TABLE OF CONTENTS

S. No.	Experiment	Date Assigned	Date Submitted	Signature
1.	Write a Python program for the computation of central tendency measures and measures of dispersion.	05/01/2023	12/01/2023	
2.	Write a Python program to a. Study basic Python modules such as statistics, math, NumPy, and SciPy. b. Study of Python machine learning modules such as pandas and Matplotlib.	12/01/2023	02/02/2023	
3.	Write a Python program for exploration and implementation of pandas library and DataFrames.	12/01/2023	02/02/2023	
4.	Write a Python program to study and implement various data cleaning operations on the dataset.	02/02/2023	09/02/2023	
5.	Write a Python program to study and implement various Statistical Analysis methods and Basic Plotting operations on the dataset.	09/02/2023	16/02/2023	
6.	Write a Python program to predict using: a. Simple Linear Regression on the dataset. b. Gradient Descent on the dataset.	16/02/2023	23/02/2023	
7.	Write a Python program to perform Prediction using Logistic Regression on the dataset.	23/02/2023	02/03/2023	
8.	Write a Python program to perform Prediction using Naïve Bayes on the dataset.	02/03/2023	09/03/2023	
9.	Write a Python program to perform Prediction using Decision Tree and compare results with Naïve Bayes on the dataset.	09/03/2023	23/03/2023	
10.	Write a Python program to perform clustering using K-Means Clustering and compare results for different values of k on the dataset.	09/03/2023	23/03/2023	
11.	Open-ended Experiment Write a Python program to perform the implementation of Principle Component Analysis for feature selection on the dataset.	23/03/2023	06/04/2023	
12.	Open-ended Experiment Write a Python program to build an ML model using Support Vector Machine for classification.	06/04/2023	13/04/2023	

EXPERIMENT 1

Q1. a. To study basic Python modules such as statistics , math , numpy and scipy

1. math Module:

This module provides access to the mathematical functions defined by the C standard.
These functions cannot be used with complex numbers, use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place. Except when explicitly noted otherwise, all return values are floats.

In [1]:
```
import math
```

In [2]:
```
print("Constants in math module:")
print(f"math.pi: {math.pi}")
print(f"math.e: {math.e}")
print(f"math.tau: {math.tau}")
print(f"math.inf: {math.inf}")
print(f"math.nan: {math.nan}")
```

```
Constants in math module:
math.pi: 3.141592653589793
math.e: 2.718281828459045
math.tau: 6.283185307179586
math.inf: inf
math.nan: nan
```

In [3]:
```
# Using the sqrt() function
num1 = 16
num2 = 25
print("Square root of", num1, "is: ", math.sqrt(num1))

# Using the log() function
num3 = 100
print("Logarithm of", num3, "is: ", math.log(num3))

# Using the log10() function
print("The logarithm (base 10) of", num3, "is: ", math.log10(num3))

# Using the exp() function
print("e^2 is: ", math.exp(2))

# Using the sin(), cos() and tan() functions
angle = math.pi / 4
print("The sine of", angle, "is: ", math.sin(angle))
print("The cosine of", angle, "is: ", math.cos(angle))
print("The tangent of", angle, "is: ", math.tan(angle))

# Using the asin(), acos() and atan() functions
print("The arcsine of", math.sin(angle), "is: ", math.asin(math.sin(angle)))
print("The arccosine of", math.cos(angle), "is: ", math.acos(math.cos(angle)))
print("The arctangent of", math.tan(angle), "is: ", math.atan(math.tan(angle)))

# Using the degrees() and radians() functions
degrees = 90
print(degrees, "degrees is equal to ", math.radians(degrees), "radians.")
radians = math.pi / 2
print(radians, "radians is equal to ", math.degrees(radians), "degrees.")

# Using the floor() and ceil() functions
num4 = 12.345
print("The floor value of", num4, "is: ", math.floor(num4))
print("The ceil value of", num4, "is: ", math.ceil(num4))

# Using the fabs() function
```

```
num5 = -12.345
print("The absolute value of", num5, "is: ", math.fabs(num5))

# Using the pow() function
print(num1, "raised to the power of 3 is: ", math.pow(num1,3))

# Using the factorial() function
num6 = 5
print(num6," factorial is: ", math.factorial(num6))

# Using the modf() function
print(num4, "in fractional and integer parts: ", math.modf(num4))

# Using the fmod() function
print(num1, "modulo", num2, "is: ", math.fmod(num1,num2))

# Using the gcd() function
print("The greatest common divisor of", num1, "and", num2, "is: ", math.gcd(num1,num2))

# Using the trunc() function
print("The truncated value of", num4, "is: ", math.trunc(num4))
```

```
Square root of 16 is:  4.0
Logarithm of 100 is:  4.605170185988092
The logarithm (base 10) of 100 is:  2.0
e^2 is:  7.38905609893065
The sine of 0.7853981633974483 is:  0.7071067811865476
The cosine of 0.7853981633974483 is:  0.7071067811865476
The tangent of 0.7853981633974483 is:  0.9999999999999999
The arcsine of 0.7071067811865476 is:  0.7853981633974484
The arccosine of 0.7071067811865476 is:  0.7853981633974483
The arctangent of 0.9999999999999999 is:  0.7853981633974483
90 degrees is equal to  1.5707963267948966 radians.
1.5707963267948966 radians is equal to  90.0 degrees.
The floor value of 12.345 is:  12
The ceil value of 12.345 is:  13
The absolute value of -12.345 is:  12.345
16 raised to the power of 3 is:  4096.0
5  factorial is:  120
12.345 in fractional and integer parts:  (0.34500000000000064, 12.0)
16 modulo 25 is:  16.0
The greatest common divisor of 16 and 25 is:  1
The truncated value of 12.345 is:  12
```

2. statistics Module:

This module provides functions for calculating mathematical statistics of numeric (Real-valued) data.

This module is not intended to be a competitor to third-party libraries such as NumPy, SciPy, or proprietary full-featured statistics packages aimed at professional statisticians such as Minitab, SAS and Matlab. It is aimed at the level of graphing and scientific calculators.

In [4]:

```
import statistics as stats

# Using the mean() function
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Mean of the data set is: ", stats.mean(data))

# Using the median() function
print("Median of the data set is: ", stats.median(data))

# Using the mode() function
data2 = [1, 2, 2, 3, 4, 4, 4, 5, 5, 6, 7, 8, 8, 9, 9, 10]
print("Mode of the data set is: ", stats.mode(data2))

# Using the variance() function
print("Variance of the data set is: ", stats.variance(data))

# Using the stdev() function
print("Standard deviation of the data set is: ", stats.stdev(data))

# Using the pstdev() function
print("Population standard deviation of the data set is: ", stats.pstdev(data))

# Using the pvariance() function
print("Population variance of the data set is: ", stats.pvariance(data))
```

```
# Using the harmonic_mean() function
print("Harmonic mean of the data set is: ", stats.harmonic_mean(data))
```

```
Mean of the data set is:  5.5
Median of the data set is:  5.5
Mode of the data set is:  4
Variance of the data set is:  9.166666666666666
Standard deviation of the data set is:  3.0276503540974917
Population standard deviation of the data set is:  2.8722813232690143
Population variance of the data set is:  8.25
Harmonic mean of the data set is:  3.414171521474055
```

3. numpy Module:

NumPy is a library for the Python programming langugage, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. The core functionality of NumPy is its ndarray , for -dimensional array, data structure. These arrays are strided views on memory. In contrast to Python's built-in list data structure, these arrays are homogenously typed; all elements of a single array must be of the same type.

In [5]:

```
import numpy as np

# Create a 1-D array
arr = np.array([1, 2, 3, 4, 5])

# Using the min() and max() functions
print("Minimum value in the array: ", np.min(arr))
print("Maximum value in the array: ", np.max(arr))

# Using the mean() and sum() functions
print("Mean of the array: ", np.mean(arr))
print("Sum of the array: ", np.sum(arr))

# Using the sort() function
arr = np.sort(arr)
print("Sorted array: ", arr)

# Create a 2-D array
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Using the reshape() function
arr2d = np.reshape(arr2d, (9, 1))
print("Reshaped array:\n", arr2d)

# Using the transpose() function
print("Transpose of the array:\n", np.transpose(arr2d))

# Using the trace() function
print("Trace of the array: ", np.trace(arr2d))

# Using the linspace() function
linspace = np.linspace(1,10,5)
print("linspace function : ", linspace)

# Using the logspace() function
logspace = np.logspace(1,10,5)
print("logspace function : ", logspace)

# Using the eye() function
eye = np.eye(3)
print("Identity matrix using eye function: \n", eye)

# Using the ones() and zeros() functions
ones = np.ones((3,3))
print("Ones matrix : \n",ones)
zeros = np.zeros((3,3))
print("Zeros matrix : \n",zeros)

# Creating a Random Array
rand = np.random.rand(3,3)
print("3x3 random matrix: \n", rand)
```

```
# Using the dot() function
dot_product = np.dot(rand, eye)
print("Dot product of the random matrix and identity matrix:\n", dot_product)

# Using the diag() function
diag = np.diag(dot_product)
print("Diagonal elements of dot product matrix: ", diag)

# Using the unique() function
arr = np.array([1, 2, 2, 3, 4, 4, 5, 5, 6])
unique = np.unique(arr)
print("Unique elements of the array: ", unique)
```

```
Minimum value in the array:  1
Maximum value in the array:  5
Mean of the array:  3.0
Sum of the array:  15
Sorted array:  [1 2 3 4 5]
Reshaped array:
 [[1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
Transpose of the array:
 [[1 2 3 4 5 6 7 8 9]]
Trace of the array:  1
linspace function :  [ 1.    3.25  5.5   7.75 10.  ]
logspace function :  [1.00000000e+01 1.77827941e+03 3.16227766e+05 5.62341325e+07
 1.00000000e+10]
Identity matrix using eye function:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Ones matrix :
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
Zeros matrix :
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
3x3 random matrix:
 [[0.71814251 0.80551707 0.81264586]
 [0.23095896 0.32860839 0.36146529]
 [0.01532045 0.12602035 0.46349364]]
Dot product of the random matrix and identity matrix:
 [[0.71814251 0.80551707 0.81264586]
 [0.23095896 0.32860839 0.36146529]
 [0.01532045 0.12602035 0.46349364]]
Diagonal elements of dot product matrix:  [0.71814251 0.32860839 0.46349364]
Unique elements of the array:  [1 2 3 4 5 6]
```

scipy Module:

SciPy is a free and open-source Python library used for scientific computing and technical computing. It contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

The SciPy package is at the core of Python's computing capabilities. The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for linear algebra, Fourier transforms, and random number generation, but not with the generality of the equivalent functions in SciPy. NumPy can also be used as an efficient multidimensional container of data with arbitrary datatypes. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. Older versions of SciPy used Numeric as an array type, which is now deprecated in favor of the newer NumPy array code.

In [6]:

```
import numpy as np
from scipy import optimize, integrate, interpolate, stats, special, signal

# Defining the function
def function(x):
    return x**2 + 10*np.sin(x)
```

```
# Using the minimize() function
result = optimize.minimize(function, x0=3)
print("Minimum value of the function: ", result.fun)
print("Minimum point of the function: ", result.x)

# Using the integrate() function
result, _ = integrate.fixed_quad(function, 0, 5)
print("Integral of the function: ", result)

# Using the interpolate() function
x = np.linspace(0, 10, 10)
y = function(x)
interp = interpolate.interp1d(x, y, kind='cubic')
print("Interpolated value of the function: ", interp(5.5))

# Using the norm() function
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Probability density of the data set under normal distribution: ", stats.norm.pdf(data))

# Using the erf() function
x = np.linspace(-3,3,100)
print("Values of error function: ", special.erf(x))

# Using the convolve() function
data1 = [1, 2, 3]
data2 = [2, 3, 4]
conv = signal.convolve(data1, data2)
print("Convolution of data sets: ", conv)

# Using the fft() function
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
fft_data = np.fft.fft(data)
print("Fast Fourier Transform of the data set: ", fft_data)
```

```
Minimum value of the function:  8.315585579477458
Minimum point of the function:  [3.83746709]
Integral of the function:  48.83014606536962
Interpolated value of the function:  23.20014037630698
Probability density of the data set under normal distribution:  [2.41970725e-01 5.39909665e-02 4.43184
841e-03 1.33830226e-04
 1.48671951e-06 6.07588285e-09 9.13472041e-12 5.05227108e-15
 1.02797736e-18 7.69459863e-23]
Values of error function:  [-0.99997791 -0.99996774 -0.99995323 -0.99993266 -0.99990373 -0.99986331
 -0.99980729 -0.99973018 -0.99962483 -0.99948196 -0.99928962 -0.99903257
 -0.99869156 -0.99824246 -0.99765536 -0.99689344 -0.99591189 -0.99465667
 -0.99306318 -0.99105509 -0.98854301 -0.98542346 -0.98157787 -0.97687194
 -0.97115532 -0.96426173 -0.95600967 -0.94620369 -0.9346364  -0.92109126
 -0.90534611 -0.8871775  -0.86636577 -0.84270079 -0.81598822 -0.78605615
 -0.75276189 -0.71599866 -0.67570198 -0.63185531 -0.58449492 -0.5337134
 -0.47966192 -0.42255075 -0.36264811 -0.30027714 -0.23581117 -0.16966722
 -0.10229803 -0.03418284  0.03418284  0.10229803  0.16966722  0.23581117
  0.30027714  0.36264811  0.42255075  0.47966192  0.5337134   0.58449492
  0.63185531  0.67570198  0.71599866  0.75276189  0.78605615  0.81598822
  0.84270079  0.86636577  0.8871775   0.90534611  0.92109126  0.9346364
  0.94620369  0.95600967  0.96426173  0.97115532  0.97687194  0.98157787
  0.98542346  0.98854301  0.99105509  0.99306318  0.99465667  0.99591189
  0.99689344  0.99765536  0.99824246  0.99869156  0.99903257  0.99928962
  0.99948196  0.99962483  0.99973018  0.99980729  0.99986331  0.99990373
  0.99993266  0.99995323  0.99996774  0.99997791]
Convolution of data sets:  [ 2  7 16 17 12]
Fast Fourier Transform of the data set:  [55.+0.00000000e+00j -5.+1.53884177e+01j -5.+6.88190960e+00j
 -5.+3.63271264e+00j -5.+1.62459848e+00j -5.-1.33226763e-15j
 -5.-1.62459848e+00j -5.-3.63271264e+00j -5.-6.88190960e+00j
 -5.-1.53884177e+01j]
```

In [7]:

```
import scipy.constants
print("SciPy constants")
print(f"Speed of light: {scipy.constants.speed_of_light}")
print(f"Speed of sound: {scipy.constants.speed_of_sound}")
print(f"Gravitational constant: {scipy.constants.gravitational_constant}")
print(f"Acceleration due to gravity: {scipy.constants.g}")
print(f"Elementary charge: {scipy.constants.elementary_charge}")
print(f"Molar gas constant: {scipy.constants.gas_constant}")
print(f"Fine-structure constant: {scipy.constants.alpha}")
print(f"Avogadro's constant: {scipy.constants.Avogadro}")
print(f"Boltzmann's constant: {scipy.constants.Boltzmann}")
```

```
print(f"Stefan-Boltzmann's constant: {scipy.constants.Stefan_Boltzmann}")
print(f"Electron mass: {scipy.constants.electron_mass}")
print(f"Proton mass: {scipy.constants.proton_mass}")
print(f"Neutron mass: {scipy.constants.neutron_mass}")
print(f"Plank's constant: {scipy.constants.Planck}")
```

```
SciPy constants
Speed of light: 299792458.0
Speed of sound: 340.5
Gravitational constant: 6.6743e-11
Acceleration due to gravity: 9.80665
Elementary charge: 1.602176634e-19
Molar gas constant: 8.314462618
Fine-structure constant: 0.0072973525693
Avogadro's constant: 6.02214076e+23
Boltzmann's constant: 1.380649e-23
Stefan-Boltzmann's constant: 5.670374419e-08
Electron mass: 9.1093837015e-31
Proton mass: 1.67262192369e-27
Neutron mass: 1.67492749804e-27
Plank's constant: 6.62607015e-34
```

Q1. b. To study Python modules such as pandas and matplotlib

pandas Module:

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. This library is highly optimized for performance, with critical code paths written in Cython or C.

pandas is mainly used for data analysis and associated manipulation of tabular data in DataFrames. Pandas allows importing of data. from various file formats such as comma-separated values, JSON, Parquet, SQL database tables or queries, and Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features. The development of pandas introduced into Python many comparable features of working with DataFrames that were established in the R programming language. The pandas library is built upon another library NumPy, which is oriented to efficiently working with arrays instead of the features of working on DataFrames.

In [8]:

```
import pandas as pd

# Creating a DataFrame
data = {'Name':['John', 'Mike', 'Sarah'],
        'Age':[32, 45, 28],
        'Gender':['M', 'M', 'F']}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)

# Using the head() function
print("\nDataFrame with first 2 rows:")
print(df.head(2))

# Using the tail() function
print("\nDataFrame with last 2 rows:")
print(df.tail(2))

# Using the shape() function
print("\nShape of the DataFrame:", df.shape)

# Using the describe() function
print("\nStatistical Description of the DataFrame:")
print(df.describe())

# Using the sort_values() function
print("\nDataFrame sorted by Age:")
print(df.sort_values(by='Age'))

# Using the groupby() function
print("\nDataFrame grouped by Gender:")
print(df.groupby(by='Gender').mean())

# Using the set_index() function
df = df.set_index('Name')
print("DataFrame with Name column set as index:")
```

```python
print(df)

# Using the rename() function
df = df.rename(columns={'Age': 'Age(Years)'})
print("\nDataFrame with column names changed:")
print(df)

# Using the drop() function
df = df.drop('Gender', axis=1)
print("\nDataFrame after dropping Gender column:")
print(df)

# Using the fillna() function
df = df.fillna(0)
print("\nDataFrame after filling missing values with 0:")
print(df)
```

```
Original DataFrame:
    Name  Age Gender
0   John   32      M
1   Mike   45      M
2  Sarah   28      F

DataFrame with first 2 rows:
   Name  Age Gender
0  John   32      M
1  Mike   45      M

DataFrame with last 2 rows:
    Name  Age Gender
1   Mike   45      M
2  Sarah   28      F

Shape of the DataFrame: (3, 3)

Statistical Description of the DataFrame:
             Age
count   3.000000
mean   35.000000
std     8.888194
min    28.000000
25%    30.000000
50%    32.000000
75%    38.500000
max    45.000000

DataFrame sorted by Age:
    Name  Age Gender
2  Sarah   28      F
0   John   32      M
1   Mike   45      M

DataFrame grouped by Gender:
         Age
Gender
F       28.0
M       38.5
DataFrame with Name column set as index:
       Age Gender
Name
John    32      M
Mike    45      M
Sarah   28      F

DataFrame with column names changed:
       Age(Years) Gender
Name
John           32      M
Mike           45      M
Sarah          28      F

DataFrame after dropping Gender column:
       Age(Years)
Name
John           32
Mike           45
Sarah          28

DataFrame after filling missing values with 0:
       Age(Years)
```

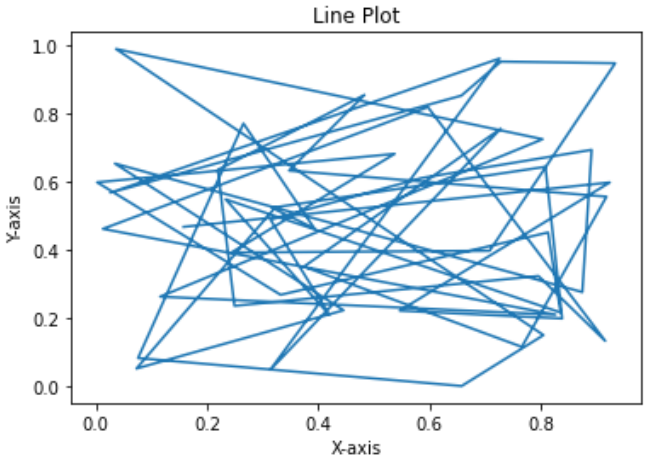
```
Name
John          32
Mike          45
Sarah         28
```

matplotlib module:

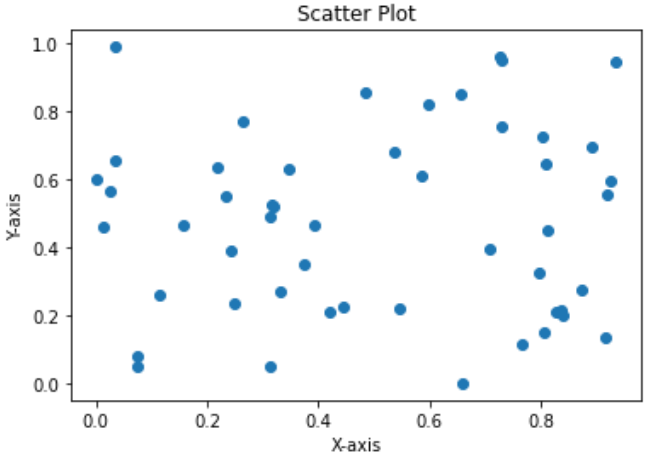
Matplotlib is a plottling library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt

# Create data
x = np.random.rand(50)
y = np.random.rand(50)
```

In [10]:
```
# Create a line plot
plt.plot(x, y)
plt.title('Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

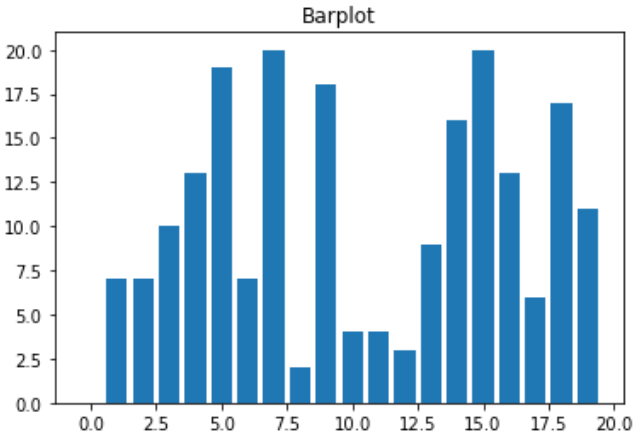


In [11]:
```
# Create a scatter plot
plt.scatter(x, y)
plt.title('Scatter Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

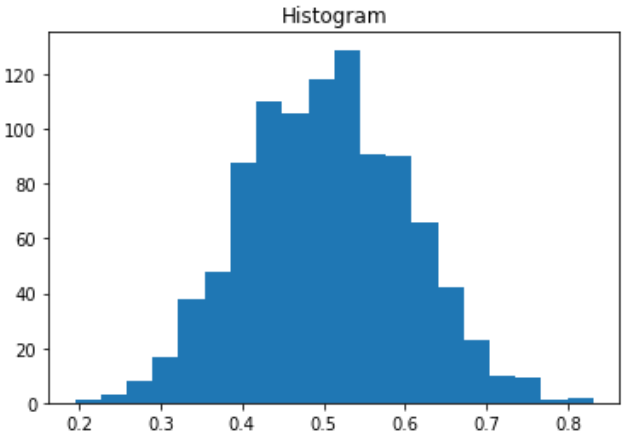


In [13]:
```
import random
```

In [14]:
```
# Create a bar plot
x = [i for i in range(20)]
y = [random.randint(0, 20) for _ in range(20)]
plt.title(label="Barplot")
plt.bar(x=x, height=y)
plt.show()
```



In [15]:
```
y = np.random.normal(loc=0.5, scale=0.1, size=1000)
plt.title(label="Histogram")
plt.hist(x=y, bins=20)
plt.show()
```



EXPERIMENT 2

Q1. WAP to implement measures of central tendency and measures of dispersion in python.

MEASURES OF CENTRAL TENDENCY

In [1]:
```
import random
data = [random.randint(a=0, b=100) for _ in range(20)]
print(data)
```

```
[100, 72, 81, 4, 45, 53, 26, 90, 71, 43, 58, 78, 25, 74, 67, 39, 22, 59, 13, 83]
```

1. MEAN

In [2]:
```
mean_val = sum(data) / len(data)
print("Mean: ", mean_val)
```

```
Mean:  55.15
```

1. MEDIAN

In [3]:
```
data.sort()
mid = len(data)//2
if len(data) % 2 == 0:
    median_val = (data[mid - 1] + data[mid]) / 2
else:
    median_val = data[mid]
print("Median: ", median_val)
```

```
Median:  58.5
```

1. MODE

In [4]:
```
from collections import Counter
mode_val = Counter(data).most_common(1)[0][0]
print("Mode: ", mode_val)
```

```
Mode:  4
```

MESAURES OF DISPERSION

1. VARIANCE

In [5]:
```
def variance(data):
  n = len(data)
  mean = sum(data) / n
  dev = [(x - mean) ** 2 for x in data]
  var = sum(dev) / n
  return var
```

In [6]:
```
var = variance(data)
print(var)
```

```
700.6274999999999
```

1. STANDARD DEVIATION

In [7]:
```
import math
std = math.sqrt(var)
print(std)
```

