Haris Hasan

Abstract

The dataset represents taxi rides and data about those rides. My goal was to run a machine learning model to predict the fare of these rides. The main problems existed in the data available. It was not in a format that could provide meaningful results. To this end, there was quite some preprocessing and cleaning that I had to do before I was able to fit and run the model. However, the final outcome was such that I predicted a list of taxi fare amounts.

Introduction

The overall goal was to create a machine learning model to predict the fare for a taxi ride in New York City given information like pickup date & time, pickup location, drop location and no. of passengers.

This task represents a supervised learning regression problem.

I found this data through Kaggle which provided a dataset that was split into two: one for training and the other for testing. The training data was 5.5 GB in size and had 5.5 million rows available. In contrast, the test set is much smaller and had about 10,000 rows of data.

The training set had the following 8 columns:

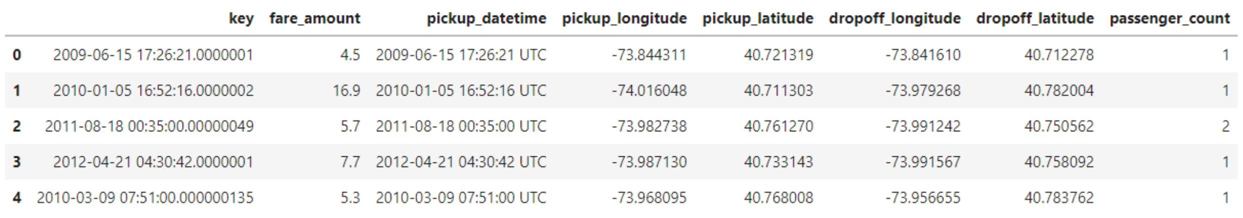
- key (a unique identifier)
- fare_amount (target column)
- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- passenger_count

The test set had all columns except the target column fare_amount which is what I set out to calculate. The problem is very important because appropriate taxi fare rides can determine the success or failure of a business. Ensuring that it is optimally calculated and takes into account everything that is important is itself highly important. The approach I used was to conduct some exploratory data analysis to best set up a final dataset for the machine learning model. The results found that I was indeed able to find accurate fares, placing importance on the distance of the ride, the region the ride took place, and the day and time associated with it.

Data

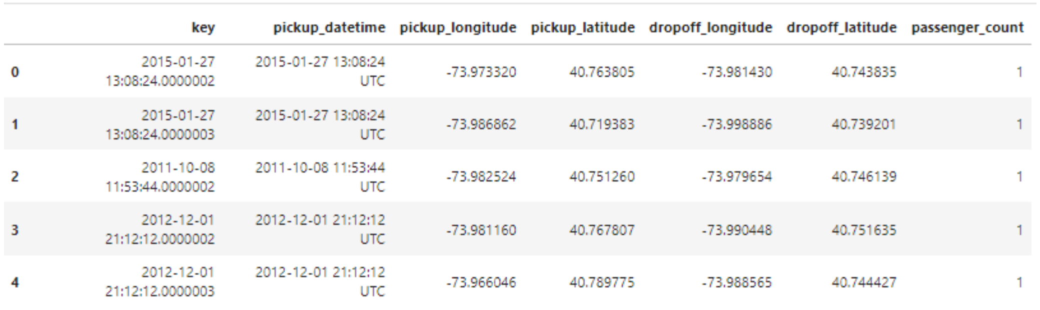
Dataset Link: https://www.kaggle.com/c/new-york-city-taxi-fare-prediction
This dataset is taken from a Kaggle competition organized by Google Cloud. It contains over 55 million rows of training data. We'll attempt to achieve a respectable score in the competition

using just a fraction of the data. Along the way, we'll also look at some practical tips for machine learning. most of the ideas & techniques covered in this notebook are derived from other public notebooks & blog posts.
Here is what the head of the given training dataset looks like:



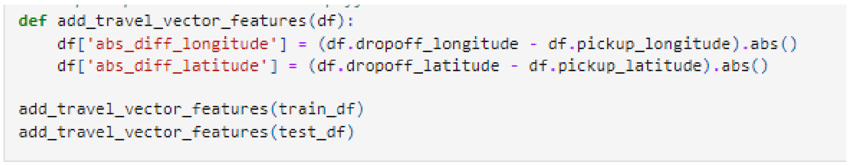
	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1

