```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

```
df=pd.read_csv('/content/Video_Games.csv')
df
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Criti
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	3.77	8.45	82.53	
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24	
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	3.79	3.29	35.52	
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	3.28	2.95	32.77	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37	
...	...	...	...	...	...	...	...	...	...	...	
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00	0.01	0.00	0.01	
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01	0.00	0.00	0.01	

```
df.shape
```

```
(16719, 16)
```

```
df.dtypes
```

```
Name                object
Platform            object
Year_of_Release    float64
Genre               object
Publisher           object
NA_Sales           float64
EU_Sales           float64
JP_Sales           float64
Other_Sales        float64
Global_Sales       float64
Critic_Score       float64
Critic_Count       float64
User_Score          object
User_Count         float64
Developer           object
Rating              object
dtype: object
```

```
df['Developer']=df['Developer'].str.lower()
df
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89
...	...	...	...	...	...	...	...
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01

```
df['EU_Sales']=df['EU_Sales'].round(1)
df
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	29.0
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.6
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.8
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.9
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.9
...	...	...	...	...	...	...	...
16714	Samurai Warriors:	PS3	2016.0	Action	Tecmo Koei	0.00	0.0

The ceil function returns the smallest integer value which is greater than or equal to the specified number, whereas the floor function returns the largest integer value which is less than or equal to the specified number.

```
df['Other_Sales']=df['Other_Sales'].apply(np.ceil)
df
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	29.0
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	4.0
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	13.0
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	11.0
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	9.0
...	...	...	...	...	...	...	...
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.0
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.0

```
df['Other_Sales'].unique()
```

```
array([8.450e+00, 7.700e-01, 3.290e+00, 2.950e+00, 1.000e+00, 5.800e-01,
       2.880e+00, 2.840e+00, 2.240e+00, 4.700e-01, 2.740e+00, 1.900e+00,
       7.100e-01, 2.150e+00, 1.690e+00, 1.770e+00, 3.960e+00, 1.057e+01,
       5.500e-01, 2.040e+00, 1.360e+00, 4.200e-01, 4.600e-01, 1.410e+00,
       1.780e+00, 5.000e-01, 1.180e+00, 8.000e-01, 1.160e+00, 1.320e+00,
       5.900e-01, 2.380e+00, 1.130e+00, 7.800e-01, 2.420e+00, 1.120e+00,
       1.280e+00, 1.570e+00, 1.300e+00, 1.010e+00, 9.100e-01, 1.790e+00,
       1.970e+00, 8.600e-01, 1.210e+00, 2.300e-01, 7.600e-01, 7.400e-01,
       7.530e+00, 2.900e-01, 1.030e+00, 5.200e-01, 2.110e+00, 1.600e+00,
       1.610e+00, 3.500e-01, 9.700e-01, 1.060e+00, 6.300e-01, 1.500e-01,
       7.900e-01, 9.600e-01, 1.250e+00, 9.000e-01, 8.100e-01, 3.900e-01,
       6.800e-01, 8.500e-01, 1.800e-01, 8.000e-02, 6.700e-01, 7.000e-01,
       4.100e-01, 3.300e-01, 6.000e-01, 5.400e-01, 1.730e+00, 1.230e+00,
       1.600e-01, 1.110e+00, 3.100e-01, 4.800e-01, 6.200e-01, 1.900e-01,
       6.900e-01, 1.020e+00, 7.300e-01, 1.080e+00, 4.500e-01, 2.800e-01,
       5.100e-01, 2.200e-01, 1.090e+00, 9.900e-01, 3.000e-01, 6.400e-01,
       6.600e-01, 9.800e-01, 1.390e+00, 1.400e-01, 1.370e+00, 7.000e-02,
       2.100e-01, 6.100e-01, 1.700e-01, 1.200e-01, 0.000e+00, 7.200e-01,
       2.400e-01, 8.200e-01, 1.740e+00, 8.700e-01, 9.200e-01, 5.700e-01,
       1.100e-01, 4.000e-02, 5.600e-01, 2.000e-01, 3.400e-01, 9.000e-02,
       8.300e-01, 4.400e-01, 6.000e-02, 3.200e-01, 3.800e-01, 1.480e+00,
       3.700e-01, 1.000e-01, 2.500e-01, 3.600e-01, 1.300e-01, 4.300e-01,
       5.000e-02, 2.000e-02, 2.600e-01, 4.000e-01, 7.500e-01, 1.930e+00,
       8.400e-01, 5.300e-01, 8.900e-01, 1.670e+00, 2.700e-01, 2.930e+00,
       4.900e-01, 1.000e-02, 2.460e+00, 3.000e-02, 1.510e+00, 2.050e+00,
       1.680e+00, 1.820e+00, 1.330e+00, 9.400e-01, 9.300e-01])
```