```
26.469369089572194
```

EXPERIMENT 3

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
# a. Load this file into a python DataFrame
df = pd.read_csv('student.csv')
```

In [3]:
```
# b. Find the number of rows and columns in it
print('Rows and columns in the dataframe: ' + str(df.shape))
```

```
Rows and columns in the dataframe: (6, 5)
```

In [4]:
```
# c. Print column names
print('Columns in the dataframe: \n' + str(df.columns))
```

```
Columns in the dataframe:
Index(['Name', 'Age', 'Sex', 'Marks', 'Grade'], dtype='object')
```

In [5]:
```
# d. Change column name `Name' with new name `FirstName'
df.rename(columns={'Name': 'FirstName'}, inplace=True)
df.head()
```

Out[5]:

	FirstName	Age	Sex	Marks	Grade
0	Nihar	23	M	23	C
1	Ranjan	34	M	40	B
2	Roy	23	M	47	A
3	Sunita	18	F	49	A
4	Gita	23	F	23	D

In [6]:
```
# e. Print last 5 rows from the bottom
df.tail()
```

Out[6]:

	FirstName	Age	Sex	Marks	Grade
1	Ranjan	34	M	40	B
2	Roy	23	M	47	A
3	Sunita	18	F	49	A
4	Gita	23	F	23	D
5	Sita	34	F	34	C

In [7]:
```
# f. Print the details of student with lowest marks
df[df['Marks'] == min(df['Marks'])]
```

Out[7]:

	FirstName	Age	Sex	Marks	Grade
0	Nihar	23	M	23	C
4	Gita	23	F	23	D

In [8]:
```
# g. Find total marks of all female students
l1 = df[df['Sex'] == 'F']['Marks']
l1 = l1.tolist()

sum = 0
```

```
for i in l1:
    sum = sum + i
print('Total marks of all female students = ' + str(sum))
```

```
Total marks of all female students = 106
```

In [9]:
```
# h. List names of all the male students
df[df['Sex'] == 'M']['FirstName']
```

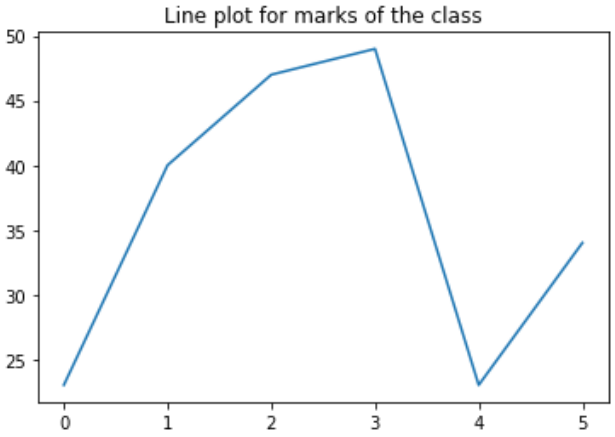
Out[9]:
```
0     Nihar
1    Ranjan
2       Roy
Name: FirstName, dtype: object
```

In [10]:
```
# i. Find mean age of the class
print('Mean age of the class = ' + str(np.mean(df.Age)))
```

```
Mean age of the class = 25.833333333333332
```

In [11]:
```
# j. Line plot marks of the class
plt.plot(df.Marks)
plt.title('Line plot for marks of the class')
```

Out[11]:
```
Text(0.5, 1.0, 'Line plot for marks of the class')
```



In [12]:
```
# k. Find the index of record of oldest student in the class
list(df[df['Age'] == max(df['Age'])].index.values)
```

Out[12]:
```
[1, 5]
```

In [13]:
```
# l. Sort and print the data on the basis of Name followed by Age
df.sort_values(by=['FirstName','Age'])
```

Out[13]:

	FirstName	Age	Sex	Marks	Grade
4	Gita	23	F	23	D
0	Nihar	23	M	23	C
1	Ranjan	34	M	40	B
2	Roy	23	M	47	A
5	Sita	34	F	34	C
3	Sunita	18	F	49	A

In [14]:
```
# m. Change the name 'Nihar' to 'Jason Bourne' in name column of the DataFrame
ind = df[df['FirstName'] == 'Nihar'].index.values
df.at[ind[0],'FirstName'] = 'Jason Bourne'
df.head()
```

Out[14]:

	FirstName	Age	Sex	Marks	Grade
0	Jason Bourne	23	M	23	C
1	Ranjan	34	M	40	B
2	Roy	23	M	47	A
3	Sunita	18	F	49	A
4	Gita	23	F	23	D

In [15]:
```
# n. Change and print order of the columns (Name, Sex, Age, Marks, Grade)
df.iloc[:,[0, 2, 1, 3, 4]]
```

Out[15]:

	FirstName	Sex	Age	Marks	Grade
0	Jason Bourne	M	23	23	C
1	Ranjan	M	34	40	B
2	Roy	M	23	47	A
3	Sunita	F	18	49	A
4	Gita	F	23	23	D
5	Sita	F	34	34	C

In [16]:
```
# o. Count and print number of students sex wise and display result with suitable column headers
df.groupby('Sex').groups
gr = df.groupby('Sex')

for name,group in gr:
   print(name)
   print(group)
```

```
F
  FirstName  Age Sex  Marks Grade
3    Sunita   18   F     49     A
4      Gita   23   F     23     D
5      Sita   34   F     34     C
M
      FirstName  Age Sex  Marks Grade
0  Jason Bourne   23   M     23     C
1        Ranjan   34   M     40     B
2           Roy   23   M     47     A
```

In [17]:
```
# p. Delete and print row where age=18
ind2 = df[df['Age'] == 18].index
df.drop(df.index[ind2], inplace=True)
df
```

Out[17]:

	FirstName	Age	Sex	Marks	Grade
0	Jason Bourne	23	M	23	C
1	Ranjan	34	M	40	B
2	Roy	23	M	47	A
4	Gita	23	F	23	D
5	Sita	34	F	34	C

In [18]:
```
# q. Print the data types of individual columns of the data frame
df.dtypes
```

Out[18]:
```
FirstName    object
Age           int64
Sex          object
Marks         int64
```

```
Grade        object
dtype: object
```

In [19]:
```
# r. Convert and print the datatype of a given column Age (int to float)
df['Age'] = df['Age'].astype(float)
df.head()
```

Out[19]:

	FirstName	Age	Sex	Marks	Grade
0	Jason Bourne	23.0	M	23	C
1	Ranjan	34.0	M	40	B
2	Roy	23.0	M	47	A
4	Gita	23.0	F	23	D
5	Sita	34.0	F	34	C

In [20]:
```
# s. Create a new column named “UpdatedMarks” which as 5.5% more marks than the existing Marks column
df['UpdatedMarks'] = df['Marks']*1.055
df.head()
```

Out[20]:

	FirstName	Age	Sex	Marks	Grade	UpdatedMarks
0	Jason Bourne	23.0	M	23	C	24.265
1	Ranjan	34.0	M	40	B	42.200
2	Roy	23.0	M	47	A	49.585
4	Gita	23.0	F	23	D	24.265
5	Sita	34.0	F	34	C	35.870

In [21]:
```
# t. Delete the “Marks Column”
df.drop("Marks", axis = 1, inplace = True)
df.head()
```

Out[21]:

	FirstName	Age	Sex	Grade	UpdatedMarks
0	Jason Bourne	23.0	M	C	24.265
1	Ranjan	34.0	M	B	42.200
2	Roy	23.0	M	A	49.585
4	Gita	23.0	F	D	24.265
5	Sita	34.0	F	C	35.870

EXPERIMENT 4

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
# 1. Write the command to read this data in the dataframe.
df = pd.read_csv('RSData.csv', na_values=['NA', 'n/a', 'na', 'NaN', 'nil'])
```

In [3]:
```
df.head()
```

Out[3]:

	propertyid	stno	stname	owneroccupied	numbedrooms	numbathrooms	areasqft
0	100001000.0	10.0	Kranti Road	Y	3.0	1	1000
1	100002000.0	17.0	Bhagat Singh Road	N	3.0	1.5	nil
2	100003000.0	NaN	Bhagat Singh Road	N	NaN	1	850
3	100004000.0	20.0	Azad Marg	12	1.0	NaN	700
4	NaN	23.0	Azad Marg	Y	3.0	2	1600

In [4]:
```
# 2. List number of rows and columns in this dataset
print('Number of rows, columns: ' + str(df.shape))
```

```
Number of rows, columns: (9, 7)
```

In [5]:
```
# 3. Check if there is any missing value in this entire dataset?
df.isnull().sum()
```

Out[5]:
```
propertyid       1
stno             2
stname           0
owneroccupied    1
numbedrooms      3
numbathrooms     1
areasqft         1
dtype: int64
```

In [6]:
```
# 4. Which column(s) does not have any missing value?
print('Columns without missing value')
for cols in df.columns:
    if df[cols].isnull().sum() == 0:
        print(cols)
```

```
Columns without missing value
stname
```

In [7]:
```
# 5. Which columns have maximum number of missing values?
print('Column having maximum number of missing values: ')
df.count().idxmin()
```

```
Column having maximum number of missing values:
```

Out[7]:
```
'numbedrooms'
```

In [8]:
```
# 6. There are how many rows, which does not have any missing value(s)?
rowswithna = sum([True for idx,row in df.iterrows() if any(row.isnull())])
rowswithoutna = len(df) - rowswithna
print('Rows that does not have any missing value(s): ' + str(rowswithoutna))
```

```
Rows that does not have any missing value(s): 2
```

In [9]:
```
# 7. If there is any missing value in the “areasqft” replace it with 900
df['areasqft'] = df['areasqft'].fillna(900)
```

```
df.tail()
```

Out[9]:

	propertyid	stno	stname	owneroccupied	numbedrooms	numbathrooms	areasqft
4	NaN	23.0	Azad Marg	Y	3.0	2	1600
5	100006000.0	20.0	Azad Marg	Y	NaN	1	800
6	100007000.0	NaN	Shivaji Road	NaN	2.0	Surya	950
7	100008000.0	13.0	Netaji Marg	Y	1.0	1	900
8	100009000.0	15.0	Netaji Marg	Y	NaN	2	1800

In [10]:

```
# 8. Fill the street number of record at index 2 with 77
df['stno'].loc[df.index[2]] = 77
df.head()
```

```
C:\Users\HP\anaconda3\lib\site-packages\pandas\core\indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

Out[10]:

	propertyid	stno	stname	owneroccupied	numbedrooms	numbathrooms	areasqft
0	100001000.0	10.0	Kranti Road	Y	3.0	1	1000
1	100002000.0	17.0	Bhagat Singh Road	N	3.0	1.5	nil
2	100003000.0	77.0	Bhagat Singh Road	N	NaN	1	850
3	100004000.0	20.0	Azad Marg	12	1.0	NaN	700
4	NaN	23.0	Azad Marg	Y	3.0	2	1600

In [11]:

```
# 9. If there is any missing value in the “number of bedrooms” columns, then replace it with the media
df['numbedrooms'] = df['numbedrooms'].fillna(np.nanmedian(df['numbedrooms']))
df.head()
```

Out[11]:

	propertyid	stno	stname	owneroccupied	numbedrooms	numbathrooms	areasqft
0	100001000.0	10.0	Kranti Road	Y	3.0	1	1000
1	100002000.0	17.0	Bhagat Singh Road	N	3.0	1.5	nil
2	100003000.0	77.0	Bhagat Singh Road	N	2.5	1	850
3	100004000.0	20.0	Azad Marg	12	1.0	NaN	700
4	NaN	23.0	Azad Marg	Y	3.0	2	1600

In [12]:

```
# 10. Give your comment on the “owner occupied” column.
df['owneroccupied'].value_counts()
```

Out[12]:

```
Y     5
N     2
12    1
Name: owneroccupied, dtype: int64
```

'owneroccupied' is a categorical variable. Most of the RS are owner occupied.

In [13]:

```
# 11. It is believed that the data entry operator might have entered integer values in “owner occupied
df['owneroccupied'].value_counts()
df['owneroccupied'].loc[df.index[3]] = np.nan
```

```
C:\Users\HP\anaconda3\lib\site-packages\pandas\core\indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

In [14]:
```
df.head()
```

Out[14]:

	propertyid	stno	stname	owneroccupied	numbedrooms	numbathrooms	areasqft
0	100001000.0	10.0	Kranti Road	Y	3.0	1	1000
1	100002000.0	17.0	Bhagat Singh Road	N	3.0	1.5	nil
2	100003000.0	77.0	Bhagat Singh Road	N	2.5	1	850
3	100004000.0	20.0	Azad Marg	NaN	1.0	NaN	700
4	NaN	23.0	Azad Marg	Y	3.0	2	1600

EXPERIMENT 5

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('mtcars.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [4]:
```
# a.Generate various summary statistics such as mean, standard deviation, minimum value,
# maximum value, and “1,2 & 3rd quantiles” for all the numerical attributes

df.describe(include = np.number)
```

Out[4]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32
mean	20.090625	6.187500	230.721875	146.687500	3.596563	3.217250	17.848750	0.437500	0.406250	3.687500	2
std	6.026948	1.785922	123.938694	68.562868	0.534679	0.978457	1.786943	0.504016	0.498991	0.737804	1
min	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1
25%	15.425000	4.000000	120.825000	96.500000	3.080000	2.581250	16.892500	0.000000	0.000000	3.000000	2
50%	19.200000	6.000000	196.300000	123.000000	3.695000	3.325000	17.710000	0.000000	0.000000	4.000000	2
75%	22.800000	8.000000	326.000000	180.000000	3.920000	3.610000	18.900000	1.000000	1.000000	4.000000	4
max	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8

In [5]:
```
# b. Count the number of non-NaN items per feature
df.isnull().sum()
```

Out[5]:
```
model    0
mpg      0
cyl      0
disp     0
hp       0
drat     0
wt       0
qsec     0
vs       0
am       0
gear     0
carb     0
dtype: int64
```

In [6]:
```
# c.Calculate mean absolute deviation
df.mad()
```

Out[6]:
```
mpg       4.714453
cyl       1.585938
disp    108.785742
hp       56.480469
drat      0.453242
wt        0.730187
qsec      1.376172
vs        0.492188
am        0.482422
gear      0.644531
carb      1.300781
dtype: float64
```

In [7]:
```
# d. Calculate median for any of the numeric type column
num_var  = df.select_dtypes(include=np.number).columns.tolist()

for col in num_var:
    print("Median of " + str(col) + " = " + str(np.median(df[col])))
```

```
Median of mpg = 19.2
Median of cyl = 6.0
Median of disp = 196.3
Median of hp = 123.0
Median of drat = 3.6950000000000003
Median of wt = 3.325
Median of qsec = 17.71
Median of vs = 0.0
Median of am = 0.0
Median of gear = 4.0
Median of carb = 2.0
```

In [8]:
```
# e. Calculate mean any of the numeric type column
for col in num_var:
    print("Mean of " + str(col) + " = " + str(np.mean(df[col])))
```

```
Mean of mpg = 20.090624999999996
Mean of cyl = 6.1875
Mean of disp = 230.72187500000004
Mean of hp = 146.6875
Mean of drat = 3.5965625000000006
Mean of wt = 3.2172499999999995
Mean of qsec = 17.848750000000003
Mean of vs = 0.4375
Mean of am = 0.40625
Mean of gear = 3.6875
Mean of carb = 2.8125
```

In [9]:
```
# f. Calculate mode for “hp” column
import statistics
print("Mode of df['hp'] = " + str(statistics.mode(df['hp'])))
```

```
Mode of df['hp'] = 110
```

In [10]:
```
# g. Calculate skewness for “disp”
import scipy.stats
print("Skewness of df['disp'] = " + str(scipy.stats.skew(df['disp'], axis = 0, bias = True)))
```

```
Skewness of df['disp'] = 0.40027244847286664
```

In [11]:
```
# h. Find the coefficient of correlation between all the numeric attributes
df.corr()
```

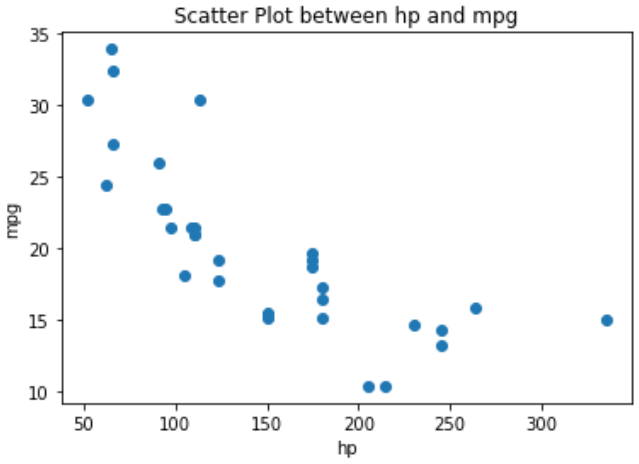
Out[11]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carl
mpg	1.000000	-0.852162	-0.847551	-0.776168	0.681172	-0.867659	0.418684	0.664039	0.599832	0.480285	-0.55092
cyl	-0.852162	1.000000	0.902033	0.832447	-0.699938	0.782496	-0.591242	-0.810812	-0.522607	-0.492687	0.52698
disp	-0.847551	0.902033	1.000000	0.790949	-0.710214	0.887980	-0.433698	-0.710416	-0.591227	-0.555569	0.39497
hp	-0.776168	0.832447	0.790949	1.000000	-0.448759	0.658748	-0.708223	-0.723097	-0.243204	-0.125704	0.74981
drat	0.681172	-0.699938	-0.710214	-0.448759	1.000000	-0.712441	0.091205	0.440278	0.712711	0.699610	-0.09079
wt	-0.867659	0.782496	0.887980	0.658748	-0.712441	1.000000	-0.174716	-0.554916	-0.692495	-0.583287	0.42760

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carl
qsec	0.418684	-0.591242	-0.433698	-0.708223	0.091205	-0.174716	1.000000	0.744535	-0.229861	-0.212682	-0.65624
vs	0.664039	-0.810812	-0.710416	-0.723097	0.440278	-0.554916	0.744535	1.000000	0.168345	0.206023	-0.56960
am	0.599832	-0.522607	-0.591227	-0.243204	0.712711	-0.692495	-0.229861	0.168345	1.000000	0.794059	0.05753
gear	0.480285	-0.492687	-0.555569	-0.125704	0.699610	-0.583287	-0.212682	0.206023	0.794059	1.000000	0.27407
carb	-0.550925	0.526988	0.394977	0.749812	-0.090790	0.427606	-0.656249	-0.569607	0.057534	0.274073	1.00000

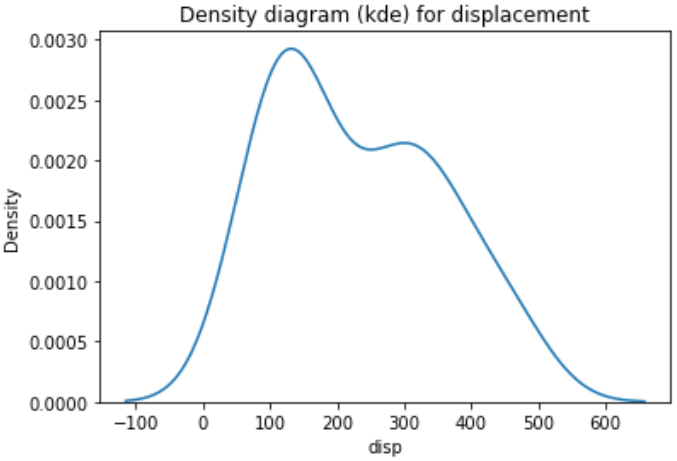
In [12]:
```
# i. Scatter plot a graph between “hp” vs “mpg”
plt.scatter(df['hp'], df['mpg'])
plt.title('Scatter Plot between hp and mpg')
plt.xlabel('hp')
plt.ylabel('mpg')
```

Out[12]: Text(0, 0.5, 'mpg')



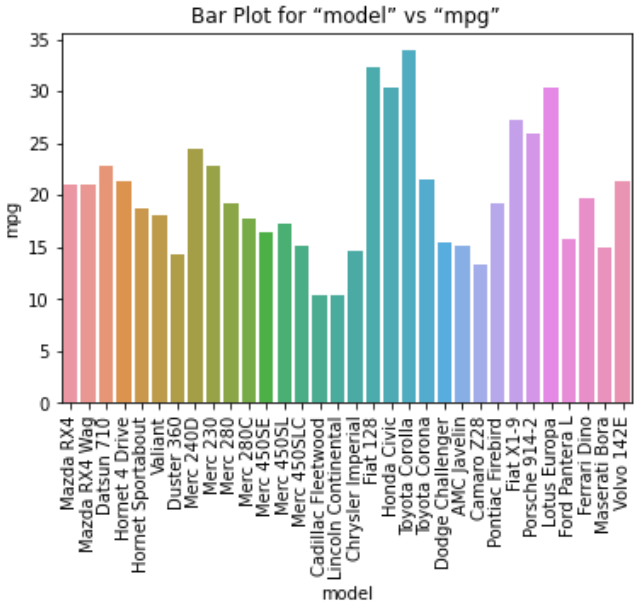
In [13]:
```
# j. Plot a density diagram (kde- kernel desity estimation) for “displacement”
sns.kdeplot(df['disp'])
plt.title('Density diagram (kde) for displacement')
plt.xlabel('disp')
```

Out[13]: Text(0.5, 0, 'disp')



In [14]:
```
# k. Plot a bar diagram for “model” vs “mpg”
sns.barplot(data = df,  x = "model", y = "mpg")
plt.xticks(rotation = 90)
plt.title('Bar Plot for “model” vs “mpg”')
plt.xlabel('model')
plt.ylabel('mpg')
```

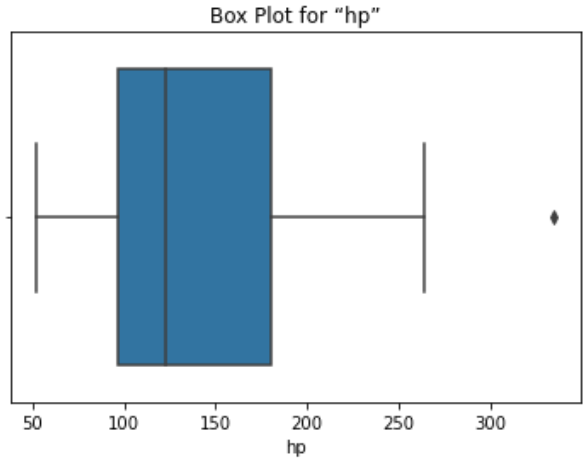
Out[14]: Text(0, 0.5, 'mpg')



In [15]:
```
# l. Box plot the “hp” attribute.
sns.boxplot(df['hp'])
plt.title('Box Plot for “hp”')
plt.xlabel('hp')
```

```
C:\Users\HP\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(
```

Out[15]: Text(0.5, 0, 'hp')



EXPERIMENT 6

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('CO2Small.csv')
```

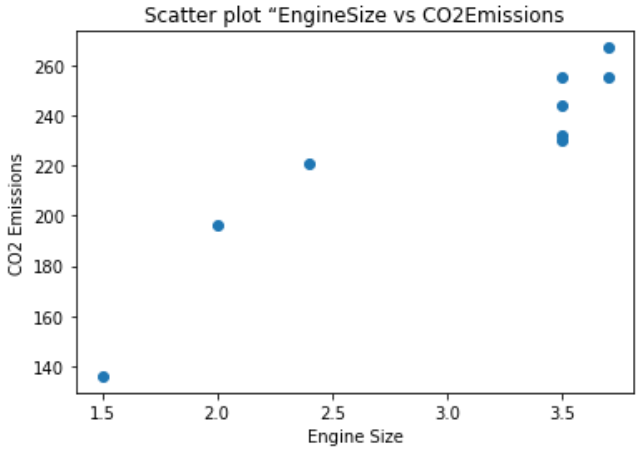
In [3]:
```
df.head()
```

Out[3]:

	EngineSize	Cylinders	FuelConsum	CO2Emissions
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244

In [4]:
```
# 1. Scatter plot “EngineSize vs CO2Emissions”
plt.scatter(x = df['EngineSize'], y = df['CO2Emissions'])
plt.title("Scatter plot “EngineSize vs CO2Emissions")
plt.xlabel("Engine Size")
plt.ylabel("CO2 Emissions")
```

Out[4]:
```
Text(0, 0.5, 'CO2 Emissions')
```



In [5]:
```
X = df.iloc[:,0:1].values
Y = df.iloc[:,-1].values
```

In [6]:
```
X
```

Out[6]:
```
array([[2. ],
       [2.4],
       [1.5],
       [3.5],
       [3.5],
       [3.5],
       [3.5],
       [3.7],
       [3.7]])
```

In [7]:
```
Y
```

Out[7]:
```
array([196, 221, 136, 255, 244, 230, 232, 255, 267], dtype=int64)
```

In [8]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X, Y)
```

Out[8]:
```
▾ LinearRegression

LinearRegression()
```

In [9]:
```
print(regressor.coef_)
print(regressor.intercept_)
```