And here is what the head of the given testing dataset looks like:



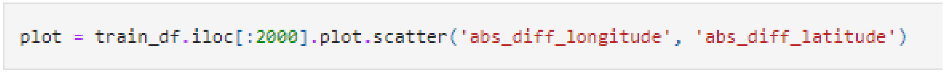
	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24.0000002	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24.0000003	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44.0000002	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12.0000002	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12.0000003	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

Note that all the columns are the same as the training dataset except there is no column for the far amount. I then sought to add two new features 'abs_diff_longitude' and 'abs_diff_latitude' that represent the "Manhattan vector" from the pickup location to the dropoff location.

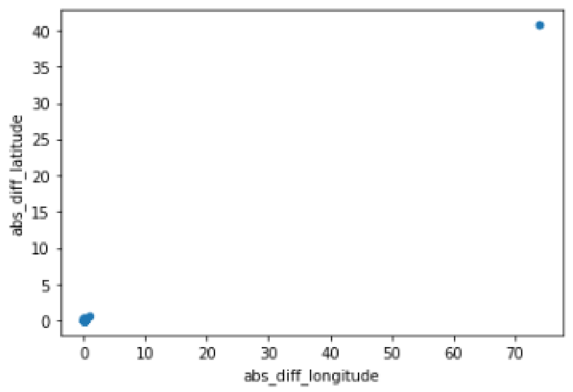


```
def add_travel_vector_features(df):
    df['abs_diff_longitude'] = (df.dropoff_longitude - df.pickup_longitude).abs()
    df['abs_diff_latitude'] = (df.dropoff_latitude - df.pickup_latitude).abs()

add_travel_vector_features(train_df)
add_travel_vector_features(test_df)
```

Our next step is to explore how what this new data may look like by plotting it.



```
plot = train_df.iloc[:2000].plot.scatter('abs_diff_longitude', 'abs_diff_latitude')
```



This demonstrates that since the data is closely grouped together, there are likely few outliers. However, just in case, it may be worth removing negligible values from the training set:

```
print('Old size: %d' % len(train_df))
train_df = train_df[(train_df.abs_diff_longitude < 5.0) & (train_df.abs_diff_latitude < 5.0)]
print('New size: %d' % len(train_df))
```

This gives removes several values which we can see here:

```
Old size: 30000
New size: 29952
```

We can consider that fares may vary not just based on the distance but also based on the day of the week the taxi ride is taken. This is something we can explore by introducing columns that represent the day of the week the ride was taken in a binary fashion.
Note: You will see in the code that I first considered merely dividing it by weekday and weekend but then realized that may not provide as much information as labelling it by each day,
Here is how I achieved this:

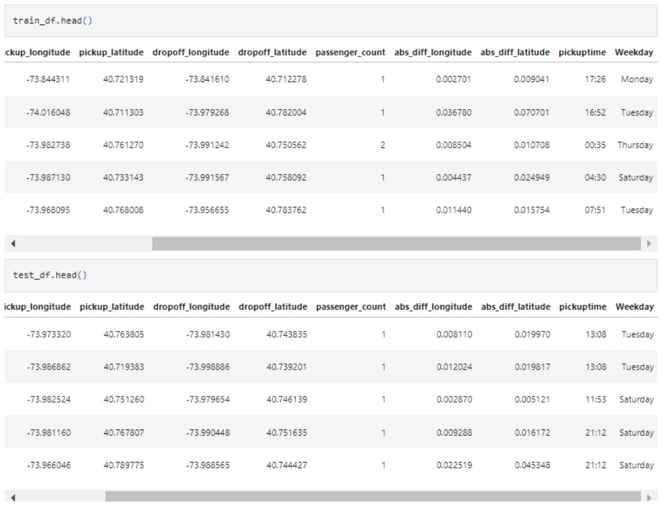
```
def creating_time(df):
    ls1=list(df['pickup_datetime'])
    for i in range(len(ls1)):
        ls1[i]=ls1[i][11:-7:]
    df['pickuptime']=ls1

creating_time(train_df)
creating_time(test_df)
```

```
def creating_weekdays(df):
    ls1=list(df['pickup_datetime'])
    for i in range(len(ls1)):
        ls1[i]=ls1[i][:-4:]
        ls1[i]=pd.Timestamp(ls1[i])
        ls1[i]=ls1[i].weekday()
    df['Weekday']=ls1

creating_weekdays(train_df)
creating_weekdays(test_df)
```

```
train_df.drop('pickup_datetime',inplace=True,axis=1)
test_df.drop('pickup_datetime',inplace=True,axis=1)
```

```
def replace_weekday(df):
    df['Weekday'].replace(to_replace=[i for i in range(0,7)],
                                   value=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'],
                                     inplace=True)

replace_weekday(train_df)
replace_weekday(test_df)
```