```
df['Other_Sales'].isna()
```

```
0        False
1        False
2        False
3        False
4        False
         ...
16714    False
16715    False
16716    False
16717    False
16718    False
Name: Other_Sales, Length: 16719, dtype: bool
```

```
df['Other_Sales'].nunique()
```

```
155
```

```
missing_values=df.isnull().sum()
print(missing_values)
```

```
Name                 2
Platform             0
Year_of_Release    269
Genre                2
Publisher           54
NA_Sales             0
```

```
EU_Sales              0
JP_Sales              0
Other_Sales           0
Global_Sales          0
Critic_Score       8582
Critic_Count       8582
User_Score         6704
User_Count         9129
Developer          6623
Rating             6769
dtype: int64
```

```
data=pd.read_csv('/content/homelessness (1).csv')
data
print(data.head(10))
```

```
   Unnamed: 0              region                 state  individuals  \
0           0  East South Central               Alabama       2570.0   
1           1             Pacific                Alaska       1434.0   
2           2            Mountain               Arizona       7259.0   
3           3  West South Central              Arkansas       2280.0   
4           4             Pacific            California     109008.0   
5           5            Mountain              Colorado       7607.0   
6           6         New England           Connecticut       2280.0   
7           7      South Atlantic              Delaware        708.0   
8           8      South Atlantic  District of Columbia       3770.0   
9           9      South Atlantic               Florida      21443.0   

   family_members  state_pop  
0           864.0    4887681  
1           582.0     735139  
2          2606.0    7158024  
3           432.0    3009733  
4         20964.0   39461588  
5          3250.0    5691287  
6          1696.0    3571520  
7           374.0     965479  
8          3134.0     701547  
9          9587.0   21244317  
```

```
data.dropna(axis=1,inplace=True)
data
```

	Unnamed: 0	region	state	individuals	family_members	state_pop	family
0	0	East South Central	Alabama	2570.0	0.02	4887681	0.016593
1	1	Pacific	Alaska	1434.0	0.01	735139	0.011177
2	2	Mountain	Arizona	7259.0	0.05	7158024	0.050048
3	3	West South Central	Arkansas	2280.0	0.01	3009733	0.008297
4	4	Pacific	California	109008.0	0.40	39461588	0.402612
5	5	Mountain	Colorado	7607.0	0.06	5691287	0.062416
6	6	New England	Connecticut	2280.0	0.03	3571520	0.032572
7	7	South Atlantic	Delaware	708.0	0.01	965479	0.007183
8	8	South Atlantic	District of Columbia	3770.0	0.06	701547	0.060188
9	9	South Atlantic	Florida	21443.0	0.18	21244317	0.184118
10	10	South Atlantic	Georgia	6943.0	0.05	10511131	0.049088
11	11	Pacific	Hawaii	4131.0	0.05	1420593	0.046073
12	12	Mountain	Idaho	1297.0	0.01	1750536	0.013732
13	13	East North Central	Illinois	6752.0	0.07	12723071	0.074726

single feature scaling

It converts every value of a column between 0 and 1. Every number of a column will be divided by the max no of a coulmn.

