Experiment-9

Aim: Create a Machine Learning Model using Logistic Regression algorithm on ("Titanic.csv ") dataset.

Bhrigu Soni

Step-1: Data Pre-processing

1. Importing the required libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

2. Importing the dataset using the pandas library

In [ ]:
```
data = pd.read_csv("titanic.csv")
data.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [ ]:
```
data.shape
```

Out[ ]: (891, 12)

In [ ]:
```
data.describe()
```

Out[ ]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

3. Handling the missing values

In [ ]:
```
print('Handling missing values in the dataset:')
print(data.isnull().sum())
```

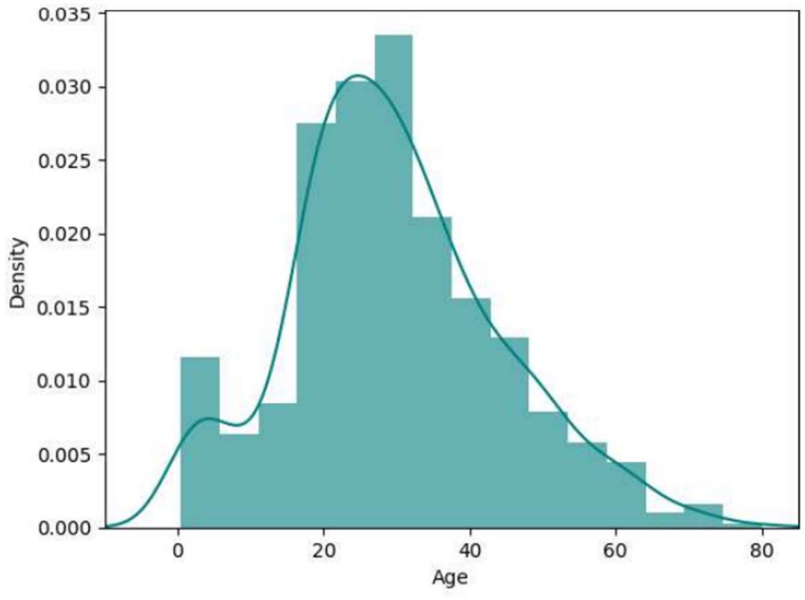
```
Handling missing values in the dataset:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

--> Handling the missing values in 'Age' variable

In [ ]:
```
# percent of missing "Age"
missing_age_percentage = (data['Age'].isnull().sum() / data.shape[0]) * 100

# printing the percentage
print('Percent of missing "Age" records is ', missing_age_percentage, "%")
```

```
Percent of missing "Age" records is  19.865319865319865 %
```

In [ ]:
```
# Assuming you have a list or array containing the 'Age' data
age_data = data["Age"]

# Create a histogram
plt.hist(age_data, bins=15, density=True,
         stacked=True, color='teal', alpha=0.6)

# Create a kernel density plot
age_data.plot(kind='kde', color='teal')

# Set the x-axis label
plt.xlabel('Age')

# Set the x-axis limit
plt.xlim(-10, 85)

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



Here, we have seen from above grapn it is right skewed so using mean should gives biased results so for this we use 'median'.

In [ ]:
```
# mean age
print('The mean of "Age" is ', (data["Age"].mean(skipna=True)))

# median age
print('The median of "Age" is', (data["Age"].median(skipna=True)))
```

```
The mean of "Age" is  29.69911764705882
The median of "Age" is 28.0
```

--> Handling the missing values in 'cabin'

In [ ]:
```
# percent of missing "Cabin"
print('Percent of missing "Cabin" records is ',
      ((data['Cabin'].isnull().sum()/data.shape[0])*100))
```

```
Percent of missing "Cabin" records is  77.10437710437711
```

--> Handling the missing values in 'embarked'

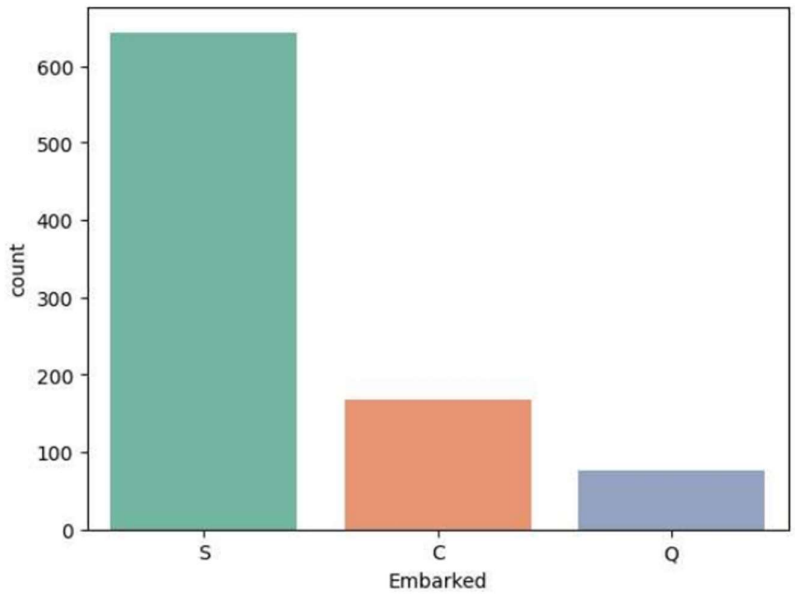
In [ ]:
```
# percent of missing "Embarked"
print('Percent of missing "Embarked" records is ',
      ((data['Embarked'].isnull().sum()/data.shape[0])*100))
```

```
Percent of missing "Embarked" records is  0.22446689113355783
```

There are only 2 (0.22%) missing values for "Embarked", so we can just impute with the port where most people boarded.

In [ ]:
```
print('''Boarded passengers grouped by port of embarkation
            (C = Cherbourg, Q = Queenstown, S = Southampton):''')
print(data['Embarked'].value_counts())

# plotting graph
sns.countplot(x='Embarked', data=data, palette='Set2')
plt.show()
```

```
Boarded passengers grouped by port of embarkation
            (C = Cherbourg, Q = Queenstown, S = Southampton):
Embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```



In [ ]:
```
print('The most common boarding port of embarkation is',
```

```
      data['Embarked'].value_counts().idxmax())
```

```
The most common boarding port of embarkation is S
```

By far the most passengers boarded in Southhampton, so we'll impute those 2 NaN's w/ "S".

