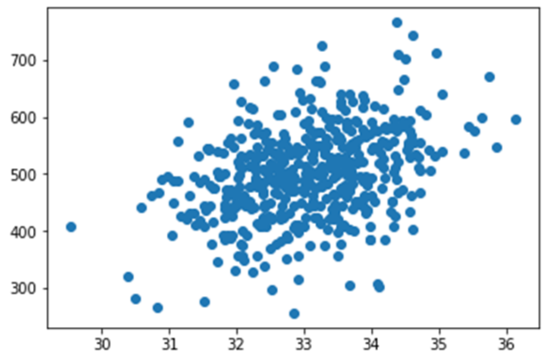
Q1. Write a python program to prepare scatter plot (Use Iris Datasets).

```
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as pt

df=pd.read_csv("data.csv")

df.head()
df.tail()
df.describe()
sb.pairplot(df)
pt.scatter(df['Avg. Session Length'],df['Yearly Amount Spent'])
```



Q2. Write a python program to find all null values in a given data set and remove them.

```
import pandas as pd
df=pd.read_csv('titanic_train.csv',sep=",",index_col=0)
df.count()
print("Original 'input.csv' CSV Data: \n")
print(df)
df.drop('Age', inplace=True, axis=1)
print(df)
df.drop('Cabin', inplace=True, axis=1)
print(df)
df.drop('Embarked', inplace=True, axis=1)
print(df)
df.count()
```

FINAL OP Here:

OUTPUT:

```
Survived        891
Pclass    891
Name            891
```

```
Sex	891
Age	714
SibSp	891
Parch	891
Ticket	891
Fare	891
Cabin	204
Embarked		889
dtype: int64
```

Q3. Write a python program the categorical values in numeric format for a given dataset.

```
 import pandas as pd
df=pd.read_csv("PlayTennis.csv")
df.head()
oMap={"Sunny":1,"Overcast":2,"Rain":3}
df['Outlook']=df['Outlook'].map(oMap)
tMap={"Hot":1,"Mild":2,"Cool":3}
df['Temperature']=df['Temperature'].map(tMap)
hMap={"High":1,"Normal":2}
df['Humidity']=df['Humidity'].map(hMap)
wMap={"Weak":1,"Strong":2}
df['Wind']=df['Wind'].map(wMap)
ptMap={"Yes":1,"No":2}
df['Play Tennis']=df['Play Tennis'].map(ptMap)
df.head()
```

OUTPUT:

```
OutlookTemperature		Humidity  Wind		Play Tennis

0	1	1	1	1	2

1	1	1	1	2	2

2	2	1	1	1	1

3	3	2	1	1	1

4	3	3	2	1	1
```

Q4. Write a python program to implement simple Linear Regression for predicting house price.

```
import pandas as pd
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
df=pd.read_csv("data.csv")
 df.head()
df.tail()
df.describe()
 xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=42)
 xtrain.describe()
xtest.describe()
model=LinearRegression()
model.fit(xtrain,ytrain)
```

OP:

```
 LinearRegression

 LinearRegression()
```

```
 ypred=model.predict(xtest)

 ypred

ytest

from sklearn import metrics

mse=metrics.mean_squared_error(ypred,ytest)

mse
```

OUTPUT:

```
113.35446845209206
```

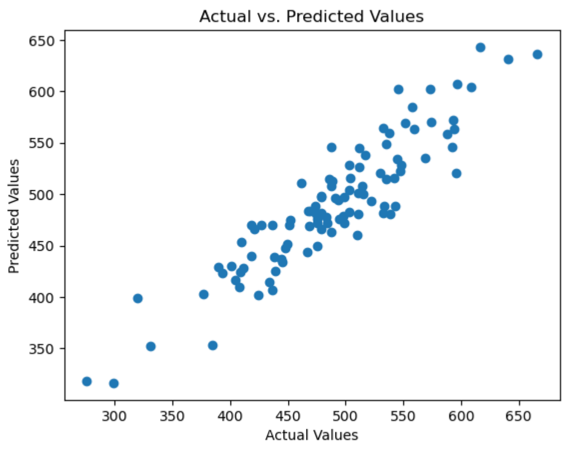
Q5.Write a python program to implement multiple Linear Regression for a given dataset.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error, r2_score


data = pd.read_csv('Ecommerce Customers.csv')


X = data[['Time on Website', 'Time on App', 'Length of Membership']]
```

```
y = data['Yearly Amount Spent']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

coefficients = model.coef_
intercept = model.intercept_
print("Coefficients:", coefficients)
print("Intercept:", intercept)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared:", r2)

plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.show()
```

Output:

```
Coefficients: [ 0.53288794 38.00802494 62.7706933 ]
Intercept: -199.99563527969264
Mean Squared Error: 807.1362660554383
R-squared: 0.8369991649814932
```



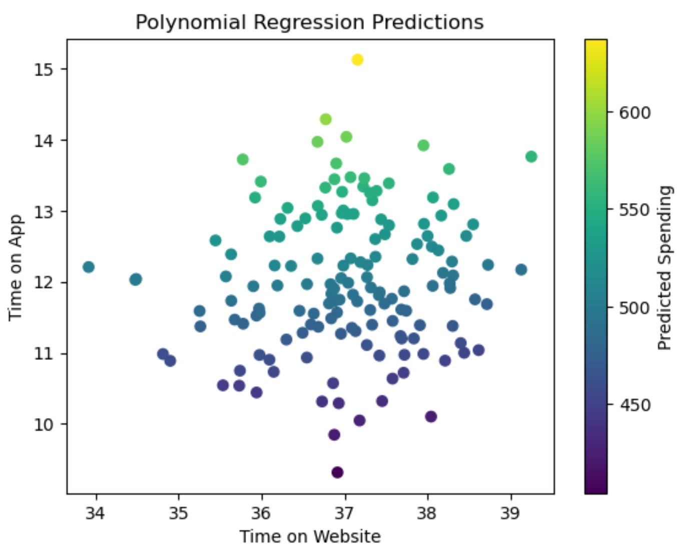
Q6. Write a python program to implement Polynomial Regression for given dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

data = pd.read_csv('Ecommerce Customers.csv')
X = data[['Time on Website', 'Time on App']]
y = data['Yearly Amount Spent']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

degree = 2
poly_features = PolynomialFeatures(degree=degree)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)

model = LinearRegression()
model.fit(X_train_poly, y_train)

y_pred = model.predict(X_test_poly)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")

if X_train_poly.shape[1] > 3:
   plt.scatter(X_test['Time on Website'], X_test['Time on App'], c=y_pred, cmap='viridis', 
label='Predicted Spending')
   plt.xlabel('Time on Website')
   plt.ylabel('Time on App')
```

```
    plt.colorbar(label='Predicted Spending')
    plt.title('Polynomial Regression Predictions')
    plt.show()
```

Output:

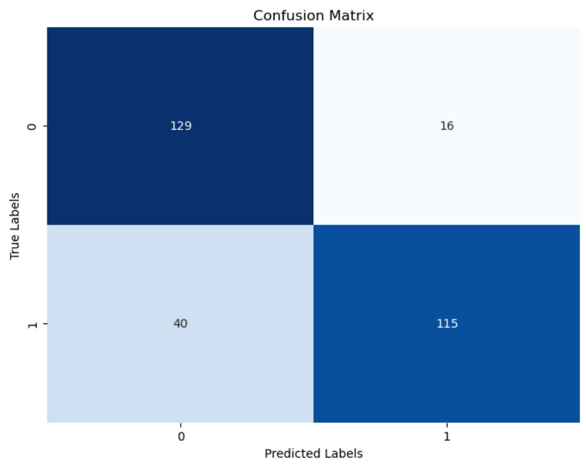
```
Mean Squared Error: 4732.0283
```



Q7. Write a python program to Implement Naïve Bayes.

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Generate a synthetic dataset for binary classification
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Gaussian Naïve Bayes classifier
nb_classifier = GaussianNB()

# Train the classifier on the training data
nb_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = nb_classifier.predict(X_test)

# Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

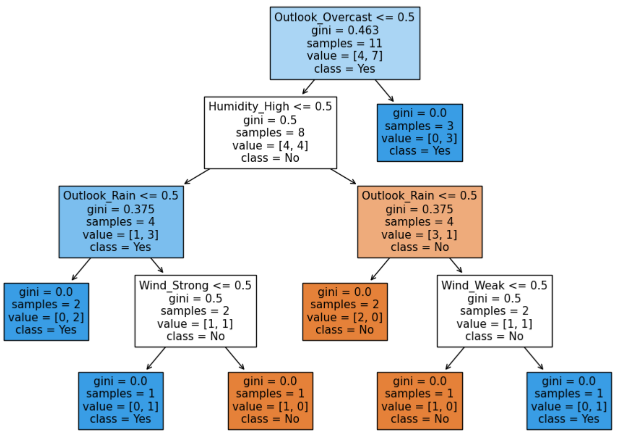


Q8. Write a python program to Implement Decision Tree whether or not to play tennis.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv("PlayTennis.csv")  # Replace "PlayTennis.csv" with your dataset file path

# Check the column names in your dataset
print(data.columns)

# For example, if the column name is 'Decision', use: data['Decision']
if 'Play Tennis' not in data.columns:
   print("Column 'Play Tennis' not found in the dataset. Please check the column name.")
   exit()