```
data['family']=data['family_members']/data['family_members'].max()
data.head()
```

	Unnamed: 0	region	state	individuals	family_members	state_pop	family
0	0	East South Central	Alabama	2570.0	0.016593	4887681	0.016593
1	1	Pacific	Alaska	1434.0	0.011177	735139	0.011177
2	2	Mountain	Arizona	7259.0	0.050048	7158024	0.050048
3	3	West South Central	Arkansas	2280.0	0.008297	3009733	0.008297

```
data['family_members']=data['family'].round(2)
data.head()
```

	Unnamed: 0	region	state	individuals	family_members	state_pop	family
0	0	East South Central	Alabama	2570.0	0.02	4887681	0.016593
1	1	Pacific	Alaska	1434.0	0.01	735139	0.011177
2	2	Mountain	Arizona	7259.0	0.05	7158024	0.050048
3	3	West South Central	Arkansas	2280.0	0.01	3009733	0.008297

West

Min-Max

It same as single feature scaling, here also value of a column between 0 and 1. Every no.of coulmn will be subtractedby the min no.of that column and divided by the range of that column.

Central

```
data['state_pop']=(data['state_pop']-data['state_pop'].min())/(data['state_pop'].max()-data['state_pop'].min())
data.head()
```

	Unnamed: 0	region	state	individuals	family_members	state_pop	family
0	0	East South Central	0.110845	2570.0	0.02	0.110845	0.016593
1	1	Pacific	0.004051	1434.0	0.01	0.004051	0.011177
2	2	Mountain	0.169232	7259.0	0.05	0.169232	0.050048
3	3	West South Central	0.062548	2280.0	0.01	0.062548	0.008297

zscore

It converts every value of a column into a number around 0. Typical values obtained by a z-score transformation range from -3 and 3. The new value is calculated as the difference between the current value and the average value, divided by the standard deviation. The average value of a column can be obtained through the `mean()` function, while the standard deviation through the `std()` function.

```
data['individuals']=(data['individuals']-data['individuals'].mean())/data['individuals'].std()
data.head()
```

Data Normalization technique on categorial data

We used the label encoder library file and some variabes to store the columns and transform to fit them.

```
col_to_encode = 'region'
data_to_encode = data[col_to_encode]
lae = LabelEncoder()
lae.fit(data_to_encode)

encoded_data = lae.transform(data_to_encode)
data[col_to_encode] = encoded_data
data.head()
```

	Unnamed: 0	region	state	individuals	family_members	state_pop
0	0	1	Alabama	2570.0	864.0	4887681
1	1	5	Alaska	1434.0	582.0	735139
2	2	3	Arizona	7259.0	2606.0	7158024
3	3	8	Arkansas	2280.0	432.0	3009733
4	4	5	California	109008.0	20964.0	39461588

A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression.

```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(2)

print(mytripler(11))
```

```
22
```

apply() must be applied to every row (through the parameter axis = 1) and then through the lambda operator we can select the specific row and apply it the function set_pattern().

```
import re
def set_pattern(x):
    pattern = r'[(A-Z)]\w+,([A-Z])\w+'
    res = re.match(pattern, x)
    if res:
        x = x.replace(',', ', ')
    return x
```

```
data['state'] = data.apply(lambda x: set_pattern(x['state']), axis=1)
print(data)
```

