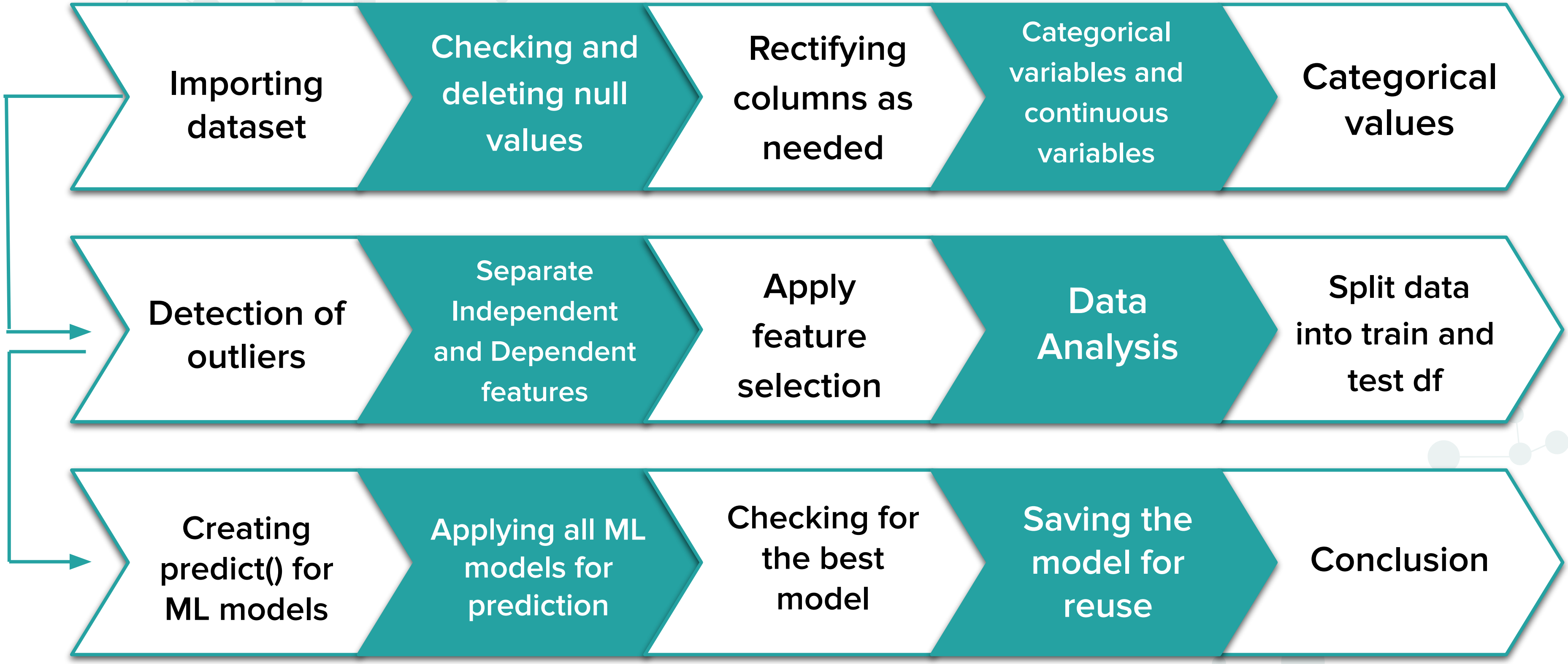
Flight Price Prediction

Sayli Bhavsar (1914009) | Dhwani Doshi (1914016)

Flow Of The Presentation

1. Problem Statement
2. Data Preprocessing
3. Data Analysis
4. Machine Learning

Steps Of The Mini Project



01

Problem Definition

Problem Definition

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. We are gonna prove that given the right data anything can be predicted. Here we will be provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.

About the dataset

- **Size of training set:** 10683 records & **test set:** 2671 records
- **Airline:** The name of the airline.
- **Destination:** The destination where the service ends.
- **Route:** The route taken by the flight to reach the destination.
- **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

Modules Used

01
Numpy

02
Pandas

03
Seaborn

04
Matplotlib

05
Pickle

06
Sklearn:

- LabelEncoder from sklearn.preprocessing
- Mutual_info_classif from sklearn.feature_selection
- train_test_split from sklearn.model_selection
- metrics from sklearn
- RandomForestRegressor from sklearn.ensemble
- LinearRegression from sklearn.linear_model
- KNeighborsRegressor from sklearn.neighbors
- DecisionTreeRegressor from sklearn.tree
- RandomizedSearchCV from sklearn.model_selection

02

Data Preprocessing

Overview

```
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

Rectifying Columns

- Column “Date_of_Journey” → “Journey_day” and “Journey_month”
- Column “Dep_Time” → “Dep_Time_hour” and “Dep_Time_Minute”
- Column “Arrival_Time” → “Arrival_Time_hour” and “Arrival_Time_minute”
- Column “Duration” → “Duration_hours” and “Duration_mins”

```python
def change_into_datetime(col):
    train_data[col]=pd.to_datetime(train_data[col])
```

```python
train_data['Journey_day']=train_data['Date_of_Journey'].dt.day
train_data['Journey_month']=train_data['Date_of_Journey'].dt.month
```

```python
def extract_hour(df,col):
    df[col+"_hour"]=df[col].dt.hour

def extract_min(df,col):
    df[col+"_minute"]=df[col].dt.minute

def drop_column(df,col):
    df.drop(col,axis=1,inplace=True)
```

```python
extract_hour(train_data,'Dep_Time
extract_min(train_data,'Dep_Time'
drop_column(train_data,'Dep_Time'
```

