Regression

Agenda

In this lesson, we will cover the following concepts with the help of a business use case:

- Use Case: Regression
- Regression Algorithms
- Types of Model Evaluation Metrics
- Gradient Descent
- Types of Gradient Descents
- Use Case: Stochastic Gradient Descent (SGD)

Use Case: Regression

Note: At first, with the help of a use case, we are going to perform all the basic steps to reach the model training and predicition part.

Problem Statement:

Google Play Store team is about to launch a new feature wherein, certain apps that are promising are boosted in visibility. The boost will manifest in multiple ways including higher priority in recommendations sections ("Similar apps", "You might also like", "New and updated games"). These will also get a boost in search results visibility. This feature will help bring more attention to newer apps that have the potential.

Analysis to be done:

The problem is to identify the apps that are going to be good for Google to promote. App ratings, which are provided by the customers, are always great indicators of the goodness of the app. The problem reduces to: predict which apps will have high ratings.

Dataset

Google Play Store data (**googleplaystore.csv**)

Link: https://www.dropbox.com/sh/i06ohrau3ucfgbm/AACeYXumL56543KnDNQFlj8ma?dl=0

Data Dictionary:

Variables	Description
App	Application name
Category	Category to which the app belongs
Rating	Overall user rating of the app
Reviews	Number of user reviews for the app
Size	Size of the app
Installs	Number of user downloads/installs for the app
Type	Paid or Free
Price	Price of the app
Content Rating	Age group the app is targeted at - Children / Mature 21+ / Adult
Genres	An app can belong to multiple genres (apart from its main category) For example, a musical family game will belong to Music, Game, Family genres
Last Updated	Date when the app was last updated on Play Store
Current Ver	Current version of the app available on Play Store
Android Ver	Minimum required Android version

Solution:

Import Libraries

In [1]:

```
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
```

Import and Check Dataset

In [2]:

```
inp0 = pd.read_csv("googleplaystore.csv")
```

In [3]:

```
# Check first five rows
inp0.head(2)
```

Out[3]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content_Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art_&_Design	January 7, 2018	1.0.0	[illegible]
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art_&_Design_Pretend_Play	January 15, 2018	2.0.0	[illegible]

Observations

The data will be displayed on the screen.

In [4]:

```
#Check number of columns and rows, and data types
inp0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   App             10841 non-null  object
 1   Category        10841 non-null  object
 2   Rating          9367 non-null   float64
 3   Reviews         10841 non-null  object
 4   Size            10841 non-null  object
 5   Installs        10841 non-null  object
 6   Type            10840 non-null  object
 7   Price           10841 non-null  object
 8   Content_Rating  10840 non-null  object
 9   Genres          10841 non-null  object
 10  Last Updated    10841 non-null  object
 11  Current Ver     10833 non-null  object
 12  Android Ver     10838 non-null  object
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

Check Data Types

In [5]:

```
#checking datatypes
inp0.dtypes
```

Out[5]:

```
App                object
Category           object
Rating            float64
Reviews            object
Size               object
Installs           object
Type               object
Price              object
Content_Rating     object
Genres             object
[illegible]
```

[illegible]

Handling the Variables

1. Clean the price column

In [9]:

```
#Cleaning the price column
inp0.Price.value_counts()[:5]
```

Out[9]:

```
0        8715
$2.99     114
$0.99     106
$4.99      70
$1.99      59
Name: Price, dtype: int64
```

Observations

Some have dollars, some have 0

- We need to conditionally handle this.
- First, let's modify the column to take 0 if value is 0, else take the first letter onwards.

In [10]:

```
#Modifying the column
inp0['Price'] = inp0.Price.map(lambda x: 0 if x=='0' else float(x[1:]))
```

The other columns with numeric data are:

1. Reviews
2. Installs
3. Size

2. Convert reviews to numeric

In [11]:

```
#Converting reviews to numeric
inp0.Reviews = inp0.Reviews.astype("int32")
```

In [12]:

```
inp0.Reviews.describe()
```

Out[12]:

```
count    9.360000e+03
mean     5.143767e+05
std      3.145023e+06
min      1.000000e+00
25%      1.867500e+02
50%      5.955000e+03
75%      8.162750e+04
max      7.815831e+07
Name: Reviews, dtype: float64
```

3. Handle the installs column

In [13]:

```
#Handling the installs column
inp0.Installs.value_counts()
```

Out[13]:

```
1,000,000+        1576
10,000,000+       1252
100,000+          1150
10,000+           1009
5,000,000+         752
1,000+             712
500,000+           537
50,000+            466
5,000+             431
100,000,000+       409
100+               309
50,000,000+        289
500+               201
500,000,000+        72
10+                 69
1,000,000,000+      58
50+                 56
5+                   9
1+                   3
Name: Installs, dtype: int64
```

We'll need to remove the commas and the plus signs.

Defining function for the same

In [14]:

```
def clean_installs(val):
    return int(val.replace(",","").replace("+",""))
```

In [15]:

```
inp0.Installs = inp0.Installs.map(clean_installs)
```

In [16]:

```
inp0.Installs.describe()
```

Out[16]:

```
count    9.360000e+03
mean     1.790875e+07
std      9.126637e+07
min      1.000000e+00
25%      1.000000e+04
50%      5.000000e+05
75%      5.000000e+06
max      1.000000e+09
Name: Installs, dtype: float64
```

4. Handle the app size field

In [17]:

```
#Handling the app size field
def change_size(size):
    if 'M' in size:
        x = size[:-1]
        x = float(x)*1000
        return(x)
    elif 'k' == size[-1:]:
        x = size[:-1]
        x = float(x)
        return(x)
    else:
        return None
```

In [18]:

```
inp0["Size"] = inp0["Size"].map(change_size)
```

In [19]:

```
inp0.Size.describe()
```

Out[19]:

```
count      7723.000000
mean      22970.456105
std       23449.628935
min           8.500000
25%        5300.000000
50%       14000.000000
75%       33000.000000
max      100000.000000
Name: Size, dtype: float64
```

In [20]:

```
#Filling Size which had NA
inp0.Size.fillna(method = 'ffill', inplace = True)
```

In [21]:

```
#Checking datatypes
inp0.dtypes
```

Out[21]:

```
App                object
Category           object
Rating            float64
Reviews             int32
Size              float64
Installs            int64
Type               object
Price             float64
Content_Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

Sanity checks

1. Average rating should be between 1 and 5, as only these values are allowed on Play Store. Drop any rows that have a value outside this range.

In [22]:

```
#Checking the rating
inp0.Rating.describe()
```

Out[22]:

```
count    9360.000000
mean        4.191838
std         0.515263
min         1.000000
25%         4.000000
50%         4.300000
75%         4.500000
max         5.000000
Name: Rating, dtype: float64
```

Observations

Min is 1 and max is 5. None of the values have rating outside the range.

1. Reviews should not be more than installs as only those who installed can review the app.

Checking if reviews are more than installs. Counting total rows like this.

In [23]:

```
#Checking and counting the rows
len(inp0[inp0.Reviews > inp0.Installs])
```

Out[23]:

```
7
```

In [24]:

```
inp0[inp0.Reviews > inp0.Installs]
```

Out[24]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content_Rating	Genres	Last Updated	Current Ver	Android Ver
2454	KBA-EZ Health Guide	MEDICAL	5.0	4	25000.0	1	Free	0.00	Everyone	Medical	August 2, 2018	1.0.72	4.0.3 and up
4663	Alarmy (Sleep If U Can) - Pro	LIFESTYLE	4.8	10249	30000.0	10000	Paid	2.49	Everyone	Lifestyle	July 30, 2018	Varies with device	Varies with device
5917	Ra Ga Ba	GAME	5.0	2	20000.0	1	Paid	1.49	Everyone	Arcade	February 8, 2017	1.0.4	2.3 and up
6700	Brick Breaker BR	GAME	5.0	7	19000.0	5	Free	0.00	Everyone	Arcade	July 23, 2018	1	4.1 and up
7402	Trovami se ci riesci	GAME	5.0	11	6100.0	10	Free	0.00	Everyone	Arcade	March 11, 2017	0.1	2.3 and up
8591	DN Blog	SOCIAL	5.0	20	4200.0	10	Free	0.00	Teen	Social	July 23, 2018	1	4.0 and up
10697	Mu.F.O.	GAME	5.0	2	16000.0	1	Paid	0.99	Everyone	Arcade	March 3, 2017	1	2.3 and up

In [25]:

```
inp0 = inp0[inp0.Reviews <= inp0.Installs].copy()
```

In [26]:

```
inp0.shape
```

Out[26]:

```
(9353, 13)
```

1. For free apps **(Type == "Free")**, the price should not be **> 0**. Drop any such rows.

In [27]:

```
len(inp0[(inp0.Type == "Free") & (inp0.Price>0)])
```

Out[27]:

```
0
```

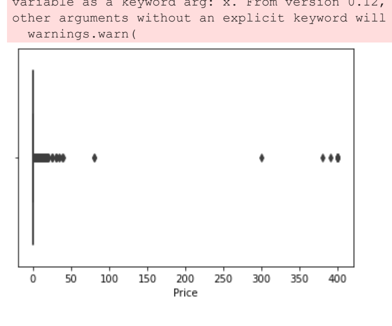
EDA

Box Plot: Price

In [28]:

```
#Are there any outliers? Think about the price of usual apps on the Play Store.
sns.boxplot(inp0.Price)
plt.show()
```

```
C:\Users\alpika.gupta\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

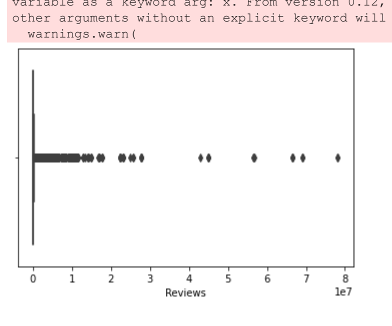


Box Plot: Reviews

In [29]:

```
#Are there any apps with very high number of reviews? Do the values seem right?
sns.boxplot(inp0.Reviews)
plt.show()
```

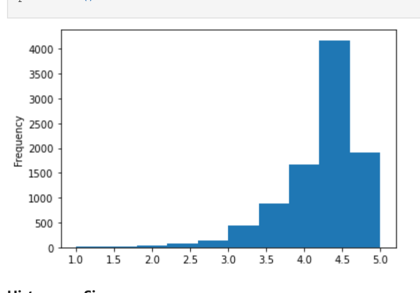
```
C:\Users\alpika.gupta\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



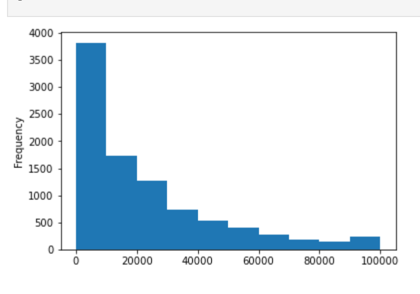
Checking Distribution and Skewness:

How are the ratings distributed? Is it more toward higher ratings?

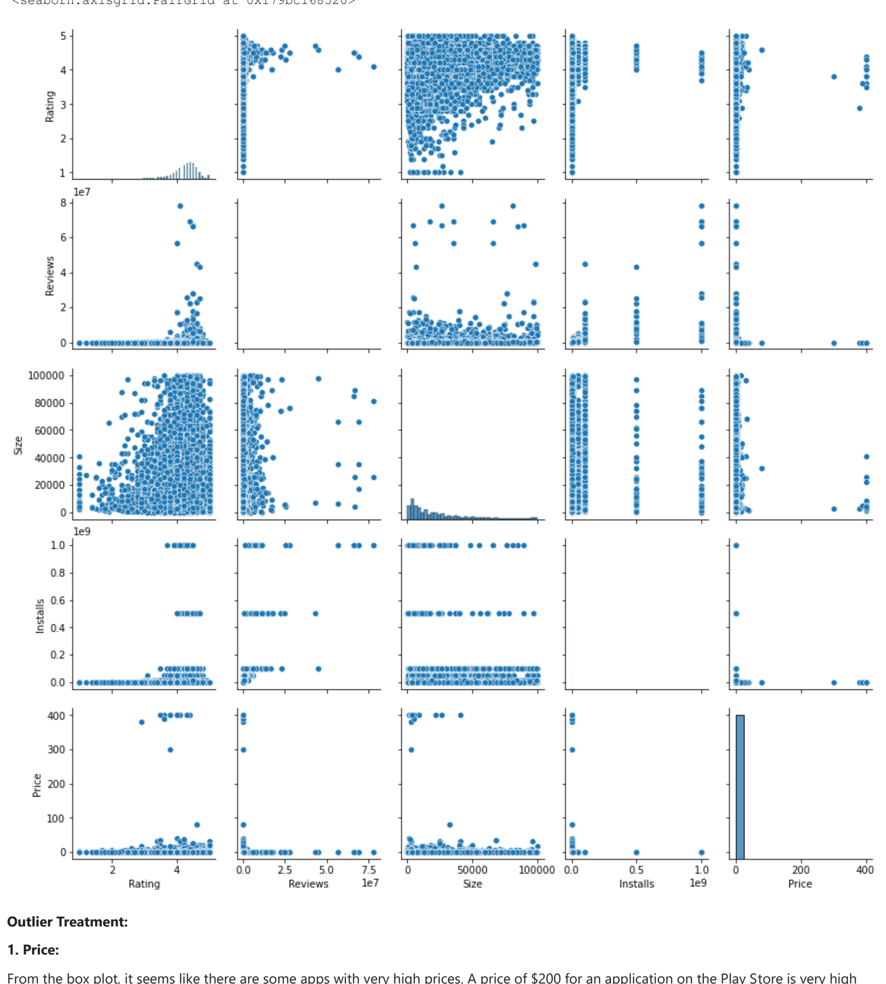
[illegible]



[illegible]



[illegible]



[illegible]

1. Price:

From the box plot, it seems like there are some apps with very high prices. A price of $200 for an application on the Play Store is very high and suspicious. Check the records that have very high price: Is 200 a high price?

