Capstone Project

NYC Taxi Trip Time Prediction

Name- Pranav Singhal

Batch-AlmaBetter Pro

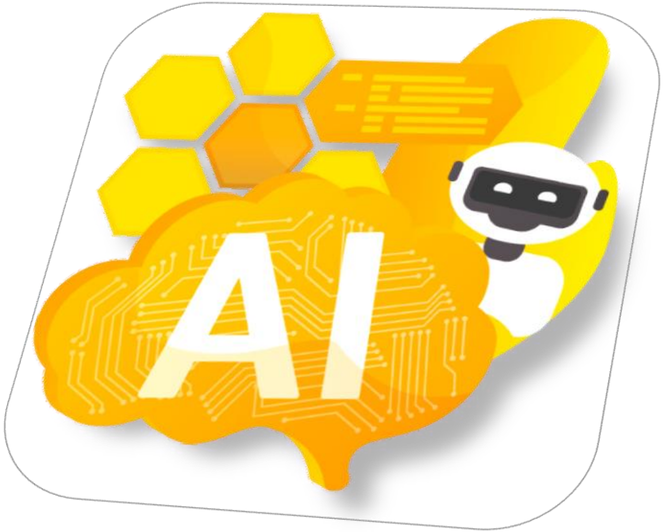
Github- https://github.com/pranav168/Credit-Card-Default-Prediction

Why NYC Taxi Trip Time Prediction?

Trip time prediction is an important problem. Taxi passengers often want to know when they will arrive at their destinations. We design a method of predicting taxi trip time by finding historical similar trips. Trips are clustered based on origin, destination, and start time. Then similar trips are mapped to road networks to find frequent sub-trajectories that are used to model travel time of the various parts of the routes. Experimental results show this method is effective.

Predicting Taxi Trip Time with Machine Learning

- Machine learning has been of significant help as it has helped businesses in abundant ways. ML is a subset of AI and does not need to be directly trained like AI to perform tasks. ML is used for prediction analysis in businesses, which we will learn in this case study. ML Systems created a solution that can forecast time-based on initial partial trajectories. For someone in the logistics business, this is indispensable. It is important to predict how long a driver will have his taxi occupied. If a dispatcher got estimates about the taxi driver's current ride time, they could better recognize which driver to allocate for each pickup request.



The success of a machine learning model

The success of a machine learning model, however, does not depend solely on the selection of a machine learning method. Key factors contributing to the success of the machine learning model include:

- **Data**

Data is the very prerequisite for any successful machine learning model. No matter how great your machine learning models are, you cannot get a reliable high-performance model from the prediction model without a sufficient amount of rich data.

- **Feature Engineering**

Processing raw data and making it a suitable input for the machine learning models includes **data cleaning, creating new features, and feature selection**. Feature engineering usually is the most time-consuming machine learning problem, especially when it comes to building prediction models for structured data.

- **Models**

Even though there are many machine learning methods available for certain machine learning problems, such as binary classification, for example, each method has its own strengths and weaknesses. Based on our demands and requirements, we may need to choose different methods.

- **Performance Metrics**

Given two machine learning methods, how do we evaluate them to select the better one?
We need well-designed performance metrics based on our dataset and experience. For example, r2 score and root mean square error.

Problem Description

My task is to build a model that predicts the total ride duration of taxi trips in New York City. primary dataset is one released by the NYC Taxi and Limousine Commission, which includes pickup time, geo-coordinates, number of passengers, and several other variables.

Objective

Predicting total ride duration of taxi trips in New York City

Data Description

The dataset is based on the 2016 NYC Yellow Cab trip record data made available in Big Query on Google Cloud Platform. The data was originally published by the NYC Taxi and Limousine Commission (TLC). The data was sampled and cleaned for the purposes of this project. Based on individual trip attributes, you should predict the duration of each trip in the test set.

NYC Taxi Data.csv - the training set (contains 1458644 trip records)

Data fields

id - a unique identifier for each trip

vendor_id - a code indicating the provider associated with the trip record

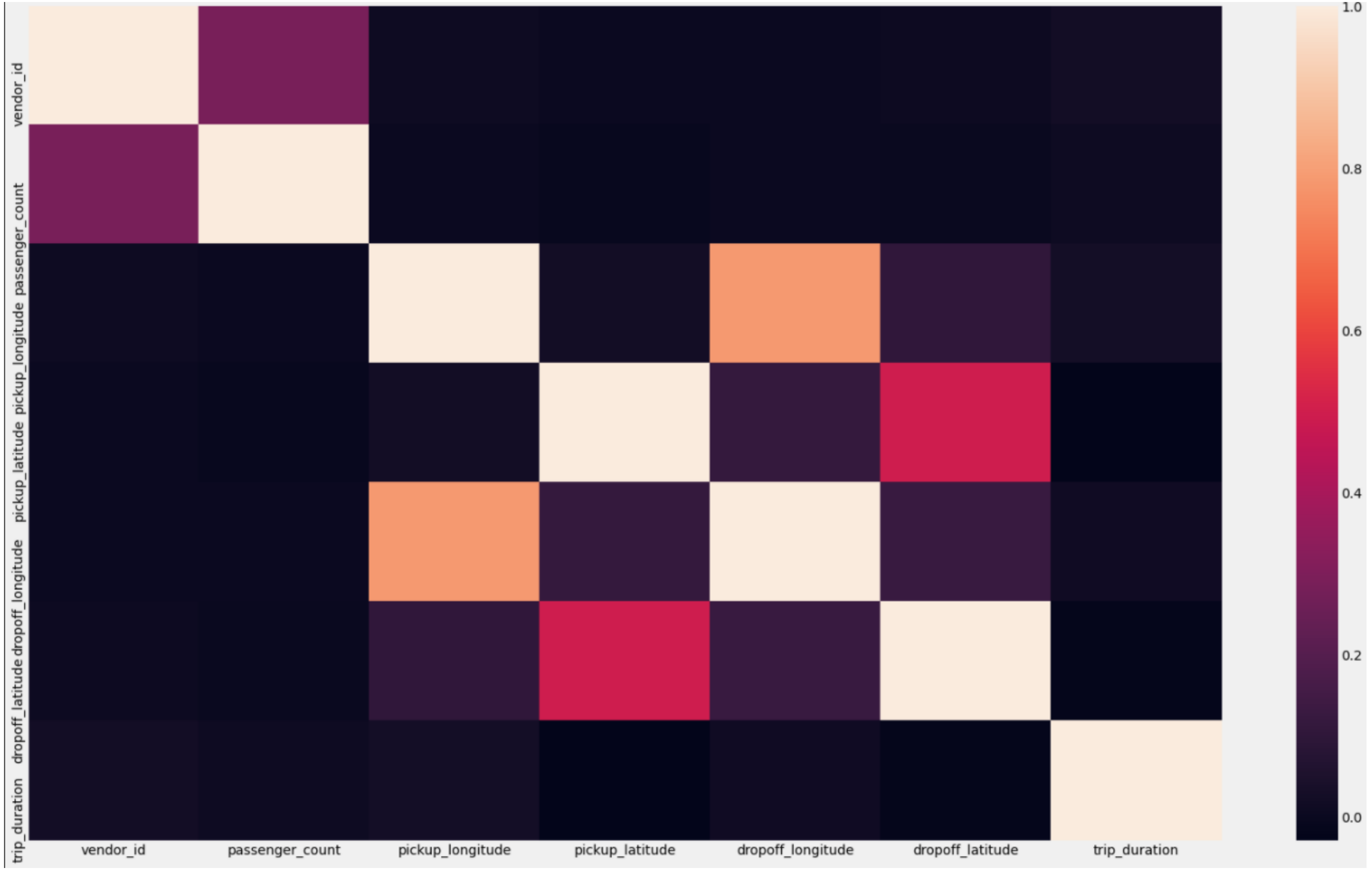
- pickup_datetime - date and time when the meter was engaged
- dropoff_datetime - date and time when the meter was disengaged
- passenger_count - the number of passengers in the vehicle (driver entered value)
- pickup_longitude - the longitude where the meter was engaged
- pickup_latitude - the latitude where the meter was engaged
- dropoff_longitude - the longitude where the meter was disengaged
- dropoff_latitude - the latitude where the meter was disengaged
- store_and_fwd_flag - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
- trip_duration - duration of the trip in seconds

