Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travelers saying that flight ticket prices are so unpredictable. As data scientists, we are gonna prove that given the right data anything can be predicted. Here you will be provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.

Datasets

We will be using two datasets — Train data and Test data

FEATURES:

Airline: The name of the airline.

Date_of_Journey: The date of the journey

Source: The source from which the service begins.

Destination: The destination where the service ends.

Route: The route taken by the flight to reach the destination.

Dep_Time: The time when the journey starts from the source.

Arrival_Time: Time of arrival at the destination.

Duration: Total duration of the flight.

Total_Stops: Total stops between the source and destination.

Additional_Info: Additional information about the flight

Price: The price of the ticket

Training data is combination of both categorical and numerical also we can see some special character also being used because of which we have to do data Transformation on it before applying it to our model

The test data is similar to the training data set, minus the 'Price' column (To be predicted using the model).

Importing the necessary libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

```
train_data=pd.read_excel(r"C:\Users\RAKESH~1\AppData\Local\Temp\Rar$DIa10428.48239\Data_Train.xlsx","Sheet1")
train_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

Checking the data types of the training data

In [143…

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

Checking for null values,shape of the training data

Dropping the null values if any

Counting the unique values in the variable called ‘Duration’

```
In [144… train_data["Duration"].value_counts()

Out[144… 2h 50m     550
         1h 30m     386
         2h 55m     337
         2h 45m     337
         2h 35m     329
                   ...
         32h 20m      1
         30h 25m      1
         37h 10m      1
         33h 20m      1
         29h 30m      1
         Name: Duration, Length: 368, dtype: int64

In [145… train_data.dropna(inplace = True)

In [146… train_data.isnull().sum()

Out[146… Airline            0
         Date_of_Journey    0
         Source             0
         Destination        0
         Route              0
         Dep_Time           0
         Arrival_Time       0
         Duration           0
         Total_Stops        0
         Additional_Info    0
         Price              0
         dtype: int64
```

Step 3: Feature Generation

In this step we mainly work on the data set and do some transformation like creating different bins of particular columns ,clean the messy data so that it can be used in our ML model . This step is very important because for a high prediction score you need to continuously make changes in it

Date_of_Journey:

In the column 'Date_of_Journey', we can see the date format is given as dd/mm/yyyy and as you can see the datatype is given as object So there is two ways to tackle this column, either convert the column into Timestamp or divide the column into date,Month ,Year. Here , i am splitting the columns

```
In [147…  ## no null values are found

In [148…  ##EDA

In [149…  train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/%Y").dt.day

In [150…  train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month

In [151…  train_data.head()
```

Arrival_Time:

In the column '**Arrival_Time**',if we see we have combination of both time and month but we need only the time details out of it so we split the time into 'Hours' and 'Minute'.

```
In [155…
# Time taken by plane to reach destination is called Duration
# It is the differnce betwwen Departure Time and Arrival time


# Assigning and converting Duration column into list
duration = list(train_data["Duration"])

for i in range(len(duration)):
    if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
        else:
            duration[i] = "0h " + duration[i]           # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duratic

In [156…
# Adding duration_hours and duration_mins list to train_data dataframe

train_data["Duration_hours"] = duration_hours
train_data["Duration_mins"] = duration_mins

In [157…
train_data.drop(["Duration"], axis = 1, inplace = True)
train_data.head()
```

Step 4: Prepare categorical variables for model using label encoder

To convert categorical text data into model-understandable numerical data, we use the Label Encoder class. So all we have to do, to label encode a column is import the LabelEncoder class from the sklearn library, fit and transform the column of the data, and then replace the existing text data with the new encoded data.

To convert categorical text data into model-understandable numerical data, we use the Label Encoder class. So all we have to do, to label encode a column is

```python
from sklearn.preprocessing import LabelEncoder

lb_encode = LabelEncoder()
big_df["Additional_Info"] = lb_encode.fit_transform(big_df["Additional_Info"])
big_df["Airline"] = lb_encode.fit_transform(big_df["Airline"])
big_df["Destination"] = lb_encode.fit_transform(big_df["Destination"])
big_df["Source"] = lb_encode.fit_transform(big_df["Source"])
big_df['Route_1']= lb_encode.fit_transform(big_df["Route_1"])
big_df['Route_2']= lb_encode.fit_transform(big_df["Route_2"])
big_df['Route_3']= lb_encode.fit_transform(big_df["Route_3"])
big_df['Route_4']= lb_encode.fit_transform(big_df["Route_4"])
big_df['Route_5']= lb_encode.fit_transform(big_df["Route_5"])
```

encoding all the categorical variables

In [164…

