FLIP ROBO

Flight: Price Prediction

Submitted by:

Dipak Someshwar

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to SME. Sapna Verma.

We are highly indebted to Flip Robo technology for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I thank and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

Thanks all.

Dipak Someshwar

INTRODUCTION

- Anyone who has booked a flight ticket knows how unexpectedly the prices vary.

- The cheapest available ticket on a given flight gets more and less expensive over time.

- This usually happens as an attempt to maximize revenue based on -

1. Time of purchase patterns (making sure last-minute purchases are expensive)

2. Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases)

1. So, you have to work on a project where you collect data of flight fares with other features and work to make a model to predict fares of flights.

1. **Data Collection Phase:**

   In this section scrape the data of flights from different websites (yatra.com, skyscanner.com, official websites of airlines, etc).

   The number of columns for data doesn't have limit, it's up to you and your creativity.

   Generally, these columns are airline name, date of journey, source, destination, route, departure time, arrival time, duration, total stops and the target variable price.

   You can make changes to it, you can add or you can remove some columns, it completely depends on the website from which you are fetching the data.

2. **Data Analysis :**

   After cleaning the data, you have to do some analysis on the data.

   Do airfares change frequently?

   Do they move in small increments or in large jumps?

   Do they tend to go up or down over time? What is the best time to buy so that the consumer can save the most by taking the least risk?

   Does price increase as we get near to departure date?

   Is Indigo cheaper than Jet Airways?

   Are morning flights expensive?

3. Model Building :

After collecting the data, build a machine learning model. Before model building do all data pre-processing steps.

Try different models with different hyper parameters and select the best model.

1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best model

Analytical Problem Framing

- Import library and load the dataset.

Import the libraries.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Load the dataset.

In [2]:
```
import pandas as pd
df = pd.read_excel(r'flight_data_final.xlsx')
df
```

Out[2]:

	Airline_name	Date_of_Journey	Source	Destination	Departure_time	Arrival_time	Duration	Total_stops	Price
0	Jet Airways	9/05/2022	Kolkata	Banglore	09:35	10:55:00	25h 20m	1 stop	10953
1	SpiceJet	3/04/2022	Kolkata	Banglore	17:10	19:40:00	2h 30m	non-stop	3841
2	IndiGo	1/06/2022	Delhi	Cochin	05:10	10:05:00	4h 55m	1 stop	6842
3	Jet Airways	27/05/2022	Delhi	Cochin	09:00	12:35:00	27h 35m	1 stop	12898
4	Vistara	7/09/2022	Mumbai	Bangalore	17:35	09:00	15h 25m	1 stop	9280
...	...	...	...	...	...	...	...	...	...
1530	SpiceJet	21/03/2022	Banglore	New Delhi	10:20	18:15:00	7h 55m	1 stop	7139
1531	Jet Airways	27/04/2022	Banglore	Delhi	06:00	08:45:00	2h 45m	non-stop	4544
1532	IndiGo	10/11/2022	Pune	Bangalore	22:20	23:55	1h 35m	non-stop	5418
1533	IndiGo	18/05/2022	Delhi	Cochin	08:35	16:10:00	7h 35m	1 stop	6442
1534	Air India	3/06/2022	Delhi	Cochin	03:50	19:15:00	15h 25m	1 stop	8669

1535 rows × 9 columns

In [3]:
```
# Get the numbers of rows and columns.
df.shape
```

Out[3]: (1535, 9)

- Display all column name of dataset.

In [4]:
```
# Check column of the dataframe.
df.columns
```

Out[4]:
```
Index(['Airline_name', 'Date_of_Journey', 'Source', 'Destination',
       'Departure_time', 'Arrival_time', 'Duration', 'Total_stops', 'Price'],
      dtype='object')
```

- Display datatypes and sum of null values.