EDA(Count Dataset)

	id	id_count	vendor_id	vendor_id_count	pickup_datetime	pickup_datetime_count	dropoff_datetime	dropoff_datetime_count	passenger_count	passenger_count_count	pickup_longitude	pickup_longitude_count	pickup_latitude	pickup_latitude_count
0	id0290975	1	2.000000	780302.000000	2016-01-12 18:48:44	5	2016-05-16 19:40:28	5	1	1033540	-73.982201	633	40.774101	414
1	id2060444	1	1.000000	678342.000000	2016-05-07 13:18:07	5	2016-02-19 19:25:04	5	2	210318	-73.982140	607	40.774090	411
2	id3875012	1			2016-04-05 18:55:21	5	2016-02-28 02:41:12	4	5	78088	-73.982101	587	40.774120	410
3	id3654481	1			2016-06-10 23:17:17	5	2016-03-04 19:33:28	4	3	59896	-73.982117	585	40.774109	392
4	id1342396	1			2016-03-04 08:07:34	5	2016-02-07 15:35:02	4	6	48333	-73.982224	584	40.774078	390
5	id0424790	1			2016-02-09 21:03:38	5	2016-05-03 18:27:19	4	4	28404	-73.982094	575	40.774052	376
6	id1573751	1			2016-06-15 22:28:10	4	2016-03-30 22:12:02	4	0	60	-73.982246	558	40.774132	356
7	id0113927	1			2016-06-29 13:13:50	4	2016-04-10 20:01:29	4	7	3	-73.982208	551	40.774139	352
8	id2863653	1			2016-03-19 22:28:48	4	2016-03-03 20:20:32	4	8	1	-73.982307	546	40.774071	347
9	id1964522	1			2016-06-09 19:59:43	4	2016-06-05 14:10:03	4	9	1	-73.982239	545	40.774158	335

dropoff_longitude	dropoff_longitude_count	dropoff_latitude	dropoff_latitude_count	store_and_fwd_flag	store_and_fwd_flag_count	trip_duration	trip_duration_count
-73.982330	443	40.774311	269	N	1450599.000000	368	1624
-73.982094	433	40.774330	263	Y	8045.000000	408	1584
-73.982246	430	40.750149	259			348	1582
-73.982117	427	40.750118	253			367	1581
-73.991379	420	40.750198	250			358	1577
-73.982201	419	40.750172	247			399	1573
-73.982269	414	40.774319	245			418	1570
-73.991402	406	40.774342	244			417	1568
-73.982384	405	40.750038	242			388	1567
-73.982307	403	40.750111	242			344	1561

This Dataset set gives count of each and every Variable in the Original Dataset



Correlation Chart

Correlation is not at high for between any of my dependent variable with the independent variable, also the number of features are so less that we could not make a model using these only, so now we will be going a lot of feature engineering

Feature Engineering

```
df.describe()
```

```
#their are few cases with passanger count 0, lets explore it
#also minimum drip duretion is 1 sec which show an anomaly, lets remove them first
# Minimum pickup and dropoff longitude are really low than mean
```

	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
count	1.458644e+06	1.458644e+06	1.458644e+06	1.458644e+06	1.458644e+06	1.458644e+06	1.458644e+06
mean	1.534950e+00	1.664530e+00	-7.397349e+01	4.075092e+01	-7.397342e+01	4.075180e+01	9.594923e+02
std	4.987772e-01	1.314242e+00	7.090186e-02	3.288119e-02	7.064327e-02	3.589056e-02	5.237432e+03
min	1.000000e+00	0.000000e+00	-1.219333e+02	3.435970e+01	-1.219333e+02	3.218114e+01	1.000000e+00
25%	1.000000e+00	1.000000e+00	-7.399187e+01	4.073735e+01	-7.399133e+01	4.073588e+01	3.970000e+02
50%	2.000000e+00	1.000000e+00	-7.398174e+01	4.075410e+01	-7.397975e+01	4.075452e+01	6.620000e+02
75%	2.000000e+00	2.000000e+00	-7.396733e+01	4.076836e+01	-7.396301e+01	4.076981e+01	1.075000e+03
max	2.000000e+00	9.000000e+00	-6.133553e+01	5.188108e+01	-6.133553e+01	4.392103e+01	3.526282e+06

- Their are few cases with passanger count 0, lets explore it
- Also minimum drip duretion is 1 sec which show an anomaly, lets remove them first
- Minimum pickup and dropoff longitude are really low than mean

```python
print(np.percentile(df.trip_duration,0.1),
np.percentile(df.trip_duration,0.5),
np.percentile(df.trip_duration,1.5),
np.percentile(df.trip_duration,2),
np.percentile(df.trip_duration,2.5),
np.percentile(df.trip_duration,3),
np.percentile(df.trip_duration,3.5))
```

```
7.0 51.0 107.0 122.0 135.0 146.0 156.0
```

```python
print(np.percentile(df.trip_duration,98.5),
np.percentile(df.trip_duration,99),
np.percentile(df.trip_duration,99.5),
np.percentile(df.trip_duration,99.9))
```

```
3072.0 3440.0 4139.0 85127.41700000013
```

```python
df=df[(df.trip_duration>=107) & (df.trip_duration<=4139)]
```

```python
print(np.percentile(df.pickup_longitude,0.1),
      np.percentile(df.pickup_longitude,0.05),
      np.percentile(df.pickup_longitude,0.01),
      np.percentile(df.pickup_longitude,0.001),
      np.percentile(df.pickup_longitude,0.0001))
df=df[df.pickup_longitude>-74.017]
print(np.percentile(df.dropoff_longitude,0.1),
      np.percentile(df.dropoff_longitude,0.05),
      np.percentile(df.dropoff_longitude,0.01),
      np.percentile(df.dropoff_longitude,0.001),
      np.percentile(df.dropoff_longitude,0.0001))
df=df[df.dropoff_longitude>=-74.467]
```

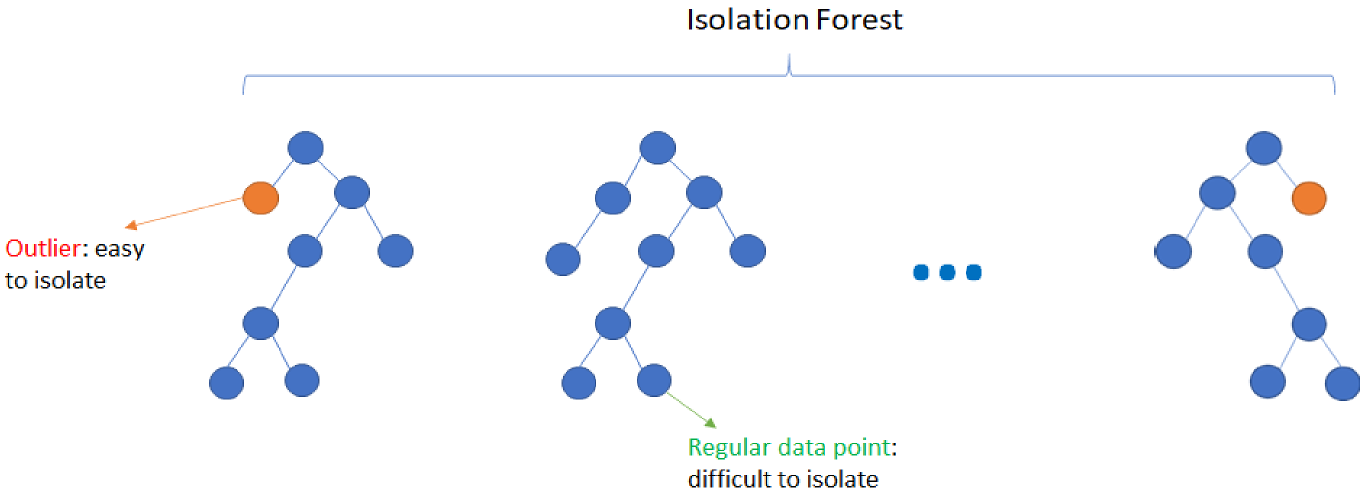
Using Percentile's to get the values fitting in the distribution correctly.