At this point, here is what the result of the above was:



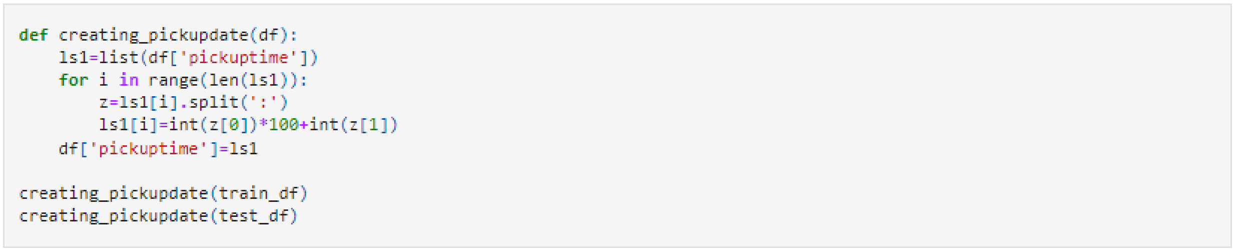
```
train_df.head()
```

ckup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	abs_diff_longitude	abs_diff_latitude	pickuptime	Weekday
-73.844311	40.721319	-73.841610	40.712278	1	0.002701	0.009041	17:26	Monday
-74.016048	40.711303	-73.979268	40.782004	1	0.036780	0.070701	16:52	Tuesday
-73.982738	40.761270	-73.991242	40.750562	2	0.008504	0.010708	00:35	Thursday
-73.987130	40.733143	-73.991567	40.758092	1	0.004437	0.024949	04:30	Saturday
-73.968095	40.768008	-73.956655	40.783762	1	0.011440	0.015754	07:51	Tuesday

```
test_df.head()
```

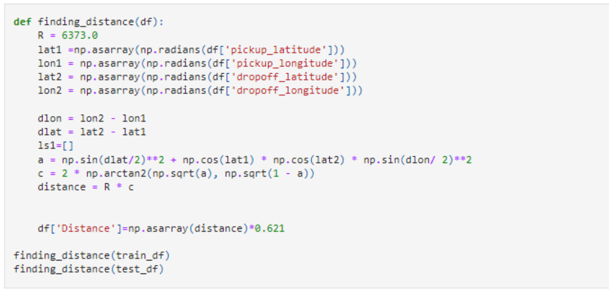
ickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	abs_diff_longitude	abs_diff_latitude	pickuptime	Weekday
-73.973320	40.763805	-73.981430	40.743835	1	0.008110	0.019970	13:08	Tuesday
-73.986862	40.719383	-73.998886	40.739201	1	0.012024	0.019817	13:08	Tuesday
-73.982524	40.751260	-73.979654	40.746139	1	0.002870	0.005121	11:53	Saturday
-73.981160	40.767807	-73.990448	40.751635	1	0.009288	0.016172	21:12	Saturday
-73.966046	40.789775	-73.988565	40.744427	1	0.022519	0.045348	21:12	Saturday

Then, I also chose to drop the colon in the 'pickuptime' column to ensure the model would be able to work in a simpler way, while still fully taking into account the meaning behind it.