In [5]:
```
# Get the column datatypes.
df.dtypes
```

Out[5]:
```
Airline_name       object
Date_of_Journey    object
Source             object
Destination        object
Departure_time     object
Arrival_time       object
Duration           object
Total_stops        object
Price               int64
dtype: object
```

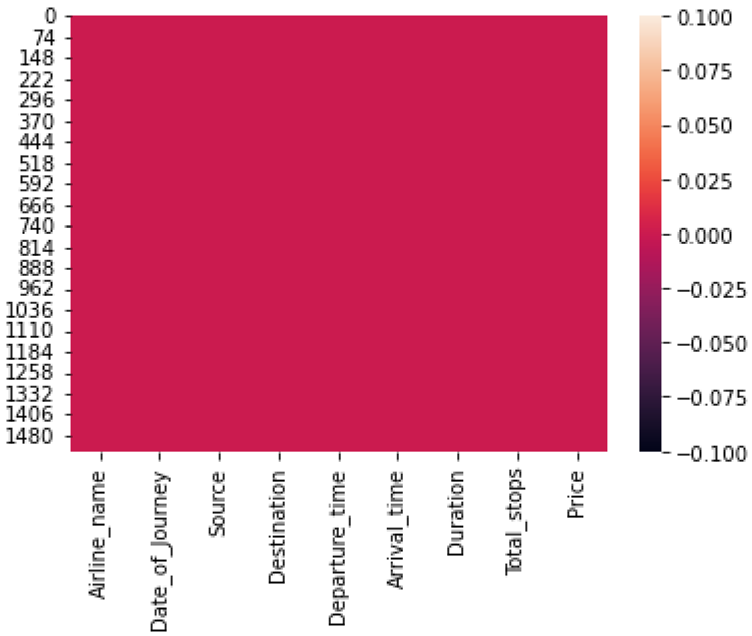
In [7]:
```
# Get a count of the empty values for each column.
df.isna().sum()
```

Out[7]:
```
Airline_name       0
Date_of_Journey    0
Source             0
Destination        0
Departure_time     0
Arrival_time       0
Duration           0
Total_stops        0
Price              0
dtype: int64
```

- Display null values of columns using heatmap.

In [9]:
```
#Checking for null values using heatmap.
sns.heatmap(df.isnull())
```

Out[9]:
```
<AxesSubplot:>
```



- Display statistical summary.

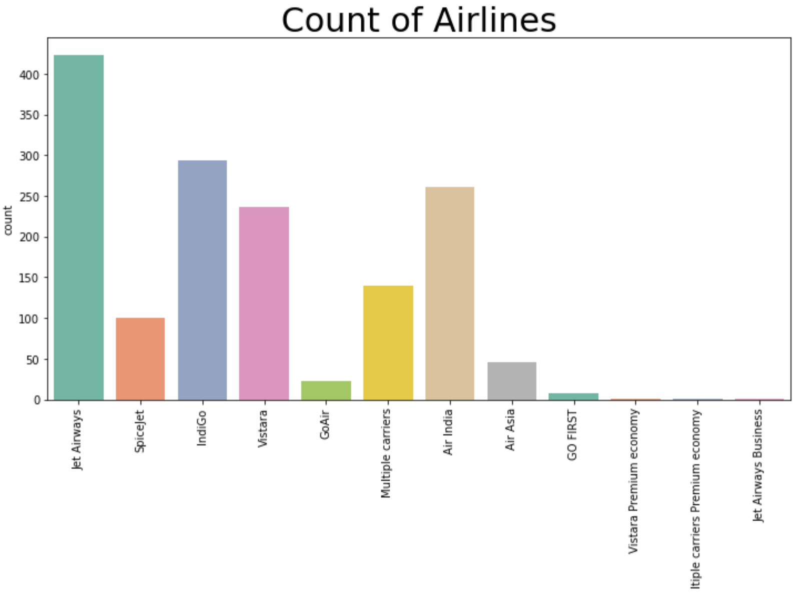
In [16]:
```
# Summary statistics:
df.describe().style.background_gradient()
```

Out[16]:

	Price
count	1533.000000
mean	9229.936725
std	4485.641009
min	1965.000000
25%	5418.000000
50%	9211.000000
75%	11789.000000
max	52229.000000