--> Final Adjustments to remove missing values

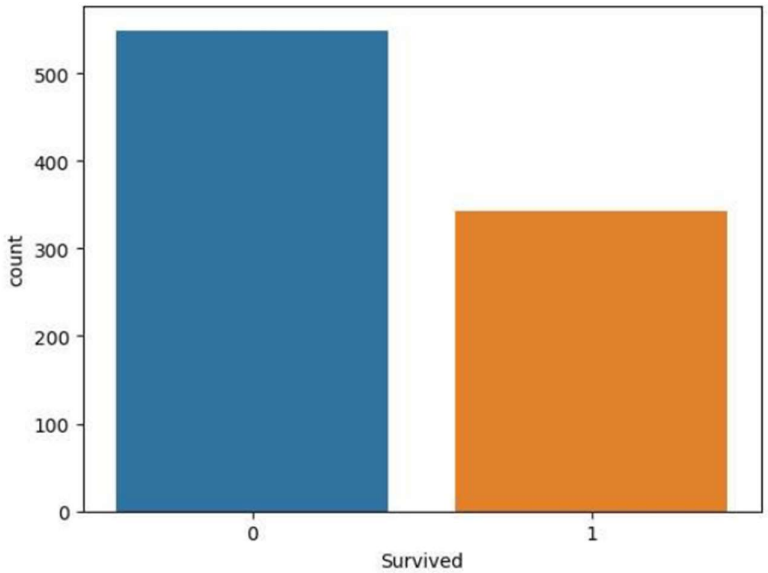
In [ ]:
```
# If "Age" is missing for a given row, I'll impute with 28 (median age).
data["Age"].fillna(data["Age"].median(skipna=True), inplace=True)

# I'll ignore "Cabin" as a variable.
# There are too many missing values for imputation. Based on the information available,
# it appears that this value is associated with the passenger's class and fare paid.
data.drop('Cabin', axis=1, inplace=True)

# If "Embarked" is missing for a riven row,
# I'll impute with "S" (the most common boarding port).
data["Embarked"].fillna(data['Embarked'].value_counts().idxmax(), inplace=True)

data.isnull().sum()
```

Out[ ]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

4. Exploratory Data Analysis

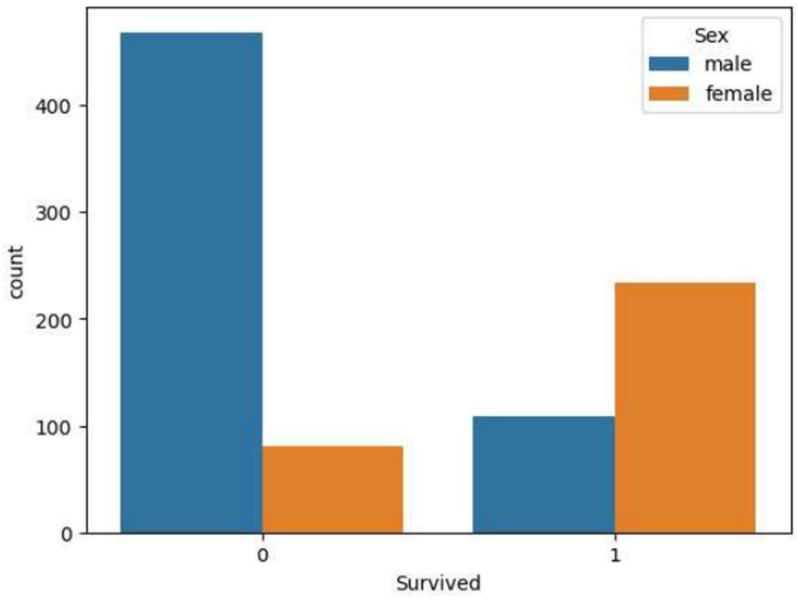
In [ ]:
```
sns.countplot(x="Survived", data=data)
```

Out[ ]:
```
<Axes: xlabel='Survived', ylabel='count'>
```



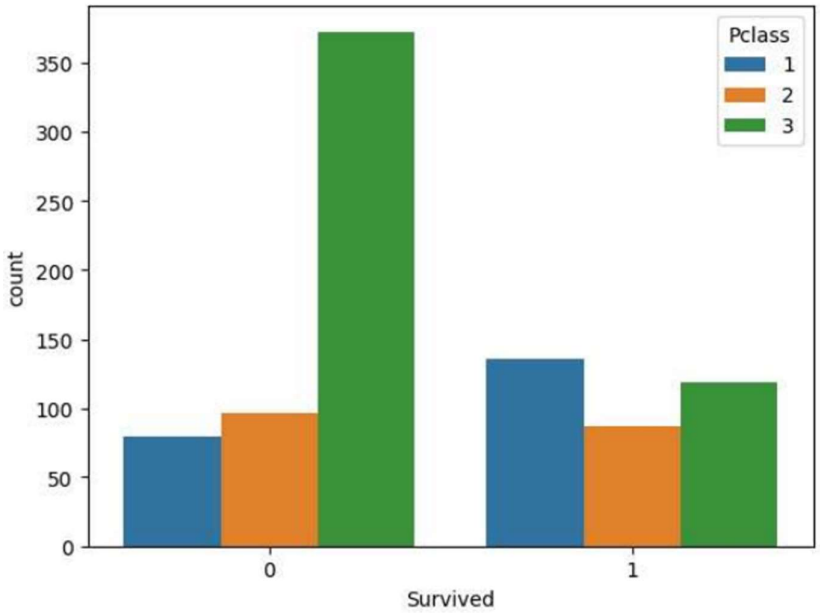
In [ ]:
```
sns.countplot(x="Survived", hue="Sex", data=data)
```

Out[ ]:
```
<Axes: xlabel='Survived', ylabel='count'>
```



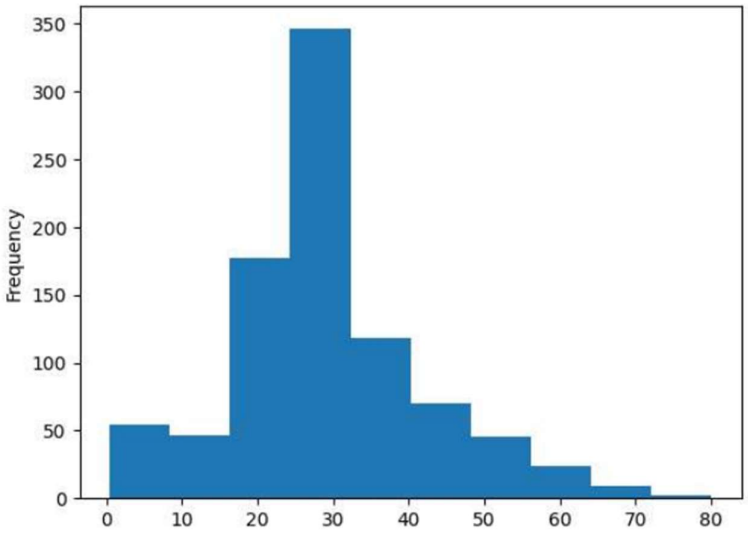
In [ ]:
```
sns.countplot(x="Survived", hue="Pclass", data=data)
```

