
```
NAME = "Matthew Brennan"
COLLABORATORS = "Connor McCormick"
```

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 [34]:
```
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 [35]:
```
# 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")
```

Testing

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

In [36]:
```
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[36]:

	record_id	VendorID	tpep_pickup_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fw
0	10000	1	2016-01-02 01:45:37	1	1.20	-73.982224	40.768620	1	
1	19000	2	2016-01-02 03:05:16	1	10.90	-73.999977	40.738121	1	
2	21000	1	2016-01-02 03:24:36	1	1.80	-73.986618	40.747379	1	
3	23000	2	2016-01-02 03:47:38	1	5.95	-74.002922	40.744572	1	
4	27000	1	2016-01-02 04:36:44	1	1.60	-73.986366	40.759464	1	

In [37]:
```
test_df.describe()
```

Out[37]:

	record_id	VendorID	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	dropoff_longitude	dro
count	1.377400e+04	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	1
mean	3.465950e+07	1.536082	1.663642	2.954688	-72.953619	40.187999	1.043778	-73.055577	
std	2.015133e+07	0.498714	1.311739	3.704427	8.628431	4.753186	0.877637	8.191366	
min	1.000000e+04	1.000000	0.000000	0.000000	-77.039436	0.000000	1.000000	-77.039436	
25%	1.719975e+07	1.000000	1.000000	1.000000	-73.992058	40.735166	1.000000	-73.991318	
50%	3.457400e+07	2.000000	1.000000	1.700000	-73.981846	40.752432	1.000000	-73.979897	
75%	5.216875e+07	2.000000	2.000000	3.157500	-73.967119	40.767264	1.000000	-73.962749	
max	6.940400e+07	2.000000	6.000000	104.800000	0.000000	40.868210	99.000000	0.000000	

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 [38]:
```
# 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 [39]:
```
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 [40]:
```
# 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: Attrib
ute 'is_copy' is deprecated and will be removed in a future version.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attrib
ute 'is_copy' is deprecated and will be removed in a future version.
  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 [41]:
```
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 [42]:
```
X_train.iloc[0, :].dot(guided_model_1.coef_) + guided_model_1.intercept_
```

Out[42]:
```
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 [43]:
```
y_train_pred[0]
```

Out[43]:
```
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 [44]:
```
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 [45]:
```
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). Some of these may come in handy as part of your feature engineering!

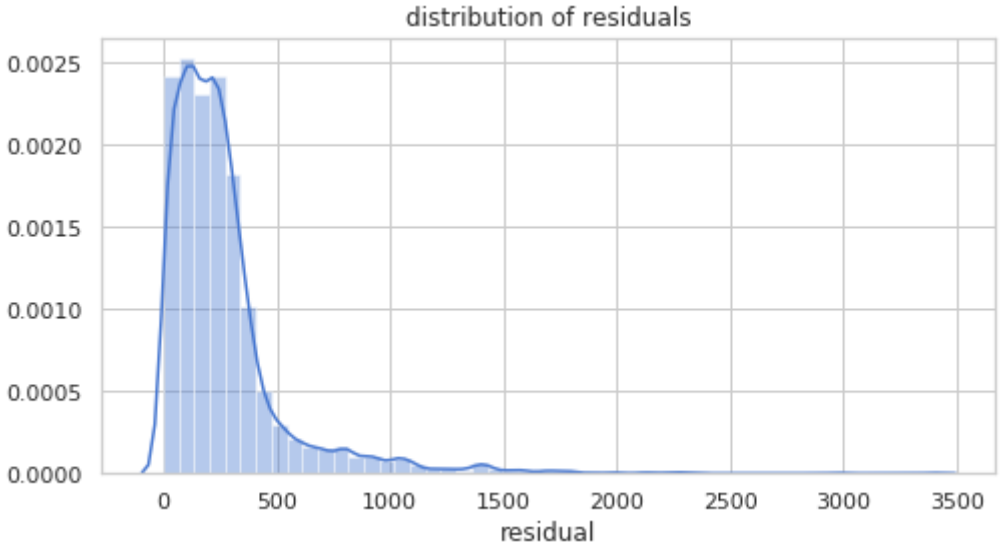
Visualizing Error

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 [46]:
```
# 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 [47]:
```
# 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 [48]:
```
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 [49]:
```
X_test, _ = process_data_gm1(test_df, True)
```

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

In [50]:
```
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_2018-12-06T00:44:07.csv
You may now upload this CSV file to Kaggle for scoring.
```

In [51]:
```
# 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) 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!