```python
extract_hour(train_data,'Arrival_Time')
extract_min(train_data,'Arrival_Time')
drop_column(train_data,'Arrival_Time')
```

```python
duration=list(train_data['Duration'])

for i in range(len(duration)):
    if len(duration[i].split(' '))==2:
        pass
    else:
        if 'h' in duration[i]:                    # Check if duration contains only hour
            duration[i]=duration[i] + ' 0m'       # Adds 0 minute
        else:
            duration[i]='0h '+ duration[i]        # if duration contains only second, Adds 0 hour
```

```python
train_data['Duration']=duration
```

```python
def hour(x):
    return x.split(' ')[0][0:-1]

def min(x):
    return x.split(' ')[1][0:-1]
```

```python
train_data['Duration_hours']=train_data['Duration'].apply(hour)
train_data['Duration_mins']=train_data['Duration'].apply(min)
```

Dataset After Changes

Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_Time_hour	Dep_Time_minute	Arrival_Time_hour	Arrival_Time_minute	Duration_hours	Duration_mins
BLR → DEL	non-stop	No info	3897	24	3	22	20	1	10	2	50
CCU → IXR → BBI → BLR	2 stops	No info	7662	5	1	5	50	13	15	7	25
DEL → LKO → BOM → COK	2 stops	No info	13882	6	9	9	25	4	25	19	0
CCU → NAG → BLR	1 stop	No info	6218	5	12	18	5	23	30	5	25
BLR → NAG → DEL	1 stop	No info	13302	3	1	16	50	21	35	4	45

Dividing Into Categorical & Numerical Variables

- Nominal data --> data are not in any order --> OneHotEncoder is used in this case
- Ordinal data --> data are in order --> LabelEncoder is used in this case

	Airline	Source	Destination	Route	Total_Stops	Additional_Info
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info

Categorical Columns

- After Performing OneHotEncoding on Airlines Column

	Air India	GoAir	IndiGo	Jet Airways	Jet Airways Business	Multiple carriers	Multiple carriers Premium economy	SpiceJet	Trujet	Vistara	Vistara Premium economy
0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	0

Categorical Columns

- After Performing OneHotEncoding on Source Column

	Cochin	Delhi	Hyderabad	Kolkata	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

Categorical Columns

- After Performing OneHotEncoding on Destination Column

	Chennai	Delhi	Kolkata	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

Categorical Columns

- After Splitting Route Column:

	Airline	Source	Destination	Total_Stops	Additional_Info	Route_1	Route_2	Route_3	Route_4	Route_5
0	IndiGo	Banglore	New Delhi	non-stop	No info	BLR	DEL	None	None	None
1	Air India	Kolkata	Banglore	2 stops	No info	CCU	IXR	BBI	BLR	None
2	Jet Airways	Delhi	Cochin	2 stops	No info	DEL	LKO	BOM	COK	None
3	IndiGo	Kolkata	Banglore	1 stop	No info	CCU	NAG	BLR	None	None
4	IndiGo	Banglore	New Delhi	1 stop	No info	BLR	NAG	DEL	None	None

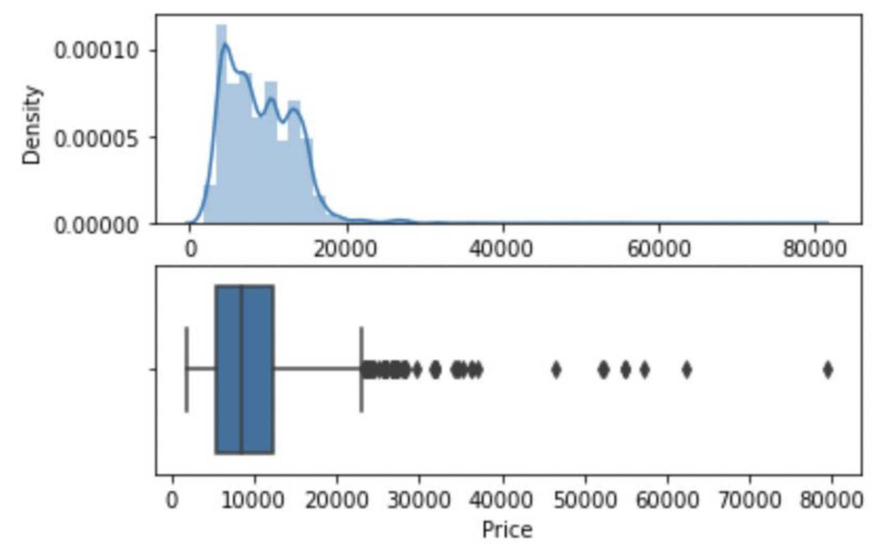
Categorical Columns

- Using LabelEncoder on Route 1, Route 2 … and Total Stops

	Airline	Source	Destination	Total_Stops	Route_1	Route_2	Route_3	Route_4	Route_5
0	IndiGo	Banglore	New Delhi	0	0	13	29	13	5
1	Air India	Kolkata	Banglore	2	2	25	1	3	5
2	Jet Airways	Delhi	Cochin	2	3	32	4	5	5
3	IndiGo	Kolkata	Banglore	1	2	34	3	13	5
4	IndiGo	Banglore	New Delhi	1	0	34	8	13	5

Outliers: Detection and Imputing

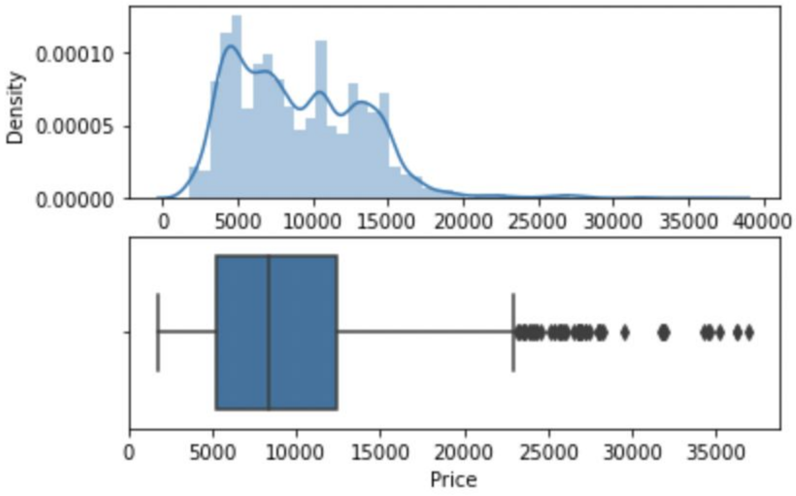
```
plot(data_train,'Price')
```



Imputing outliers (price>=40000) by median of the data

