In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
#Load the dataset
df = pd.read_excel('Data_Train.xlsx')
```

In [3]:

```
df.head()
```

Out[3]:

	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

In [4]:

```
df.tail()
```

Out[4]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753

In [5]:

```
df.shape
```

Out[5]:

```
(10683, 11)
```

In [6]:

```
df.dtypes
```

Out[6]:

```
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops        object
Additional_Info    object
Price               int64
dtype: object
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

In [8]:
```
df.dropna(inplace=True)
```

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
220
```

In [10]:
```
df.drop_duplicates(inplace=True)
```

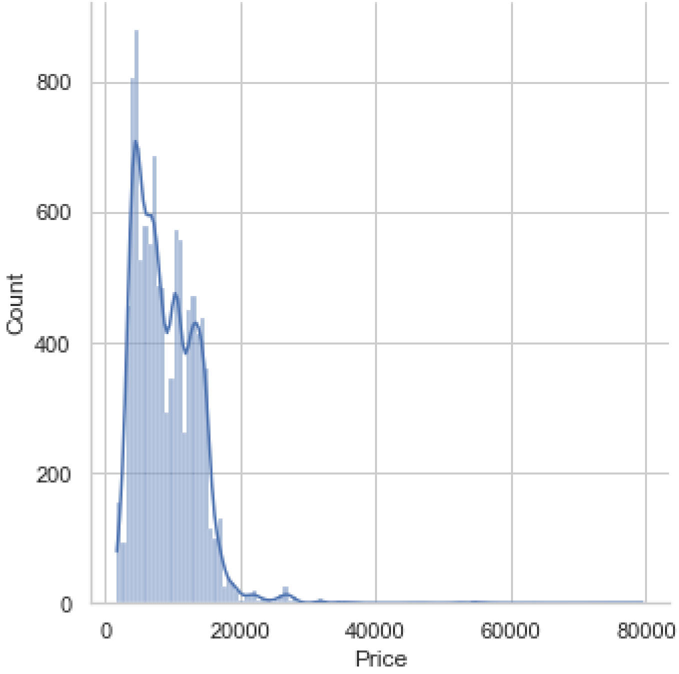
In [11]:
```
df.shape
```

Out[11]:
```
(10462, 11)
```

