In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
# Loading our cleaned dataset file.
```

In [3]:
```
data = pd.read_csv('cleaned_df.csv')
data.head()
```

Out[3]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration	vendor_id_1	vendor_id_2	store_and_fwd_flag_N	store_and_fwd_flag_Y	weekday	hour_of_day	month
0	1	-73.953918	40.778873	-73.963875	40.771164	400	0	1	1	0	0	16	2
1	2	-73.988312	40.731743	-73.994751	40.694931	1100	1	0	1	0	4	23	3
2	2	-73.997314	40.721458	-73.948029	40.774918	1635	0	1	1	0	6	17	2
3	6	-73.961670	40.759720	-73.956779	40.780628	1141	0	1	1	0	1	9	1
4	1	-74.017120	40.708469	-73.988182	40.740631	848	1	0	1	0	2	6	2

In [4]:
```
data.shape
```

Out[4]:
```
(729322, 13)
```

In [5]:
```
# separating independent and dependent variables

x = data.drop(['trip_duration'] , axis = 1)
y = data['trip_duration']
```

In [6]:
```
from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split(x , y , random_state = 100)
```

Random Forest Regressor Model

In [7]:
```
# importing the Random Forest Regressor from sklearn

from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(random_state = 100)
```

In [8]:
```
# importing mean absolute error for model evaluation since the problem is of regression type
from sklearn.metrics import mean_absolute_error as mae
```

In [9]:
```
clf.fit(x_train , y_train)
# generating predictions
pred1 = clf.predict(x_test)
pred1[:10] , mae(y_test , pred1)
```

Out[9]:
```
(array([1249.48, 1473.16,  338.06, 1283.1 ,  724.22,  532.52, 1322.62,
         525.65, 1510.85, 1401.08]),
 419.1827074288754)
```

In [10]:
```
# After making the random forest model , we scale our dataset for linear models.
```

In [11]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

In [12]:
```
# creating a dataframe for these scaled entities.
x = pd.DataFrame(x_scaled , columns = x.columns)
x.head()
```

Out[12]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	vendor_id_1	vendor_id_2	store_and_fwd_flag_N	store_and_fwd_flag_Y	weekday	hour_of_day	month
0	-0.504444	0.280911	0.832127	0.137198	0.538014	-0.931533	0.931533	0.074634	-0.074634	-1.560057	0.373006	-0.903461
1	0.257493	-0.212156	-0.570815	-0.306500	-1.577382	1.073500	-1.073500	0.074634	-0.074634	0.486536	1.466269	-0.308456
2	0.257493	-0.341220	-0.876953	0.364913	0.642175	-0.931533	0.931533	0.074634	-0.074634	1.509832	0.529187	-0.903461
3	3.305240	0.169785	0.261980	0.239160	0.800639	-0.931533	0.931533	0.074634	-0.074634	-1.048408	-0.720257	-1.498465
4	-0.504444	-0.625160	-1.263600	-0.212103	-0.309245	1.073500	-1.073500	0.074634	-0.074634	-0.536760	-1.188799	-0.903461

In [13]:
```
train_x , test_x , train_y , test_y = train_test_split(x , y , random_state = 100)
```

KNN Model

In [14]:
```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors = 390)
```

In [15]:
```
# since we obtained the minimum value of knn at n_neighbors = 390 , so we directly use that value here.
```

In [16]:
```
knn.fit(train_x , train_y)
pred2 = knn.predict(test_x)
pred2[:10] , mae(test_y , pred2)
```

Out[16]:
```
(array([ 833.50769231, 1133.19230769,  478.55384615,  671.92307692,
         755.64358974,  633.93076923, 1208.99230769,  626.95128205,
        1049.68974359, 1322.87179487]),
 437.03731116514086)
```

In [ ]:
```

```

In [17]:
```
# Building linear models with regularization
```

Lasso Regularization

In [18]:
```
from sklearn.linear_model import Lasso
lasso_reg = Lasso(alpha = 0.01 , normalize = True , max_iter = 1000 , tol = 0.00001)
lasso_reg.fit(train_x , train_y)
```

Out[18]:
```
Lasso(alpha=0.01, normalize=True, tol=1e-05)
```

In [19]:
```
pred3 = lasso_reg.predict(test_x)
pred3[:10] , mae(test_y , pred3)
```

Out[19]:
```
(array([ 701.83602365,  910.81124194,  882.27406855,  900.15441065,
         850.834994  ,  830.75923096, 1035.31314043,  784.62248751,
        1152.97194154,  861.95943972]),
 588.5287028506159)
```

Ridge Regularization

In [20]:
```
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha = 0.01 , normalize = True , max_iter = 1000 , tol = 0.00001)
ridge_reg.fit(train_x , train_y)
```

Out[20]:
```
Ridge(alpha=0.01, max_iter=1000, normalize=True, tol=1e-05)
```

In [21]:
```
pred4 = ridge_reg.predict(test_x)
pred4[:10] , mae(test_y , pred4)
```

Out[21]:
```
(array([ 648.68680887,  908.54527713,  877.34909048,  866.09899629,
         858.76118563,  817.57483251, 1044.33619914,  789.29516856,
        1176.95962808,  873.1314123 ]),
 588.8825527015712)
```

Gradient Boosting Algorithm

In [22]:
```
from sklearn.ensemble import GradientBoostingRegressor
reg = GradientBoostingRegressor(random_state = 100)
```

In [23]:
```
reg.fit(train_x , train_y)
pred5 = reg.predict(test_x)
pred5[:10] , mae(test_y , pred5)
```

Out[23]:
```
(array([ 803.32697893, 1066.75319169,  570.74889607, 1008.24955084,
         791.298212  ,  749.92083953,  913.80818112,  603.91161188,
         916.97626336,  905.89840421]),
 467.1633668740825)
```

Averaging

In [24]:
```
from statistics import mean
final_pred = np.array([])