```
[43.98446834]
92.80266825965754
```

In [10]:
```
Y_train_prediction = regressor.predict(X)
Y_train_prediction
```

Out[10]:
```
array([180.77160494, 198.36539227, 158.77937077, 246.74830745,
       246.74830745, 246.74830745, 246.74830745, 255.54520112,
       255.54520112])
```

In [11]:
```
X_train = [[2.4]]
```

In [12]:
```
# 2. Predict the value of CO2 emission on the basis of Engine size for enginesize = 2.4. using data giver
Y_test_prediction = regressor.predict(X_train)
print(Y_test_prediction)
```

```
[198.36539227]
```

In [13]:
```
# 3. Calculate R^2
from sklearn.metrics import r2_score
r2_score(Y, Y_train_prediction)
```

Out[13]:
```
0.8457874165566486
```

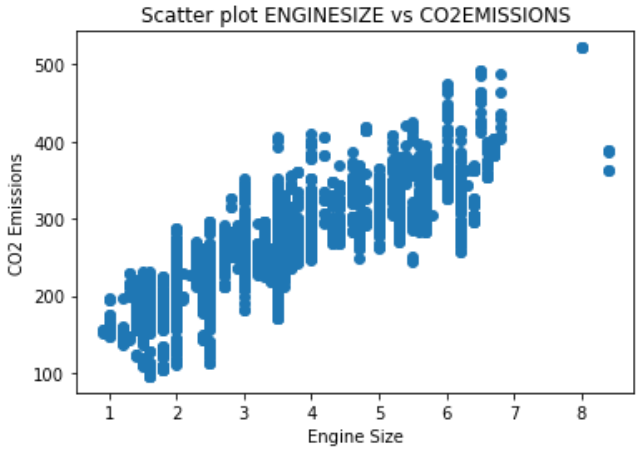
In [14]:
```
# 4. Calculate the above details on the larger dataset i.e. file "CO2FullData.csv"
df2 = pd.read_csv('CO2FullData.csv')
df2.head()
```

Out[14]:

	Make	Model	Vehicle Class	ENGINESIZE	CYLINDERS	Transmission	Fuel Type	FUELCONSUMPTION_CITY	FUELCONSUMPTION_I
0	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	
1	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	

In [15]:
```
# Scatter plot "EngineSize vs CO2Emissions"
plt.scatter(x = df2['ENGINESIZE'], y = df2['CO2EMISSIONS'])
plt.title("Scatter plot ENGINESIZE vs CO2EMISSIONS")
plt.xlabel("Engine Size")
plt.ylabel("CO2 Emissions")
```

Out[15]:
```
Text(0, 0.5, 'CO2 Emissions')
```



In [16]:
```
X2 = df2.iloc[:,3:4].values
Y2 = df2.iloc[:,-1].values
```

In [17]:
```
X2
```

Out[17]:
```
array([[2. ],
       [2.4],
       [1.5],
       ...,
       [2. ],
       [2. ],
       [2. ]])
```

In [18]:
```
Y2
```

Out[18]:
```
array([196, 221, 136, ..., 240, 232, 248], dtype=int64)
```

In [19]:
```
regressor2 = LinearRegression()
regressor2.fit(X2, Y2)
```

Out[19]:
```
▾ LinearRegression

LinearRegression()
```

In [20]:
```
print(regressor2.coef_)
print(regressor2.intercept_)
```

```
[36.77731519]
134.36589272349642
```

In [21]:
```
Y_train_prediction2 = regressor2.predict(X2)
Y_train_prediction2
```

Out[21]:
```
array([207.9205231 , 222.63144917, 189.5318655 , ..., 207.9205231 ,
       207.9205231 , 207.9205231 ])
```

In [22]:
```
# Calculate R^2
r2_score(Y2, Y_train_prediction2)
```

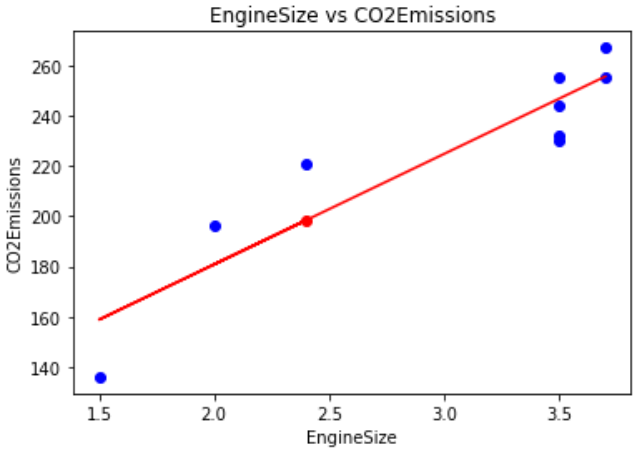
Out[22]:
```
0.724447204652408
```

In [23]:
```
# 5. Print the final equation of the linear
print("Equation of the regression line: ")
print("y = " + str(regressor.coef_[0]) + " * x + " + str(regressor.intercept_))
```

```
Equation of the regression line:
y = 43.98446833930704 * x + 92.80266825965754
```

In [24]:
```
# 6. Plot final line (best fit line) along with other data points
plt.scatter(X, Y, color ='blue')
```

```
plt.scatter(2.4, Y_test_prediction, color ='red')
plt.plot(X, Y_train_prediction, color ='red')
plt.xlabel('EngineSize')
plt.ylabel('CO2Emissions')
plt.title('EngineSize vs CO2Emissions')
plt.show()
```



In [25]:

```
# 7. Give your comments by comparing the R2 of both the datasets
print("R2 Score of small dataset: " + str(r2_score(Y, Y_train_prediction)))
print("R2 Score of full dataset: " + str(r2_score(Y2, Y_train_prediction2)))
```

```
R2 Score of small dataset: 0.8457874165566486
R2 Score of full dataset: 0.724447204652408
```

EXPERIMENT 7

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('../Lab 4/CO2Small.csv')
```

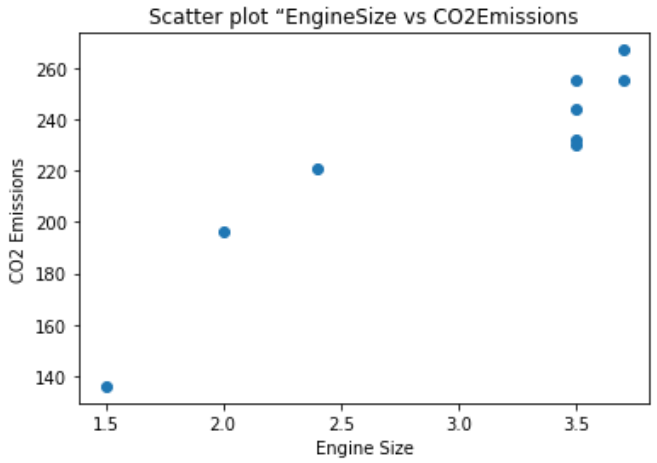
In [3]:
```
df.head()
```

Out[3]:

	EngineSize	Cylinders	FuelConsum	CO2Emissions
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244

In [4]:
```
# 1. Scatter plot “EngineSize vs CO2Emissions”
plt.scatter(x = df['EngineSize'], y = df['CO2Emissions'])
plt.title("Scatter plot “EngineSize vs CO2Emissions")
plt.xlabel("Engine Size")
plt.ylabel("CO2 Emissions")
```

Out[4]:
```
Text(0, 0.5, 'CO2 Emissions')
```



In [5]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit_transform(df)
```

Out[5]:
```
array([[-1.31233465, -1.41421356, -0.81110711, -0.80253099],
       [-0.80433414, -1.41421356, -0.14194374, -0.13867263],
       [-1.94733528, -1.41421356, -2.39276596, -2.39579105],
       [ 0.59266726,  0.70710678,  0.77055175,  0.76417473],
       [ 0.59266726,  0.70710678,  0.46638659,  0.47207705],
       [ 0.59266726,  0.70710678,  0.10138839,  0.10031637],
       [ 0.59266726,  0.70710678,  0.16222142,  0.15342504],
       [ 0.84666751,  0.70710678,  0.77055175,  0.76417473],
       [ 0.84666751,  0.70710678,  1.07471691,  1.08282674]])
```

In [6]:
```
X = df.iloc[:,0:1].values
Y = df.iloc[:,-1].values
```

In [7]:
```
from sklearn.linear_model import SGDRegressor
regressor = SGDRegressor(max_iter = 1000, tol = 1e-3)
regressor.fit(X, Y)
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:1527: Convergence
Warning: Maximum number of iteration reached before convergence. Consider increasing max_iter to impro
ve the fit.
  warnings.warn(
```

Out[7]:
```
▾ SGDRegressor

SGDRegressor()
```

In [8]:
```
print(regressor.coef_)
print(regressor.intercept_)
```

```
[54.88440972]
[57.59307236]
```

In [9]:
```
Y_prediction = regressor.predict(X)
print(Y_prediction)
```

```
[167.36189179 189.31565568 139.91968693 249.68850637 249.68850637
 249.68850637 249.68850637 260.66538831 260.66538831]
```

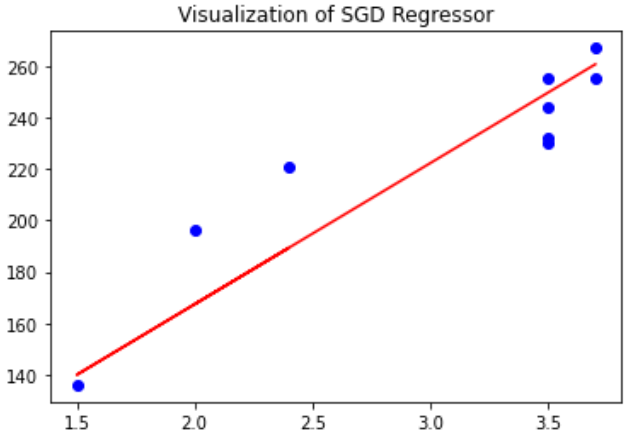
In [10]:
```
# 2. Calculate R^2
from sklearn.metrics import r2_score
print('R2 Score: ' +  str(r2_score(Y, Y_prediction)))
```

```
R2 Score: 0.790597659828152
```

In [11]:
```
# 3. Print the final equation of the line
print("Equation of the SGD regression line: ")
print("y = " + str(regressor.coef_[0]) + " * x + " + str(regressor.intercept_))
```

```
Equation of the SGD regression line:
y = 54.88440971674993 * x + [57.59307236]
```

In [12]:
```
# 4. Plot final line (best fit line) along with other data points
plt.scatter(X, Y,color ='blue')
plt.plot(X, Y_prediction,color ='red')
plt.title('Visualization of SGD Regressor')
plt.show()
```



EXPERIMENT 7

Step 1: Problem Statement

To perform prediction using Logistic Regression on the dataset

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Data Collection

In [2]:
```
df = pd.read_csv(r'./datasets/ChurnData.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...	pager	internet	callwait	confer	ebill	lo
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40	...	1.0	0.0	1.0	1.0	0.0	
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.45	...	0.0	0.0	0.0	0.0	0.0	
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30	...	0.0	0.0	0.0	1.0	0.0	
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.05	...	1.0	1.0	1.0	1.0	1.0	
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...	0.0	0.0	1.0	1.0	0.0	

5 rows × 28 columns

In [4]:
```
df = df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip', 'callcard', 'wireless','churn']]
```

In [5]:
```
df.head()
```

Out[5]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1.0
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1.0
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0.0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0.0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0.0

Step 3: Exploratory Data Analysis

In [6]:
```
df.shape
```

Out[6]:
```
(200, 10)
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   tenure    200 non-null    float64
 1   age       200 non-null    float64
 2   address   200 non-null    float64
 3   income    200 non-null    float64
 4   ed        200 non-null    float64
 5   employ    200 non-null    float64
 6   equip     200 non-null    float64
 7   callcard  200 non-null    float64
 8   wireless  200 non-null    float64
 9   churn     200 non-null    float64
```

```
dtypes: float64(10)
memory usage: 15.8 KB
```

In [8]:
```
df.describe()
```

Out[8]:

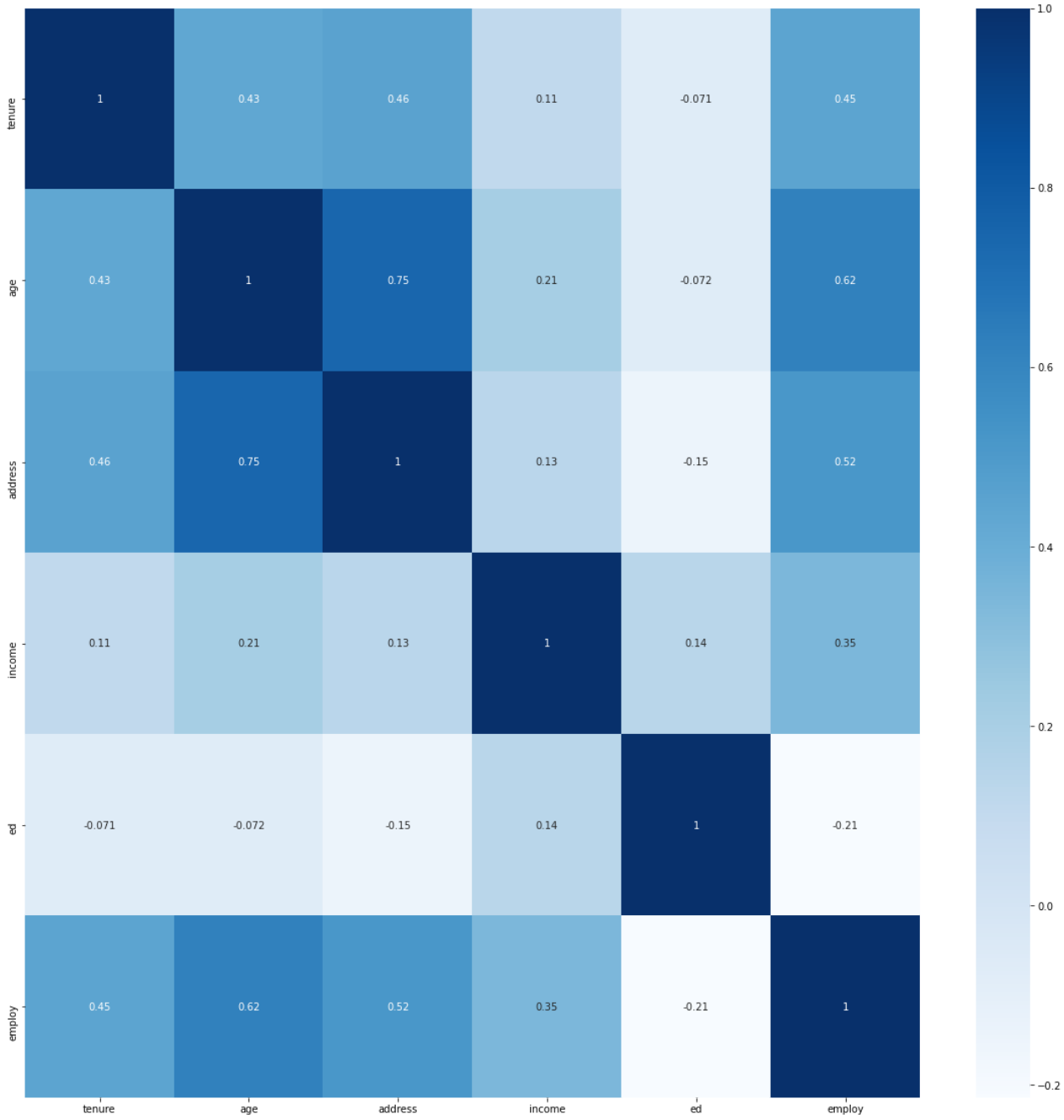
	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
count	200.000000	200.000000	200.000000	200.000000	200.00000	200.00000	200.000000	200.000000	200.000000	200.000000
mean	35.505000	41.165000	11.650000	75.130000	2.82500	10.22500	0.425000	0.705000	0.290000	0.290000
std	21.640971	13.076803	10.158419	128.430468	1.28555	8.95743	0.495584	0.457187	0.454901	0.454901
min	1.000000	19.000000	0.000000	9.000000	1.00000	0.00000	0.000000	0.000000	0.000000	0.000000
25%	16.750000	31.000000	3.000000	31.000000	2.00000	3.00000	0.000000	0.000000	0.000000	0.000000
50%	33.500000	40.000000	9.000000	48.000000	3.00000	7.50000	0.000000	1.000000	0.000000	0.000000
75%	55.250000	51.000000	18.000000	80.000000	4.00000	17.00000	1.000000	1.000000	1.000000	1.000000
max	72.000000	76.000000	48.000000	1668.000000	5.00000	44.00000	1.000000	1.000000	1.000000	1.000000

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

Out[9]:
```
tenure      float64
age         float64
address     float64
income      float64
ed          float64
employ      float64
equip       float64
callcard    float64
wireless    float64
churn       float64
dtype: object
```

In [10]:
```
# Correlation matrix
plt.figure(figsize=(20,20))
sns.heatmap(df[["tenure", "age", "address", "income", "ed", "employ"]].corr(), annot=True, cmap="Blues")
```

Out[10]:
```
<AxesSubplot:>
```



In [11]:

```
# Checking for NaN values in the dataset
df.isnull().sum()
```

Out[11]:

```
tenure      0
age         0
address     0
income      0
ed          0
employ      0
equip       0
callcard    0
wireless    0
churn       0
dtype: int64
```

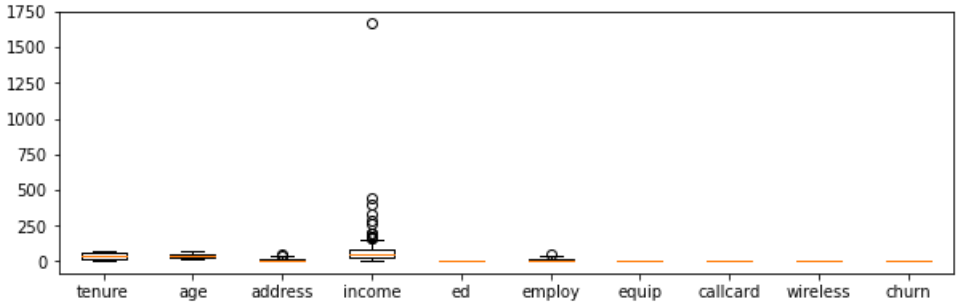
In [13]:

```
# Visualizing the outliers
fig=plt.figure(figsize=(10,3))
plt.boxplot(df, labels=df.columns)
```

Out[13]:

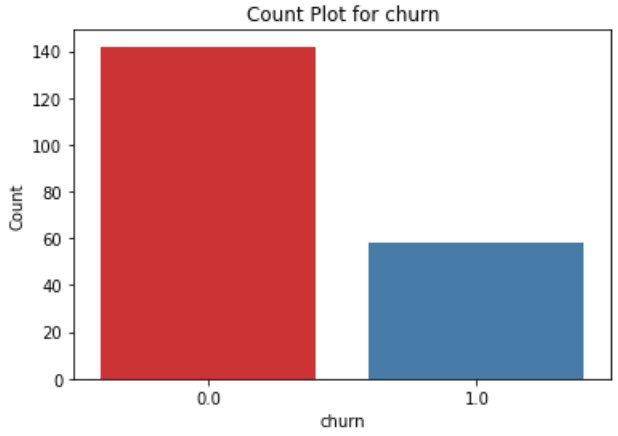
```
{'whiskers': [<matplotlib.lines.Line2D at 0x185d1432220>,
  <matplotlib.lines.Line2D at 0x185d1432580>,
  <matplotlib.lines.Line2D at 0x185d143ba30>,
  <matplotlib.lines.Line2D at 0x185d143bd90>,
  <matplotlib.lines.Line2D at 0x185d1450250>,
  <matplotlib.lines.Line2D at 0x185d14505b0>,
  <matplotlib.lines.Line2D at 0x185d145fa30>,
  <matplotlib.lines.Line2D at 0x185d145fd90>,
```

```
 <matplotlib.lines.Line2D at 0x185d1476280>,
 <matplotlib.lines.Line2D at 0x185d14765e0>,
 <matplotlib.lines.Line2D at 0x185d1483a60>,
 <matplotlib.lines.Line2D at 0x185d1483dc0>,
 <matplotlib.lines.Line2D at 0x185d149b280>,
 <matplotlib.lines.Line2D at 0x185d149b5e0>,
 <matplotlib.lines.Line2D at 0x185d14a3a60>,
 <matplotlib.lines.Line2D at 0x185d14a3dc0>,
 <matplotlib.lines.Line2D at 0x185d14bd280>,
 <matplotlib.lines.Line2D at 0x185d14bd5e0>,
 <matplotlib.lines.Line2D at 0x185d14c7a60>,
 <matplotlib.lines.Line2D at 0x185d14c7dc0>],
'caps': [<matplotlib.lines.Line2D at 0x185d14328e0>,
 <matplotlib.lines.Line2D at 0x185d1432c40>,
 <matplotlib.lines.Line2D at 0x185d1449130>,
 <matplotlib.lines.Line2D at 0x185d1449490>,
 <matplotlib.lines.Line2D at 0x185d1450910>,
 <matplotlib.lines.Line2D at 0x185d1450c70>,
 <matplotlib.lines.Line2D at 0x185d146b130>,
 <matplotlib.lines.Line2D at 0x185d146b490>,
 <matplotlib.lines.Line2D at 0x185d1476940>,
 <matplotlib.lines.Line2D at 0x185d1476ca0>,
 <matplotlib.lines.Line2D at 0x185d148e160>,
 <matplotlib.lines.Line2D at 0x185d148e4c0>,
 <matplotlib.lines.Line2D at 0x185d149b940>,
 <matplotlib.lines.Line2D at 0x185d149bca0>,
 <matplotlib.lines.Line2D at 0x185d14b2160>,
 <matplotlib.lines.Line2D at 0x185d14b24c0>,
 <matplotlib.lines.Line2D at 0x185d14bd940>,
 <matplotlib.lines.Line2D at 0x185d14bdca0>,
 <matplotlib.lines.Line2D at 0x185d14d4160>,
 <matplotlib.lines.Line2D at 0x185d14d44c0>],
'boxes': [<matplotlib.lines.Line2D at 0x185d1421e80>,
 <matplotlib.lines.Line2D at 0x185d143b6d0>,
 <matplotlib.lines.Line2D at 0x185d1449eb0>,
 <matplotlib.lines.Line2D at 0x185d145f6d0>,
 <matplotlib.lines.Line2D at 0x185d146bee0>,
 <matplotlib.lines.Line2D at 0x185d1483700>,
 <matplotlib.lines.Line2D at 0x185d148eee0>,
 <matplotlib.lines.Line2D at 0x185d14a3700>,
 <matplotlib.lines.Line2D at 0x185d14b2ee0>,
 <matplotlib.lines.Line2D at 0x185d14c7700>],
'medians': [<matplotlib.lines.Line2D at 0x185d1432fa0>,
 <matplotlib.lines.Line2D at 0x185d14497f0>,
 <matplotlib.lines.Line2D at 0x185d1450fd0>,
 <matplotlib.lines.Line2D at 0x185d146b820>,
 <matplotlib.lines.Line2D at 0x185d1483040>,
 <matplotlib.lines.Line2D at 0x185d148e820>,
 <matplotlib.lines.Line2D at 0x185d14a3040>,
 <matplotlib.lines.Line2D at 0x185d14b2820>,
 <matplotlib.lines.Line2D at 0x185d14c7040>,
 <matplotlib.lines.Line2D at 0x185d14d4820>],
'fliers': [<matplotlib.lines.Line2D at 0x185d143b340>,
 <matplotlib.lines.Line2D at 0x185d1449b50>,
 <matplotlib.lines.Line2D at 0x185d145f370>,
 <matplotlib.lines.Line2D at 0x185d146bb80>,
 <matplotlib.lines.Line2D at 0x185d14833a0>,
 <matplotlib.lines.Line2D at 0x185d148eb80>,
 <matplotlib.lines.Line2D at 0x185d14a33a0>,
 <matplotlib.lines.Line2D at 0x185d14b2b80>,
 <matplotlib.lines.Line2D at 0x185d14c73a0>,
 <matplotlib.lines.Line2D at 0x185d14d4b80>],
'means': []}
```



In [14]:

```
countplot = sns.countplot(x=df['churn'], data=df, palette='Set1')
plt.xlabel('churn')
plt.ylabel('Count')
plt.title('Count Plot for churn')
plt.show()
```

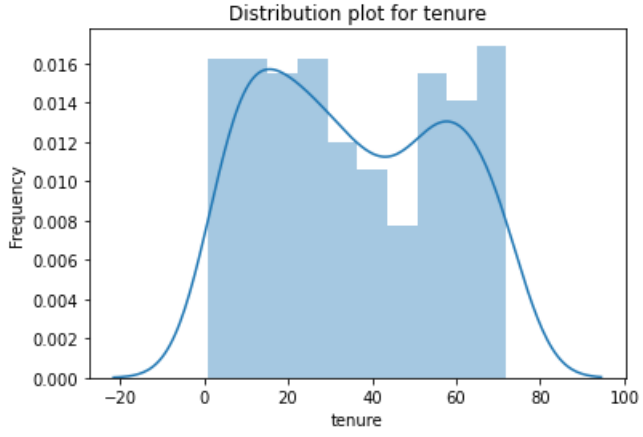


Dataset is not imbalanced

In [15]:

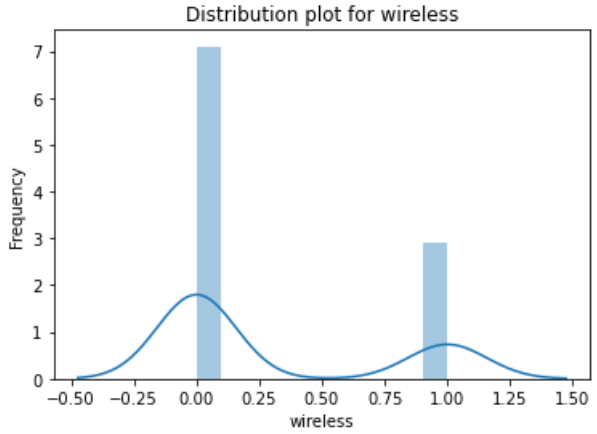
```
# Visualization of the features of the dataset using Distribution Plots
for cols in df.columns:
    distributionplot = sns.distplot(a=df[cols], bins=10, hist=True, kde=True)
    plt.xlabel(cols)
    plt.ylabel('Frequency')
    plt.title('Distribution plot for ' + cols)
    plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

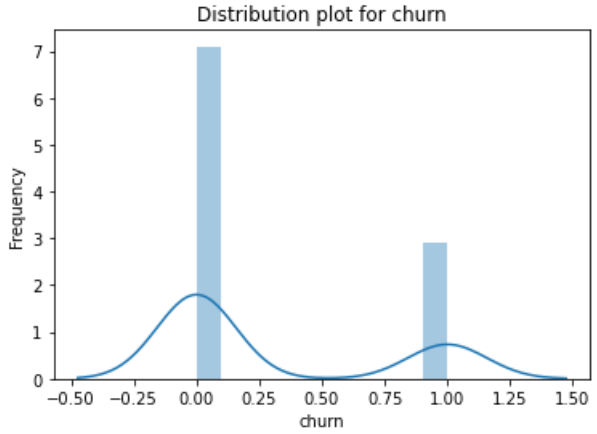



```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```


```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Step 4: Data Preprocessing

Data Preprocessing on Train Data

In [16]:
```
# Input DataFrame
X=df.iloc[:,0:-1]
X.head()
```

Out[16]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0

In [17]:
```
# Target DataFrame
Y=df.iloc[:,-1]
Y.head()
```

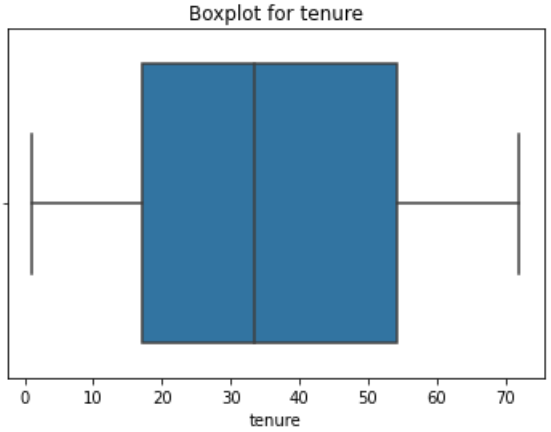
Out[17]:
```
0    1.0
1    1.0
2    0.0
3    0.0
4    0.0
Name: churn, dtype: float64
```

In [18]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

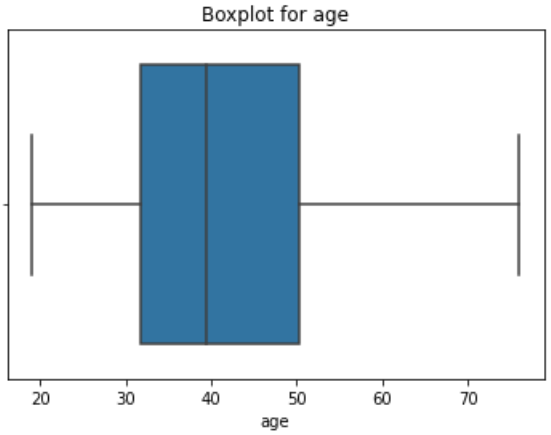
In [19]:
```
for cols in X_train.columns:
    bp = sns.boxplot(X_train[cols])
    plt.xlabel(cols)
```

```
    plt.title('Boxplot for ' + cols)
    plt.show()
```

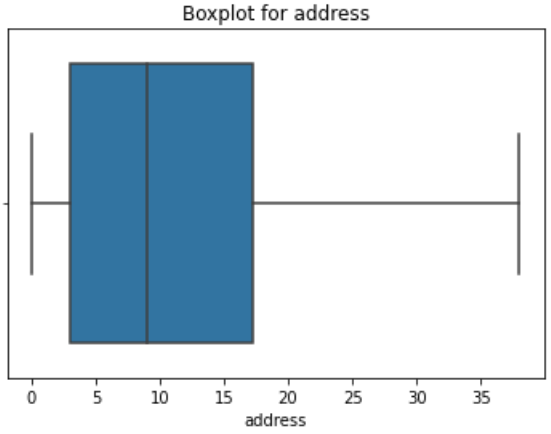
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



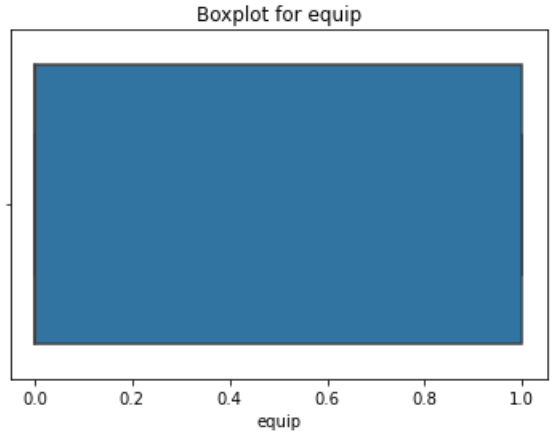

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



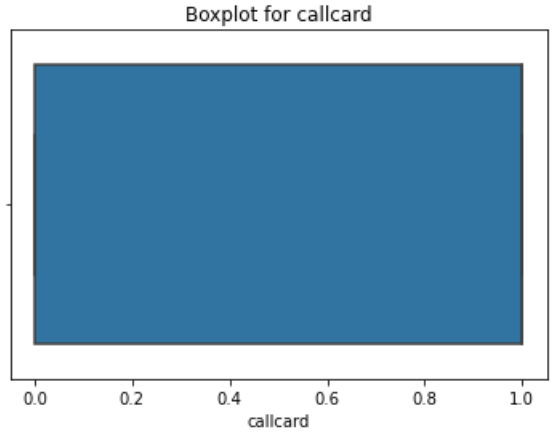

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



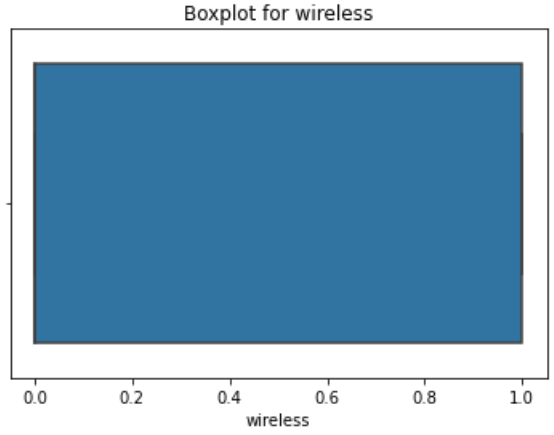

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```




```
C:\Users\HP\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(
```




```
C:\Users\HP\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(
```

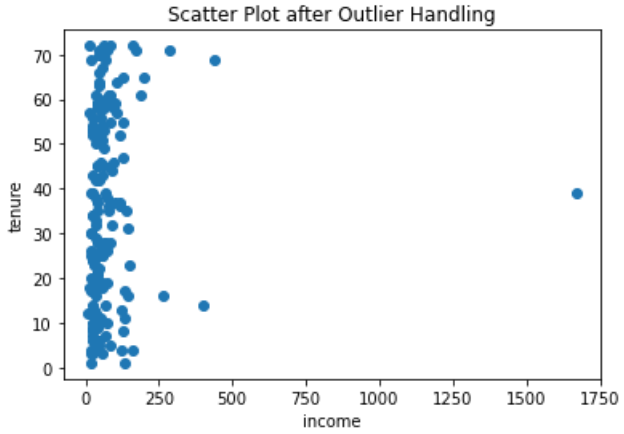


In [20]:

```
#Visualization using Scatter Plot after Outlier Handling
plt.scatter(X_train['income'],X_train['tenure'])
plt.xlabel('income')
plt.ylabel('tenure')
plt.title('Scatter Plot after Outlier Handling')
```

Out[20]:

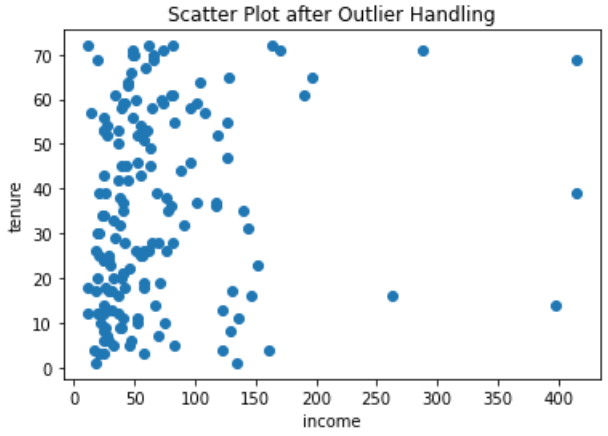
```
Text(0.5, 1.0, 'Scatter Plot after Outlier Handling')
```



In [21]:
```
# Handling the outliers using quantile-based Capping and Flooring techniqueupper_limit = X_train['bmi'].quant
upper_limit = X_train['income'].quantile(0.99)
lower_limit = X_train['income'].quantile(0.01)
X_train['income'] = np.where(X_train['income'] >= upper_limit, upper_limit,
        np.where(X_train['income'] <= lower_limit, lower_limit, X_train['income']))
```

In [22]:
```
#Visualization using Scatter Plot after Outlier Handling
plt.scatter(X_train['income'],X_train['tenure'])
plt.xlabel('income')
plt.ylabel('tenure')
plt.title('Scatter Plot after Outlier Handling')
```

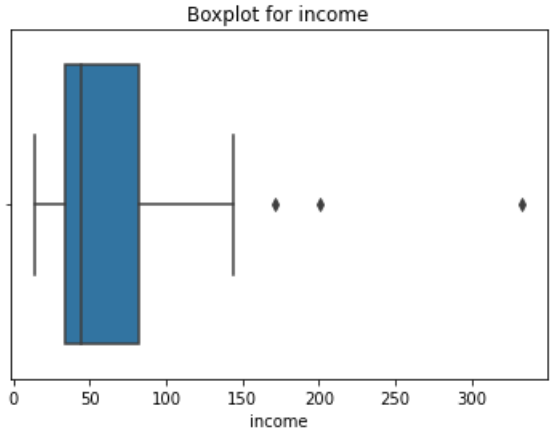
Out[22]:
```
Text(0.5, 1.0, 'Scatter Plot after Outlier Handling')
```



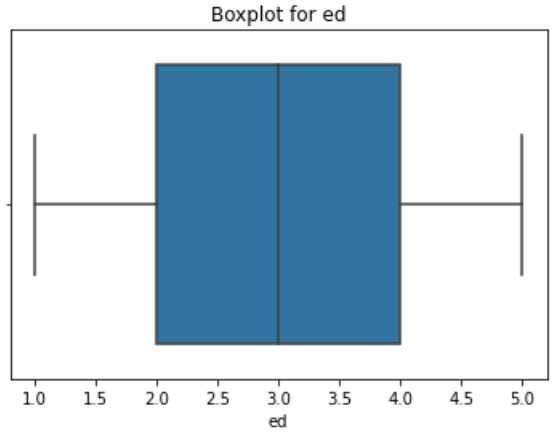
Data Preprocessing on Testing Data

In [23]:
```
for cols in X_test.columns:
    bp = sns.boxplot(X_test[cols])
    plt.xlabel(cols)
    plt.title('Boxplot for ' + cols)
    plt.show()
```

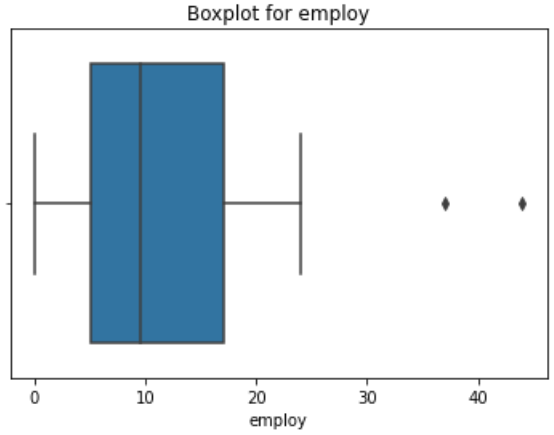
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



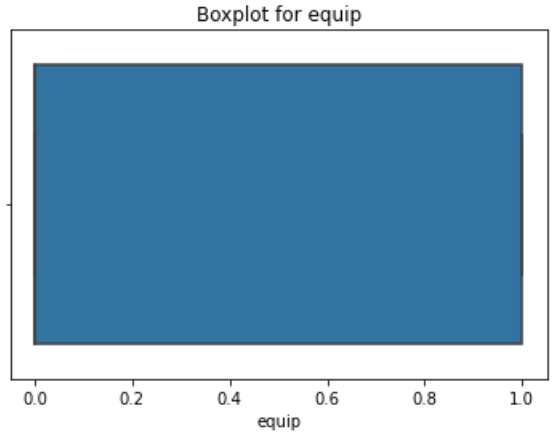

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



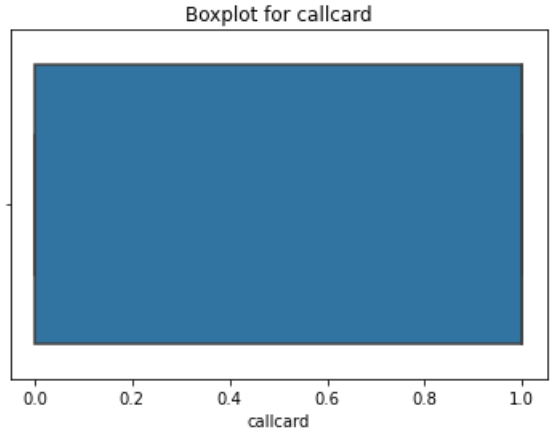

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



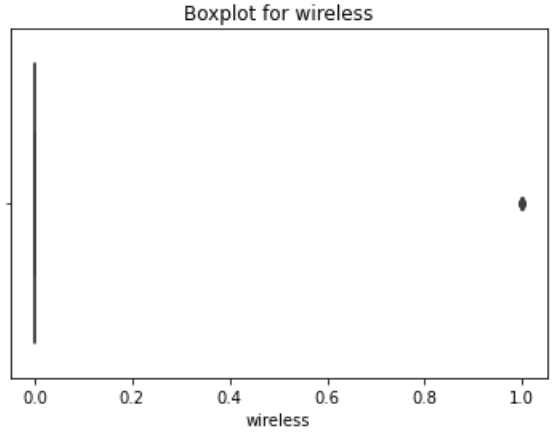

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```




```
C:\Users\HP\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(
```




```
C:\Users\HP\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(
```



In [24]:

```
# Handling the outliers using quantile-based Capping and Flooring techniqueupper_limit = X_train['bmi'].quant
upper_limit = X_train['income'].quantile(0.99)
lower_limit = X_train['income'].quantile(0.01)
X_train['income'] = np.where(X_train['income'] >= upper_limit, upper_limit,
        np.where(X_train['income'] <= lower_limit, lower_limit, X_train['income']))
```

In [25]:

```
# Handling the outliers using quantile-based Capping and Flooring techniqueupper_limit = X_train['bmi'].quant
upper_limit = X_train['employ'].quantile(0.99)
lower_limit = X_train['employ'].quantile(0.01)
X_train['employ'] = np.where(X_train['employ'] >= upper_limit, upper_limit,
        np.where(X_train['employ'] <= lower_limit, lower_limit, X_train['employ']))
```

Feature Scaling (standardizing the features)

In [26]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
scaler.fit_transform(X_train)
scaler.transform(X_test)
```

Out[26]:
```
array([[ 1.34853581, -1.25143165, -0.74850189,  0.11192614,  0.12608293,
        -0.90652149,  1.13389342,  0.6352234 , -0.67419986],
       [-0.89658776,  0.66387227,  0.60413402, -0.42613956, -0.64981202,
        -0.55808832,  1.13389342, -1.5742493 ,  1.4832397 ],
       [-1.31754842, -1.09182299, -0.8525508 , -0.45688617,  0.12608293,
        -0.90652149, -0.8819171 , -1.5742493 , -0.67419986],
       [ 1.25498899,  2.81858919,  0.91628076, -0.51837939,  0.12608293,
         0.95178876, -0.8819171 ,  0.6352234 , -0.67419986],
       [ 0.36629425, -0.93221433, -0.74850189, -0.549126  , -1.42570698,
         0.37106681, -0.8819171 ,  0.6352234 ,  1.4832397 ],
       [-0.14821324,  0.26485062, -0.12420839,  2.03358934, -0.64981202,
         1.64865511, -0.8819171 ,  0.6352234 , -0.67419986],
       [ 1.53562944,  0.90328526,  0.60413402,  0.78835159, -1.42570698,
         1.64865511, -0.8819171 ,  0.6352234 , -0.67419986],
       [-1.36432183, -0.93221433, -0.74850189, -0.76435228, -0.64981202,
        -0.55808832, -0.8819171 , -1.5742493 , -0.67419986],
       [ 0.97434855,  0.7436766 , -1.06064863,  0.17341936, -0.64981202,
         1.64865511, -0.8819171 ,  0.6352234 ,  1.4832397 ],
       [-0.84981435, -0.13417103,  0.08388944, -0.36464634,  0.90197788,
         0.02263364,  1.13389342, -1.5742493 , -0.67419986],
       [ 1.16144218,  0.34465495,  1.22842751,  0.74223167,  0.90197788,
         0.13877803,  1.13389342,  0.6352234 ,  1.4832397 ],
       [ 1.72272307,  2.73878486,  3.8296504 , -0.84121881, -0.64981202,
        -0.44194392, -0.8819171 ,  0.6352234 , -0.67419986],
       [ 0.97434855,  0.98308959,  1.22842751,  0.48088547,  1.67787284,
         0.4872112 ,  1.13389342,  0.6352234 , -0.67419986],
       [ 1.16144218,  0.98308959,  1.12437859,  1.57239017, -1.42570698,
         3.15853219, -0.8819171 ,  0.6352234 , -0.67419986],
       [ 0.78725492,  1.70132856,  1.22842751, -0.50300609,  0.90197788,
         0.83564437, -0.8819171 ,  0.6352234 , -0.67419986],
       [-1.22400161, -1.57064897, -1.06064863, -0.79509889, -0.64981202,
        -0.90652149, -0.8819171 , -1.5742493 , -0.67419986],
       [-0.10143983,  1.06289392,  0.70818293, -0.18016667,  0.90197788,
        -0.67423271,  1.13389342, -1.5742493 , -0.67419986],
       [ 1.72272307,  2.73878486,  2.68511233, -0.549126  , -1.42570698,
         3.97154292, -0.8819171 ,  0.6352234 , -0.67419986],
       [-0.33530687, -0.93221433, -0.74850189, -0.70285906,  0.12608293,
        -0.55808832, -0.8819171 , -1.5742493 , -0.67419986],
       [ 0.36629425, -1.09182299, -0.8525508 , -0.73360567, -0.64981202,
        -1.02266588, -0.8819171 ,  0.6352234 , -0.67419986],
       [ 0.74048151,  0.66387227,  1.852721  , -0.36464634, -1.42570698,
         0.71949998, -0.8819171 ,  0.6352234 , -0.67419986],
       [ 1.53562944,  0.10524196,  0.5000851 ,  0.31177911,  0.90197788,
         0.25492242,  1.13389342,  0.6352234 , -0.67419986],
       [ 0.45984106,  2.02054589,  3.30940582,  1.15731092, -0.64981202,
         0.37106681, -0.8819171 ,  0.6352234 , -0.67419986],
       [-1.1772282 , -1.6504533 , -1.06064863, -0.74897898, -0.64981202,
        -1.13881027, -0.8819171 , -1.5742493 , -0.67419986],
       [ 1.48885603,  1.5417199 ,  2.16486775,  0.37327234, -1.42570698,
         1.30022194,  1.13389342,  0.6352234 , -0.67419986],
       [-1.08368139, -0.77260567, -0.33230622, -0.77972559,  0.90197788,
        -0.67423271,  1.13389342, -1.5742493 , -0.67419986],
       [ 0.74048151,  0.42445928, -0.33230622,  0.58849861, -0.64981202,
         1.30022194, -0.8819171 ,  0.6352234 ,  1.4832397 ],
       [-0.05466642, -0.0543667 ,  1.02032968, -0.70285906,  0.90197788,
        -0.09351075,  1.13389342,  0.6352234 ,  1.4832397 ],
       [-0.5224005 , -0.45338835, -0.12420839, -0.42613956,  1.67787284,
        -0.44194392,  1.13389342, -1.5742493 , -0.67419986],
       [-0.10143983, -0.61299701,  0.08388944, -0.41076625,  0.90197788,
        -0.32579953,  1.13389342,  0.6352234 ,  1.4832397 ],
       [-0.38208027,  0.82348093, -0.8525508 ,  0.17341936,  1.67787284,
         0.13877803,  1.13389342, -1.5742493 , -0.67419986],
       [-0.94336116, -0.45338835, -0.64445297, -0.5337527 ,  0.12608293,
        -0.20965514,  1.13389342, -1.5742493 , -0.67419986],
       [-1.41109524,  0.18504629,  0.5000851 ,  0.05043292,  0.12608293,
         0.83564437, -0.8819171 ,  0.6352234 , -0.67419986],
       [-1.41109524,  0.50426361, -0.43635514, -0.34927303, -1.42570698,
        -0.44194392, -0.8819171 ,  0.6352234 , -0.67419986],
       [ 0.45984106, -1.49084464, -0.95659972, -0.50300609,  0.90197788,
        -1.13881027,  1.13389342, -1.5742493 , -0.67419986],
       [ 1.44208262,  0.7436766 , -0.95659972,  4.06286569,  1.67787284,
         1.64865511, -0.8819171 ,  0.6352234 , -0.67419986],
       [-0.5224005 ,  0.42445928,  0.08388944, -0.39539295, -0.64981202,
        -0.44194392, -0.8819171 ,  0.6352234 ,  1.4832397 ],
       [-1.31754842, -0.45338835,  0.29198727,  0.17341936, -0.64981202,
         0.60335559, -0.8819171 ,  0.6352234 , -0.67419986],
       [-1.1772282 , -0.61299701, -0.95659972, -0.04180692,  0.12608293,
        -0.09351075, -0.8819171 ,  0.6352234 , -0.67419986],
       [-0.75626753, -1.17162732, -0.95659972, -0.31852642,  0.12608293,
        -1.13881027, -0.8819171 ,  0.6352234 , -0.67419986]])
```

Step 5: Model Building

In [27]:
```
from sklearn.linear_model import LogisticRegression
model_logistic = LogisticRegression()
model_logistic.fit(X_train,Y_train)

Y_pred_train_logistic=model_logistic.predict(X_train)
Y_pred_test_logistic=model_logistic.predict(X_test)
```

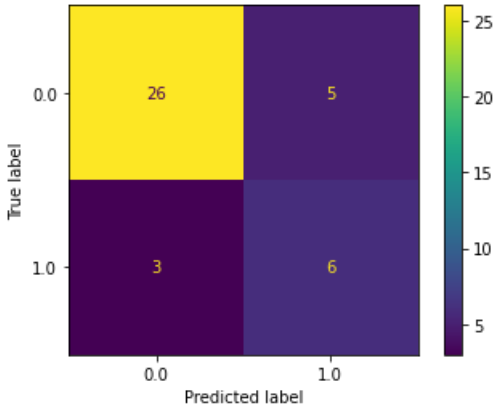
```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:444: ConvergenceWarning: lbfgs fail
ed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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(
```

Step 6: Model Evaluation

In [28]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, recall_score, fbeta_scor
print('Test accuracy: ' + str(accuracy_score(Y_test, Y_pred_test_logistic)))

print('\nClassification Report:\n')
print(classification_report(Y_test, Y_pred_test_logistic))

print('\nConfusion Matrix:\n')
plot_confusion_matrix(model_logistic,X_test, Y_test)
```

```
Test accuracy: 0.8

Classification Report:

              precision    recall  f1-score   support

         0.0       0.90      0.84      0.87        31
         1.0       0.55      0.67      0.60         9

    accuracy                           0.80        40
   macro avg       0.72      0.75      0.73        40
weighted avg       0.82      0.80      0.81        40


Confusion Matrix:

C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusi
on_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Us
e one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[28]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x185d15423d0>
```



Test accuracy: 0.8

Recall Score: 0.84

Precision: 0.90

F1-Score: 0.87

EXPERIMENT 8

Step 1: Problem Statement

Perform classification using Naive Bayes on the IRIS dataset

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Data Collection

In [2]:
```
df = pd.read_csv(r'../6ML SEM 6/datasets/Iris.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Step 3: Exploratory Data Analysis

In [4]:
```
df.shape
```

Out[4]:
```
(150, 5)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

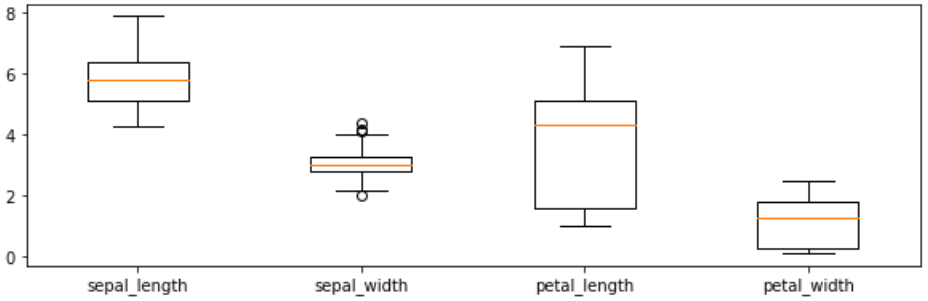
	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [7]:
```
# Checking for NaN values in the dataset
df.isnull().sum()
```

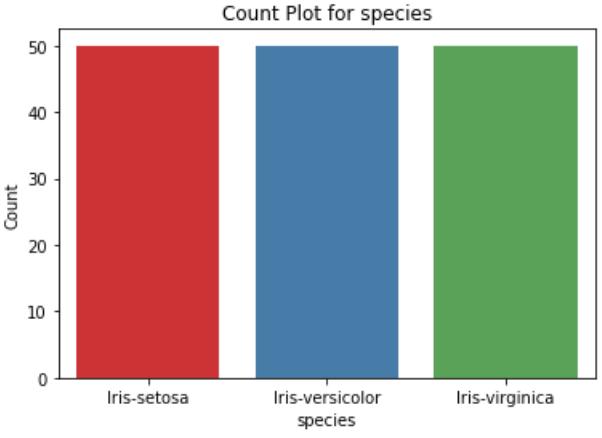
Out[7]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [9]:
```
# Visualizing the outliers
fig=plt.figure(figsize=(10,3))
plt.boxplot(df[["sepal_length", "sepal_width", "petal_length", "petal_width"]], labels = ["sepal_length", "se
```

Out[9]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2b2e0fb1100>,
  <matplotlib.lines.Line2D at 0x2b2e0fb1460>,
  <matplotlib.lines.Line2D at 0x2b2e0fbb940>,
  <matplotlib.lines.Line2D at 0x2b2e0fbbca0>,
  <matplotlib.lines.Line2D at 0x2b2e0fd3160>,
  <matplotlib.lines.Line2D at 0x2b2e0fd34c0>,
  <matplotlib.lines.Line2D at 0x2b2e0fde940>,
  <matplotlib.lines.Line2D at 0x2b2e0fdeca0>],
 'caps': [<matplotlib.lines.Line2D at 0x2b2e0fb17c0>,
  <matplotlib.lines.Line2D at 0x2b2e0fb1b20>,
  <matplotlib.lines.Line2D at 0x2b2e0fc7040>,
  <matplotlib.lines.Line2D at 0x2b2e0fc73a0>,
  <matplotlib.lines.Line2D at 0x2b2e0fd3820>,
  <matplotlib.lines.Line2D at 0x2b2e0fd3b80>,
  <matplotlib.lines.Line2D at 0x2b2e0fe9040>,
  <matplotlib.lines.Line2D at 0x2b2e0fe93a0>],
 'boxes': [<matplotlib.lines.Line2D at 0x2b2def99d60>,
  <matplotlib.lines.Line2D at 0x2b2e0fbb5e0>,
  <matplotlib.lines.Line2D at 0x2b2e0fc7dc0>,
  <matplotlib.lines.Line2D at 0x2b2e0fde5e0>],
 'medians': [<matplotlib.lines.Line2D at 0x2b2e0fb1e80>,
  <matplotlib.lines.Line2D at 0x2b2e0fc7700>,
  <matplotlib.lines.Line2D at 0x2b2e0fd3ee0>,
  <matplotlib.lines.Line2D at 0x2b2e0fe9700>],
 'fliers': [<matplotlib.lines.Line2D at 0x2b2e0fbb250>,
  <matplotlib.lines.Line2D at 0x2b2e0fc7a60>,
  <matplotlib.lines.Line2D at 0x2b2e0fde280>,
  <matplotlib.lines.Line2D at 0x2b2e0fe9a60>],
 'means': []}
```



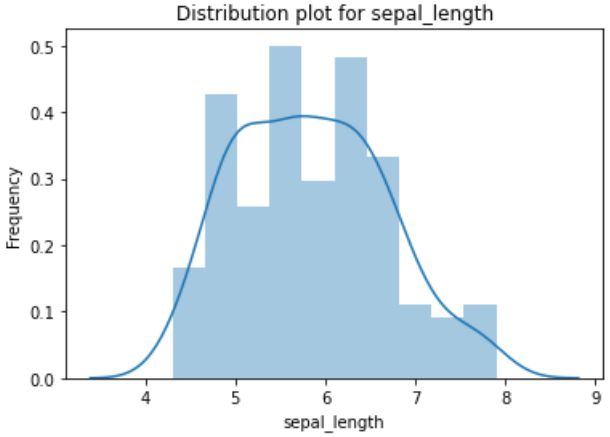
In [10]:
```
countplot = sns.countplot(x=df['species'], data=df, palette='Set1')
plt.xlabel('species')
plt.ylabel('Count')
plt.title('Count Plot for species')
plt.show()
```



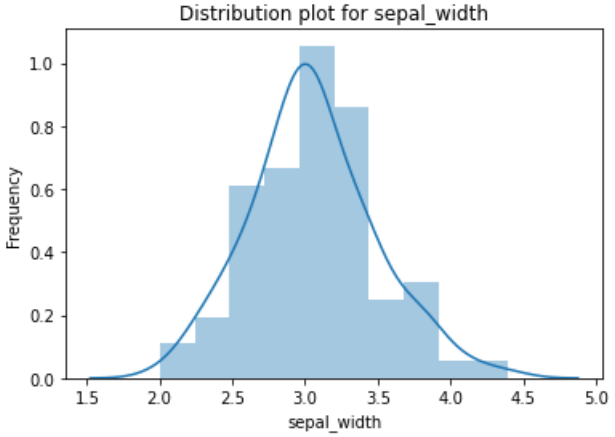
In [11]:
```
# Visualization of the features of the dataset using Distribution Plots
for cols in ["sepal_length", "sepal_width", "petal_length", "petal_width"]:
    distributionplot = sns.distplot(a=df[cols], bins=10, hist=True, kde=True)
```

```
    plt.xlabel(cols)
    plt.ylabel('Frequency')
    plt.title('Distribution plot for ' + cols)
    plt.show()
```

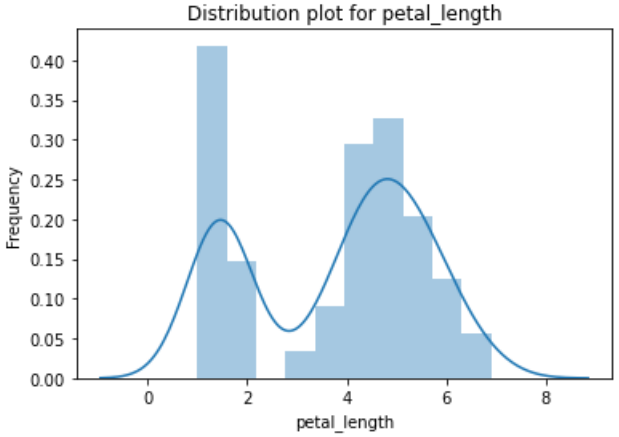
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



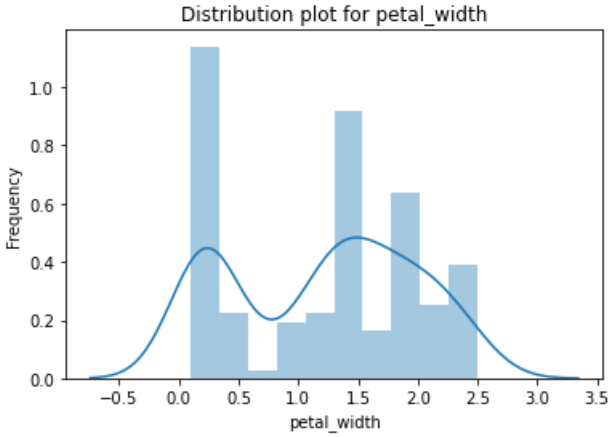

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Step 4: Data Preprocessing

Data Preprocessing on Train Data

In [12]:
```
# Input DataFrame
X=df.iloc[:,0:-1]
X.head()
```

Out[12]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [13]:
```
# Target DataFrame
Y=df.iloc[:,-1]
Y.head()
```

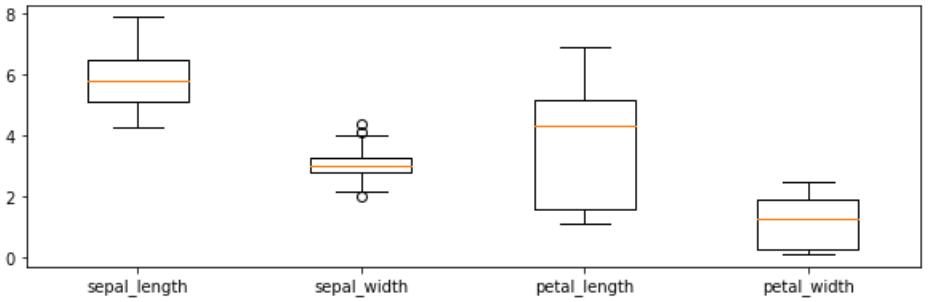
Out[13]:
```
0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
Name: species, dtype: object
```

In [14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

In [15]:
```
# Visualizing the outliers
fig=plt.figure(figsize=(10,3))
plt.boxplot(X_train[["sepal_length", "sepal_width", "petal_length", "petal_width"]], labels = ["sepal_length"
```

Out[15]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2b2e27c5460>,
  <matplotlib.lines.Line2D at 0x2b2e27c57c0>,
  <matplotlib.lines.Line2D at 0x2b2e27d1c40>,
  <matplotlib.lines.Line2D at 0x2b2e27d1fa0>,
  <matplotlib.lines.Line2D at 0x2b2e27e8460>,
  <matplotlib.lines.Line2D at 0x2b2e27e87c0>,
  <matplotlib.lines.Line2D at 0x2b2e27f4c70>,
  <matplotlib.lines.Line2D at 0x2b2e27f4fd0>],
 'caps': [<matplotlib.lines.Line2D at 0x2b2e27c5b20>,
  <matplotlib.lines.Line2D at 0x2b2e27c5e80>,
  <matplotlib.lines.Line2D at 0x2b2e27dc340>,
  <matplotlib.lines.Line2D at 0x2b2e27dc6a0>,
  <matplotlib.lines.Line2D at 0x2b2e27e8b20>,
  <matplotlib.lines.Line2D at 0x2b2e27e8e80>,
  <matplotlib.lines.Line2D at 0x2b2e27ff370>,
  <matplotlib.lines.Line2D at 0x2b2e27ff6d0>],
 'boxes': [<matplotlib.lines.Line2D at 0x2b2e27c5100>,
  <matplotlib.lines.Line2D at 0x2b2e27d18e0>,
  <matplotlib.lines.Line2D at 0x2b2e27e8100>,
  <matplotlib.lines.Line2D at 0x2b2e27f4910>],
 'medians': [<matplotlib.lines.Line2D at 0x2b2e27d1220>,
  <matplotlib.lines.Line2D at 0x2b2e27dca00>,
```

```
 <matplotlib.lines.Line2D at 0x2b2e27f4220>,
 <matplotlib.lines.Line2D at 0x2b2e27ffa30>],
'fliers': [<matplotlib.lines.Line2D at 0x2b2e27d1580>,
 <matplotlib.lines.Line2D at 0x2b2e27dcd60>,
 <matplotlib.lines.Line2D at 0x2b2e27f45b0>,
 <matplotlib.lines.Line2D at 0x2b2e27ffd90>],
'means': []}
```

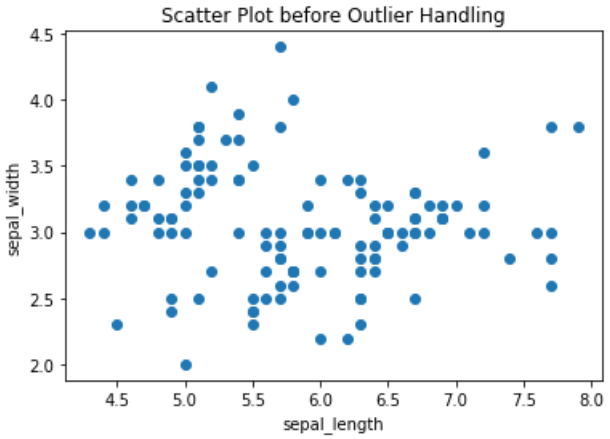


In [16]:

```
#Visualization using Scatter Plot before Outlier Handling
plt.scatter(X_train['sepal_length'],X_train['sepal_width'])
plt.xlabel('sepal_length')
plt.ylabel('sepal_width')
plt.title('Scatter Plot before Outlier Handling')
```

Out[16]:

```
Text(0.5, 1.0, 'Scatter Plot before Outlier Handling')
```



In [17]:

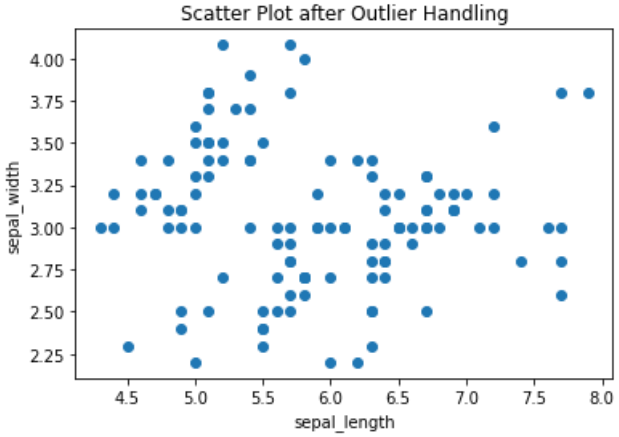
```
# Handling the outliers using quantile-based Capping and Flooring techniqueupper_limit = X_train['bmi'].quant
upper_limit = X_train['sepal_width'].quantile(0.99)
lower_limit = X_train['sepal_width'].quantile(0.01)
X_train['sepal_width'] = np.where(X_train['sepal_width'] >= upper_limit, upper_limit,
        np.where(X_train['sepal_width'] <= lower_limit, lower_limit, X_train['sepal_width']))
```

In [18]:

```
#Visualization using Scatter Plot after Outlier Handling
plt.scatter(X_train['sepal_length'],X_train['sepal_width'])
plt.xlabel('sepal_length')
plt.ylabel('sepal_width')
plt.title('Scatter Plot after Outlier Handling')
```

Out[18]:

```
Text(0.5, 1.0, 'Scatter Plot after Outlier Handling')
```



Feature Scaling

In [19]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit_transform(X_train)
scaler.transform(X_test)
```

Out[19]:
```
array([[-0.09544771, -0.6025416 ,  0.72247648,  1.51195265],
       [ 0.14071157, -2.03612149,  0.10361279, -0.29851096],
       [-0.44968663,  2.74247816, -1.35915595, -1.3330616 ],
       [ 1.6757469 , -0.36361161,  1.39760052,  0.73603967],
       [-1.04008484,  0.8310383 , -1.30289562, -1.3330616 ],
       [ 0.49495049,  0.59210832,  1.22881951,  1.64127148],
       [-1.04008484,  1.06996828, -1.41541629, -1.20374277],
       [ 0.96726906,  0.11424835,  0.49743514,  0.34808318],
       [ 1.0853487 , -0.6025416 ,  0.55369548,  0.21876435],
       [ 0.25879121, -0.6025416 ,  0.10361279,  0.08944552],
       [ 0.25879121, -1.08040156,  1.00377816,  0.21876435],
       [ 0.61303014,  0.35317834,  0.38491447,  0.34808318],
       [ 0.25879121, -0.6025416 ,  0.49743514, -0.03987331],
       [ 0.73110978, -0.6025416 ,  0.4411748 ,  0.34808318],
       [ 0.25879121, -0.36361161,  0.49743514,  0.21876435],
       [-1.15816448,  0.11424835, -1.30289562, -1.46238043],
       [ 0.14071157, -0.36361161,  0.38491447,  0.34808318],
       [-0.44968663, -1.08040156,  0.32865413, -0.03987331],
       [-1.27624412, -0.12468163, -1.35915595, -1.20374277],
       [-0.56776627,  2.02568822, -1.41541629, -1.07442394],
       [-0.33160699, -0.6025416 ,  0.60995581,  0.99467733],
       [-0.33160699, -0.12468163,  0.38491447,  0.34808318],
       [-1.27624412,  0.8310383 , -1.07785427, -1.3330616 ],
       [-1.74856268, -0.36361161, -1.35915595, -1.3330616 ],
       [ 0.37687085, -0.6025416 ,  0.55369548,  0.73603967],
       [-1.5124034 ,  1.30889827, -1.5841973 , -1.3330616 ],
       [-0.9220052 ,  1.78675823, -1.07785427, -1.07442394],
       [ 0.37687085, -0.36361161,  0.27239379,  0.08944552],
       [-1.04008484, -1.79719151, -0.29020957, -0.29851096],
       [-1.04008484,  0.8310383 , -1.24663528, -1.07442394]])
```

Step 5: Model Building

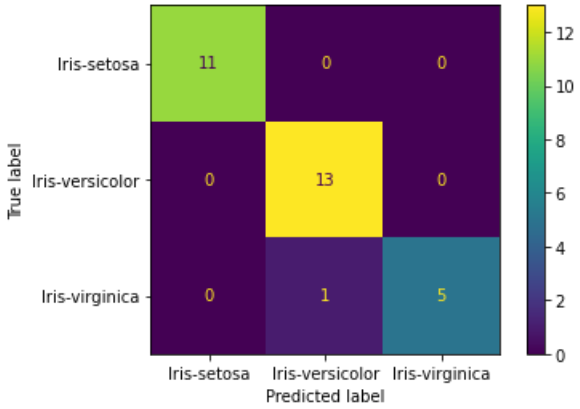
In [20]:
```
from sklearn.naive_bayes import GaussianNB
model_naive_bayes = GaussianNB()

model_naive_bayes.fit(X_train,Y_train)

Y_pred_train_naive_bayes = model_naive_bayes.predict(X_train)
Y_pred_test_naive_bayes = model_naive_bayes.predict(X_test)
```

Step 6: Model Evaluation

In [22]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, recall_score, fbeta_scor

print('\nConfusion Matrix:\n')
plot_confusion_matrix(model_naive_bayes,X_test, Y_test)
```

```
Confusion Matrix:

C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusi
on_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Us
e one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[22]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b2e22c1070>
```



In [24]:

```
print('Train accuracy: ' + str(accuracy_score(Y_train, Y_pred_train_naive_bayes)))
print('Test accuracy: ' + str(accuracy_score(Y_test, Y_pred_test_naive_bayes)))

print('\nClassification Report:\n')
print(classification_report(Y_test, Y_pred_test_naive_bayes))
```

```
Train accuracy: 0.95
Test accuracy: 0.9666666666666667

Classification Report:

                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.93      1.00      0.96        13
 Iris-virginica       1.00      0.83      0.91         6

       accuracy                           0.97        30
      macro avg       0.98      0.94      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

EXPERIMENT 9

Step 1: Problem Statement

Build an ML model on "diabetes" dataset using Decision Trees

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
pd.set_option('display.max_columns', None)
```

Step 2: Data Collection

In [3]:
```
df = pd.read_csv(r'../6ML SEM 6/datasets/diabetes.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Step 3: Exploratory Data Analysis

In [5]:
```
df.shape
```

Out[5]:
```
(768, 9)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [7]:
```
df.describe()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Ou
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.3
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.4
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.0

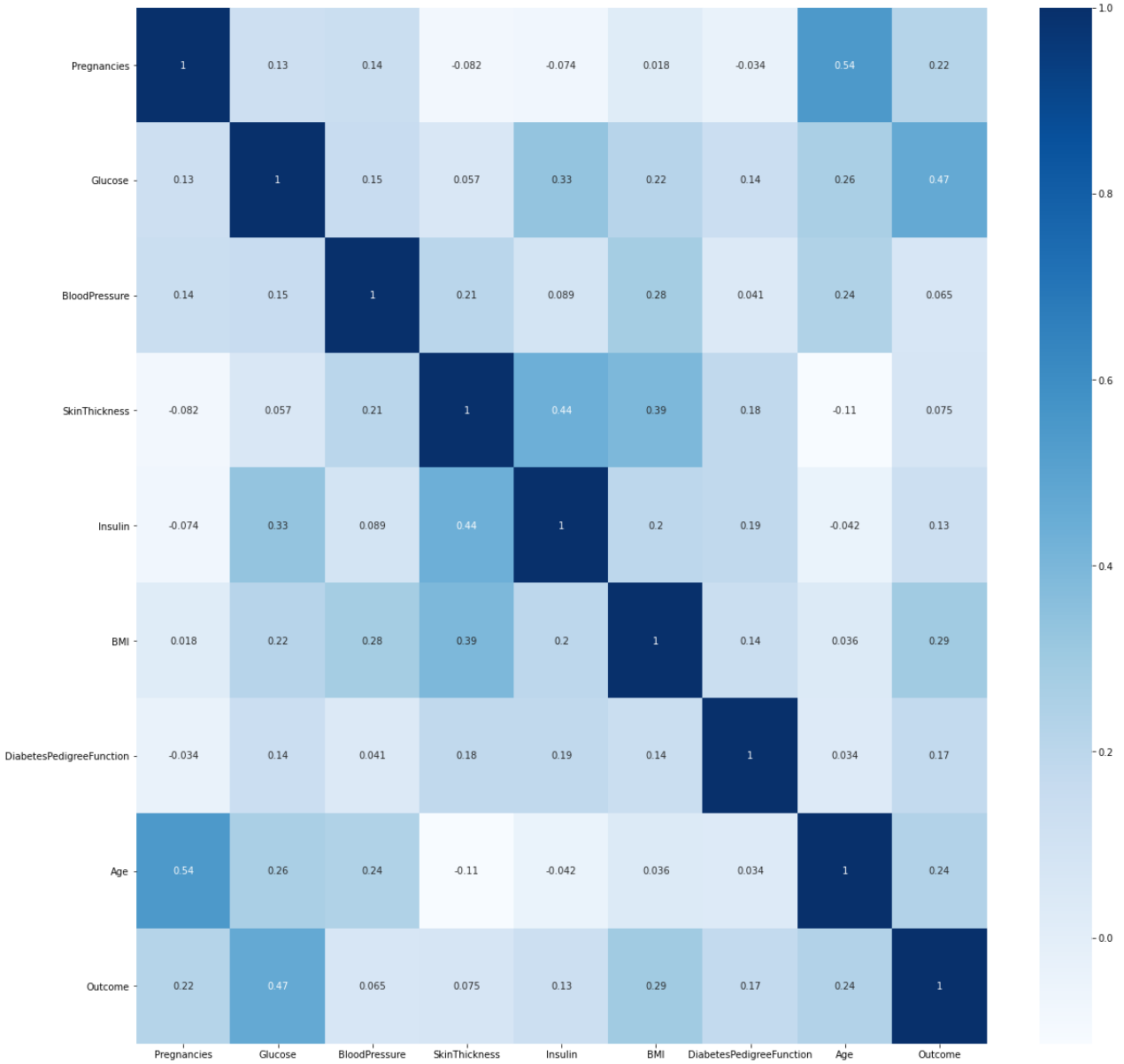
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Ou
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.0

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

Out[8]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [9]:
```
# Correlation matrix
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), annot=True, cmap="Blues")
```