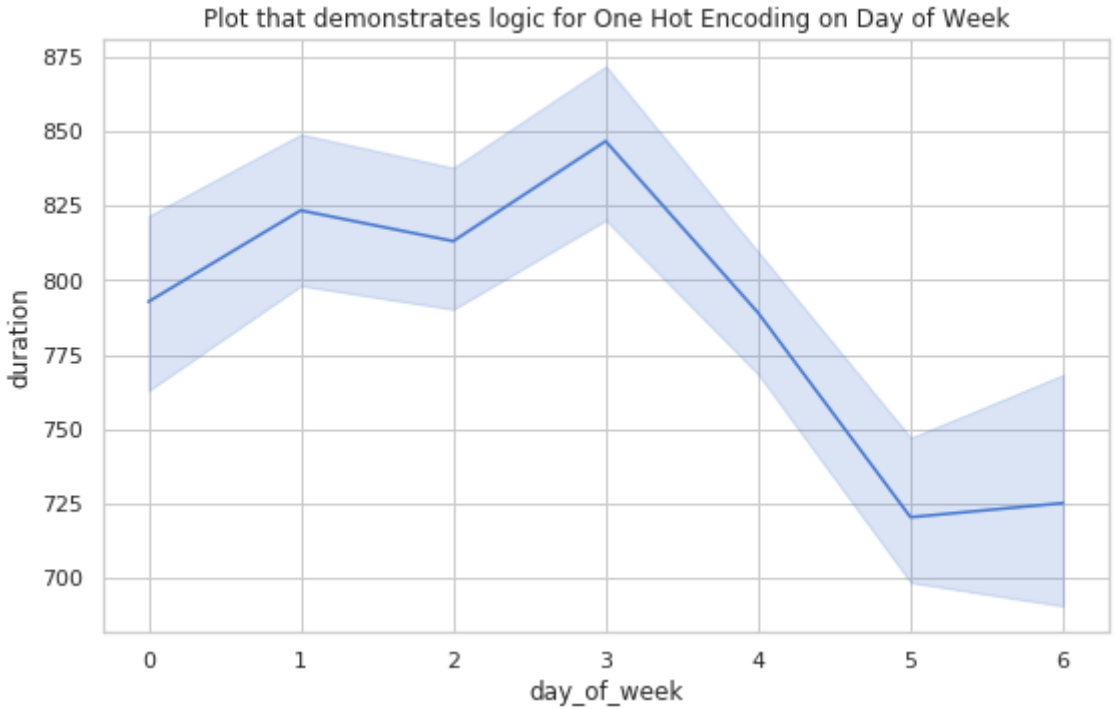
Question 1a

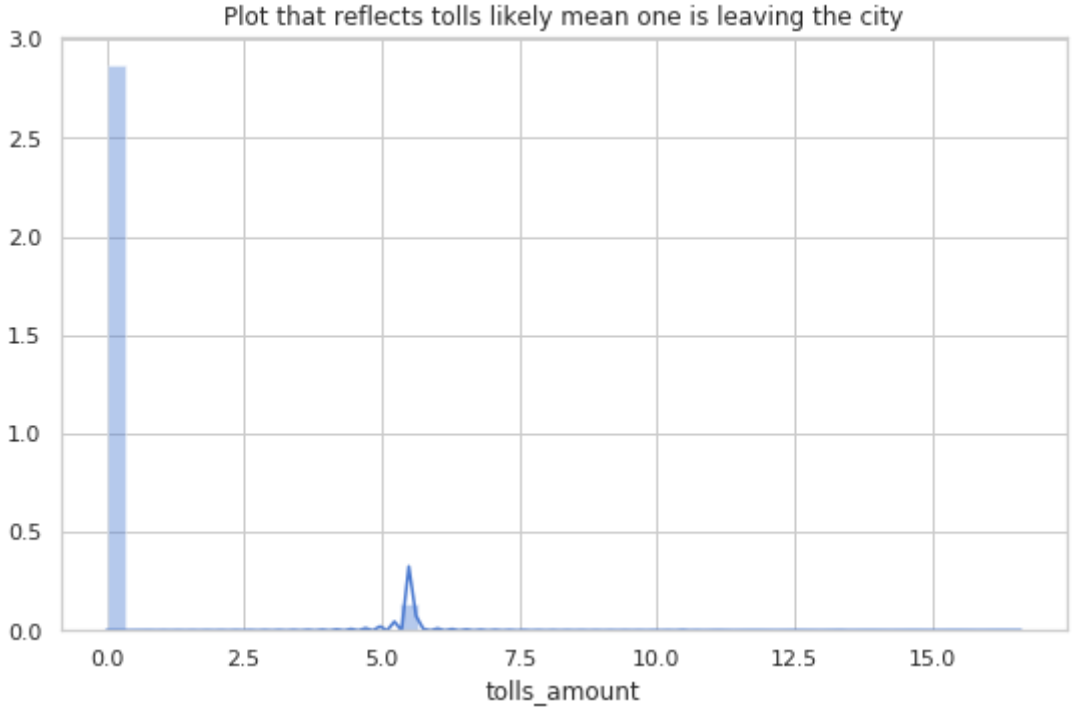
How did you find better features for your model?