# Encode categorical features into numerical values
data = pd.get_dummies(data, columns=['Outlook', 'Temperature', 'Humidity', 'Wind'])
```

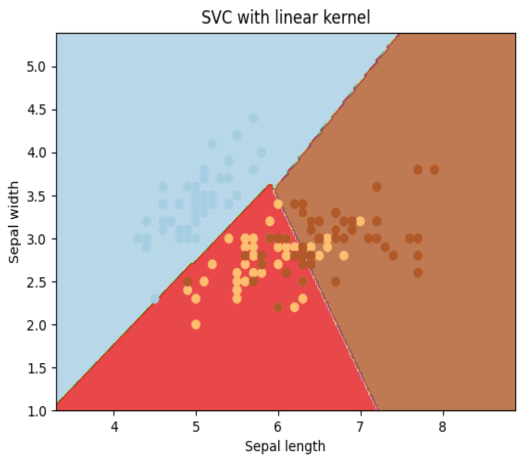
```
# Split the dataset into features and target
X = data.drop('Play Tennis', axis=1)
y = data['Play Tennis']

# 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, random_state=42)

# Create a Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Train the classifier on the training data
dt_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = dt_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(dt_classifier, feature_names=X.columns.tolist(), class_names=["No", "Yes"],
filled=True)
plt.show()
```

```
Index(['Outlook', 'Temperature', 'Humidity', 'Wind', 'Play Tennis'], dtype='object')
Accuracy: 100.00%
```


Outlook_Overcast <= 0.5 gini = 0.463 samples = 11 value = [4, 7] class = Yes
Humidity_High <= 0.5 gini = 0.5 samples = 8 value = [4, 4] class = No
gini = 0.0 samples = 3 value = [0, 3] class = Yes
Outlook_Rain <= 0.5 gini = 0.375 samples = 4 value = [1, 3] class = Yes
Outlook_Rain <= 0.5 gini = 0.375 samples = 4 value = [3, 1] class = No
gini = 0.0 samples = 2 value = [0, 2] class = Yes
Wind_Strong <= 0.5 gini = 0.5 samples = 2 value = [1, 1] class = No
gini = 0.0 samples = 2 value = [2, 0] class = No
Wind_Weak <= 0.5 gini = 0.5 samples = 2 value = [1, 1] class = No
gini = 0.0 samples = 1 value = [0, 1] class = Yes
gini = 0.0 samples = 1 value = [1, 0] class = No
gini = 0.0 samples = 1 value = [1, 0] class = No
gini = 0.0 samples = 1 value = [0, 1] class = Yes

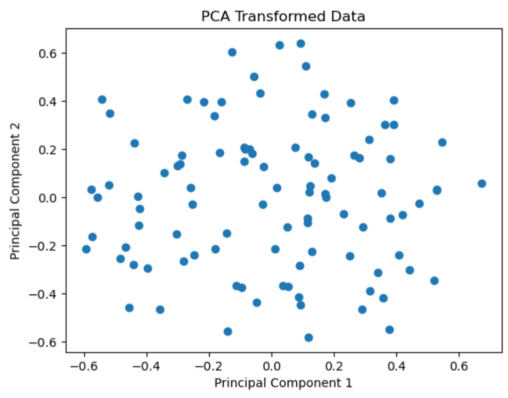
Q9. Write a python program to implement linear SVM.

```
# Import the Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

# Import some Data from the iris Data Set
iris = datasets.load_iris()

# Take only the first two features of Data.
# To avoid the slicing, Two-Dim Dataset can be used

X = iris.data[:, :2]
y = iris.target

# C is the SVM regularization parameter
C = 1.0

# Create an Instance of SVM and Fit out the data.
# Data is not scaled so as to be able to plot the support vectors
svc = svm.SVC(kernel ='linear', C = 1).fit(X, y)

# create a mesh to plot
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
        np.arange(y_min, y_max, h))

# Plot the data for Proper Visual Representation
plt.subplot(1, 1, 1)

# Predict the result by giving Data to the model
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap = plt.cm.Paired, alpha = 0.8)

plt.scatter(X[:, 0], X[:, 1], c = y, cmap = plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')

# Output the Plot
plt.show()
```



Q10. Write a python program to transfer data with Principal Component Analysis(PCA)

```
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Create a sample dataset (you can replace this with your own data)
np.random.seed(0)
data = np.random.rand(100, 3)

# Instantiate a PCA object with 2 components for 2D visualization
n_components = 2
pca = PCA(n_components=n_components)

# Fit and transform the data to the lower-dimensional space
transformed_data = pca.fit_transform(data)

# Create a scatter plot of the transformed data
plt.scatter(transformed_data[:, 0], transformed_data[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Transformed Data')
plt.show()
```



Q11. Write a python program to implement k-nearest Neighbours ML algorithm to build prediction model (Use Forge/Iris/housing Dataset).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset (you can replace this with other datasets)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a KNN classifier with a specified number of neighbors (k)
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Train the classifier on the training data
knn_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = knn_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

OutPut:

```
Accuracy: 100.00%
```