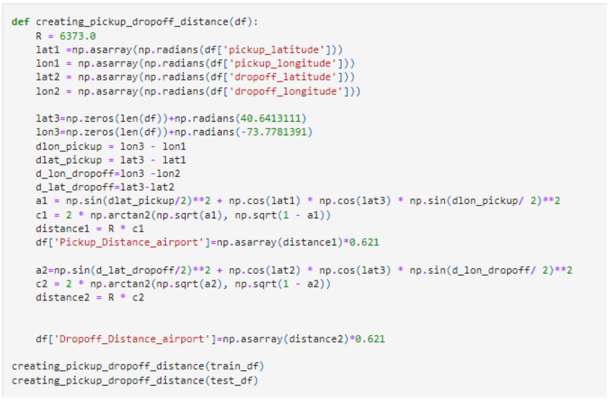


```python
def creating_pickupdate(df):
    ls1=list(df['pickuptime'])
    for i in range(len(ls1)):
        z=ls1[i].split(':')
        ls1[i]=int(z[0])*100+int(z[1])
    df['pickuptime']=ls1

creating_pickupdate(train_df)
creating_pickupdate(test_df)
```

I then chose to create new columns that represented the total distance travelled, the pickup distance from the airport, and the dropoff distance from the airport:

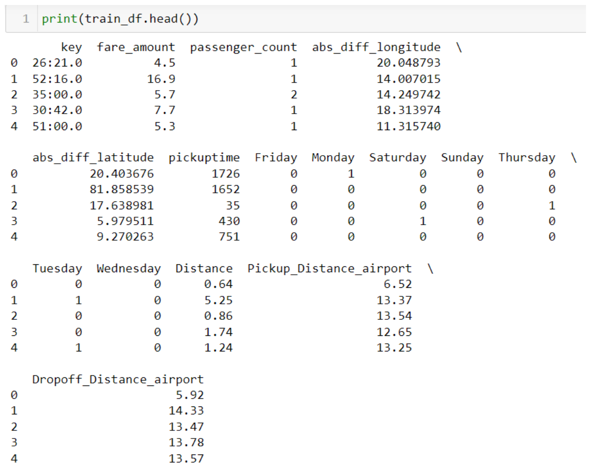


```python
def finding_distance(df):
    R = 6373.0
    lat1 =np.asarray(np.radians(df['pickup_latitude']))
    lon1 = np.asarray(np.radians(df['pickup_longitude']))
    lat2 = np.asarray(np.radians(df['dropoff_latitude']))
    lon2 = np.asarray(np.radians(df['dropoff_longitude']))

    dlon = lon2 - lon1
    dlat = lat2 - lat1
    ls1=[]
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/ 2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    distance = R * c


    df['Distance']=np.asarray(distance)*0.621

finding_distance(train_df)
finding_distance(test_df)
```



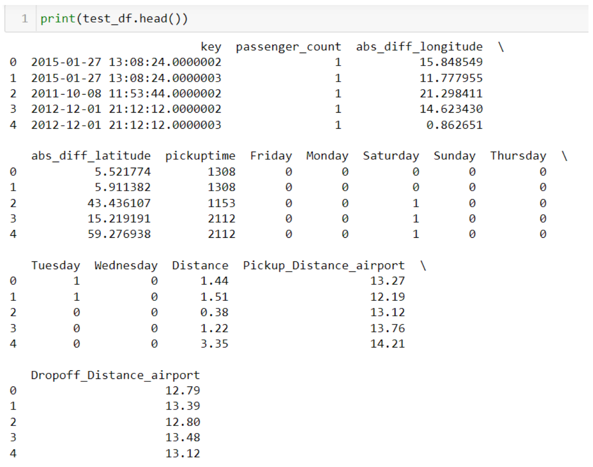
```python
def creating_pickup_dropoff_distance(df):
    R = 6373.0
    lat1 =np.asarray(np.radians(df['pickup_latitude']))
    lon1 = np.asarray(np.radians(df['pickup_longitude']))
    lat2 = np.asarray(np.radians(df['dropoff_latitude']))
    lon2 = np.asarray(np.radians(df['dropoff_longitude']))

    lat3=np.zeros(len(df))+np.radians(40.6413111)
    lon3=np.zeros(len(df))+np.radians(-73.7781391)
    dlon_pickup = lon3 - lon1
    dlat_pickup = lat3 - lat1
    d_lon_dropoff=lon3 -lon2
    d_lat_dropoff=lat3-lat2
    a1 = np.sin(dlat_pickup/2)**2 + np.cos(lat1) * np.cos(lat3) * np.sin(dlon_pickup/ 2)**2
    c1 = 2 * np.arctan2(np.sqrt(a1), np.sqrt(1 - a1))
    distance1 = R * c1
    df['Pickup_Distance_airport']=np.asarray(distance1)*0.621

    a2=np.sin(d_lat_dropoff/2)**2 + np.cos(lat2) * np.cos(lat3) * np.sin(d_lon_dropoff/ 2)**2
    c2 = 2 * np.arctan2(np.sqrt(a2), np.sqrt(1 - a2))
    distance2 = R * c2


    df['Dropoff_Distance_airport']=np.asarray(distance2)*0.621

creating_pickup_dropoff_distance(train_df)
creating_pickup_dropoff_distance(test_df)
```

