```
import pandas as pd
df=pd.read_csv("Flight_Booking.csv")
```

```
df.head()
```

	Unnamed: 0	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	
0	0	SpiceJet	SG-8709	Delhi	Evening	zero	Night	Mumbai	E
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	zero	Morning	Mumbai	E
2	2	AirAsia	I5-764	Delhi	Early_Morning	zero	Early_Morning	Mumbai	E
			UK-						

```
df=df.drop(columns=["Unnamed: 0"])
```

```
df.head()
```

	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	d
0	SpiceJet	SG-8709	Delhi	Evening	zero	Night	Mumbai	Economy	
1	SpiceJet	SG-8157	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	
2	AirAsia	I5-764	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	
3	Vistara	UK-995	Delhi	Morning	zero	Afternoon	Mumbai	Economy	

```
df.shape
```

```
(300153, 11)
```

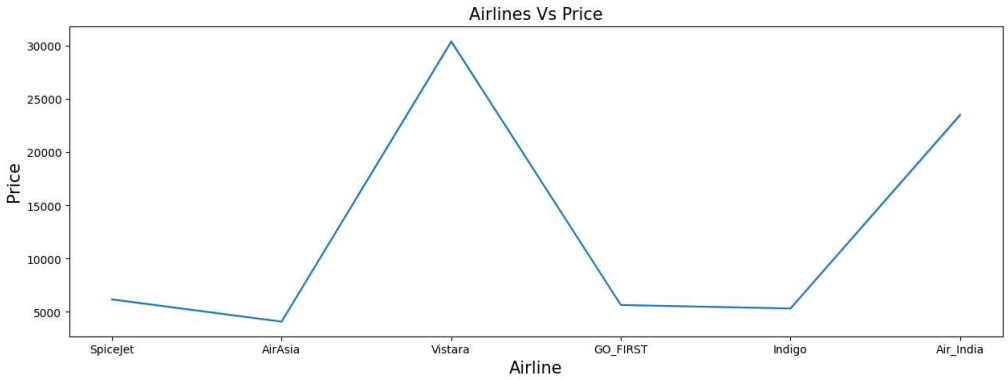
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   airline           300153 non-null  object 
 1   flight            300153 non-null  object 
 2   source_city       300153 non-null  object 
 3   departure_time    300153 non-null  object 
 4   stops             300153 non-null  object 
 5   arrival_time      300153 non-null  object 
 6   destination_city  300153 non-null  object 
 7   class             300153 non-null  object 
 8   duration          300153 non-null  float64
 9   days_left         300153 non-null  int64  
 10  price             300153 non-null  int64  
dtypes: float64(1), int64(2), object(8)
memory usage: 25.2+ MB
```

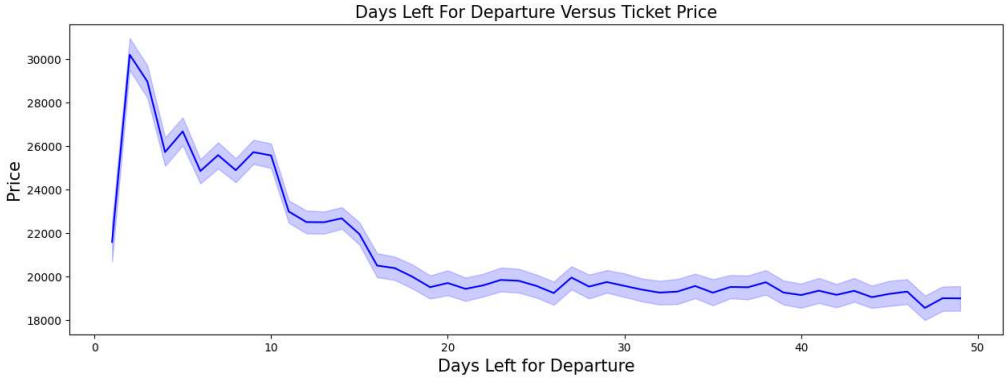
```
df.isnull().sum()
```

```
airline             0
flight              0
source_city         0
departure_time      0
stops               0
arrival_time        0
destination_city    0
class               0
duration            0
days_left           0
price               0
dtype: int64
```

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(15,5))
sns.lineplot(x=df['airline'],y=df['price'])
plt.title('Airlines Vs Price',fontsize=15)
plt.xlabel('Airline',fontsize=15)
plt.ylabel('Price',fontsize=15)
plt.show()
```




```
plt.figure(figsize=(15,5))
sns.lineplot(data=df,x='days_left',y='price',color='blue')
plt.title('Days Left For Departure Versus Ticket Price',fontsize=15)
plt.xlabel('Days Left for Departure',fontsize=15)
plt.ylabel('Price',fontsize=15)
plt.show()
```

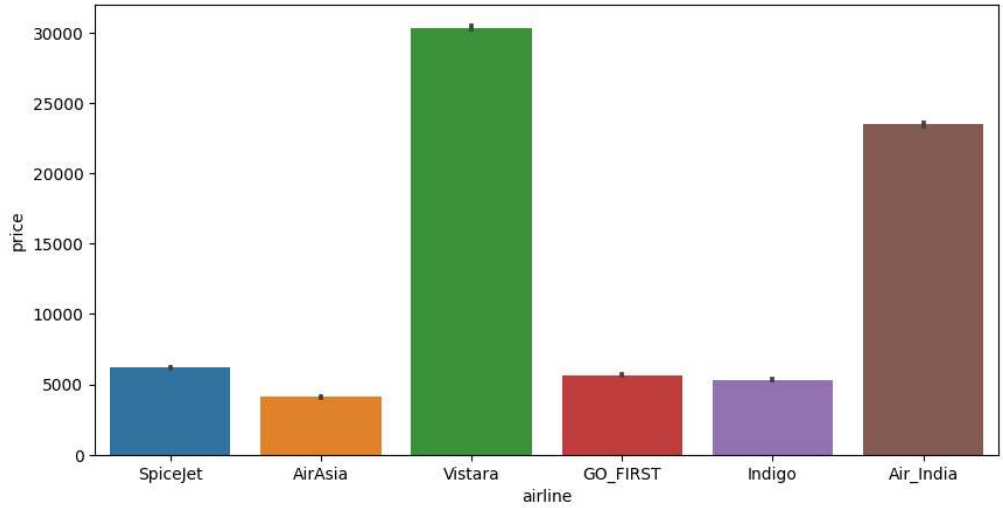



```
df["airline"].value_counts()
```

```
Vistara      127859
Air_India     80892
Indigo        43120
GO_FIRST      23173
AirAsia       16098
SpiceJet       9011
Name: airline, dtype: int64
```