Here,

- After 1.5 percentile, value starts to get saturated thus anything below 107 sec is mostly a outlier, now lets look for maximum value as well.
- After 99.5 Percentile the value starts to get increase suddenly thus anything above 4139 sec is mostly a outlier, lets remove these values

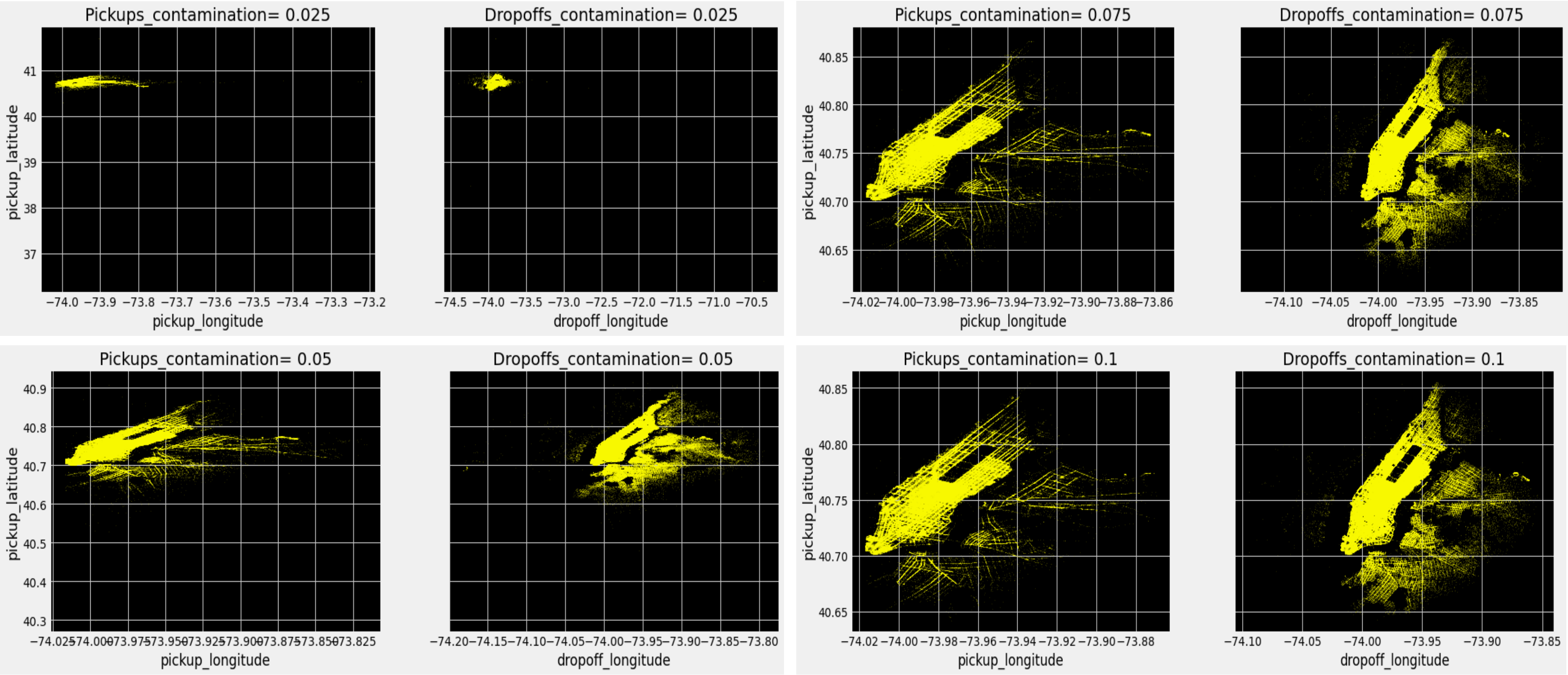
Similarly I did it for Longitude and Latitude

Outlier Detection with Isolation Forest



Lets do a Anomaly detection now, we will be doing a multi-variate detection, using features like Latitude, longitude for both pickup and drop-off as well as time duration and distance in km. But contamination determination is really important as we want to loose much data, thus we I will plot the data using a scatter plot and choose best contamination value.

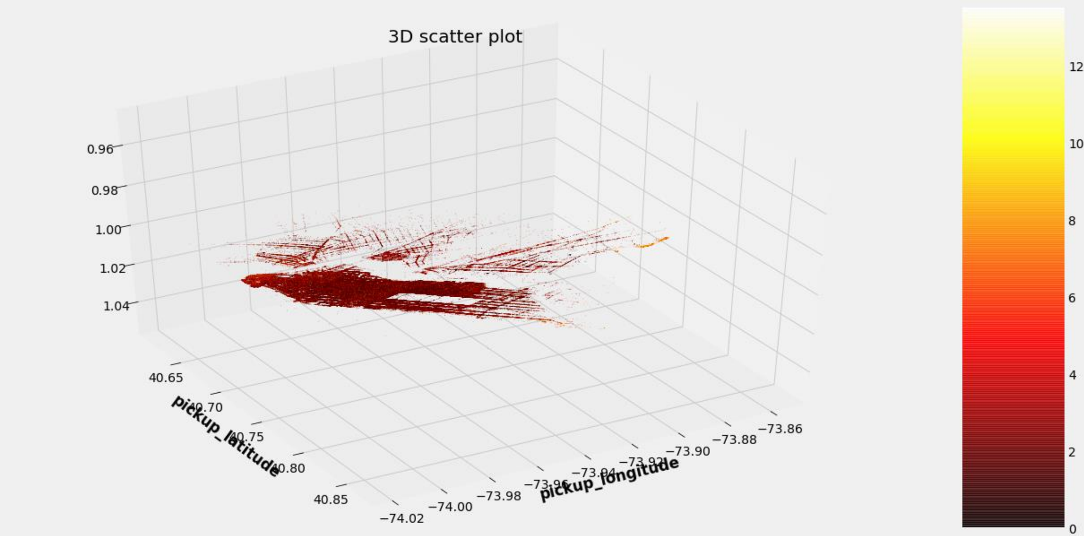
Contamination determination for Isolation Forest

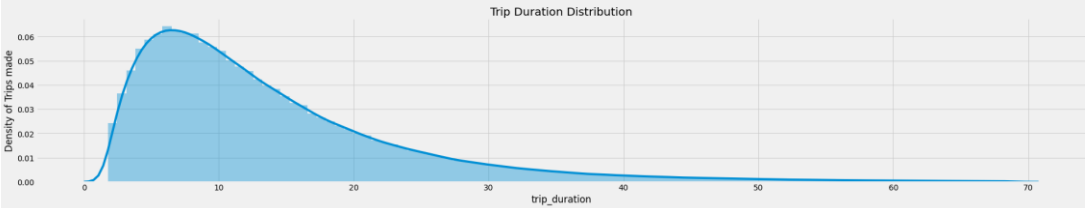


Contamination = 0.075 looks the best, without loosing on data

```python
'''Thus contamination=0.075 works best as we want to retain maximum amount of data'''
minmax = MinMaxScaler(feature_range=(0, 1))                                        #Using
X = minmax.fit_transform(df[['trip_duration','haversine distance (km)',
                             'pickup_longitude','pickup_latitude',
                             'dropoff_longitude','dropoff_latitude']])
clf = IsolationForest(n_estimators=100, contamination=0.075, random_state=0)
clf.fit(X)
df['multivariate_outlier'] = clf.predict(X)                                        # pred
df=df[df.multivariate_outlier==1]
df.drop('multivariate_outlier',axis=1,inplace=True)
```