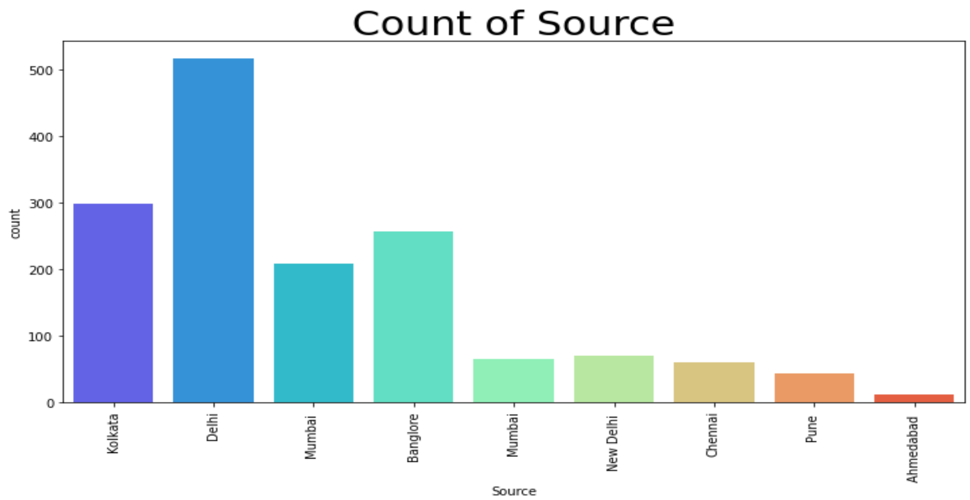
- Display count of Airline column.

In [17]:
```
# Here ploting Count of Airlines :
plt.figure(figsize=(12,6))
sns.countplot(df['Airline_name'], palette='Set2')
plt.title('Count of Airlines', size=30)
plt.xticks(rotation=90)
plt.show()
```



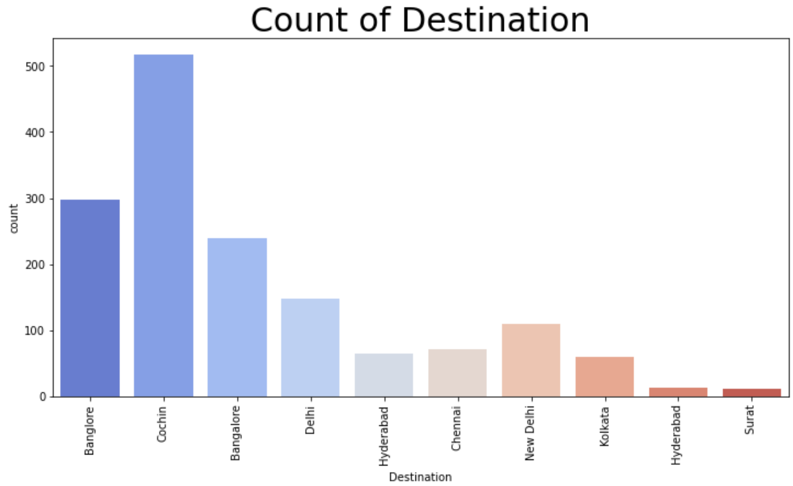
- Display count of Source column.

In [19]:
```
# Here ploting Count of Source :
plt.figure(figsize=(12,6))
sns.countplot(df['Source'], palette='rainbow')
plt.title('Count of Source', size=30)
plt.xticks(rotation=90)
plt.show()
```



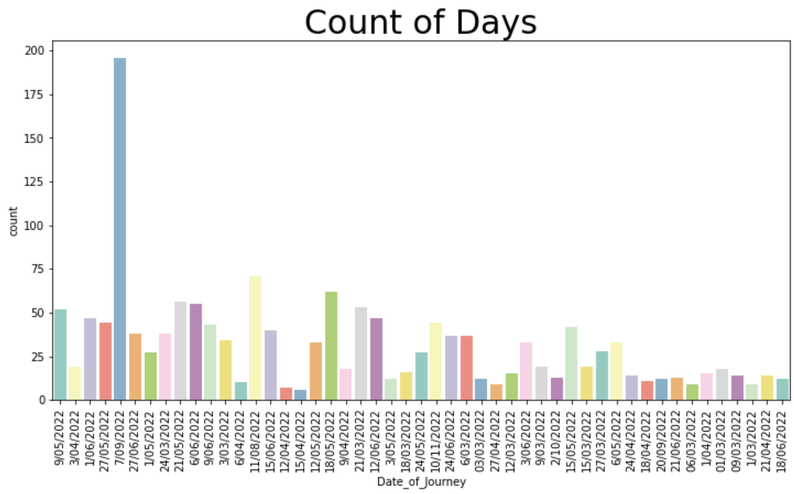
- Display count of Destination column.