In [33]:

```
#Checking the records
len(inp0[inp0.Price > 200])
```

Out[33]:

```
15
```

In [34]:

```
inp0[inp0.Price > 200]
```

Out[34]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content_Rating	Genres	Last Updated	Current Ver	Android Ver
4197	most expensive app (H)	FAMILY	4.3	6	1500.0	100	Paid	399.99	Everyone	Entertainment	July 16, 2018	1	7.0 and up
4362	💎 I'm rich	LIFESTYLE	3.8	718	26000.0	10000	Paid	399.99	Everyone	Lifestyle	March 11, 2018	1.0.0	4.4 and up
4367	I'm Rich - Trump Edition	LIFESTYLE	3.6	275	7300.0	10000	Paid	400.00	Everyone	Lifestyle	May 3, 2018	1.0.1	4.1 and up
5351	I am rich	LIFESTYLE	3.8	3547	1800.0	100000	Paid	399.99	Everyone	Lifestyle	January 12, 2018	2	4.0.3 and up
5354	I am Rich Plus	FAMILY	4.0	856	8700.0	10000	Paid	399.99	Everyone	Entertainment	May 19, 2018	3	4.4 and up
5355	I am rich VIP	LIFESTYLE	3.8	411	2600.0	10000	Paid	299.99	Everyone	Lifestyle	July 21, 2018	1.1.1	4.3 and up
5356	I Am Rich Premium	FINANCE	4.1	1867	4700.0	50000	Paid	399.99	Everyone	Finance	November 12, 2017	1.6	4.0 and up
5357	I am extremely Rich	LIFESTYLE	2.9	41	2900.0	1000	Paid	379.99	Everyone	Lifestyle	July 1, 2018	1	4.0 and up
5358	I am Rich!	FINANCE	3.8	93	22000.0	1000	Paid	399.99	Everyone	Finance	December 11, 2017	1	4.1 and up
5359	I am rich(premium)	FINANCE	3.5	472	965.0	5000	Paid	399.99	Everyone	Finance	May 1, 2017	3.4	4.4 and up
5362	I Am Rich Pro	FAMILY	4.4	201	2700.0	5000	Paid	399.99	Everyone	Entertainment	May 30, 2017	1.54	1.6 and up
5364	I am rich (Most expensive app)	FINANCE	4.1	129	2700.0	1000	Paid	399.99	Teen	Finance	December 6, 2017	2	4.0.3 and up
5366	I Am Rich	FAMILY	3.6	217	4900.0	10000	Paid	389.99	Everyone	Entertainment	June 22, 2018	1.5	4.2 and up
5369	I am Rich	FINANCE	4.3	180	3800.0	5000	Paid	399.99	Everyone	Finance	March 22, 2018	1	4.2 and up
5373	I AM RICH PRO PLUS	FINANCE	4.0	36	41000.0	1000	Paid	399.99	Everyone	Finance	June 25, 2018	1.0.2	4.1 and up

In [35]:

```
inp0 = inp0[inp0.Price <= 200].copy()
inp0.shape
```

Out[35]:

```
(9338, 13)
```

2. Reviews:

Very few apps have very high number of reviews. These are all star apps that don't help with the analysis and, in fact, will skew it. Drop records having more than 2 million reviews.

In [36]:

```
#Dropping the records with more than 2 million reviews
inp0 = inp0[inp0.Reviews <= 2000000]
inp0.shape
```

Out[36]:

```
(8885, 13)
```

3. Installs:

There seem to be some outliers in this field too. Apps having a very high number of installs should be dropped from the analysis. Find out the different percentiles – 10, 25, 50, 70, 90, 95, 99.

Decide a threshold as the cutoff for outliers and drop records having values more than the threshold.

In [37]:

```
#Dropping the apps that have a very high number of installs
inp0.Installs.quantile([0.1, 0.25, 0.5, 0.70, 0.9, 0.95, 0.99])
```

Out[37]:

```
0.10         1000.0
0.25        10000.0
0.50       500000.0
0.70      1000000.0
0.90     10000000.0
0.95     10000000.0
0.99    100000000.0
Name: Installs, dtype: float64
```

Observations

Looks like there are just 1% of apps having more than 100M installs. These apps might be genuine, but will definitely skew our analysis. We need to drop these.

In [38]:

```
#Dropping the apps with more than 100M installs
len(inp0[inp0.Installs >= 100000000])
```

Out[38]:

```
6
```

In [39]:

```
inp0 = inp0[inp0.Installs < 100000000].copy()
inp0.shape
```

Out[39]:

```
(8879, 13)
```

In [40]:

```
#Importing warnings
import warnings
warnings.filterwarnings("ignore")
```

Bi-variate Analysis:

Let's look at how the available predictors relate to the variable of interest, i.e., our target variable rating. Make scatter plots (for numeric features) and box plots (for character features) to assess the relationhips between rating and the other features.

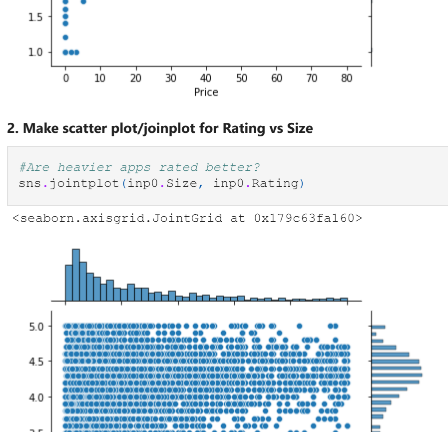
1. Make scatter plot/join plot for Rating vs. Price

In [41]:

```
#What pattern do you observe? Does rating increase with price?
sns.jointplot(inp0.Price, inp0.Rating)
```

Out[41]:

```
<seaborn.axisgrid.JointGrid at 0x179c5f68640>
```



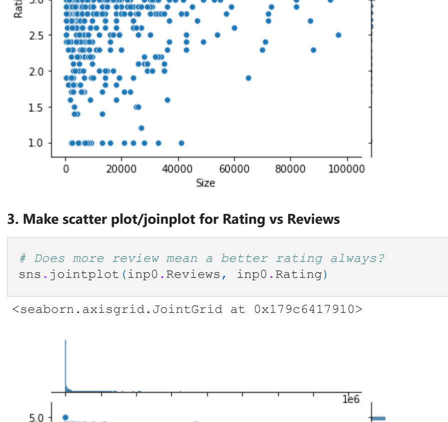
2. Make scatter plot/joinplot for Rating vs Size

In [42]:

```
#Are heavier apps rated better?
sns.jointplot(inp0.Size, inp0.Rating)
```

Out[42]:

```
<seaborn.axisgrid.JointGrid at 0x179c63fa160>
```



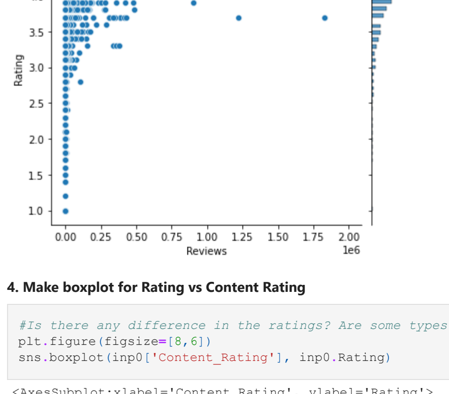
3. Make scatter plot/joinplot for Rating vs Reviews

In [43]:

```
# Does more review mean a better rating always?
sns.jointplot(inp0.Reviews, inp0.Rating)
```

Out[43]:

```
<seaborn.axisgrid.JointGrid at 0x179c6417910>
```



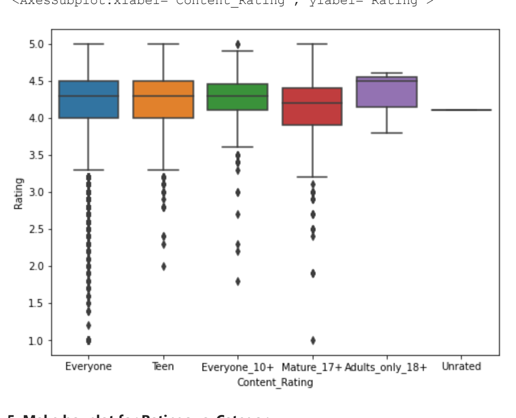
4. Make boxplot for Rating vs Content Rating

In [44]:

```
#Is there any difference in the ratings? Are some types liked better?
plt.figure(figsize=[8,6])
sns.boxplot(inp0['Content_Rating'], inp0.Rating)
```

Out[44]:

```
<AxesSubplot:xlabel='Content_Rating', ylabel='Rating'>
```



5. Make boxplot for Ratings vs. Category

In [45]:

```
#Which genre has the best ratings?
plt.figure(figsize=[18,6])
g = sns.boxplot(inp0.Category, inp0.Rating)
plt.xticks(rotation=90)
```

Out[45]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]),
 [Text(0, 0, 'ART_AND_DESIGN'),
  Text(1, 0, 'AUTO_AND_VEHICLES'),
  Text(2, 0, 'BEAUTY'),
  Text(3, 0, 'BOOKS_AND_REFERENCE'),
  Text(4, 0, 'BUSINESS'),
  Text(5, 0, 'COMICS'),
  Text(6, 0, 'COMMUNICATION'),
  Text(7, 0, 'DATING'),
  Text(8, 0, 'EDUCATION'),
  Text(9, 0, 'ENTERTAINMENT'),
  Text(10, 0, 'EVENTS'),
  Text(11, 0, 'FINANCE'),
  Text(12, 0, 'FOOD_AND_DRINK'),
  Text(13, 0, 'HEALTH_AND_FITNESS'),
  Text(14, 0, 'HOUSE_AND_HOME'),
  Text(15, 0, 'LIBRARIES_AND_DEMO'),
  Text(16, 0, 'LIFESTYLE'),
  Text(17, 0, 'GAME'),
  Text(18, 0, 'FAMILY'),
  Text(19, 0, 'MEDICAL'),
  Text(20, 0, 'SOCIAL'),
  Text(21, 0, 'SHOPPING'),
  Text(22, 0, 'PHOTOGRAPHY'),
  Text(23, 0, 'SPORTS'),
  Text(24, 0, 'TRAVEL_AND_LOCAL'),
  Text(25, 0, 'TOOLS'),
  Text(26, 0, 'PERSONALIZATION'),
  Text(27, 0, 'PRODUCTIVITY'),
  Text(28, 0, 'PARENTING'),
  Text(29, 0, 'WEATHER'),
  Text(30, 0, 'VIDEO_PLAYERS'),
  Text(31, 0, 'NEWS_AND_MAGAZINES'),
  Text(32, 0, 'MAPS_AND_NAVIGATION')])
```

Pre-processing the Dataset

1. Make a copy of the dataset

In [46]:
```
#Making a copy
inp1 = inp0.copy()
```

2. Apply log transformation (np.log1p) to Reviews and Installs

Reviews and Installs have some values that are still relatively very high. Before building a linear regression model, you need to reduce the skew.

In [47]:
```
#Reducing the skew
inp0.Installs.describe()
```

Out[47]:
```
count    8.879000e+03
mean     5.595862e+06
std      2.421042e+07
min      5.000000e+00
25%      1.000000e+04
50%      5.000000e+05
75%      5.000000e+06
max      5.000000e+08
Name: Installs, dtype: float64
```

In [48]:
```
inp1.Installs = inp1.Installs.apply(np.log1p)
```

In [49]:
```
inp1.Reviews = inp1.Reviews.apply(np.log1p)
```

3. Drop columns App, Last Updated, Current Ver, and Android Ver

These variables are not useful for our task.

In [50]:
```
inp1.dtypes
```

Out[50]:
```
App                object
Category           object
Rating            float64
Reviews           float64
Size              float64
Installs          float64
Type               object
Price             float64
Content_Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

In [51]:
```
#Dropping the variables that are not useful for our task
inp1.drop(["App", "Last Updated", "Current Ver", "Android Ver"], axis=1, inplace=True)
inp1.shape
```

Out[51]:
```
(8879, 9)
```

4. Dummy Columns:

Get dummy columns for Category, Genres, and Content Rating. This needs to be done as the models do not understand categorical data, and all data should be numeric. Dummy encoding is one way to convert character fields to numeric fields. Name of the dataframe should be **inp2**.

In [52]:
```
inp2 = pd.get_dummies(inp1, drop_first=True)
```

In [53]:
```
inp2.columns
```

Out[53]:
```
Index(['Rating', 'Reviews', 'Size', 'Installs', 'Price',
       'Category_AUTO_AND_VEHICLES', 'Category_BEAUTY',
       'Category_BOOKS_AND_REFERENCE', 'Category_BUSINESS', 'Category_COMICS',
       ...
       'Genres_Tools', 'Genres_Tools_Education', 'Genres_Travel_&_Local',
       'Genres_Travel_&_Local_Action_&_Adventure', 'Genres_Trivia',
       'Genres_Video_Players_&_Editors',
       'Genres_Video_Players_&_Editors_Creativity',
       'Genres_Video_Players_&_Editors_Music_&_Video', 'Genres_Weather',
       'Genres_Word'],
      dtype='object', length=157)
```

In [54]:
```
inp2.shape
```

Out[54]:
```
(8879, 157)
```

Train-test split

Let us distribute the data into **training** and **test** datasets using the **train_test_split()** function.