Out[ ]: <Axes: xlabel='Survived', ylabel='count'>



In [ ]:
```
data["Age"].plot.hist()
```

Out[ ]: <Axes: ylabel='Frequency'>



Step-2: Adjusting additional variables

In [ ]:
```
data['TravelAlone'] = np.where((data["SibSp"] + data["Parch"]) > 0, 0, 1)
data.drop(['SibSp', 'Parch'], axis=1, inplace=True)
data.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	Ticket	Fare	Embarked	TravelAlone
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	A/5 21171	7.2500	S	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	PC 17599	71.2833	C	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	STON/O2. 3101282	7.9250	S	1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	113803	53.1000	S	0
4	5	0	3	Allen, Mr. William Henry	male	35.0	373450	8.0500	S	1

Step-3: Creating dummy variables for 'Pclass', 'Sex', and 'Embarked'

In [ ]:
```
data = pd.get_dummies(data, columns=["Pclass", "Embarked", "Sex"])
data.drop(['Sex_female', 'Pclass_3', 'Embarked_S',
           'PassengerId', 'Name', 'Ticket'], axis=1, inplace=True)
```

Step-4: Extracting independent and dependent variables

In [ ]:
```
X = data.drop('Survived', axis=1)
y = data['Survived']
```

Step-5: Splitting the dataset into Training and Testing dataset

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
```

Step-6: Fitting Logistic Regression to the Training set

In [ ]:
```
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

```
c:\Users\sonib\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\linear_model\_logistic.p
y: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(
```

Out[ ]:
```
▾ LogisticRegression

LogisticRegression()
```

Step-7: Predicting the Test Result

In [ ]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[ ]:
```
array([0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0], dtype=int64)
```

Step-8: Test accuracy of the result

In [ ]:
```
# Test accuracy of the result
classification_report(y_test, y_pred)
```

Out[ ]:
```
'              precision    recall  f1-score   support\n\n           0       0.84      0.84      0.84
110\n           1       0.74      0.74      0.74        69\n\n    accuracy                           0.8
0       179\n   macro avg       0.79      0.79      0.79       179\nweighted avg       0.80      0.80
0.80       179\n'
```

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[ ]:
```
array([[92, 18],
       [18, 51]], dtype=int64)
```

Step-9: Finding the Accuracy of the model

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy * 100, "%")
```

```
Accuracy: 79.88826815642457 %
```

Hence,the accuracy of the model is approximate 80%.

Experiment-10

Aim: Create a Machine Learning Model using Support Vector Machine algorithm on ("Breast Cancer.csv ") dataset.

Bhrigu Soni

Step-1: Data Pre-processing

1. Importing the required libraries

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import pandas as pd
import numpy as np
import seaborn as sns
```

2. Load the Breast Cancer Wisconsin dataset

In [ ]:
```
data = pd.read_csv("Breast Cancer.csv")
data.head().T
```

Out[ ]:

	0	1	2	3	4
id	842302	842517	84300903	84348301	84358402
diagnosis	M	M	M	M	M
radius_mean	17.99	20.57	19.69	11.42	20.29
texture_mean	10.38	17.77	21.25	20.38	14.34
perimeter_mean	122.8	132.9	130.0	77.58	135.1
area_mean	1001.0	1326.0	1203.0	386.1	1297.0
smoothness_mean	0.1184	0.08474	0.1096	0.1425	0.1003
compactness_mean	0.2776	0.07864	0.1599	0.2839	0.1328
concavity_mean	0.3001	0.0869	0.1974	0.2414	0.198
concave points_mean	0.1471	0.07017	0.1279	0.1052	0.1043
symmetry_mean	0.2419	0.1812	0.2069	0.2597	0.1809
fractal_dimension_mean	0.07871	0.05667	0.05999	0.09744	0.05883
radius_se	1.095	0.5435	0.7456	0.4956	0.7572
texture_se	0.9053	0.7339	0.7869	1.156	0.7813
perimeter_se	8.589	3.398	4.585	3.445	5.438
area_se	153.4	74.08	94.03	27.23	94.44
smoothness_se	0.006399	0.005225	0.00615	0.00911	0.01149
compactness_se	0.04904	0.01308	0.04006	0.07458	0.02461
concavity_se	0.05373	0.0186	0.03832	0.05661	0.05688
concave points_se	0.01587	0.0134	0.02058	0.01867	0.01885
symmetry_se	0.03003	0.01389	0.0225	0.05963	0.01756
fractal_dimension_se	0.006193	0.003532	0.004571	0.009208	0.005115

radius_worst	25.38	24.99	23.57	14.91	22.54
texture_worst	17.33	23.41	25.53	26.5	16.67
perimeter_worst	184.6	158.8	152.5	98.87	152.2
area_worst	2019.0	1956.0	1709.0	567.7	1575.0
smoothness_worst	0.1622	0.1238	0.1444	0.2098	0.1374
compactness_worst	0.6656	0.1866	0.4245	0.8663	0.205
concavity_worst	0.7119	0.2416	0.4504	0.6869	0.4
concave points_worst	0.2654	0.186	0.243	0.2575	0.1625
symmetry_worst	0.4601	0.275	0.3613	0.6638	0.2364
fractal_dimension_worst	0.1189	0.08902	0.08758	0.173	0.07678
Unnamed: 32	NaN	NaN	NaN	NaN	NaN

In [ ]:
```
data.shape
```

Out[ ]:
```
(569, 33)
```

In [ ]:
```
data.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
id	569.0	3.037183e+07	1.250206e+08	8670.000000	869218.000000	906024.000000	8.813129e+06	9.113205e+08
radius_mean	569.0	1.412729e+01	3.524049e+00	6.981000	11.700000	13.370000	1.578000e+01	2.811000e+01
texture_mean	569.0	1.928965e+01	4.301036e+00	9.710000	16.170000	18.840000	2.180000e+01	3.928000e+01
perimeter_mean	569.0	9.196903e+01	2.429898e+01	43.790000	75.170000	86.240000	1.041000e+02	1.885000e+02
area_mean	569.0	6.548891e+02	3.519141e+02	143.500000	420.300000	551.100000	7.827000e+02	2.501000e+03
smoothness_mean	569.0	9.636028e-02	1.406413e-02	0.052630	0.086370	0.095870	1.053000e-01	1.634000e-01
compactness_mean	569.0	1.043410e-01	5.281276e-02	0.019380	0.064920	0.092630	1.304000e-01	3.454000e-01
concavity_mean	569.0	8.879932e-02	7.971981e-02	0.000000	0.029560	0.061540	1.307000e-01	4.268000e-01
concave points_mean	569.0	4.891915e-02	3.880284e-02	0.000000	0.020310	0.033500	7.400000e-02	2.012000e-01
symmetry_mean	569.0	1.811619e-01	2.741428e-02	0.106000	0.161900	0.179200	1.957000e-01	3.040000e-01
fractal_dimension_mean	569.0	6.279761e-02	7.060363e-03	0.049960	0.057700	0.061540	6.612000e-02	9.744000e-02
radius_se	569.0	4.051721e-01	2.773127e-01	0.111500	0.232400	0.324200	4.789000e-01	2.873000e+00
texture_se	569.0	1.216853e+00	5.516484e-01	0.360200	0.833900	1.108000	1.474000e+00	4.885000e+00
perimeter_se	569.0	2.866059e+00	2.021855e+00	0.757000	1.606000	2.287000	3.357000e+00	2.198000e+01
area_se	569.0	4.033708e+01	4.549101e+01	6.802000	17.850000	24.530000	4.519000e+01	5.422000e+02
smoothness_se	569.0	7.040979e-03	3.002518e-03	0.001713	0.005169	0.006380	8.146000e-03	3.113000e-02
compactness_se	569.0	2.547814e-02	1.790818e-02	0.002252	0.013080	0.020450	3.245000e-02	1.354000e-01
concavity_se	569.0	3.189372e-02	3.018606e-02	0.000000	0.015090	0.025890	4.205000e-02	3.960000e-01
concave points_se	569.0	1.179614e-02	6.170285e-03	0.000000	0.007638	0.010930	1.471000e-02	5.279000e-02
symmetry_se	569.0	2.054230e-02	8.266372e-03	0.007882	0.015160	0.018730	2.348000e-02	7.895000e-02
fractal_dimension_se	569.0	3.794904e-03	2.646071e-03	0.000895	0.002248	0.003187	4.558000e-03	2.984000e-02
radius_worst	569.0	1.626919e+01	4.833242e+00	7.930000	13.010000	14.970000	1.879000e+01	3.604000e+01