```
In [12]: df.nunique()
```

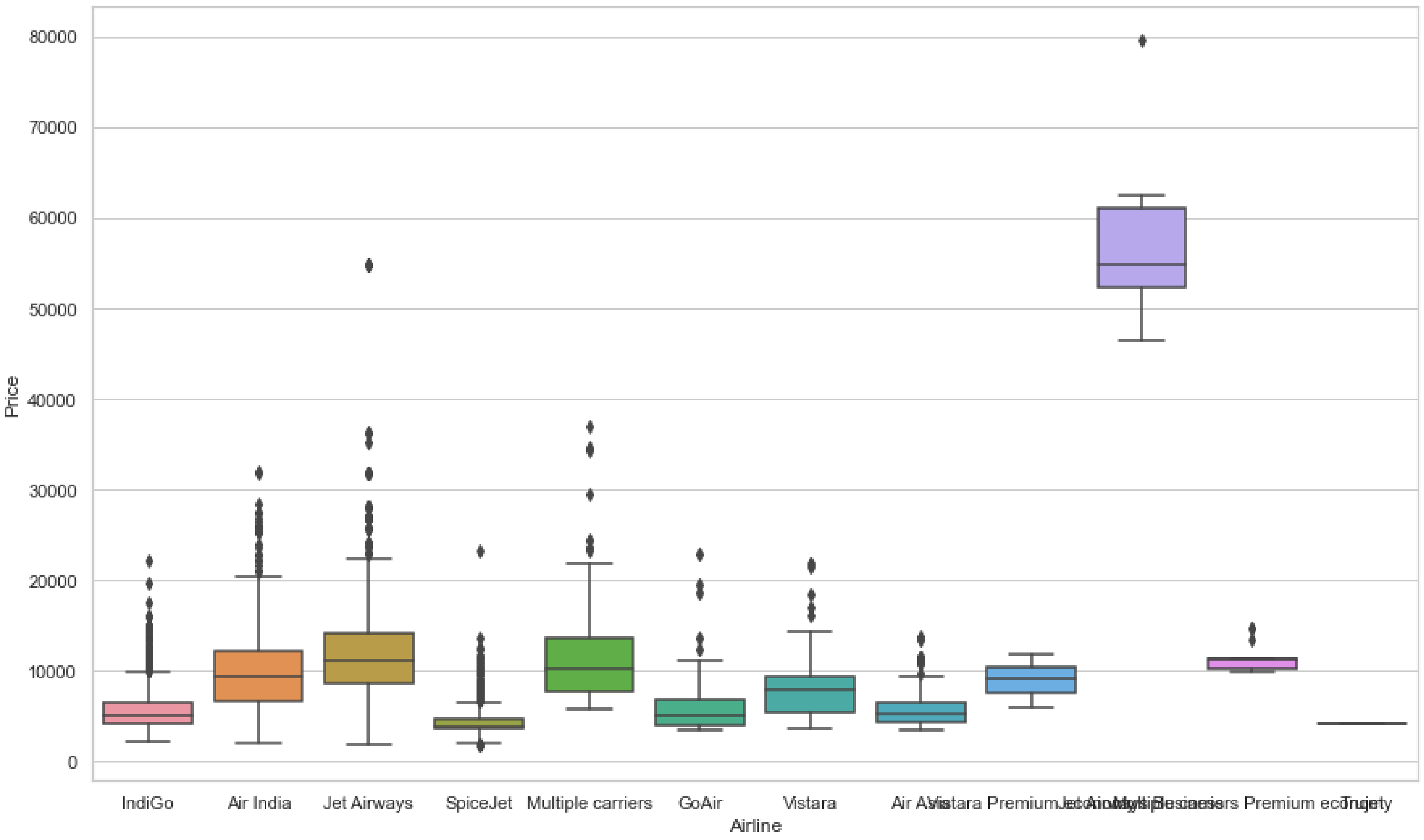
```
Out[12]: Airline              12
         Date_of_Journey      44
         Source                5
         Destination           6
         Route               128
         Dep_Time            222
         Arrival_Time       1343
         Duration            368
         Total_Stops           5
         Additional_Info      10
         Price              1870
         dtype: int64
```

```
In [13]: sns.set_theme(style='whitegrid')
         sns.displot(df['Price'],kde = True)
         plt.show()
```



```
In [14]: plt.figure(figsize=(50,30))
         plt.subplot(3,3,1)
         sns.boxplot(x='Airline',y='Price',data=df)
```

```
Out[14]: <AxesSubplot:xlabel='Airline', ylabel='Price'>
```



In [15]:

```
#Convert the date of journey to date-time and extract date, month, year
df['Journey_date'] = pd.to_datetime(df['Date_of_Journey'],format="%d/%m/%Y").dt.day
df['Journey_month'] = pd.to_datetime(df['Date_of_Journey'],format="%d/%m/%Y").dt.month
df['Journey_year'] = pd.to_datetime(df['Date_of_Journey'],format="%d/%m/%Y").dt.year
```

In [16]:

```
df['Journey_year']
```

Out[16]:

```
0        2019
1        2019
2        2019
3        2019
4        2019
         ... 
10678    2019
10679    2019
10680    2019
10681    2019
10682    2019
Name: Journey_year, Length: 10462, dtype: int64
```

In [17]:

```
#drop the date of journey column
df = df.drop("Date_of_Journey",axis=1)
```

In [18]:

```
df["Arrival_hour"] = pd.to_datetime(df["Arrival_Time"]).dt.hour
df["Arrival_min"] = pd.to_datetime(df["Arrival_Time"]).dt.minute
```

In [19]:

```
#drop Arrival time
df = df.drop("Arrival_Time",axis=1)
```

In [20]:

```
#Get dep hour and minute
df['Dep_Hour'] = pd.to_datetime(df['Dep_Time']).dt.hour
df['Dep_min'] = pd.to_datetime(df['Dep_Time']).dt.minute
```

In [21]:

```
#drop dep_time
df = df.drop("Dep_Time",axis=1)
```

In [22]:

```
#Convert duration column to minutes

x = list(df['Duration'])
a = []
v = []
for i in x:
    if "h" in i and "m" in i:
        c = i.replace("h","")
        b = c.replace("m","")
        a.append(b)
    elif "h" in i and "m" not in i:
        c = i.replace("h","")
        a.append(c)
    elif "h" not in i and "m" in i:
        g = i.replace("m","")
        a.append(g)
for x in a:
    n = x.split(sep = " ")
    v.append(n)
v[2]
dur_hr = []
dur_min = []
for i in v:
    if len(i) == 2:
        dur_hr.append(int(i[0]))
        dur_min.append(int(i[1]))
    else:
        dur_hr.append(int(i[0]))
        dur_min.append(int(0))

Dur_h = pd.Series(dur_hr)
df['Duration_hour'] = Dur_h.values
Dur_m = pd.Series(dur_min)
df['Duration_min'] = Dur_m.values
```

In [23]:

```
#drop duration column from dataframe
df = df.drop("Duration",axis=1)
```

In [24]:
```
df.head()
```

Out[24]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_date	Journey_month	Journey_year	Arrival_hour
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	3	2019	1
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	2019	13
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	2019	4
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	12	5	2019	23
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	1	3	2019	21

In [25]:
```
# One hot encode the categorical variables
airline = pd.get_dummies(df['Airline'],drop_first=True)
Source = pd.get_dummies(df['Source'],drop_first=True)
Destination = pd.get_dummies(df['Destination'],drop_first=True)
```

In [26]:
```
df.drop(["Airline","Source","Destination"],axis=1,inplace=True)
df.head()
```

Out[26]:

	Route	Total_Stops	Additional_Info	Price	Journey_date	Journey_month	Journey_year	Arrival_hour	Arrival_min	Dep_Hour	Dep_m
0	BLR → DEL	non-stop	No info	3897	24	3	2019	1	10	22	
1	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	2019	13	15	5	
2	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	2019	4	25	9	
3	CCU → NAG → BLR	1 stop	No info	6218	12	5	2019	23	30	18	
4	BLR → NAG → DEL	1 stop	No info	13302	1	3	2019	21	35	16	

In [27]:
```
df = pd.concat([df,airline,Source,Destination], axis=1)
```

In [28]:
```
df["Additional_Info"].value_counts()
df["Additional_Info"] = df["Additional_Info"].replace("No info", "No Info")
```

In [29]:
```
#Get stops from total stops
df["Stops"] = df["Total_Stops"].str.split().str[0]
df["Stops"] = df["Stops"].replace("non-stop", "0")
df["Stops"] = df["Stops"].astype(int)
df = df.drop("Total_Stops",axis=1)
```

In [30]:
```
df.head()
```

Out[30]:

	Route	Additional_Info	Price	Journey_date	Journey_month	Journey_year	Arrival_hour	Arrival_min	Dep_Hour	Dep_min	...	Chen
0	BLR → DEL	No Info	3897	24	3	2019	1	10	22	20	...	
1	CCU → IXR → BBI → BLR	No Info	7662	1	5	2019	13	15	5	50	...	
2	DEL → LKO → BOM → COK	No Info	13882	9	6	2019	4	25	9	25	...	
3	CCU → NAG → BLR	No Info	6218	12	5	2019	23	30	18	5	...	
4	BLR → NAG → DEL	No Info	13302	1	3	2019	21	35	16	50	...	

5 rows × 33 columns

In [31]:
```
df["Additional_Info"].value_counts()
```

Out[31]:
```
No Info                         8185
In-flight meal not included     1926
No check-in baggage included     318
1 Long layover                    19
Change airports                    7
Business class                     4
1 Short layover                    1
Red-eye flight                     1
2 Long layover                     1
Name: Additional_Info, dtype: int64
```

In [32]:
```
df = df.drop(["Additional_Info","Route"],axis=1)
```

In [33]:
```
df.columns
```

Out[33]:
```
Index(['Price', 'Journey_date', 'Journey_month', 'Journey_year',
       'Arrival_hour', 'Arrival_min', 'Dep_Hour', 'Dep_min', 'Duration_hour',
       'Duration_min', 'Air India', 'GoAir', 'IndiGo', 'Jet Airways',
       'Jet Airways Business', 'Multiple carriers',
       'Multiple carriers Premium economy', 'SpiceJet', 'Trujet', 'Vistara',
       'Vistara Premium economy', 'Chennai', 'Delhi', 'Kolkata', 'Mumbai',
       'Cochin', 'Delhi', 'Hyderabad', 'Kolkata', 'New Delhi', 'Stops'],
      dtype='object')
```

In [34]:
```
from sklearn.model_selection import train_test_split
X = df.drop("Price",axis=1)
Y = df["Price"]
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,random_state=(32))
```

In [35]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
```