In [52]:
```
q1a_answer = r"""

The process for me to find better features was focussed around both intuition about what could cause du

"""
print(q1a_answer)
# YOUR CODE HERE
#raise NotImplementedError()
```

The process for me to find better features was focussed around both intuition about what could cause d
uration to increase as well as by looking at plots that I created. We were given a fair amount of data
already which enabled me to just ponder columns of data such as tolls_amount where a toll in general m
ight mean something overall. Although tolls are ambiguous because tolls normally mean that one avoids
traffic, in New York this likely means that they are leaving the city. Further, plots played a huge ro
le in the data selection process as they enabled me to see which values were clear outliers in areas a
s well as for me to visualize trends, such as Wednesday tends to have more traffic it appears than oth
er days. As far as the day of the week, we were then able to one hot encode on this feature to allow f
or this to heavily be taken into account.

In [53]:
```
#Useful graphs that made me consider better features
fig, ((ax1), (ax2)) =  plt.subplots(ncols=1, nrows = 2, figsize=(9, 12))

sns.lineplot(x = 'day_of_week', y = 'duration', data = train_df, ax = ax1)
sns.distplot(train_df['tolls_amount'], ax = ax2)

ax1.set(title ='Plot that demonstrates logic for One Hot Encoding on Day of Week')
ax2.set(title = 'Plot that reflects tolls likely mean one is leaving the city');
```


Plot that demonstrates logic for One Hot Encoding on Day of Week
duration
875
850
825
800
775
750
725
700
0
1
2
3
4
5
6
day_of_week


Plot that reflects tolls likely mean one is leaving the city
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
tolls_amount