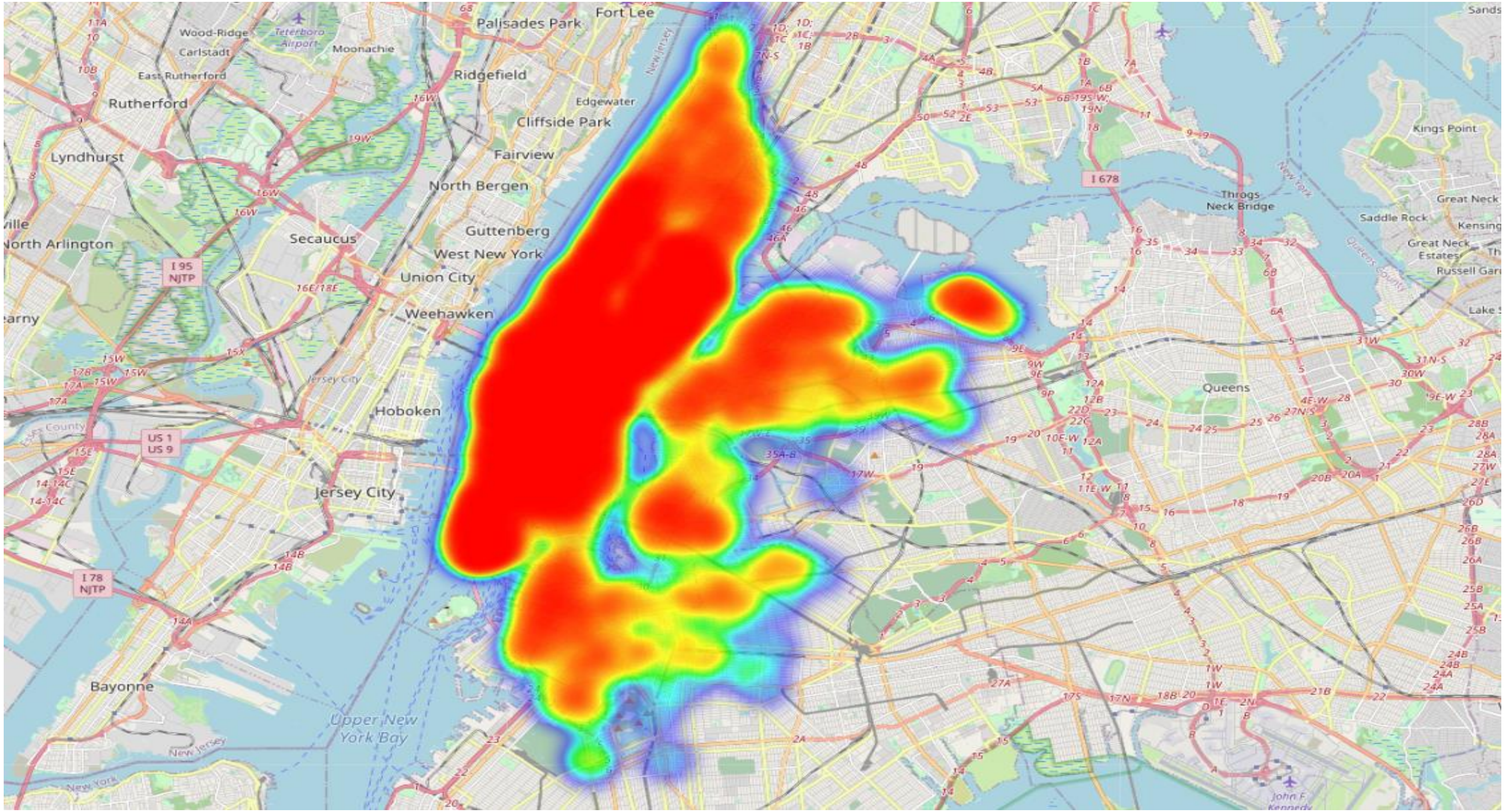
```python
def plot_scatter(df):
  fig = plt.figure(figsize = (20, 10))
  ax = plt.axes(projection ="3d")
  # Creating color map
  z=range(1,len(df['pickup_longitude'])+1)
  Z=[]
  count=0
  my_cmap = plt.get_cmap('hot')
  plot=ax.scatter3D(df['pickup_longitude'],df['pickup_latitude'],
                  1,cmap = my_cmap,alpha = 0.8,
                  c =df['haversine distance (km)'],s=0.02)
  plt.title("3D scatter plot")
  ax.set_xlabel('pickup_longitude', fontweight ='bold')
  ax.set_ylabel('pickup_latitude', fontweight ='bold')
  fig.colorbar(plot, ax = ax, aspect = 5)
  ax.view_init(-140, -60)
  plt.show()

plot_scatter(df)
```





Implementing Isolation Forest & Plotting 3D scatter plot and distribution plot for trip durationof the final data , it looks a lot better now .

Plotting Heatmap using Folium

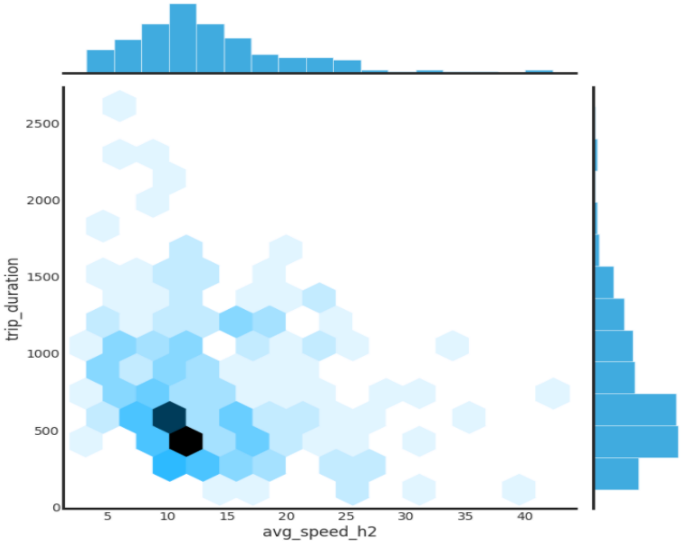


Creating New Features

```
#adding another column with distance metric calculted using lat and long
df['haversine distance (km)'] = df.apply(lambda x: float(hs.haversine((x['pickup_latitude'],x['pickup_longitude']),(x['dropoff_latitude'], x['dropoff_longitude']))),axis=1)
df.shape
```

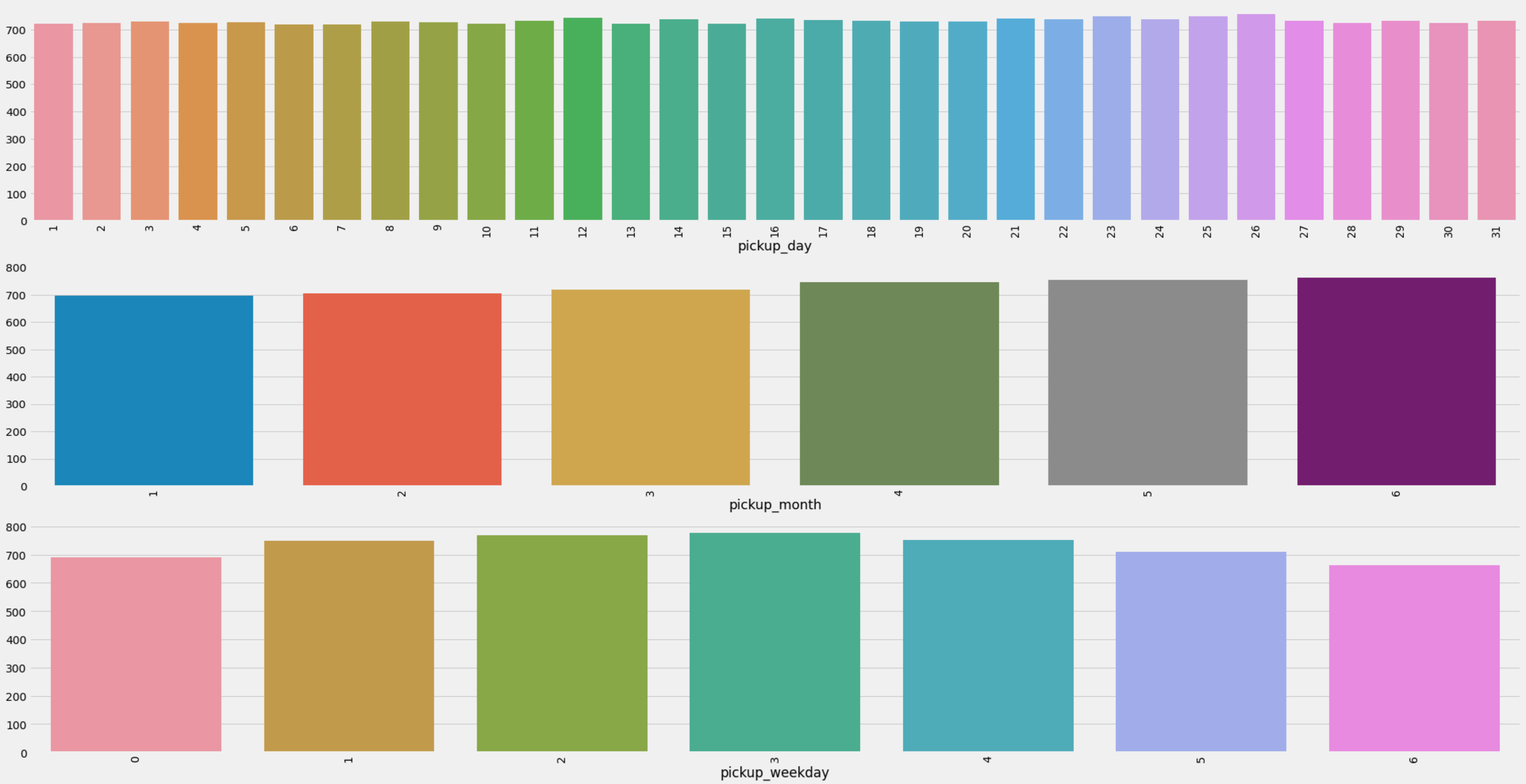
```
list=['pickup_','dropoff_']
for i in list:
  df[i+'date'] = df[i+'datetime'].dt.date
  df[i+'day'] = df[i+'datetime'].dt.day
  df[i+'month'] = df[i+'datetime'].dt.month
  df[i+'weekday'] = df[i+'datetime'].dt.weekday
  df[i+'weekofyear'] = df[i+'datetime'].dt.weekofyear
  df[i+'time'] = df[i+'datetime'].dt.hour
```