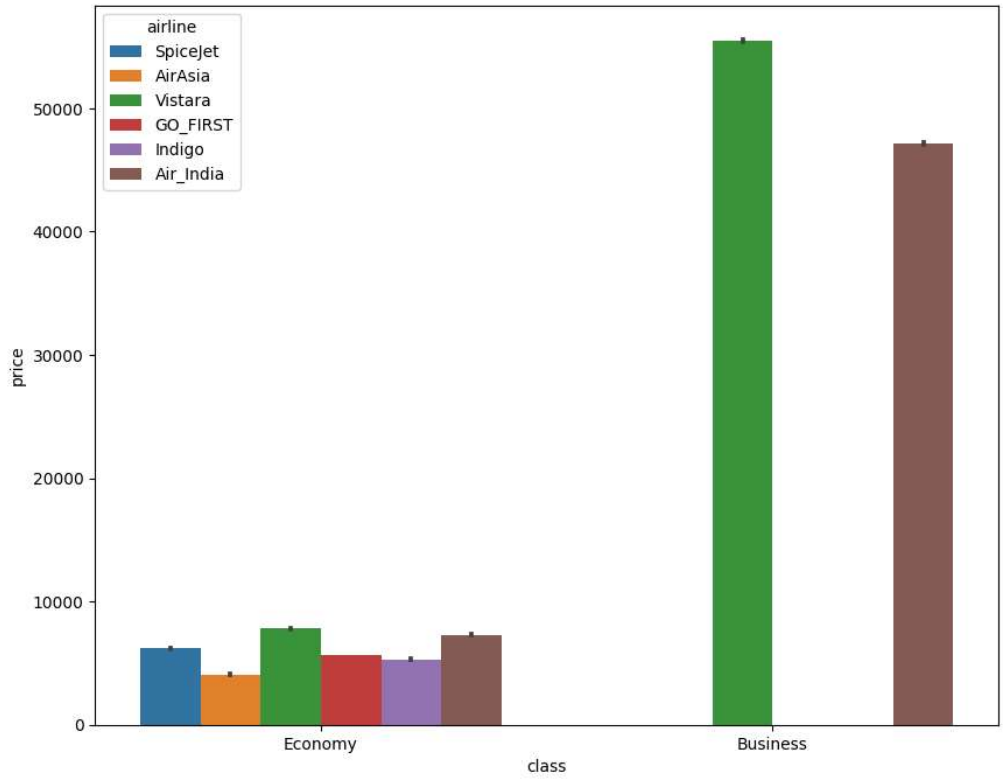
```
plt.figure(figsize=(10,5));
sns.barplot(x='airline',y='price',data=df)
```

```
<Axes: xlabel='airline', ylabel='price'>
```




```
plt.figure(figsize=(10,8));
sns.barplot(x='class',y='price',data=df,hue='airline')
```

```
<Axes: xlabel='class', ylabel='price'>
```

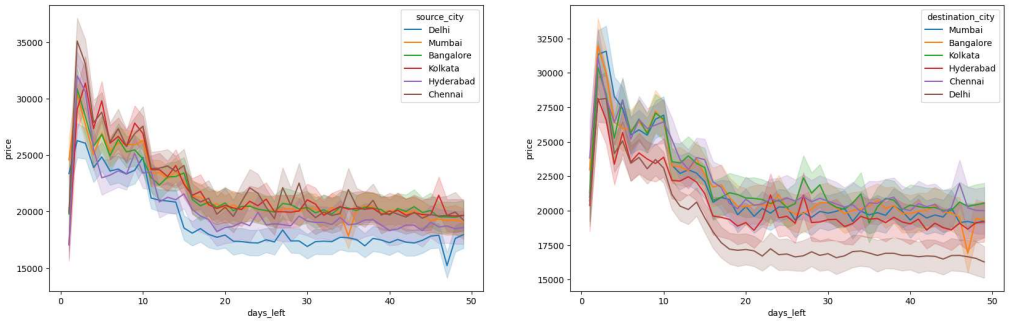



```
df1=df.loc[(df["source_city"]=="Delhi") & (df["destination_city"]=="Mumbai")]
```

```
df1.describe()
```