```
data_train['Price']=np.where(data_train['Price']>=40000,data_train['Price'].median(),data_train['Price'])
```

```
plot(data_train,'Price')
```



Apply Feature Selection

- Find the feature which contributes most to the target variable i.e price

```
[ ] imp= pd.DataFrame(mutual_info_classif(X,y),index=X.columns)
```

Print columns from highest to lowest importance

```
[ ] imp.columns=['importance']
    imp.sort_values(by='importance',ascending=False)
```

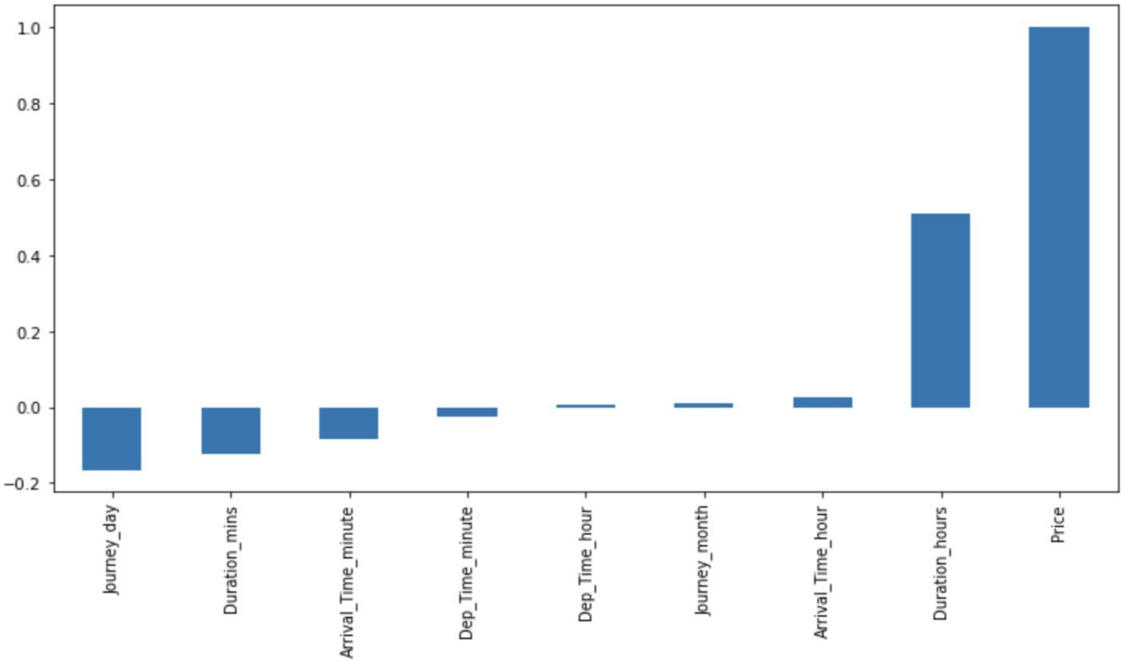
Total_Stops	2.152682
Route_1	2.009530
Arrival_Time_hour	1.862940
Duration_hours	1.804452
Delhi	1.544504
Cochin	1.525066
Arrival_Time_minute	1.505011
Route_4	1.465240
Dep_Time_hour	1.441511
Dep_Time_minute	1.188782
Journey_day	1.076345
Duration_mins	1.063301

03

Data Analysis

Correlation Of Columns With Price

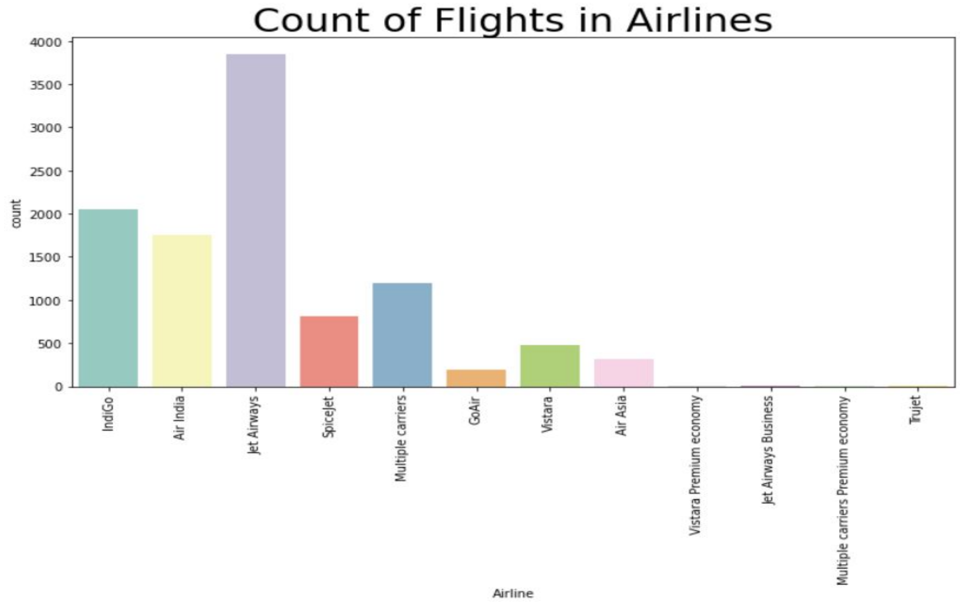
```
plt.figure(figsize=(12,6))
train_data.corr()['Price'].sort_values().plot(kind='bar');
```



* Departure time, Journey month, Arrival time and duration have positive correlation with price
* Duration hours have maximum correlation with price.

Count Of Flights In Airlines