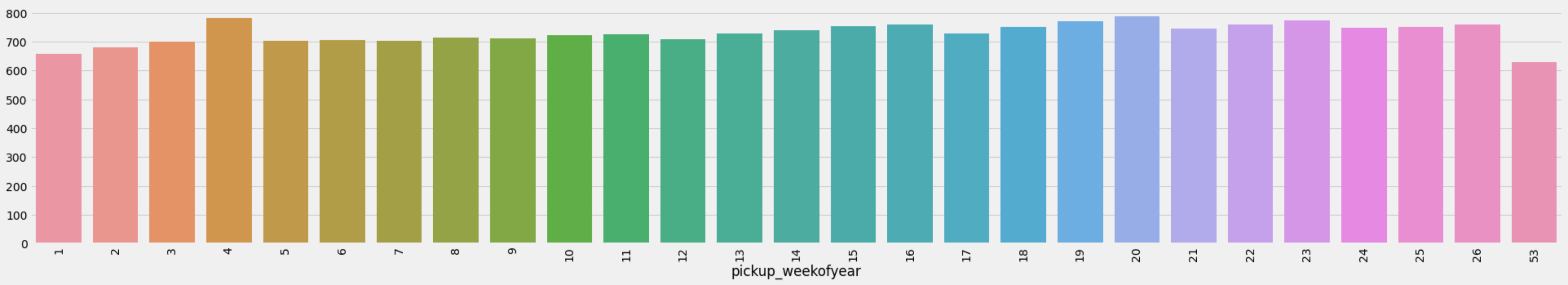
```
df['avg_speed_h2']=df['haversine distance (km)']*3600  / df['trip_duration']    #converting sec to hours
with sns.axes_style('white'):
    sns.jointplot('avg_speed_h2', "trip_duration", df[:200], kind='hex',height=10)
avg_speed_vs_hr=df.groupby('pickup_time')['avg_speed_h2'].mean()
avg_speed_vs_week=df.groupby('pickup_weekday')['avg_speed_h2'].mean()
avg_speed_vs_weekofyear=df.groupby('pickup_weekofyear')['avg_speed_h2'].mean()
avg_speed_vs_month=df.groupby('pickup_month')['avg_speed_h2'].mean()
avg_speed_vs_day=df.groupby('pickup_day')['avg_speed_h2'].mean()
avg_speed_vs_date=df.groupby('pickup_date')['avg_speed_h2'].mean()
dict_hr=dict(zip(avg_speed_vs_hr.index,avg_speed_vs_hr.values))
dict_week=dict(zip(avg_speed_vs_week.index,avg_speed_vs_week.values))
dict_weekofyear=dict(zip(avg_speed_vs_weekofyear.index,avg_speed_vs_weekofyear.values))
dict_month=dict(zip(avg_speed_vs_month.index,avg_speed_vs_month.values))
dict_day=dict(zip(avg_speed_vs_day.index,avg_speed_vs_day.values))
```

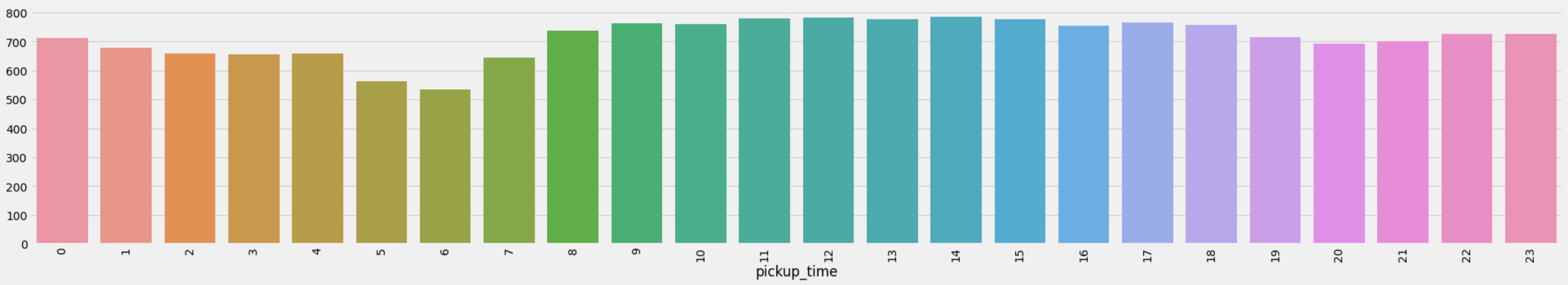


While calculating time, most useful features are Distance and Avg speed at particular time interval, also I did not wanted my model to get biased so I removed date from being a parameter of avg speed.
This plot shows the relationship vs avg speed and trip duration, conveying shorter duration trip have high low avg speed.

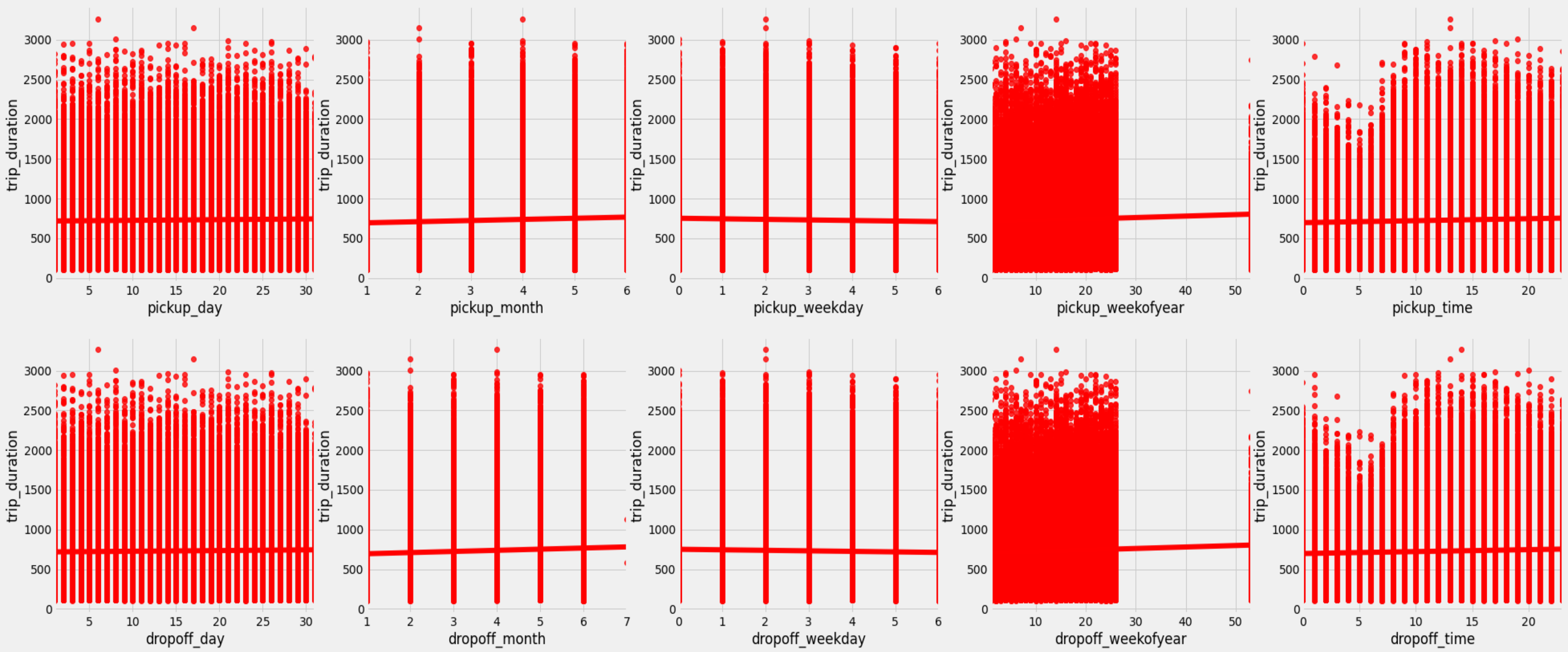
Plots between Time duration and different time formats for pickup and drop offs