#taking the mean of the predictions since the predictions are of continuous type

for i in range(0 , len(test_x)):
    final_pred = np.append(final_pred , mean([pred1[i] , pred2[i] , pred3[i] , pred4[i] , pred5[i]]))
```

In [25]:
```
mae(test_y , final_pred)
```

Out[25]:
```
460.2463925817601
```

In [ ]:
```

```

Weighted Averaging

In [27]:
```
from statistics import mean
final_pred = np.array([])

# Giving weightage to the knn model , random forest model and gradient boosting model for their low error values.

for i in range(0 , len(test_x)):
    final_pred = np.append(final_pred , mean([pred1[i] , pred1[i] , pred2[i] , pred2[i] , pred3[i] , pred4[i] , pred5[i] , pred5[i]]))
```

In [28]:
```
mae(test_y , final_pred)
```

Out[28]:
```
437.43629804948006
```

In [ ]:
```

```

Rank Averaging

In [29]:
```
m1_mae = mae(y_test , pred1)
m2_mae = mae(test_y , pred2)
m3_mae = mae(test_y , pred3)
m4_mae = mae(test_y , pred4)
m5_mae = mae(test_y , pred5)
```

In [32]:
```
# all  5 error values  from all 5 models.
m1_mae , m2_mae , m3_mae , m4_mae , m5_mae
```

Out[32]:
```
(419.1827074288754,
 437.03731116514086,
 588.5287028506159,
 588.8825527015712,
 467.1633668740825)
```

In [33]:
```
# Creating a dataframe with all the errors indexwise.

index_ = [1,2,3,4,5]
valid_mae = [m1_mae , m2_mae , m3_mae , m4_mae , m5_mae]

rank_eval = pd.DataFrame({
    'mae' : valid_mae
} , index = index_)
rank_eval
```

Out[33]:

	mae
1	419.182707
2	437.037311
3	588.528703
4	588.882553
5	467.163367

In [34]:
```
# Sorting those errors in descending order.
sorted_rank = rank_eval.sort_values('mae' , ascending = False)
sorted_rank
```

Out[34]:

	mae
4	588.882553
3	588.528703
5	467.163367
2	437.037311
1	419.182707

In [35]:
```
# Sorting rank wrt mae values. Largest error gets the least ranking and vice versa.
sorted_rank['rank'] = [i for i in range(1,6)]
sorted_rank
```

Out[35]:

	mae	rank
4	588.882553	1
3	588.528703	2
5	467.163367	3
2	437.037311	4
1	419.182707	5

In [36]:
```
# assigning weights according to their ranks.
sorted_rank['weight'] = sorted_rank['rank']/sorted_rank['rank'].sum()
sorted_rank
```

Out[36]:

	mae	rank	weight
4	588.882553	1	0.066667
3	588.528703	2	0.133333
5	467.163367	3	0.200000
2	437.037311	4	0.266667
1	419.182707	5	0.333333

In [37]:
```
wt_pred1 = pred1*float(sorted_rank.loc[[1] , ['weight']].values)
wt_pred2 = pred2*float(sorted_rank.loc[[2] , ['weight']].values)
wt_pred3 = pred3*float(sorted_rank.loc[[3] , ['weight']].values)
wt_pred4 = pred4*float(sorted_rank.loc[[4] , ['weight']].values)
wt_pred5 = pred5*float(sorted_rank.loc[[5] , ['weight']].values)

# calculating the final pedicted values
rank_prediction = wt_pred1 + wt_pred2 + wt_pred3 + wt_pred4 + wt_pred5
rank_prediction
```

Out[37]:
```
array([ 936.25137081, 1188.59977112,  530.57728669, ..., 1868.54330612,
       1625.53378687,  960.01857239])
```

In [38]:
```
mae(test_y , rank_prediction)
```

Out[38]:
```
425.5500361973428
```

In [ ]:
```
# this is the final error score for the averaged models based on rank averaging.
```

In [ ]:
```

```