In [20]:
```
# Here ploting Count of Source :
plt.figure(figsize=(12,6))
sns.countplot(df['Destination'], palette='coolwarm')
plt.title('Count of Destination', size=30)
plt.xticks(rotation=90)
plt.show()
```



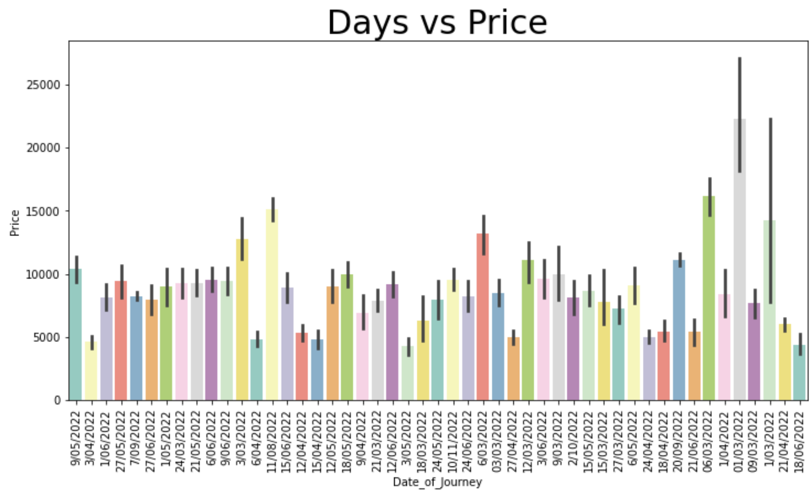
- Display count of Days.

In [21]:
```
# Here ploting Count of Days :
plt.figure(figsize=(12,6))
sns.countplot(df['Date_of_Journey'], palette='Set3')
plt.title('Count of Days', size=30)
plt.xticks(rotation=90)
plt.show()
```



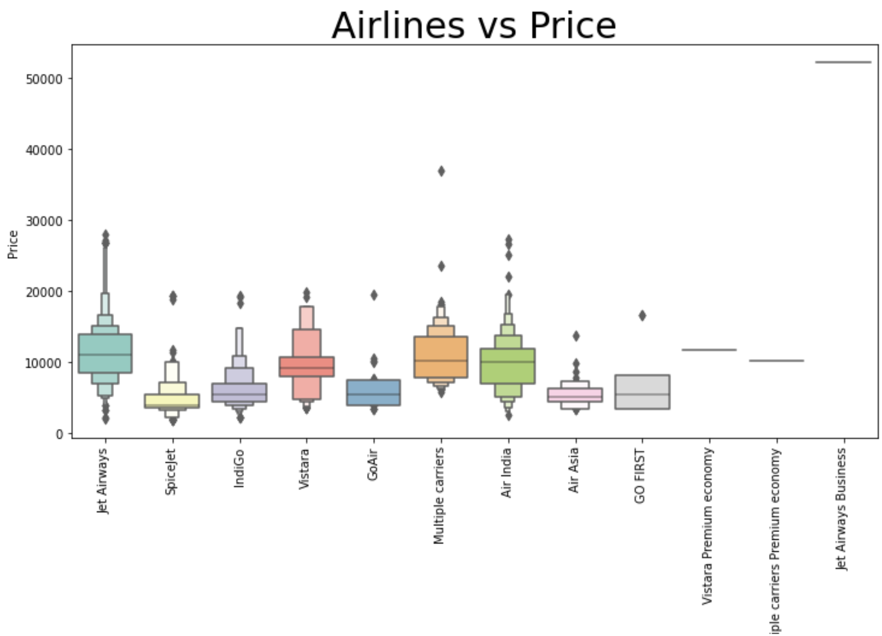
- Display Days vs Price.

```
In [22]:
# Here Plotting days vs price plot:
plt.figure(figsize=(12,6))
sns.barplot(df['Date_of_Journey'], df['Price'], palette='Set3')
plt.title('Days vs Price', size=30)
plt.xticks(rotation=90)
plt.show()
```



- Display Airline vs Price.