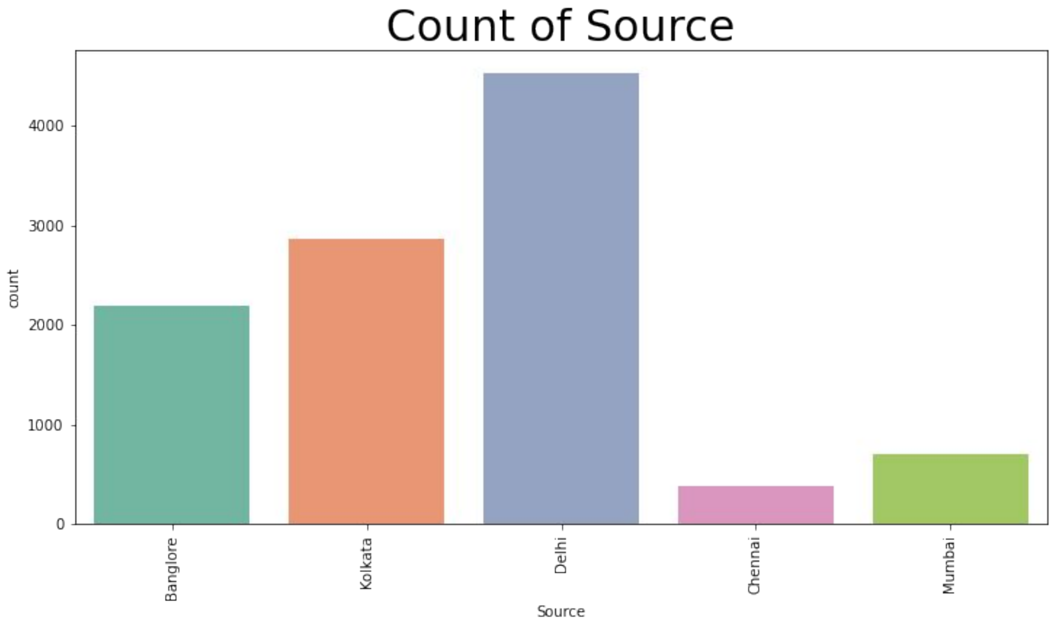
```
plt.figure(figsize=(12,6))
sns.countplot(train_data['Airline'], palette='Set3')
plt.title('Count of Flights in Airlines', size=30)
plt.xticks(rotation=90)
plt.show()
```



- Jet Airways have the maximum number of flights
- Vistara Premium economy, Jet Airways Business, Multiple carriers Premium economy and Trujet have very few or none as compared to the rest of the airlines.

Count Of Source

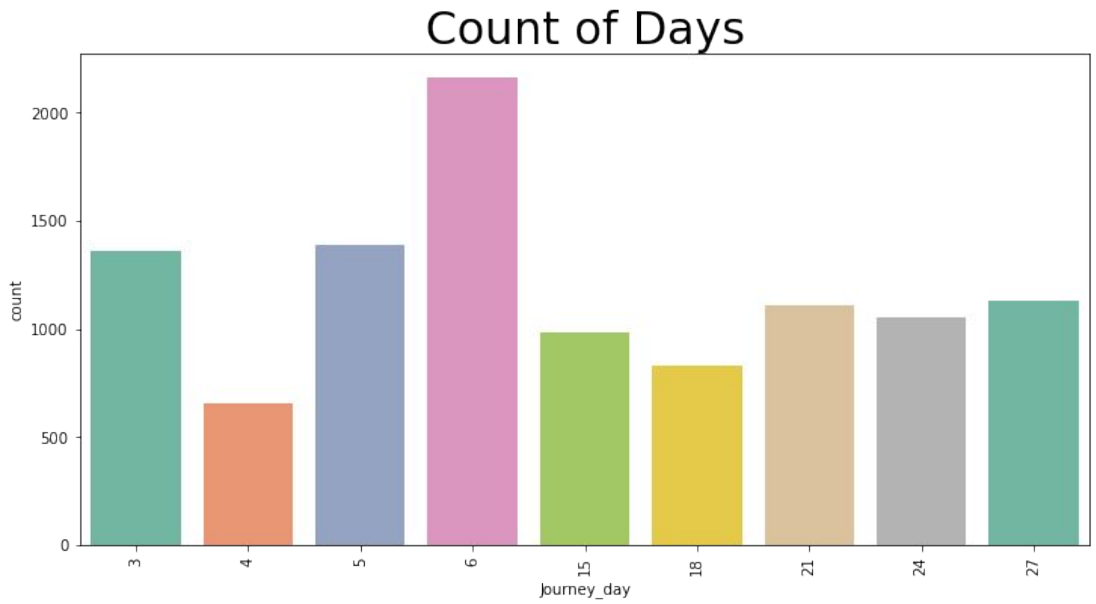
```
[ ] plt.figure(figsize=(12,6))
    sns.countplot(train_data['Source'], palette='Set2')
    plt.title('Count of Source', size=30)
    plt.xticks(rotation=90)
    plt.show()
```



- Delhi has the maximum number of flights that leave it.
- Chennai has the minimum number of flights that leave it.

Count Of Days

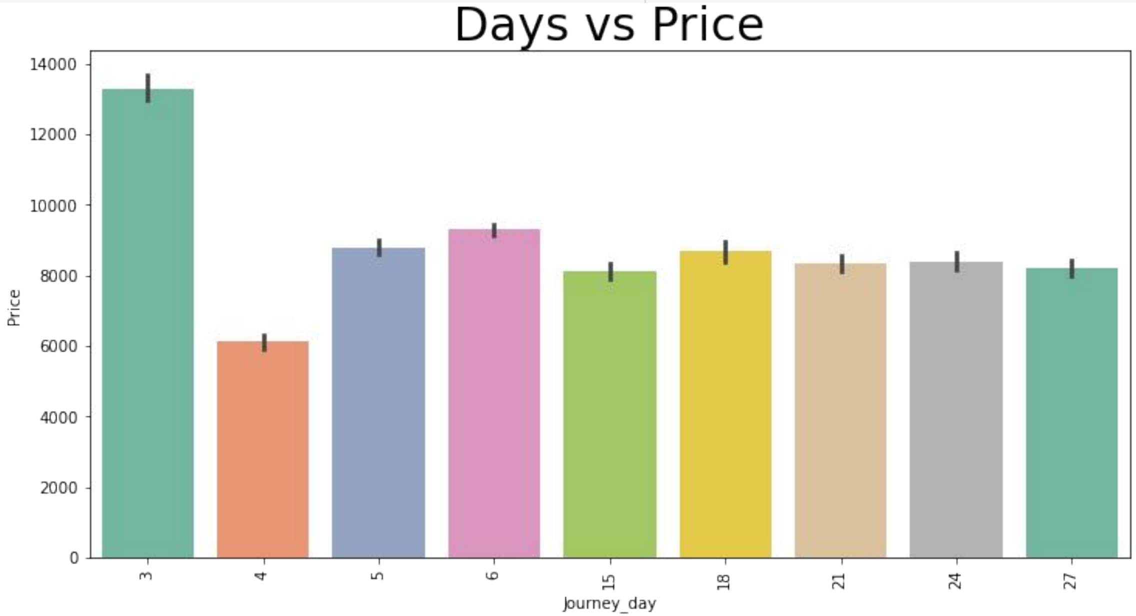
```
[ ] plt.figure(figsize=(12,6))
    sns.countplot(train_data['Journey_day'], palette='Set2')
    plt.title('Count of Days', size=30)
    plt.xticks(rotation=90)
    plt.show()
```



- The first week of any month have more flights as compared to the rest of the month.

Days vs Price