texture_worst	569.0	2.567722e+01	6.146258e+00	12.020000	21.080000	25.410000	2.972000e+01	4.954000e+01
perimeter_worst	569.0	1.072612e+02	3.360254e+01	50.410000	84.110000	97.660000	1.254000e+02	2.512000e+02
area_worst	569.0	8.805831e+02	5.693570e+02	185.200000	515.300000	686.500000	1.084000e+03	4.254000e+03
smoothness_worst	569.0	1.323686e-01	2.283243e-02	0.071170	0.116600	0.131300	1.460000e-01	2.226000e-01
compactness_worst	569.0	2.542650e-01	1.573365e-01	0.027290	0.147200	0.211900	3.391000e-01	1.058000e+00
concavity_worst	569.0	2.721885e-01	2.086243e-01	0.000000	0.114500	0.226700	3.829000e-01	1.252000e+00
concave points_worst	569.0	1.146062e-01	6.573234e-02	0.000000	0.064930	0.099930	1.614000e-01	2.910000e-01
symmetry_worst	569.0	2.900756e-01	6.186747e-02	0.156500	0.250400	0.282200	3.179000e-01	6.638000e-01
fractal_dimension_worst	569.0	8.394582e-02	1.806127e-02	0.055040	0.071460	0.080040	9.208000e-02	2.075000e-01
Unnamed: 32	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3. Handling the missing values

In [ ]:
```
print('Handling missing values in the dataset:')
print(data.isnull().sum())
```

```
Handling missing values in the dataset:
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

--> Adjustments to remove missing values

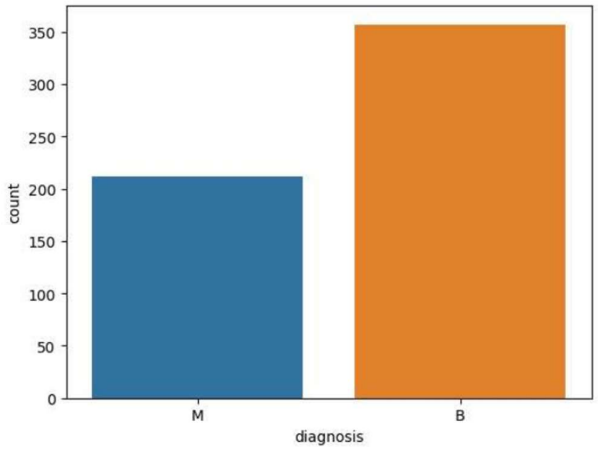
In [ ]:
```
data.drop('Unnamed: 32',axis=1,inplace=True)
data.isnull().sum()
```

Out[ ]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

4. Exploratory Data Analysis

In [ ]:
```
sns.countplot(x='diagnosis',data=data)
```

Out[ ]:
```
<Axes: xlabel='diagnosis', ylabel='count'>
```



Step-2: Adjusting additional variables

In [ ]:
```
data.replace({'M':1,'B':0},inplace=True)
data.head().T
```

Out[ ]:

	0	1	2	3	4
id	842302.000000	842517.000000	8.430090e+07	8.434830e+07	8.435840e+07
diagnosis	1.000000	1.000000	1.000000e+00	1.000000e+00	1.000000e+00
radius_mean	17.990000	20.570000	1.969000e+01	1.142000e+01	2.029000e+01
texture_mean	10.380000	17.770000	2.125000e+01	2.038000e+01	1.434000e+01
perimeter_mean	122.800000	132.900000	1.300000e+02	7.758000e+01	1.351000e+02
area_mean	1001.000000	1326.000000	1.203000e+03	3.861000e+02	1.297000e+03
smoothness_mean	0.118400	0.084740	1.096000e-01	1.425000e-01	1.003000e-01
compactness_mean	0.277600	0.078640	1.599000e-01	2.839000e-01	1.328000e-01
concavity_mean	0.300100	0.086900	1.974000e-01	2.414000e-01	1.980000e-01
concave points_mean	0.147100	0.070170	1.279000e-01	1.052000e-01	1.043000e-01
symmetry_mean	0.241900	0.181200	2.069000e-01	2.597000e-01	1.809000e-01
fractal_dimension_mean	0.078710	0.056670	5.999000e-02	9.744000e-02	5.883000e-02
radius_se	1.095000	0.543500	7.456000e-01	4.956000e-01	7.572000e-01
texture_se	0.905300	0.733900	7.869000e-01	1.156000e+00	7.813000e-01
perimeter_se	8.589000	3.398000	4.585000e+00	3.445000e+00	5.438000e+00
area_se	153.400000	74.080000	9.403000e+01	2.723000e+01	9.444000e+01
smoothness_se	0.006399	0.005225	6.150000e-03	9.110000e-03	1.149000e-02
compactness_se	0.049040	0.013080	4.006000e-02	7.458000e-02	2.461000e-02
concavity_se	0.053730	0.018600	3.832000e-02	5.661000e-02	5.688000e-02
concave points_se	0.015870	0.013400	2.058000e-02	1.867000e-02	1.885000e-02
symmetry_se	0.030030	0.013890	2.250000e-02	5.963000e-02	1.756000e-02
fractal_dimension_se	0.006193	0.003532	4.571000e-03	9.208000e-03	5.115000e-03
radius_worst	25.380000	24.990000	2.357000e+01	1.491000e+01	2.254000e+01
texture_worst	17.330000	23.410000	2.553000e+01	2.650000e+01	1.667000e+01
perimeter_worst	184.600000	158.800000	1.525000e+02	9.887000e+01	1.522000e+02
area_worst	2019.000000	1956.000000	1.709000e+03	5.677000e+02	1.575000e+03
smoothness_worst	0.162200	0.123800	1.444000e-01	2.098000e-01	1.374000e-01
compactness_worst	0.665600	0.186600	4.245000e-01	8.663000e-01	2.050000e-01
concavity_worst	0.711900	0.241600	4.504000e-01	6.869000e-01	4.000000e-01
concave points_worst	0.265400	0.186000	2.430000e-01	2.575000e-01	1.625000e-01
symmetry_worst	0.460100	0.275000	3.613000e-01	6.638000e-01	2.364000e-01
fractal_dimension_worst	0.118900	0.089020	8.758000e-02	1.730000e-01	7.678000e-02

Step-3: Extracting independent and dependent variables

In [ ]:
```
X=data.drop(['id','diagnosis'],axis=1)
y=data['diagnosis']
```

Step-4: Splitting the dataset into Training and Testing dataset

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
```