```python
# As Source is Nominal Categorical data we will perform OneHotEncoding
Source = train_data[["Source"]]
Source = pd.get_dummies(Source, drop_first= True)
Source.head()
```

Out[164…

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

In [165…

```python
train_data["Destination"].value_counts()
```

Out[165…

```
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

In [166…

```python
# As Destination is Nominal Categorical data we will perform OneHotEncoding
Destination = train_data[["Destination"]]
Destination = pd.get_dummies(Destination, drop_first = True)
Destination.head()
```

In [164…

```python
# As Source is Nominal Categorical data we will perform OneHotEncoding
Source = train_data[["Source"]]
Source = pd.get_dummies(Source, drop_first= True)
Source.head()
```

Out[164…

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

In [165…

```python
train_data["Destination"].value_counts()
```

Out[165…

```
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

In [166…

```python
# As Destination is Nominal Categorical data we will perform OneHotEncoding
Destination = train_data[["Destination"]]
Destination = pd.get_dummies(Destination, drop_first = True)
Destination.head()
```

Perform one hot encoding , label encoding and ordinal encoding wherever necessary

In [166…
```python
# As Destination is Nominal Categorical data we will perform OneHotEncoding

Destination = train_data[["Destination"]]

Destination = pd.get_dummies(Destination, drop_first = True)

Destination.head()
```

Out[166…

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	0	0	0	0	1
1	0	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

In [167…
```python
train_data["Route"]
```

Out[167…
```
0                    BLR → DEL
1        CCU → IXR → BBI → BLR
2        DEL → LKO → BOM → COK
3              CCU → NAG → BLR
4              BLR → NAG → DEL
                  ...
10678                CCU → BLR
10679                CCU → BLR
10680                BLR → DEL
10681                BLR → DEL
10682    DEL → GOI → BOM → COK
Name: Route, Length: 10682, dtype: object
```

In [168…
```python
# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other

train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

In [169…
```python
train_data["Total_Stops"].value_counts()
```

Out[169…
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [170…
```python
# As this is case of Ordinal Categorical type we perform LabelEncoder
# Here Values are assigned with corresponding keys

train_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)
```

In [171…
```python
train_data.head()
```

Out[171…

	Airline	Source	Destination	Arrival_Time	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Duration_hours	Duratio
0	IndiGo	Banglore	New Delhi	01:10 22 Mar	0	3897	24	3	22	20	2	
1	Air India	Kolkata	Banglore	13:15	2	7662	1	5	5	50	7	
2	Jet Airways	Delhi	Cochin	04:25 10 Jun	2	13882	9	6	9	25	19	
3	IndiGo	Kolkata	Banglore	23:30	1	6218	12	5	18	5	5	
4	IndiGo	Banglore	New Delhi	21:35	1	13302	1	3	16	50	4	

Concatenating the categorical and continuous variable in single data frame

In [172…

```
# Concatenate dataframe --> train_data + Airline + Source + Destination

data_train = pd.concat([train_data, Airline, Source, Destination], axis = 1)
data_train.head()
```

Out[172…

	Airline	Source	Destination	Arrival_Time	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	...	Airline_Vistara Premium economy	Sour
0	IndiGo	Banglore	New Delhi	01:10 22 Mar	0	3897	24	3	22	20	...	0	
1	Air India	Kolkata	Banglore	13:15	2	7662	1	5	5	50	...	0	
2	Jet Airways	Delhi	Cochin	04:25 10 Jun	2	13882	9	6	9	25	...	0	
3	IndiGo	Kolkata	Banglore	23:30	1	6218	12	5	18	5	...	0	
4	IndiGo	Banglore	New Delhi	21:35	1	13302	1	3	16	50	...	0	

5 rows × 32 columns

In [173…

```
data_train.drop(["Airline", "Source", "Destination","Arrival_Time"], axis = 1, inplace = True)
data_train
```

Out[173…

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	...	Ai
0	0	3897	24	3	22	20	2	50	0	0	...	
1	2	7662	1	5	5	50	7	25	1	0	...	
2	2	13882	9	6	9	25	19	0	0	0	...	
3	1	6218	12	5	18	5	5	25	0	0	...	
4	1	13302	1	3	16	50	4	45	0	0	...	

