In [1]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [2]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from tqdm import tqdm
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from sklearn import model_selection as sk_model_selection
from xgboost.sklearn import XGBRegressor
from sklearn.metrics import mean_squared_error,roc_auc_score,precision_score
from sklearn import metrics
from sklearn.metrics import log_loss
from optuna.samplers import TPESampler
import functools
from functools import partial
import xgboost as xgb
import joblib
from matplotlib_venn import venn2, venn2_circles, venn2_unweighted
from matplotlib_venn import venn3, venn3_circles
import statsmodels.api as sm
import pylab
from xgboost import plot_tree
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import mean_squared_error,roc_auc_score,precision_score
from sklearn import metrics
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix, recall_score, precision_score, precision_recall_curve, auc, f1_score, average_precision_score, accuracy_score, r
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.layers import Concatenate, LSTM, GRU
from tensorflow.keras.layers import Bidirectional, Multiply
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

```
SEED = 42
```

In [3]:
```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

In [5]:
```
train_data = pd.read_excel('C:/Users/HP/Desktop/Predict_Book_Price/Participants_Data/Data_Train.xlsx')
test_data = pd.read_excel('C:/Users/HP/Desktop/Predict_Book_Price/Participants_Data/Data_Test.xlsx')
# submission_data = test_data[['ID']].copy()
```

In [6]:
```
train_data['ID'] = train_data.index
```

In [7]:
```
test_data['ID'] = test_data.index
```

In [8]:
```
train_data.dtypes
```

Out[8]:
```
Title            object
Author           object
Edition          object
Reviews          object
Ratings          object
Synopsis         object
Genre            object
BookCategory     object
Price           float64
ID                int64
dtype: object
```

In [9]:
```
test_data.dtypes
```

Out[9]:
```
Title           object
Author          object
Edition         object
Reviews         object
Ratings         object
```

```
Synopsis        object
Genre           object
BookCategory    object
ID               int64
```

Looking at the data

In [10]:

```
pd.set_option('display.max_columns',None)
print(train_data.shape, train_data['ID'].nunique())
print(train_data.columns)
train_data.head()
```

```
(6237, 10) 6237
Index(['Title', 'Author', 'Edition', 'Reviews', 'Ratings', 'Synopsis', 'Genre',
       'BookCategory', 'Price', 'ID'],
      dtype='object')
```

Out[10]:

	Title	Author	Edition	Reviews	Ratings	Synopsis	Genre	BookCategory	Price	ID
0	The Prisoner's Gold (The Hunters 3)	Chris Kuzneski	Paperback,– 10 Mar 2016	4.0 out of 5 stars	8 customer reviews	THE HUNTERS return in their third brilliant no...	Action & Adventure (Books)	Action & Adventure	220.00	0
1	Guru Dutt: A Tragedy in Three Acts	Arun Khopkar	Paperback,– 7 Nov 2012	3.9 out of 5 stars	14 customer reviews	A layered portrait of a troubled genius for wh...	Cinema & Broadcast (Books)	Biographies, Diaries & True Accounts	202.93	1
2	Leviathan (Penguin Classics)	Thomas Hobbes	Paperback,– 25 Feb 1982	4.8 out of 5 stars	6 customer reviews	"During the time men live without a common Pow...	International Relations	Humour	299.00	2
3	A Pocket Full of Rye (Miss Marple)	Agatha Christie	Paperback,– 5 Oct 2017	4.1 out of 5 stars	13 customer reviews	A handful of grain is found in the pocket of a...	Contemporary Fiction (Books)	Crime, Thriller & Mystery	180.00	3
4	LIFE 70 Years of Extraordinary Photography	Editors of Life	Hardcover,– 10 Oct 2006	5.0 out of 5 stars	1 customer review	For seven decades, "Life" has been thrilling t...	Photography Textbooks	Arts, Film & Photography	965.62	4

In [11]:

```
train_data['Price'].describe()
```

Out[11]:

```
count     6237.000000
mean       560.707516
std        690.110657
min         25.000000
25%        249.180000
50%        373.000000
75%        599.000000
max      14100.000000
Name: Price, dtype: float64
```

In [12]:
```
print(test_data.shape)
print(test_data.columns)
test_data.head()
```

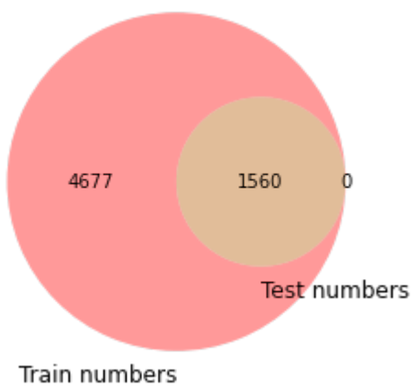
```
(1560, 9)
Index(['Title', 'Author', 'Edition', 'Reviews', 'Ratings', 'Synopsis', 'Genre',
       'BookCategory', 'ID'],
      dtype='object')
```

Out[12]:

	Title	Author	Edition	Reviews	Ratings	Synopsis	Genre	BookCategory	ID
0	The Complete Sherlock Holmes: 2 Boxes sets	Sir Arthur Conan Doyle	Mass Market Paperback,– 1 Oct 1986	4.4 out of 5 stars	960 customer reviews	A collection of entire body of work of the She...	Short Stories (Books)	Crime, Thriller & Mystery	0
1	Learn Docker - Fundamentals of Docker 18.x: Ev...	Gabriel N. Schenker	Paperback,– Import, 26 Apr 2018	5.0 out of 5 stars	1 customer review	Enhance your software deployment workflow usin...	Operating Systems Textbooks	Computing, Internet & Digital Media	1
2	Big Girl	Danielle Steel	Paperback,– 17 Mar 2011	5.0 out of 5 stars	4 customer reviews	'Watch out, world. Here I come!'\nFor Victoria...	Romance (Books)	Romance	2
3	Think Python: How to Think Like a Computer Sci...	Allen B. Downey	Paperback,– 2016	4.1 out of 5 stars	11 customer reviews	If you want to learn how to program, working w...	Programming & Software Development (Books)	Computing, Internet & Digital Media	3
4	Oxford Word Skills: Advanced - Idioms & Phrasa...	Redman Gairns	Paperback,– 26 Dec 2011	4.4 out of 5 stars	9 customer reviews	Learn and practise the verbs, prepositions and...	Linguistics (Books)	Language, Linguistics & Writing	4

In [13]:
```
set_numbers_train = set(train_data[['ID']].drop_duplicates().sort_values(by = 'ID')['ID'].tolist())
set_numbers_test = set(test_data[['ID']].drop_duplicates().sort_values(by = 'ID')['ID'].tolist())
venn2((set_numbers_train, set_numbers_test), set_labels = ('Train numbers', 'Test numbers'))
```

Out[13]:
```
<matplotlib_venn._common.VennDiagram at 0x1967b7faa60>
```



Fill rate across columns

In [14]:
```
print('Number of rows in train:',train_data.shape[0])
(train_data.isnull().sum().sort_values(ascending = False)/train_data.shape[0] * 100)
```

```
Number of rows in train: 6237
```

Out[14]:
```
Title           0.0
Author          0.0
Edition         0.0
Reviews         0.0
Ratings         0.0
Synopsis        0.0
Genre           0.0
BookCategory    0.0
Price           0.0
ID              0.0
dtype: float64
```

In [15]:
```
print('Number of rows:',test_data.shape[0])
(test_data.isnull().sum().sort_values(ascending = False)/test_data.shape[0] * 100)
```

```
Number of rows: 1560
```

Out[15]:
```
Title           0.0
Author          0.0
Edition         0.0
Reviews         0.0
Ratings         0.0
Synopsis        0.0
Genre           0.0
BookCategory    0.0
ID              0.0
dtype: float64
```

In [16]:
```
print('# Records in train data:',train_data.shape[0])
train_data.nunique().sort_values().head(20)
```

```
# Records in train data: 6237
```

Out[16]:
```
BookCategory      11
Reviews           36
Ratings          342
Genre            345
Price           1614
Edition         3370
Author          3679
Synopsis        5549
Title           5568
ID              6237
dtype: int64
```

In [17]:
```
test_data.nunique().sort_values()
```

Out[17]:
```
BookCategory      11
Reviews           30
Ratings          163
Genre            225
Author          1224
Edition         1259
Synopsis        1519
Title           1521
ID              1560
dtype: int64
```

In [18]:
```
for col in train_data.nunique().sort_values().head(13).reset_index()['index'].tolist():
    print(col,'\n')
    display(train_data.groupby(col).size().reset_index())
    print('--'*50,'\n')
```