```
[ ] plt.figure(figsize=(12,6))
    sns.barplot(train_data['Journey_day'], train_data['Price'], palette='Set2')
    plt.title('Days vs Price', size=30)
    plt.xticks(rotation=90)
    plt.show()
```



- The cost of flights for the first week of any month is more as compared to the rest of the month.
- Price seems to dip on the 4th day of the month and increases back on the 5th.

Month vs Price

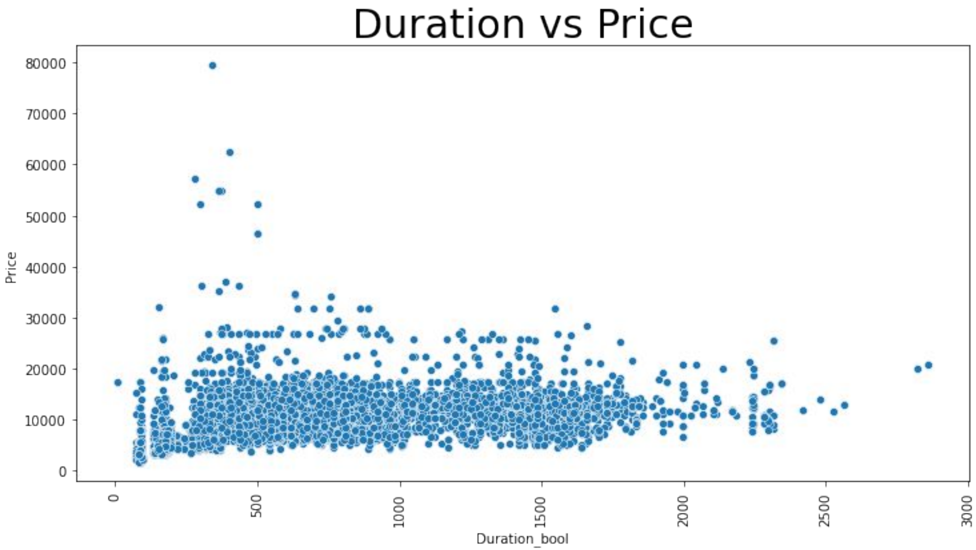
```
[ ] train_data['Journey_month'] = train_data['Journey_month'].map({1:'JAN',2:'FEB',3:'MAR',4:'APR',5:'MAY',:'JUN',7:'JUL',8:'AUG',9:'SEP',10:'OCT',11:'NOV',12:'DEC'})
    plt.figure(figsize=(12,6))
    sns.barplot(train_data['Journey_month'], train_data['Price'], palette='Set2')
    plt.title('Month vs Price', size=30)
    plt.xticks(rotation=90)
    plt.show()
```

- The price of flights is highest in the month of January
- The price of flights is lowest in the month of April

Duration vs Price