In [55]:
```
from sklearn.model_selection import train_test_split
```

In [56]:
```
#train_test_split
```

In [57]:
```
df_train, df_test = train_test_split(inp2, train_size = 0.7, random_state = 100)
```

In [58]:
```
df_train.shape, df_test.shape
```

Out[58]:
```
((6215, 157), (2664, 157))
```

Let us separate the dataframes into **X_train, y_train, X_test, y_test.**

In [59]:
```
y_train = df_train.pop("Rating")
X_train = df_train
```

In [60]:
```
X_train.head(1)
```

Out[60]:

	Reviews	Size	Installs	Price	Category_AUTO_AND_VEHICLES	Category_BEAUTY	Category_BOOKS_AND_REFERENCE	Category_BUSINE
1279	7.63627	6900.0	11.512935	0.0	0	0	0	

1 rows × 156 columns

In [61]:
```
y_test = df_test.pop("Rating")
X_test = df_test
```

In [62]:
```
X_test.head(1)
```

Out[62]:

	Reviews	Size	Installs	Price	Category_AUTO_AND_VEHICLES	Category_BEAUTY	Category_BOOKS_AND_REFERENCE	Category_BUSIN
1161	9.329456	24000.0	13.815512	0.0	0	0	0	

1 rows × 156 columns

Regression Algorithms:

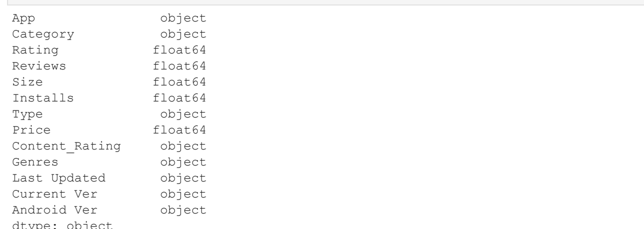
Note: Let us take a look at the theory part before moving on to the training and prediction.

Types of Regression Algorithms:

- Linear regression
- Multiple linear regression
- Polynomial regression
- Ridge regression
- Lasso regression
- ElasticNet regression

When to use regression?

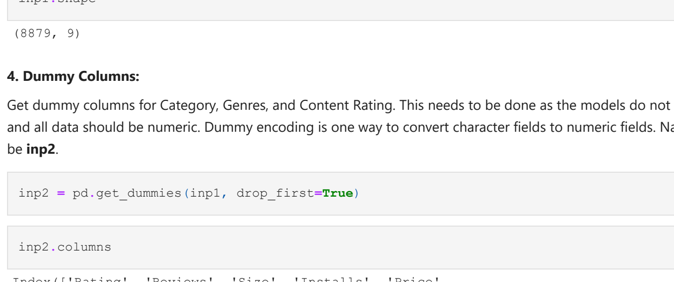
If target variable is a continuous numeric variable (100–2000), then use a regression algorithm.



Note: Let us take a look at the basics of linear regression and then move on to the model builiding part where we are going to use all the concepts that we saw in previous sessions.

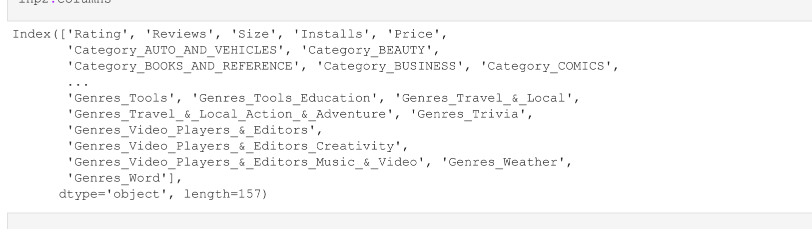
1. Linear Regression:

Linear Regression is a statistical model used to predict the relationship between independent and dependent variables denoted by x and y respectively.



2. Muliple Linear Regression:

Multiple linear regression is a statistical technique used to predict the outcome of a response variable through several explanatory variables and model the relationships between them.



3. Polynomial Regression:

Polynomial regression is applied when data is not formed in a straight line. It is used to fit a linear model to non-linear data by creating new features from powers of non-linear features.

Example: Quadratic features

Types of Model Evaluation Metrics:

Assumption

Let us consider the following:

y_i – the observed value

$\bar{y}$ – the mean value of a sample

$\hat{y}_i$ – the value estimated by the regression line

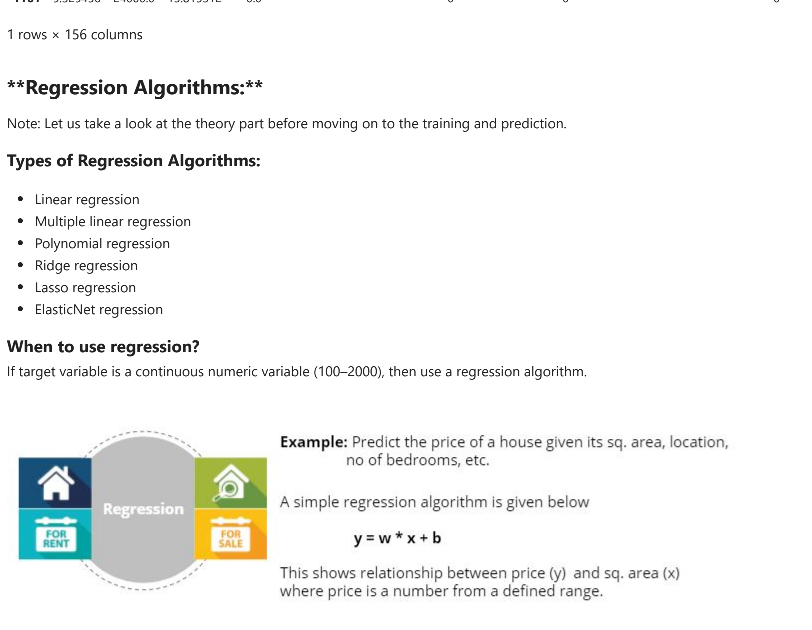
Sum of Squares Total (SST)

The squared variations between the measured dependent variable and its mean are referred to as the **Sum of Squares Total (SST)** or **Total Sum of Squares (TSS)**.

It's similar to the variation of descriptive statistics in that it's the dispersion of measured variables around the mean.

It is a measure of the dataset's overall variability.

$$SST = SSR + SSE = \sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



$$\sum_{i=1}^{n}(y_i - \bar{y})^2$$

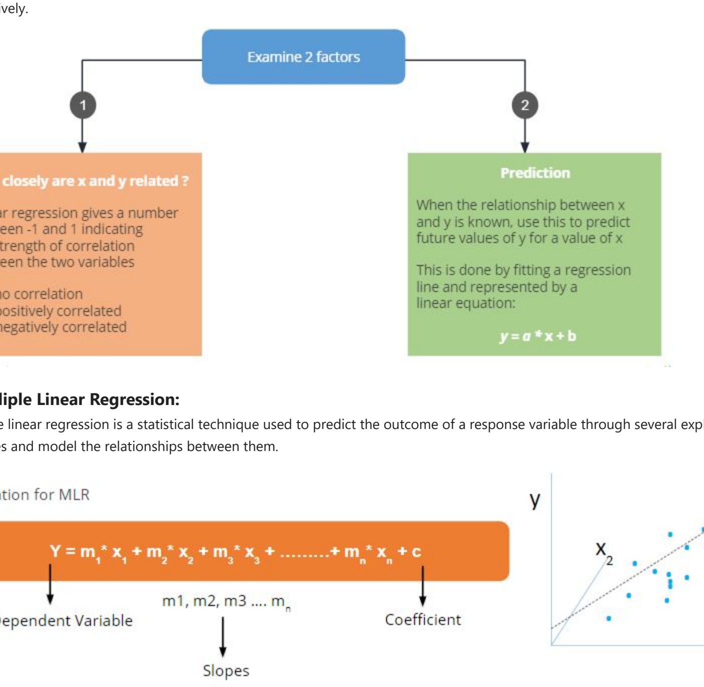
Sum of Squares due to Regression (SSR)

The difference between the predicted value and the dependent variable's mean are referred to as the **Sum of Squares due to Regression (SSR)** or **Explained Sum of Squares (ESS)**.

It can be considered as a metric for describing how well our line fits the data.

If the SSR (or ESS) is equal to the SST (or TSS), the regression model is flawless and captures all observed variability.

$$SSR = \sum_{i=1}^{n}(y_i - \bar{y})^2$$



$$\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$$

Sum of Squares Error (SSE)

The difference between the observed and predicted values are referred to as the **Sum of Squares Error (SSE)** or **Residual Sum of Squares (RSS)**, where `residual` stands for `remaining` or `unexplained`.

This error must be reduced since the smaller it is, the better the regression's estimation power.

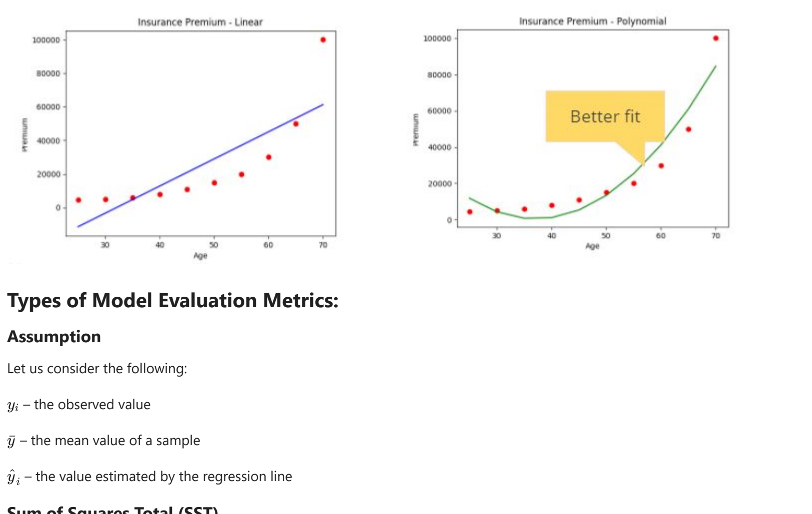
$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$Or$$

$$SSE = \sum_{i=1}^{n} e_i^2$$

$$where,$$

$$e_i = (y_i - \hat{y}_i)$$



$$\sum_{i=1}^{n} e_i^2$$

Relation Among SST, SSR, and SSE

Since certain people use these abbreviations in various ways, it can be very confusing.

We use one of two sets of notations for these abbreviations: SST, SSR, and SSE or TSS, ESS, and RSS.

These equations are related in the following ways:

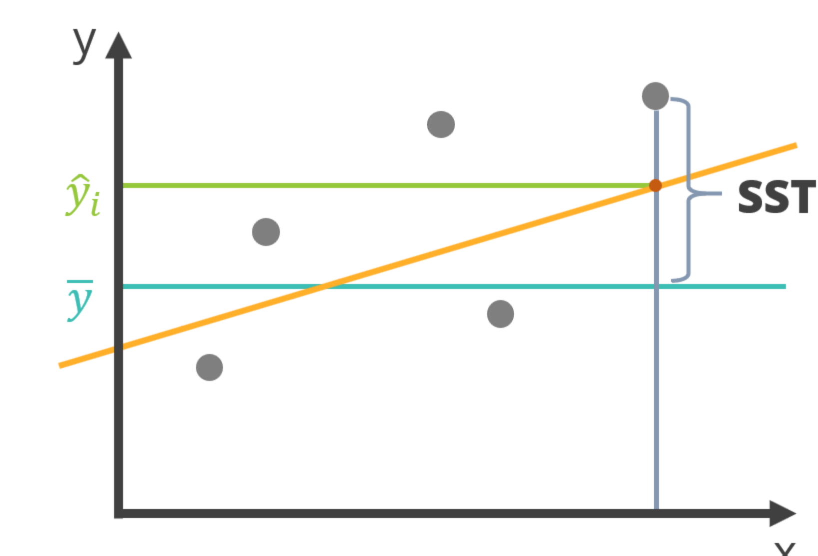
$$SST = SSR + SSE$$

$$Or$$

$$TSS = ESS + RSS$$

This is because the overall variability of the dataset is equivalent to the variability described by the regression line and the unknown variability (also known as error).

For a constant total variability, a lower error would result in a better regression. A higher error, on the other hand, would result in a weaker regression. This should always be remembered regardless of the notation set used.



$$\sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n} e_i^2$$

R-Square Matrix

The determination coefficient also known as **R2 (R-squared) score** is used for the performance evaluation of a linear regression model.

R2 displays the proportion of data points inside the regression equation line.

A higher R2 value means improved results.

It is calculated as follows:

$$R^2 = 1 - \frac{SSE}{SSR}$$

$$Or$$

$$R^2 = 1 - \frac{RSS}{ESS}$$

The highest possible score is 1, which is achieved when the predicted and actual values are the same.

The R2 score is 0 for a baseline model.

In the worst-case scenario, the R2 score can also be negative.