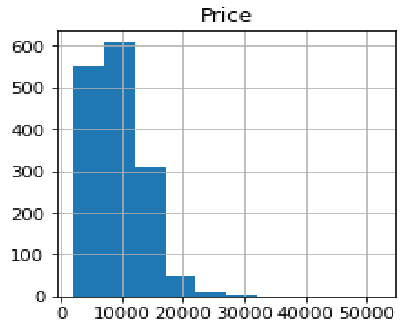
```
In [23]:
# Here Plotting Price vs Airline plot:
plt.figure(figsize=(12,6))
sns.boxenplot(df['Airline_name'], df['Price'],palette='Set3')
plt.title('Airlines vs Price', size=30)
plt.xticks(rotation=90)
plt.show()
```



- Display Histogram of Price.

```
In [24]:
# display histogram:
df.hist(figsize=(12,12), layout=(3,3), sharex=False)
```

```
Out[24]: array([[<AxesSubplot:title={'center':'Price'}>, <AxesSubplot:>,
                <AxesSubplot:>],
               [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>],
               [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



- Perform Feature Engineering.

Feature Engineering

```
In [30]:
# Here converting the hours in minutes:
df['Duration'] = df['Duration'].str.replace("h", '*60').str.replace(' ','+').str.replace('m','*1').apply(eval)
```

```
In [32]:
# Here we are organizing the format of the date of journey in our dataset for better:
df["Journey_day"] = df['Date_of_Journey'].str.split('/').str[0].astype(int)
df["Journey_month"] = df['Date_of_Journey'].str.split('/').str[1].astype(int)
df.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

```
In [34]:
# Converting departure time into hours and minutes:
df["Dep_hour"] = pd.to_datetime(df["Departure_time"]).dt.hour
df["Dep_min"] = pd.to_datetime(df["Departure_time"]).dt.minute
df.drop(["Departure_time"], axis = 1, inplace = True)
```

```
In [35]:
# Here converting the arrival time into hours and minutes:
df["Arrival_hour"] = pd.to_datetime(df.Arrival_time).dt.hour
df["Arrival_min"] = pd.to_datetime(df.Arrival_time).dt.minute
df.drop(["Arrival_time"], axis = 1, inplace = True)
```

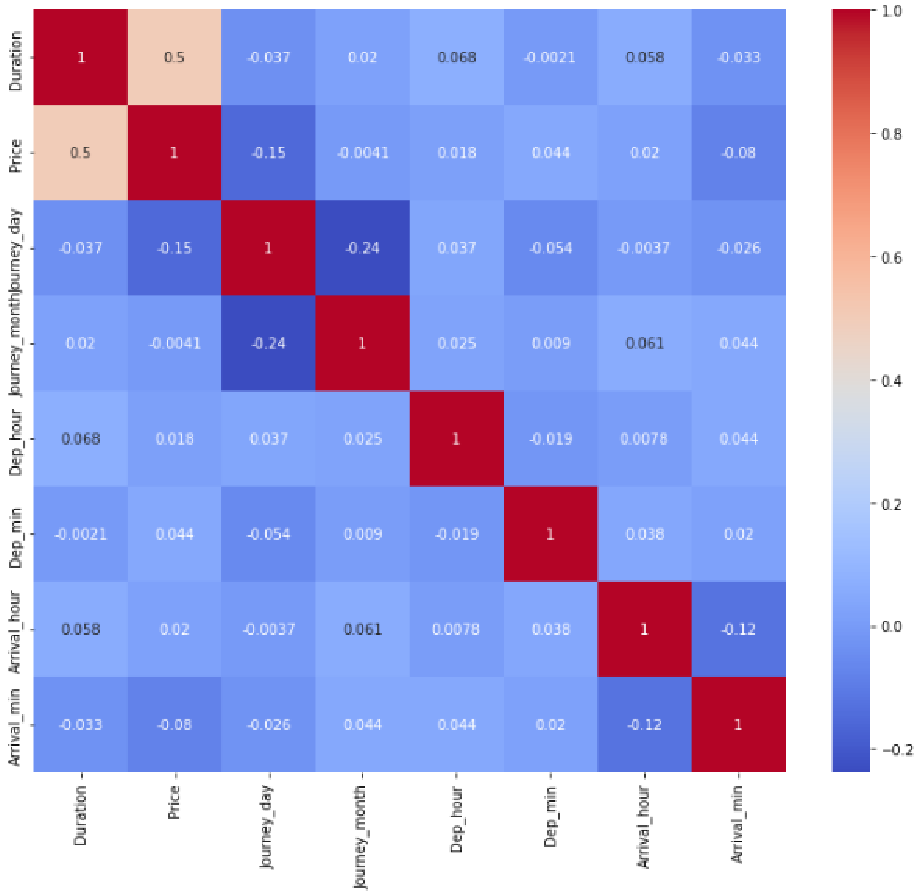
```
In [36]:
# Showing dataframe after performing feature engineering.
df.head()
```

Out[36]:

	Airline_name	Source	Destination	Duration	Total_stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	Jet Airways	Kolkata	Banglore	1520	1 stop	10953	9	5	9	35	10	55
1	SpiceJet	Kolkata	Banglore	150	non-stop	3841	3	4	17	10	19	40
2	IndiGo	Delhi	Cochin	295	1 stop	6842	1	6	5	10	10	5
3	Jet Airways	Delhi	Cochin	1655	1 stop	12898	27	5	9	0	12	35
4	Vistara	Mumbai	Bangalore	925	1 stop	9280	7	9	17	35	9	0

- Display correlation of columns using heatmap.

```
In [38]:
plt.figure(figsize = (12,10))
sns.heatmap(df.corr(), annot = True, cmap = "coolwarm")
plt.show()
```

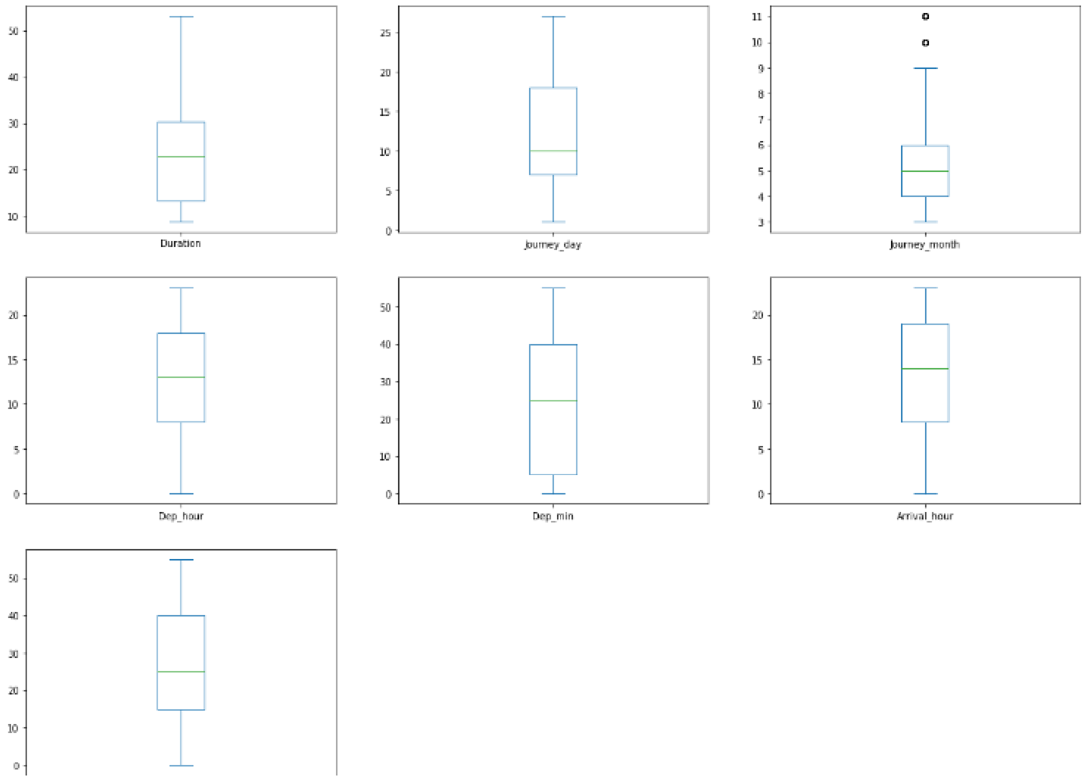


	Duration	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
Duration	1	0.5	-0.037	0.02	0.068	-0.0021	0.058	-0.033
Price	0.5	1	-0.15	-0.0041	0.018	0.044	0.02	-0.08
Journey_day	-0.037	-0.15	1	-0.24	0.037	-0.054	-0.0037	-0.026
Journey_month	0.02	-0.0041	-0.24	1	0.025	0.009	0.061	0.044
Dep_hour	0.068	0.018	0.037	0.025	1	-0.019	0.0078	0.044
Dep_min	-0.0021	0.044	-0.054	0.009	-0.019	1	0.038	0.02
Arrival_hour	0.058	0.02	-0.0037	0.061	0.0078	0.038	1	-0.12
Arrival_min	-0.033	-0.08	-0.026	0.044	0.044	0.02	-0.12	1

- Display outliers of all columns.