```
    Unnamed: 0  region                 state  individuals  family_members  \
0            0       1               Alabama       2570.0           864.0
1            1       5                Alaska       1434.0           582.0
2            2       3               Arizona       7259.0          2606.0
3            3       8              Arkansas       2280.0           432.0
4            4       5            California     109008.0         20964.0
5            5       3              Colorado       7607.0          3250.0
6            6       4           Connecticut       2280.0          1696.0
7            7       6              Delaware        708.0           374.0
8            8       6  District of Columbia       3770.0          3134.0
9            9       6               Florida      21443.0          9587.0
10          10       6               Georgia       6943.0          2556.0
11          11       5                Hawaii       4131.0          2399.0
12          12       3                 Idaho       1297.0           715.0
13          13       0              Illinois       6752.0          3891.0
14          14       0               Indiana       3776.0          1482.0
15          15       7                  Iowa       1711.0          1038.0
16          16       7                Kansas       1443.0           773.0
17          17       1              Kentucky       2735.0           953.0
18          18       8             Louisiana       2540.0           519.0
19          19       4                 Maine       1450.0          1066.0
20          20       6              Maryland       4914.0          2230.0
21          21       4         Massachusetts       6811.0         13257.0
22          22       0              Michigan       5209.0          3142.0
23          23       7             Minnesota       3993.0          3250.0
24          24       1           Mississippi       1024.0           328.0
25          25       7              Missouri       3776.0          2107.0
26          26       3               Montana        983.0           422.0
27          27       7              Nebraska       1745.0           676.0
28          28       3                Nevada       7058.0           486.0
29          29       4         New Hampshire        835.0           615.0
30          30       2            New Jersey       6048.0          3350.0
31          31       3            New Mexico       1949.0           602.0
32          32       2              New York      39827.0         52070.0
33          33       6        North Carolina       6451.0          2817.0
34          34       7          North Dakota        467.0            75.0
35          35       0                  Ohio       6929.0          3320.0
36          36       8              Oklahoma       2823.0          1048.0
37          37       5                Oregon      11139.0          3337.0
38          38       2          Pennsylvania       8163.0          5349.0
39          39       4          Rhode Island        747.0           354.0
40          40       6        South Carolina       3082.0           851.0
41          41       7          South Dakota        836.0           323.0
42          42       1             Tennessee       6139.0          1744.0
43          43       8                 Texas      19199.0          6111.0
44          44       3                  Utah       1904.0           972.0
45          45       4               Vermont        780.0           511.0
46          46       6              Virginia       3928.0          2047.0
47          47       5            Washington      16424.0          5880.0
48          48       6         West Virginia       1021.0           222.0
49          49       0             Wisconsin       2740.0          2167.0
50          50       3               Wyoming        434.0           205.0

    state_pop
0     4887681
1      735139
2     7158024
3     3009733
```

Data Analysis

1. Univariate Analysis: Count Plot, Histogram, Skewness, Kurtosis, Joint Plot, Box Plot, Pie Chart, Line Chart, Distribution Plot
2. Bivariate Analysis: Scatter Plot, Joint Plot, Line Plot, Bar Plot, Box Plot
3. Multivariate Analysis: Pair Plot, Heatmap, Bar Plot, Line Plot, Scatter Plot

IV. Perform the following data exploration analysis on the downloaded dataset:

1. Count Plot
2. Box Plot
3. Joint Plot
4. Correlation
5. Skewness
6. Distribution Plot
7. Pair Plot
8. Heatmap

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style()
import numpy as pd
```

```
den=pd.read_csv("/content/tweets.csv")
```

```
den.head()
```

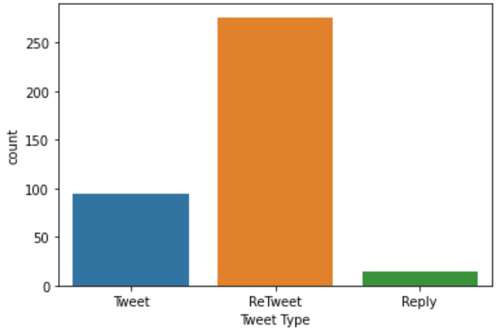
	Tweet Id	Tweet URL	Tweet Posted Time (UTC)	Tweet Con
0	"1167429261210218497"	https://twitter.com/animalhealthEU/status/1167...	30 Aug 2019 13:30:00	Pets change lives &a become a pa
1	"1167375334670557185"	https://twitter.com/PennyBrohnUK/status/116737...	30 Aug 2019 09:55:43	Another spot o #morethanmed bus ir
2	"1167237977615097861"	https://twitter.com/lordbyronaf/status/1167237...	30 Aug 2019 00:49:54	What a great t @HealthSourc @Local
3	"1167236897078480898"	https://twitter.com/CountessDavis/status/11672...	30 Aug 2019 00:45:37	What a great t @HealthSourc @Local
4	"1167228378191204353"	https://twitter.com/Local12/status/11672283781...	30 Aug 2019 00:11:46	What a great t @HealthSourc @Local