Import *statsmodels* Library for Linear Regression

In [63]:
```
#Importing the statsmodel library
import statsmodels.api as sm
```

In [64]:
```
#Applying linear regression
model1 = sm.OLS(y_train, X_train)
```

In [65]:
```
model1 = model1.fit()
```

In [66]:
```
#Finding the summary
model1.summary()
```

Out[66]:

OLS Regression Results

Dep. Variable:	Rating	**R-squared (uncentered):**	0.987
Model:	OLS	**Adj. R-squared (uncentered):**	0.987
Method:	Least Squares	**F-statistic:**	3717.
Date:	Fri, 19 Aug 2022	**Prob (F-statistic):**	0.00
Time:	22:24:49	**Log-Likelihood:**	-4281.7
No. Observations:	6215	**AIC:**	8811.
Df Residuals:	6091	**BIC:**	9646.
Df Model:	124		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Reviews	0.1752	0.006	28.037	0.000	0.163	0.187
Size	-2.896e-07	3.37e-07	-0.859	0.391	-9.51e-07	3.72e-07
Installs	-0.1499	0.006	-24.184	0.000	-0.162	-0.138
Price	0.0006	0.003	0.177	0.860	-0.006	0.007
Category_AUTO_AND_VEHICLES	1.6381	0.125	13.136	0.000	1.394	1.883
Category_BEAUTY	1.7213	0.128	13.462	0.000	1.471	1.972
Category_BOOKS_AND_REFERENCE	1.6843	0.122	13.826	0.000	1.445	1.923
Category_BUSINESS	1.6046	0.121	13.272	0.000	1.368	1.842
Category_COMICS	2.3786	0.226	10.517	0.000	1.935	2.822
Category_COMMUNICATION	1.5478	0.121	12.765	0.000	1.310	1.786
Category_DATING	1.5013	0.122	12.262	0.000	1.261	1.741
Category_EDUCATION	2.8642	0.251	11.419	0.000	2.373	3.356
Category_ENTERTAINMENT	2.7757	0.251	11.065	0.000	2.284	3.267
Category_EVENTS	1.7447	0.126	13.792	0.000	1.497	1.993
Category_FAMILY	2.8988	0.243	11.920	0.000	2.422	3.376
Category_FINANCE	1.5685	0.121	12.976	0.000	1.332	1.806
Category_FOOD_AND_DRINK	1.5993	0.123	13.006	0.000	1.358	1.840
Category_GAME	3.1390	0.241	13.046	0.000	2.667	3.611
Category_HEALTH_AND_FITNESS	1.6296	0.121	13.464	0.000	1.392	1.867
Category_HOUSE_AND_HOME	1.6151	0.124	13.002	0.000	1.372	1.859
Category_LIBRARIES_AND_DEMO	1.6541	0.125	13.247	0.000	1.409	1.899
Category_LIFESTYLE	1.5913	0.121	13.156	0.000	1.354	1.828
Category_MAPS_AND_NAVIGATION	1.5212	0.123	12.399	0.000	1.281	1.762
Category_MEDICAL	1.6353	0.121	13.543	0.000	1.399	1.872
Category_NEWS_AND_MAGAZINES	1.5639	0.121	12.897	0.000	1.326	1.802
Category_PARENTING	2.5348	0.213	11.914	0.000	2.118	2.952
Category_PERSONALIZATION	1.6774	0.121	13.866	0.000	1.440	1.915
Category_PHOTOGRAPHY	1.5848	0.121	13.077	0.000	1.347	1.822
Category_PRODUCTIVITY	1.6213	0.121	13.399	0.000	1.384	1.859
Category_SHOPPING	1.6088	0.121	13.245	0.000	1.371	1.847
Category_SOCIAL	1.6116	0.121	13.299	0.000	1.374	1.849
Category_SPORTS	3.0934	0.421	7.352	0.000	2.269	3.918
Category_TOOLS	2.1455	0.228	9.412	0.000	1.699	2.592
Category_TRAVEL_AND_LOCAL	2.1361	0.228	9.359	0.000	1.689	2.583
Category_VIDEO_PLAYERS	2.8638	0.644	4.444	0.000	1.601	4.127
Category_WEATHER	1.6195	0.125	12.999	0.000	1.375	1.864
Type_Paid	-0.0839	0.032	-2.641	0.008	-0.146	-0.022
Content_Rating_Everyone	1.3855	0.239	5.807	0.000	0.918	1.853
Content_Rating_Everyone_10+	1.3848	0.240	5.760	0.000	0.913	1.856
Content_Rating_Mature_17+	1.3697	0.241	5.689	0.000	0.898	1.842
Content_Rating_Teen	1.3954	0.239	5.843	0.000	0.927	1.864
Content_Rating_Unrated	1.3438	0.543	2.477	0.013	0.280	2.407
Genres_Action_Action_&_Adventure	0.2739	0.153	1.793	0.073	-0.026	0.573
Genres_Adventure	-0.0817	0.078	-1.053	0.292	-0.234	0.070
Genres_Adventure_Action_&_Adventure	0.3259	0.352	0.925	0.355	-0.364	1.016
Genres_Adventure_Brain_Games	0.5704	0.493	1.157	0.247	-0.396	1.537
Genres_Adventure_Education	0.0545	0.353	0.155	0.877	-0.637	0.746
Genres_Arcade	0.0809	0.055	1.466	0.143	-0.027	0.189
Genres_Arcade_Action_&_Adventure	0.2865	0.164	1.746	0.081	-0.035	0.608
Genres_Arcade_Pretend_Play	0.5474	0.493	1.111	0.267	-0.419	1.513
Genres_Art_&_Design	3.4662	0.253	13.711	0.000	2.971	3.962
Genres_Art_&_Design_Creativity	2.5998	0.258	10.079	0.000	2.094	3.105
Genres_Art_&_Design_Pretend_Play	3.2802	0.543	6.039	0.000	2.215	4.345
Genres_Auto_&_Vehicles	1.6381	0.125	13.136	0.000	1.394	1.883
Genres_Beauty	1.7213	0.128	13.462	0.000	1.471	1.972
Genres_Board	0.0196	0.093	0.210	0.834	-0.163	0.202
Genres_Board_Action_&_Adventure	0.0976	0.493	0.198	0.843	-0.868	1.063
Genres_Board_Brain_Games	0.4762	0.171	2.782	0.005	0.141	0.812
Genres_Board_Pretend_Play	2.419e-11	5.21e-11	0.464	0.642	-7.8e-11	1.26e-10
Genres_Books_&_Reference	1.6843	0.122	13.826	0.000	1.445	1.923
Genres_Books_&_Reference_Education	0.3094	0.352	0.879	0.380	-0.381	1.000
Genres_Business	1.6046	0.121	13.272	0.000	1.368	1.842
Genres_Card	-0.1032	0.088	-1.175	0.240	-0.275	0.069
Genres_Card_Action_&_Adventure	0.0730	0.353	0.207	0.836	-0.619	0.765
Genres_Card_Brain_Games	0.7041	0.492	1.430	0.153	-0.261	1.669
Genres_Casino	0.0818	0.097	0.845	0.398	-0.108	0.272
Genres_Casual	0.1815	0.082	2.226	0.026	0.022	0.341
Genres_Casual_Action_&_Adventure	0.4097	0.159	2.577	0.010	0.098	0.721
Genres_Casual_Brain_Games	0.6562	0.185	3.550	0.000	0.294	1.018
Genres_Casual_Creativity	0.3377	0.227	1.489	0.136	-0.107	0.782
Genres_Casual_Education	0.3845	0.352	1.092	0.275	-0.306	1.075
Genres_Casual_Music_&_Video	0.3886	0.493	0.789	0.430	-0.577	1.354
Genres_Casual_Pretend_Play	0.2678	0.126	2.123	0.034	0.020	0.515
Genres_Comics	0.8128	0.185	4.385	0.000	0.449	1.176
Genres_Comics_Creativity	1.5659	0.337	4.653	0.000	0.906	2.226
Genres_Communication	1.5478	0.121	12.765	0.000	1.310	1.786
Genres_Communication_Creativity	0.4552	0.493	0.924	0.355	-0.510	1.421
Genres_Dating	1.5013	0.122	12.262	0.000	1.261	1.741
Genres_Education	0.4813	0.080	5.998	0.000	0.324	0.639
Genres_Education_Action_&_Adventure	0.8246	0.292	2.826	0.005	0.253	1.397
Genres_Education_Brain_Games	0.3587	0.293	1.223	0.222	-0.216	0.934
Genres_Education_Creativity	0.7808	0.232	3.370	0.001	0.327	1.235
Genres_Education_Education	0.5631	0.115	4.894	0.000	0.338	0.789
Genres_Education_Music_&_Video	0.3892	0.493	0.790	0.430	-0.577	1.355
Genres_Education_Pretend_Play	0.5627	0.133	4.236	0.000	0.302	0.823
Genres_Educational	0.0147	0.125	0.118	0.906	-0.230	0.260
Genres_Educational_Action_&_Adventure	0.3506	0.255	1.377	0.169	-0.149	0.850
Genres_Educational_Brain_Games	0.4424	0.255	1.737	0.082	-0.057	0.942
Genres_Educational_Creativity	0.2078	0.291	0.713	0.476	-0.364	0.779
Genres_Educational_Education	0.4526	0.118	3.822	0.000	0.221	0.685
Genres_Educational_Pretend_Play	0.3586	0.150	2.391	0.017	0.065	0.653
Genres_Entertainment	0.2606	0.079	3.294	0.001	0.106	0.416
Genres_Entertainment_Action_&_Adventure	0.3654	0.291	1.256	0.209	-0.205	0.935
Genres_Entertainment_Brain_Games	0.4481	0.199	2.248	0.025	0.057	0.839
Genres_Entertainment_Creativity	0.7096	0.354	2.007	0.045	0.017	1.403
Genres_Entertainment_Education	0.6296	0.493	1.278	0.201	-0.336	1.595
Genres_Entertainment_Music_&_Video	0.4017	0.155	2.598	0.009	0.099	0.705
Genres_Entertainment_Pretend_Play	0.0043	0.352	0.012	0.990	-0.686	0.695
Genres_Events	1.7447	0.126	13.792	0.000	1.497	1.993
Genres_Finance	1.5685	0.121	12.976	0.000	1.332	1.806
Genres_Food_&_Drink	1.5993	0.123	13.006	0.000	1.358	1.840
Genres_Health_&_Fitness	1.6296	0.121	13.464	0.000	1.392	1.867
Genres_Health_&_Fitness_Action_&_Adventure	-0.1025	0.493	-0.208	0.835	-1.068	0.863
Genres_Health_&_Fitness_Education	0.5455	0.492	1.108	0.268	-0.420	1.511
Genres_House_&_Home	1.6151	0.124	13.002	0.000	1.372	1.859
Genres_Libraries_&_Demo	1.6541	0.125	13.247	0.000	1.409	1.899
Genres_Lifestyle	1.5913	0.121	13.156	0.000	1.354	1.828
Genres_Lifestyle_Education	-5.31e-12	1.43e-11	-0.372	0.710	-3.33e-11	2.27e-11
Genres_Lifestyle_Pretend_Play	3.872e-12	7.93e-12	0.488	0.625	-1.17e-11	1.94e-11
Genres_Maps_&_Navigation	1.5212	0.123	12.399	0.000	1.281	1.762
Genres_Medical	1.6353	0.121	13.543	0.000	1.399	1.872
Genres_Music	-0.0491	0.134	-0.365	0.715	-0.313	0.215
Genres_Music_&_Audio_Music_&_Video	0.7194	0.493	1.460	0.144	-0.246	1.685
Genres_Music_Music_&_Video	0.7628	0.352	2.166	0.030	0.072	1.453
Genres_News_&_Magazines	1.5639	0.121	12.897	0.000	1.326	1.802
Genres_Parenting	0.8856	0.142	6.229	0.000	0.607	1.164
Genres_Parenting_Brain_Games	1.288e-12	2.25e-12	0.571	0.568	-3.13e-12	5.71e-12
Genres_Parenting_Education	0.6841	0.275	2.485	0.013	0.144	1.224
Genres_Parenting_Music_&_Video	0.9651	0.236	4.081	0.000	0.502	1.429
Genres_Personalization	1.6774	0.121	13.866	0.000	1.440	1.915
Genres_Photography	1.5848	0.121	13.077	0.000	1.347	1.822
Genres_Productivity	1.6213	0.121	13.399	0.000	1.384	1.859
Genres_Puzzle	0.3274	0.077	4.254	0.000	0.177	0.478
Genres_Puzzle_Action_&_Adventure	0.3040	0.230	1.320	0.187	-0.147	0.755
Genres_Puzzle_Brain_Games	0.4838	0.146	3.311	0.001	0.197	0.770
Genres_Puzzle_Creativity	0.5716	0.493	1.160	0.246	-0.394	1.537
Genres_Puzzle_Education	0.8685	0.492	1.764	0.078	-0.097	1.834
Genres_Racing	-0.0124	0.076	-0.162	0.871	-0.162	0.137