Dropping the unnecessary columns such as Airline,source,Destination,Arrival time and displaying the data frame will be our independent variable and dependent variable

And the same process will be applied for the test data

In [176…

```
## test data
```

In [177…

```
test_data=pd.read_excel(r"C:\Users\RAKESH~1\AppData\Local\Temp\Rar$DIa10428.46228\Test_set.xlsx","Sheet1")
```

In [178…

```
test_data.head()
```

Out[178…

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No info
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	1 stop	No info
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	19:15	19:00 22 May	23h 45m	1 stop	In-flight meal not included
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	1 stop	No info
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL	23:55	02:45 25 Jun	2h 50m	non-stop	No info

In [179…

Preprocessing is done as same as training data

In [180…
```
x=D
x
```
Out[180…

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	...	Ai
0	0	3897	24	3	22	20	2	50	0	0	...	
1	2	7662	1	5	5	50	7	25	1	0	...	
2	2	13882	9	6	9	25	19	0	0	0	...	
3	1	6218	12	5	18	5	5	25	0	0	...	
4	1	13302	1	3	16	50	4	45	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
10678	0	4107	9	4	19	55	2	30	0	0	...	
10679	0	4145	27	4	20	45	2	35	1	0	...	
10680	0	7229	27	4	8	20	3	0	0	0	...	
10681	0	12648	1	3	11	30	2	40	0	0	...	
10682	2	11753	9	5	10	55	8	20	1	0	...	

10682 rows × 28 columns

In [182…
```
y = data_train.iloc[:, 1]
y.head()
```
Out[182…
```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

Separating the independent variables and depending variables x and y

Feature selection method

In [185…
```
# Important feature using ExtraTreesRegressor

from sklearn.ensemble import ExtraTreesRegressor
selection = ExtraTreesRegressor()
selection.fit(x, y)
```
Out[185… ExtraTreesRegressor()

In [186…
```
print(selection.feature_importances_)
```
```
[1.58712947e-01 5.83804843e-01 3.93237783e-03 2.19301036e-03
 2.20178817e-04 1.28583830e-04 8.11642089e-02 4.11788062e-04
 1.60907910e-03 6.66170008e-06 4.96049487e-03 1.01279398e-01
 2.95070104e-02 2.95166223e-03 7.66556294e-08 2.85793802e-04
 2.91672625e-09 4.77059810e-04 3.41604658e-07 2.94609524e-05
 4.24576205e-03 3.23842082e-04 1.13701119e-03 6.31342510e-03
 8.31441315e-03 1.73246811e-03 4.03622940e-05 6.21773604e-03]
```

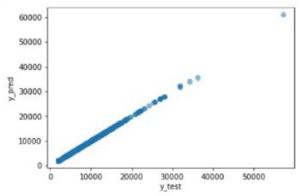
In [188…
```
#plot graph of feature importances for better visualization

plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=x.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```

Step 6: Build Model

The goal in this step is to develop a benchmark model that serves us as a baseline, upon which we will measure the performance of a better and more tuned algorithm. We are using different Regression Technique and comparing them to see which algorithm is giving better performance then other and At the end we will combine all of them using Stacking and see how our model is predicting

```
In [190…
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)

In [191…
#importing models
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor

In [192…
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)

Out[192… RandomForestRegressor()

In [193…
y_pred = reg_rf.predict(X_test)

In [194…
reg_rf.score(X_test, y_test)

Out[194… 0.9996212896151876
```

```
In [195…
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```




```
In [196…
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

In [198…
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_test, y_pred)))

MAE: 4.42296209639681
MSE: 8165.774647870852
RMSE: 90.36467588538595

In [199…
r2_score(y_test,y_pred)

Out[199… 0.9996212896151876
```

We will do this modeling for all the models and we select the model with the good accuracy and we do hyper parameter tuning

```
## hyperparamter tuning
```

In [225..
```
from sklearn.model_selection import RandomizedSearchCV
```

In [226..
```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [228..
```
n_estimators
```