Some final cleaning up of the data including rounding of the numbers and keeping only the relevant distance columns was achieved and this was the final head of the test and train datasets respectively:



```python
print(train_df.head())
```

```
       key  fare_amount  passenger_count  abs_diff_longitude  \
0  26:21.0          4.5                1           20.048793
1  52:16.0         16.9                1           14.007015
2  35:00.0          5.7                2           14.249742
3  30:42.0          7.7                1           18.313974
4  51:00.0          5.3                1           11.315740

   abs_diff_latitude  pickuptime  Friday  Monday  Saturday  Sunday  Thursday  \
0          20.403676        1726       0       1         0       0         0
1          81.858539        1652       0       0         0       0         0
2          17.638981          35       0       0         0       0         1
3           5.979511         430       0       0         1       0         0
4           9.270263         751       0       0         0       0         0

   Tuesday  Wednesday  Distance  Pickup_Distance_airport  \
0        0          0      0.64                     6.52
1        1          0      5.25                    13.37
2        0          0      0.86                    13.54
3        0          0      1.74                    12.65
4        1          0      1.24                    13.25

   Dropoff_Distance_airport
0                      5.92
1                     14.33
2                     13.47
3                     13.78
4                     13.57
```



```python
print(test_df.head())
```

```
                           key  passenger_count  abs_diff_longitude  \
0  2015-01-27 13:08:24.0000002                1           15.848549
1  2015-01-27 13:08:24.0000003                1           11.777955
2  2011-10-08 11:53:44.0000002                1           21.298411
3  2012-12-01 21:12:12.0000002                1           14.623430
4  2012-12-01 21:12:12.0000003                1            0.862651

   abs_diff_latitude  pickuptime  Friday  Monday  Saturday  Sunday  Thursday  \
0           5.521774        1308       0       0         0       0         0
1           5.911382        1308       0       0         0       0         0
2          43.436107        1153       0       0         1       0         0
3          15.219191        2112       0       0         1       0         0
4          59.276938        2112       0       0         1       0         0

   Tuesday  Wednesday  Distance  Pickup_Distance_airport  \
0        1          0      1.44                    13.27
1        1          0      1.51                    12.19
2        0          0      0.38                    13.12
3        0          0      1.22                    13.76
4        0          0      3.35                    14.21

   Dropoff_Distance_airport
0                     12.79
1                     13.39
2                     12.80
3                     13.48
4                     13.12
```

The above ensures that we have completed the appropriate preprocessing and cleaning of the data.

Problems and Methods

Using the same dataset for both training and testing leaves room for miscalculations, thus increasing the chances of inaccurate predictions. Therefore, using the train_test_split function allowed me to break a dataset with ease while pursuing an ideal model.
The ideal split is said to be 80:20 for training and testing which is what I chose to use. This does well to ensure the samples of the dataset are shuffled randomly and then split into the training and test sets according to the size you defined.
Now that we had our attributes and labels, the next step was to split this data into training and test sets which I did by using Scikit-Learn's built-in train_test_split() method:

```
from sklearn.model_selection import train_test_split
X=train_df.drop(['key','fare_amount'],axis=1)
y=train_df['fare_amount']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=80)
```

The above script splits 80% of the data into a training set while 20% of the data into the test set.