Step-5: Fitting Support Vector to the Training set

In [ ]:
```
classifier = SVC()
classifier.fit(X_train, y_train)
```

Out[ ]:
```
SVC()
```

Step-6: Predicting the Test Result

In [ ]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0], dtype=int64)
```

Step-7: Test accuracy of the result

In [ ]:
```
# Test accuracy of the result
classification_report(y_test, y_pred)
```

Out[ ]:
```
'              precision    recall  f1-score   support\n\n           0       0.90      0.99      0.94
67\n           1       0.98      0.85      0.91        47\n\n    accuracy                           0.93
114\n   macro avg       0.94      0.92      0.93       114\nweighted avg       0.93      0.93      0.93
114\n'
```

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[ ]:
```
array([[66,  1],
       [ 7, 40]], dtype=int64)
```

Step-8: Finding the Accuracy of the model

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy * 100, "%")
```

```
Accuracy: 92.98245614035088 %
```

Hence,the accuracy of the model is approximate 93%.

Experiment-11

Aim: Create a Machine Learning Model using Random Forest algorithm on ("Loan Prediction.csv ") dataset.

Bhrigu Soni

Step-1: Data Pre-processing

1. Importing the required libraries

In [ ]:
```
# Step-1: Data Pre-processing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns;sns.set(style='whitegrid')
from sklearn.preprocessing import LabelEncoder
```

2. Importing the dataset using the pandas library

In [ ]:
```
data = pd.read_csv("Loan Prediction.csv")
data.head().T
```

Out[ ]:

	0	1	2	3	4
Loan_ID	LP001002	LP001003	LP001005	LP001006	LP001008
Gender	Male	Male	Male	Male	Male
Married	No	Yes	Yes	Yes	No
Dependents	0	1	0	0	0
Education	Graduate	Graduate	Graduate	Not Graduate	Graduate
Self_Employed	No	No	Yes	No	No
ApplicantIncome	5849	4583	3000	2583	6000
CoapplicantIncome	0.0	1508.0	0.0	2358.0	0.0
LoanAmount	NaN	128.0	66.0	120.0	141.0
Loan_Amount_Term	360.0	360.0	360.0	360.0	360.0
Credit_History	1.0	1.0	1.0	1.0	1.0
Property_Area	Urban	Rural	Urban	Urban	Urban
Loan_Status	Y	N	Y	Y	Y

In [ ]:
```
data.shape
```

Out[ ]: (614, 13)

In [ ]:
```
data.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
ApplicantIncome	614.0	5403.459283	6109.041673	150.0	2877.5	3812.5	5795.00	81000.0
CoapplicantIncome	614.0	1621.245798	2926.248369	0.0	0.0	1188.5	2297.25	41667.0
LoanAmount	592.0	146.412162	85.587325	9.0	100.0	128.0	168.00	700.0
Loan_Amount_Term	600.0	342.000000	65.120410	12.0	360.0	360.0	360.00	480.0
Credit_History	564.0	0.842199	0.364878	0.0	1.0	1.0	1.00	1.0

3. Handling the missing values

In [ ]:
```
print('Handling missing values in the dataset:')
print(data.isnull().sum())
```

```
Handling missing values in the dataset:
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

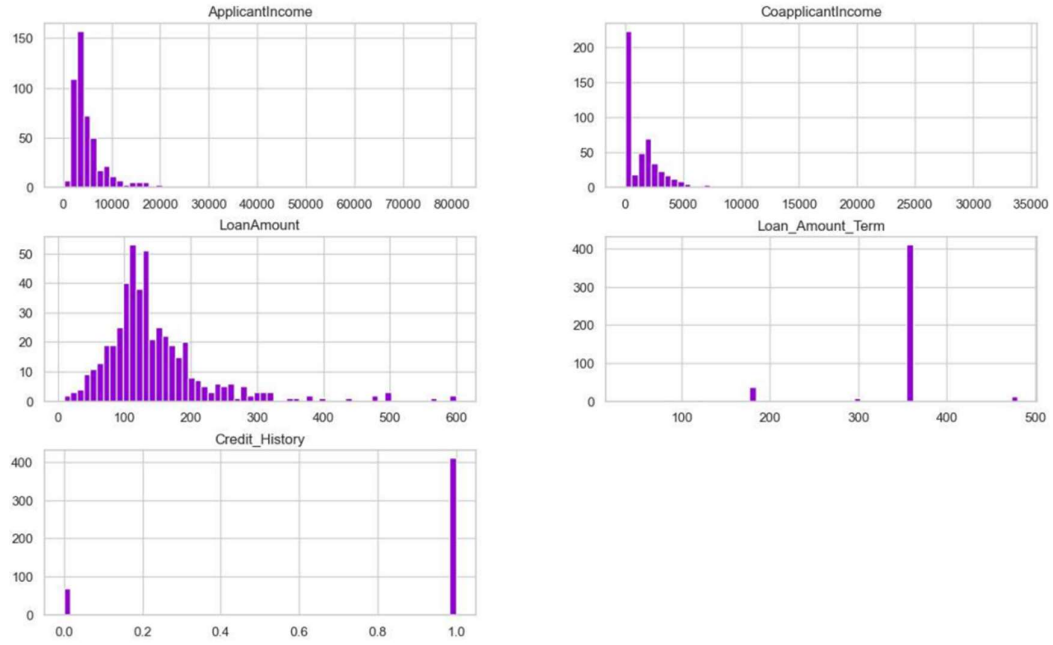
--> Adjustments to remove missing values

In [ ]:
```
data=data.dropna()
data = data.drop(['Loan_ID'], axis=1)
data.isnull().sum()
```