```
BookCategory
```

	BookCategory	0
0	Action & Adventure	818
1	Arts, Film & Photography	517
2	Biographies, Diaries & True Accounts	596
3	Comics & Mangas	583
4	Computing, Internet & Digital Media	510
5	Crime, Thriller & Mystery	723
6	Humour	540
7	Language, Linguistics & Writing	594
8	Politics	325
9	Romance	560
10	Sports	471

```
----------------------------------------------------------------------------------------------------
```

```
Reviews
```

	Reviews	0
0	1.0 out of 5 stars	49
1	1.4 out of 5 stars	2
2	1.5 out of 5 stars	5

	Reviews	0
3	1.6 out of 5 stars	1
4	1.7 out of 5 stars	1
5	2.0 out of 5 stars	39
6	2.1 out of 5 stars	1
7	2.2 out of 5 stars	3
8	2.3 out of 5 stars	7
9	2.4 out of 5 stars	4
10	2.5 out of 5 stars	18
11	2.6 out of 5 stars	4
12	2.7 out of 5 stars	16
13	2.8 out of 5 stars	10
14	2.9 out of 5 stars	26
15	3.0 out of 5 stars	138
16	3.1 out of 5 stars	49
17	3.2 out of 5 stars	41
18	3.3 out of 5 stars	57
19	3.4 out of 5 stars	75
20	3.5 out of 5 stars	115
21	3.6 out of 5 stars	110
22	3.7 out of 5 stars	167
23	3.8 out of 5 stars	190
24	3.9 out of 5 stars	241
25	4.0 out of 5 stars	570
26	4.1 out of 5 stars	310
27	4.2 out of 5 stars	324
28	4.3 out of 5 stars	359
29	4.4 out of 5 stars	389
30	4.5 out of 5 stars	507

	Reviews	0
31	4.6 out of 5 stars	394
32	4.7 out of 5 stars	343
33	4.8 out of 5 stars	222

Ratings

	Ratings	0
0	1 customer review	1040
1	1,097 customer reviews	1
2	1,142 customer reviews	1
3	1,227 customer reviews	1
4	1,248 customer reviews	1
...	...	...
337	97 customer reviews	7
338	970 customer reviews	1
339	973 customer reviews	1
340	98 customer reviews	8
341	99 customer reviews	4

342 rows × 2 columns

Genre

	Genre	0
0	API & Operating Environments	2
1	Action & Adventure (Books)	947
2	Active Outdoor Pursuits (Books)	1
3	Aeronautical Engineering	1
4	Aesthetics	1
...	...	...

	Genre	0
340	World African & Middle Eastern Literature	2
341	Writing Guides (Books)	71
342	XHTML Software Programming	2
343	Young Adults' Money & Jobs (Books)	1
344	Zoology	1

```
Price
```

	Price	0
0	25.00	2
1	28.00	1
2	30.00	2
3	31.00	2
4	36.00	2
...	...	...
1609	9096.00	1
1610	9984.00	1
1611	11715.12	1
1612	13244.67	1
1613	14100.00	1

1614 rows × 2 columns

```
Edition
```

	Edition	0
0	(French),Paperback,– 31 Oct 2010	1
1	(German),Paperback,– 17 Nov 2014	1
2	(Kannada),Paperback,– 2014	1

	Edition	0
3	(Spanish),Paperback,– Import, 7 Jun 2012	1
4	Board book,– 9 Jul 2013	1
...	...	...
3365	Spiral-bound,– 1 Mar 2007	1
3366	Spiral-bound,– 21 Jun 2016	1
3367	Tankobon Softcover,– 2016	1
3368	Tankobon Softcover,– Apr 2016	1
3369	Tankobon Softcover,– Import, 22 May 2017	1

Author

	Author	0
0	0, Butterfield, Ngondi, Kerr	1
1	0, Jonathan Law, Richard Rennie	1
2	0, Kerr, Wright	1
3	0, Rennie, Law	1
4	0, Speake	1
...	...	...
3674	Zygmunt Miloszewski	1
3675	dodie	1
3676	r.h. Sin	1
3677	renu and neena kaul	1
3678	sister Jesme	1

3679 rows × 2 columns

Synopsis

	Synopsis	0

	Synopsis	0
0	The Scarecrows' Wedding is a wonderfully h...	1
1	A heart-warming picture book story of friend...	2
2	An eye-catching new edition of this Horrible...	1
3	From the bestselling author and illustrator ...	2
4	Howl with laughter with the FIFTH book in th...	2
...	...	...
5544	• One of the comedy world's fastest-rising sta...	1
5545	• This VIZBIG edition of Dragon Ball Z contain...	1
5546	• This VIZBIG edition of Vagabond contains Vol...	1
5547	• This is the official tie-in volume for Girat...	1
5548	• Thousands of Africans head to China each yea...	1

```
Title
```

	Title	0
0	#GIRLBOSS	2
1	'One Who Will':the Search for Steve Waugh	1
2	(ISC)2 CISSP Certified Information Systems Sec...	1
3	1 Page at a Time: A Daily Creative Companion	1
4	1,339 Qi Facts to Make Your Jaw Drop	1
...	...	...
5563	Zurich International Chess Tournament, 1953	1
5564	integration of the Indian States	1
5565	internet of Things: A Hands-On Approach	1
5566	karma and Reincarnation	1
5567	orange: The Complete Collection 1	1

5568 rows × 2 columns

ID

	ID	0
0	0	1
1	1	1
2	2	1
3	3	1
4	4	1
...	...	...
6232	6232	1
6233	6233	1
6234	6234	1
6235	6235	1
6236	6236	1

6237 rows × 2 columns

In [19]:

```
cols_with_very_low_distinct_values = ['BookCategory','Reviews']
for col in cols_with_very_low_distinct_values:
    print(col,'\n')
    display(test_data.groupby(col).size().reset_index())
    print('--'*50,'\n')
```

BookCategory

	BookCategory	0
0	Action & Adventure	218
1	Arts, Film & Photography	121
2	Biographies, Diaries & True Accounts	136
3	Comics & Mangas	161
4	Computing, Internet & Digital Media	138
5	Crime, Thriller & Mystery	155
6	Humour	130

	BookCategory	0
7	Language, Linguistics & Writing	139
8	Politics	77
9	Romance	142

Reviews

	Reviews	0
0	1.0 out of 5 stars	16
1	1.5 out of 5 stars	1
2	2.0 out of 5 stars	5
3	2.4 out of 5 stars	1
4	2.5 out of 5 stars	5
5	2.6 out of 5 stars	1
6	2.7 out of 5 stars	2
7	2.8 out of 5 stars	2
8	2.9 out of 5 stars	10
9	3.0 out of 5 stars	38
10	3.1 out of 5 stars	8
11	3.2 out of 5 stars	9
12	3.3 out of 5 stars	17
13	3.4 out of 5 stars	17
14	3.5 out of 5 stars	28
15	3.6 out of 5 stars	35
16	3.7 out of 5 stars	36
17	3.8 out of 5 stars	49
18	3.9 out of 5 stars	59
19	4.0 out of 5 stars	143
20	4.1 out of 5 stars	71

	Reviews	0
21	4.2 out of 5 stars	82
22	4.3 out of 5 stars	91
23	4.4 out of 5 stars	99
24	4.5 out of 5 stars	119
25	4.6 out of 5 stars	97
26	4.7 out of 5 stars	69
27	4.8 out of 5 stars	50
28	4.9 out of 5 stars	24

In [20]:
```
train_data.dtypes
```

Out[20]:
```
Title            object
Author           object
Edition          object
Reviews          object
Ratings          object
Synopsis         object
Genre            object
BookCategory     object
Price           float64
ID                int64
dtype: object
```

In [21]:
```
def box_plot_by_metadata(metadata, numerical_col):
    print('\n# Unique values in',numerical_col,':',train_data[numerical_col].nunique(),'\n')
    display(train_data[numerical_col].describe())
    if metadata == 'All':
        fig = px.box(train_data, y=numerical_col)
    else:
        fig = px.box(train_data, y=numerical_col, x = metadata)
    fig.show()

metadata_list = ['All'] + ['BookCategory']
numerical_col_list = ['Price', 'Ratings', 'Reviews', 'Synopsis']