```
[ ] train_data['Duration_bool'] = (train_data['Duration_hours']*60)+train_data['Duration_mins']

    plt.figure(figsize=(12,6))
    sns.scatterplot(train_data['Duration_bool'], train_data['Price'], palette='Set2')
    plt.title('Duration vs Price', size=30)
    plt.xticks(rotation=90)
    plt.show()
```



- The cost of flights for the first week of any month is more as compared to the rest of the month.
- Price seems to dip on the 4th day of the month and increases back on the 5th.

Stops vs Price

```
[ ] plt.figure(figsize=(12,6))
    sns.barplot(train_data['Total_Stops'], train_data['Price'], palette='Set2')
    plt.title('Stops vs Price', size=30)
    plt.xticks(rotation=90)
    plt.show()
```

- The price of increases with the number of stops
- The price suddenly increases for flights with 2 stops

04

Machine Learning

Splitting data into train and test

```
[ ] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

```
[ ] from sklearn import metrics
    ##dump your model using pickle so that we will re-use
    import pickle
    def predict(ml_model,dump):
        model=ml_model.fit(X_train,y_train)
        print('Training score : {}'.format(model.score(X_train,y_train)))
        y_prediction=model.predict(X_test)
        df = pd.DataFrame({'Actual': y_test, 'Predicted': y_prediction})
        r2_score=metrics.r2_score(y_test,y_prediction)
        print('r2 score: {}'.format(r2_score))
        print('MAE:',metrics.mean_absolute_error(y_test,y_prediction))
        print('MSE:',metrics.mean_squared_error(y_test,y_prediction))
        print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,y_prediction)))
        sns.distplot(y_test-y_prediction)
        print('\n')
        print(df)

        if dump==1:
            ##dump your model using pickle so that we will re-use
            file=open('model.pkl','wb')
            pickle.dump(model,file)
```

Function predict()

```
[ ] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

```
[ ] from sklearn import metrics
    ##dump your model using pickle so that we will re-use
    import pickle
    def predict(ml_model,dump):
        model=ml_model.fit(X_train,y_train)
        print('Training score : {}'.format(model.score(X_train,y_train)))
        y_prediction=model.predict(X_test)
        df = pd.DataFrame({'Actual': y_test, 'Predicted': y_prediction})
        r2_score=metrics.r2_score(y_test,y_prediction)
        print('r2 score: {}'.format(r2_score))
        print('MAE:',metrics.mean_absolute_error(y_test,y_prediction))
        print('MSE:',metrics.mean_squared_error(y_test,y_prediction))
        print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,y_prediction)))
        sns.distplot(y_test-y_prediction)
        print('\n')
        print(df)

        if dump==1:
            ##dump your model using pickle so that we will re-use
            file=open('model.pkl','wb')
            pickle.dump(model,file)
```

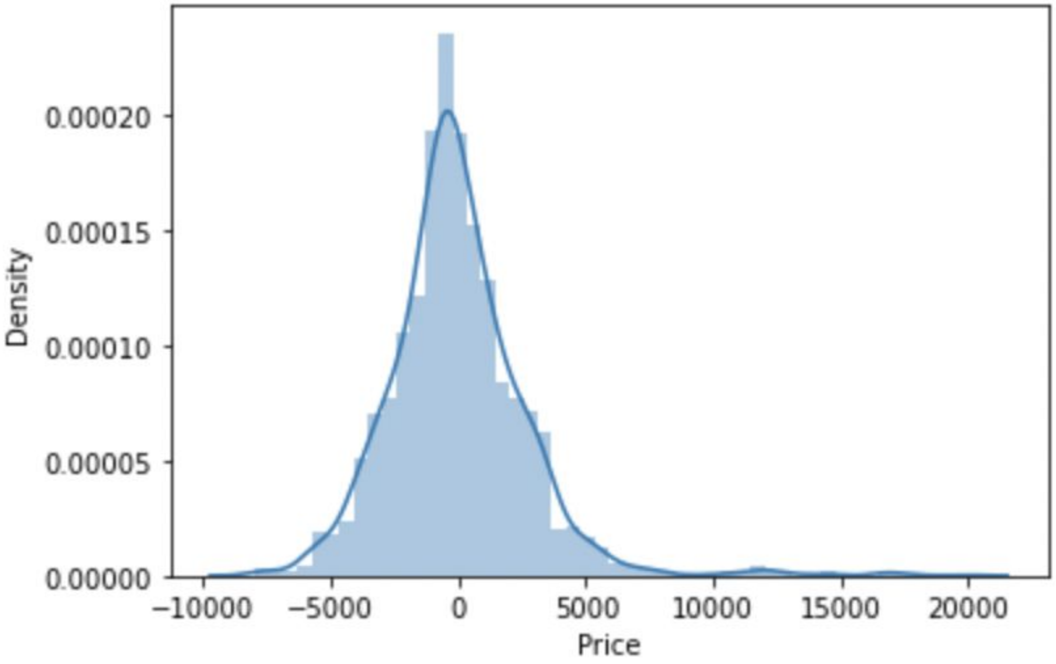
Creating predict function to pass all the models in for prediction of price

Linear Regression