5 rows × 21 columns

count plot

The countplot is used to represent the occurrence(counts) of the observation present in the variable. It uses the concept of a bar chart for the visual depiction.

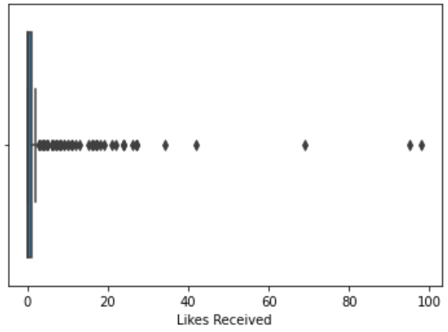
```
sns.countplot(x='Tweet Type',data=den)
plt.show()
```



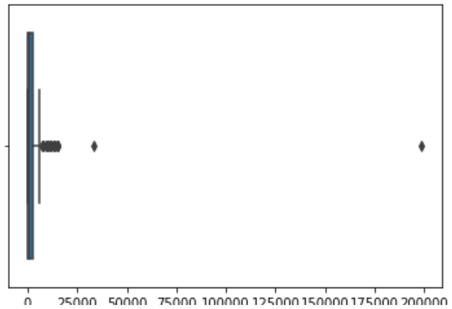
Box plot

Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum.

```
sns.boxplot(x='Likes Received',data=den)
plt.show()
```



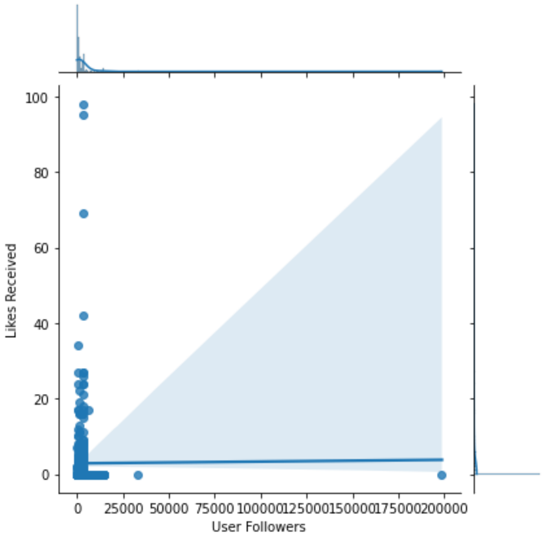

```
sns.boxplot(x='User Followers',data=den)
plt.show()
```



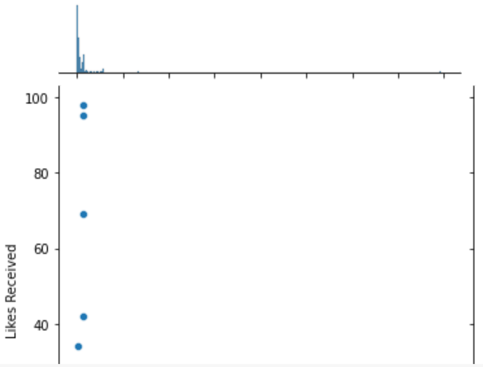
Joint plot

The joint plot is a way of understanding the relationship between two variables and the distribution of individuals of each variable. kind : { “scatter” | “reg” | “resid” | “kde” | “hex” } Kind of plot to draw.