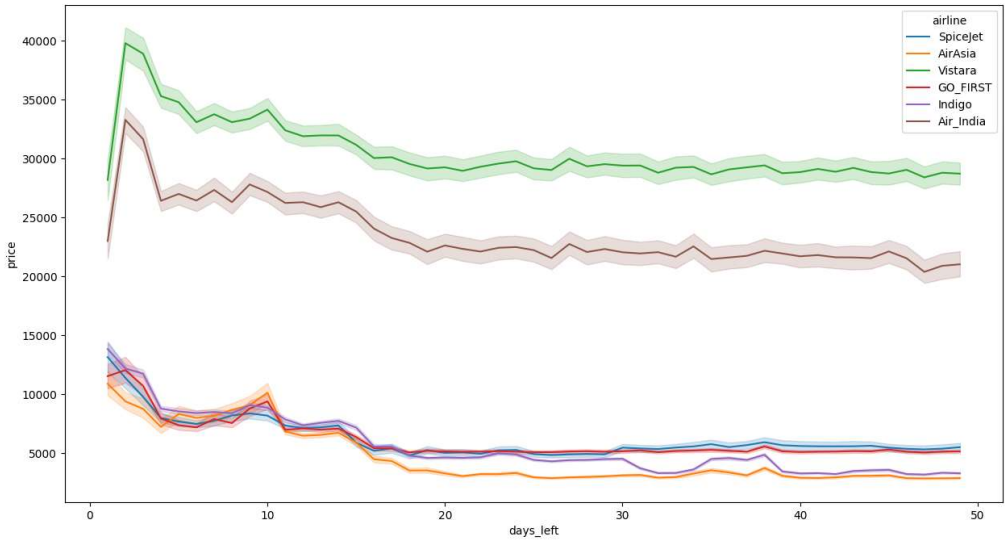
	duration	days_left	price
count	15289.000000	15289.000000	15289.000000

```
fig,ax=plt.subplots(1,2,figsize=(20,6))
sns.lineplot(x='days_left',y='price',data=df,hue='source_city',ax=ax[0])
sns.lineplot(x='days_left',y='price',data=df,hue='destination_city',ax=ax[1])
plt.show()
```



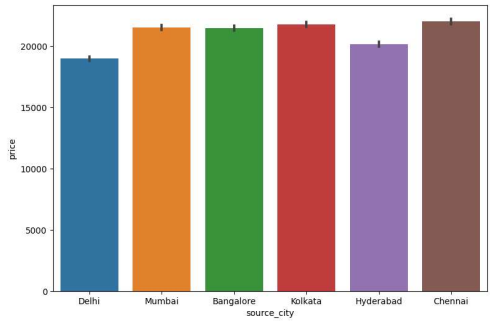

```
plt.figure(figsize=(15,8))
sns.lineplot(x='days_left',y='price',data=df,hue='airline')
```

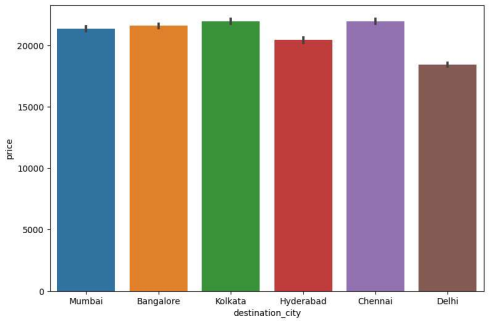
```
<Axes: xlabel='days_left', ylabel='price'>
```



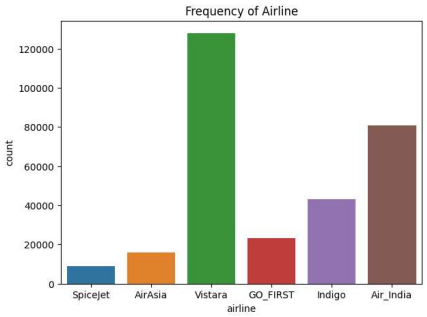

```
fig,ax=plt.subplots(1,2,figsize=(20,6))

sns.barplot(x='source_city',y='price',data=df,ax=ax[0])
sns.barplot(x='destination_city',y='price',data=df,ax=ax[1])
```

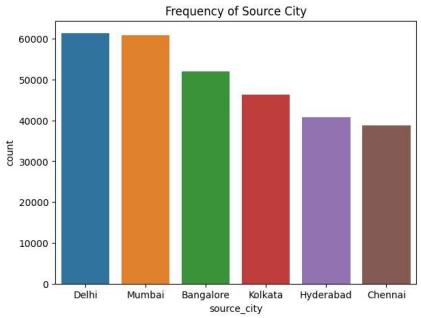
```
<Axes: xlabel='destination_city', ylabel='price'>
```

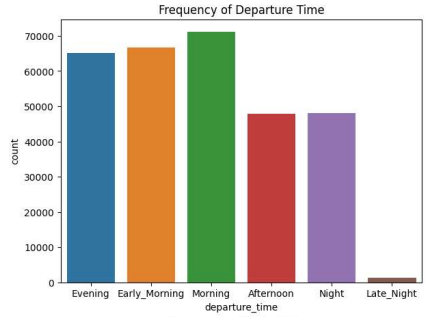


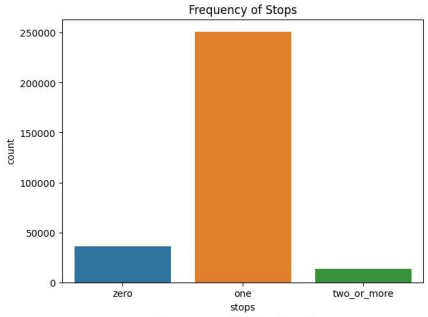


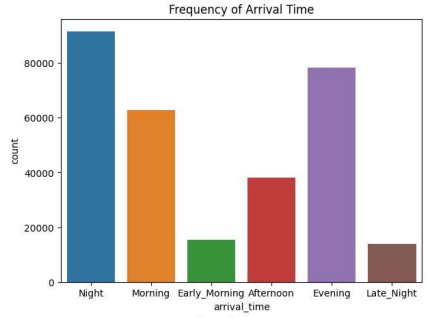

```
# Visualizations of categoric features with countplot
plt.figure(figsize=(15,23))

plt.subplot(4, 2, 1)
sns.countplot(x=df["airline"], data=df)
plt.title("Frequency of Airline")

plt.subplot(4, 2, 2)
sns.countplot(x=df["source_city"], data=df)
plt.title("Frequency of Source City")

plt.subplot(4, 2, 3)
sns.countplot(x=df["departure_time"], data=df)
plt.title("Frequency of Departure Time")

plt.subplot(4, 2, 4)
sns.countplot(x=df["stops"], data=df)
plt.title("Frequency of Stops")

plt.subplot(4, 2, 5)
sns.countplot(x=df["arrival_time"], data=df)
plt.title("Frequency of Arrival Time")

plt.subplot(4, 2, 6)
sns.countplot(x=df["destination_city"], data=df)
plt.title("Frequency of Destination City")

plt.subplot(4, 2, 7)
sns.countplot(x=df["class"], data=df)
plt.title("Class Frequency")

plt.show()
```