Out[9]: <AxesSubplot:>



In [10]:
```
# Checking for NaN values in the dataset
df.isnull().sum()
```

Out[10]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
```

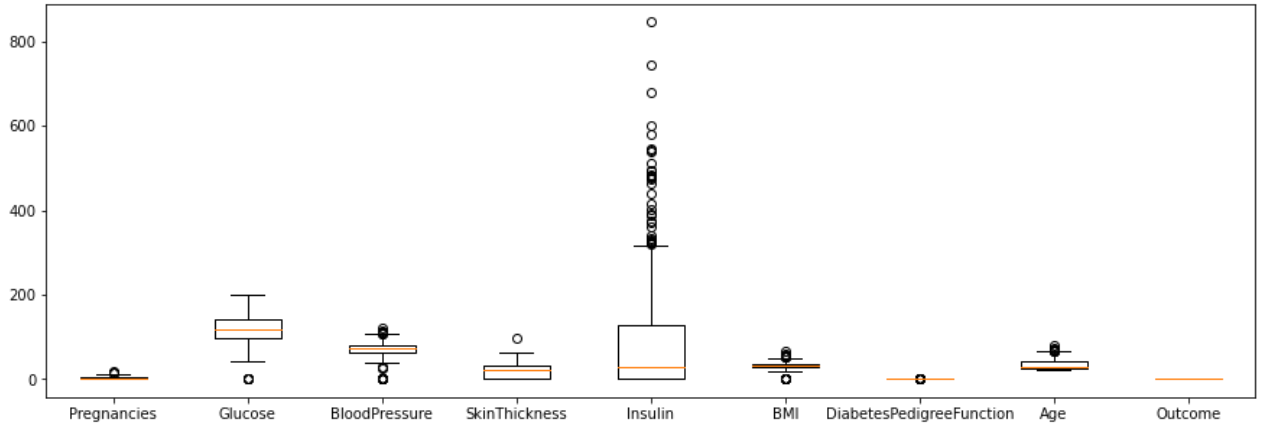
```
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [12]:

```
# Visualizing the outliers
fig=plt.figure(figsize=(15,5))
plt.boxplot(df, labels=df.columns)
```

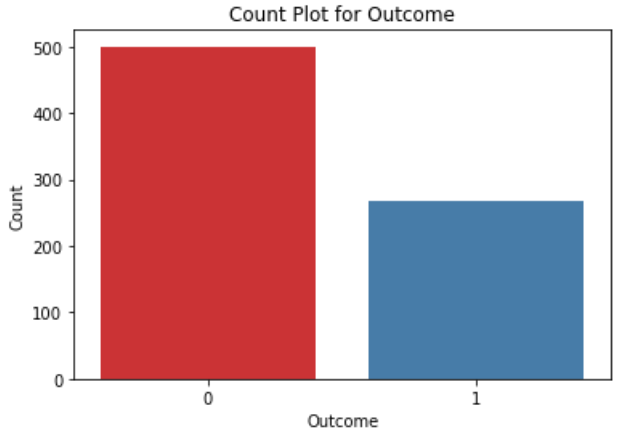
Out[12]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1f5ff69da00>,
  <matplotlib.lines.Line2D at 0x1f5ff69dd60>,
  <matplotlib.lines.Line2D at 0x1f5ff6bc250>,
  <matplotlib.lines.Line2D at 0x1f5ff6bc5b0>,
  <matplotlib.lines.Line2D at 0x1f5ff6c6a30>,
  <matplotlib.lines.Line2D at 0x1f5ff6c6d90>,
  <matplotlib.lines.Line2D at 0x1f5ff6df250>,
  <matplotlib.lines.Line2D at 0x1f5ff6df5b0>,
  <matplotlib.lines.Line2D at 0x1f5ff6d2a30>,
  <matplotlib.lines.Line2D at 0x1f5ff6d2d90>,
  <matplotlib.lines.Line2D at 0x1f5ff701250>,
  <matplotlib.lines.Line2D at 0x1f5ff7015b0>,
  <matplotlib.lines.Line2D at 0x1f5ff70da30>,
  <matplotlib.lines.Line2D at 0x1f5ff70dd90>,
  <matplotlib.lines.Line2D at 0x1f5ff725250>,
  <matplotlib.lines.Line2D at 0x1f5ff7255b0>,
  <matplotlib.lines.Line2D at 0x1f5ff72fa60>,
  <matplotlib.lines.Line2D at 0x1f5ff72fdc0>],
 'caps': [<matplotlib.lines.Line2D at 0x1f5ff6b0100>,
  <matplotlib.lines.Line2D at 0x1f5ff6b0460>,
  <matplotlib.lines.Line2D at 0x1f5ff6bc910>,
  <matplotlib.lines.Line2D at 0x1f5ff6bcc70>,
  <matplotlib.lines.Line2D at 0x1f5ff6d3130>,
  <matplotlib.lines.Line2D at 0x1f5ff6d3490>,
  <matplotlib.lines.Line2D at 0x1f5ff6df910>,
  <matplotlib.lines.Line2D at 0x1f5ff6dfc70>,
  <matplotlib.lines.Line2D at 0x1f5ff6f7130>,
  <matplotlib.lines.Line2D at 0x1f5ff6f7490>,
  <matplotlib.lines.Line2D at 0x1f5ff701910>,
  <matplotlib.lines.Line2D at 0x1f5ff701c70>,
  <matplotlib.lines.Line2D at 0x1f5ff718130>,
  <matplotlib.lines.Line2D at 0x1f5ff718490>,
  <matplotlib.lines.Line2D at 0x1f5ff725910>,
  <matplotlib.lines.Line2D at 0x1f5ff725c70>,
  <matplotlib.lines.Line2D at 0x1f5ff73c160>,
  <matplotlib.lines.Line2D at 0x1f5ff73c4c0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1f5ff69d6a0>,
  <matplotlib.lines.Line2D at 0x1f5ff6b0eb0>,
  <matplotlib.lines.Line2D at 0x1f5ff6c66d0>,
  <matplotlib.lines.Line2D at 0x1f5ff6d3eb0>,
  <matplotlib.lines.Line2D at 0x1f5ff6d26d0>,
  <matplotlib.lines.Line2D at 0x1f5ff6f7eb0>,
  <matplotlib.lines.Line2D at 0x1f5ff70d6d0>,
  <matplotlib.lines.Line2D at 0x1f5ff718eb0>,
  <matplotlib.lines.Line2D at 0x1f5ff72f6d0>],
 'medians': [<matplotlib.lines.Line2D at 0x1f5ff6b07c0>,
  <matplotlib.lines.Line2D at 0x1f5ff6bcfd0>,
  <matplotlib.lines.Line2D at 0x1f5ff6d37f0>,
  <matplotlib.lines.Line2D at 0x1f5ff6dffd0>,
  <matplotlib.lines.Line2D at 0x1f5ff6f77f0>,
  <matplotlib.lines.Line2D at 0x1f5ff701fd0>,
  <matplotlib.lines.Line2D at 0x1f5ff7187f0>,
  <matplotlib.lines.Line2D at 0x1f5ff725fd0>,
  <matplotlib.lines.Line2D at 0x1f5ff73c820>],
 'fliers': [<matplotlib.lines.Line2D at 0x1f5ff6b0b20>,
  <matplotlib.lines.Line2D at 0x1f5ff6c6370>,
  <matplotlib.lines.Line2D at 0x1f5ff6d3b50>,
  <matplotlib.lines.Line2D at 0x1f5ff6d2370>,
  <matplotlib.lines.Line2D at 0x1f5ff6f7b50>,
  <matplotlib.lines.Line2D at 0x1f5ff70d370>,
  <matplotlib.lines.Line2D at 0x1f5ff718b50>,
  <matplotlib.lines.Line2D at 0x1f5ff72f370>,
  <matplotlib.lines.Line2D at 0x1f5ff73cb80>],
 'means': []}
```



In [13]:

```
countplot = sns.countplot(x=df['Outcome'], data=df, palette='Set1')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.title('Count Plot for Outcome')
plt.show()
```

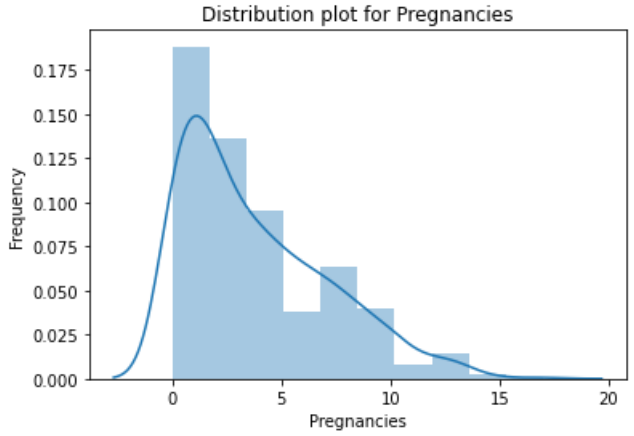


Dataset is not imbalanced

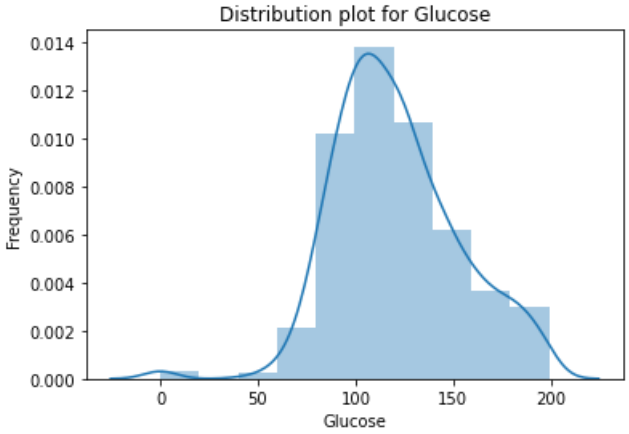
In [14]:

```
# Visualization of the features of the dataset using Distribution Plots
for cols in df.columns:
    distributionplot = sns.distplot(a=df[cols], bins=10, hist=True, kde=True)
    plt.xlabel(cols)
    plt.ylabel('Frequency')
    plt.title('Distribution plot for ' + cols)
    plt.show()
```

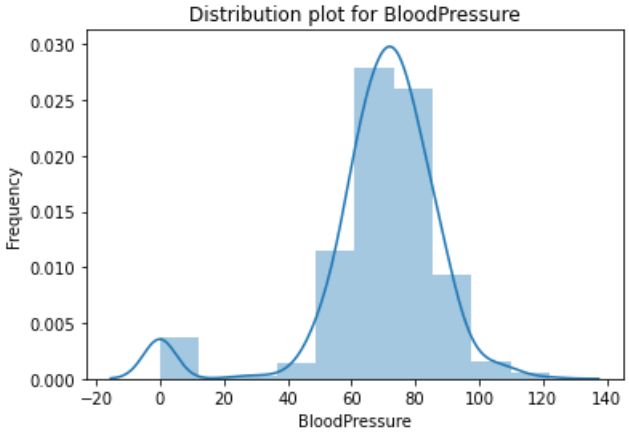
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a depreca
ted function and will be removed in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



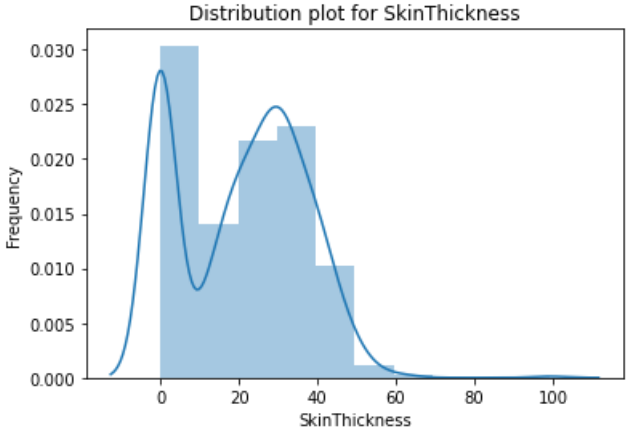

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a depreca
ted function and will be removed in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



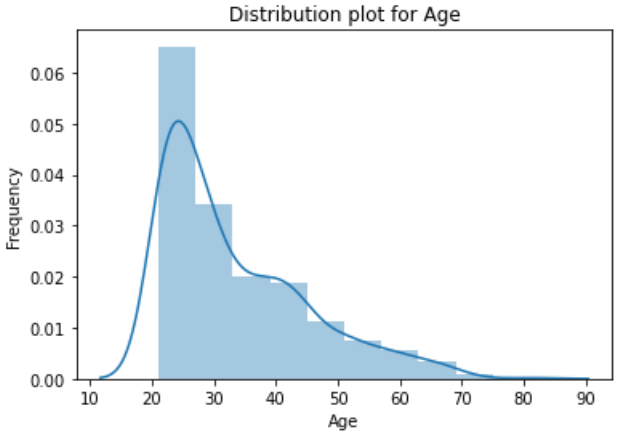

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



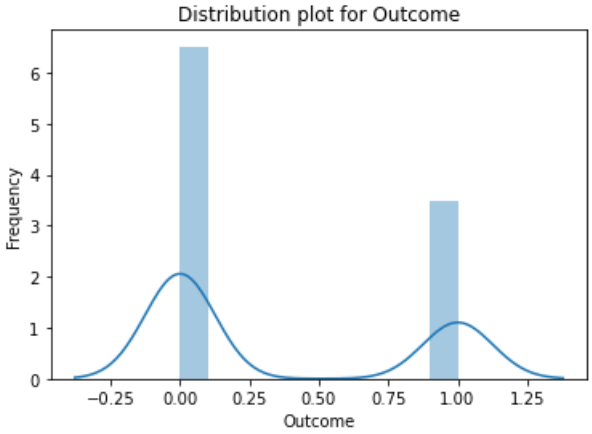

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Step 4: Data Preprocessing

Data Preprocessing on Train Data

In [15]:
```
# Input DataFrame
X=df.iloc[:,0:-1]
X.head()
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [16]:
```
# Target DataFrame
Y=df.iloc[:,-1]
Y.head()
```

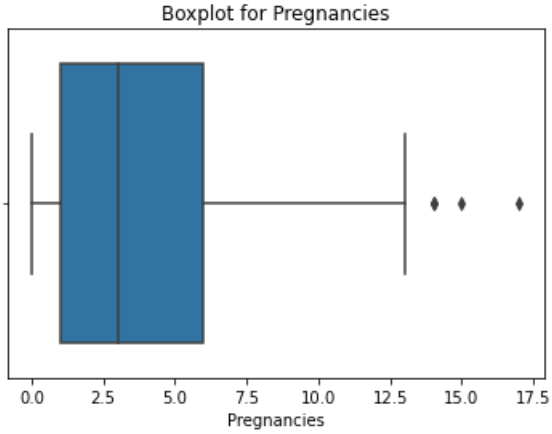
Out[16]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

In [17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

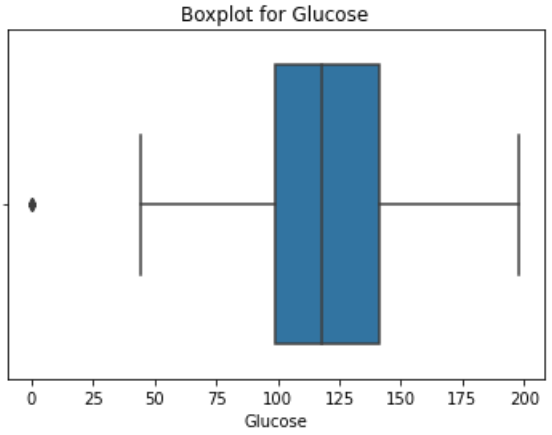
In [18]:
```
for cols in X_train.columns:
    bp = sns.boxplot(X_train[cols])
    plt.xlabel(cols)
```

```
    plt.title('Boxplot for ' + cols)
    plt.show()
```

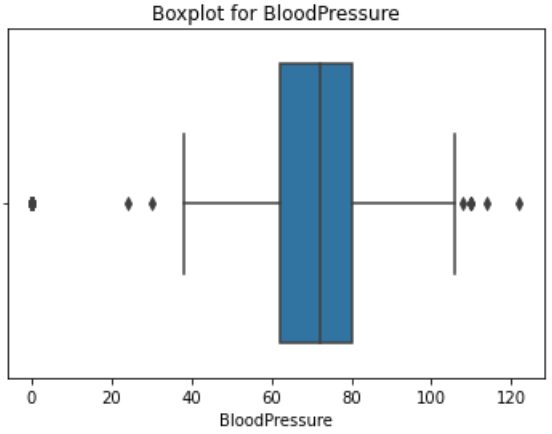
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



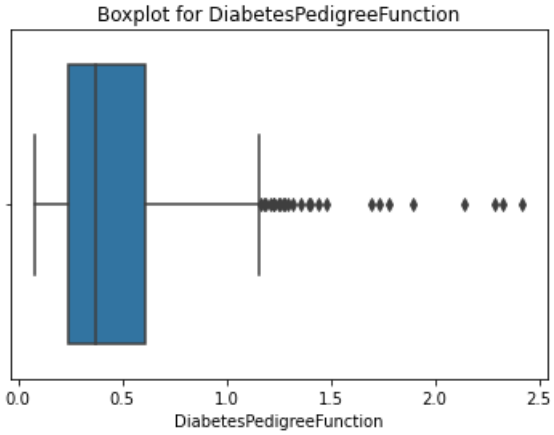

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



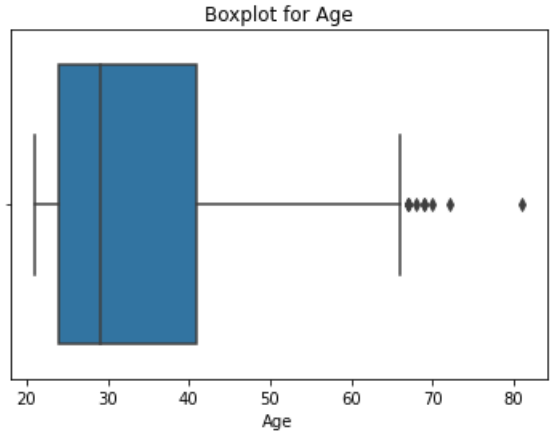

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```




```
C:\Users\HP\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(
```




```
C:\Users\HP\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(
```



In [19]:

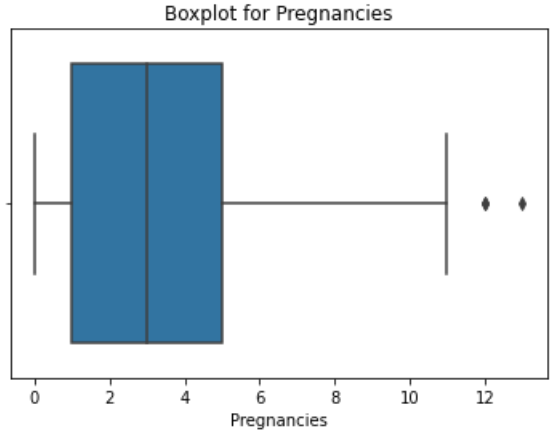
```
# Handling the outliers using quantile-based Capping and Flooring techniqueupper_limit = X_train['bmi'].quant
for col in X_train.columns:
    upper_limit = X_train[col].quantile(0.99)
    lower_limit = X_train[col].quantile(0.01)
    X_train[col] = np.where(X_train[col] >= upper_limit, upper_limit,
            np.where(X_train[col] <= lower_limit, lower_limit, X_train[col]))
```

Data Preprocessing on Testing Data

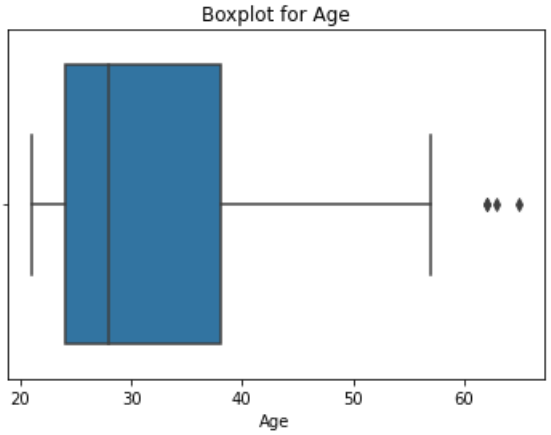
In [20]:

```
for cols in X_test.columns:
    bp = sns.boxplot(X_test[cols])
    plt.xlabel(cols)
    plt.title('Boxplot for ' + cols)
    plt.show()
```

```
C:\Users\HP\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(
```




```
C:\Users\HP\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
```



In [21]:

```
# Handling the outliers using quantile-based Capping and Flooring techniqueupper_limit = X_train['bmi'].quant
for col in X_test.columns:
    upper_limit = X_test[col].quantile(0.99)
    lower_limit = X_test[col].quantile(0.01)
    X_test[col] = np.where(X_test[col] >= upper_limit, upper_limit,
            np.where(X_test[col] <= lower_limit, lower_limit, X_test[col]))
```

Step 5: Model Building

Decision Tree

In [22]:

```
from sklearn import tree
model_decision_tree = tree.DecisionTreeClassifier()
model_decision_tree.fit(X_train,Y_train)

Y_pred_train_decision_tree = model_decision_tree.predict(X_train)
Y_pred_test_decision_tree = model_decision_tree.predict(X_test)
```

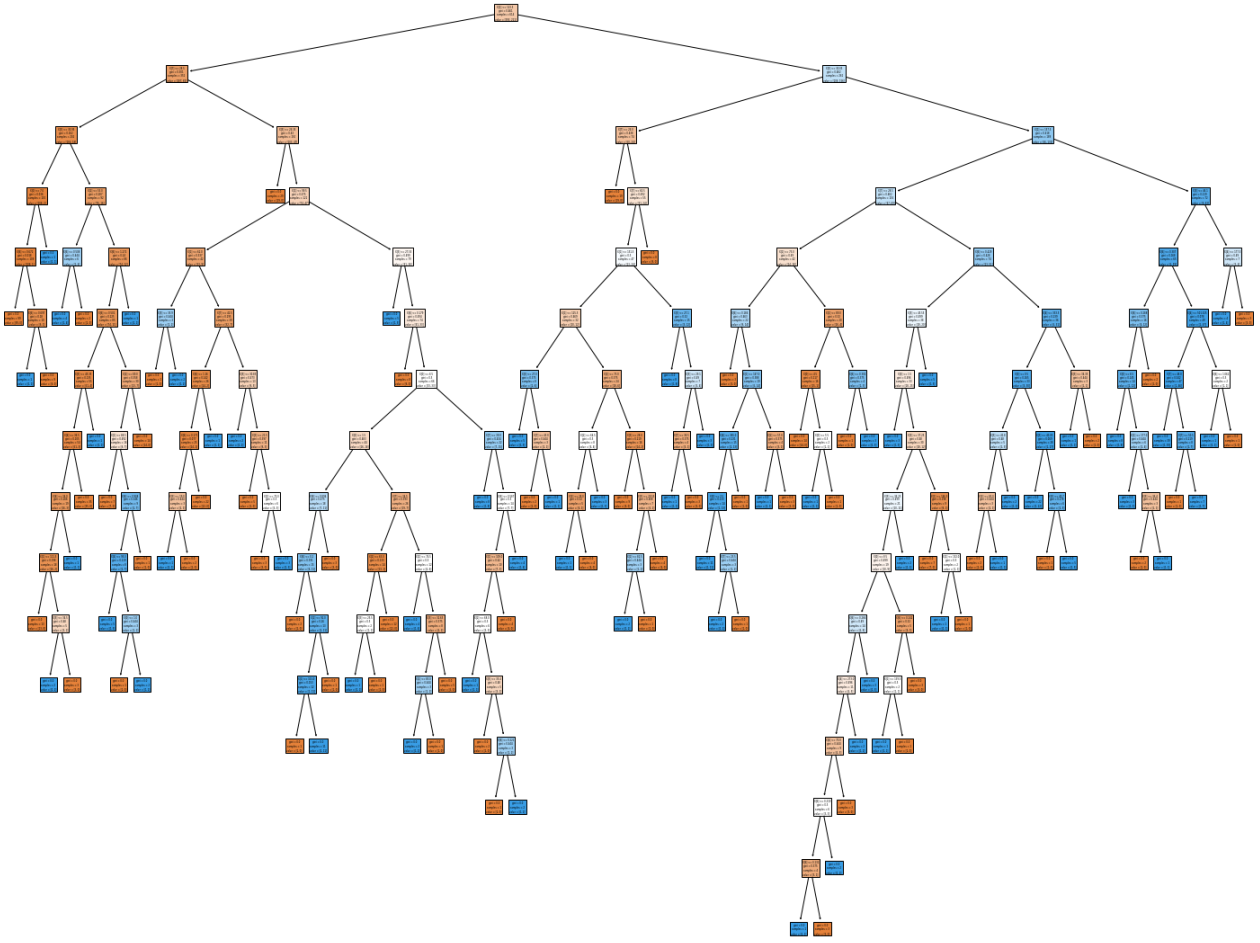
In [23]:

```
fig = plt.figure(figsize=(25,20))

tree.plot_tree(model_decision_tree, filled=True)
```