Genres_Racing_Action_&_Adventure	0.4695	0.150	3.126	0.002	0.175	0.764
Genres_Racing_Pretend_Play	0.9584	0.493	1.945	0.052	-0.008	1.925
Genres_Role_Playing	0.1774	0.085	2.094	0.036	0.011	0.344
Genres_Role_Playing_Action_&_Adventure	0.3217	0.230	1.399	0.162	-0.129	0.772
Genres_Role_Playing_Brain_Games	0	0	nan	nan	0	0
Genres_Role_Playing_Pretend_Play	0.7804	0.493	1.584	0.113	-0.186	1.746
Genres_Shopping	1.6088	0.121	13.245	0.000	1.371	1.847
Genres_Simulation	0.2076	0.082	2.538	0.011	0.047	0.368
Genres_Simulation_Action_&_Adventure	0.4333	0.176	2.460	0.014	0.088	0.779
Genres_Simulation_Education	0.2594	0.287	0.904	0.366	-0.303	0.822
Genres_Simulation_Pretend_Play	0.2141	0.291	0.737	0.461	-0.355	0.784
Genres_Social	1.6116	0.121	13.299	0.000	1.374	1.849
Genres_Sports	0.0669	0.346	0.193	0.847	-0.612	0.746
Genres_Sports_Action_&_Adventure	0.2767	0.291	0.951	0.342	-0.294	0.847
Genres_Strategy	0.1675	0.095	1.768	0.077	-0.018	0.353
Genres_Strategy_Action_&_Adventure	0.3910	0.493	0.793	0.428	-0.576	1.358
Genres_Strategy_Creativity	0	0	nan	nan	0	0
Genres_Strategy_Education	0.8569	0.493	1.740	0.082	-0.109	1.822
Genres_Tools	0.9933	0.181	5.473	0.000	0.638	1.349
Genres_Tools_Education	1.1522	0.334	3.445	0.001	0.497	1.808
Genres_Travel_&_Local	1.0204	0.183	5.578	0.000	0.662	1.379
Genres_Travel_&_Local_Action_&_Adventure	1.1156	0.335	3.335	0.001	0.460	1.771
Genres_Trivia	-0.1316	0.108	-1.221	0.222	-0.343	0.080
Genres_Video_Players_&_Editors	0.2532	0.603	0.420	0.675	-0.930	1.436
Genres_Video_Players_&_Editors_Creativity	0.0344	0.493	0.070	0.944	-0.931	1.000
Genres_Video_Players_&_Editors_Music_&_Video	0.1329	0.352	0.377	0.706	-0.558	0.823
Genres_Weather	1.6195	0.125	12.999	0.000	1.375	1.864
Genres_Word	0.1096	0.109	1.001	0.317	-0.105	0.324

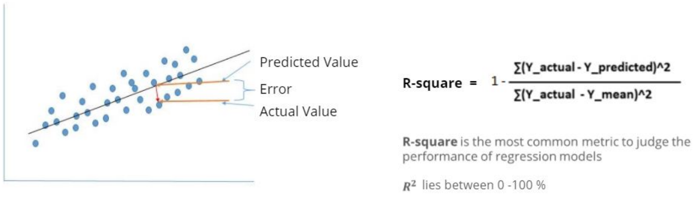
Omnibus:	2313.775	**Durbin-Watson:**	2.022
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	13556.581
Skew:	-1.673	**Prob(JB):**	0.00
Kurtosis:	9.416	**Cond. No.**	5.96e+20

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The smallest eigenvalue is 1.73e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Identified values for *model1*:

- **R^2 (uncentered): 0.987**

  Overview of R^2 (R-square):

  R-square = 1 - $\frac{\sum(Y\_actual - Y\_predicted)^2}{\sum(Y\_actual - Y\_mean)^2}$

  R-square is the most common metric to judge the performance of regression models

  R^2 lies between 0 -100 %

- **Adjusted R^2 (uncentered): 0.987**

  Overview of adjusted R^2:

  The disadvantage with R-squared is that it assumes every independent variable in the model and explains variations in the dependent variable. Use adjusted R-squared when working on a multiple linear regression problem.

  $$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

  where R^2 is R-squared value
  P is number of predictor variables
  N is number data points

- **F-statistic: 3717**

  Overview of F-statistic:

  The F-test indicates whether the linear regression model provides a better fit to the data in comparison to a model that contains no independent variables.

  Note: In general, the models with high F-statistic are considered as optimum.

  An F-test provides 2 values:

  - F-critical: This value is also called as F statistic
  - F-value: This value is without the “critical” part

    Note: The F-value is always observed along with the p-value. In general, the higher the F-value of a variable, the lesser the p-value.

- **AIC: 8811**

- **BIC: 9646**

  Overview of AIC and BIC:

  - AIC and BIC are penalized-likelihood criteria.
  - They are used for selecting the best predictor subsets in regression and comparing the nonnested models.
  - AIC: "It is an estimate of a constant plus the relative distance between the unknown true likelihood function of the data and the fitted likelihood function of the model."
  - BIC: "It is an estimate of a function of the posterior probability of a model being true, under a certain Bayesian setup."

  `Source: https://www.methodology.psu.edu/resources/AIC-vs-BIC/`

- **Formula:**

  The AIC or BIC for a model is usually written in the form:

  $[-2logL + kp]$

  Where:

  - L = likelihood function
  - p = number of parameters in the model
  - k = 2 for AIC and log(n) for BIC

- In popular opinion:
  - Lower AIC tells that the model is closer to the truth
  - Lower BIC tells that the model is more likely to be the true model

Prediction and evaluation using "model1"

In [67]:

```
#Prediction and evaluation
import sklearn.metrics as metrics
y_test_pred= model1.predict(X_test)
```

Metrics used for regression models:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Lower values of RMSE indicate better fit.

In [68]:

```
#Assessing the model using RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))

#RMSE of "model1" = 0.49203326281981113
```

```
0.4920332628197782
```

MPE:

Mean Percentage Error measures the errors that indicate whether or not the prediction is biased.

Bias is one component of the mean squared error and is measured as the variance of the errors plus the square of the mean error.

$$MSE = VAR(E) + (MPE)^2$$

Therefore, minimizing the MSE will implicitly minimize the bias and the variance of the errors.

MPE is the computed average of percentage errors by which prediction of a model differ from actual values of the quantity being predicted.

Formula:

$$MPE = \frac{100\%}{n}\sum_{i=1}^{n}\frac{a_i - p_i}{a_i}$$

Where:

- a_i = Actual value of the quantity being predicted
- p_i = Predicted value
- n = Number of different times for which the variable is predicted

```
In [69]:
#Calculating MPE
def MPE(y_test_pred,y_test):
    return ( ((y_test - y_test_pred) / y_test).mean()) * 100

print ('MPE: ' + str(MPE(y_test_pred,y_test)) )
```

```
MPE: -2.573037835061549
```

MAPE:

Mean Absolute Percentage Error measures the prediction accuracy of a model.

Formula:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i - P_i|}{|A_i|}$$

Where:

- A_i = Actual value of the quantity being predicted
- P_i = Predicted value
- n = Number of different times for which the variable is predicted

Mean Absolute Percentage Error is similar to Mean Absolute Error but is normalized by true observation.

The mean absolute percentage error is useful for reporting purpose and is expressed in generic percentage terms.

Note: In general, the desirable MAPE value is less than 20.

```
In [70]:
#Calculating MAPE
def MAPE(y_test_pred,y_test):
    return np.mean(np.abs((y_test - y_test_pred) / y_test)) * 100

print ('MAPE: ' + str(MAPE(y_test_pred,y_test)) )
```

```
MAPE: 9.49093433792546
```

Reiterating the Model

1. Feature selection using p-value:

In regression analysis, p-values and coefficients together indicate which relationships in the model are statistically significant and the nature of those relationships.

Coefficients describe the mathematical relationship between each independent variable and the dependent variable.

p-values for the coefficients indicate whether these relationships are statistically significant.

In general, the threshold for the p-value is taken as ≤ 0.05.

Let's find the variable with value ≤ 0.05.

2. List:

Identified 99 variables that have p-value ≤ 0.05.

'Rating','Reviews','Installs','Category_AUTO_AND_VEHICLES','Category_BEAUTY','Category_BOOKS_AND_REFERENCE', 'Category_BUSINESS','Category_COMICS','Category_COMMUNICATION','Category_DATING','Category_EDUCATION', 'Category_EDUCATION','Category_ENTERTAINMENT','Category_EVENTS','Category_FAMILY','Category_FINANCE', 'Category_FOOD_AND_DRINK','Category_GAME','Category_HEALTH_AND_FITNESS','Category_HOUSE_AND_HOME', 'Category_LIBRARIES_AND_DEMO','Category_LIFESTYLE','Category_MAPS_AND_NAVIGATION','Category_MEDICAL', 'Category_NEWS_AND_MAGAZINES','Category_PARENTING','Category_PERSONALIZATION','Category_PHOTOGRAPHY', 'Category_PRODUCTIVITY','Category_SHOPPING','Category_SOCIAL','Category_SPORTS','Category_TOOLS', 'Category_TRAVEL_AND_LOCAL','Category_VIDEO_PLAYERS','Category_WEATHER','Type_Paid', 'Content_Rating_Everyone','Content_Rating_Everyone_10+','Content_Rating_Mature_17+', 'Content_Rating_Teen','Content_Rating_Unrated','Genres*Art*&_Design','Genres*Art*&_Design;Creativity', 'Genres*Art*&_Design;Pretend_Play','Genres*Auto*&_Vehicles','Genres_Beauty','Genres_Board;Brain_Games', 'Genres*Books*&_Reference','Genres_Business','Genres_Casual','Genres*Casual;Action*&_Adventure', 'Genres_Casual;Brain_Games','Genres_Casual;Pretend_Play','Genres_Comics','Genres_Comics;Creativity', 'Genres_Communication','Genres_Dating','Genres_Education','Genres*Education;Action*&_Adventure', 'Genres_Education;Creativity','Genres_Education;Education','Genres_Education;Pretend_Play', 'Genres_Educational;Education','Genres_Educational;Pretend_Play','Genres_Entertainment', 'Genres_Entertainment;Brain_Games','Genres_Entertainment;Creativity','Genres*Entertainment; Music*&_Video','Genres_Events','Genres_Finance','Genres*Food*&_Drink','Genres*Health*&_Fitness', 'Genres*House*&_Home','Genres*Libraries*&_Demo','Genres_Lifestyle','Genres*Maps*&_Navigation', 'Genres_Medical','Genres*Music;Music*&_Video','Genres*News*&_Magazines','Genres_Parenting', 'Genres_Parenting;Education','Genres*Parenting;Music*&_Video','Genres_Personalization', 'Genres_Photography','Genres_Productivity','Genres_Puzzle','Genres_Puzzle;Brain_Games', 'Genres*Racing;Action*&_Adventure','Genres_Role_Playing','Genres_Shopping','Genres_Simulation', 'Genres*Simulation;Action*&_Adventure','Genres_Social','Genres_Strategy','Genres_Tools', 'Genres_Tools;Education','Genres*Travel*&_Local','Genres*Travel&Local;Action*&_Adventure','Genres_Weather'

3. Create a new dataset with chosen variables and split it into training and testing dataset

```
In [71]:
#Create inp3
inp3 = inp2[['Rating','Reviews','Installs','Category_AUTO_AND_VEHICLES','Category_BEAUTY',
             'Category_BOOKS_AND_REFERENCE','Category_BUSINESS','Category_COMICS',
             'Category_COMMUNICATION','Category_DATING','Category_EDUCATION','Category_EDUCATION',
             'Category_ENTERTAINMENT','Category_EVENTS','Category_FAMILY','Category_FINANCE',
             'Category_FOOD_AND_DRINK','Category_GAME','Category_HEALTH_AND_FITNESS',
             'Category_HOUSE_AND_HOME','Category_LIBRARIES_AND_DEMO','Category_LIFESTYLE',
             'Category_MAPS_AND_NAVIGATION','Category_MEDICAL','Category_NEWS_AND_MAGAZINES',
             'Category_PARENTING','Category_PERSONALIZATION','Category_PHOTOGRAPHY',
             'Category_PRODUCTIVITY',
             'Category_SHOPPING','Category_SOCIAL','Category_SPORTS','Category_TOOLS',
             'Category_TRAVEL_AND_LOCAL',
             'Category_VIDEO_PLAYERS','Category_WEATHER','Type_Paid','Content_Rating_Everyone',
             'Content_Rating_Everyone_10+','Content_Rating_Mature_17+','Content_Rating_Teen',
             'Content_Rating_Unrated',
             'Genres_Art_&_Design','Genres_Art_&_Design_Creativity','Genres_Art_&_Design_Pretend_Play',
             'Genres_Auto_&_Vehicles','Genres_Beauty','Genres_Board_Brain_Games','Genres_Books_&_Reference',
             'Genres_Business','Genres_Casual','Genres_Casual_Action_&_Adventure','Genres_Casual_Brain_Games',
             'Genres_Casual_Pretend_Play','Genres_Comics','Genres_Comics_Creativity','Genres_Communication',
             'Genres_Dating','Genres_Education','Genres_Education_Action_&_Adventure',
             'Genres_Education_Creativity',
             'Genres_Education_Education','Genres_Education_Pretend_Play','Genres_Educational_Education',
             'Genres_Educational_Pretend_Play','Genres_Entertainment','Genres_Entertainment_Brain_Games',
             'Genres_Entertainment_Creativity','Genres_Entertainment_Music_&_Video','Genres_Events',
             'Genres_Finance','Genres_Food_&_Drink','Genres_Health_&_Fitness','Genres_House_&_Home',
             'Genres_Libraries_&_Demo','Genres_Lifestyle','Genres_Maps_&_Navigation','Genres_Medical',
             'Genres_Music_Music_&_Video','Genres_News_&_Magazines','Genres_Parenting',
             'Genres_Parenting_Education',
             'Genres_Parenting_Music_&_Video','Genres_Personalization','Genres_Photography',
             'Genres_Productivity',
             'Genres_Puzzle','Genres_Puzzle_Brain_Games','Genres_Racing_Action_&_Adventure',
             'Genres_Role_Playing',
             'Genres_Shopping','Genres_Simulation','Genres_Simulation_Action_&_Adventure',
             'Genres_Social',
             'Genres_Strategy','Genres_Tools','Genres_Tools_Education','Genres_Travel_&_Local',
             'Genres_Travel_&_Local_Action_&_Adventure','Genres_Weather']]
```

```
In [72]:
#Checking datatypes
inp3.dtypes
```

```
Out[72]:
Rating                                       float64
Reviews                                      float64
Installs                                     float64
Category_AUTO_AND_VEHICLES                     uint8
Category_BEAUTY                                uint8
                                             ...
Genres_Tools                                   uint8
Genres_Tools_Education                         uint8
Genres_Travel_&_Local                          uint8
Genres_Travel_&_Local_Action_&_Adventure       uint8
Genres_Weather                                 uint8
Length: 100, dtype: object
```

```
In [73]:
inp3.shape
```

```
Out[73]:
(8879, 100)
```

```
In [74]:
#split imp3
df_train, df_test = train_test_split(inp3, train_size = 0.7, random_state = 100)
```

```
In [75]:
df_train.shape, df_test.shape
```

```
Out[75]:
((6215, 100), (2664, 100))
```