w = widgets.interactive(box_plot_by_metadata, metadata = metadata_list, numerical_col = numerical_col_list)
display(w)
```

Distribution by `BookCategory` with respect to the `Ratings`

In [22]:
```
train_data.groupby(['BookCategory']).size().reset_index().rename(columns = {0:'# Ratings'}).sort_values(by = '# Ratings', ascending = False).head(10)
```

Out[22]:

	BookCategory	# Ratings
0	Action & Adventure	818
5	Crime, Thriller & Mystery	723
2	Biographies, Diaries & True Accounts	596
7	Language, Linguistics & Writing	594
3	Comics & Mangas	583
9	Romance	560
6	Humour	540
1	Arts, Film & Photography	517
4	Computing, Internet & Digital Media	510
10	Sports	471

In [23]:
```
test_data.groupby(['BookCategory']).size().reset_index().rename(columns = {0:'# Price'}).sort_values(by = '# Price', ascending = False).head(10)
```

Out[23]:

	BookCategory	# Price
0	Action & Adventure	218
3	Comics & Mangas	161
5	Crime, Thriller & Mystery	155
10	Sports	143
9	Romance	142
7	Language, Linguistics & Writing	139
4	Computing, Internet & Digital Media	138
2	Biographies, Diaries & True Accounts	136
6	Humour	130
1	Arts, Film & Photography	121

Distribution by `Price`

In [24]:
```
temp = train_data.groupby('Price').size().sort_values(ascending = False).reset_index().rename(columns = {0:'# ID'})
temp.head(20)
```

Out[24]:

	Price	# ID
0	299.0	108
1	399.0	85
2	449.0	59
3	295.0	49
4	319.0	48
5	224.0	46
6	199.0	46
7	499.0	45
8	479.0	41
9	309.0	40
10	247.0	38
11	359.0	38
12	150.0	38
13	239.0	37
14	350.0	37
15	339.0	33
16	300.0	32
17	250.0	32
18	374.0	32
19	200.0	32

In [25]:
```
dict_loan_title = dict(zip(temp['Price'].tolist(), [x for x in range(0, len(temp['Price'].tolist()))]))
```

In [26]:
```
top_loan_titles = temp.head(20)['Price'].tolist()
top_loan_titles
```

Out[26]:
```
[299.0,
 399.0,
 449.0,
 295.0,
 319.0,
 224.0,
 199.0,
 499.0,
 479.0,
 309.0,
 247.0,
 359.0,
 150.0,
 239.0,
 350.0,
 339.0,
 300.0,
 250.0,
 374.0,
 200.0]
```

Scatter plot

In [27]:
```
train_data.head(3)
```

Out[27]:

	Title	Author	Edition	Reviews	Ratings	Synopsis	Genre	BookCategory	Price	ID
0	The Prisoner's Gold (The Hunters 3)	Chris Kuzneski	Paperback,– 10 Mar 2016	4.0 out of 5 stars	8 customer reviews	THE HUNTERS return in their third brilliant no...	Action & Adventure (Books)	Action & Adventure	220.00	0
1	Guru Dutt: A Tragedy in Three Acts	Arun Khopkar	Paperback,– 7 Nov 2012	3.9 out of 5 stars	14 customer reviews	A layered portrait of a troubled genius for wh...	Cinema & Broadcast (Books)	Biographies, Diaries & True Accounts	202.93	1
2	Leviathan (Penguin Classics)	Thomas Hobbes	Paperback,– 25 Feb 1982	4.8 out of 5 stars	6 customer reviews	"During the time men live without a common Pow...	International Relations	Humour	299.00	2

In [28]:
```
fig = px.scatter(train_data, x="Price", y="Ratings",color="Price", hover_data=['BookCategory'])
fig.show()
```

Percentage of target by categorical cols

In [29]:

```
def percentage_of_target_by_col(col):
    data = train_data.copy()
    if (col in train_data.dtypes[(train_data.dtypes=='float') | (train_data.dtypes=='int')].index.tolist()) & (data[col].nunique()>10):
        data[col + '- categorical'] = pd.qcut(data[col], 10)
        col = col + '- categorical'
    temp = data.groupby(col).agg({'Price':['count','mean']}).reset_index()
    temp.columns = [col,'# Records', '% Target']
    temp = temp.sort_values(by = ['% Target'], ascending = False).reset_index(drop = True)
    display(temp.head(20))

w = widgets.interactive(percentage_of_target_by_col, col = train_data.columns.tolist())
display(w)
```

In [30]:

```
train_data.head(3)
```

Out[30]:

	Title	Author	Edition	Reviews	Ratings	Synopsis	Genre	BookCategory	Price	ID
0	The Prisoner's Gold (The Hunters 3)	Chris Kuzneski	Paperback,– 10 Mar 2016	4.0 out of 5 stars	8 customer reviews	THE HUNTERS return in their third brilliant no...	Action & Adventure (Books)	Action & Adventure	220.00	0
1	Guru Dutt: A Tragedy in Three Acts	Arun Khopkar	Paperback,– 7 Nov 2012	3.9 out of 5 stars	14 customer reviews	A layered portrait of a troubled genius for wh...	Cinema & Broadcast (Books)	Biographies, Diaries & True Accounts	202.93	1
2	Leviathan (Penguin Classics)	Thomas Hobbes	Paperback,– 25 Feb 1982	4.8 out of 5 stars	6 customer reviews	"During the time men live without a common Pow...	International Relations	Humour	299.00	2

In [31]:
```
train_data.BookCategory.value_counts()
```

Out[31]:
```
Action & Adventure                    818
Crime, Thriller & Mystery             723
Biographies, Diaries & True Accounts  596
Language, Linguistics & Writing       594
Comics & Mangas                       583
Romance                               560
Humour                                540
Arts, Film & Photography              517
Computing, Internet & Digital Media   510
Sports                                471
Politics                              325
Name: BookCategory, dtype: int64
```

In [32]:
```
train_data['ReviewsNumeric'] = train_data.Reviews.apply(lambda r: float(r.split()[0]))
```

In [33]:
```
train_data = train_data.drop(columns="Reviews")
```

In [34]:
```
train_data = train_data.rename(columns={"ReviewsNumeric": "Reviews"})
```

Similarly, for the `Ratings` columns

In [35]:
```
train_data.Ratings
```

Out[35]:
```
0         8 customer reviews
1        14 customer reviews
2         6 customer reviews
3        13 customer reviews
4          1 customer review
               ...
6232      2 customer reviews
6233      9 customer reviews
6234      3 customer reviews
```

```
6235     4 customer reviews
6236     2 customer reviews
```

In [36]:
```
train_data['Ratings'] = train_data['Ratings'].str.replace(r'\D+', '').replace('','0').astype(int)
```

In [37]:
```
train_data.head(2)
```

Out[37]:

	Title	Author	Edition	Ratings	Synopsis	Genre	BookCategory	Price	ID	Reviews
0	The Prisoner's Gold (The Hunters 3)	Chris Kuzneski	Paperback,– 10 Mar 2016	8	THE HUNTERS return in their third brilliant no...	Action & Adventure (Books)	Action & Adventure	220.00	0	4.0
1	Guru Dutt: A Tragedy in Three Acts	Arun Khopkar	Paperback,– 7 Nov 2012	14	A layered portrait of a troubled genius for wh...	Cinema & Broadcast (Books)	Biographies, Diaries & True Accounts	202.93	1	3.9

Encoding Of Rest Variables

In [38]:
```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
train_data['Title']= label_encoder.fit_transform(train_data['Title'])