Out[ ]:
```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

4. Exploratory Data Analysis

In [ ]:
```
data.hist(bins=60, figsize=(15,9),color='darkviolet');plt.show()
```



Step-2: Creating categorical encoding for 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed','Credit_History', 'Property_Area', 'Loan_Status'

In [ ]:
```
# Encode categorical variables
categorical_cols = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
                    'Credit_History', 'Property_Area', 'Loan_Status']
for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
```

Step-3: Extracting independent and dependent variables

In [ ]:
```
# Split the data into features and target variable
y = data['Loan_Status']
X= data.drop(['Loan_Status'], axis=1)
```

Step-4: Splitting the dataset into Training and Testing dataset

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

Step-5: Fitting Random Forest to the Training set

In [ ]:
```
classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)
```

Out[ ]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

Step-6: Predicting the Test Result

In [ ]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[ ]:
```
array([1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1])
```

Step-7: Test accuracy of the result

In [ ]:
```
# Test accuracy of the result
classification_report(y_test, y_pred)
```

Out[ ]:
```
'              precision    recall  f1-score   support\n\n           0       0.82      0.50      0.62
28\n           1       0.82      0.96      0.88        68\n\n    accuracy                           0.82
96\n   macro avg       0.82      0.73      0.75        96\nweighted avg       0.82      0.82      0.81
96\n'
```

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[ ]:
```
array([[14, 14],
       [ 3, 65]], dtype=int64)
```

Step-8: Finding the Accuracy of the model

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy * 100, "%")
```

```
Accuracy: 82.29166666666666 %
```

Hence,the accuracy of the model is approximate 83%.

Experiment-12

Aim: Create a Machine Learning Model using Decision Tree algorithm on ("Drug.csv ") dataset.

Bhrigu Soni

Step-1: Data Pre-processing

1. Importing the required libraries

In [ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
```

2. Importing the dataset using the pandas library

In [ ]:
```
data = pd.read_csv("Drug.csv")
data.head()
```

Out[ ]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

In [ ]:
```
data.shape
```

Out[ ]:
```
(200, 6)
```

In [ ]:
```
data.describe()
```

Out[ ]:

	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

3. Handling the missing values

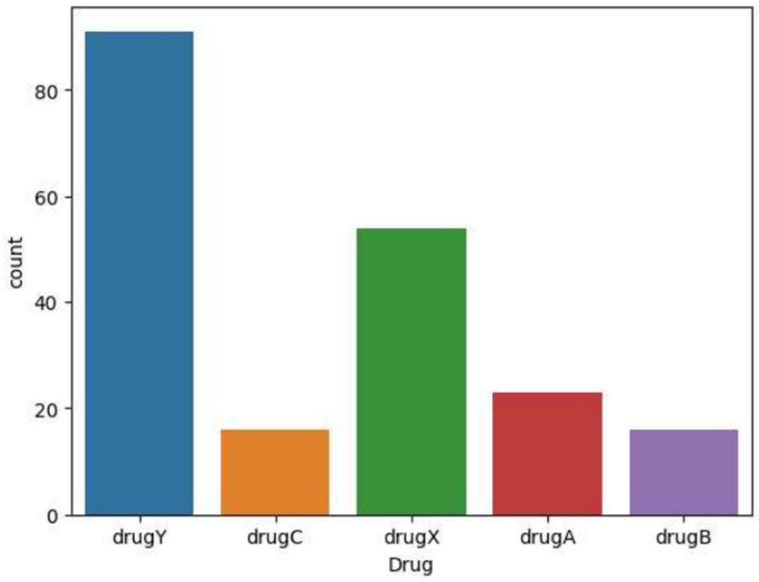
In [ ]:
```
print('Handling missing values in the dataset:')
print(data.isnull().sum())
```

```
Handling missing values in the dataset:
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

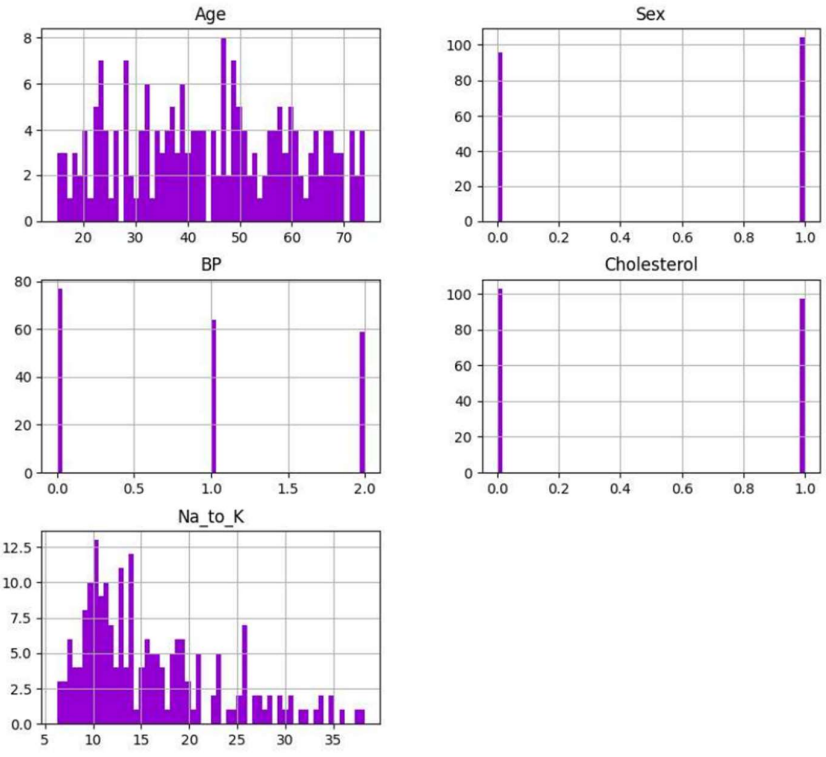
4. Exploratory Data Analysis

In [ ]:
```
sns.countplot(x="Drug", data=data)
```

Out[ ]: <Axes: xlabel='Drug', ylabel='count'>



In [ ]:
```
data.hist(bins=60, figsize=(10,9),color='darkviolet');plt.show()
```



Step-2: Creating Creating categorical encoding for 'Sex','BP','Cholesterol'.