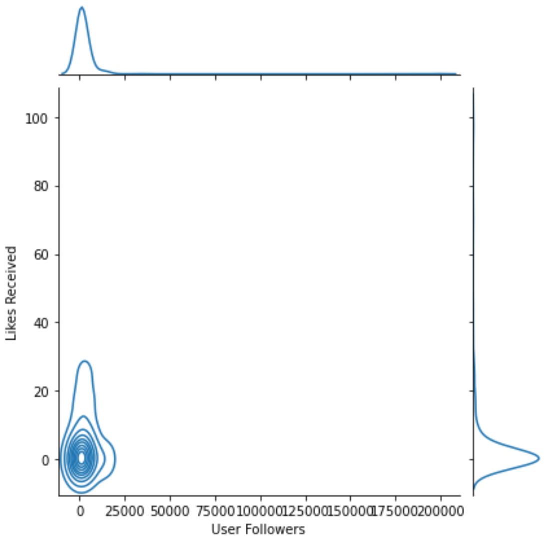
```
sns.jointplot(x='User Followers',y='Likes Received',data=den, kind='reg')
plt.show()
```



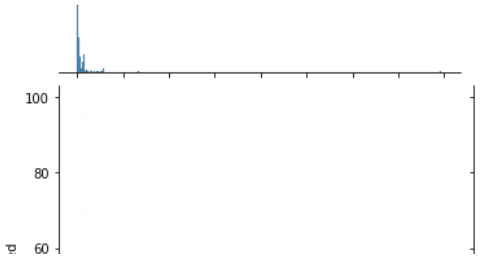

```
sns.jointplot(x='User Followers',y='Likes Received',data=den, kind='scatter')
plt.show()
```




```
sns.jointplot(x='User Followers',y='Likes Received',data=den, kind='kde')
plt.show()
```




```
sns.jointplot(x='User Followers',y='Likes Received',data=den, kind='hex')
plt.show()
```



correlation

The pearsonr() SciPy function can be used to calculate the Pearson's correlation coefficient between two data samples with the same length. Correlation is a statistical measure that expresses the extent to which two variables are linearly related. A null hypothesis is a type of statistical hypothesis that proposes that no statistical significance exists in a set of given observations.

```
from scipy.stats import pearsonr

def get_correlation(column1, column2, df):
    pearson_corr, p_value = pearsonr(df[column1], df[column2])
    print("Correlation between {} and {} is {}".format(column1, column2, pearson_corr))
    print("P-value of this correlation is {}".format(p_value))
```

```
get_correlation('User Followers','Likes Received', den)
```

```
Correlation between User Followers and Likes Received is 0.005301145045724385
P-value of this correlation is 0.9173170162772105
```

skewness

Skewness is a measure of the asymmetry of a distribution. A distribution is asymmetrical when its left and right side are not mirror images.

```
from scipy.stats import skew
den.skew(axis = 0, skipna = True)
plt.show()
```

```
<ipython-input-17-bb391122d770>:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select or
  den.skew(axis = 0, skipna = True)
```

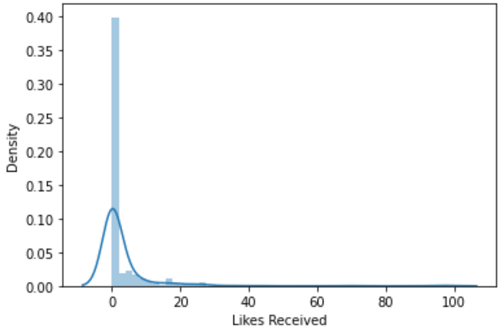
```
den['Likes Received'].skew(axis = 0, skipna = True)
plt.show()
```