train_data['Title'].unique()
```

Out[38]:
```
array([4763, 1721, 2446, ..., 4264, 1559, 4325])
```

In [39]:
```
train_data['Author']= label_encoder.fit_transform(train_data['Author'])
```

In [40]:
```
train_data['Edition']= label_encoder.fit_transform(train_data['Edition'])
```

In [41]:
```
train_data['Synopsis']= label_encoder.fit_transform(train_data['Synopsis'])
```

In [42]:
```
train_data['Genre']= label_encoder.fit_transform(train_data['Genre'])
```

In [43]:
```
train_data['BookCategory']= label_encoder.fit_transform(train_data['BookCategory'])
```

Correlation analysis

In [44]:
```
target = 'Author'
corr = train_data[[target] + numerical_col_list].corr()
corr.style.background_gradient(cmap='coolwarm', axis=None)
```

Out[44]:

	Author	Price	Ratings	Reviews	Synopsis
Author	1.000000	-0.002670	-0.002371	-0.015712	0.052001
Price	-0.002670	1.000000	-0.078063	0.108373	0.011334
Ratings	-0.002371	-0.078063	1.000000	-0.000347	-0.001242
Reviews	-0.015712	0.108373	-0.000347	1.000000	0.015895
Synopsis	0.052001	0.011334	-0.001242	0.015895	1.000000

PPS

In [45]:
```
train_data.dtypes[(train_data.dtypes!='float') & (train_data.dtypes!='int')].index.tolist()
```

Out[45]:
```
['ID']
```

In [46]:
```
train_data.head()
```

Out[46]:

	Title	Author	Edition	Ratings	Synopsis	Genre	BookCategory	Price	ID	Reviews
0	4763	615	1042	8	3760	1	0	220.00	0	4.0
1	1721	307	2762	14	586	74	2	202.93	1	3.9
2	2446	3390	1960	6	31	193	6	299.00	2	4.8
3	155	62	2615	13	560	92	5	180.00	3	4.1
4	2343	953	81	1	1822	253	1	965.62	4	5.0

Similarly for the `Test Dataset`

In [47]:
```
test_data.head(3)
```

Out[47]:

	Title	Author	Edition	Reviews	Ratings	Synopsis	Genre	BookCategory	ID
0	The Complete Sherlock Holmes: 2 Boxes sets	Sir Arthur Conan Doyle	Mass Market Paperback,– 1 Oct 1986	4.4 out of 5 stars	960 customer reviews	A collection of entire body of work of the She...	Short Stories (Books)	Crime, Thriller & Mystery	0
1	Learn Docker - Fundamentals of Docker 18.x: Ev...	Gabriel N. Schenker	Paperback,– Import, 26 Apr 2018	5.0 out of 5 stars	1 customer review	Enhance your software deployment workflow usin...	Operating Systems Textbooks	Computing, Internet & Digital Media	1
2	Big Girl	Danielle Steel	Paperback,– 17 Mar 2011	5.0 out of 5 stars	4 customer reviews	'Watch out, world. Here I come!'\nFor Victoria...	Romance (Books)	Romance	2

In [48]:
```
test_data['Reviews'] = test_data.Reviews.apply(lambda r: float(r.split()[0]))
```

In [49]:
```
test_data['Ratings'] = test_data['Ratings'].str.replace(r'\D+', '').replace('','0').astype(int)
```

In [50]:
```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
test_data['Title']= label_encoder.fit_transform(test_data['Title'])

test_data['Title'].unique()
```

Out[50]:
```
array([1142,  665,  184, ..., 1221,  873,  278])
```

In [51]:
```
test_data['Author']= label_encoder.fit_transform(test_data['Author'])
```

In [52]:
```
test_data['Edition']= label_encoder.fit_transform(test_data['Edition'])
```

In [53]:
```
test_data['Synopsis']= label_encoder.fit_transform(test_data['Synopsis'])
```

In [54]:
```
test_data['Genre']= label_encoder.fit_transform(test_data['Genre'])
```

In [55]:
```
test_data['BookCategory']= label_encoder.fit_transform(test_data['BookCategory'])
```

Feature Columns

In [56]:
```
train_data.columns
```

Out[56]:
```
Index(['Title', 'Author', 'Edition', 'Ratings', 'Synopsis', 'Genre',
       'BookCategory', 'Price', 'ID', 'Reviews'],
      dtype='object')
```

In [57]:
```
## Checking for Outliers
```

In [58]:
```
# Checking for outliers in the continuous variables
num_train_data = train_data[['Title', 'Author', 'Edition', 'Ratings', 'Synopsis', 'Genre','BookCategory', 'Price', 'ID', 'Reviews']]
```

In [59]:
```
# Checking outliers at 25%, 50%, 75%, 90%, 95% and 99%
num_train_data.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[59]:

	Title	Author	Edition	Ratings	Synopsis	Genre	BookCategory	Price	ID	Reviews
count	6237.000000	6237.000000	6237.000000	6237.000000	6237.000000	6237.000000	6237.000000	6237.000000	6237.000000	6237.000000
mean	2782.260061	1795.325317	1734.837101	35.984287	2771.244028	136.275453	4.627385	560.707516	3118.000000	4.293202
std	1614.438441	1058.152146	901.926928	149.995031	1607.072809	103.891373	3.169320	690.110657	1800.611146	0.662501
min	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	25.000000	0.000000	1.000000
25%	1374.000000	893.000000	992.000000	2.000000	1372.000000	34.000000	2.000000	249.180000	1559.000000	4.000000
50%	2797.000000	1787.000000	1749.000000	7.000000	2782.000000	103.000000	5.000000	373.000000	3118.000000	4.400000
75%	4178.000000	2699.000000	2485.000000	22.000000	4159.000000	218.000000	7.000000	599.000000	4677.000000	4.800000
90%	4999.400000	3281.400000	2964.400000	65.400000	4992.400000	282.000000	9.000000	1088.822000	5612.400000	5.000000
95%	5283.200000	3482.200000	3169.200000	133.200000	5268.200000	311.000000	10.000000	1575.000000	5924.200000	5.000000
99%	5513.640000	3628.640000	3322.640000	559.960000	5491.640000	341.000000	10.000000	3403.360000	6173.640000	5.000000
max	5567.000000	3678.000000	3369.000000	6090.000000	5548.000000	344.000000	10.000000	14100.000000	6236.000000	5.000000

In [60]:
```
Q1 = train_data.quantile(0.25)
Q3 = train_data.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Title           2804.00
Author          1806.00
Edition         1493.00
Ratings           20.00
Synopsis        2787.00
Genre            184.00
BookCategory       5.00
Price            349.82
ID              3118.00
Reviews            0.80
dtype: float64
```

In [61]:
```
print(train_data.quantile(0.05))
print(train_data.quantile(0.95))
```

```
Title           268.8
Author          134.0
Edition         210.0
Ratings           1.0
Synopsis        266.8
Genre             1.0
BookCategory      0.0
Price           123.0
ID              311.8
Reviews           3.0
Name: 0.05, dtype: float64
Title           5283.2
Author          3482.2
Edition         3169.2
Ratings          133.2
Synopsis        5268.2
Genre            311.0
BookCategory      10.0
Price           1575.0
ID              5924.2
Reviews            5.0
Name: 0.95, dtype: float64
```

In [62]:
```
train_data.head(4)
```

Out[62]:

	Title	Author	Edition	Ratings	Synopsis	Genre	BookCategory	Price	ID	Reviews
0	4763	615	1042	8	3760	1	0	220.00	0	4.0
1	1721	307	2762	14	586	74	2	202.93	1	3.9

	Title	Author	Edition	Ratings	Synopsis	Genre	BookCategory	Price	ID	Reviews
2	2446	3390	1960	6	31	193	6	299.00	2	4.8

In [63]:
```
train_data.isnull().sum()
```

Out[63]:
```
Title           0
Author          0
Edition         0
Ratings         0
Synopsis        0
Genre           0
BookCategory    0
Price           0
ID              0
Reviews         0
dtype: int64
```

In [64]:
```
train_data = train_data[~((train_data < (Q1 - 1.5 * IQR)) | (train_data > (Q3 + 1.5 * IQR))).any(axis=1)]
print(train_data.shape)
```

```
(4774, 10)
```

In [65]:
```
# Checking for outliers in the continuous variables
num_test_data = test_data[['Title', 'Author', 'Edition', 'Ratings', 'Synopsis', 'Genre','BookCategory',  'ID', 'Reviews']]
```

In [66]:
```
# Checking outliers at 25%, 50%, 75%, 90%, 95% and 99%
num_test_data.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[66]:

	Title	Author	Edition	Ratings	Synopsis	Genre	BookCategory	ID	Reviews
count	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000
mean	760.113462	605.391667	633.660897	33.667949	755.710897	92.969231	4.666667	779.500000	4.306410
std	439.987434	353.293793	351.290303	164.601527	438.503209	70.284461	3.247802	450.477524	0.667651
min	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	378.750000	299.750000	338.750000	2.000000	377.750000	22.000000	2.000000	389.750000	4.000000
50%	760.500000	614.000000	638.000000	6.000000	754.500000	73.000000	5.000000	779.500000	4.400000
75%	1144.250000	910.250000	938.250000	20.000000	1135.250000	150.250000	7.000000	1169.250000	4.900000
90%	1367.100000	1093.000000	1116.100000	56.000000	1364.100000	189.000000	9.000000	1403.100000	5.000000
95%	1444.050000	1158.050000	1189.050000	124.000000	1442.050000	208.050000	10.000000	1481.050000	5.000000

	Title	Author	Edition	Ratings	Synopsis	Genre	BookCategory	ID	Reviews
99%	1505.410000	1209.410000	1243.410000	476.940000	1502.410000	220.410000	10.000000	1543.410000	5.000000

In [67]:
```
Q1 = test_data.quantile(0.25)
Q3 = test_data.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Title           765.50
Author          610.50
Edition         599.50
Reviews           0.90
Ratings          18.00
Synopsis        757.50
Genre           128.25
BookCategory      5.00
ID              779.50
dtype: float64
```

In [68]:
```
test_data["Price"] = 0.0
```

In [69]:
```
X_test = test_data.drop(columns="Price")
```

In [70]:
```
y_test = test_data["Price"]
```

In [71]:
```
X_train = train_data.drop(columns = "Price")
```

In [72]:
```
y_train = train_data["Price"]
```

In [73]:
```
print(X_test.shape)
print(y_test.shape)
print(X_train.shape)
print(y_train.shape)
```