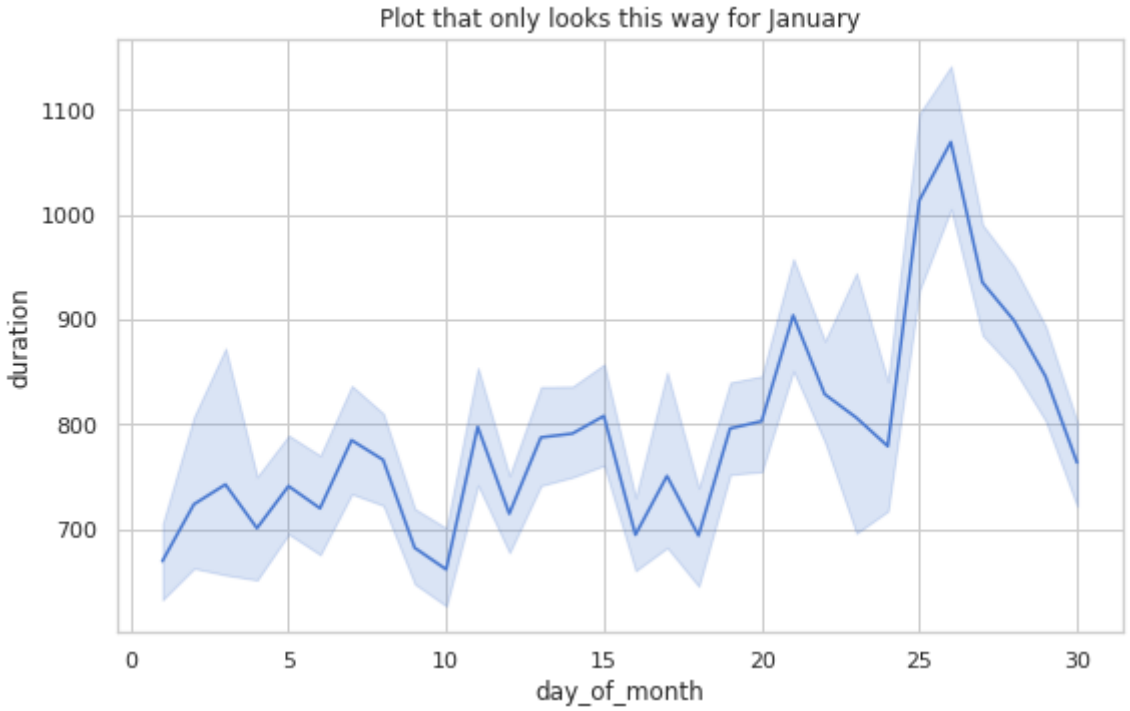
Question 1b

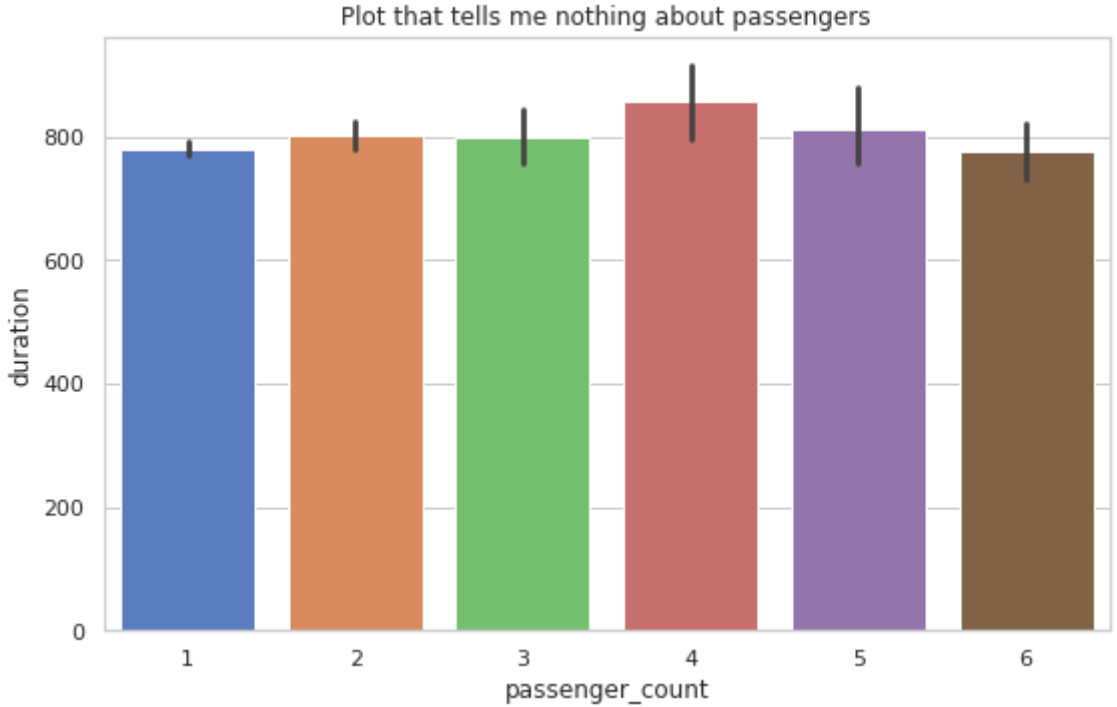
What did you try that worked / didn't work?

In [54]:
```
q1b_answer = r"""
One major problem that I endured was with correcting from the January month data versus the randomness o

"""
print(q1b_answer)
# YOUR CODE HERE
#raise NotImplementedError()
```

```
One major problem that I endured was with correcting from the January month data versus the randomness
of days for the test data. The reason for this stems from an immense problem of bias with the specific
days of the Train data because of the blizzard that occurred. With this being said, I created a featur
e called bad_day that focussed on whether day of the month was in the [23,26] range referring to the b
ad days of the blizzard where traffic was difficult. This feature was incredibly dumb because it treat
ed every [23,26] value as if the duration should be high, even if it was in a different month. Furthe
r, I created a feature that gives scores based upon proximity to an airport which seemed to not work m
ainly because of the difficulty of knowing what range of latitude and longitude effectively suggests p
roximity to the airport given that these values reflect such a large mass for a small increase or decr
ease in the coordinate.
```