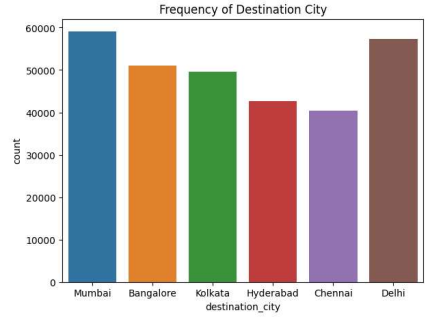


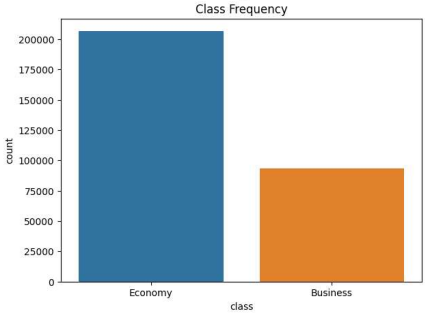












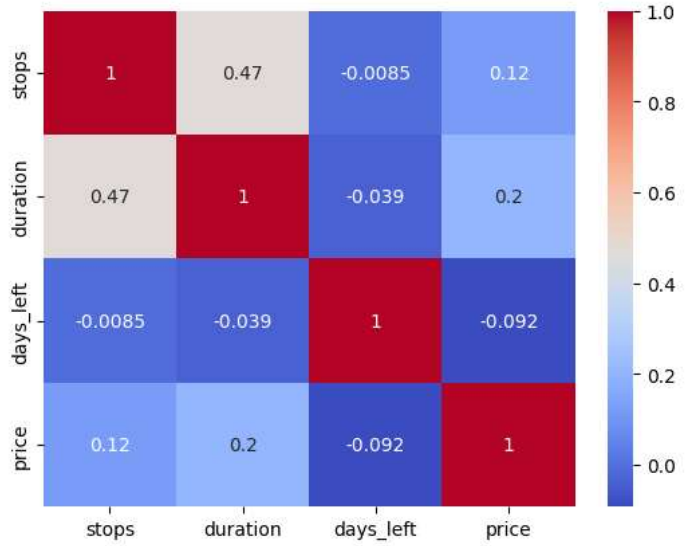

```
sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
```

```
<ipython-input-20-0dd8dea822e2>:1: FutureWarning: The default value of numeric_only
  sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
<Axes: >
```

```
df["stops"]=df["stops"].replace(["zero","one","two_or_more"],[0,1,2])
```

```
sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
```

```
<ipython-input-22-0dd8dea822e2>:1: FutureWarning: The default value of numeric_only in DataFrame.c
  sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
<Axes: >
```




```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df["airline"]=le.fit_transform(df["airline"])
df["source_city"]=le.fit_transform(df["source_city"])
df["departure_time"]=le.fit_transform(df["departure_time"])
df["arrival_time"]=le.fit_transform(df["arrival_time"])
df["destination_city"]=le.fit_transform(df["destination_city"])
df["class"]=le.fit_transform(df["class"])
```

```
df.info()
```

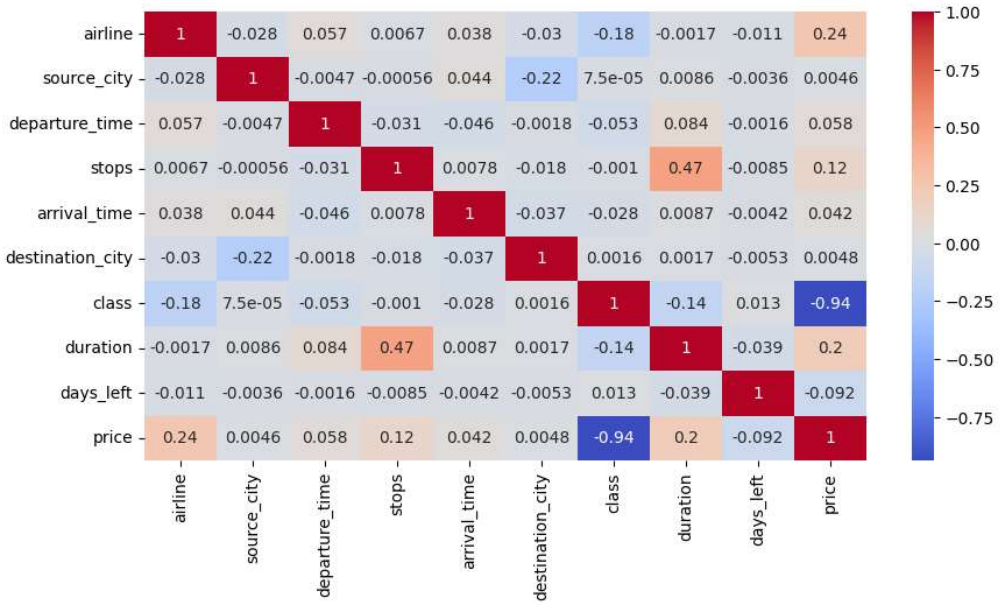
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   airline           300153 non-null  int64
 1   flight            300153 non-null  object
 2   source_city       300153 non-null  int64
 3   departure_time    300153 non-null  int64
 4   stops             300153 non-null  int64
 5   arrival_time      300153 non-null  int64
 6   destination_city  300153 non-null  int64
 7   class             300153 non-null  int64
 8   duration          300153 non-null  float64
 9   days_left         300153 non-null  int64
 10  price             300153 non-null  int64
dtypes: float64(1), int64(9), object(1)
memory usage: 25.2+ MB
```