```
(1560, 9)
(1560,)
(4774, 9)
(4774,)
```

FEATURE SELECTION

In [74]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

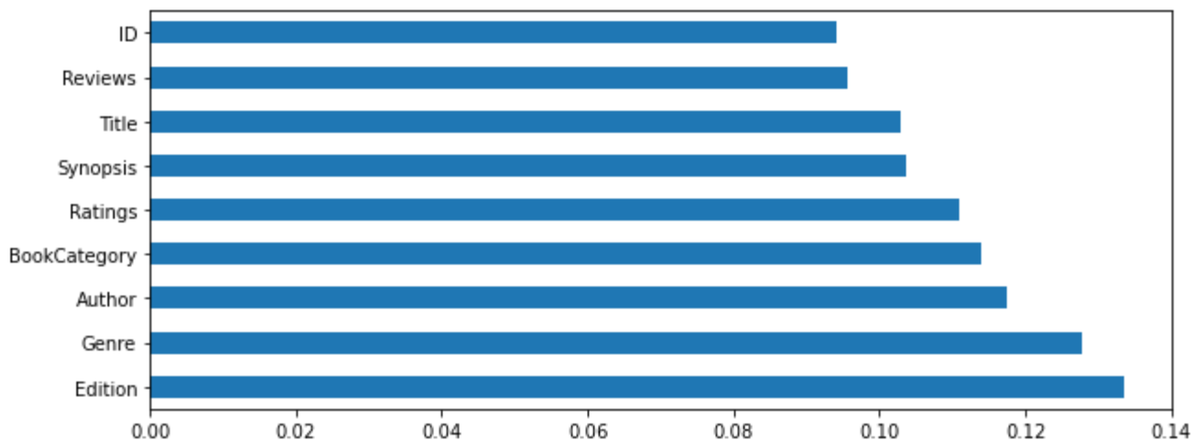
In [75]:
```
from sklearn.ensemble import ExtraTreesRegressor
import matplotlib.pyplot as plt
model=ExtraTreesRegressor()
model.fit(X_train,y_train)
```

Out[75]:
```
ExtraTreesRegressor()
```

In [76]:
```
print(model.feature_importances_)
```

```
[0.1028797  0.11754755 0.13344148 0.11095595 0.10372177 0.12775022
 0.11402371 0.09398671 0.09569291]
```

In [77]:
```
plt.figure(figsize = [10,4])
ranked_features=pd.Series(model.feature_importances_,index=X_train.columns)
ranked_features.nlargest(12).plot(kind='barh')
plt.show()
```



In [78]:
```
### Importance Of The Features Wrt, `Label/Target Variable`
```

In [79]:
```
ranked_features.nlargest(12, keep='all')
```

Out[79]:
```
Edition          0.133441
Genre            0.127750
```

```
Author           0.117548
BookCategory     0.114024
Ratings          0.110956
Synopsis         0.103722
Title            0.102880
Reviews          0.095693
ID               0.093987
```

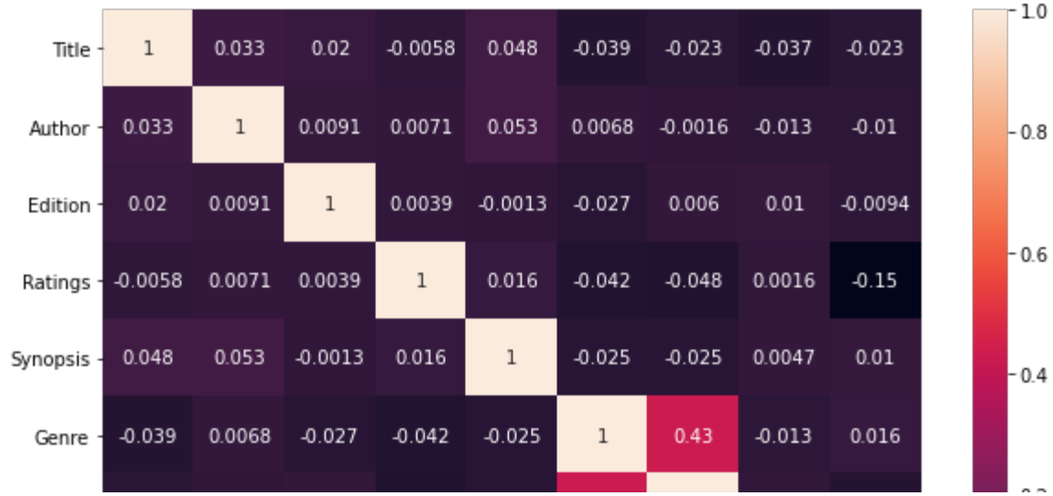
In [80]:
```
X_train.corr()
```

Out[80]:

	Title	Author	Edition	Ratings	Synopsis	Genre	BookCategory	ID	Reviews
Title	1.000000	0.032515	0.020409	-0.005835	0.047760	-0.038733	-0.022812	-0.036652	-0.022817
Author	0.032515	1.000000	0.009085	0.007083	0.053211	0.006838	-0.001552	-0.013237	-0.009990
Edition	0.020409	0.009085	1.000000	0.003875	-0.001336	-0.027304	0.005993	0.010339	-0.009369
Ratings	-0.005835	0.007083	0.003875	1.000000	0.015626	-0.041764	-0.048433	0.001639	-0.148433
Synopsis	0.047760	0.053211	-0.001336	0.015626	1.000000	-0.025052	-0.025396	0.004719	0.010483
Genre	-0.038733	0.006838	-0.027304	-0.041764	-0.025052	1.000000	0.429704	-0.013345	0.016264
BookCategory	-0.022812	-0.001552	0.005993	-0.048433	-0.025396	0.429704	1.000000	0.001259	-0.040549
ID	-0.036652	-0.013237	0.010339	0.001639	0.004719	-0.013345	0.001259	1.000000	0.028557
Reviews	-0.022817	-0.009990	-0.009369	-0.148433	0.010483	0.016264	-0.040549	0.028557	1.000000

In [81]:
```
import seaborn as sns
corr=X_train.corr()
top_features=corr.index
plt.figure(figsize=(10,7))
sns.heatmap(X_train[top_features].corr(),annot=True)
```

Out[81]: <AxesSubplot:>



Reduction Of `Multi Collinearity`

In [82]:
```
threshold=0.6
```

In [83]:
```
# find and remove correlated features
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

In [84]:
```
correlation(X_train,threshold)
```

Out[84]: set()

In [85]:
```
#### Information Gain
```

In [86]:
```
from sklearn.feature_selection import mutual_info_regression
```

In [87]:
```
mutual_info=mutual_info_regression(X_train,y_train)
```

In [88]:
```
mutual_data=pd.Series(mutual_info,index=X_train.columns)
mutual_data.sort_values(ascending=False)
```

Out[88]:
```
Genre           0.258042
Author          0.238431
Edition         0.175030
BookCategory    0.152386
Title           0.144648
Synopsis        0.118059
Ratings         0.090741
Reviews         0.059628
ID              0.000000
dtype: float64
```

In [89]:
```
X_train = X_train.drop(columns = ['ID'])
```

In [90]:
```
X_test = X_test.drop(columns = ['ID'])
```

In [91]:
```
# Importing RFE and LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

In [92]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [93]:
```
train_data.columns
```

Out[93]:
```
Index(['Title', 'Author', 'Edition', 'Ratings', 'Synopsis', 'Genre',
       'BookCategory', 'Price', 'ID', 'Reviews'],
      dtype='object')
```

In [94]:
```
X_train = train_data.drop(columns="Price")
```

In [95]:
```
y_train = train_data["Price"]
```

In [96]:
```
test_data["Price"] = 0
```

In [97]:
```
X_test = test_data.drop(columns="Price")
```

In [98]:
```
y_test = test_data["Price"]
```

Outlier Treatment

Perhaps the most important hyperparameter in the model is the “contamination” argument, which is used to help estimate the number of outliers in the dataset. This is a value between 0.0 and 0.5 and by default is set to 0.1.

Isolation Forest

In [99]:
```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import IsolationForest
from sklearn.metrics import mean_absolute_error
```

In [100…
```
# identify outliers in the training dataset
iso = IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train)
```

In [101…
```
# select all rows that are not outliers
mask = yhat != -1
```

In [102…
```
X_train = X_train[mask]
```

In [103…
```
y_train = y_train[mask]
```

In [104…
```
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
```