```
predict(LinearRegression(),1)
```

```
Training score : 0.614552623000792
r2 score: 0.6198821088302715
MAE: 1905.8563174482993
MSE: 7273979.8175415145
RMSE: 2697.0316678788763
```

	Actual	Predicted
8998	13067.0	11452.329714
3646	3807.0	5070.146688
6416	13817.0	14258.609919
2371	7845.0	10173.194801
10056	10844.0	11922.678207
...	...	...
3960	10844.0	11590.386320
235	4409.0	5185.366849
1347	17057.0	11998.052529
4976	4804.0	4031.888676
993	4990.0	3580.542926



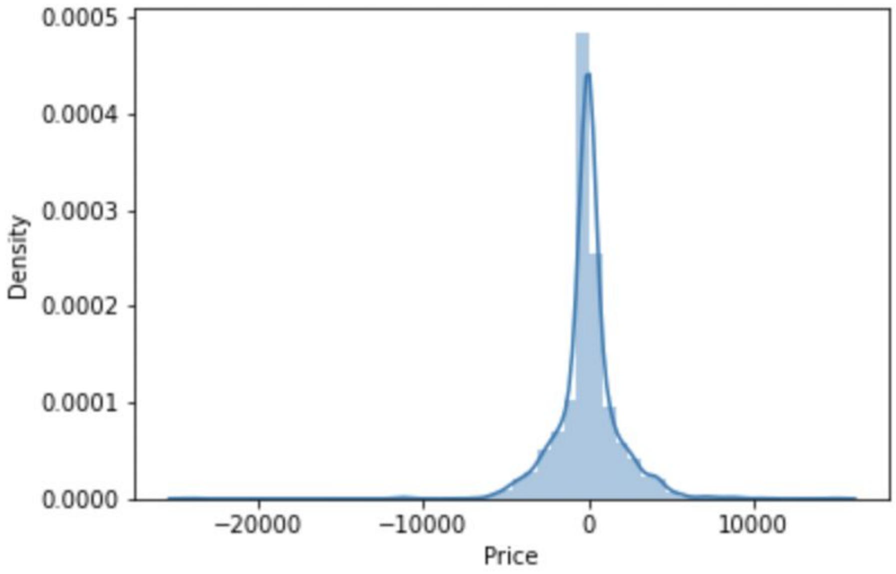
Apply Random Forest Model

```
predict(RandomForestRegressor(),1)
```

```
Training score : 0.9538271440941618
r2 score: 0.8081810239275149
MAE: 1153.3750860557577
MSE: 3670670.0552269397
RMSE: 1915.899281075845
```

	Actual	Predicted
8998	13067.0	12476.445333
3646	3807.0	4105.210000
6416	13817.0	13169.575000
2371	7845.0	8136.807500
10056	10844.0	10952.620000
...	...	...
3960	10844.0	13601.216667
235	4409.0	4683.280000
1347	17057.0	16607.510833
4976	4804.0	4724.210000
993	4990.0	4957.330000

[2137 rows x 2 columns]

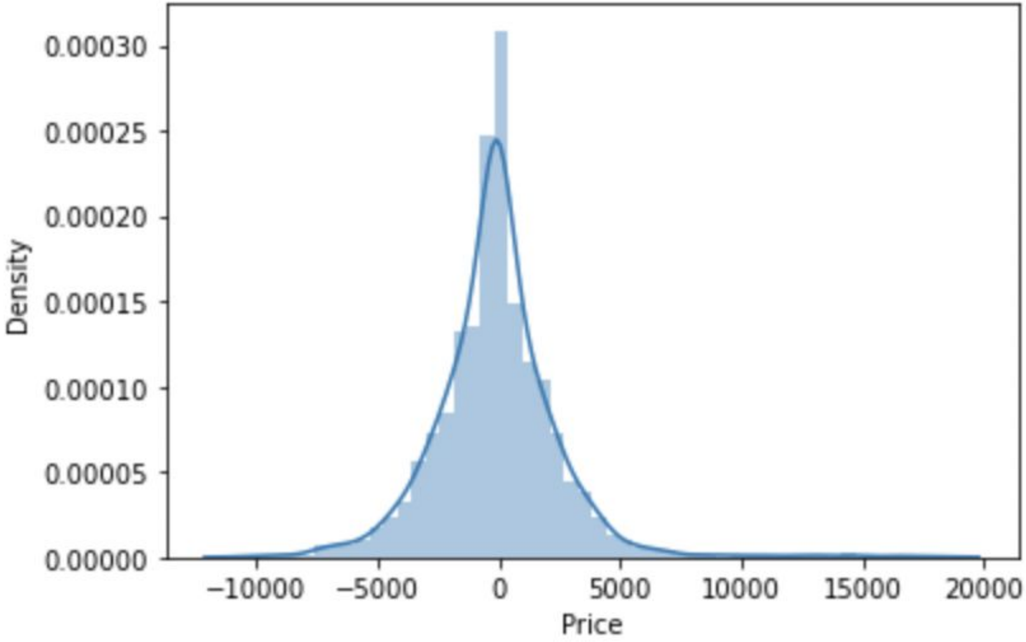


Apply KNN