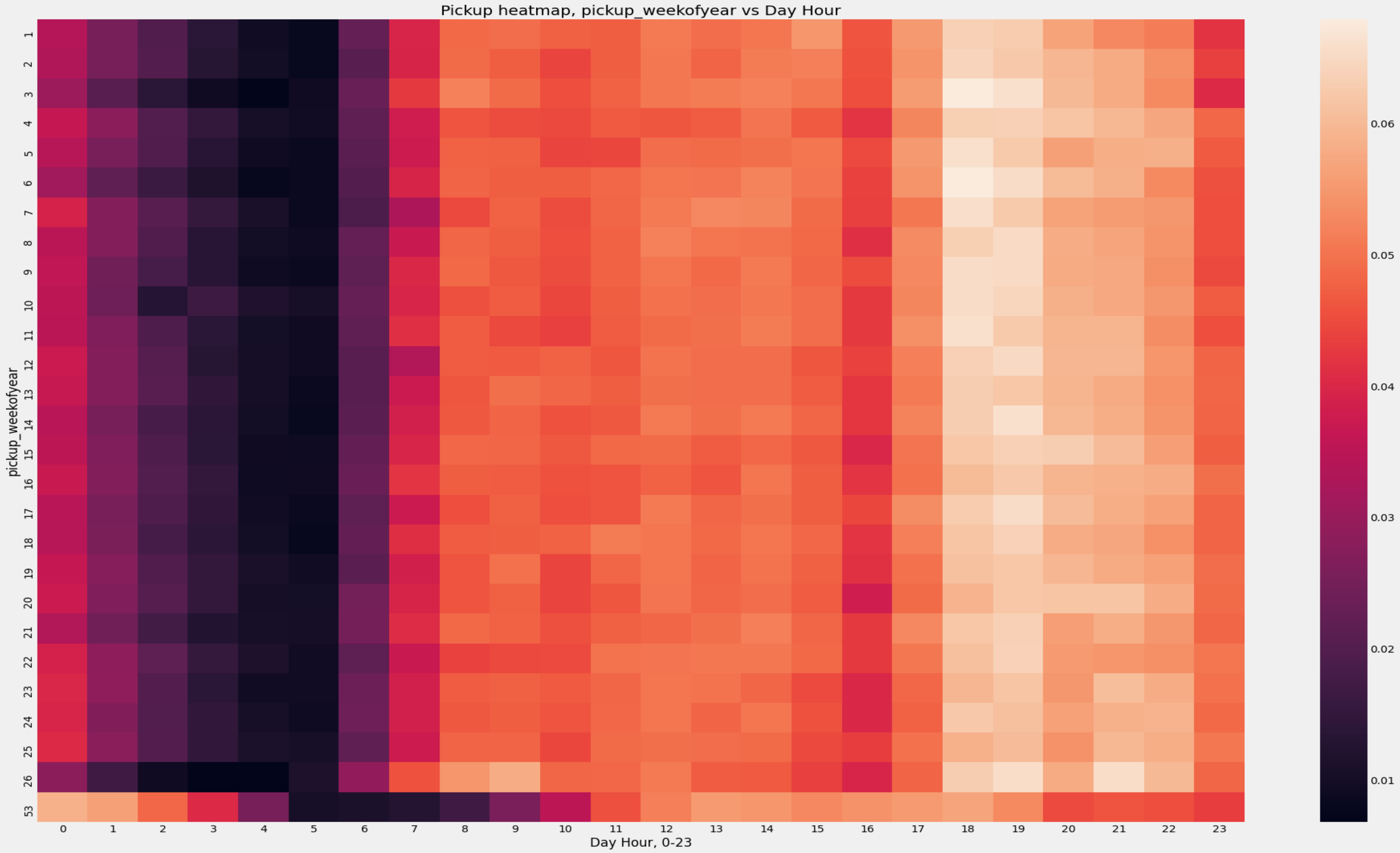




Out of all a prominent pattern is visible for pickup hour on the day only.



- Again Reg Plot tells a similar story, also it represents a linear relation between trip_duration and all these time formats.



This Plot tells the complete history about the data

```
df['lat_diff']=df.dropoff_latitude-df.pickup_latitude
df['long_diff']=df.dropoff_longitude-df.pickup_longitude          # west yeild -ve on long_diff, vice versa & north yeild + lat_diff and vice versa, thus we can now give directions.
# df.drop(['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude'],axis=1,inplace=True)
```

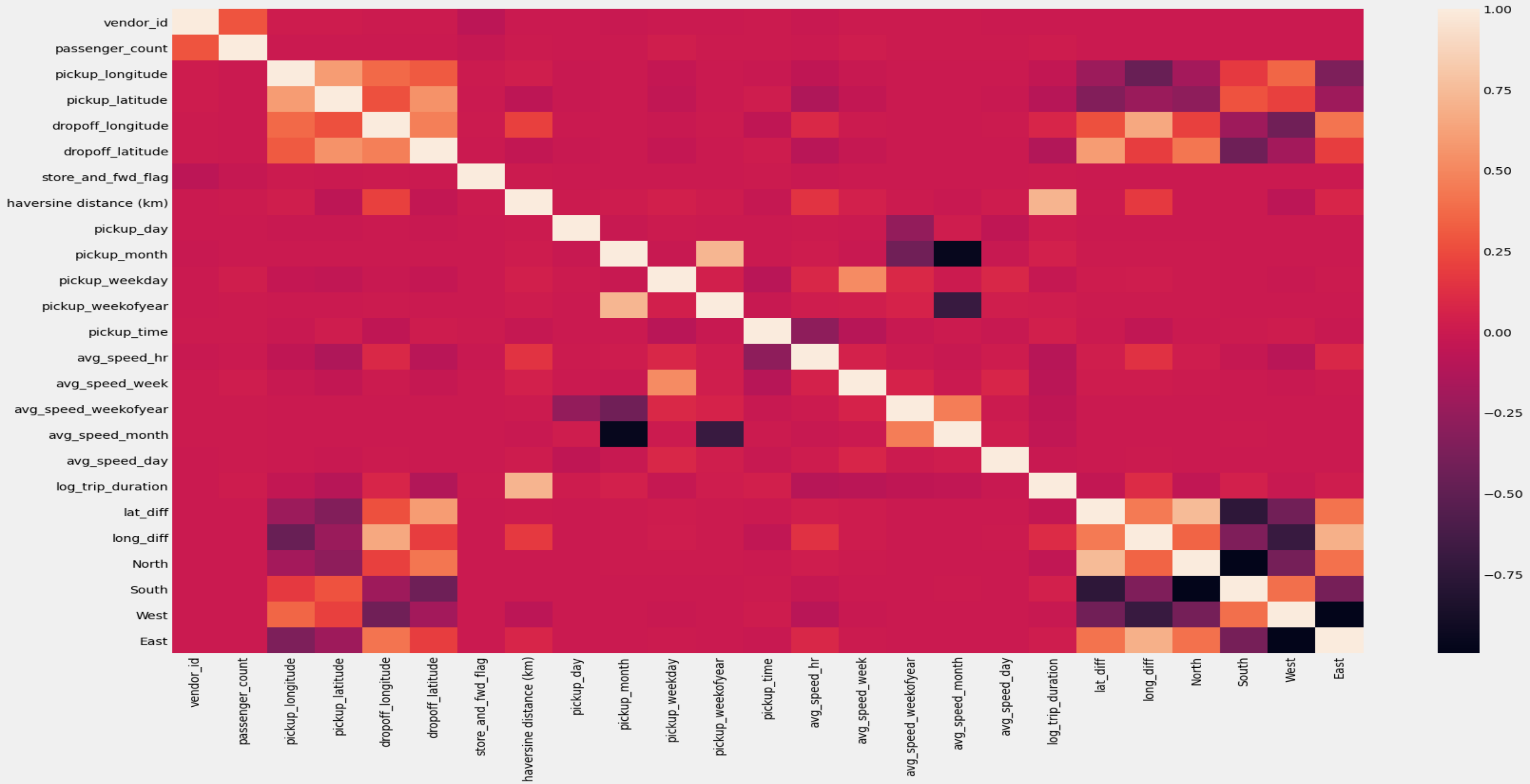
```
df['North']=df['lat_diff'].apply(lambda x: np.where(x>0,1,0))
df['South']=df['lat_diff'].apply(lambda x: np.where(x<0,1,0))
df['West']=df['long_diff'].apply(lambda x: np.where(x<0,1,0))
df['East']=df['long_diff'].apply(lambda x: np.where(x>0,1,0))

df.head()
```