I implemented the linear regression model with Scikit-Learn by importing the LinearRegression class, instantiating it, and calling the fit() method along with our training data. I chose this method as it is able to call on the machine learning library to train on the data in a straightforward manner. The linear regression model served well to find the best value for the intercept and slope, which results in a line that best fits the data.

At this point, I trained my algorithm by executing the following command:

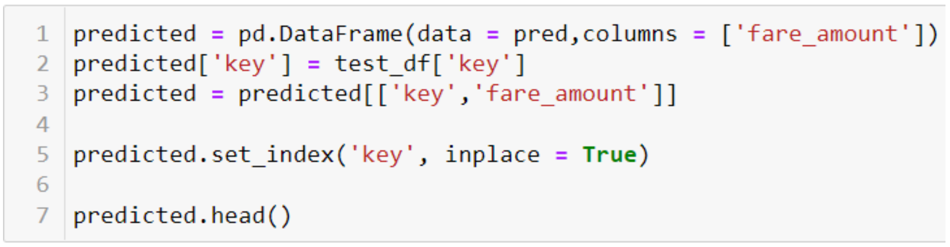
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression(normalize=True)
lr.fit(X_train,y_train)
print(lr.score(X_test,y_test))
```

```
0.598669846466314
```

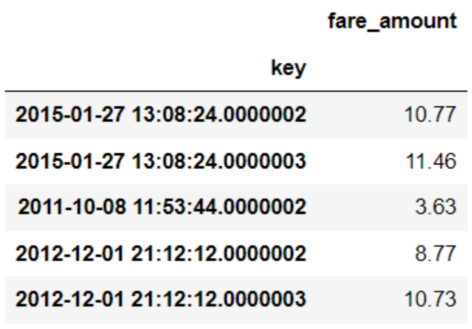
Predicting a response from our model:

```
pred=np.round(lr.predict(test_df.drop('key',axis=1)),2)
print(pred)
```

```
[10.77 11.46  3.63 ... 34.39 14.06  4.99]
```



```python
predicted = pd.DataFrame(data = pred,columns = ['fare_amount'])
predicted['key'] = test_df['key']
predicted = predicted[['key','fare_amount']]

predicted.set_index('key', inplace = True)

predicted.head()
```



	fare_amount
key	
2015-01-27 13:08:24.0000002	10.77
2015-01-27 13:08:24.0000003	11.46
2011-10-08 11:53:44.0000002	3.63
2012-12-01 21:12:12.0000002	8.77
2012-12-01 21:12:12.0000003	10.73

I chose these methods for their simplicity. The modelling and prediction itself were not nearly as complex as the data cleaning and preprocessing - which is something I was intrigued to realize. I found that the main problem simply lies in presenting the data correctly to the model itself. To tackle this, I did my best to simplify the final dataset with information that was highly relevant and necessary.

Results and Discussion

I chose to look toward Kaggle and consider the best-performing submissions after I had run my model. Through their interface, one can view projects that had the best accuracy and were best rated. I then took the resulting fare amounts from the best-performing submission and chose to test my accuracy on that. The resulting accuracy was 82% calculated by considering the means squares.

I provided in-depth details of my exploration in my data section (kindly refer to that section and the information I presented when marking this section) as the processing and cleaning was perhaps the crux of my project. The sub-analysis that I chose to perform as I did those were able to simplify the dataset for the model to run.

The overall results presented that I was perhaps correct to assume that a fairly accurate estimate of the taxi fare could be derived from mainly considering just the distance of the ride, the region the ride took place, and the day and time associated with it.

Conclusion and Notes

Overall, I was proud of this result considering that the model results I compared mine to were far more advanced than my own. It was interesting to then see what they had done and see where areas of improvement lay for me. I found that they had utilized advanced visualization tools to cluster rides and consider these deeply before further cleaning the data in the dataset. This is something that I would like to look into next time as well.

I learned that exploratory data analysis is perhaps even more important than the model itself. Through such, I realized what was important and what wasn't. In addition, I learned that by focusing on a few key variables, a near-accurate result can be obtained. Nevertheless, one must also take into account hidden variables and nuances to obtain a fully accurate result. In order to improve my performance next time, I hope to improve my visualization skills. I am interested in learning Plotly specifically.