4. Separate the new dataframes into X_train, y_train, X_test, y_test

```
In [76]:
#Separating the new dataframes
y_train1 = df_train.pop("Rating")
X_train1 = df_train
```

```
In [77]:
X_train1.head(1)
```

Out[77]:

	Reviews	Installs	Category_AUTO_AND_VEHICLES	Category_BEAUTY	Category_BOOKS_AND_REFERENCE	Category_BUSINESS	Category_
1279	7.63627	11.512935	0	0	0	0	

1 rows × 99 columns

```
In [78]:
y_test1 = df_test.pop("Rating")
X_test1 = df_test
```

```
In [79]:
X_test1.head(1)
```

Out[79]:

	Reviews	Installs	Category_AUTO_AND_VEHICLES	Category_BEAUTY	Category_BOOKS_AND_REFERENCE	Category_BUSINESS	Category_
1161	9.329456	13.815512	0	0	0	0	

1 rows × 99 columns

5. Create the second model

```
In [80]:
#Apply linear regression
model2 = sm.OLS(y_train1, X_train1)
```

```
In [81]:
model2 = model2.fit()
```

```
In [82]:
#Find the summary
model2.summary()
```

```
Out[82]:
                         OLS Regression Results
   Dep. Variable:          Rating      R-squared (uncentered):     0.987
           Model:             OLS  Adj. R-squared (uncentered):     0.987
          Method:   Least Squares                 F-statistic:     6412.
            Date: Fri, 19 Aug 2022           Prob (F-statistic):      0.00
            Time:        22:25:08              Log-Likelihood:   -4302.6
No. Observations:            6215                         AIC:     8749.
    Df Residuals:            6143                         BIC:     9234.
        Df Model:              72
 Covariance Type:       nonrobust

                                             coef  std err        t  P>|t|  [0.025  0.975]
                                 Reviews   0.1738    0.006   28.120  0.000   0.162   0.186
                                Installs  -0.1479    0.006  -24.049  0.000  -0.160  -0.136
                Category_AUTO_AND_VEHICLES   1.5996    0.124   12.891  0.000   1.356   1.843
                         Category_BEAUTY   1.6841    0.127   13.232  0.000   1.435   1.934
             Category_BOOKS_AND_REFERENCE   1.6477    0.121   13.593  0.000   1.410   1.885
                       Category_BUSINESS   1.5672    0.120   13.029  0.000   1.331   1.803
                         Category_COMICS   2.3320    0.226   10.340  0.000   1.890   2.774
                  Category_COMMUNICATION   1.5106    0.121   12.522  0.000   1.274   1.747
                         Category_DATING   1.4687    0.122   12.045  0.000   1.230   1.708
                      Category_EDUCATION   1.4881    0.124   12.048  0.000   1.246   1.730
                      Category_EDUCATION   1.4881    0.124   12.048  0.000   1.246   1.730
                  Category_ENTERTAINMENT   2.8894    0.247   11.696  0.000   2.405   3.374
                         Category_EVENTS   1.7089    0.126   13.571  0.000   1.462   1.956
                         Category_FAMILY   3.0092    0.239   12.567  0.000   2.540   3.479
                        Category_FINANCE   1.5305    0.120   12.728  0.000   1.295   1.766
                  Category_FOOD_AND_DRINK   1.5606    0.122   12.755  0.000   1.321   1.800
                           Category_GAME   3.0847    0.239   12.904  0.000   2.616   3.553
              Category_HEALTH_AND_FITNESS   1.5906    0.120   13.211  0.000   1.355   1.827
                 Category_HOUSE_AND_HOME   1.5765    0.124   12.756  0.000   1.334   1.819
             Category_LIBRARIES_AND_DEMO   1.6164    0.124   13.009  0.000   1.373   1.860
                      Category_LIFESTYLE   1.5542    0.120   12.914  0.000   1.318   1.790
            Category_MAPS_AND_NAVIGATION   1.4830    0.122   12.149  0.000   1.244   1.722
                        Category_MEDICAL   1.5982    0.120   13.307  0.000   1.363   1.834
             Category_NEWS_AND_MAGAZINES   1.5284    0.121   12.664  0.000   1.292   1.765
                      Category_PARENTING   2.4735    0.212   11.676  0.000   2.058   2.889
                Category_PERSONALIZATION   1.6408    0.120   13.633  0.000   1.405   1.877
                    Category_PHOTOGRAPHY   1.5468    0.121   12.830  0.000   1.310   1.783
                   Category_PRODUCTIVITY   1.5833    0.120   13.154  0.000   1.347   1.819
                       Category_SHOPPING   1.5714    0.121   13.002  0.000   1.334   1.808
                         Category_SOCIAL   1.5772    0.121   13.073  0.000   1.341   1.814
                         Category_SPORTS   3.0836    0.240   12.873  0.000   2.614   3.553
                          Category_TOOLS   2.0963    0.227    9.224  0.000   1.651   2.542
               Category_TRAVEL_AND_LOCAL   2.0848    0.228    9.163  0.000   1.639   2.531
                  Category_VIDEO_PLAYERS   3.0405    0.243   12.503  0.000   2.564   3.517
                        Category_WEATHER   1.5817    0.124   12.759  0.000   1.339   1.825
                               Type_Paid  -0.0792    0.027   -2.924  0.003  -0.132  -0.026
                 Content_Rating_Everyone   1.4445    0.238    6.082  0.000   0.979   1.910
              Content_Rating_Everyone_10+   1.4346    0.239    5.990  0.000   0.965   1.904
               Content_Rating_Mature_17+   1.4171    0.240    5.907  0.000   0.947   1.887
                     Content_Rating_Teen   1.4392    0.238    6.049  0.000   0.973   1.906
                  Content_Rating_Unrated   1.4046    0.542    2.593  0.010   0.343   2.466
                     Genres_Art_&_Design   3.3910    0.252   13.478  0.000   2.898   3.884
          Genres_Art_&_Design_Creativity   2.4854    0.255    9.740  0.000   1.985   2.986
        Genres_Art_&_Design_Pretend_Play   3.2010    0.542    5.904  0.000   2.138   4.264
                 Genres_Auto_&_Vehicles   1.5996    0.124   12.891  0.000   1.356   1.843
                           Genres_Beauty   1.6841    0.127   13.232  0.000   1.435   1.934
                Genres_Board_Brain_Games   0.2888    0.159    1.815  0.070  -0.023   0.601
                Genres_Books_&_Reference   1.6477    0.121   13.593  0.000   1.410   1.885
                         Genres_Business   1.5672    0.120   13.029  0.000   1.331   1.803
                           Genres_Casual   0.0148    0.059    0.252  0.801  -0.101   0.130
        Genres_Casual_Action_&_Adventure   0.2113    0.146    1.451  0.147  -0.074   0.497
               Genres_Casual_Brain_Games   0.4865    0.175    2.773  0.006   0.143   0.830
              Genres_Casual_Pretend_Play   0.0809    0.110    0.734  0.463  -0.135   0.297
                           Genres_Comics   0.7945    0.185    4.292  0.000   0.432   1.157
                Genres_Comics_Creativity   1.5375    0.336    4.574  0.000   0.879   2.196
                    Genres_Communication   1.5106    0.121   12.522  0.000   1.274   1.747
                           Genres_Dating   1.4687    0.122   12.045  0.000   1.230   1.708
                        Genres_Education   0.2968    0.050    5.956  0.000   0.199   0.395
     Genres_Education_Action_&_Adventure   0.6262    0.284    2.201  0.028   0.068   1.184
             Genres_Education_Creativity   0.5810    0.223    2.611  0.009   0.145   1.017
              Genres_Education_Education   0.3685    0.096    3.836  0.000   0.180   0.557
           Genres_Education_Pretend_Play   0.3641    0.117    3.124  0.002   0.136   0.593
            Genres_Educational_Education   0.2567    0.100    2.568  0.010   0.061   0.453
         Genres_Educational_Pretend_Play   0.1588    0.136    1.170  0.242  -0.107   0.425
                    Genres_Entertainment   0.0790    0.049    1.616  0.106  -0.017   0.175
        Genres_Entertainment_Brain_Games   0.2560    0.189    1.357  0.175  -0.114   0.626
         Genres_Entertainment_Creativity   0.5172    0.347    1.489  0.137  -0.164   1.198
    Genres_Entertainment_Music_&_Video   0.2087    0.141    1.479  0.139  -0.068   0.485
                           Genres_Events   1.7089    0.126   13.571  0.000   1.462   1.956
                          Genres_Finance   1.5305    0.120   12.728  0.000   1.295   1.766
                     Genres_Food_&_Drink   1.5606    0.122   12.755  0.000   1.321   1.800
                 Genres_Health_&_Fitness   1.5906    0.120   13.211  0.000   1.355   1.827
                    Genres_House_&_Home   1.5765    0.124   12.756  0.000   1.334   1.819
                 Genres_Libraries_&_Demo   1.6164    0.124   13.009  0.000   1.373   1.860
                        Genres_Lifestyle   1.5542    0.120   12.914  0.000   1.318   1.790
                 Genres_Maps_&_Navigation   1.4830    0.122   12.149  0.000   1.244   1.722
                          Genres_Medical   1.5982    0.120   13.307  0.000   1.363   1.834
                Genres_Music_Music_&_Video   0.5720    0.346    1.652  0.099  -0.107   1.251
                Genres_News_&_Magazines   1.5284    0.121   12.664  0.000   1.292   1.765
                        Genres_Parenting   0.8720    0.142    6.145  0.000   0.594   1.150
              Genres_Parenting_Education   0.6579    0.275    2.393  0.017   0.119   1.197
          Genres_Parenting_Music_&_Video   0.9436    0.236    3.995  0.000   0.481   1.407
                  Genres_Personalization   1.6408    0.120   13.633  0.000   1.405   1.877
                      Genres_Photography   1.5468    0.121   12.830  0.000   1.310   1.783
                     Genres_Productivity   1.5833    0.120   13.154  0.000   1.347   1.819
                           Genres_Puzzle   0.1939    0.061    3.203  0.001   0.075   0.313
                Genres_Puzzle_Brain_Games   0.2859    0.131    2.175  0.030   0.028   0.544
        Genres_Racing_Action_&_Adventure   0.2674    0.136    1.970  0.049   0.001   0.534
                     Genres_Role_Playing   0.0154    0.065    0.238  0.812  -0.112   0.142
                         Genres_Shopping   1.5714    0.121   13.002  0.000   1.334   1.808
                       Genres_Simulation   0.0243    0.055    0.438  0.662  -0.084   0.133
    Genres_Simulation_Action_&_Adventure   0.2534    0.166    1.526  0.127  -0.072   0.579
                           Genres_Social   1.5772    0.121   13.073  0.000   1.341   1.814
                         Genres_Strategy  -0.0036    0.076   -0.048  0.962  -0.152   0.145
                            Genres_Tools   0.9686    0.181    5.346  0.000   0.613   1.324
                  Genres_Tools_Education   1.1277    0.334    3.376  0.001   0.473   1.782
                   Genres_Travel_&_Local   0.9922    0.183    5.434  0.000   0.634   1.350
 Genres_Travel_&_Local_Action_&_Adventure   1.0926    0.334    3.270  0.001   0.438   1.748
                          Genres_Weather   1.5817    0.124   12.759  0.000   1.339   1.825

       Omnibus: 2310.962     Durbin-Watson:      2.020
Prob(Omnibus):    0.000  Jarque-Bera (JB):  13517.788
          Skew:   -1.671          Prob(JB):       0.00
      Kurtosis:    9.406         Cond. No.   1.23e+16


Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The smallest eigenvalue is 9.48e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

- Rating R-squared (uncentered): 0.987
- Adj. R-squared (uncentered): 0.987
- F-statistic: 6412
- AIC: 8749
- BIC: 9234

How Well Does the Model Fit the Data?

***model2* performs better:**

- R-squared value:

  The most common way to evaluate the overall fit of a linear model is by the **R-squared** value.

  R-squared is between 0 and 1 (or between 0 to 100%), and higher is better because it means that more variance is explained by the model.

  In case of our models, model1 and model2 have same R-squared values.

- F-statistics:

  In case of dataset that only have numerical values, it is said higher the F-statistic better the model.

  In case of our models, model2 has more F-statistic value than model1.

- AIC and BIC:

  In case of models, lesser the AIC and BIC, better the model.

  In case of models, model2 has lesser AIC and BIC values than the model1.

Prediction using "model2"

Let us perform prediction using model2 by importing **sklearn.metrics**

In [83]:

```
#Importing sklearn.metrics
import sklearn.metrics as metrics
y_test_pred1= model2.predict(X_test1)
print(np.sqrt(metrics.mean_squared_error(y_test, y_test_pred1)))


#RMSE value of model1 = 0.49203326281981113
```

```
0.49102040642334177
```

In [84]:

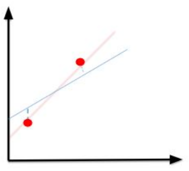
```
#Calculating MAPE
def MAPE(y_test_pred1,y_test1):
    return np.mean(np.abs((y_test1 - y_test_pred1) / y_test1)) * 100
print ('MAPE: ' + str(MAPE(y_test_pred1,y_test1)) )
```

```
MAPE: 9.467552790023404
```

model2 shows slight improvement as the RMSE and MAPE value of this model is lesser than that of **model1**.

4. Ridge Regression:

- Ridge Regression (L2) is used when there is a problem of multicollinearity.
- By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.



The main idea is to find a new line that has some bias with respect to the training data

In return for that small amount of bias, a significant drop in variance is achieved

Minimization objective = LS Obj + λ * (sum of the square of coefficients)

LS Obj refers to least squares objective

λ controls the strength of the penalty term

5. Lasso Regression:

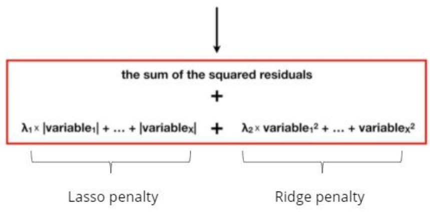
- Lasso Regression (L1) is similar to ridge, but it also performs feature selection.
- It will set the coefficient value for features that do not help in decision making very low, potentially zero.

Minimization objective = LS Obj + λ * (sum of absolute coefficient values)

- Lasso regression tends to exclude variables that are not required from the equation, whereas ridge tends to do better when all variables are present

6. ElasticNet Regression:

ElasticNet regression combines
the strength of **lasso and ridge regression**



If you are not sure whether to use lasso or ridge, use ElasticNet

Use Case: Ridge, Lasso, ElasticNet Regression for Training and Prediction:

We are going to use the same dataset that we used in the previous use case at the time of training of "model2" i.e. inp3

1. Ridge Regression:

In [85]:

```
#Importing Ridge
import sklearn
from sklearn.linear_model import Ridge
ridgeReg = Ridge(alpha=0.001, normalize=True)
ridgeReg.fit(X_train1,y_train1)
```

Out[85]:

```
▾            Ridge
Ridge(alpha=0.001, normalize=True)
```

Evaluating using RMSE:

In [86]:

```
print(np.sqrt(sklearn.metrics.mean_squared_error(y_train1, ridgeReg.predict(X_train1))))
print(np.sqrt(sklearn.metrics.mean_squared_error(y_test1, ridgeReg.predict(X_test1))))
print('R2 Value/Coefficient of Determination: {}'.format(ridgeReg.score(X_test1, y_test1)))
```

```
0.48012289929471325
0.4870128321509223
R2 Value/Coefficient of Determination: 0.13640031615542214
```

2. Lasso Regression:

In [87]:

```
#Importing Lasso
from sklearn.linear_model import Lasso
lassoreg = Lasso(alpha=0.001, normalize=True)
lassoreg.fit(X_train1,y_train1)
```

Out[87]:

```
▾            Lasso
Lasso(alpha=0.001, normalize=True)
```

Evaluating using RMSE:

In [88]:

```
print(np.sqrt(sklearn.metrics.mean_squared_error(y_train1, lassoreg.predict(X_train1))))
print(np.sqrt(sklearn.metrics.mean_squared_error(y_test1, lassoreg.predict(X_test1))))
print('R2 Value/Coefficient of Determination: {}'.format(lassoreg.score(X_test1, y_test1)))
```

```
0.520679541079235
0.5212800927272304
R2 Value/Coefficient of Determination: 0.01059535365885067
```

3. ElasticNet Regression:

In [89]:

```
#Importing ElasticNet
from sklearn.linear_model import ElasticNet
Elastic = ElasticNet(alpha=0.001, normalize=True)
Elastic.fit(X_train1,y_train1)
```

Out[89]:

```
▾            ElasticNet
ElasticNet(alpha=0.001, normalize=True)
```

Evaluating using RMSE:

In [90]:

```
print(np.sqrt(sklearn.metrics.mean_squared_error(y_train1, Elastic.predict(X_train1))))
print(np.sqrt(sklearn.metrics.mean_squared_error(y_test1, Elastic.predict(X_test1))))
print('R2 Value/Coefficient of Determination: {}'.format(Elastic.score(X_test1, y_test1)))
```

```
0.5214331925073523
0.5220643755303499
R2 Value/Coefficient of Determination: 0.007615931276933607
```

Exercise:

- Perform the iteration of the model with Lasso, Ridge, and ElasticNet Regression by using the original dataset i.e., **inp0** as done in the case of Linear Regression.
- Use the following metrics to evaluate the model:
  - RMSE
  - MAPE
  - R^2 error

Cost Function:

- A cost function is a function that evaluates a model's performance for a given dataset.
- It evaluates and expresses the error between predicted values and expected values as a single, real number.

Gradient:

- A gradient is a measurement of how much a function's output varies as its inputs are changed.

Gradient Descent:

- Gradient descent is an optimization algorithm that is used to find the values of the parameters (coefficients) of a function that minimizes the cost function by iteratively moving in the direction of steepest descent as determined by the gradient's negative.
- It's an optimization algorithm to discover the local minimum of a differentiable function or feature.
- We use gradient descent to update the parameters of our model. In linear regression, parameters correspond to coefficients, and in neural networks, parameters correspond to weights.
- The gradient descent equation is as follows:

Cost Function

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}[\underset{\substack{\uparrow \\ \text{Predicted Value}}}{h_\theta(x_i)} - \underset{\substack{\uparrow \\ \text{True Value}}}{y_i}]^2$$

Gradient Descent

$$\theta_j = \theta_j - \underset{\overset{\uparrow}{\text{Learning Rate}}}{\alpha} \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Now,

$$\frac{\partial}{\partial \theta} J_\theta = \frac{\partial}{\partial \theta} \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x_i) - y_i]^2$$

$$\frac{\partial}{\partial \theta} J_\theta = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i). \ \frac{\partial}{\partial \theta_j} (\theta x_i - y_i)$$

$$\frac{\partial}{\partial \theta} J_\theta = \frac{1}{m} \sum_{i=1}^{m} [(h_\theta(x_i) - y)\, x_i]$$

Therefore,

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} [(h_\theta(x_i) - y_i) x_i]$$

Types of Gradient Descents:

Gradient descents are divided into three categories, which vary primarily in the amount of data they use. These categories are:

1. Batch Gradient Descent (BGD)
2. Stochastic Gradient Descent (SGD)
3. Mini-Batch Gradient Descent (Mini BGD)

1. Batch Gradient Descent (BGD):

- Batch gradient descent (BGD), also known as vanilla gradient descent, measures the error for each example in the training dataset, but the model is updated only after all of the training examples have been evaluated.
- This whole process is referred to as a training epoch because it resembles a loop.
- For each gradient descent iteration, it processes all the training samples. However, batch gradient descent is computationally very expensive when the number of training examples is high.
- Thus, if the number of training examples is high, we tend to use stochastic gradient descent (SGD) or mini-batch gradient descent (Mini BGD) instead.

2. Stochastic Gradient Descent (SGD):

- Stochastic gradient descent (SGD) is a form of gradient descent that processes one training example per iteration.
- It estimates the error for each example in the training dataset and updates the parameters one by one.
- Based on the problem statement, batch gradient descent (BGD) can be much faster because the parameters are modified even after an iteration in which only a single example has been processed.
- One advantage is that the frequent updates allow us to track our progress in great detail.
- However, even if the number of training examples is high, it can only process one of them, which will add to the system's overhead and the number of iterations needed.

3. Mini-Batch Gradient Descent (Mini BGD):

- Mini-batch gradient descent (Mini BGD) combines the principles of stochastic gradient descent (SGD) and batch gradient descent (BGD) and is faster than both.
- It divides the training dataset into small batches and updates each of those batches. This establishes a balance between the robustness of stochastic gradient descent (SGD) and the efficiency of batch gradient descent (BGD).
- It is compatible with both larger and smaller training examples.

Use Case: Stochastic Gradient Descent (SGD):

Importing Required Libraries

In [91]:

```
!pip install tabulate
```

```
Requirement already satisfied: tabulate in c:\users\alpika.gupta\anaconda3\lib\site-packages (0.8.10)
```

In [92]:

```
#Importing libraries
import numpy as np
import pandas as pd

from tabulate import tabulate

from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score

import warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
%matplotlib inline
```

Load the Data

In [93]:

```
#Load the data dictionary
boston = load_boston()

#Find the dict keys
print(boston.keys())
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_module'])
```

Dataset Description

In [94]:

```
#To print the boston dataset description
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the targe
t.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity',
Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Internat
ional Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

Find Features and Target

Let us split the data into **test** and **train.**

In [95]:

```
X = boston.data
Y = boston.target

#Splitting the data to test and train
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
```

Find Feature's Name

In [96]:

```
columns = boston.feature_names
columns
```

Out[96]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [97]:

```
boston_df = pd.DataFrame(boston.data)
boston_df.columns = columns
boston_df["MEDV"] = Y
```

In [98]:

```
boston_df.head()
```

Out[98]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [99]:

```
print(boston_df.describe())
```

```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000   
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634   
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617   
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000   
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500   
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500   
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500   
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000   

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000   
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032   
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864   
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000   
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500   
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000   
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000   
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000   

            LSTAT        MEDV  
count  506.000000  506.000000  
mean    12.653063   22.532806  
std      7.141062    9.197104  
min      1.730000    5.000000  
25%      6.950000   17.025000  
50%     11.360000   21.200000  
75%     16.955000   25.000000  
max     37.970000   50.000000  
```

Standardizing Data

Let us standardize the **test** and **train** data using the **StandardScaler()** function.

In [100…

```
scaler = StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

In [101…

```
train_data=pd.DataFrame(x_train)
train_data['price']=y_train
train_data.head(3)
```

Out[101…

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.136091	-0.497256	1.230966	-0.238152	0.475052	0.019297	0.975502	-0.804562	-0.494534	-0.010185	-1.660267	-0.711900	-0.191552
1	2.123654	-0.497256	1.018537	-0.238152	1.659763	-0.665059	0.845579	-0.911559	1.752922	1.588807	0.834362	0.381788	1.344974
2	-0.397641	1.990489	-1.158867	-0.238152	-1.315178	0.408602	-1.738831	1.080745	-0.967683	-0.849199	-1.252055	0.210827	-1.129378

In [102…

```
x_test = np.array(x_test)
y_test = np.array(y_test)
```

In [103…

```
#Shape of test and train data metrics
print(x_train.shape)
print(y_train.shape)

print(x_test.shape)
print(y_test.shape)
```

```
(354, 13)
(354,)
(152, 13)
(152,)
```

Linear Regression: Boston Housing Prediction

Let us calculate **R-Squared**, **Linear Regressor Model Accuracy**, **MAE**, **MSE**, and **RMSE**.

In [104…

```
print("Linear Regression: Boston Housing Prediction")
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)
lin_score = lin_reg.score(x_train, y_train)
print("R-squared:", lin_score)

lin_y_pred = lin_reg.predict(x_test)
lin_accuracy = explained_variance_score(y_test, lin_y_pred)
lin_accuracy = round(lin_accuracy*100, 6)
print("Linear Regressor Model Accuracy:", lin_accuracy, "%")
print()

lin_mae = mean_absolute_error(y_test, lin_y_pred)
lin_mse = mean_squared_error(y_test, lin_y_pred)
lin_rmse = lin_mse*(1/2.0)

print("MAE:", lin_mae)
print("MSE:", lin_mae)
print("RMSE:", lin_rmse)
print()
```

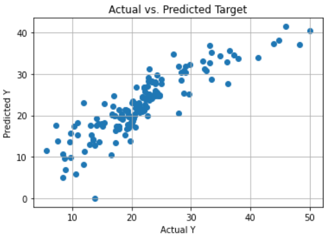
```
Linear Regression: Boston Housing Prediction
R-squared: 0.7257142937737184
Linear Regressor Model Accuracy: 77.561924 %

MAE: 3.00356528810303
MSE: 3.00356528810303
RMSE: 7.916910937918455
```

Let us plot a graph of the **Actual vs. Predicted Target**.

In [105…

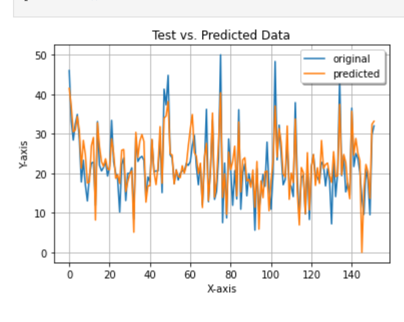
```
plt.scatter(y_test, lin_y_pred)
plt.grid()
plt.xlabel('Actual Y')
plt.ylabel('Predicted Y')
plt.title('Actual vs. Predicted Target')
plt.show()
```



Let us plot a graph of the **Test vs. Predicted Data**.

In [106…

```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, lin_y_pred, label="predicted")
plt.title("Test vs. Predicted Data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



SGD Regression: Boston Housing Prediction

Let us calculate **R-squared**, **SGD Regressor Model Accuracy**, **MAE**, **MSE**, and **RMSE**.