```
(4296, 9) (4296,)
```

In [107…
```
import xgboost as xgb
from xgboost.sklearn import XGBRegressor
```

In [112…
```
clf = XGBRegressor(learning_rate =0.1,n_estimators=1000,max_depth=5,min_child_weight=1,gamma=0,subsample=0.8,colsample_bytree=0.8,nthread=4,scale_pos_weight=
```

In [113…
```
#y_pred_test.to_csv("C:/Users/HP/Desktop/Predict_Book_Price/predictbookprice/Submission.csv")
```

In [114…
```
clf.fit(X_train, y_train)
```

Out[114…
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=5,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=1000, n_jobs=4, nthread=4, num_parallel_tree=1,
             random_state=2, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             seed=2, subsample=0.8, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

In [115…
```
y_pred = clf.predict(X_test)
```

In [116…
```
clf.score(X_train, y_train)
```

Out[116…
```
0.9762239272986748
```

In [117…
```
print(XGBRegressor())
```

```
XGBRegressor(base_score=None, booster=None, colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=None, gamma=None,
             gpu_id=None, importance_type='gain', interaction_constraints=None,
             learning_rate=None, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             random_state=None, reg_alpha=None, reg_lambda=None,
             scale_pos_weight=None, subsample=None, tree_method=None,
             validate_parameters=None, verbosity=None)
```

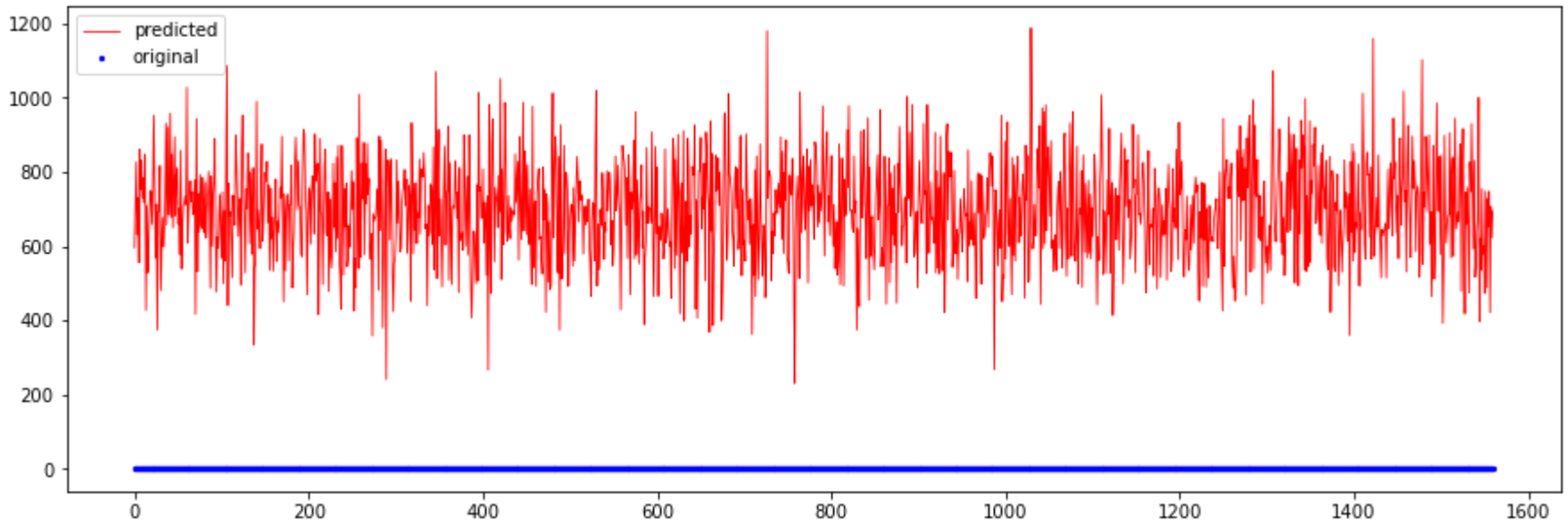
In [118…
```
y_pred = clf.predict(X_test)
mse = mean_squared_error(y_test,y_pred)
```

In [119…
```
print("MSE: %.2f" % mse)
```

```
MSE: 505042.39
```

In [120…

```
plt.figure(figsize = [15,5])
x_ax = range(len(y_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, y_pred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



In [121…

```
y_pred
```

Out[121…

```
array([596.712  , 718.8384 , 826.682  , ..., 707.77515, 623.84766,
       695.37195], dtype=float32)
```

In [133…

```
param_grid = {
    "n_estimators"     : [50, 100, 150 ,200],
    "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
     "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
     "min_child_weight" : [ 1, 3, 5, 7 ],
     "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
     "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
}
```

In [134…

```
from sklearn.model_selection import RandomizedSearchCV
```

In [135…

```
import xgboost as xgb
from xgboost.sklearn import XGBRegressor
rf = XGBRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = param_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
```

In [136…

```
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   12.7s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   58.8s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  1.7min finished
```

Out[136…

```
RandomizedSearchCV(cv=3,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_child_weight=None, missing=nan,
                                          monotone_constraints=None,
                                          n_estimators=100, n...
                                          scale_pos_weight=None, subsample=None,
                                          tree_method=None,
                                          validate_parameters=None,
                                          verbosity=None),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7],
                                        'n_estimators': [50, 100, 150, 200]},
                   random_state=42, verbose=2)
```

In [143…

```
rf_random.best_params_
```

Out[143…

```
{'n_estimators': 200,
 'min_child_weight': 3,
 'max_depth': 12,
 'learning_rate': 0.05,
```

```
'gamma': 0.0,
'colsample_bytree': 0.4}
```

In [144…]
```
rf_random
```

Out[144…]
```
RandomizedSearchCV(cv=3,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_child_weight=None, missing=nan,
                                          monotone_constraints=None,
                                          n_estimators=100, n...
                                          scale_pos_weight=None, subsample=None,
                                          tree_method=None,
                                          validate_parameters=None,
                                          verbosity=None),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7],
                                        'n_estimators': [50, 100, 150, 200]},
                   random_state=42, verbose=2)
```

In [145…]
```
best_random_grid=rf_random.best_estimator_
```

In [146…]
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
y_pred=best_random_grid.predict(X_test)
```

In [147…]
```
y_pred
```

Out[147…]
```
array([517.5785 , 591.13696, 493.46722, ..., 585.2463 , 655.39746,
       576.8665 ], dtype=float32)
```

In [148…]
```
best_random_grid.score(X_train, y_train)
```

Out[148…]
```
0.9646400699817028
```

In [149…

```
### GRID SEARCH VIEW
```

In [162…

```
rf_random.best_params_
```

Out[162…

```
{'n_estimators': 200,
 'min_child_weight': 3,
 'max_depth': 12,
 'learning_rate': 0.05,
 'gamma': 0.0,
 'colsample_bytree': 0.4}
```

In [169…

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [rf_random.best_params_['n_estimators'] - 50, rf_random.best_params_['n_estimators'], rf_random.best_params_['n_estimators'] + 50],
    'min_child_weight': [rf_random.best_params_['min_child_weight'] - 2, rf_random.best_params_['min_child_weight'], rf_random.best_params_['min_child_weight
    'max_depth': [rf_random.best_params_['max_depth'] - 5, rf_random.best_params_['max_depth'], rf_random.best_params_['learning_rate'] + 5],
    'learning_rate': [rf_random.best_params_['learning_rate']],
    'gamma': [rf_random.best_params_['gamma']],
    'colsample_bytree': [rf_random.best_params_['colsample_bytree']]
}

print(param_grid)
```

```
{'n_estimators': [150, 200, 250], 'min_child_weight': [1, 3, 5], 'max_depth': [7, 12, 5.05], 'learning_rate': [0.05], 'gamma': [0.0], 'colsample_bytree': [0.
4]}
```