Out[23]:

```
[Text(0.4017231308411215, 0.96875, 'X[1] <= 123.5\ngini = 0.461\nsamples = 614\nvalue = [393, 221]'),
 Text(0.13960280373831777, 0.90625, 'X[7] <= 28.5\ngini = 0.301\nsamples = 352\nvalue = [287, 65]'),
 Text(0.0514018691588785, 0.84375, 'X[5] <= 30.95\ngini = 0.162\nsamples = 202\nvalue = [184, 18]'),
 Text(0.028037383177570093, 0.78125, 'X[0] <= 7.0\ngini = 0.036\nsamples = 110\nvalue = [108, 2]'),
 Text(0.018691588785046728, 0.71875, 'X[6] <= 0.672\ngini = 0.018\nsamples = 109\nvalue = [108, 1]'),
 Text(0.009345794392523364, 0.65625, 'gini = 0.0\nsamples = 99\nvalue = [99, 0]'),
 Text(0.028037383177570093, 0.65625, 'X[6] <= 0.697\ngini = 0.18\nsamples = 10\nvalue = [9, 1]'),
 Text(0.018691588785046728, 0.59375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.037383177570093455, 0.59375, 'gini = 0.0\nsamples = 9\nvalue = [9, 0]'),
 Text(0.037383177570093455, 0.71875, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.07476635514018691, 0.78125, 'X[2] <= 53.0\ngini = 0.287\nsamples = 92\nvalue = [76, 16]'),
 Text(0.056074766355140186, 0.71875, 'X[6] <= 0.508\ngini = 0.444\nsamples = 6\nvalue = [2, 4]'),
 Text(0.04672897196261682, 0.65625, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.06542056074766354, 0.65625, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.09345794392523364, 0.71875, 'X[6] <= 1.272\ngini = 0.24\nsamples = 86\nvalue = [74, 12]'),
 Text(0.08411214953271028, 0.65625, 'X[6] <= 0.501\ngini = 0.225\nsamples = 85\nvalue = [74, 11]'),
 Text(0.06542056074766354, 0.59375, 'X[5] <= 45.35\ngini = 0.135\nsamples = 55\nvalue = [51, 4]'),
 Text(0.056074766355140186, 0.53125, 'X[4] <= 36.5\ngini = 0.105\nsamples = 54\nvalue = [51, 3]'),
 Text(0.04672897196261682, 0.46875, 'X[4] <= 34.0\ngini = 0.266\nsamples = 19\nvalue = [16, 3]'),
 Text(0.037383177570093455, 0.40625, 'X[1] <= 111.5\ngini = 0.198\nsamples = 18\nvalue = [16, 2]'),
 Text(0.028037383177570093, 0.34375, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
 Text(0.04672897196261682, 0.34375, 'X[5] <= 34.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.037383177570093455, 0.28125, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.056074766355140186, 0.28125, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.056074766355140186, 0.40625, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.06542056074766354, 0.46875, 'gini = 0.0\nsamples = 35\nvalue = [35, 0]'),
 Text(0.07476635514018691, 0.53125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.102803738317757, 0.59375, 'X[2] <= 69.0\ngini = 0.358\nsamples = 30\nvalue = [23, 7]'),
 Text(0.09345794392523364, 0.53125, 'X[1] <= 88.5\ngini = 0.492\nsamples = 16\nvalue = [9, 7]'),
 Text(0.08411214953271028, 0.46875, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
 Text(0.102803738317757, 0.46875, 'X[6] <= 0.908\ngini = 0.346\nsamples = 9\nvalue = [2, 7]'),
 Text(0.09345794392523364, 0.40625, 'X[4] <= 95.0\ngini = 0.219\nsamples = 8\nvalue = [1, 7]'),
 Text(0.08411214953271028, 0.34375, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.102803738317757, 0.34375, 'X[0] <= 1.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.09345794392523364, 0.28125, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.11214953271028037, 0.28125, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.11214953271028037, 0.40625, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.11214953271028037, 0.53125, 'gini = 0.0\nsamples = 14\nvalue = [14, 0]'),
 Text(0.102803738317757, 0.65625, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.22780373831775702, 0.84375, 'X[5] <= 26.35\ngini = 0.43\nsamples = 150\nvalue = [103, 47]'),
```

```
Text(0.9158878504672897, 0.59375, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.9532710280373832, 0.65625, 'X[4] <= 501.525\ngini = 0.078\nsamples = 49\nvalue = [2, 47]'),
Text(0.9345794392523364, 0.59375, 'X[7] <= 48.0\ngini = 0.042\nsamples = 47\nvalue = [1, 46]'),
Text(0.9252336448598131, 0.53125, 'gini = 0.0\nsamples = 39\nvalue = [0, 39]'),
Text(0.9439252336448598, 0.53125, 'X[7] <= 50.5\ngini = 0.219\nsamples = 8\nvalue = [1, 7]'),
Text(0.9345794392523364, 0.46875, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.9532710280373832, 0.46875, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(0.9719626168224299, 0.59375, 'X[6] <= 1.062\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.9626168224299065, 0.53125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.9813084112149533, 0.53125, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.9813084112149533, 0.71875, 'X[4] <= 177.5\ngini = 0.49\nsamples = 7\nvalue = [3, 4]'),
Text(0.9719626168224299, 0.65625, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(0.9906542056074766, 0.65625, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]')]
```



Naive Bayes

In [26]:

```
from sklearn.naive_bayes import GaussianNB
model_naive_bayes = GaussianNB()

model_naive_bayes.fit(X_train,Y_train)

Y_pred_train_naive_bayes = model_naive_bayes.predict(X_train)
Y_pred_test_naive_bayes = model_naive_bayes.predict(X_test)
```

Step 6: Model Evaluation

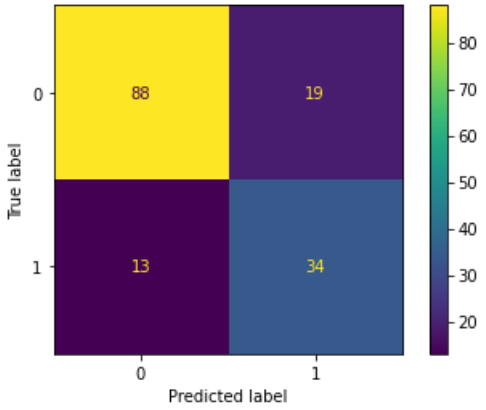
Decision trees

In [33]:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, recall_score, f1_score,
print('\nConfusion Matrix:\n')
plot_confusion_matrix(model_decision_tree, X_test, Y_test)

print('Train accuracy: ' + str(accuracy_score(Y_train, Y_pred_train_decision_tree)))
print('Test accuracy: ' + str(accuracy_score(Y_test, Y_pred_test_decision_tree)))
```

```
Confusion Matrix:

Train accuracy: 1.0
Test accuracy: 0.7922077922077922
C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusi
on_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Us
```

```
e one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [28]:
```
print('\nClassification Report:\n')
print(classification_report(Y_test, Y_pred_test_decision_tree))
```

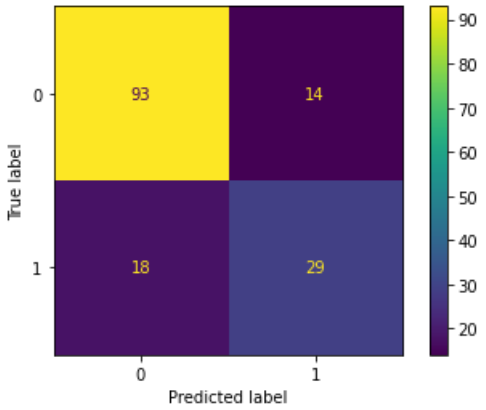
```
Classification Report:

              precision    recall  f1-score   support

           0       0.87      0.82      0.85       107
           1       0.64      0.72      0.68        47

    accuracy                           0.79       154
   macro avg       0.76      0.77      0.76       154
weighted avg       0.80      0.79      0.80       154
```

Naive Bayes

In [32]:
```
print('Train accuracy: ' + str(accuracy_score(Y_train, Y_pred_train_naive_bayes)))
print('Test accuracy: ' + str(accuracy_score(Y_test, Y_pred_test_naive_bayes)))

print('\nConfusion Matrix:\n')
plot_confusion_matrix(model_naive_bayes,X_test, Y_test)
```

```
Train accuracy: 0.758957654723127
Test accuracy: 0.7922077922077922

Confusion Matrix:

C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusi
on_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Us
e one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[32]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f5917d79d0>



In [31]:
```
print('\nClassification Report:\n')
print(classification_report(Y_test, Y_pred_test_naive_bayes))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.84      0.87      0.85       107
           1       0.67      0.62      0.64        47
```

```
              precision    recall  f1-score   support

    accuracy                         0.79       154
   macro avg       0.76      0.74      0.75       154
weighted avg       0.79      0.79      0.79       154
```

Since it is a healthcare problem statement, recall is most important performance metric.

Recall of Decision Tree = 82% Recall of Naive Bayes = 87%

EXPERIMENT 10

Step 1: Problem Statement

Perform K-Means on the "Mall_Customer.csv"

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Data Collection

In [2]:
```
df = pd.read_csv(r'./datasets/Mall_Customers.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Step 3: Exploratory Data Analysis

In [4]:
```
df.shape
```

Out[4]:
```
(200, 5)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [7]:
```
df.dtypes
```

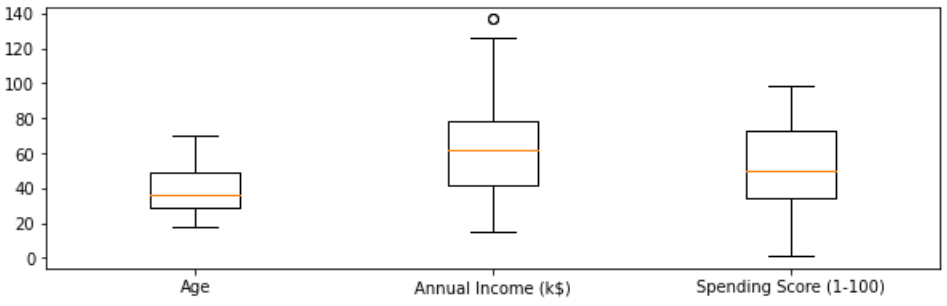
Out[7]:
```
CustomerID                int64
Genre                    object
Age                       int64
Annual Income (k$)        int64
Spending Score (1-100)    int64
dtype: object
```

In [8]:
```
# Checking for NaN values in the dataset
df.isnull().sum()
```

Out[8]:
```
CustomerID                0
Genre                     0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

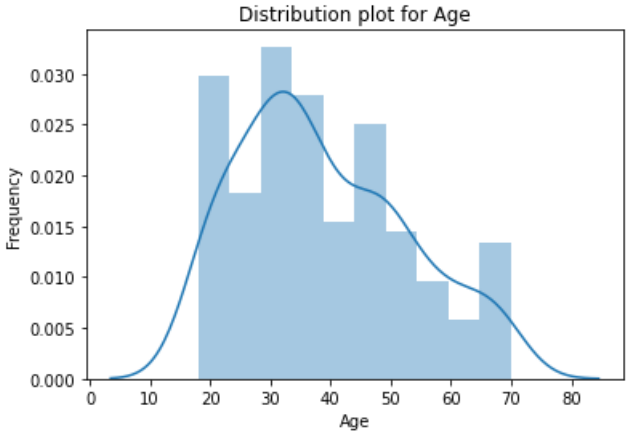
In [9]:
```
# Visualizing the outliers
fig=plt.figure(figsize=(10,3))
plt.boxplot(df[["Age", "Annual Income (k$)", "Spending Score (1-100)"]], labels = ["Age", "Annual Income (k$)
```

Out[9]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2679d8b3220>,
  <matplotlib.lines.Line2D at 0x2679d8b3580>,
  <matplotlib.lines.Line2D at 0x2679f8c1a30>,
  <matplotlib.lines.Line2D at 0x2679f8c1d90>,
  <matplotlib.lines.Line2D at 0x2679f8d9250>,
  <matplotlib.lines.Line2D at 0x2679f8d95b0>],
 'caps': [<matplotlib.lines.Line2D at 0x2679d8b38e0>,
  <matplotlib.lines.Line2D at 0x2679d8b3c40>,
  <matplotlib.lines.Line2D at 0x2679f8ce130>,
  <matplotlib.lines.Line2D at 0x2679f8ce490>,
  <matplotlib.lines.Line2D at 0x2679f8d9910>,
  <matplotlib.lines.Line2D at 0x2679f8d9c70>],
 'boxes': [<matplotlib.lines.Line2D at 0x2679d89ee80>,
  <matplotlib.lines.Line2D at 0x2679f8c16d0>,
  <matplotlib.lines.Line2D at 0x2679f8ceeb0>],
 'medians': [<matplotlib.lines.Line2D at 0x2679d8b3fa0>,
  <matplotlib.lines.Line2D at 0x2679f8ce7f0>,
  <matplotlib.lines.Line2D at 0x2679f8d9fd0>],
 'fliers': [<matplotlib.lines.Line2D at 0x2679f8c1340>,
  <matplotlib.lines.Line2D at 0x2679f8ceb50>,
  <matplotlib.lines.Line2D at 0x2679f8e4370>],
 'means': []}
```

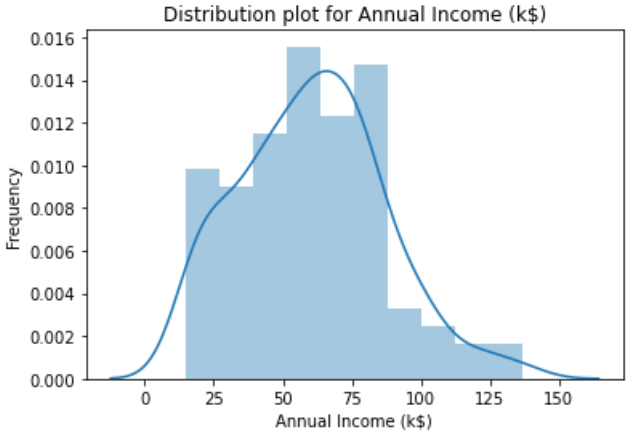


In [10]:
```
# Visualization of the features of the dataset using Distribution Plots
for cols in ["Age", "Annual Income (k$)", "Spending Score (1-100)"]:
    distributionplot = sns.distplot(a=df[cols], bins=10, hist=True, kde=True)
    plt.xlabel(cols)
    plt.ylabel('Frequency')
    plt.title('Distribution plot for ' + cols)
    plt.show()
```

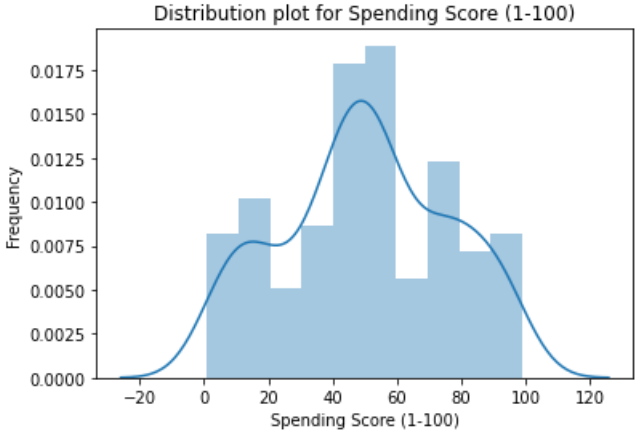
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a depreca
ted function and will be removed in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



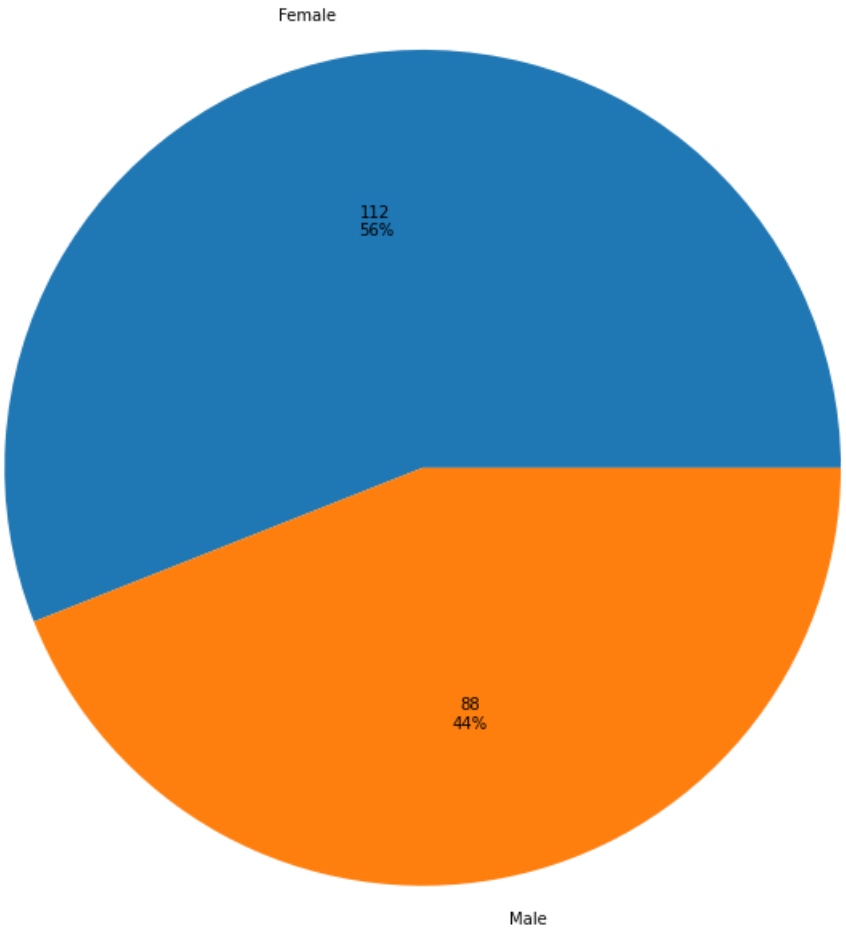

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Visualizing the Gender feature using a pie chart

In [11]:

```
def label_function(val):
    return f'{val / 100 * len(df):.0f}\n{val:.0f}%'

X=df.groupby('Genre').size()
mylabels=['Female','Male']
plt.pie(X, labels = mylabels,radius=3,autopct=label_function)

plt.show()
```



Step 4: Data Preprocessing

Label Encoding

In [12]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Genre'] = le.fit_transform(df['Genre'])
```

In [13]:
```
df.head()
```

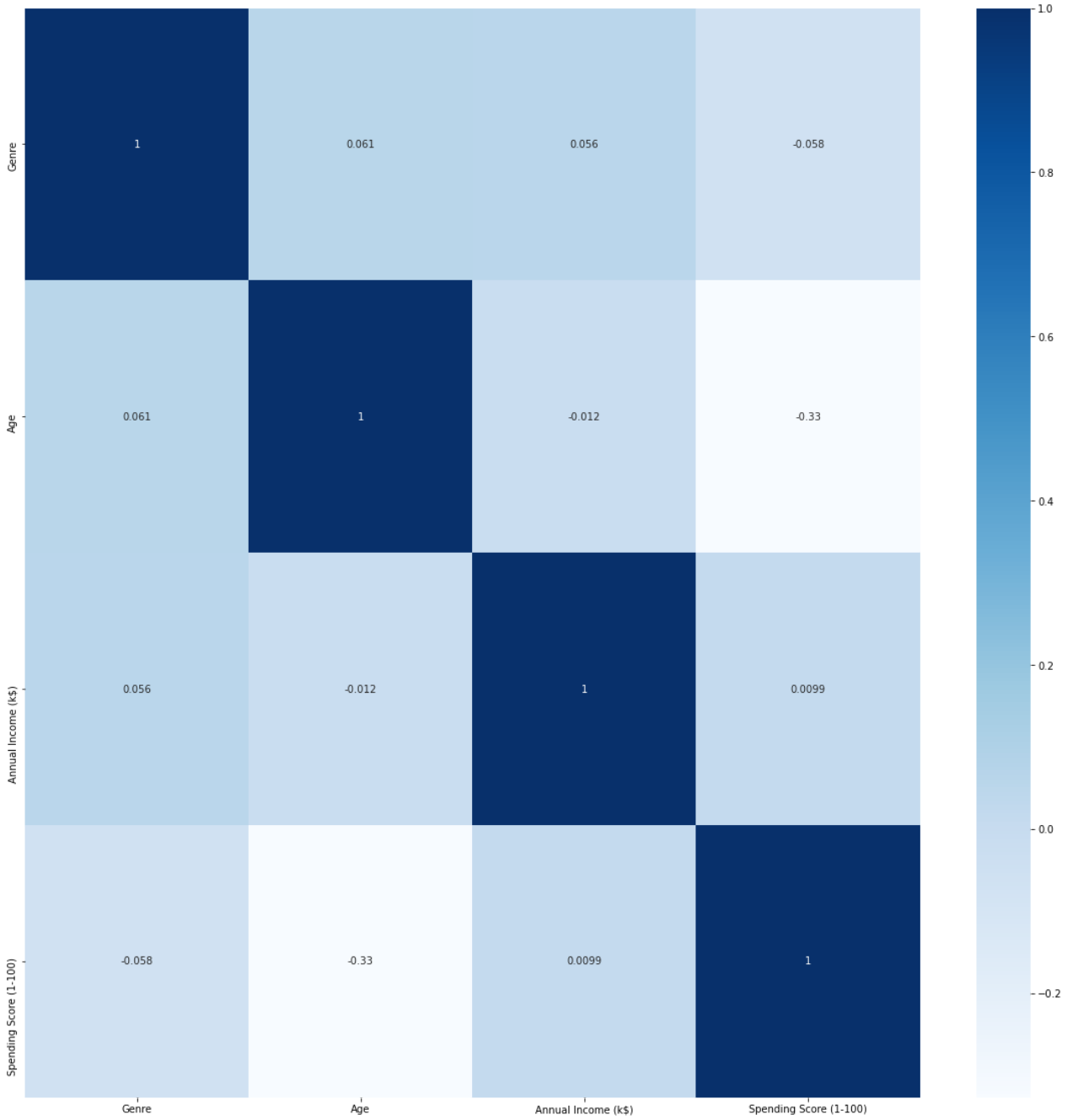
Out[13]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40

In [14]:
```
df.drop(['CustomerID'], axis = 1, inplace = True)
```

In [15]:
```
# Correlation matrix
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), annot=True, cmap="Blues")
```

Out[15]: <AxesSubplot:>



Feature Scaling (standardizing the features)

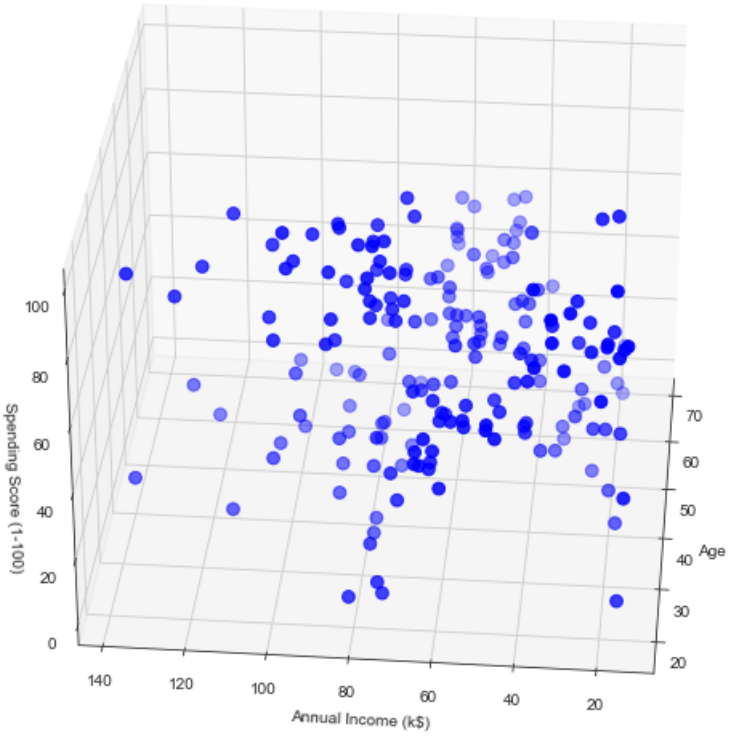
In [16]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit_transform(df)
```

Out[16]:
```
array([[ 1.12815215, -1.42456879, -1.73899919, -0.43480148],
       [ 1.12815215, -1.28103541, -1.73899919,  1.19570407],
       [-0.88640526, -1.3528021 , -1.70082976, -1.71591298],
       [-0.88640526, -1.13750203, -1.70082976,  1.04041783],
       [-0.88640526, -0.56336851, -1.66266033, -0.39597992],
       [-0.88640526, -1.20926872, -1.66266033,  1.00159627],
       [-0.88640526, -0.27630176, -1.62449091, -1.71591298],
       [-0.88640526, -1.13750203, -1.62449091,  1.70038436],
       [ 1.12815215,  1.80493225, -1.58632148, -1.83237767],
       [-0.88640526, -0.6351352 , -1.58632148,  0.84631002],
       [ 1.12815215,  2.02023231, -1.58632148, -1.4053405 ],
       [-0.88640526, -0.27630176, -1.58632148,  1.89449216],
       [-0.88640526,  1.37433211, -1.54815205, -1.36651894],
       [-0.88640526, -1.06573534, -1.54815205,  1.04041783],
       [ 1.12815215, -0.13276838, -1.54815205, -1.44416206],
       [ 1.12815215, -1.20926872, -1.54815205,  1.11806095],
       [-0.88640526, -0.27630176, -1.50998262, -0.59008772],
       [ 1.12815215, -1.3528021 , -1.50998262,  0.61338066],
       [ 1.12815215,  0.94373197, -1.43364376, -0.82301709],
       [-0.88640526, -0.27630176, -1.43364376,  1.8556706 ],
       [ 1.12815215, -0.27630176, -1.39547433, -0.59008772],
       [ 1.12815215, -0.99396865, -1.39547433,  0.88513158],
```