```
df=df.drop(columns=["flight"])
```

```
df
```

	airline	source_city	departure_time	stops	arrival_time	destination_city	class	duratic
0	4	2	2	0	5	5	1	2.1
1	4	2	1	0	4	5	1	2.3
2	0	2	1	0	1	5	1	2.1
3	5	2	4	0	0	5	1	2.2
4	5	2	4	0	4	5	1	2.3

```
plt.figure(figsize=(10,5))
sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
plt.show()
```




```
from statsmodels.stats.outliers_influence import variance_inflation_factor
col_list = []
for col in df.columns:
    if ((df[col].dtype != 'object') & (col != 'price') ):
        col_list.append(col)

X = df[col_list]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]
print(vif_data)
```

```
              feature       VIF
     0          airline  3.461766
     1      source_city  2.933064
     2   departure_time  2.746367
     3            stops  7.464236
     4     arrival_time  3.684695
     5  destination_city  2.893218
     6            class  2.917521
     7         duration  5.037943
     8        days_left  4.035735
```

```
df=df.drop(columns=["stops"])
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
col_list = []
for col in df.columns:
    if ((df[col].dtype != 'object') & (col != 'price') ):
        col_list.append(col)

X = df[col_list]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]
print(vif_data)
```

```
                feature       VIF
     0           airline  3.370020
     1       source_city  2.895803
     2    departure_time  2.746255
     3      arrival_time  3.632792
     4  destination_city  2.857808
     5             class  2.776721
     6          duration  3.429344
     7         days_left  3.950132
```

```
X = df.drop(columns=["price"])
y = df['price']


from sklearn.linear_model import LinearRegression


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)
```

LINEAR REGRESSION

```
lr=LinearRegression()


from sklearn.preprocessing import StandardScaler


sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)


lr.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_pred=lr.predict(x_test)


from sklearn.metrics import r2_score
r2result= r2_score(y_test,y_pred)
r2_score(y_test,y_pred)
```

```
0.897752737512321
```

```
from sklearn import metrics
mean_abs_error= metrics.mean_absolute_error(y_test,y_pred)
mean_abs_error
```

```
4468.426673542113
```

```
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, y_pred)
```

```
0.34765804610681816
```

```
mean_sq_error=metrics.mean_squared_error(y_test,y_pred)
mean_sq_error
```

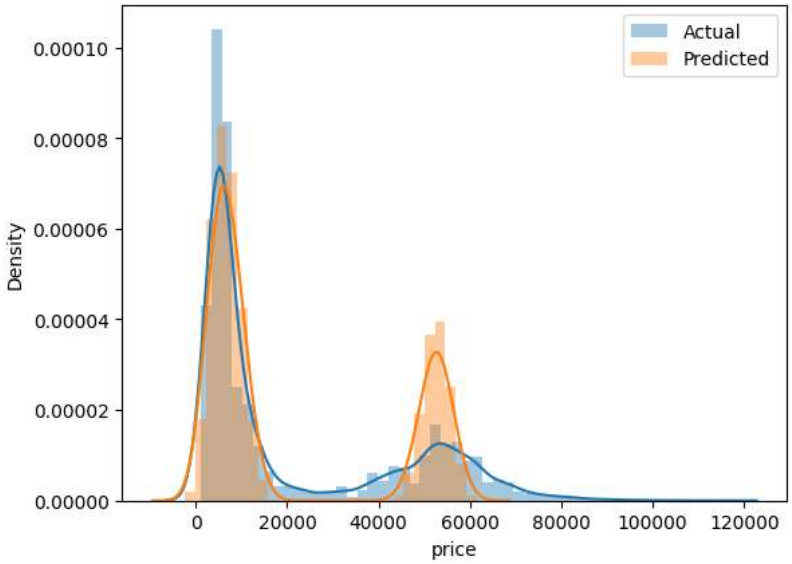
```
52706651.33334208
```

```
import numpy as np
root_mean_sq_error = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
root_mean_sq_error
```

```
7259.934664536733
```

```
sns.distplot(y_test,label="Actual")
sns.distplot(y_pred,label="Predicted")
plt.legend()
```

```
<ipython-input-45-c0da3bcb0fc3>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test,label="Actual")
<ipython-input-45-c0da3bcb0fc3>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_pred,label="Predicted")
<matplotlib.legend.Legend at 0x7f3c1b2ef820>
```



*** DESCISION TREE REGRESSION MODEL***

```
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)
r2_score(y_test,y_pred)
```

```
0.9746608498709947
```

```
mean_abs_error= metrics.mean_absolute_error(y_test,y_pred)
mean_abs_error
```

```
1217.1550254590684
```

```
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, y_pred)
```

```
0.07726568550667867
```

```
mean_sq_error=metrics.mean_squared_error(y_test,y_pred)
mean_sq_error
```

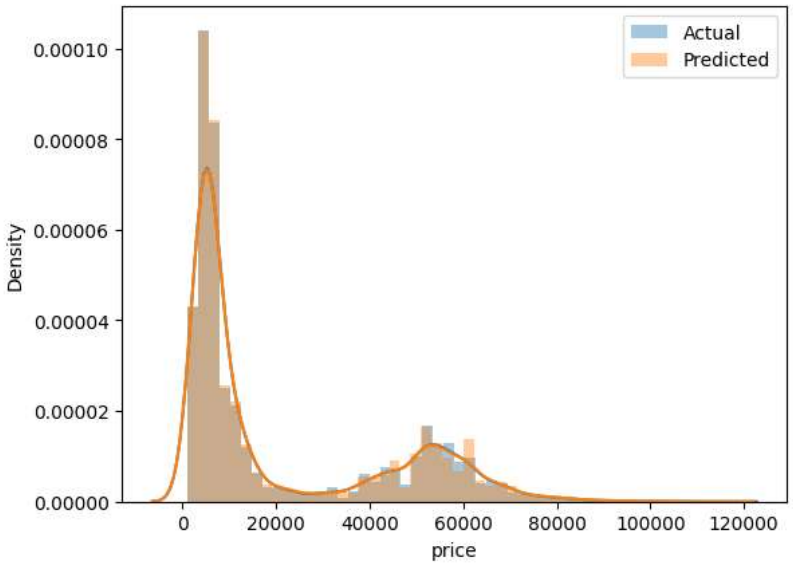
```
13061882.718802692
```