In [55]:
```
#These are unuseful plots that were useful in understanding

fig, ((ax1), (ax2)) =  plt.subplots(ncols=1, nrows = 2, figsize=(9, 12))

sns.lineplot(x = 'day_of_month', y = 'duration', data = train_df, ax = ax1)
sns.barplot(x = train_df['passenger_count'] , y = train_df['duration'], ax = ax2)

ax1.set(title ='Plot that only looks this way for January')
ax2.set(title = 'Plot that tells me nothing about passengers');
```


Plot that only looks this way for January
duration
1100
1000
900
800
700
0
5
10
15
20
25
30
day_of_month


Plot that tells me nothing about passengers
duration
800
600
400
200
0
1
2
3
4
5
6
passenger_count

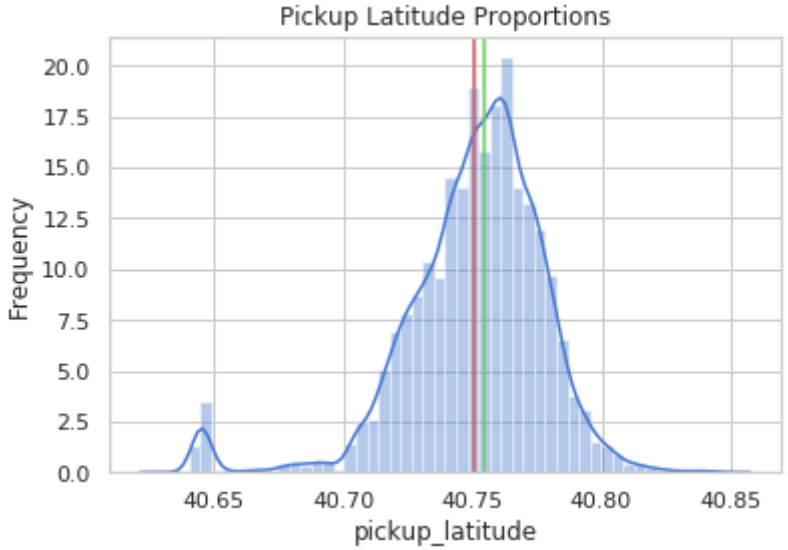
Question 1c

What was surprising in your search for good features?

In [56]:
```
q1c_answer = r"""

The most surprising area in my features stemmed from how many great features we had already accumulated

"""
print(q1c_answer)
# YOUR CODE HERE
#raise NotImplementedError()
```

```
The most surprising area in my features stemmed from how many great features we had already accumulate
d througout the project thus far. All of the cleaning that we did in previous sections and the graphs
that we created were amazing indicators of the directions that we should go in for finding features. I
still find it incredibly fascinating that the overall day of the week provides an immense amount of in
formation about what the duration should be as this seems like such an arbitrary factor. Lastly, I not
iced through some analytics work that the mean of the pickup_latiude, dropoff latitude, etc., were hea
vily influenced by the outliers and that the median location values basically always provided a better
location to replace the outlier with.
```

In [57]:
```
#Why we use the median over the mean for replacing values:
median_pl = np.median(train_df['pickup_latitude'])
mean_pl = np.mean(train_df['pickup_latitude'])

fig, ax = plt.subplots()

sns.distplot(train_df['pickup_latitude'], hist = True)
plt.axvline(mean_pl, color = 'r')
plt.axvline(median_pl, color = 'g')

plt.xlabel('pickup_latitude')
plt.ylabel('Frequency')
plt.title('Pickup Latitude Proportions');
```