Adding directions, and magnitude of those directions in the data, this might help the model to determine which sides are the most crowded and thus speeds are kind of slow and only contrary which direction are faster routes.

```
df.drop(['avg_speed_h2'],axis=1,inplace=True)
```

```
df.drop(['pickup_datetime','dropoff_datetime','trip_duration','pickup_date','dropoff_date','dropoff_day','dropoff_month','dropoff_weekday','dropoff_weekofyear','dropoff_time'],axis=1,inplace=True)
```

Dropping all the unwanted columns and making our final dataset, with all the features we would like to retain.



Now, we our dependent variable is better correlated with other varaibles.

Train/Test Split

Segregating Data into Train, Test and Validation sets (which will be used by CatBoost model to facilitate backpropogation)

```
X=df.drop('log_trip_duration',axis=1)
y=df['log_trip_duration']
X_train, X_test, y_train, y_test= train_test_split(X,y,test_size=0.05)
X_train, X_val, y_train, y_val= train_test_split(X_train,y_train,test_size=0.1) # Validation Set, which is used in Catboost Regressor to facilitate backpropogation.
print(f'Shape of X_train = {X_train.shape}')
print(f'Shape of X_test = {X_test.shape}')
print(f'Shape of X_val = {X_val.shape}')
```

X have our independent variables and y have Dependent Variable.

No. of observation in X_train, y_train = 1129284

No. of observation in X_test, y_test = 66040

No. of Observation in X_val,y_val = 125476

Model Selection & Hyper Parameter Tuning

AI

Two Model I will be using for this problem are XG Boost and Cat Boost, here I am not choosing Linear regression because it would not fit the data at all, polynomial regression and Random forest are computationally expensive and Decision tree is very basic, thus XGBoost and CatBoost seemed to be the apt choice.

In order to get best result I want my model to fit over the data Optimally, thus I will have to tune my hyper Parameters, here I would not go with cross validation along with grid search because data set is quite huge, well distributed and quite varied, thus K-fold cross validation we be computationally expensive and an extra step bearing no fruit.

```
[ ] from sklearn.model_selection import GridSearchCV
    xgb_model = XGBRegressor()
    learning_rate= [0.1,0.25,0.5]                                   # various learning rates i will tryout
    nax_depth = [10,12]
    n_estimators=[100,200,300]                                      # various depths that i will try out
    parameters = dict(learning_rate=learning_rate, nax_depth=nax_depth, n_estimators=n_estimators,objective=['reg:squarederror'])
    grid = GridSearchCV(xgb_model,parameters,scoring='r2', cv=None) # we can also use 'neg_mean_squared_error', here i am not using cv
    grid_result=grid.fit(X_train, y_train)
    print ("r2 / variance : ", grid.best_score_,'with parameter: ',grid_result.best_params_)
    print("RMSE score: %.2f"
                  % np.sqrt(metrics.mean_squared_error(y_test,grid.predict(X_test))))

    r2 / variance :  0.7442043087873197 with parameter:  {'learning_rate': 0.5, 'n_estimators': 300, 'nax_depth': 10, 'objective': 'reg:squarederror'}
    RMSE score: 0.33
```

Hyper Parameter Tuning for XGBoost using Grid Search

```
from sklearn.model_selection import GridSearchCV
cat_model = CatBoostRegressor()
learning_rate= [0.05,0.1,0.15]                                               # various learning rates i will tryout
depth = [6,8,10]                                                              # various depths that i will try out
parameters = dict(depth=depth,learning_rate=learning_rate,iterations=[1000], # max iterations are set to 1000
                  od_type=["Iter"],od_wait=[200],metric_period=[999],        # i have used overfitting detector & enables the use of best model
                  use_best_model = [True] )
grid = GridSearchCV(cat_model,parameters,scoring='r2', cv=None)              #we can also use 'neg_mean_squared_error', here i am not using cv as dataset is quite large and well distributed.
grid_result=grid.fit(X_train, y_train, eval_set=(X_val,y_val))
print ("r2 / variance : ", grid.best_score_,'with parameter: ',grid_result.best_params_)
print("RMSE score: %.2f"
              % np.sqrt(metrics.mean_squared_error(y_test,grid.predict(X_test))))
```

```
Warning: Overfitting detector is active, thus evaluation metric is calculated on every iteration. 'metric_period' is ignored for evaluation metric.
0:      learn: 0.6190210        test: 0.6202583 best: 0.6202583 (0)     total: 285ms    remaining: 4m 44s
999:    learn: 0.3065404        test: 0.3069071 best: 0.3069071 (999)   total: 2m 33s   remaining: 0us

bestTest = 0.3069071111
bestIteration = 999

Warning: Overfitting detector is active, thus evaluation metric is calculated on every iteration. 'metric_period' is ignored for evaluation metric.
0:      learn: 0.6190108        test: 0.6202668 best: 0.6202668 (0)     total: 177ms    remaining: 2m 56s
999:    learn: 0.3065526        test: 0.3070029 best: 0.3070029 (999)   total: 2m 33s   remaining: 0us

bestTest = 0.3070029181
bestIteration = 999

Warning: Overfitting detector is active, thus evaluation metric is calculated on every iteration. 'metric_period' is ignored for evaluation metric.
0:      learn: 0.6190072        test: 0.6202573 best: 0.6202573 (0)     total: 172ms    remaining: 2m 52s
999:    learn: 0.3066868        test: 0.3070562 best: 0.3070562 (999)   total: 2m 33s   remaining: 0us

bestTest = 0.3070562348
bestIteration = 999

Warning: Overfitting detector is active, thus evaluation metric is calculated on every iteration. 'metric_period' is ignored for evaluation metric.
0:      learn: 0.6188050        test: 0.6202697 best: 0.6202697 (0)     total: 168ms    remaining: 2m 47s
999:    learn: 0.3071048        test: 0.3071123 best: 0.3071123 (999)   total: 2m 34s   remaining: 0us

bestTest = 0.3071123334
bestIteration = 999

Warning: Overfitting detector is active, thus evaluation metric is calculated on every iteration. 'metric_period' is ignored for evaluation metric.
0:      learn: 0.6187154        test: 0.6202743 best: 0.6202743 (0)     total: 167ms    remaining: 2m 46s
999:    learn: 0.3070100        test: 0.3071382 best: 0.3071382 (999)   total: 2m 34s   remaining: 0us

bestTest = 0.3071381886
bestIteration = 999

Warning: Overfitting detector is active, thus evaluation metric is calculated on every iteration. 'metric_period' is ignored for evaluation metric.
0:      learn: 0.6011650        test: 0.6023562 best: 0.6023562 (0)     total: 169ms    remaining: 2m 48s
999:    learn: 0.2980814        test: 0.2994927 best: 0.2994927 (999)   total: 2m 33s   remaining: 0us
```