```
root_mean_sq_error = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
root_mean_sq_error
```

```
3614.122676224853
```

```
sns.distplot(y_test,label="Actual")
sns.distplot(y_pred,label="Predicted")
plt.legend()
```

```
<ipython-input-51-c0da3bcb0fc3>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test,label="Actual")
<ipython-input-51-c0da3bcb0fc3>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_pred,label="Predicted")
<matplotlib.legend.Legend at 0x7f3c18efe3a0>
```



RANDOM FOREST REGRESSION

```
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor()
rfr.fit(x_train,y_train)
y_pred=rfr.predict(x_test)
r2_score(y_test,y_pred)
```

```
0.9845269382287261
```

```
from sklearn import metrics
mean_abs_error= metrics.mean_absolute_error(y_test,y_pred)
mean_abs_error
```

```
1123.8247602112526
```

```
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, y_pred)
```

```
0.07355428620976769
```

```
mean_sq_error=metrics.mean_squared_error(y_test,y_pred)
mean_sq_error
```

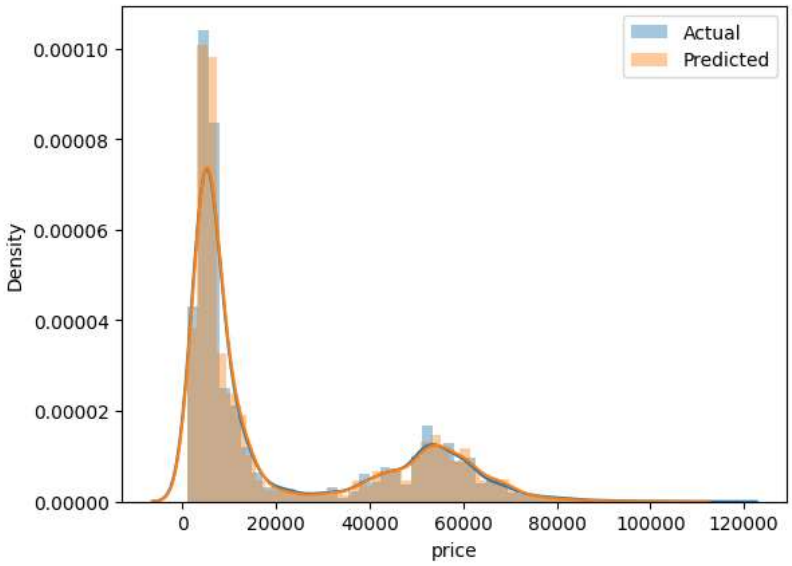
```
7976089.060927906
```

```
import numpy as np
root_mean_sq_error = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
root_mean_sq_error
```

```
2824.19706481823
```

```
sns.distplot(y_test,label="Actual")
sns.distplot(y_pred,label="Predicted")
plt.legend()
```

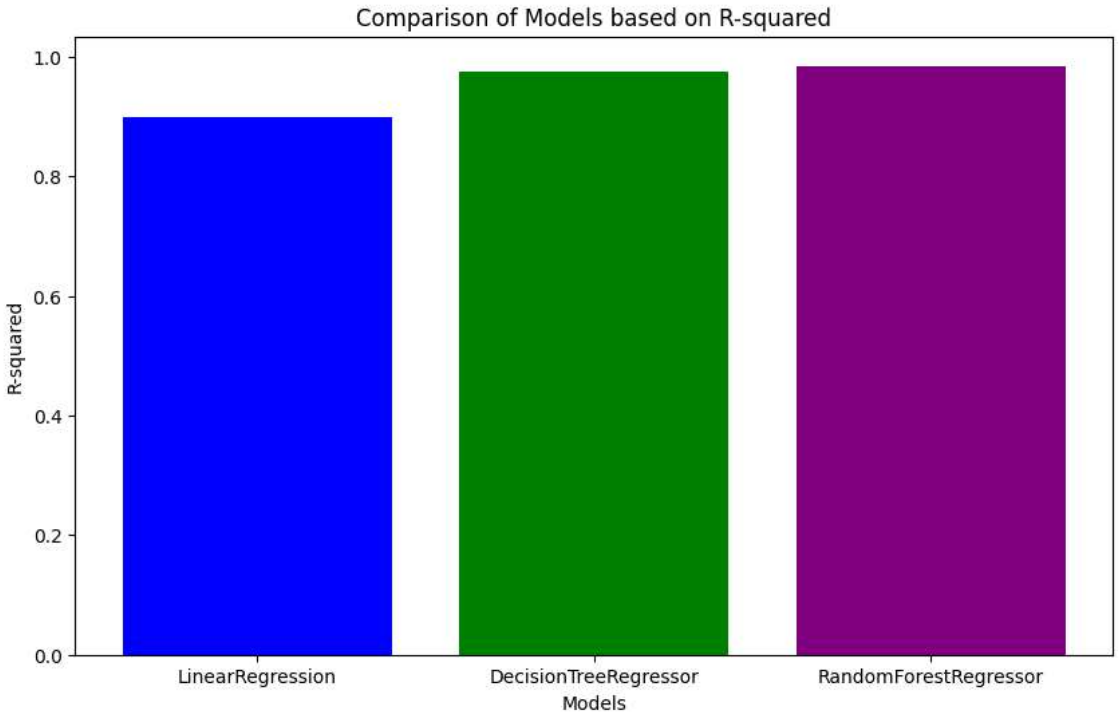
```
<ipython-input-57-c0da3bcb0fc3>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test,label="Actual")
<ipython-input-57-c0da3bcb0fc3>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_pred,label="Predicted")
<matplotlib.legend.Legend at 0x7f3c55ce3eb0>
```



COMPARISION OF DIFFERENT MODEL AS PER R(SQUARE)

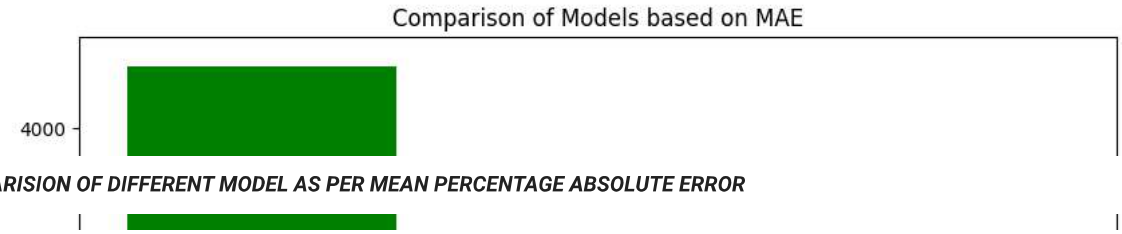
```
# R-squared values for different models
models = ['LinearRegression', 'DecisionTreeRegressor', 'RandomForestRegressor' ]
r_squared = [0.897752737512321 , 0.9746798050021364,0.9845174836070605 ]