In [ ]:
```
LE=LabelEncoder()
data['Sex']=LE.fit_transform(data['Sex'])
data['BP']=LE.fit_transform(data['BP'])
data['Cholesterol']=LE.fit_transform(data['Cholesterol'])
data
```

Out[ ]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	0	0	0	25.355	drugY
1	47	1	1	0	13.093	drugC
2	47	1	1	0	10.114	drugC
3	28	0	2	0	7.798	drugX
4	61	0	1	0	18.043	drugY
...	...	...	...	...	...	...
195	56	0	1	0	11.567	drugC
196	16	1	1	0	12.006	drugC
197	52	1	2	0	9.894	drugX
198	23	1	2	1	14.020	drugX
199	40	0	1	1	11.349	drugX

200 rows × 6 columns

Step-3: Extracting independent and dependent variables

In [ ]:
```
X = data.drop('Drug', axis=1)
y = data['Drug']
```

Step-4: Splitting the dataset into Training and Testing dataset

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
```

Step-5: Fitting Decision Tree to the Training set

In [ ]:
```
classifier = tree.DecisionTreeClassifier()
classifier.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Step-6: Predicting the Test Result

In [ ]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[ ]:
```
array(['drugC', 'drugX', 'drugY', 'drugY', 'drugY', 'drugX', 'drugX',
       'drugX', 'drugY', 'drugX', 'drugA', 'drugY', 'drugY', 'drugY',
       'drugB', 'drugC', 'drugY', 'drugY', 'drugX', 'drugY', 'drugY',
       'drugX', 'drugX', 'drugX', 'drugY', 'drugY', 'drugY', 'drugY',
       'drugY', 'drugX', 'drugX', 'drugC', 'drugA', 'drugX', 'drugY',
       'drugY', 'drugX', 'drugY', 'drugA', 'drugX'], dtype=object)
```

Step-7: Test accuracy of the result

In [ ]:
```
# Test accuracy of the result
classification_report(y_test, y_pred)
```

Out[ ]:
```
'              precision    recall  f1-score   support\n\n       drugA       1.00      1.00      1.00
3\n       drugB       1.00      1.00      1.00         1\n       drugC       1.00      1.00      1.00
3\n       drugX       1.00      1.00      1.00        14\n       drugY       1.00      1.00      1.00
19\n\n    accuracy                           1.00        40\n   macro avg       1.00      1.00      1.00
40\nweighted avg       1.00      1.00      1.00        40\n'
```

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[ ]:
```
array([[ 3,  0,  0,  0,  0],
       [ 0,  1,  0,  0,  0],
       [ 0,  0,  3,  0,  0],
       [ 0,  0,  0, 14,  0],
       [ 0,  0,  0,  0, 19]], dtype=int64)
```

Step-8: Finding the Accuracy of the model

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy * 100, "%")
```

```
Accuracy: 100.0 %
```

Hence,the accuracy of the model is 100%.

Experiment-13

Aim: Create a Machine Learning Model using KMeans Clustering algorithm on ("Meta Data.csv ") dataset.

Bhrigu Soni

Step-1: Data Pre-processing

1. Importing the required libraries

In [ ]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
```

2. Importing the dataset using the pandas library

In [ ]:
```
data = pd.read_csv("Meta Data.csv")
data.head().T
```

Out[ ]:

	0	1	
status_id	246675545449582_1649696485147474	246675545449582_1649426988507757	246675545449582_1648730
status_type	video	photo	
status_published	4/22/2018 6:00	4/21/2018 22:45	4/2
num_reactions	529	150	
num_comments	512	0	
num_shares	262	0	
num_likes	432	150	
num_loves	92	0	
num_wows	3	0	
num_hahas	1	0	
num_sads	1	0	
num_angrys	0	0	
Column1	NaN	NaN	
Column2	NaN	NaN	
Column3	NaN	NaN	
Column4	NaN	NaN	

In [ ]:
```
data.shape
```

Out[ ]:
```
(7050, 16)
```

In [ ]:
```
data.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
num_reactions	7050.0	230.117163	462.625309	0.0	17.0	59.5	219.00	4710.0
num_comments	7050.0	224.356028	889.636820	0.0	0.0	4.0	23.00	20990.0
num_shares	7050.0	40.022553	131.599965	0.0	0.0	0.0	4.00	3424.0
num_likes	7050.0	215.043121	449.472357	0.0	17.0	58.0	184.75	4710.0
num_loves	7050.0	12.728652	39.972930	0.0	0.0	0.0	3.00	657.0
num_wows	7050.0	1.289362	8.719650	0.0	0.0	0.0	0.00	278.0
num_hahas	7050.0	0.696454	3.957183	0.0	0.0	0.0	0.00	157.0
num_sads	7050.0	0.243688	1.597156	0.0	0.0	0.0	0.00	51.0
num_angrys	7050.0	0.113191	0.726812	0.0	0.0	0.0	0.00	31.0
Column1	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Column2	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Column3	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Column4	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3. Handling the missing values

In [ ]:
```
print('Handling missing values in the dataset:')
print(data.isnull().sum())
```