In [48]:
```
# Checking outliers with boxplot:
numerical_data.plot(kind='box', subplots=True, layout=(3,3), figsize=(20,15))
```

Out[48]:
```
Duration          AxesSubplot(0.125,0.657941;0.227941x0.222059)
Journey_day     AxesSubplot(0.398529,0.657941;0.227941x0.222059)
Journey_month   AxesSubplot(0.672059,0.657941;0.227941x0.222059)
Dep_hour          AxesSubplot(0.125,0.391471;0.227941x0.222059)
Dep_min         AxesSubplot(0.398529,0.391471;0.227941x0.222059)
Arrival_hour    AxesSubplot(0.672059,0.391471;0.227941x0.222059)
Arrival_min           AxesSubplot(0.125,0.125;0.227941x0.222059)
dtype: object
```



- Data Pre-processing and Scalling the data.

Data Preprocessing (split the data into independent 'x' and dependent 'y' datasets)

In [52]:
```
# Here Concatenating both Categorical Data and Numerical Data:
x = pd.concat([categorical_data, numerical_data], axis=1)
y = df['Price']
```

In [54]:
```
x.head()
```

Out[54]:

	Airline_name	Source	Destination	Total_stops	Duration	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	5	4	4	0	38.987177	9	5	9	35	10	55
1	9	4	4	3	12.247449	3	4	17	10	19	40
2	4	3	5	0	17.175564	1	6	5	10	10	5
3	5	3	5	0	40.681691	27	5	9	0	12	35
4	10	6	0	0	30.413813	7	9	17	35	9	0

In [55]:
```
y.head()
```

Out[55]:
```
0    10953
1     3841
2     6842
3    12898
4     9280
Name: Price, dtype: int64
```

In [56]:
```
print(x.shape, y.shape)
```

```
(1533, 11) (1533,)
```

Scaling the data.

In [57]:
```
from sklearn.preprocessing import StandardScaler

st = StandardScaler()
x = st.fit_transform(x)
x
```

Out[57]:
```
array([[-0.05204319,  0.20716433, -0.15155037, ...,  0.54456632,
        -0.50335041,  1.72537747],
       [ 1.27766042,  0.20716433, -0.15155037, ..., -0.78265175,
         0.81543531,  0.84125368],
       [-0.3844691 , -0.3311118 ,  0.24558931, ..., -0.78265175,
        -0.50335041, -1.22170182],
       ...,
       [-0.3844691 ,  2.36026881, -1.74010908, ..., -0.25176452,
         1.4015623 ,  1.72537747],
       [-0.3844691 , -0.3311118 ,  0.24558931, ...,  0.54456632,
         0.37584007, -0.92699389],
       [-1.38174681, -0.3311118 ,  0.24558931, ...,  1.34089716,
         0.81543531, -0.63228596]])
```

- Run and evaluate selected models.

Finding best random_state

In [58]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()

from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()

from sklearn.ensemble import AdaBoostRegressor
abr = AdaBoostRegressor()

from sklearn.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor()

from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()

from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [59]:

```
model = [lr,rf,abr,gbr,dtr]
max_r2_score = 0
for r_state in range(0,100):
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.20,random_state = r_state)
    for i in model:
        i.fit(x_train,y_train)
        pred_test = i.predict(x_test)
        r2_sc = r2_score(y_test,pred_test)
        print("R2 score correspond to random state ",r_state,"is",r2_sc)
        if r2_sc > max_r2_score:
            max_r2_score = r2_sc
            final_state = r_state
            final_model = i
```

```
R2 score correspond to random state  9 is 0.6196406794640958
R2 score correspond to random state  9 is 0.3822601727419733
R2 score correspond to random state  10 is 0.44376074740412497
R2 score correspond to random state  10 is 0.7877210630853698
R2 score correspond to random state  10 is 0.4924833375519503
R2 score correspond to random state  10 is 0.7700066152629447
R2 score correspond to random state  10 is 0.5325174298757969
R2 score correspond to random state  11 is 0.46852978946636037
R2 score correspond to random state  11 is 0.7388002704805672
R2 score correspond to random state  11 is 0.5397650556851779
R2 score correspond to random state  11 is 0.7145815649700709
R2 score correspond to random state  11 is 0.43000474467977345
R2 score correspond to random state  12 is 0.39447416751259723
R2 score correspond to random state  12 is 0.8061866381796754
R2 score correspond to random state  12 is 0.5736027845522467
R2 score correspond to random state  12 is 0.709091996036882
R2 score correspond to random state  12 is 0.5410057640727091
R2 score correspond to random state  13 is 0.4178068600662399
R2 score correspond to random state  13 is 0.7345934403372618
R2 score correspond to random state  13 is 0.5151725353435027
```