# Set the figure size
plt.figure(figsize=(10, 6))
colors = ['blue', 'green', 'purple']
# Create a bar plot
plt.bar(models, r_squared, color=colors)

# Add labels and title
```

```
plt.xlabel('Models')
plt.ylabel('R-squared')
plt.title('Comparison of Models based on R-squared')

# Show the plot
plt.show()
```



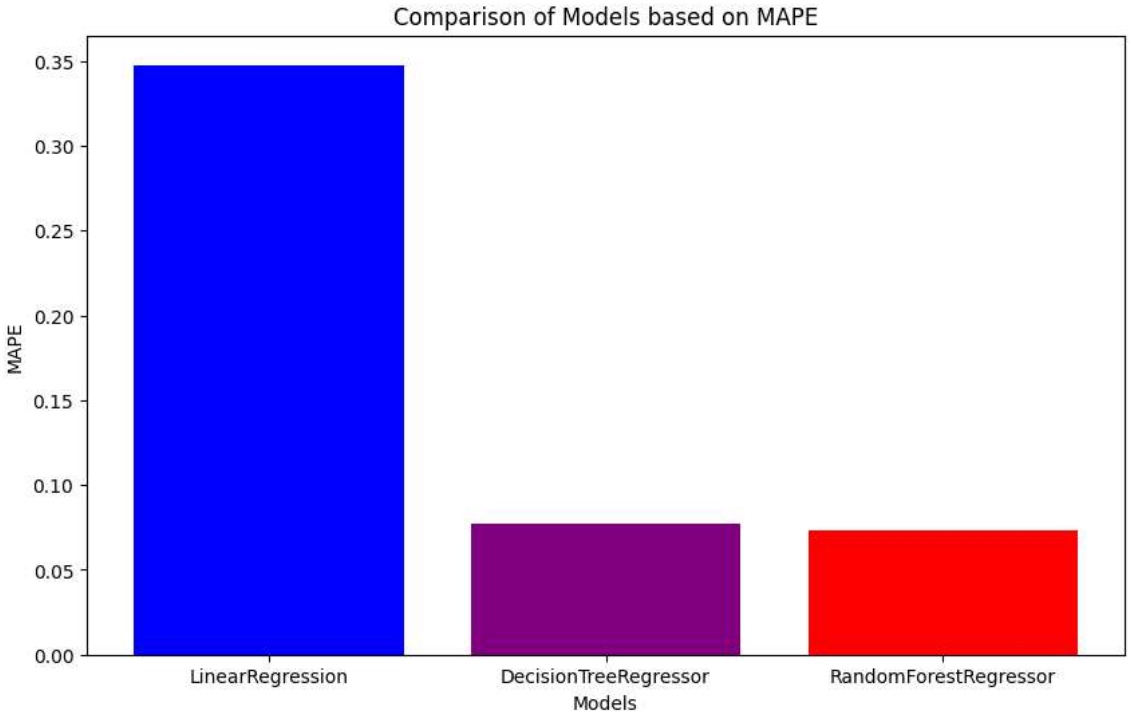
COMPARISION OF DIFFERENT MODEL AS PER MEAN ABSOLUTE ERROR

```
# MAE values for different models
models = ['LinearRegression', 'DecisionTreeRegressor', 'RandomForestRegressor' ]
MAE = [4468.426673542113, 1217.1550254590684,1123.8247602112526]
# Set the figure size
plt.figure(figsize=(10, 6))
colors = ['green', 'pink', 'orange']
# Create a bar plot
plt.bar(models, MAE, color=colors)