```
Handling missing values in the dataset:
status_id              0
status_type            0
status_published       0
num_reactions          0
num_comments           0
num_shares             0
num_likes              0
num_loves              0
num_wows               0
num_hahas              0
num_sads               0
num_angrys             0
Column1             7050
Column2             7050
Column3             7050
Column4             7050
dtype: int64
```

--> Adjustments to remove missing values

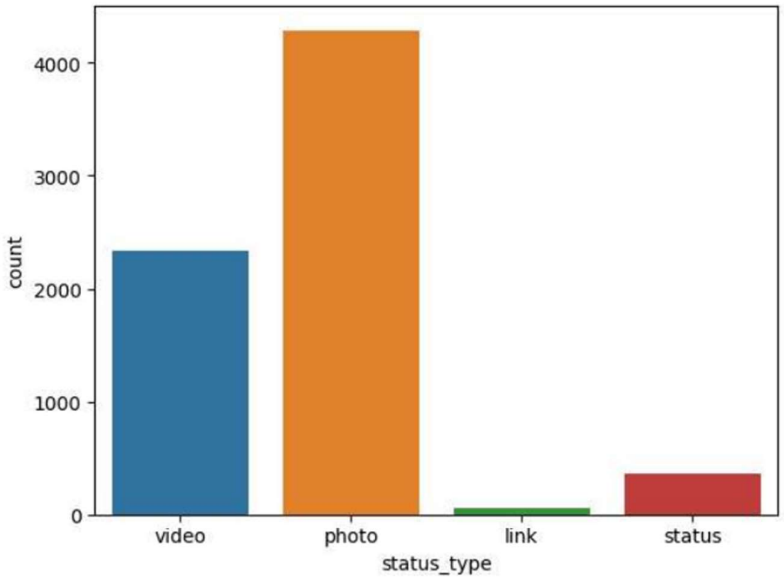
In [ ]:
```
data.drop(['Column1','Column2','Column3','Column4'], axis=1, inplace=True)
data.isnull().sum()
```

Out[ ]:
```
status_id           0
status_type         0
status_published    0
num_reactions       0
num_comments        0
num_shares          0
num_likes           0
num_loves           0
num_wows            0
num_hahas           0
num_sads            0
num_angrys          0
dtype: int64
```

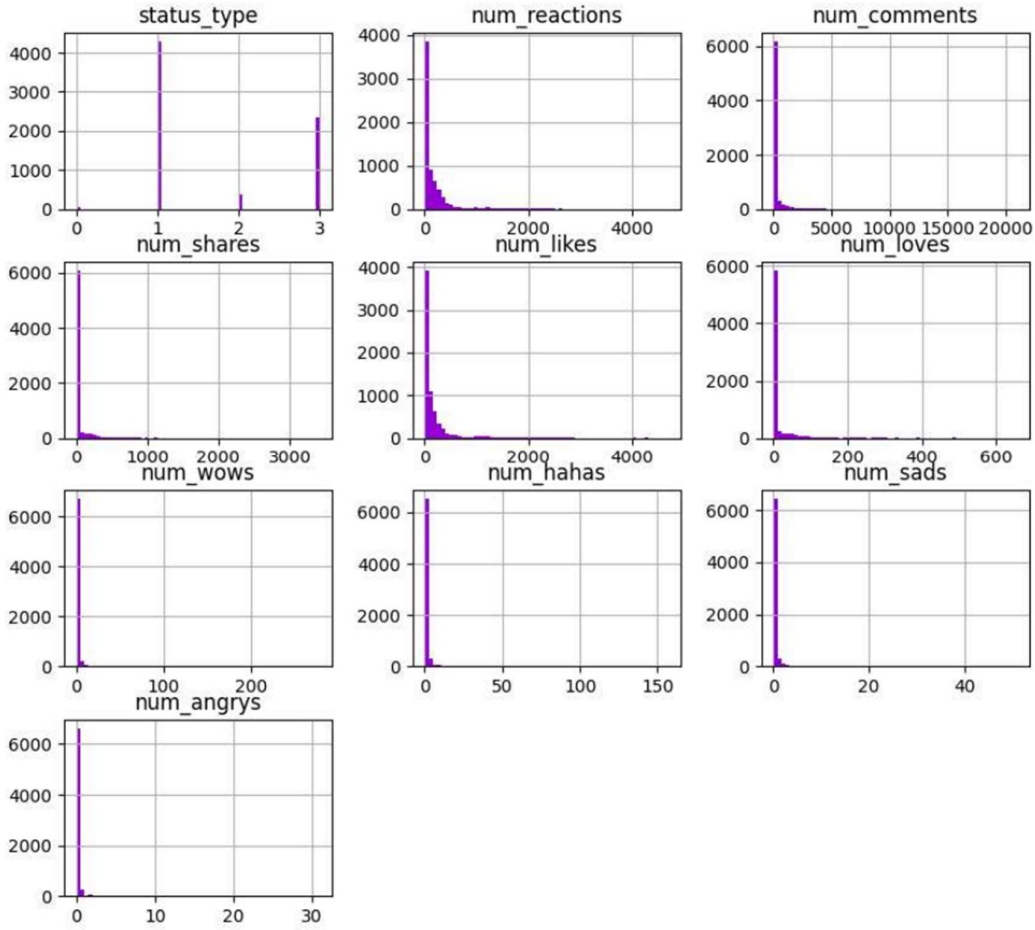
4. Exploratory Data Analysis

In [ ]:
```
sns.countplot(x="status_type", data=data)
```

Out[ ]: <Axes: xlabel='status_type', ylabel='count'>



In [ ]:
```
data.hist(bins=60, figsize=(10,9),color='darkviolet');plt.show()
```



Step-2: Adjusting additional variables

Explore `status_id` variable

In [ ]:
```
# view the labels in the variable

data['status_id'].unique()
```

Out[ ]:
```
array(['246675545449582_1649696485147474',
       '246675545449582_1649426988507757',
       '246675545449582_1648730588577397', ...,
       '1050855161656896_1060126464063099',
       '1050855161656896_1058663487542730',
       '1050855161656896_1050858841656528'], dtype=object)
```

In [ ]:
```
# view how many different types of variables are there

len(data['status_id'].unique())
```

Out[ ]:
```
6997
```

We can see that there are 6997 unique labels in the `status_id` variable. The total number of instances in the dataset is 7050. So, it is approximately a unique identifier for each of the instances. Thus this is not a variable that we can use. Hence, I will drop it.

Explore `status_published` variable

In [ ]:
```
# view the labels in the variable

data['status_published'].unique()
```

Out[ ]:
```
array(['4/22/2018 6:00', '4/21/2018 22:45', '4/21/2018 6:17', ...,
       '9/21/2016 23:03', '9/20/2016 0:43', '9/10/2016 10:30'],
      dtype=object)
```

In [ ]:
```
# view how many different types of variables are there

len(data['status_published'].unique())
```

Out[ ]:
```
6913
```

Again, we can see that there are 6913 unique labels in the `status_published` variable. The total number of instances in the dataset is 7050. So, it is also a approximately a unique identifier for each of the instances. Thus this is not a variable that we can use. Hence, I will drop it also.

In [ ]:
```
data.drop(['status_id', 'status_published'], axis=1, inplace=True)
data.head(2)
```

Out[ ]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_
0	video	529	512	262	432	92	3	1	
1	photo	150	0	0	150	0	0	0	

Step-3: Convert categorical variable into integers

In [ ]:
```
le = LabelEncoder()

data['status_type'] = le.fit_transform(data['status_type'])

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   status_type    7050 non-null   int32
 1   num_reactions  7050 non-null   int64
 2   num_comments   7050 non-null   int64
 3   num_shares     7050 non-null   int64
 4   num_likes      7050 non-null   int64
 5   num_loves      7050 non-null   int64
 6   num_wows       7050 non-null   int64
 7   num_hahas      7050 non-null   int64
 8   num_sads       7050 non-null   int64
 9   num_angrys     7050 non-null   int64
dtypes: int32(1), int64(9)
memory usage: 523.4 KB
```

Step-4: Extracting independent and dependent variables

In [ ]:
```
X = data
y = data['status_type']
```

Step-5: Feature Scaling

In [ ]:
```
cols = X.columns
ms = MinMaxScaler()

X = ms.fit_transform(X)
X = pd.DataFrame(X, columns=[cols])
X.head(3)
```

Out[ ]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_
0	1.000000	0.112314	0.024393	0.076519	0.091720	0.140030	0.010791	0.006369	0.01
1	0.333333	0.031847	0.000000	0.000000	0.031847	0.000000	0.000000	0.000000	0.00
2	1.000000	0.048195	0.011243	0.016647	0.043312	0.031963	0.003597	0.006369	0.00

Step-6: KMeans Model with 4 clusters

In [ ]:
```
classifier = KMeans(n_clusters=4, random_state=0)
classifier.fit(X)
```

Out[ ]:
```
KMeans(n_clusters=4, random_state=0)
```

Step-7: Predicting the Test Result

In [ ]:
```
y_pred = classifier.predict(X)
y_pred
```

Out[ ]:
```
array([0, 1, 0, ..., 1, 1, 1])
```

Step-8: Test accuracy of the result

In [ ]:
```
labels = classifier.labels_

correct_labels = sum(y == labels)
print("  Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
  Result: 4340 out of 7050 samples were correctly labeled.
Accuracy score: 0.62
```

Step-9: Finding the Accuracy of the model

In [ ]:
```
accuracy = correct_labels/float(y.size)
print("Accuracy:", accuracy * 100, "%")
```

```
Accuracy: 61.56028368794326 %
```

Hence,the accuracy of the model is approximate 62%.