In [36]:

```
#train and evaluate linear regressor
lr = LinearRegression(fit_intercept=False)
lr.fit(X_train,Y_train)
y_pred_lr = lr.predict(X_test)
print("Linear Regression r2_score:{:.2f}".format(r2_score(Y_test,y_pred_lr)))
print("Linear Regression mean_abs_err: {:.2f}".format(mean_absolute_error(Y_test,y_pred_lr)))
print("Linear Regression mean_square_error: {:.2f}".format(mean_squared_error(Y_test,y_pred_lr)))
print("Linear Regression root_mean-square_error: {:.2f}".format(mean_squared_error(Y_test,y_pred_lr,squared=Fa
```

```
Linear Regression r2_score:0.59
Linear Regression mean_abs_err: 1952.80
Linear Regression mean_square_error: 8431614.75
Linear Regression root_mean-square_error: 2903.72
```

In [37]:

```
#train and evaluate Decision tree regressor model
dt = DecisionTreeRegressor(random_state=(23))
dt.fit(X_train,Y_train)
y_pred_dt = dt.predict(X_test)
print("Dt_regressor-R2_score: {:.2f}%".format((r2_score(Y_test, y_pred_dt)*100)))
print("Dt_regressor-Mean_absolute_error: {:.2f}".format((mean_absolute_error(Y_test, y_pred_dt))))
print("Dt_regressor-Mean_squared_error: {:.2f}".format((mean_squared_error(Y_test, y_pred_dt))))
print("Dt_regressor-root_mean_square_error: {:.2f}".format((mean_squared_error(Y_test, y_pred_dt,squared=False
```

```
Dt_regressor-R2_score: 68.52%
Dt_regressor-Mean_absolute_error: 1412.20
Dt_regressor-Mean_squared_error: 6531217.95
Dt_regressor-root_mean_square_error: 2555.62
```

In [38]:

```
#train and evaluate Random forest regressor model
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train,Y_train)
y_pred_rf = rf.predict(X_test)
print("Random forest regressor-r2_score : {:.2f}".format(r2_score(Y_test,y_pred_rf)))
print("Random forest regressor-mean_abs_err: {:.2f}".format(mean_absolute_error(Y_test,y_pred_rf)))
print("Random forest regressor-mean_square_error: {:.2f}".format(mean_squared_error(Y_test,y_pred_rf)))
print("Random forest regressor-root_mean-square_error: {:.2f}".format(mean_squared_error(Y_test,y_pred_rf,squa
```

```
Random forest regressor-r2_score : 0.79
Random forest regressor-mean_abs_err: 1223.11
Random forest regressor-mean_square_error: 4401984.66
Random forest regressor-root_mean-square_error: 2098.09
```