```
[-0.88640526,  0.44136514,  2.49780745, -0.86183865],
[ 1.12815215, -0.49160182,  2.49780745,  0.92395314],
[ 1.12815215, -0.49160182,  2.91767117, -1.25005425],
[ 1.12815215, -0.6351352 ,  2.91767117,  1.27334719]])
```

In [17]:

```
from mpl_toolkits.mplot3d import Axes3D

sns.set_style("white")
fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df.Age, df["Annual Income (k$)"], df["Spending Score (1-100)"], c='blue', s=60)
ax.view_init(30, 185)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.show()
```



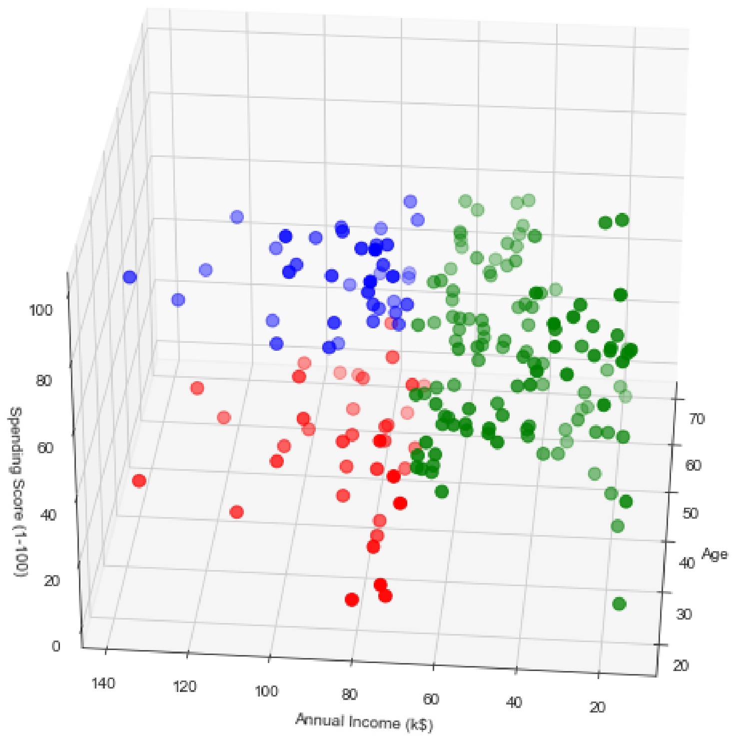
Step 5: Model Building

In [21]:

```
km = KMeans(n_clusters=3)
clusters = km.fit_predict(df.iloc[:,1:])

df["label"] = clusters

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df.Age[df.label == 0], df["Annual Income (k$)"][df.label == 0], df["Spending Score (1-100)"][df.la
ax.scatter(df.Age[df.label == 1], df["Annual Income (k$)"][df.label == 1], df["Spending Score (1-100)"][df.la
ax.scatter(df.Age[df.label == 2], df["Annual Income (k$)"][df.label == 2], df["Spending Score (1-100)"][df.la
ax.view_init(30, 185)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
```

ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

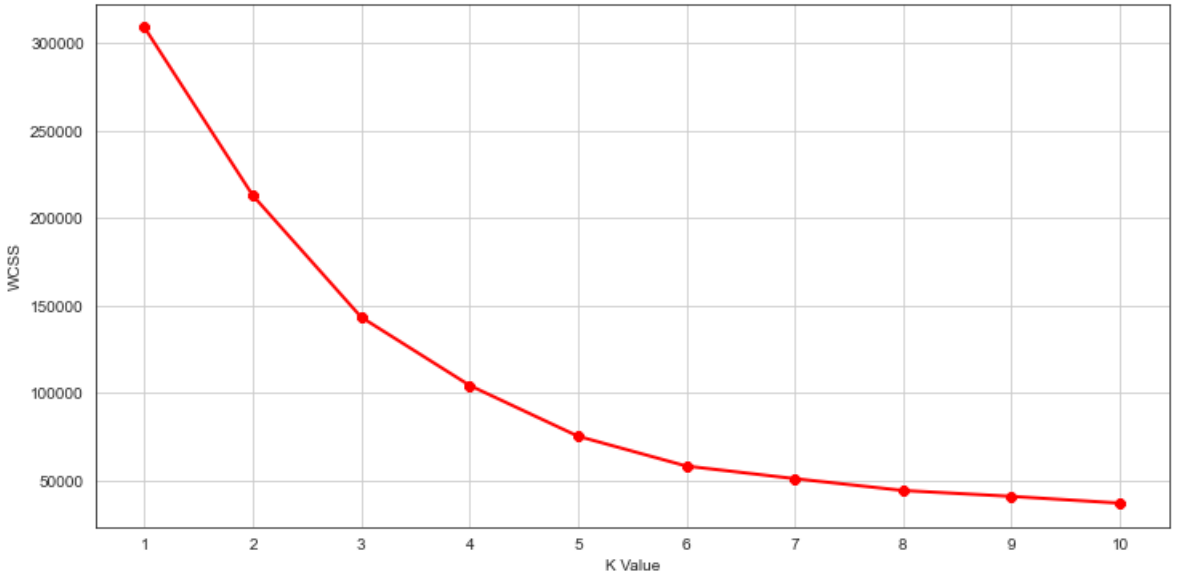


In [19]:

```
from sklearn.cluster import KMeans

wcss = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(df.iloc[:,1:])
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(12,6))
plt.grid()
plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1))
plt.ylabel("WCSS")
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
```

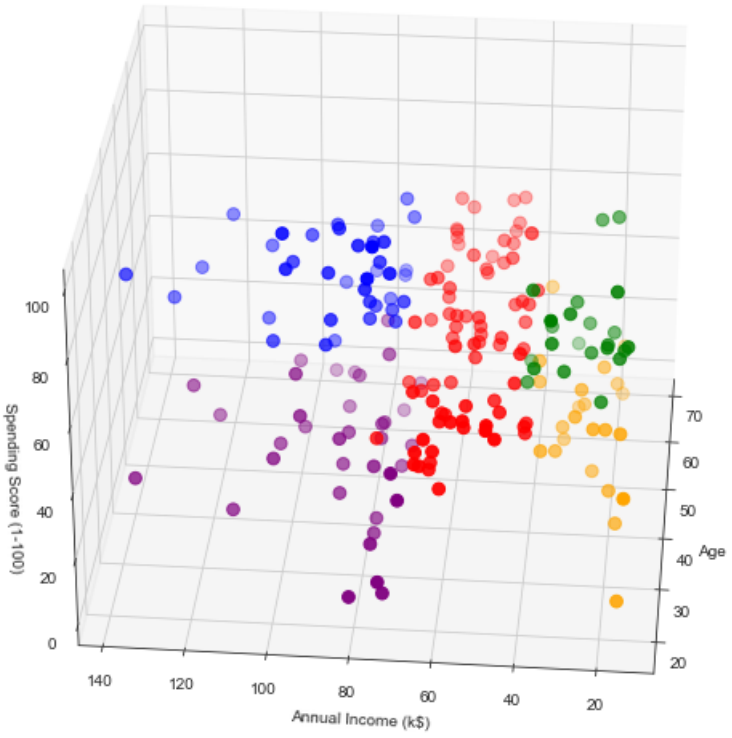
```
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [20]:

```
km = KMeans(n_clusters=5)
clusters = km.fit_predict(df.iloc[:,1:])

df["label"] = clusters

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df.Age[df.label == 0], df["Annual Income (k$)"][df.label == 0], df["Spending Score (1-100)"][df.la
ax.scatter(df.Age[df.label == 1], df["Annual Income (k$)"][df.label == 1], df["Spending Score (1-100)"][df.la
ax.scatter(df.Age[df.label == 2], df["Annual Income (k$)"][df.label == 2], df["Spending Score (1-100)"][df.la
ax.scatter(df.Age[df.label == 3], df["Annual Income (k$)"][df.label == 3], df["Spending Score (1-100)"][df.la
ax.scatter(df.Age[df.label == 4], df["Annual Income (k$)"][df.label == 4], df["Spending Score (1-100)"][df.la
ax.view_init(30, 185)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)')
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have
a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```


Spending Score (1-100)
100
80
60
40
20
0
140
120
100
80
60
40
20
Annual Income (k$)
70
60
50
40
30
20
Age

OPEN-ENDED EXPERIEMENT

EXPERIMENT 11

Step 1: Problem Statement

Perform dimensionality reduction using Principal Component Analysis

In [28]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Data collection

In [2]:
```
from sklearn.datasets import load_breast_cancer
```

In [3]:
```
cancer=load_breast_cancer()
```

In [4]:
```
cancer.keys()
```

Out[4]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

In [5]:
```
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
```

```
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ========================================= ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

In [6]:
```
df=pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
```

Step 3: Exploratory Data Analysis

In [7]:
```
df.head(5)
```

Out[7]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	p
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	p
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	

5 rows × 30 columns

In [21]:
```
df.shape
```

Out[21]:
```
(569, 30)
```

In [22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [23]:
```
df.describe()
```

Out[23]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440

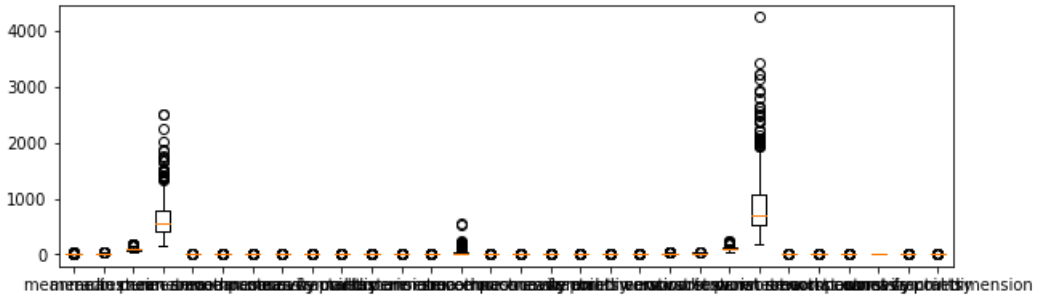
8 rows × 30 columns

In [24]:
```
# Checking for NaN values in the dataset
df.isnull().sum()
```

Out[24]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
dtype: int64
```

In [26]:
```
# Visualizing the outliers
fig=plt.figure(figsize=(10,3))
plt.boxplot(df, labels =df.columns)
```

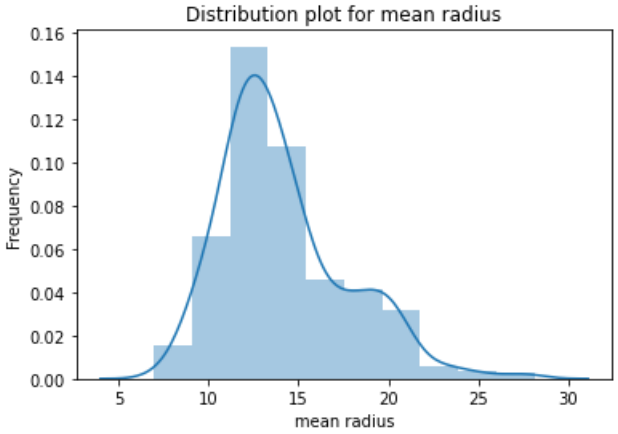
Out[26]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x29c7fc0fb80>,
  <matplotlib.lines.Line2D at 0x29c7fc0fee0>,
  <matplotlib.lines.Line2D at 0x29c7fc2d3a0>,
  <matplotlib.lines.Line2D at 0x29c7fc2d700>,
  <matplotlib.lines.Line2D at 0x29c7fc38b80>,
  <matplotlib.lines.Line2D at 0x29c7fc38ee0>,
  <matplotlib.lines.Line2D at 0x29c7fc503a0>,
  <matplotlib.lines.Line2D at 0x29c7fc50700>,
  <matplotlib.lines.Line2D at 0x29c7fc5bb80>,
  <matplotlib.lines.Line2D at 0x29c7fc5bee0>,
  <matplotlib.lines.Line2D at 0x29c7fc733a0>,
  <matplotlib.lines.Line2D at 0x29c7fc73700>,
  <matplotlib.lines.Line2D at 0x29c7fc7eb80>,
  <matplotlib.lines.Line2D at 0x29c7fc7eee0>,
  <matplotlib.lines.Line2D at 0x29c7fc973a0>,
  <matplotlib.lines.Line2D at 0x29c7fc97700>,
  <matplotlib.lines.Line2D at 0x29c7fca3b80>,
  <matplotlib.lines.Line2D at 0x29c7fca3ee0>,
  <matplotlib.lines.Line2D at 0x29c7fcb93a0>,
  <matplotlib.lines.Line2D at 0x29c7fcb9700>,
  <matplotlib.lines.Line2D at 0x29c7fcc4b80>,
  <matplotlib.lines.Line2D at 0x29c7fcc4ee0>,
  <matplotlib.lines.Line2D at 0x29c7fcd83a0>,
  <matplotlib.lines.Line2D at 0x29c7fcd8700>,
  <matplotlib.lines.Line2D at 0x29c7fce9b80>,
  <matplotlib.lines.Line2D at 0x29c7fce9ee0>,
  <matplotlib.lines.Line2D at 0x29c7fcfb3a0>,
  <matplotlib.lines.Line2D at 0x29c7fcfb700>,
  <matplotlib.lines.Line2D at 0x29c7fd0bb80>,
  <matplotlib.lines.Line2D at 0x29c7fd0bee0>,
  <matplotlib.lines.Line2D at 0x29c7fd1e3a0>,
  <matplotlib.lines.Line2D at 0x29c7fd1e700>,
  <matplotlib.lines.Line2D at 0x29c7fd2eb80>,
  <matplotlib.lines.Line2D at 0x29c7fd2eee0>,
  <matplotlib.lines.Line2D at 0x29c7fd463a0>,
  <matplotlib.lines.Line2D at 0x29c7fd46700>,
  <matplotlib.lines.Line2D at 0x29c7fd52b80>,
  <matplotlib.lines.Line2D at 0x29c7fd52ee0>,
  <matplotlib.lines.Line2D at 0x29c7fd683a0>,
  <matplotlib.lines.Line2D at 0x29c7fd68700>,
```



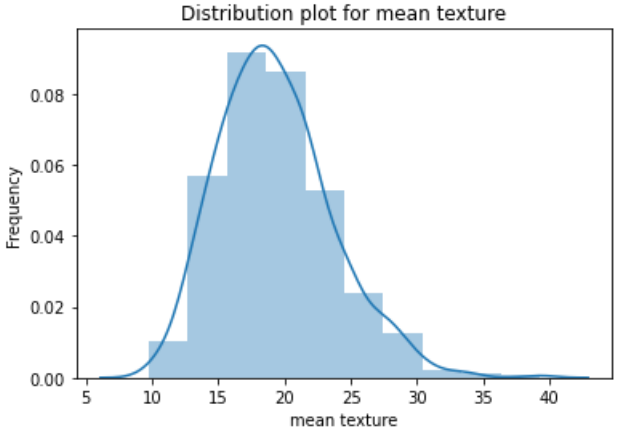
In [30]:

```
# Visualization of the features of the dataset using Distribution Plots
for cols in df.columns:
    distributionplot = sns.distplot(a=df[cols], bins=10, hist=True, kde=True)
    plt.xlabel(cols)
    plt.ylabel('Frequency')
    plt.title('Distribution plot for ' + cols)
    plt.show()
```

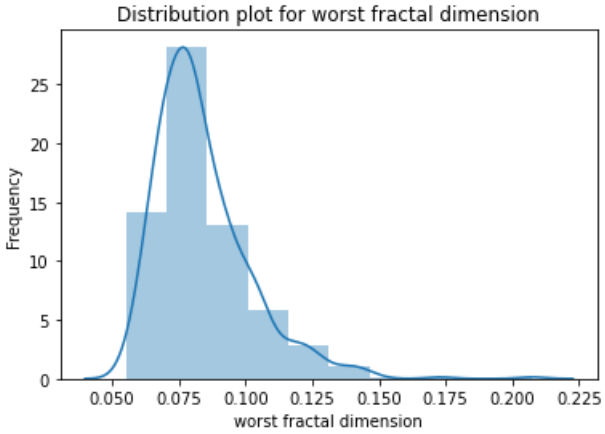
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Step 4: Data Preprocessing

In [8]:
```
from sklearn.preprocessing import StandardScaler
```

In [9]:
```
scaler=StandardScaler()
scaler.fit(df)
```

Out[9]:
```
StandardScaler()
```

In [10]:
```
scaled_data=scaler.transform(df)
```

In [11]:
```
scaled_data
```

Out[11]:
```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

Step 5: Model Building

In [12]:
```
from sklearn.decomposition import PCA
```

In [13]:
```
pca=PCA(n_components=2)
```

In [14]:
```
pca.fit(scaled_data)
```

Out[14]:
```
PCA(n_components=2)
```

In [15]:
```
x_pca=pca.transform(scaled_data)
```

In [16]:
```
scaled_data.shape
```

Out[16]:
```
(569, 30)
```

In [17]:
```
x_pca.shape
```

Out[17]: (569, 2)

In [18]:
```
scaled_data
```

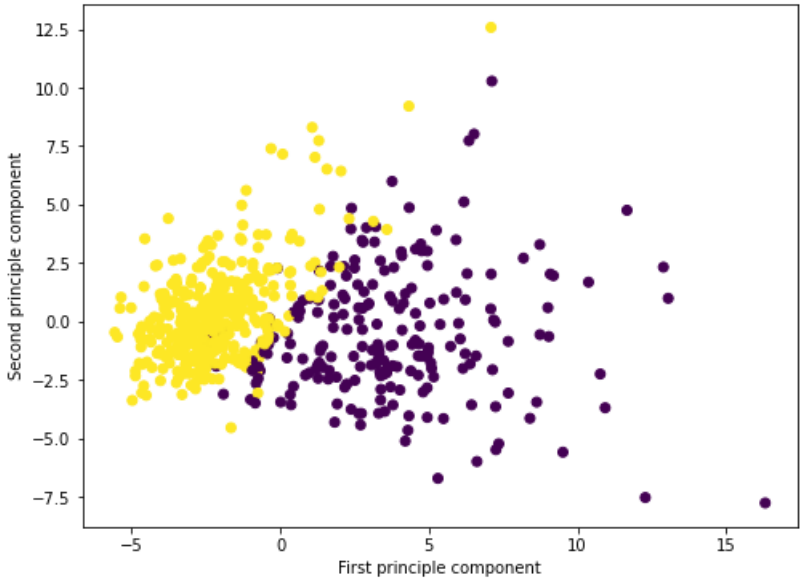
Out[18]:
```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

In [19]:
```
x_pca
```

Out[19]:
```
array([[ 9.19283683,  1.94858307],
       [ 2.3878018 , -3.76817174],
       [ 5.73389628, -1.0751738 ],
       ...,
       [ 1.25617928, -1.90229671],
       [10.37479406,  1.67201011],
       [-5.4752433 , -0.67063679]])
```

In [20]:
```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'])
plt.xlabel('First principle component')
plt.ylabel('Second principle component')
```

Out[20]: Text(0, 0.5, 'Second principle component')



OPEN-ENDED EXPERIMENT

EXPERIMENT 12

Step 1: Problem Statement

To build an e-mail spam detection system using Support Vector Machine

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
pd.set_option('display.max_columns', None)
```

Step 2: Data Collection

In [3]:
```
df = pd.read_csv('./dataset/spam.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

In [5]:
```
df = df[['v1', 'v2']]
```

In [6]:
```
df.head()
```

Out[6]:

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Step 3: Exploratory Data Analysis

In [7]:
```
df.shape
```

Out[7]:
```
(5572, 2)
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   v1      5572 non-null   object
 1   v2      5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```



```
                                    , 'v2': 'text'})
```

In [10]:
```
# Checking for NaN values in the dataset
df.isnull().sum()
```

Out[10]:
```
spam    0
text    0
dtype: int64
```

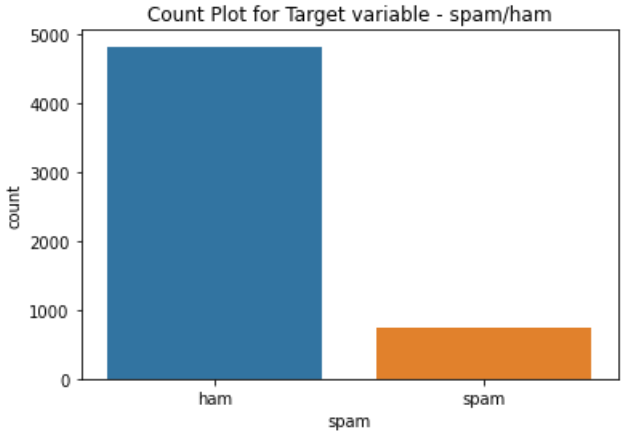
In [11]:
```
df['spam'].value_counts()
```

Out[11]:
```
ham     4825
spam     747
Name: spam, dtype: int64
```

In [12]:
```
sns.countplot(df['spam'])
plt.title('Count Plot for Target variable - spam/ham')
```

```
C:\Users\HP\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(
```

Out[12]:
```
Text(0.5, 1.0, 'Count Plot for Target variable - spam/ham')
```



Step 4: Data Preprocessing

Data Preprocessing on Train Data

In [13]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['spam'] = le.fit_transform(df['spam'])
```

In [14]:
```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[14]:
```
True
```

In [15]:
```
data_token = df

data_token['text'] = df['text'].apply(word_tokenize)
```

In [16]:
```
data_token.head()
```

Out[16]:

	spam	text
0	0	[Go, until, jurong, point, ..., crazy, ..., Avail...
1	0	[Ok, lar, ..., Joking, wif, u, oni, ...]

	spam	text
2	1	[Free, entry, in, 2, a, wkly, comp, to, win, F...
3	0	[U, dun, say, so, early, hor, ..., U, c, alrea...
4	0	[Nah, I, do, n't, think, he, goes, to, usf, ,,...

In [17]:
```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[17]: True

In [18]:
```
lemmatizer = WordNetLemmatizer()
lem_list = []

for text_token in data_token['text']:
  text_lem = [lemmatizer.lemmatize(i) for i in text_token]
  lem_list.append(text_lem)

data_lemma = data_token

data_lemma['text'] = lem_list
```

In [19]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer(stop_words='english')

text_lem = [''.join(i) for i in data_lemma['text']]
df_vec = data_lemma
df_vec['text'] = text_lem
df_vec = tv.fit_transform(df_vec['text'])
```

In [20]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, Y_train, Y_test = train_test_split(df_vec, df['spam'], train_size=0.8, random_state=1)
```

Step 5: Model Building

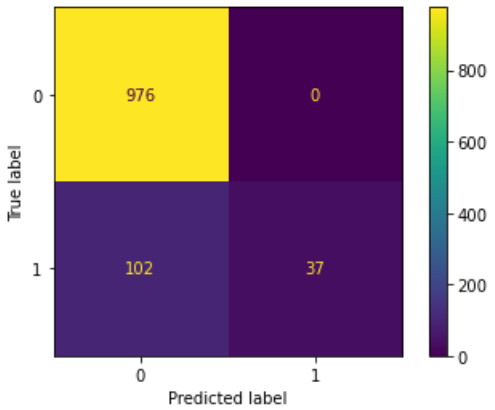
In [21]:
```
from sklearn import svm
model_svm = svm.SVC(kernel='rbf', C=1000, gamma=10, probability = True)

model_svm.fit(X_train,Y_train)

Y_pred_train_svm = model_svm.predict(X_train)
Y_pred_test_svm = model_svm.predict(X_test)
```

Step 6: Model Evaluation

In [23]:
```
from sklearn.metrics import plot_confusion_matrix
print('\nConfusion Matrix:\n')
plot_confusion_matrix(model_svm, X_test, Y_test)
```

```
Confusion Matrix:

C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusi
on_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Us
e one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[23]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x11b717a4940>



In [24]:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, recall_score, f1_score,

print('Train accuracy: ' + str(accuracy_score(Y_train, Y_pred_train_svm)))
print('Test accuracy: ' + str(accuracy_score(Y_test, Y_pred_test_svm)))
print('Recall Score: ' + str(recall_score(Y_test, Y_pred_test_svm)))
print('Precision Score: ' + str(precision_score(Y_test, Y_pred_test_svm)))
print('F1 Score: ' + str(f1_score(Y_test, Y_pred_test_svm)))
```

```
Train accuracy: 1.0
Test accuracy: 0.9085201793721973
Recall Score: 0.26618705035971224
Precision Score: 1.0
F1 Score: 0.4204545454545454
```

In [25]:

```
print('\nClassification Report:\n')
print(classification_report(Y_test, Y_pred_test_svm))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.91      1.00      0.95       976
           1       1.00      0.27      0.42       139

    accuracy                           0.91      1115
   macro avg       0.95      0.63      0.69      1115
weighted avg       0.92      0.91      0.88      1115
```

In [26]:

```
from sklearn.metrics import roc_auc_score, roc_curve, RocCurveDisplay, plot_confusion_matrix

Y_proba_test_svm = model_svm.predict_proba(X_test)
print('ROC AUC Score: ' + str(roc_auc_score(Y_test, Y_proba_test_svm[:,1])))
```

```
ROC AUC Score: 0.9507865019459842
```