In [60]:

```
print("max R2 score correspond to random state ",final_state,"is",max_r2_score,"and model is",final_model)
```

```
max R2 score correspond to random state  79 is 0.8307051042076956 and model is RandomForestRegressor()
```

Creating train-test split:

In [61]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.20,random_state = 79)
```

Apply best model:

In [62]:
```
rf.fit(x_train,y_train)
pred = rf.predict(x_test)
print("r2 score :- ",r2_score(y_test,pred))
```

```
r2 score :-  0.8297931346192062
```

In [63]:
```
# Predict the value:
print("predicted ticket price:",pred)
print("actual ticket price",y_test)
```

```
predicted ticket price: [ 3180.86       11339.54       13469.22       5187.15
 12701.6         7005.8         8448.695      14926.715
  2646.035      10070.         10492.43       14472.525
  4537.64       13219.72        6786.39        8555.87
  4462.9825      6542.58       16606.37        7713.72866667
  5887.95       11988.77       11428.82        9268.68
  3846.22       11391.27       10078.43        4627.54666667
  9282.75        3821.58       11603.04714286 11066.12833333
  8357.63        6246.35       12048.24        3866.035
 13268.71        4445.75        9446.03        9280.
  8074.42        7543.49       10697.11       12118.7
 16700.04        7618.94       11386.1        12011.775
 13075.5275     11767.53        4083.69        6830.98
  4426.09666667  4721.95        6553.71       12033.425
 10242.44166667  4703.68       10128.16       12111.16166667
 12271.13       13064.66       12244.82        9083.58166667
 16895.73        6238.31        9596.8025      7911.72
  4789.22        7558.76       12473.825      26440.47
 10070.          4818.16        3881.24        9264.415
 10319.865       4003.66        9280.          8953.49
  4550.72       12229.85       10264.85        8291.75
  5264.38       16749.48        4643.425      13714.39
  7318.43        5211.4         9166.39       12738.16
```

- Checking MAE, MSE and RMSE:

Check MAE, MSE and RMSE:

In [64]:
```
# Display MAE, MSE and RMSE:
from sklearn.metrics import mean_squared_error, mean_absolute_error
print('error:')

print('mean absolute error',mean_absolute_error(y_test,pred))
print('mean squared error',mean_squared_error(y_test,pred))

print('Root mean squared error',np.sqrt(mean_squared_error(y_test,pred)))
```

```
error:
mean absolute error 1240.9198928959206
mean squared error 3241536.9721175395
Root mean squared error 1800.426886079393
```

- Hypertuning of the Model.

Hypertuning of the model:

In [67]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [68]:
```
#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 [69]:
```
parameters = {'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 [70]:
```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = parameters, scoring='neg_mean_squared_error', n_iter =
```

In [71]:
```
rf_random.fit(x_train,y_train)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   0.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   0.3s
```

In [72]:
```
rf_random.best_params_
```

Out[72]:
```
{'n_estimators': 300,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 30}
```

In [73]:
```
rf = RandomForestRegressor(n_estimators=300, min_samples_split=2, min_samples_leaf=1, max_features='sqrt', max_depth=30)
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
pred_decision = rf.predict(x_test)

rfs = r2_score(y_test,pred_decision)
print("R2 score:",rfs*100)

rfscore = cross_val_score(rf,x,y,cv=4)
rfc = rfscore.mean()
print("Cross Val Score is",rfc*100)
```

```
R2 score: 81.46995701280109
Cross Val Score is 71.72954936874227
```

- Hardware and Software Requirements and Tools Used

- **Language :-** Python
- **Tool:-** Jupyter
- **OS:-** Windows 10
- **RAM:-** 8gb

CONCLUSION:

- This Kernel investigates different models for car price prediction.
- Different types of Machine Learning methods including LinearRegression, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor and DecisionTreeRegressor in machine learning are compared and analysed for optimal solutions.
- Even though all of those methods achieved desirable results, different models have their own pros and cons.
- The RandomForestRegressor is probably the best one and has been selected for this problem.
- Finally, the RandomForestRegressor is the best choice when parameterization is the top priority.