In [170…

```
grid_search=GridSearchCV(estimator=rf,param_grid=param_grid,cv=2,n_jobs=-1,verbose=1)
grid_search.fit(X_train,y_train)
```

```
Fitting 2 folds for each of 27 candidates, totalling 54 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   21.2s
[Parallel(n_jobs=-1)]: Done  54 out of  54 | elapsed:   22.3s finished
```

Out[170…

```
GridSearchCV(cv=2,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None, gamma=None,
                                    gpu_id=None, importance_type='gain',
                                    interaction_constraints=None,
                                    learning_rate=None, max_delta_step=None,
                                    max_depth=None, min_child_weight=None,
                                    missing=nan, monotone_constraints=None,
                                    n_estimators=100, n_jobs=None,
                                    num_parallel_tree=None, random_state=None,
                                    reg_alpha=None, reg_lambda=None,
```

```
                           scale_pos_weight=None, subsample=None,
                           tree_method=None, validate_parameters=None,
                           verbosity=None),
             n_jobs=-1,
             param_grid={'colsample_bytree': [0.4], 'gamma': [0.0],
                         'learning_rate': [0.05], 'max_depth': [7, 12, 5.05],
                         'min_child_weight': [1, 3, 5],
                         'n_estimators': [150, 200, 250]},
             verbose=1)
```

In [171…

```
columns = [f"param_{name}" for name in param_grid.keys()]
columns += ["mean_test_score", "rank_test_score"]
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results["mean_test_score"] = -cv_results["mean_test_score"]
cv_results[columns].sort_values(by="rank_test_score")
```

Out[171…

	param_n_estimators	param_min_child_weight	param_max_depth	param_learning_rate	param_gamma	param_colsample_bytree	mean_test_score	rank_test_score
16	200	5	12	0.05	0.0	0.4	-0.245712	1
17	250	5	12	0.05	0.0	0.4	-0.245470	2
15	150	5	12	0.05	0.0	0.4	-0.244550	3
12	150	3	12	0.05	0.0	0.4	-0.238120	4
11	250	1	12	0.05	0.0	0.4	-0.237740	5
10	200	1	12	0.05	0.0	0.4	-0.237426	6
9	150	1	12	0.05	0.0	0.4	-0.235598	7
13	200	3	12	0.05	0.0	0.4	-0.235002	8
8	250	5	7	0.05	0.0	0.4	-0.230095	9
14	250	3	12	0.05	0.0	0.4	-0.229616	10
2	250	1	7	0.05	0.0	0.4	-0.219198	11
1	200	1	7	0.05	0.0	0.4	-0.217359	12
7	200	5	7	0.05	0.0	0.4	-0.217349	13
0	150	1	7	0.05	0.0	0.4	-0.212491	14
5	250	3	7	0.05	0.0	0.4	-0.209798	15
4	200	3	7	0.05	0.0	0.4	-0.197091	16
3	150	3	7	0.05	0.0	0.4	-0.192876	17
6	150	5	7	0.05	0.0	0.4	-0.190408	18

	param_n_estimators	param_min_child_weight	param_max_depth	param_learning_rate	param_gamma	param_colsample_bytree	mean_test_score	rank_test_score
18	150	1	5.05	0.05	0.0	0.4	NaN	19
19	200	1	5.05	0.05	0.0	0.4	NaN	20
20	250	1	5.05	0.05	0.0	0.4	NaN	21
21	150	3	5.05	0.05	0.0	0.4	NaN	22
22	200	3	5.05	0.05	0.0	0.4	NaN	23
23	250	3	5.05	0.05	0.0	0.4	NaN	24
24	150	5	5.05	0.05	0.0	0.4	NaN	25
25	200	5	5.05	0.05	0.0	0.4	NaN	26

In [172…
```
grid_search.best_estimator_
```

Out[172…
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0.0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=12,
             min_child_weight=5, missing=nan, monotone_constraints='()',
             n_estimators=200, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [173…
```
best_grid=grid_search.best_estimator_
```

In [174…
```
best_grid
```

Out[174…
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0.0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=12,
             min_child_weight=5, missing=nan, monotone_constraints='()',
             n_estimators=200, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [175…
```
best_grid.fit(X_train, y_train)
```

Out[175…
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0.0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=12,
             min_child_weight=5, missing=nan, monotone_constraints='()',
             n_estimators=200, n_jobs=4, num_parallel_tree=1, random_state=0,
```

```
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
```

In [176…

```
y_pred = best_grid.predict(X_test)
mse = mean_squared_error(y_test,y_pred)
```

In [177…

```
best_grid.score(X_train, y_train)
```

Out[177… 0.9427738122370792

In [178…

```
y_pred = best_grid.predict(X_test)
```

In [179…

```
import optuna
import sklearn
```

In [180…

```
param_grid = {
    "n_estimators"     : [50, 100, 150 ,200],
    "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
     "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
     "min_child_weight" : [ 1, 3, 5, 7 ],
     "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
     "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
}
```

In [184…

```
def objective(trial):
    gamma = trial.suggest_float('gamma',0.0, 0.6)
    colsample_bytree = trial.suggest_float('colsample_bytree', 0.3, 0.8)
    n_estimators = trial.suggest_int('n_estimators', 50, 200)
    max_depth = trial.suggest_int('max_depth', 3, 15)
    min_child_weight =  trial.suggest_int('min_child_weight', 1, 7)

    regr = XGBRegressor(gamma = gamma,
                                colsample_bytree = colsample_bytree, n_estimators = n_estimators,
                                max_depth = max_depth,min_child_weight = min_child_weight)


    #regr.fit(X_train, y_train)
    #y_pred = regr.predict(X_val)
    #return r2_score(y_val, y_pred)

    score = cross_val_score(regr, X_train, y_train, cv=5, scoring="r2")
    r2_mean = score.mean()

    return r2_mean
```

In [185…

```
from sklearn.model_selection import cross_val_score
```

In [186…

```
#Execute optuna and set hyperparameters
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=30)
```

```
[I 2021-12-21 20:58:12,995] A new study created in memory with name: no-name-636f5da7-ef7f-4ee1-8492-b4295f6dcb36
[I 2021-12-21 20:58:17,166] Trial 0 finished with value: 0.1950692264091573 and parameters: {'gamma': 0.5567132817118967, 'colsample_bytree': 0.7472399217584744, 'n_estimators': 89, 'max_depth': 10, 'min_child_weight': 6}. Best is trial 0 with value: 0.1950692264091573.
[I 2021-12-21 20:58:32,434] Trial 1 finished with value: 0.13589383758900248 and parameters: {'gamma': 0.10509903686672448, 'colsample_bytree': 0.7245560717087705, 'n_estimators': 167, 'max_depth': 15, 'min_child_weight': 1}. Best is trial 0 with value: 0.1950692264091573.
[I 2021-12-21 20:58:34,206] Trial 2 finished with value: 0.19490393484187502 and parameters: {'gamma': 0.5562962132634843, 'colsample_bytree': 0.6023784375609362, 'n_estimators': 54, 'max_depth': 6, 'min_child_weight': 4}. Best is trial 0 with value: 0.1950692264091573.
[I 2021-12-21 20:58:38,796] Trial 3 finished with value: 0.18793741719954102 and parameters: {'gamma': 0.33567731598972983, 'colsample_bytree': 0.7552283830640516, 'n_estimators': 190, 'max_depth': 6, 'min_child_weight': 4}. Best is trial 0 with value: 0.1950692264091573.
[I 2021-12-21 20:58:42,733] Trial 4 finished with value: 0.2244034191650064 and parameters: {'gamma': 0.036392349504368246, 'colsample_bytree': 0.3919444670094215 4, 'n_estimators': 70, 'max_depth': 12, 'min_child_weight': 3}. Best is trial 4 with value: 0.2244034191650064.
[I 2021-12-21 20:58:44,112] Trial 5 finished with value: 0.23281632622485987 and parameters: {'gamma': 0.2601266752563528, 'colsample_bytree': 0.4392438536384836, 'n_estimators': 67, 'max_depth': 6, 'min_child_weight': 6}. Best is trial 5 with value: 0.23281632622485987.
[I 2021-12-21 20:58:51,494] Trial 6 finished with value: 0.19790237620719442 and parameters: {'gamma': 0.4925361783245785, 'colsample_bytree': 0.6060017247260505, 'n_estimators': 151, 'max_depth': 9, 'min_child_weight': 1}. Best is trial 5 with value: 0.23281632622485987.
[I 2021-12-21 20:59:02,358] Trial 7 finished with value: 0.18645700273093857 and parameters: {'gamma': 0.26351536376253465, 'colsample_bytree': 0.6250679740393272, 'n_estimators': 179, 'max_depth': 12, 'min_child_weight': 2}. Best is trial 5 with value: 0.23281632622485987.
[I 2021-12-21 20:59:05,297] Trial 8 finished with value: 0.17325753912900024 and parameters: {'gamma': 0.2888242994404972, 'colsample_bytree': 0.7692937727406037, 'n_estimators': 65, 'max_depth': 7, 'min_child_weight': 3}. Best is trial 5 with value: 0.23281632622485987.
```