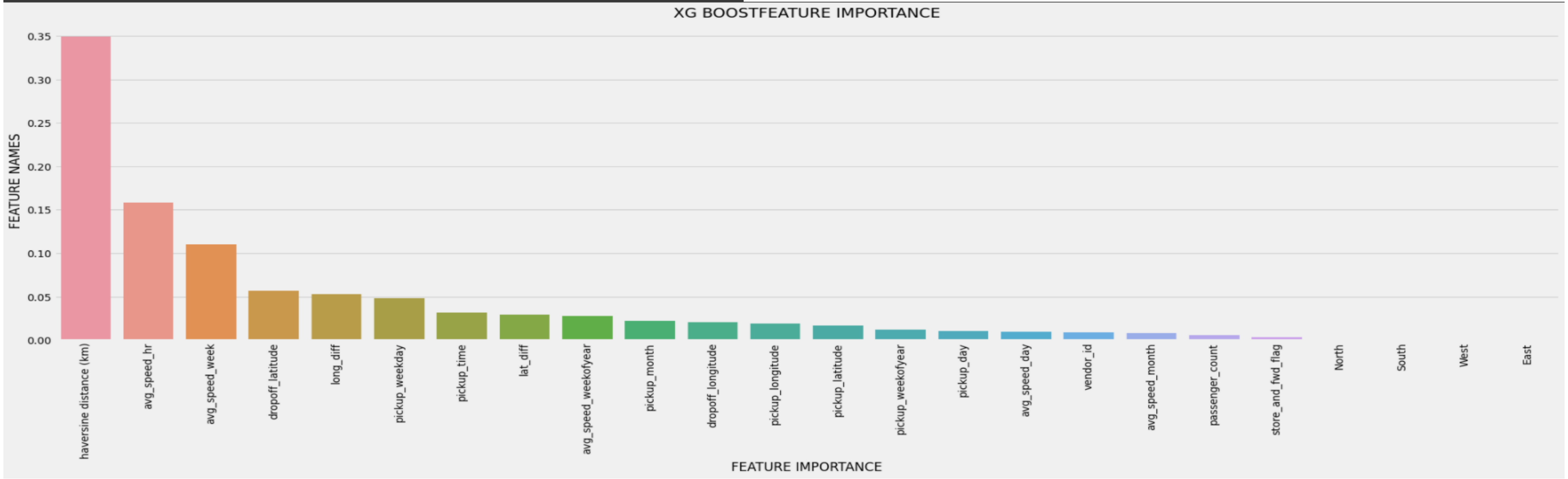
Hyper Parameter Tuning for CatBoost Model using Grid Search

Best Results For XGBoost :

```
model = XGBRegressor(max_depth=8,
                     learning_rate=0.5,
                     n_estimators=300,
                     verbosity=0,
                     objective='reg:squarederror')
model.fit(X_train,y_train)
y_pred_test=model.predict(X_test)
y_pred_train=model.predict(X_train)
print(r2_score(y_test,y_pred_test))
print('RMSE score for the CatRegressor is : {}'.format(np.sqrt(metrics.mean_squared_error(y_test,y_pred_test))))
print(r2_score(y_train,y_pred_train))
print('RMSE score for the CatRegressor is : {}'.format(np.sqrt(metrics.mean_squared_error(y_train,y_pred_train))))
```

```
0.7777962288171183
RMSE score for the CatRegressor is : 0.3003094106929618
0.794980950730289
RMSE score for the CatRegressor is : 0.2885120984088621
```

The best r2 score came out to be : 77% on Test set, while it was only 79% for the train set, increasing the max depth overfitted the model, where r2 score for test set remained almost the same, also root mean square error (rmse) score for both the model were 0.3 and 0.28 for test and train respectively.

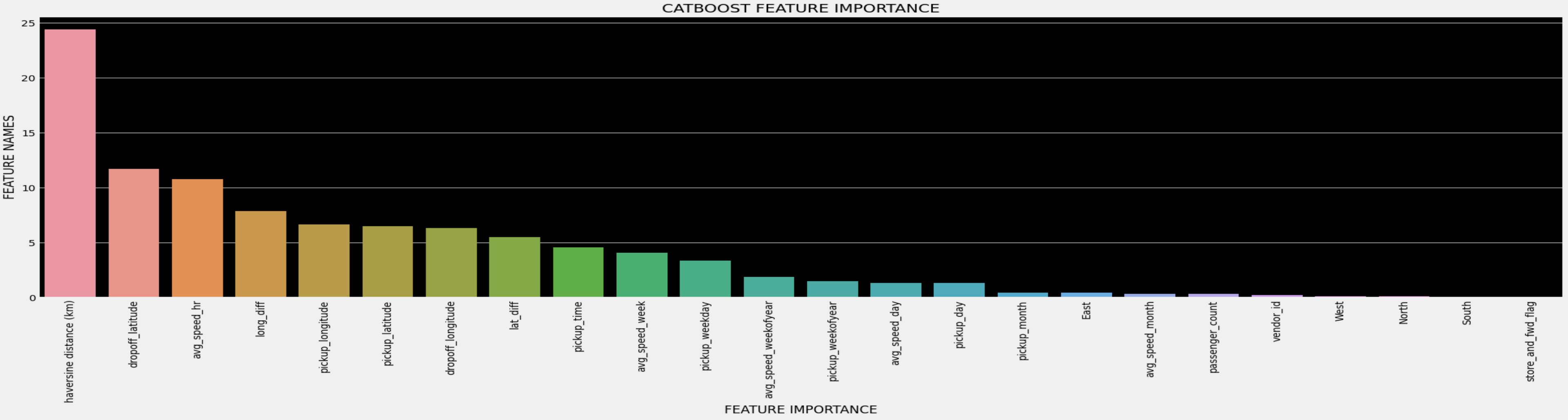


Best Results For CatBoost :

```
cat_model = CatBoostRegressor(loss_function = "RMSE", eval_metric = "RMSE", metric_period = 1000, iterations=30000,
                              use_best_model = True,
                              random_strength = 0.005,
                              learning_rate=0.1,
                              depth=8,
                              random_seed = 93,                                 # using best set of hyperparameters {'depth': 8, 'ite
                              l2_leaf_reg = 0.1,
                              verbose=True,
                              logging_level = None,od_type = "Iter",
                              od_wait = 200)
cat_model.fit( X_train, y_train, cat_features=None, eval_set=(X_val,y_val))
y_pred_test_cat=cat_model.predict(X_test)
y_pred_train_cat=cat_model.predict(X_train)
print(f'For learning rate = {0.1}, following are the scores of evaluation metrics:')
print(f'r2 score for test set using  CatRegressor is : {r2_score(y_test,y_pred_test_cat)}')
print('RMSE score for test set using CatRegressor is : {}'.format(np.sqrt(metrics.mean_squared_error(y_test,y_pred_test_cat))))
print(f'r2 score for train set using CatRegressor is : {r2_score(y_train,y_pred_train_cat)}')
print('RMSE score for train set using CatRegressor is : {}'.format(np.sqrt(metrics.mean_squared_error(y_train,y_pred_train_cat))))
# we are able to achieve 80 percent r2 score using this model
```

```
Shrink model to first 11622 iterations.
For learning rate = 0.1, following are the scores of evaluation metrics:
r2 score for test set using  CatRegressor is : 0.8070293409929914
RMSE score for test set using CatRegressor is : 0.2801441650659483
r2 score for train set using CatRegressor is : 0.8649972349058663
RMSE score for train set using CatRegressor is : 0.23420268773786626
```

The best r2 score came out to be : 80.7% on Test set, while it was only 86.5% for the train set, one the best hyperparameter, if I would have trained more, It would have led to overfitting, also the RMSE score for test and train is 0.28 and 0.23 respectively, which means model is a better fit when compared to XGBoost.



Conclusion

After building Two models to achieve my object, I conclude that both models performed really well specifically after hyper parameter tuning. R2 score for XG Boost model came out to be 77% on Test set, while it is 80.7% using CatBoost, while scores were 79% and 87% for the train set using the respective models.

In the end I can say that we have a ML model which could predict the tripduration on any ride with 80 percent certainty.

THANK YOU