Distribution plot

distplot() function is used to plot the distplot. The distplot represents the univariate distribution of data. The seaborn. distplot() function accepts the data variable as an argument and returns the plot with the density distribution.

```
sns.distplot(den['Likes Received'])
plt.show()
```

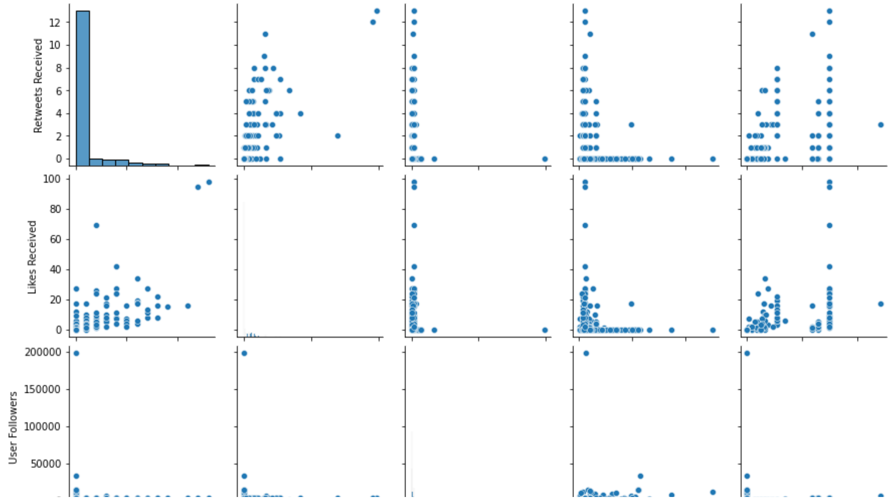
```
/usr/local/lib/python3.9/dist-packages/seaborn/distributions.py:2619: FutureWarning: `dist
  warnings.warn(msg, FutureWarning)
```



Pair plot

The Seaborn Pairplot allows us to plot pairwise relationships between variables within a dataset. This creates a nice visualisation and helps us understand the data by summarising a large amount of data in a single figure.

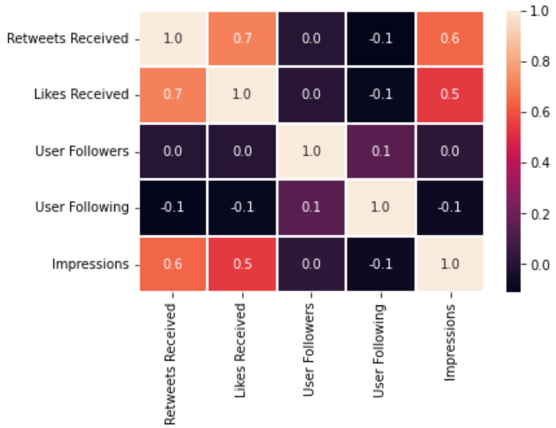
```
sns.pairplot(den)
plt.show()
```



Heat map

A heatmap is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colours. The Seaborn package allows the creation of annotated heatmaps which can be tweaked using Matplotlib tools as per the creator's requirement.

```
sns.heatmap(den.corr(),annot=True,fmt='.1f',linewidths=2)
plt.show()
```



5. Frame a problem statement based on the attributes of the dataset and predict the accuracy of the dependent variable using the values of the independent variables by using the following machine learning models:'

-> Logistic Regression Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.

-> Linear SVC Linear Support Vector Machine (Linear SVC) is an algorithm that attempts to find a hyperplane to maximize the distance between classified samples.

-> Decision Tree Classifier Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

-> Naive Bayes - Gaussian NB Gaussian Naïve Bayes classifier assumes that the data from each label is drawn from a simple Gaussian distribution. The Scikit-learn provides sklearn.naive_bayes.GaussianNB to implement the Gaussian Naïve Bayes algorithm for classification.