# Add labels and title
plt.xlabel('Models')
plt.ylabel('MAE')
plt.title('Comparison of Models based on MAE')

# Show the plot
plt.show()
```



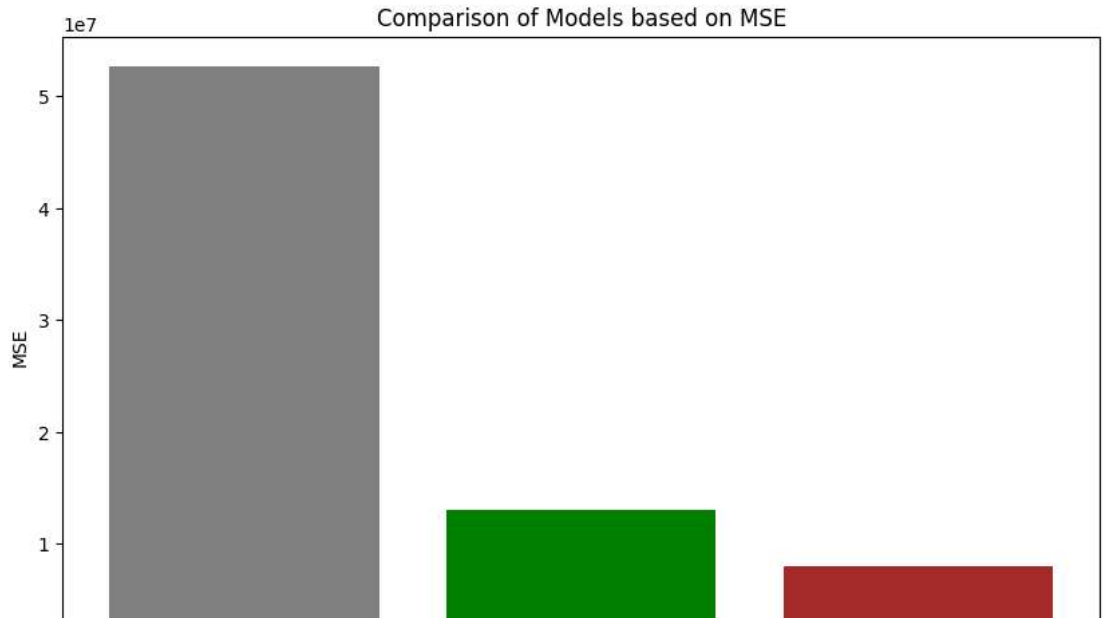
COMPARISION OF DIFFERENT MODEL AS PER MEAN PERCENTAGE ABSOLUTE ERROR

```
# MAPE values for different models
models = ['LinearRegression', 'DecisionTreeRegressor', 'RandomForestRegressor' ]
MAPE = [0.34765804610681816, 0.07726568550667867,0.07355428620976769]
# Set the figure size
plt.figure(figsize=(10, 6))
colors = ['blue', 'purple', 'red']
# Create a bar plot
plt.bar(models, MAPE, color=colors)

# Add labels and title
plt.xlabel('Models')
plt.ylabel('MAPE')
plt.title('Comparison of Models based on MAPE')

# Show the plot
plt.show()
```



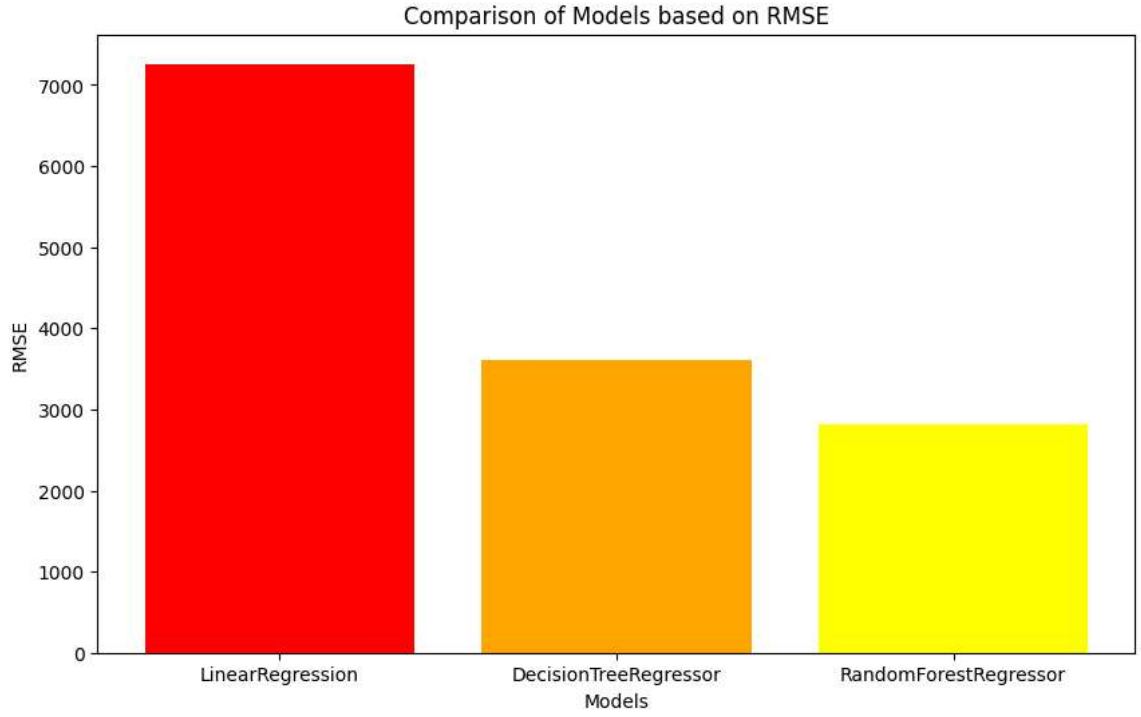
COMPARISION OF DIFFERENT MODEL AS PER MEAN SQUARE ERROR

```
# MSE values for different models
models = ['LinearRegression', 'DecisionTreeRegressor', 'RandomForestRegressor' ]
MSE = [52706651.33334208,13061882.718802692,7976089.060927906]
# Set the figure size
plt.figure(figsize=(10, 6))
colors = ['grey', 'green', 'brown']
# Create a bar plot
plt.bar(models, MSE, color=colors)

# Add labels and title
plt.xlabel('Models')
plt.ylabel('MSE')
plt.title('Comparison of Models based on MSE')

# Show the plot
plt.show()
```



COMPARISION OF DIFFERENT MODEL AS PER ROOT MEAN SQUARE ERROR

```
# RMSE values for different models
models = ['LinearRegression', 'DecisionTreeRegressor', 'RandomForestRegressor' ]
RMSE = [7259.934664536733,3614.122676224853,2824.19706481823]
# Set the figure size
plt.figure(figsize=(10, 6))
colors = ['red', 'orange', 'yellow']
# Create a bar plot
plt.bar(models, RMSE, color=colors)

# Add labels and title
plt.xlabel('Models')
plt.ylabel('RMSE')
plt.title('Comparison of Models based on RMSE')

# Show the plot
plt.show()
```



✓ 0s completed at 09:49