Out[228..
```
[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200]
```

In [229..
```
# Create the random grid

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

In [230..
```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions = random_grid,scoring='neg_mean_squared_er
```

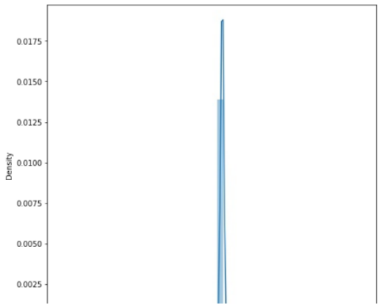
In [231..
```
rf_random.fit(X_train,y_train)
```

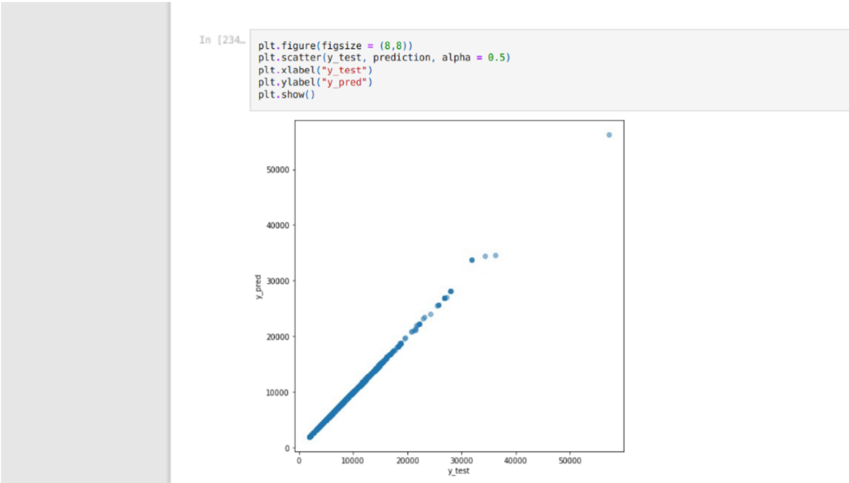
```
rf_random.best_params_
```

Out[232..
```
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

In [233..
```
prediction = rf_random.predict(X_test)
```

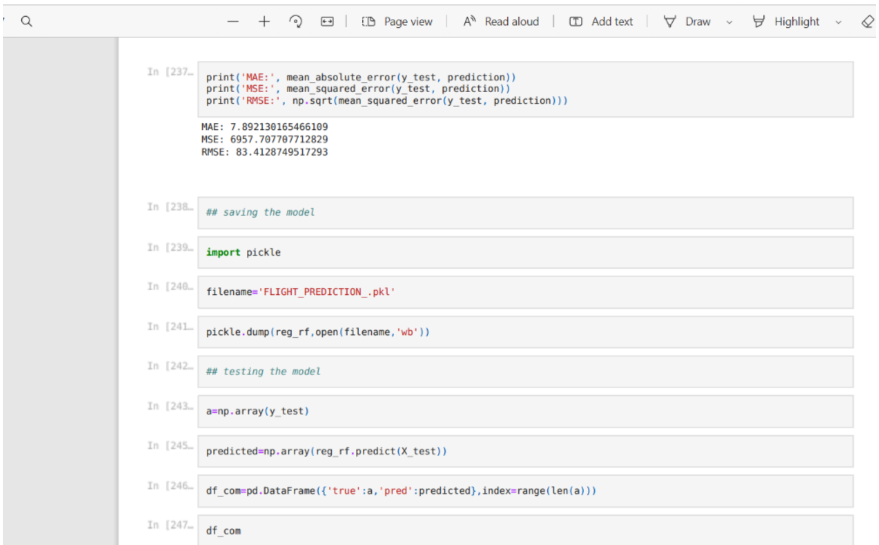
In [235..
```
plt.figure(figsize = (8,8))
sns.distplot(y_test-prediction)
plt.show()
```





In [234…

```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```

In [237…

```
print('MAE:', mean_absolute_error(y_test, prediction))
print('MSE:', mean_squared_error(y_test, prediction))
print('RMSE:', np.sqrt(mean_squared_error(y_test, prediction)))
```

```
MAE: 7.892130165466109
MSE: 6957.707707712829
RMSE: 83.4128749517293
```

In [238…

```
## saving the model
```

In [239…

```
import pickle
```

In [240…

```
filename='FLIGHT_PREDICTION_.pkl'
```

In [241…

```
pickle.dump(reg_rf,open(filename,'wb'))
```

In [242…

```
## testing the model
```

In [243…

```
a=np.array(y_test)
```

In [245…

```
predicted=np.array(reg_rf.predict(X_test))
```

In [246…

```
df_com=pd.DataFrame({'true':a,'pred':predicted},index=range(len(a)))
```

In [247…

```
df_com
```

Random forest works better and hypertuned the random forest

And we evaluated the model ..

Final Word

In this type of problem Feature Engineering is the most crucial think . You can see how we have handled the categorical and numerical data and also how we build different ML model on the same dataset . We also check the RMSE score of each model so that we can understand how it should perform in our test dataset . At last You can also further improve the Model by Tunning different parameters which are being used in the model .