```
predict(KNeighborsRegressor(),1)
```

```
Training score : 0.7813186018339361
r2 score: 0.65462899281159
MAE: 1730.280205896116
MSE: 6609059.437117455
RMSE: 2570.8091016482445
```

	Actual	Predicted
8998	13067.0	11693.6
3646	3807.0	4223.2
6416	13817.0	13271.0
2371	7845.0	10538.8
10056	10844.0	11578.8
...	...	...
3960	10844.0	12355.6
235	4409.0	4799.6
1347	17057.0	19400.6
4976	4804.0	4678.0
993	4990.0	4889.8

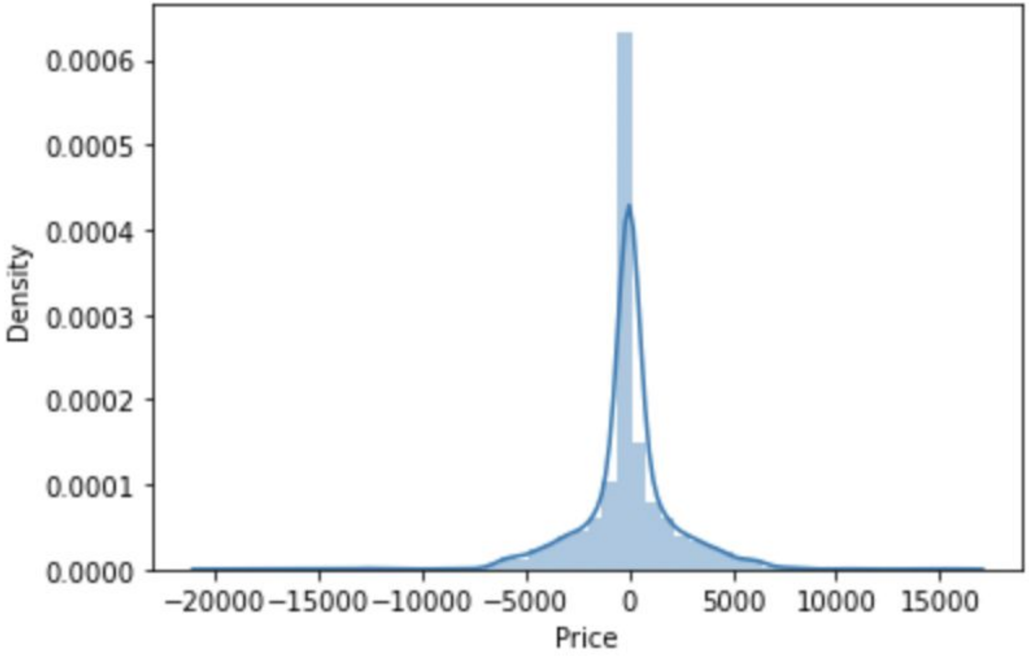


Apply Decision Tree

```
predict(DecisionTreeRegressor(),0)
```

```
Training score : 0.9666309298021185
r2 score: 0.7406579706620775
MAE: 1287.752456715021
MSE: 4962798.992278895
RMSE: 2227.7340488215586
```

	Actual	Predicted
8998	13067.0	13067.0
3646	3807.0	4107.0
6416	13817.0	13817.0
2371	7845.0	6148.0
10056	10844.0	10844.0
...	...	...
3960	10844.0	14781.0
235	4409.0	4409.0
1347	17057.0	17057.0
4976	4804.0	4804.0
993	4990.0	4823.0



Hyperparameter Tuning

1. Choose following method for hyperparameter tuning

    a. RandomizedSearchCV --> Fast way to Hypertune model

    b. GridSearchCV--> Slow way to hypertune my model

2. Assign hyperparameters in form of dictionary

3. Fit the model

4. Check best parameters and best score

Hyperparameters in form of dictionary

```
[ ]  # Number of trees in random forest
     n_estimators=[int(x) for x in np.linspace(start=100,stop=1200,num=6)]

     # 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=4)]

     # Minimum number of samples required to split a node
     min_samples_split=[5,10,15,100]
```

```
[ ]  # Create the random grid

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

Save The Model For Reuse

15) Save the model and reuse again

```
[ ] import pickle
```

```
[ ] # open a file, where you want to store the data
    file=open('rf_random.pkl','wb')
```

```
[ ] # dump information to that file
    pickle.dump(rf_random,file)
```

```
[ ] model=open('rf_random.pkl','rb')
    forest=pickle.load(model)
```

```
[ ] y_prediction=forest.predict(X_test)
```

```
[ ] y_prediction

    array([12171.47246098,  4274.17636284, 13015.10240681, ...,
           16271.70527847,  4769.46051739,  4730.27811033])
```

```
[ ] metrics.r2_score(y_test,y_prediction)

    0.835810519999809
```

Conclusion

The conclusions/inferences drawn from data analysis of the data have been mentioned before in the presentation

- The price depends most on the Total Stops and the route taken
- The models of Random Forest and Decision Tree seem to give the best accuracy for prediction with 84% and 76% accuracy respective

Accuracy (R2) and Training scores for all the models used:

1. Random Forest
   - Accuracy: 95 %
   - Training score: 84%
2. Decision Tree
   - Accuracy: 96 %
   - Training score: 76%
3. KNN
   - Accuracy: 67 %
   - Training score: 67%
4. Linear Regression
   - Accuracy: 62%
   - Training score: 61%

Conclusion

	Linear Regression	Random Forest	K Nearest Neighbour	Decision Tree
Training Score	0.6145	0.9538	0.78131	0.96663
R2 score	0.6198	0.8081	0.6546	0.7406
MAE	1905.8563	1153.3750	1730.2802	1287.7524
MSE	7273979.8175	3670670.0552	6609059.4371	4962798.992
RMSE	2697.0316	1915.8992	2570.8091	2227.7340

- **Decision Tree Model** has the best training score.
- **Random Forest Model** has the best R2 score (accuracy of price prediction) and the least value of errors: Mean Absolute Error, Mean Squared Error and Root Mean Squared Error.

Thank You