In [107…

```
print("SGD Regression: Boston Housing Prediction")
sgd_reg = SGDRegressor()
sgd_reg.fit(x_train, y_train)
sgd_score = sgd_reg.score(x_train, y_train)
print("R-squared:", sgd_score)

sgd_y_pred = sgd_reg.predict(x_test)
sgd_accuracy = explained_variance_score(y_test, sgd_y_pred)
sgd_accuracy = round(sgd_accuracy*100, 6)
print("SGD Regressor Model Accuracy:", sgd_accuracy, "%")
print()

sgd_mae = mean_absolute_error(y_test, sgd_y_pred)
sgd_mse = mean_squared_error(y_test, sgd_y_pred)
sgd_rmse = sgd_mse*(1/2.0)

print("MAE:", sgd_mae)
print("MSE:", sgd_mse)
print("RMSE:", sgd_rmse)
print()
```

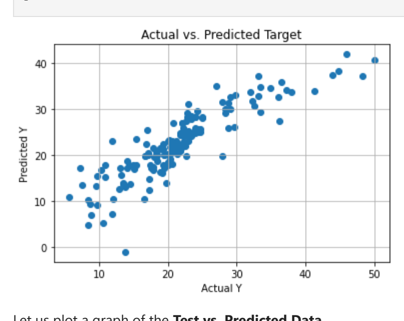
```
SGD Regression: Boston Housing Prediction
R-squared: 0.7227884809216978
SGD Regressor Model Accuracy: 77.132392 %

MAE: 3.026066392810071
MSE: 16.00535293654782
RMSE: 8.00267646827391
```

Let us plot a graph of the **Actual vs. Predicted Target**.

In [108…

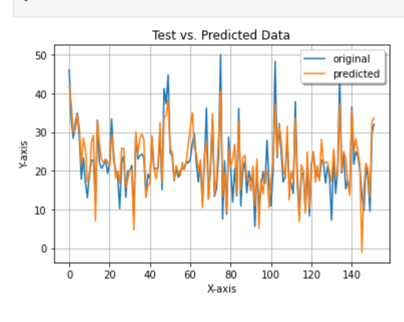
```
plt.scatter(y_test, sgd_y_pred)
plt.grid()
plt.xlabel('Actual Y')
plt.ylabel('Predicted Y')
plt.title('Actual vs. Predicted Target')
plt.show()
```



Let us plot a graph of the **Test vs. Predicted Data**.

In [109…

```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, sgd_y_pred, label="predicted")
plt.title("Test vs. Predicted Data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



Model Comparison

1. Evaluation Matrix Comparison

In [110…

```
#Performing evaluation matrix comparison
model_table = pd.DataFrame(columns = ["Comparison Matrix", "LR Model", "SGD Model"])
model_table["Comparison Matrix"] = ["Accuracy (Variance Score)", "R2 Score", "MAE", "MSE", "RMSE"]
model_table["LR Model"] = [lin_accuracy, lin_score, lin_mae, lin_mse, lin_rmse]
model_table["SGD Model"] = [sgd_accuracy, sgd_score, sgd_mae, sgd_mse, sgd_rmse]

print(tabulate(model_table, headers = 'keys', tablefmt = 'psql', numalign="left"))
```

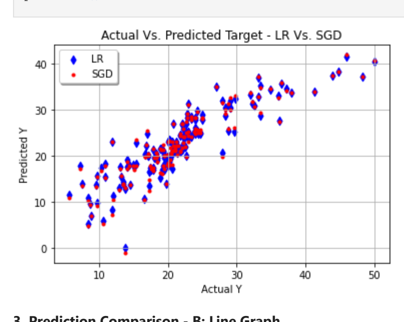
```
+----+---------------------------+------------+-------------+
|    | Comparison Matrix         | LR Model   | SGD Model   |
|----+---------------------------+------------+-------------|
| 0  | Accuracy (Variance Score) | 77.5619    | 77.1324     |
| 1  | R2 Score                  | 0.725714   | 0.722788    |
| 2  | MAE                       | 3.00357    | 3.02607     |
| 3  | MSE                       | 15.8338    | 16.0054     |
| 4  | RMSE                      | 7.91691    | 8.00268     |
+----+---------------------------+------------+-------------+
```

2. Prediction Comparison - A: Scatter Plot

Let us perform a comparison of the **Actual vs. Predicted Target** for the **Scatter Plot**.

In [111…

```
plt.scatter(y_test, lin_y_pred, c="b", marker="d", label='LR')
plt.scatter(y_test, sgd_y_pred, c="r", marker=".", label='SGD')
plt.xlabel('Actual Y')
plt.ylabel('Predicted Y')
plt.title('Actual Vs. Predicted Target - LR Vs. SGD')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid()
plt.show()
```

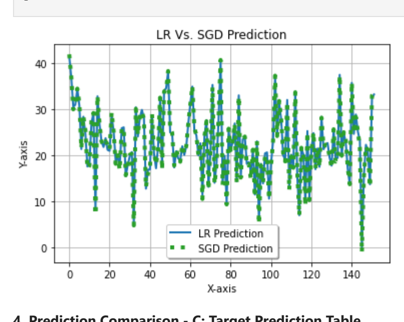


3. Prediction Comparison - B: Line Graph

Let us perform a comparison of the **LR vs. SGD Prediction** for the **Line Graph**.

In [112…

```
x_ax = range(len(y_test))
plt.plot(x_ax, lin_y_pred, label="LR Prediction")
plt.plot(x_ax, sgd_y_pred, label="SGD Prediction")
plt.title("LR Vs. SGD Prediction")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



4. Prediction Comparison - C: Target Prediction Table

Let us compare the **LR Predicted Value** and **SGD Predicted Value**.

In [113…

```
prediction_table = pd.DataFrame(columns=["LR Predicted Value", "SGD Predicted Value"])
prediction_table["LR Predicted Value"] = lin_y_pred
prediction_table["SGD Predicted Value"] = sgd_y_pred

print(tabulate(prediction_table, headers = 'keys', tablefmt = 'psql', numalign="left"))
```

```
+-----+----------------------+-----------------------+
|     | LR Predicted Value   | SGD Predicted Value   |
|-----+----------------------+-----------------------|
| 0   | 41.4908              | 41.8855               |
| 1   | 36.8945              | 37.1855               |
| 2   | 30.5095              | 29.918                |
| 3   | 30.9245              | 30.6915               |
| 4   | 34.409               | 34.4465               |
| 5   | 30.7754              | 31.3239               |
| 6   | 21.3008              | 21.4446               |
| 7   | 28.3381              | 28.3853               |
| 8   | 24.6874              | 25.4813               |
| 9   | 17.5071              | 17.1117               |
| 10  | 17.8427              | 17.7045               |
| 11  | 27.0118              | 27.3565               |
| 12  | 29.0729              | 29.1563               |
| 13  | 8.16764              | 7.0879                |
| 14  | 32.7771              | 32.8589               |
| 15  | 26.6931              | 26.8348               |
| 16  | 22.9233              | 22.9401               |
| 17  | 21.7455              | 22.097                |
| 18  | 23.6717              | 22.9954               |
| 19  | 21.1214              | 22.3216               |
| 20  | 20.9091              | 21.1535               |
| 21  | 28.743               | 29.2963               |
| 22  | 25.1269              | 25.1679               |
| 23  | 18.7371              | 18.0881               |
| 24  | 19.6255              | 19.9274               |
| 25  | 17.4443              | 16.7471               |
| 26  | 25.8447              | 25.8276               |
| 27  | 26.0272              | 25.7861               |
| 28  | 15.3838              | 15.6016               |
| 29  | 17.8436              | 17.6804               |
| 30  | 19.7745              | 20.0748               |
| 31  | 20.6521              | 20.2664               |
| 32  | 5.09075              | 4.70043               |
| 33  | 30.375               | 30.0477               |
| 34  | 24.0346              | 24.3234               |
| 35  | 28.186               | 28.3361               |
| 36  | 29.8191              | 29.4366               |
| 37  | 28.0558              | 27.885                |
| 38  | 12.6971              | 13.0116               |
| 39  | 16.8389              | 16.0123               |
| 40  | 16.853               | 16.7836               |
| 41  | 28.6068              | 29.0329               |
| 42  | 20.4021              | 20.6993               |
| 43  | 17.1312              | 17.9518               |
| 44  | 21.7044              | 21.0931               |
| 45  | 31.8228              | 32.4798               |
| 46  | 17.4694              | 16.845                |
| 47  | 33.9607              | 33.7991               |
| 48  | 34.5137              | 34.1487               |
| 49  | 38.184               | 38.2644               |
| 50  | 24.9386              | 25.6723               |
| 51  | 24.6544              | 24.8518               |
| 52  | 17.3062              | 17.7015               |
| 53  | 20.8643              | 21.347                |
| 54  | 19.2547              | 19.5196               |
| 55  | 18.9001              | 18.5391               |
| 56  | 21.9192              | 22.1642               |
| 57  | 20.0179              | 20.3514               |
| 58  | 21.9179              | 21.9919               |
| 59  | 26.8846              | 26.9629               |
| 60  | 31.1888              | 31.0366               |
| 61  | 34.8877              | 34.9995               |
| 62  | 25.2396              | 25.944                |
| 63  | 24.2967              | 24.2731               |
| 64  | 19.9444              | 19.9699               |
| 65  | 22.565               | 22.8151               |
| 66  | 11.328               | 10.4519               |
| 67  | 24.1086              | 24.0016               |
| 68  | 27.5851              | 27.3364               |
| 69  | 13.0836              | 12.5585               |
| 70  | 20.9862              | 21.3308               |
| 71  | 35.2531              | 34.764                |
| 72  | 13.9682              | 13.7433               |
| 73  | 18.1656              | 17.8449               |
| 74  | 27.7294              | 28.2534               |
| 75  | 40.3975              | 40.6656               |
| 76  | 13.8945              | 13.5083               |
| 77  | 19.862               | 19.6349               |
| 78  | 9.59307              | 9.25591               |
| 79  | 25.3634              | 25.7403               |
| 80  | 20.9301              | 21.0666               |
| 81  | 22.9832              | 22.9951               |
| 82  | 26.8731              | 26.8218               |
| 83  | 14.2712              | 13.9445               |
| 84  | 32.9805              | 32.671                |
| 85  | 15.2945              | 15.0935               |
| 86  | 23.4149              | 23.2745               |
| 87  | 23.9465              | 24.0345               |
| 88  | 18.0822              | 17.7519               |
| 89  | 18.469               | 18.0179               |
| 90  | 16.4444              | 14.7628               |
| 91  | 20.8568              | 21.184                |
| 92  | 11.419               | 10.9252               |
| 93  | 22.9933              | 23.015                |
| 94  | 5.85954              | 5.06437               |
| 95  | 18.0939              | 17.7409               |
| 96  | 13.8261              | 13.9664               |
| 97  | 20.3623              | 19.7046               |
| 98  | 20.5843              | 19.7939               |
| 99  | 10.5214              | 10.446                |
| 100 | 18.2128              | 17.706                |
| 101 | 22.9752              | 23.0065               |
| 102 | 37.0495              | 37.2562               |
| 103 | 24.1007              | 24.1251               |
| 104 | 31.3495              | 31.7625               |
| 105 | 27.7881              | 27.9345               |
| 106 | 19.9461              | 20.3962               |
| 107 | 19.0228              | 18.6115               |
| 108 | 31.9851              | 31.517                |
| 109 | 13.4063              | 12.4019               |
| 110 | 20.0744              | 19.6857               |
| 111 | 17.6475              | 17.2188               |
| 112 | 33.7576              | 33.6521               |
| 113 | 15.2282              | 16.2563               |
| 114 | 6.91515              | 6.88759               |
| 115 | 21.4677              | 21.5887               |
| 116 | 20.1582              | 19.8569               |
| 117 | 9.75699              | 8.98532               |
| 118 | 25.2296              | 24.9033               |
| 119 | 10.7292              | 10.1906               |
| 120 | 21.4701              | 20.8294               |
| 121 | 24.7724              | 24.9474               |
| 122 | 16.922               | 17.0267               |
| 123 | 21.4194              | 21.358                |
| 124 | 17.4319              | 17.4052               |
| 125 | 28.312               | 28.1284               |
| 126 | 21.2473              | 21.7699               |
| 127 | 22.2357              | 22.257                |
| 128 | 22.6046              | 22.1832               |
| 129 | 19.4701              | 19.4823               |
| 130 | 17.6914              | 17.1919               |
| 131 | 25.4865              | 25.6177               |
| 132 | 19.1826              | 18.5633               |
| 133 | 19.3867              | 19.0175               |
| 134 | 37.4434              | 37.2716               |
| 135 | 19.633               | 19.8768               |
| 136 | 24.7732              | 24.9752               |
| 137 | 22.7821              | 23.3403               |
| 138 | 16.9901              | 17.196                |
| 139 | 13.5727              | 13.6448               |
| 140 | 35.6925              | 35.7711               |
| 141 | 24.5964              | 24.8125               |
| 142 | 28.8186              | 28.2873               |
| 143 | 25.1047              | 25.2257               |
| 144 | 23.1659              | 21.6165               |
| 145 | -0.0163374           | -1.13048              |
| 146 | 15.6806              | 15.4564               |
| 147 | 22.2388              | 21.888                |
| 148 | 20.4303              | 20.6844               |
| 149 | 13.6457              | 13.1583               |
| 150 | 32.3988              | 32.9267               |
| 151 | 33.174               | 33.6081               |
+-----+----------------------+-----------------------+
```

Note: In this topic, we saw the use of the linear regression methods, but in the next topic we will be working on "Logistic Regression".