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 [58]:
```
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))

def airport_closeness(lat1, lng1, lat2, lng2):
    """Determine if the duration should be high because on way to airport"""
    la_guardia = (40.7769, -73.8740)
    jfk = (40.6413, -73.7781)
    if (lat1 >= 40.60 and lat1 <= 40.80 and lng1 >= -73.95 and lng1 <= -73.80):
        return 1
    if (lat2 >= 40.60 and lat2 <= 40.80 and lng2 >= -73.95 and lng2 <= -73.80):
        return 1
    if (lat1 >= 40.50 and lat1 <= 40.80 and lng1 >= -73.90 and lng1 <= -73.65):
```

```
        return 2
    if (lat2 >= 40.50 and lat2 <= 40.80 and lng2 >= -73.90 and lng2 <= -73.65):
        return 2
    else:
        return 0

# 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 airport_column(df):
    """
    """

    df.is_copy = False # propogate write to original dataframe
    airport_vals = []

    for a, b, c, d in zip(df['pickup_latitude'], df['pickup_longitude'], df['dropoff_latitude'], df['dr
        airport_vals.append(airport_closeness(a, b, c, d))

    df.loc[:, 'airport'] = airport_vals

    return df

def one_hot_encode(df):
    """
    One hot encode on the day of the week as this will help the model know the duration.
    """
    df.is_copy = False
    new_df = pd.get_dummies(df['day_of_week'])
    df = df.merge(new_df, how = 'outer', left_index = True, right_index = True)
    df = df.rename(columns ={0: 'Sunday', 1: 'Monday', 2: 'Tuesday', 3: 'Wednesday',
                   4: 'Thursday', 5: 'Friday', 6: 'Saturday'})
    return df

def clean_lat_long(df):
    """
    Getting rid of all of the values where lat. and long. are 0 in testdf and replacing them with media
    """
    df.is_copy = False
    median_pickup_lat = np.median(df['pickup_latitude'])
    median_pickup_long = np.median(df['pickup_longitude'])
    wrong_pickup = list(df[df['pickup_latitude'] == 0].index)

    median_dropoff_lat = np.median(df['dropoff_latitude'])
    median_dropoff_long = np.median(df['dropoff_longitude'])
    wrong_dropoff = list(df[df['dropoff_latitude'] == 0].index)

    for elem in wrong_pickup:
        df.loc[elem, 'pickup_latitude'] = median_pickup_lat
        df.loc[elem, 'pickup_longitude'] = median_pickup_long
    for elem in wrong_dropoff:
        df.loc[elem, 'dropoff_latitude'] = median_dropoff_lat
```

```
        df.loc[elem, 'dropoff_longitude'] = median_dropoff_long
    return df

def clean_manhattan(df):
    df.is_copy = False
    median_man = np.median(df['manhattan'])
    median_hav = np.median(df['haversine'])
    outliers_ind = list(df[(df['manhattan'] < .75) | (df['manhattan'] > 13)].index)
    for elem in outliers_ind:
        df.loc[elem, 'manhattan'] = median_man
        df.loc[elem, 'haversine'] = median_hav
    return df

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

In [59]:
```
def process_data_fm(data, test=False):
    # Put your final pipeline here
    X = (
        data

        # Transform data
        .pipe(add_time_columns)
        .pipe(add_distance_columns)
        .pipe(airport_column)
        .pipe(clean_lat_long)
        .pipe(clean_manhattan)
        .pipe(one_hot_encode)

        .pipe(select_columns,
              'pickup_longitude',
              'pickup_latitude',
              'dropoff_longitude',
              'dropoff_latitude',
              'manhattan',
              'fare_amount',
              'airport',
              'trip_distance',
              'tolls_amount',
              'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'
             )
    )
    if test:
        y = None
    else:
        y = data['duration']

    return X, y


X_train, y_train = process_data_fm(train_df)
X_val, y_val = process_data_fm(val_df)
# Define your final model here, feel free to try other forms of regression
final_model = lm.LinearRegression(fit_intercept=True)
final_model.fit(X_train, y_train)
# YOUR CODE HERE
#raise NotImplementedError()
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attrib
```

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

Out[59]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [60]:
```
# 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
```

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

Created a CSV file: submission_2018-12-06T00:44:23.csv
You may now upload this CSV file to Kaggle for scoring.
```

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 [55]:
```
# NO TOUCH
```

In [56]:
```
# NOH
```

In [57]:
```
# STAHP
```

In [58]:
```
# NO MOLESTE
```

In [59]:
```
# VA-T'EN
```

In [60]:
```
# NEIN
```

In [61]:
```
# PLSNO
```

In [62]:
```
# THIS SPACE IS NOT YOURS
```

In [63]:
```
# TAWDEETAW
```

In [64]:
```
# MAU LEN
```

In [65]:
```
# ALMOST
```

In [66]:
```
# TO
```

In [67]:
```
# THE
```

In [68]:
```
# END
```

In [69]:
```
# Hmph
```

In [70]:
```
# Good riddance
```

In [71]:
```
generate_submission(test_df, submission_predictions, True)
```

```
Created a CSV file: submission_2018-12-05T08:07:52.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**