In [39]:

```
#Import necessary libraries for model selection and hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV

#Define hyperparameters to be tuned for each model
params_lr = {"fit_intercept":[True,False],"normalize":[True,False]}
params_dt = {"max_depth":[3,5,7,9,11,None],"min_samples_split":[2,5,10,15,20,25]}
params_rf = {"n_estimators":[100,200,300,400,500],"max_depth":[3,5,7,9,11,None],"min_samples_split":[2,5,10,15
```

In [40]:

```
#Define the models to be tuned
models = {'Linear Regression': LinearRegression(),"Decision Tree Regressor": DecisionTreeRegressor(),"Random_f
          RandomForestRegressor()}
```

In [41]:

```
#Define the number of iterations and cv for RandomisedSearchCV
n_iter = 50
cv = 5
```

In [42]:

```
#perform hyperparameter tuning for each model
for name, model in models.items():
    if name == 'Linear Regression':
        rand_search = RandomizedSearchCV(model, param_distributions=params_lr, n_iter=n_iter, cv=cv, random_st
    elif name == 'Decision Tree Regressor':
        rand_search = RandomizedSearchCV(model, param_distributions=params_dt, n_iter=n_iter, cv=cv, random_st
    elif name == "Random_forest_regressor":
        rand_search = RandomizedSearchCV(model, param_distributions=params_rf, n_iter=n_iter, cv=cv, random_st

    #fit the parameters in models
    rand_search.fit(X_train,Y_train)

    #print the best hyperparameters and the best score for the model
    print(name)
    print("Best Hyperparameters: ",rand_search.best_params_)
    print("Best Score {:.2f}: ".format(rand_search.best_score_))
    print("")
```

```
e to silence this warning. The default behavior of this estimator is to not do any normalization. If normali
zation is needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:148: FutureWarning: 'normalize' was de
precated in version 1.0 and will be removed in 1.2. Please leave the normalize parameter to its default valu
e to silence this warning. The default behavior of this estimator is to not do any normalization. If normali
zation is needed please use sklearn.preprocessing.StandardScaler instead.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarning: 'normalize' was de
precated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the
previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pi
peline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(

Linear Regression
Best Hyperparameters:  {'normalize': True, 'fit_intercept': False}
Best Score 0.63:


C:\Users\HP\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:292: UserWarning: The total space
of parameters 36 is smaller than n_iter=50. Running 36 iterations. For exhaustive searches, use GridSearchC
V.
  warnings.warn(
Decision Tree Regressor
Best Hyperparameters:  {'min_samples_split': 25, 'max_depth': None}
Best Score 0.79:

Random_forest_regressor
Best Hyperparameters:  {'n_estimators': 200, 'min_samples_split': 10, 'max_depth': None}
Best Score 0.82:
```

In [43]:

```
# Define the models with the best hyperparameters obtained from hyperparameter tuning
lr = LinearRegression(fit_intercept=False,normalize=True)
dt = DecisionTreeRegressor(max_depth=None,min_samples_split=20,random_state=42)
rf = RandomForestRegressor(n_estimators=200,min_samples_split=10,max_depth=None,random_state=41)
```

In [44]:

```
#train and evaluate the models
for name,model in models.items():
    #fit the model on the training
    model.fit(X_train,Y_train)

    #predict target
    y_pred_test = model.predict(X_test)
```

In [45]:

```
#Calculate mean absolute and mean squared error and rmse
R2_Score = r2_score(Y_test,y_pred_test)
mae_test = mean_absolute_error(Y_test,y_pred_test)
mse_test = mean_squared_error(Y_test,y_pred_test)
Rmse_test = np.sqrt(mse_test)
```

In [47]:

```
#print evaluations
print("Test mae: {:.2f}".format(mae_test))
print("Test mse: {:.2f}".format(mse_test))
print("Test Rmse: {:.2f}".format(Rmse_test))
```

```
Test mae: 1225.60
Test mse: 4379798.55
Test Rmse: 2092.80
```