-> Ensemble Model - RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
df=pd.read_csv('/content/archive (3).zip')
df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

```
df=df[['BMI','Glucose','BloodPressure','Insulin','Age']]
df
```

	BMI	Glucose	BloodPressure	Insulin	Age
0	33.6	148	72	0	50
1	26.6	85	66	0	31
2	23.3	183	64	0	32
3	28.1	89	66	94	21
4	43.1	137	40	168	33

```
df['Glucose'].fillna(df['Glucose'].median())
```

```
0      148
1       85
2      183
3       89
4      137
      ...
763    101
764    122
765    121
766    126
767     93
Name: Glucose, Length: 768, dtype: int64
```

```
x=df.drop('Glucose',axis=1)
y=df['Glucose']
```

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
# load the diabetes dataset from a CSV file
diabetes_df = pd.read_csv('/content/diabetes.csv')
diabetes_df
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu

```
# split the dataset into features (X) and target (y)
X = diabetes_df.drop('Outcome', axis=1)
y = diabetes_df['Outcome']
```

```
# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
# create a logistic regression model
logreg = LogisticRegression()
```

764	2	122	70	27	0	36.8

```
# fit the model to the training data
logreg.fit(X_train, y_train)

# predict the target values for the testing data
y_pred = logreg.predict(X_test)

# evaluate the performance of the model
score = logreg.score(X_test, y_test)

print("Accuracy:", score)
```

```
Accuracy: 0.7467532467532467
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# split the dataset into features (X) and target (y)
X = diabetes_df.drop('Outcome', axis=1)
y = diabetes_df['Outcome']

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# create a linear support vector classification model
svc = LinearSVC()

# fit the model to the training data
svc.fit(X_train, y_train)

# predict the target values for the testing data
y_pred = svc.predict(X_test)

# evaluate the performance of the model
```

```
score = accuracy_score(y_test, y_pred)

print("Accuracy:", score)
```

```
Accuracy: 0.6363636363636364
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# create a decision tree classification model
dtc = DecisionTreeClassifier()

# fit the model to the training data
dtc.fit(X_train, y_train)

# predict the target values for the testing data
y_pred = dtc.predict(X_test)

# evaluate the performance of the model
score = accuracy_score(y_test, y_pred)

print("Accuracy:", score)
```

```
Accuracy: 0.7142857142857143
```

```
from sklearn.naive_bayes import GaussianNB
# create a Gaussian Naive Bayes classifier
gnb = GaussianNB()

# fit the model to the training data
gnb.fit(X_train, y_train)

# predict the target values for the testing data
y_pred = gnb.predict(X_test)

# evaluate the performance of the model
score = accuracy_score(y_test, y_pred)

print("Accuracy:", score)
```

```
Accuracy: 0.7662337662337663
```

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
# create a Random Forest classifier
rfc = RandomForestClassifier()

# fit the model to the training data
rfc.fit(X_train, y_train)

# predict the target values for the testing data
```

```
y_pred = rfc.predict(X_test)

# evaluate the performance of the model
score = accuracy_score(y_test, y_pred)

print("Random Forest Classifier Accuracy:", score)

# create a Gradient Boosting classifier
gbc = GradientBoostingClassifier()

# fit the model to the training data
gbc.fit(X_train, y_train)

# predict the target values for the testing data
y_pred = gbc.predict(X_test)

# evaluate the performance of the model
score = accuracy_score(y_test, y_pred)

print("Gradient Boosting Classifier Accuracy:", score)

# create an AdaBoost classifier
abc = AdaBoostClassifier()

# fit the model to the training data
abc.fit(X_train, y_train)

# predict the target values for the testing data
y_pred = abc.predict(X_test)

# evaluate the performance of the model
score = accuracy_score(y_test, y_pred)

print("AdaBoost Classifier Accuracy:", score)
```

```
Random Forest Classifier Accuracy: 0.7727272727272727
Gradient Boosting Classifier Accuracy: 0.7662337662337663
AdaBoost Classifier Accuracy: 0.7727272727272727
```