```
[I 2021-12-21 20:59:14,667] Trial 9 finished with value: 0.21963904180009813 and parameters: {'gamma': 0.12092014657064043, 'colsample_bytree': 0.534637348758
4535, 'n_estimators': 132, 'max_depth': 13, 'min_child_weight': 1}. Best is trial 5 with value: 0.23281632622485987.
[I 2021-12-21 20:59:16,065] Trial 10 finished with value: 0.2200266082569601 and parameters: {'gamma': 0.4021093954671955, 'colsample_bytree': 0.3294126498820
52, 'n_estimators': 99, 'max_depth': 3, 'min_child_weight': 7}. Best is trial 5 with value: 0.23281632622485987.
[I 2021-12-21 20:59:17,413] Trial 11 finished with value: 0.21846411538872781 and parameters: {'gamma': 0.01011643668861234, 'colsample_bytree': 0.40050657132
398815, 'n_estimators': 81, 'max_depth': 3, 'min_child_weight': 6}. Best is trial 5 with value: 0.23281632622485987.
[I 2021-12-21 20:59:19,909] Trial 12 finished with value: 0.21256063835603492 and parameters: {'gamma': 0.1707829061524178, 'colsample_bytree': 0.443416775114
3295, 'n_estimators': 51, 'max_depth': 12, 'min_child_weight': 5}. Best is trial 5 with value: 0.23281632622485987.
[I 2021-12-21 20:59:25,984] Trial 13 finished with value: 0.2342375788206831 and parameters: {'gamma': 0.0699027488275057, 'colsample_bytree': 0.3051022376216
5245, 'n_estimators': 115, 'max_depth': 15, 'min_child_weight': 3}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:27,948] Trial 14 finished with value: 0.20508632475702093 and parameters: {'gamma': 0.20387488587611902, 'colsample_bytree': 0.30243935198
166466, 'n_estimators': 125, 'max_depth': 5, 'min_child_weight': 7}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:33,456] Trial 15 finished with value: 0.2039387540827954 and parameters: {'gamma': 0.4003987205872743, 'colsample_bytree': 0.4881560019647
8776, 'n_estimators': 105, 'max_depth': 15, 'min_child_weight': 5}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:36,711] Trial 16 finished with value: 0.22798741629414043 and parameters: {'gamma': 0.07462732467887583, 'colsample_bytree': 0.35330793465
910393, 'n_estimators': 112, 'max_depth': 8, 'min_child_weight': 3}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:39,431] Trial 17 finished with value: 0.19563592490378529 and parameters: {'gamma': 0.21013787802249545, 'colsample_bytree': 0.460401020444
15874, 'n_estimators': 141, 'max_depth': 5, 'min_child_weight': 5}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:43,952] Trial 18 finished with value: 0.22218629556305686 and parameters: {'gamma': 0.3790278025587815, 'colsample_bytree': 0.300877034383
7144, 'n_estimators': 113, 'max_depth': 10, 'min_child_weight': 2}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:46,351] Trial 19 finished with value: 0.2127270145780476 and parameters: {'gamma': 0.15545216502334586, 'colsample_bytree': 0.386275234029
8028, 'n_estimators': 88, 'max_depth': 8, 'min_child_weight': 6}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:49,400] Trial 20 finished with value: 0.21502208597007977 and parameters: {'gamma': 0.23238917865642317, 'colsample_bytree': 0.500707837505
5832, 'n_estimators': 155, 'max_depth': 4, 'min_child_weight': 2}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:53,792] Trial 21 finished with value: 0.22865391392763504 and parameters: {'gamma': 0.07578070163623357, 'colsample_bytree': 0.352750823505
78684, 'n_estimators': 116, 'max_depth': 8, 'min_child_weight': 3}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 20:59:56,911] Trial 22 finished with value: 0.2188041324232941 and parameters: {'gamma': 0.06030042473379821, 'colsample_bytree': 0.356303753629
7798, 'n_estimators': 124, 'max_depth': 7, 'min_child_weight': 3}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 21:00:00,080] Trial 23 finished with value: 0.22521349075759117 and parameters: {'gamma': 0.011659032072823206, 'colsample_bytree': 0.438809432
9582203, 'n_estimators': 76, 'max_depth': 10, 'min_child_weight': 4}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 21:00:03,379] Trial 24 finished with value: 0.23237407371407226 and parameters: {'gamma': 0.13526833061178778, 'colsample_bytree': 0.316272505012
7722, 'n_estimators': 103, 'max_depth': 8, 'min_child_weight': 2}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 21:00:05,276] Trial 25 finished with value: 0.23040940573181157 and parameters: {'gamma': 0.15342466710201724, 'colsample_bytree': 0.302872067031
6201, 'n_estimators': 96, 'max_depth': 6, 'min_child_weight': 2}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 21:00:07,847] Trial 26 finished with value: 0.23379977802443497 and parameters: {'gamma': 0.32158786473682077, 'colsample_bytree': 0.414404742802
0279, 'n_estimators': 65, 'max_depth': 9, 'min_child_weight': 2}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 21:00:11,256] Trial 27 finished with value: 0.22126247283931386 and parameters: {'gamma': 0.3355288719854962, 'colsample_bytree': 0.417869055013
88016, 'n_estimators': 62, 'max_depth': 14, 'min_child_weight': 4}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 21:00:13,382] Trial 28 finished with value: 0.1986345297990432 and parameters: {'gamma': 0.49159358752046955, 'colsample_bytree': 0.555510218753
4033, 'n_estimators': 50, 'max_depth': 11, 'min_child_weight': 5}. Best is trial 13 with value: 0.2342375788206831.
[I 2021-12-21 21:00:14,783] Trial 29 finished with value: 0.22931727171264565 and parameters: {'gamma': 0.45096344241799774, 'colsample_bytree': 0.484564873240
```

In [187…

```
#Create an instance with tuned hyperparameters
optimised_rf = XGBRegressor(gamma = study.best_params['gamma'],
                            colsample_bytree = study.best_params['colsample_bytree'], n_estimators = study.best_params['n_estimators'],
                            max_depth = study.best_params['max_depth'],min_child_weight = study.best_params['min_child_weight'])
#learn
optimised_rf.fit(X_train ,y_train)
```

Out[187…

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.30510223762165245,
             gamma=0.0699027488275057, gpu_id=-1, importance_type='gain',
             interaction_constraints='', learning_rate=0.300000012,
```

```
max_delta_step=0, max_depth=15, min_child_weight=3, missing=nan,
monotone_constraints='()', n_estimators=115, n_jobs=4,
num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, subsample=1, tree_method='exact',
validate_parameters=1, verbosity=None)
```

In [188…
```
optimised_rf.score(X_train, y_train)
```

Out[188…
```
0.9992409672765115
```

In [189…
```
trial = study.best_trial
print('Accuracy: {}'.format(trial.value))
```

```
Accuracy: 0.2342375788206831
```

In [190…
```
print("Best hyperparameters: {}".format(trial.params))
```

```
Best hyperparameters: {'gamma': 0.0699027488275057, 'colsample_bytree': 0.30510223762165245, 'n_estimators': 115, 'max_depth': 15, 'min_child_weight': 3}
```

In [191…
```
optuna.visualization.plot_optimization_history(study)
```

In [192…
```
y_pred = optimised_rf.predict(X_test)
```

In [193…
```
y_pred = pd.DataFrame(y_pred)
```

In [194…
```
y_pred = y_pred.rename(columns ={0: "Price"} )
```

In [195…
```
y_pred
```

Out[195…

	Price
0	687.505005
1	407.896271
2	519.974182
3	595.488464
4	618.317932
...	...
1555	529.734863
1556	446.834106
1557	542.328003
1558	505.965118
1559	467.936157

1560 rows × 1 columns

In [196…
```
y_pred.to_csv("C:/Users/HP/Desktop/Predict_Book_Price/predictbookprice/Submission_xgb.csv")
```

In [ ]: