PRACTICAL-1

AIM :

prepare a report how python is used for machine learning and also discus python with google colab.

Introduction to Python Programming. How python used in machine learning? Discuss python with Google Colab.

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.

Below are some facts about Python Programming Language:

- Python is currently the most widely used multi-purpose, high-level programming language.
- Python allows programming in Object-Oriented and Procedural paradigms.
- Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

Machine Learning

- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)

Web scraping (like Scrapy, BeautifulSoup, Selenium)
Test frameworks

Advantages of Python

- Extensive Libraries
- Extensible
- Embeddable
- Improved Productivity
- 5 Simple and Easy

Disadvantages of Python

- Speed Limitations
- Weak in Mobile Computing and Browsers
- Design Restrictions
- Simple

How python used in machine learning?

Python is a widely used high-level programming language for general-purpose programming. Apart from being open source programming language,

Also python is huge library support. Also this library are used in machine leaning. Here some python library which are used in machine learning.

1.Numpy :- NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

2.Matplotlib:- Matplotlib is a plotting 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+.

3. TensorFlow :- TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensorflow is a symbolic math library based on dataflow and differentiable programming.

4. Keras :- Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3 Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, R, Theano, and PlaidML.

5.Theano :-Theano is a Python library and optimizing compiler for manipulating and evaluating mathematical expressions, especially matrix-valued ones. In Theano, computations are expressed using a NumPy-esque syntax and compiled to run efficiently on either CPU or GPU architectures.

6. OpenCV :- OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source Apache 2 License.

Discuss python with Google Colab.

Introduction :

Google is quite aggressive in AI research. Over many years, Google developed AI framework called TensorFlow and a development tool called colaboratory. Today TensorFlow is open-sourced and since 2017, Google made colaboratory free for public use. Colaboratory is now known as Google colab or simply colab.

Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold per-use basis.

Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

What Colab Offers You?

- Write and execute code in Python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets e.g. from Kaggle
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

Why should I use Google Colab?

- Google Colab is Free
- Easy to get started
- Allows access to GPUs/TPUs
- Easy to share code with others
- Easy graphical visualizations in Notebooks

PRACTICAL-2

AIM: Important packages for Machine learning: Numpy, Pandas and Matplotlib

Numpy

Import Numpy

In [1]:
```
import numpy as np
```

Blank array

In [2]:
```
array=np.empty(0)
print(array)
```

```
[]
```

with predefined data

In [3]:
```
array_pre=np.array([2,3,4])
print(array_pre)
```

```
[2 3 4]
```

with pattern specific data

In [4]:
```
array_one=np.ones( (3,4), dtype=np.int16 )
print(array_one)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

In [5]:
```
array_arange=np.arange( 0, 2, 0.3 )
print(array_arange)
```

```
[0.  0.3 0.6 0.9 1.2 1.5 1.8]
```

In [6]:
```
array_random=np.random.rand(2,3)
print(array_random)
```

```
[[0.30954465 0.76080685 0.08037883]
 [0.3870983  0.66667798 0.38333453]]
```

Slicing

In [7]:
```
slice_array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
slice_array_1 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
slice_array[7]=62
slice_array_1[0,3]=16
print(slice_array)
print(slice_array_1)
print(slice_array[1:7])
print(slice_array[2:])
print(slice_array[:12])
print(slice_array[-3:-1])
print(slice_array[1:8:3])
print(slice_array_1[1, 1:4])
```

```
[ 1  2  3  4  5  6  7 62  9]
[[ 1  2  3 16  5]
 [ 6  7  8  9 10]]
[2 3 4 5 6 7]
[ 3  4  5  6  7 62  9]
[ 1  2  3  4  5  6  7 62  9]
[ 7 62]
[ 2  5 62] [7 8 9]
```

Shape manipulation

In [8]:
```
a=np.random.random((15))
print(a)
A=a.reshape(3,5)
print(A)
a.shape = (3, 5)
print(a)
```

```
   [0.84336224 0.64222155 0.87445282 0.79656005 0.45053672 0.55758312 0.5711644
              0.26345514 0.45443452 0.13087534 0.74173258 0.06919156
 0.81837963 0.27055124 0.42718973]
[[0.84336224 0.64222155 0.87445282 0.79656005 0.45053672]
 [0.55758312 0.5711644  0.26345514 0.45443452 0.13087534]
 [0.74173258 0.06919156 0.81837963 0.27055124 0.42718973]]
[[0.84336224 0.64222155 0.87445282 0.79656005 0.45053672]
 [0.55758312 0.5711644  0.26345514 0.45443452 0.13087534]
 [0.74173258 0.06919156 0.81837963 0.27055124 0.42718973]]
```

Looping over arrays

In [9]:
```
print("1-dimension Array")
for x in slice_array:
     print(x)
print("2-dimension Array")
for x in slice_array_1:
    for y in x:
        print(y)
```

```
1-dimension Array
1
2
3
4
5
6
7
62
9
2-dimension Array
1
2
3
16
5
6
7
8
9
10
```

[10]:
```
from numpy import genfromtxt
my_data = genfromtxt('../input/patient-temperature-and-pulse-rate/ffffinal.csv',
print(my_data)
```

```
[[    nan     nan     nan]
 [  35.5  2767.       nan]
 [  35.5  1018.       nan]
 [  32.25 1475.       nan]
 [  30.44 1769.       nan]
 [  31.31 1691.       nan]
 [  37.88 1713.       nan]
 [  35.31 1381.       nan]
 [  34.69 2912.       nan]
 [  34.56 2367.       nan]
 [  34.75 1686.       nan]
 [  34.94 1392.       nan]
 [  35.   2029.       nan]
 [  35.06 2151.       nan]
 [  35.13 2070.       nan]
 [  35.13    0.       nan]
 [  35.19 1296.       nan]
 [  35.19 2225.       nan]
 [  35.19 2581.       nan]
 [  35 19 2780           ]
```

Use numpy vs list for matrix multiplication of 1000 X 1000 array

In [11]:
```
import time

start_time = time.time()

num_multiplies = 5000000
data = range(num_multiplies)
number = 1

for i in data:
    number *= 1.0000001

end_time = time.time()

print(number)
print("Run time = {}".format(end_time - start_time))
```

```
1.648721229963447
Run time = 0.7025384902954102
```

[12]:
```
import time
import numpy as np

start_time = time.time()

data = np.ones(shape=(1000, 1000), dtype="float")

for i in range(5):
    data *= 1.0000001

end_time = time.time()

print("Run time = {}".format(end_time - start_time))
```

```
Run time = 0.006282329559326172
```

Pandas

Importing Pandas

In [13]:
```
import pandas as pd
```

Create Dataframe

In [14]:
```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Read csv

[15]:
```
df = pd.read_csv('../input/patient-temperature-and-pulse-rate/ffffinal.csv')
print(df)
```

```
      Temp  Pulse       Time
0    35.50   2767   17:43:24
1    35.50   1018   17:43:40
2    32.25   1475   17:43:55
3    30.44   1769   17:44:18
4    31.31   1691   17:44:34 ..     ...    ...        ...
```

```
151  30.75      0   18:27:13
152  32.69   2335   18:28:03
153  32.94   2368   18:28:18
154  33.13   2705   18:28:34
155  33.25   2879   18:28:50

156  rows x 3 columns]
```

Slicing

In [16]:
```
df1 = df.iloc[:,0:2]
print(df1)
```

```
      Temp  Pulse
0    35.50   2767
1    35.50   1018
2    32.25   1475
3    30.44   1769
4    31.31   1691 ..     ...    ...
151  30.75      0
152  32.69   2335
153  32.94   2368
154  33.13   2705
155  33.25   2879

[156 rows x 2 columns]
```

Exporting data

[17]:
```
df1.to_csv('file.csv')
df2 = pd.read_csv('file.csv')
print(df2)
```

```
     Unnamed: 0   Temp  Pulse
0             0  35.50   2767
1             1  35.50   1018
2             2  32.25   1475
3             3  30.44   1769
4             4  31.31   1691 ..          ...    ...    ...
151         151  30.75      0
152         152  32.69   2335
153         153  32.94   2368
154         154  33.13   2705
155         155  33.25   2879

156         rows x 3 columns]
```

Use pandas for masking data and reading if in Boolean format

In [18]:
```
df3 = pd.DataFrame({"A":[12, 4, 5, 44, 1],
                    "B":[5, 2, 54, 3, 2],
                    "C":[20, 16, 7, 3, 8],
                    "D":[14, 3, 17, 2, 6]})
df3.mask(df3 > 15, -45)
```

Out[18]:

	A	B	C	D
0	12	5	-45	14
1	4	2	-45	3
2	5	-45	7	-45
3	-45	3	3	2
4	1	2	8	6

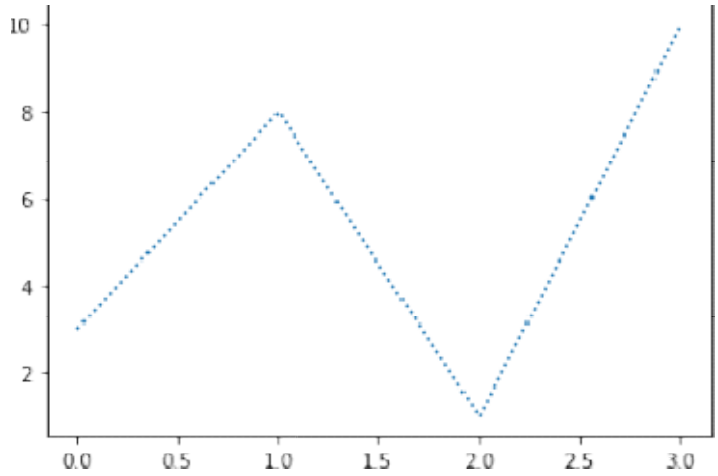
Matplotlib

Importing matplotlib

In [19]:
```
import matplotlib.pyplot as plt
```

Simple line chart

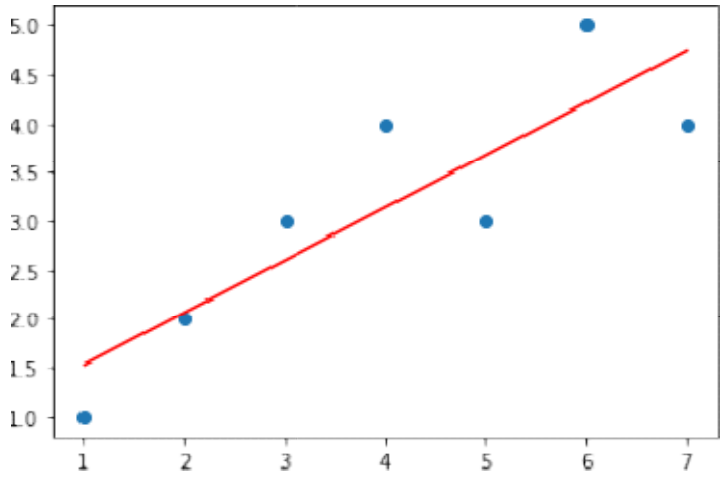
[20]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```



Correlation chart

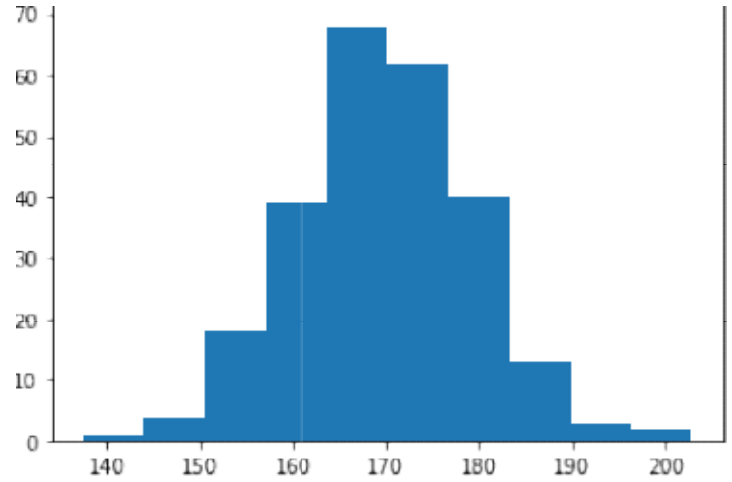
In [21]:
```
y = pd.Series([1, 2, 3, 4, 3, 5, 4])
x = pd.Series([1, 2, 3, 4, 5, 6, 7])
correlation = y.corr(x)
plt.scatter(x, y)

# This will fit the best line into the graph
plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))
         (np.unique(x)), color='red')
```

Out[21]: [<matplotlib.lines.Line2D at 0x7f0175d26ad0>]



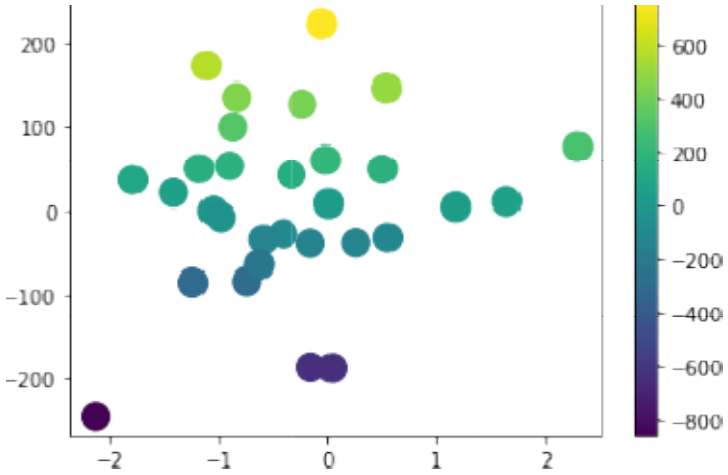
Histogram

In [22]:
```
x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```



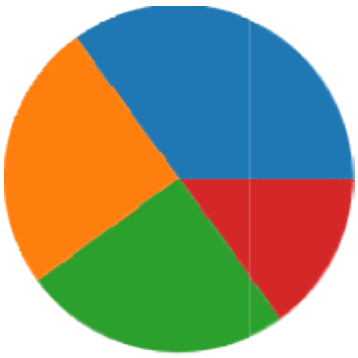
Plotting of Multivariate data

In [23]:
```
np.random.seed(2)

## generate a random data set
x, y = np.random.randn(2, 30)
y *= 100
z = 11*x + 3.4*y - 4 + np.random.randn(30) ##the model

fig, ax = plt.subplots()
scat = ax.scatter(x, y, c=z, s=200, marker='o')
fig.colorbar(scat)

plt.show()
```



Plot Pi Chart

In [24]:
```
y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```



Conclusion:

Here we have learned the basics of Python library such as pandas, numpy and matplotlib.

PRACTICAL-3

AIM:

Perform Linear Regression

Importing Libraries

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

Why you want to apply regression on selected dataset?

Because our Y which is target variable is of continuous type so we can apply regression

In [2]:
```
df = pd.read_csv('../input/student/student.csv')
df.head()
```

Out[2]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

How many total observations in data?

there are 25 observations.

How many independent variables?

There is only one independent variable which is Hours

Which is dependent variable?

Dependent variable is Scores

Which are most useful variable in estimation? Prove using correlation.

In this dataset there is only one independent variable. So there is no selection of most useful variable. But following is correlation between independent and dependent variable.

Out[3]:

[3]:
```
df.corr()
```

	Hours	Scores
Hours	1.000000	0.976191
Scores	0.976191	1.000000

Data Preparation

Finding Missing Values

Out[4]:

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

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

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

```
<class 'pandas.core.frame.DataFrame'>
 RangeIndex: 25 entries, 0 to 24
 Data columns (total 2 columns):
  #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
0   Hours   25 non-null     float64
  1   Scores   25   non-null    int64
dtypes: float64(1), int64(1) memory
usage: 528.0 bytes
```

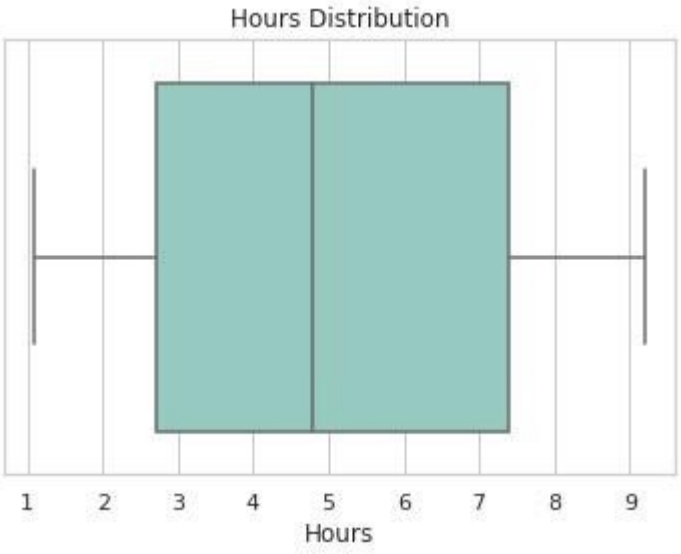
In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Hours     0 Scores
0 dtype: int64
```

Hours column

[7]:
```
import seaborn as sns

sns.set_theme(style="whitegrid")
sns.boxplot(x="Hours", data=df, palette="Set3")
plt.title("Hours Distribution")
```

Out[7]: Text(0.5, 1.0, 'Hours Distribution')



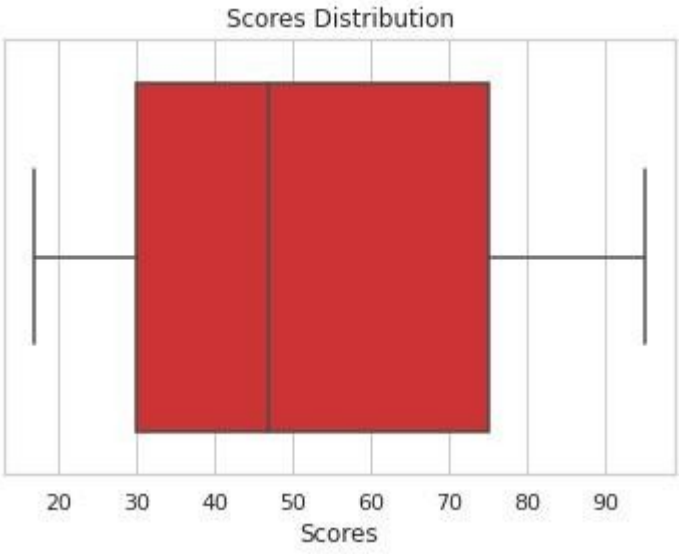
In [8]:
```
df.Hours.min(),df.Hours.max()
```

Out[8]: (1.1, 9.2)

Scores Column

[9]:
```
sns.set_theme(style="whitegrid")
sns.boxplot(x="Scores", data=df, palette="Set1")
plt.title("Scores Distribution")
```

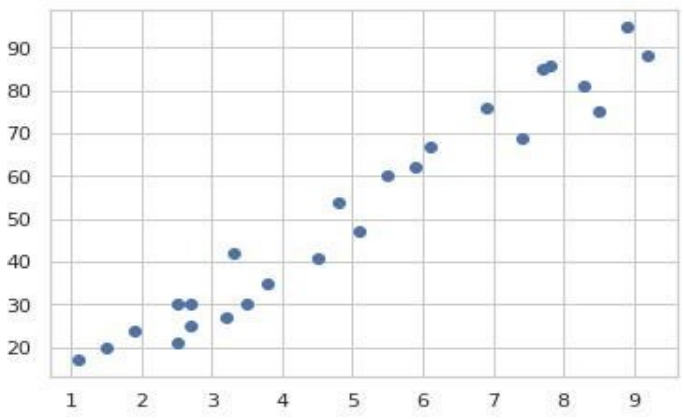
Out[9]: Text(0.5, 1.0, 'Scores Distribution')



In [10]:
```
df.Scores.min(),df.Scores.max()
```

Out[10]: (17, 95)

[11]:
```
plt.scatter(df.Hours, df.Scores)
plt.show()
```



In [12]:
```
X = df['Hours']
Y = df['Scores']
```

Implement linear regression using OLS method

In [13]:
```
x_bar = X.sum()/X.count()
y_bar = Y.sum()/Y.count()
print(x_bar,y_bar)
```

```
5.012 51.48
```

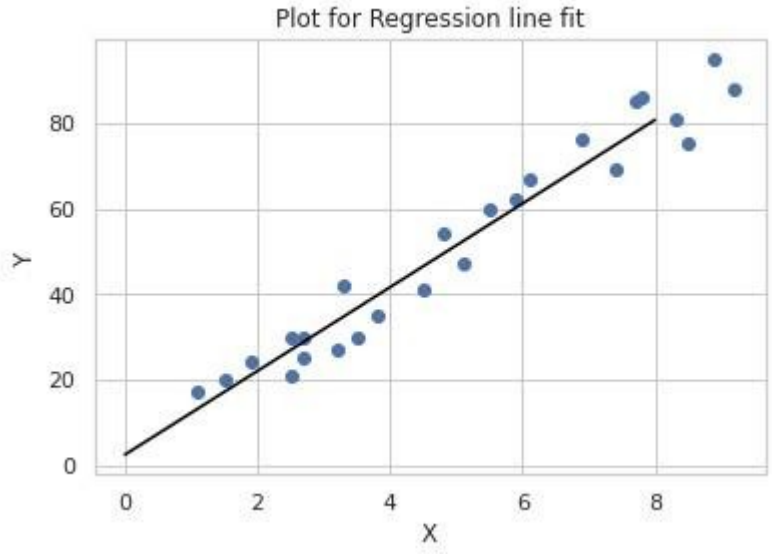
In [14]:
```
n = ((X-x_bar) * (Y-y_bar)).sum()
d = ((X-x_bar)**2).sum()
m = n/d
b = y_bar - m* x_bar
print(m, b)
```

```
9.775803390787473 2.4836734053731817
```

[15]:
```
predicted_df = pd.DataFrame(data = range(0,int(X.max())), columns={'X'})
predicted_df['Y'] = predicted_df.X*m + (b)
```

In [16]:
```
x = predicted_df['X']
y = predicted_df['Y']
```

In [17]:
```
plt.plot(x,y,c='black')
plt.scatter(X, Y)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Plot for Regression line fit')
#plt.legend()
plt.show()
```



In [18]:
```
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
Y_pred = m*X + b
print("RMSE: ", sqrt(mean_squared_error(Y,Y_pred)),"R2 score", r2_score(Y,Y_pred))
```

```
RMSE:  5.374265578592619 R2 score 0.9529481969048356
```

Implement linear regression using Gradient Descent from scratch

In [20]:
```
Y_pred = m*X + c
print("RMSE: ", sqrt(mean_squared_error(Y,Y_pred)),"R2 score", r2_score(Y,Y_pred))
```

[19]:
```
m = 0
c = 0

L      = 0.00001  # The learning Rate
epochs = 2500  # The number of iterations to perform gradient descent

n = float(len(X)) # Number of elements in X

# Performing Gradient Descent
for i in range(epochs):
    Y_pred = m*X + c  # The current predicted value of Y
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c
print (m, c)
```

```
7.920512018577363 1.287104473643262
RMSE:  12.65318741748985 R2 score 0.7391817933543926
```

Implement linear regression using sklearn API

In [21]:
```
from sklearn import linear_model
# This is using SKlearn API
X = pd.DataFrame(df.Hours)
Y = df.Scores

# Create object of algorithm
rg = linear_model.LinearRegression()
# Create model by fitting data
rg.fit(X, Y)

# RMSE and R2 Score
print("RMSE: ", sqrt(mean_squared_error(Y,rg.predict(X))), "R2 Score:", r2_score(
```

```
RMSE:  5.374265578592619 R2 Score: 0.9529481969048356
```

Conclusion:

Regression using SKlearn Api and OLS gives same RMSE and R2 Score which is good as it gives highest R2 score and low RMSE.

PRACTICAL-4

AIM:
Multiple Linear Regression

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

Load dataser

In [ ]:
```
#fetchin data
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
print(housing.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESC
R'])
```

In [ ]:
```
data = pd.DataFrame(data = housing.data, columns = housing.feature_names)
data.head()
```

Out[13]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

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

Out[14]: (20640, 8)

Out[15]:

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

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	L
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34

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

75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
  0MedInc       20640 non-null  float64
  1HouseAge     20640 non-null  float64
  2AveRooms     20640 non-null  float64
  3AveBedrms    20640 non-null  float64
  4Population   20640 non-null  float64
  5AveOccup     20640 non-null  float64
  6Latitude     20640 non-null  float64
  7Longitude    20640 non-null  float64 dtypes: float64(8) memory usage: 1.3
 MB
```

In [ ]:
```
data.isnull().sum()
```

Out[17]:
```
MedInc        0 HouseAge
0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0 dtype:
int64
```

In [ ]:
```
x = data.iloc[:,0:-1]
y = data.iloc[:,-1]
```

spiting data into training and testing

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

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.20, random_st
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score
lin_model = LinearRegression()
lin_model.fit(x_train, y_train)
```

Out[20]:
```
LinearRegression()
```

- model eveluation for training set

In [ ]:
```
y_train_predict = lin_model.predict(x_train)

RMSE = sqrt(mean_squared_error(y_train, y_train_predict))
r2 = r2_score(y_train, y_train_predict)

print("RMSE: ", RMSE)
print("r2 Score: ", r2)
```

```
RMSE:  0.6680185285231893
r2 Score:  0.8890146376553385
```

- model eveluation for testing set

In [ ]:
```
y_test_predict = lin_model.predict(x_test)

RMSE = sqrt(mean_squared_error(y_test, y_test_predict))
r2 = r2_score(y_test, y_test_predict)

print("RMSE: ", RMSE)
print("r2 Score: ", r2)
```

```
RMSE:  0.6746544360594764 r2
Score:  0.8858012453063419
```

In [ ]:
```
print("Total co-eficients of model: ", len(lin_model.coef_))
print(lin_model.intercept_)
```

```
Total co-eficients of model:  7
-86.58432748372337
```

In [ ]:
```
print('Train Score: ', lin_model.score(x_train, y_train)) print('Test
Score: ', lin_model.score(x_test, y_test))
```

```
Train Score:  0.8890146376553385
Test Score:  0.8858012453063419
```

Conclusion: Here we performed Multiple Linear Regression .

PRACTICAL-5

AIM: Logistic Regression using sklearn

Importing Libraries

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

Reading Dataset

In [2]:
```
df = pd.read_csv("../input/wine-quality-binary-classification/wine.csv")
df.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	**3**
					0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	

[4]:
```
df.info()
```

Data Prepocessing

In [3]:
```
df.shape
```

Out[3]:
```
(1599, 12)
<class 'pandas.core.frame.DataFrame'> RangeIndex:
1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
  0 fixed acidity         1599 non-null   float64
  1 volatile acidity      1599 non-null   float64
  2 citric acid           1599 non-null   float64
  3 residual sugar        1599 non-null   float64
  4 chlorides             1599 non-null   float64
  5 free sulfur dioxide   1599 non-null   float64
  6 total sulfur dioxide  1599 non-null   float64
  7 density               1599 non-null   float64
  8 pH                    1599 non-null   float64
  9 sulphates             1599 non-null   float64
 10alcohol                1599 non-null   float64
 11quality                1599 non-null   object  dtypes: float64(11),
 object(1) memory usage: 150.0+ KB
```

Why you want to apply classification on selected dataset? Discuss full story behind dataset

Here, the dependent variable which is 'quality' having data as class value so we can apply classification to classify the test data belongs to which class.

How many total observations in data?

There are total 1599 observations are there

How many independent variables?

There are total 11 independent varibales are there

Which is dependent variable?

qualtity is the dependent variable

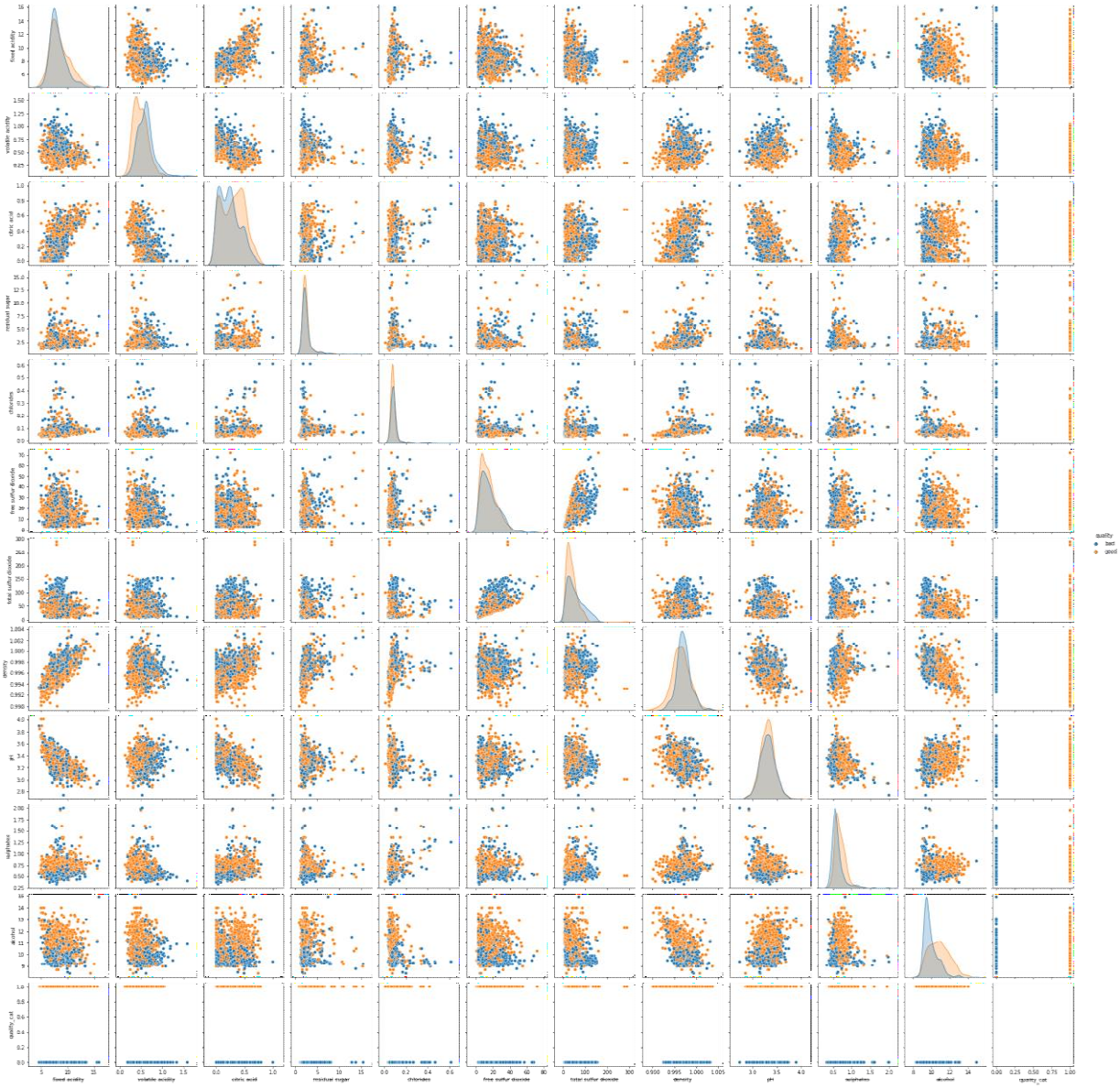
[5]:
```
# Encoding categorical variable
df['quality_cat'] = df['quality'].astype('category').cat.codes
df.head()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	**3**
	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	

```
[6]: sns.pairplot(df,hue='quality')
```

```
Out[6]: <seaborn.axisgrid.PairGrid at 0x7fcba897a0d0>
```

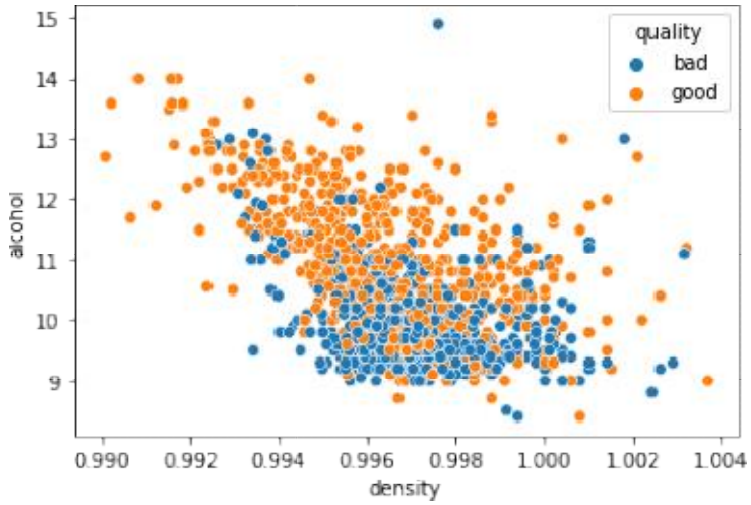


In [7]:
```
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047	-
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026	
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947	-
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283	-
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632	-
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.021946	
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.071269	-
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.000000	-
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.341699	
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.148506	-
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.496180	
quality_cat	0.095093	-0.321441	0.159129	-0.002160	-0.109494	-0.061757	-0.231963	-0.159110	-

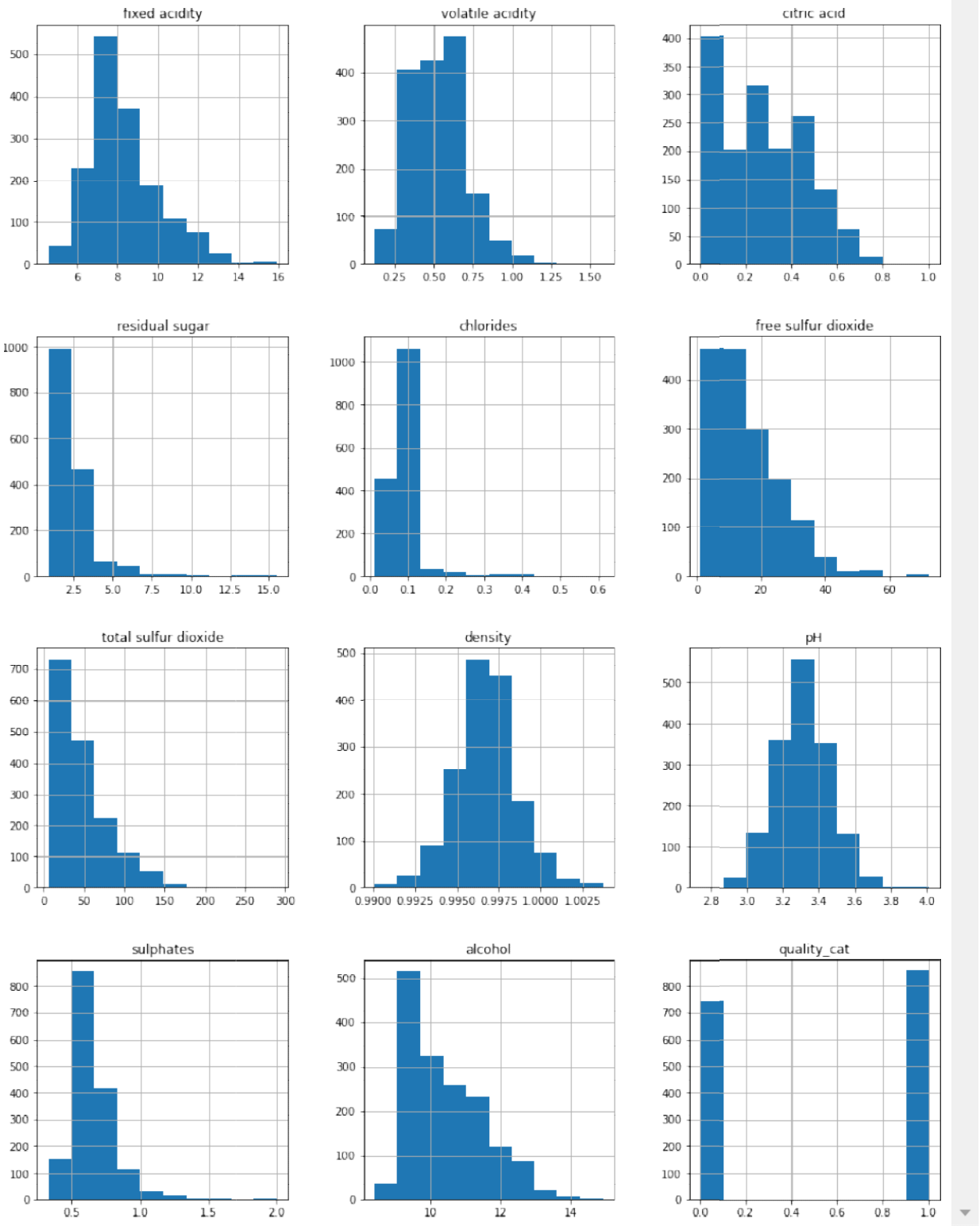
In [8]:
```
# Plotting the highest correlated pairs
sns.scatterplot(data=df, x='density', y='alcohol', hue='quality')
plt.show()
```



In [9]:
```
fig = plt.figure(figsize = (15,20))
ax = fig.gca()
df.hist(ax = ax)
```

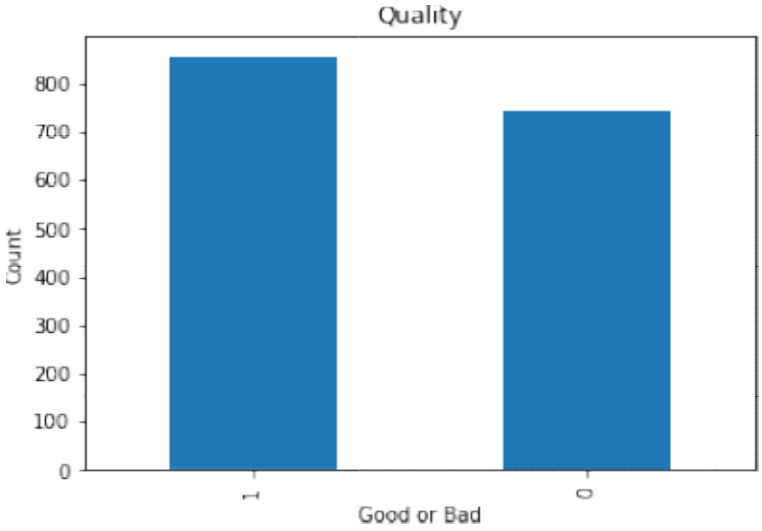
```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:3: UserWarning:
To output multiple subplots, the figure containing the passed axes is being
cleare d
  This is separate from the ipykernel package so we can avoid doing imports unt
il
```

Out[9]:
```
array([[<AxesSubplot:title={'center':'fixed acidity'}>,
        <AxesSubplot:title={'center':'volatile acidity'}>,
        <AxesSubplot:title={'center':'citric acid'}>],
       [<AxesSubplot:title={'center':'residual sugar'}>,
        <AxesSubplot:title={'center':'chlorides'}>,
        <AxesSubplot:title={'center':'free sulfur dioxide'}>],
       [<AxesSubplot:title={'center':'total sulfur dioxide'}>,
        <AxesSubplot:title={'center':'density'}>,
        <AxesSubplot:title={'center':'pH'}>],
       [<AxesSubplot:title={'center':'sulphates'}>,
        <AxesSubplot:title={'center':'alcohol'}>,
        <AxesSubpot:title={'center':'quality_cat'}>]], dtype=object)
```


fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality_cat

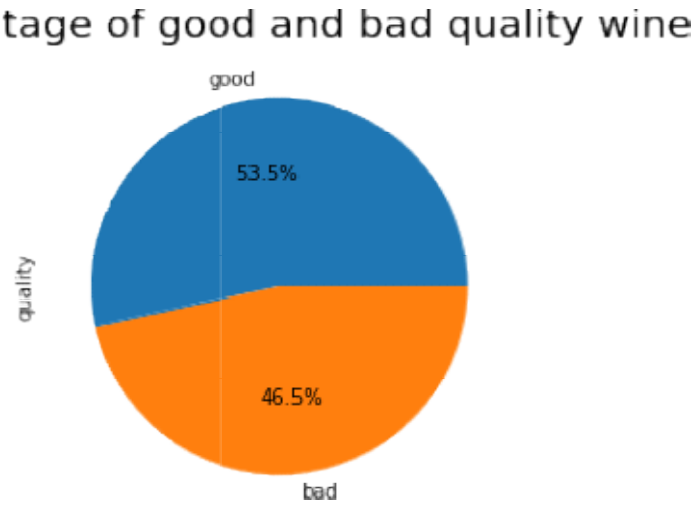
[10]:
```
    df.quality_cat.value_counts().plot(kind='bar')
plt.xlabel("Good  or  Bad")  plt.ylabel("Count")
plt.title("Quality")
    #Here we can see that dataset is not much imbalanced so there is no need to balan
```

Out[10]: Text(0.5, 1.0, 'Quality')



[11]:
```
 plt.figure(figsize=(20,10))   plt.subplots_adjust(left=0,   bottom=0.5,   right=0.9,
top=0.9, wspace=0.5, hspace=0. plt.subplot(141)
plt.title('Percentage of good and bad quality wine',fontsize = 20)
df['quality'].value_counts().plot.pie(autopct="%1.1f%%")
```

Out[11]:
```
<AxesSubplot:title={'center':'Percentage of good and bad quality wine'}, ylabel
='quality'>
```



In [12]:
```
df1 = df.drop('quality',axis=1)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
  0fixed   acidity    1599   non-null    float64
  1volatile   acidity   1599   non-null    float64
  2citric acid           1599  non-null  float64
  3residual   sugar    1599   non-null    float64
  4chlorides             1599  non-null  float64
  5free  sulfur  dioxide  1599  non-null  float64
  6total  sulfur  dioxide  1599  non-null  float64
  7density               1599 non-null   float64
  8pH                    1599 non-null   float64
  9sulphates             1599 non-null   float64
  10     alcohol               1599 non-null   float64
  11     quality_cat           1599  non-null  int8  dtypes:  float64(11),  int8(1)
 memory usage: 139.1 KB
```

In [13]:
```
X = df1.drop('quality_cat',axis=1)
Y = df1['quality_cat']
```

[14]:
```
Y.head()
```

Out[14]:
```
0    0 1
 0
 2    0
```

```
3    1
4    0
Name: quality_cat, dtype: int8
```

In [15]:
```
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_sta
```

In [16]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1279, 11)
(320, 11)
(1279,) (320,)
```

Implementing logistic Regression Using Sklearn

In [17]:
```
from sklearn.linear_model import LogisticRegression # for Logistic Regression Alg
from sklearn.preprocessing import StandardScaler
```

In [18]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [19]:
```
lr = LogisticRegression()
lr.fit(X_train ,y_train)
y_pred = lr.predict(X_test)
```

In [20]:
```
lr.score(X_test,y_test)
```

Out[20]: 0.75

In [21]:
```
from sklearn.metrics import accuracy_score
from sklearn import metrics
metrics.accuracy_score(y_test,y_pred)
```

Out[21]: 0.75

In [22]:
```
THRESHOLD = 0.5
y_pred = np.where(y_pred>0.5,1,0)
```

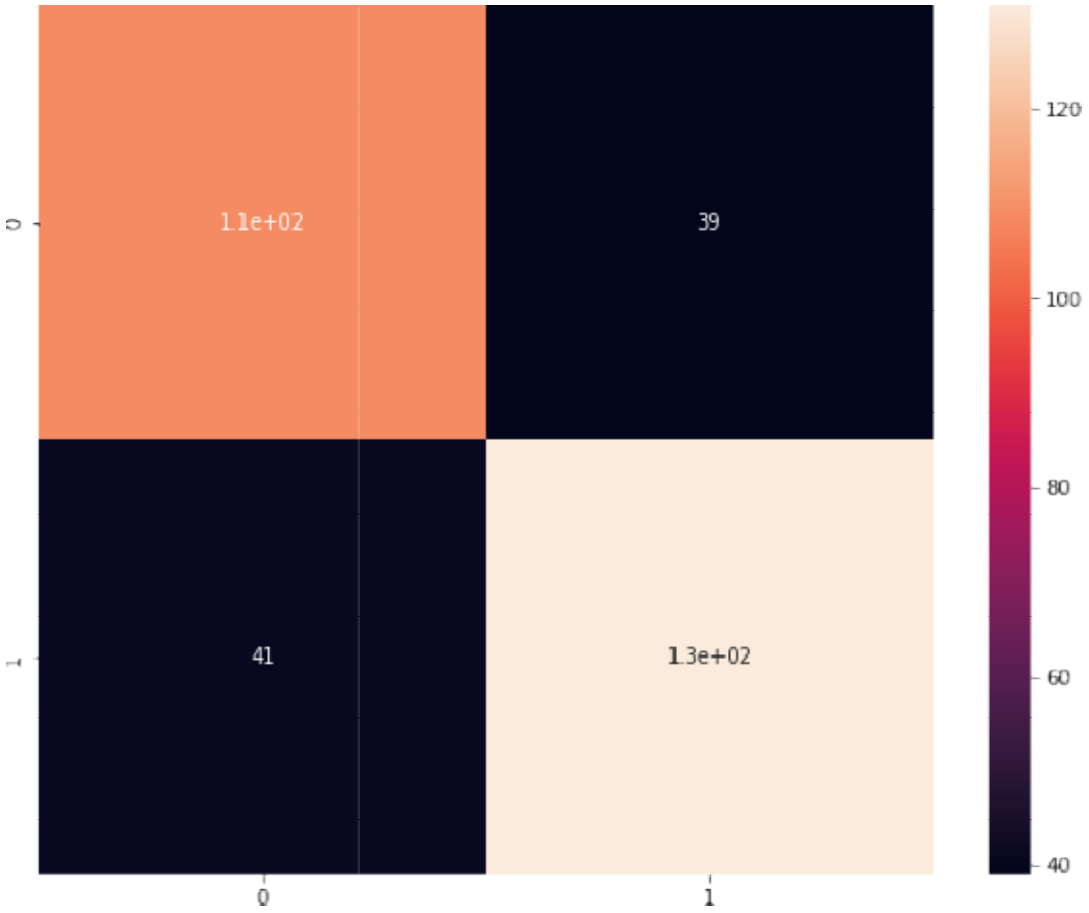
In [23]:
```
y_pred
```

Out[23]:
```
array([1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
```

Confusion Matrix

[24]:
```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,  8))
sns.heatmap(mat, annot=True)
```

Out[24]: <AxesSubplot:>



In [25]:
```
from sklearn.metrics import classification_report
target_names = ['Bad', 'Good']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

         Bad       0.73      0.74      0.73       148
        Good       0.77      0.76      0.77       172

    accuracy                           0.75       320   macro
avg       0.75      0.75      0.75       320 weighted avg
0.75      0.75      0.75       320
```

Conclusion: Here we have performed Logistic regression using Sklearn

PRACTRICAL-6

AIM:

Implementing Logistic Regression from Scratch

Importing Libraries

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

Reading Dataset

In [2]:
```
df = pd.read_csv("../input/wine-quality-binary-classification/wine.csv") df.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
			3	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
			4	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
5	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	3	11.2	0.28	0.56
							1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
	4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4		

[26]:
```
lr = 0.06

# Parameters initialization
weights = np.random.normal(0, 0.1, 11)
biais = random.normalvariate(0, 0.1)

m = X_train.shape[0]
for epoch in range(1000):

    # Forward pass
    Z = np.dot(X_train, weights) + biais
    A = 1 / (1 + np.exp(-Z))

    #Loss Computation
    J = np.sum(-(y_train * np.log(A) + (1 - y_train) * np.log(1 - A))) / m

    # Gradient computation
    dZ = A - y_train
    dw = np.dot(dZ, X_train) / m
    db = np.sum(dZ) / m

    # Update weights
    weights = weights - lr * dw
    biais = biais - lr * db

    if epoch % 10 == 0:
        print("epoch %s - loss %s" % (epoch, J))
```

```
epoch 0 - loss 0.7138506452747417
epoch 10 - loss 0.6468375918144419
epoch 20 - loss 0.6069835438228449
epoch 30 - loss 0.5825963278921259
epoch 40 - loss 0.5669954922000173
epoch 50 - loss 0.5565498218214684
epoch 60 - loss 0.549260930018061
epoch 70 - loss 0.5439881767825916
epoch 80 - loss 0.5400530962300578
epoch 90 - loss 0.5370360709711204
epoch 100 - loss 0.5346682952962636
epoch 110 - loss 0.5327720904206708
epoch 120 - loss 0.531226670066485
epoch 130 - loss 0.5299478210551694
epoch 140 - loss 0.528875467414943
epoch 150 - loss 0.527965847092534
epoch 160 - loss 0.5271864667571451
epoch 170 - loss 0.5265127739857121
epoch 180 - loss 0.5259259164464701
epoch 190 - loss 0.525411204021404
epoch 200 - loss 0.524957034543969
epoch 210 - loss 0.5245541309186348
epoch 220 - loss 0.524194990940252
epoch 230 - loss 0.5238734847192048
epoch 240 - loss 0.5235845560723806
epoch 250 - loss 0.5233239981762396
epoch 260 - loss 0.5230882829736426
epoch 270 - loss 0.5228744299826167
```

```
epoch 850 - loss 0.5203144012872417
epoch 860 - loss 0.5203103130204699
epoch 870 - loss 0.5203064350545457
epoch 880 - loss 0.5203027552848943
epoch 890 - loss 0.5202992623677728
epoch 900 - loss 0.5202959456690499
epoch 910 - loss 0.5202927952166487
epoch 920 - loss 0.5202898016563734
epoch 930 - loss 0.5202869562108615
epoch 940 - loss 0.5202842506414297
epoch 950 - loss 0.520281677212593
epoch 960 - loss 0.5202792286590618
epoch 970 - loss 0.5202768981550322
epoch 980 - loss 0.5202746792856014
epoch 990 - loss 0.5202725660201521
```

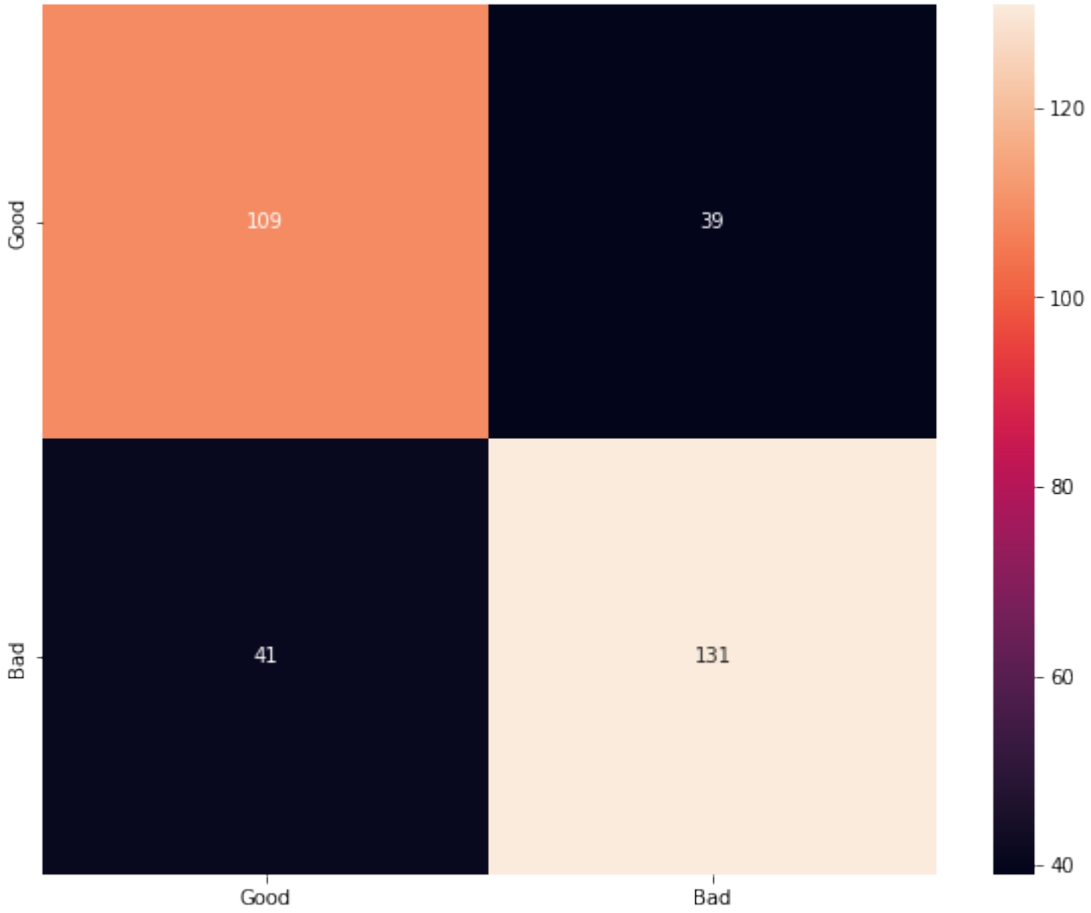
In [27]:
```
preds = []
for feats in X_test:

    z = np.dot(feats, weights) + biais
    a = 1 / (1 + np.exp(-z))

    if a > 0.5:
        preds.append(1)
    elif a <= 0.5:
        preds.append(0)
```

In [28]:
```
from sklearn.metrics import classification_report
target_names = ['Bad', 'Good']
print(classification_report(y_test, preds, target_names=target_names))
```

```
              precision    recall  f1-score   support

         Bad       0.73      0.74      0.73       148
        Good       0.77      0.76      0.77       172

    accuracy                           0.75       320
   macro avg       0.75      0.75      0.75       320
weighted avg       0.75      0.75      0.75       320
```

In [29]:
```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(y_test, preds)
plt.figure(figsize=(10, 8))
sns.heatmap(mat,xticklabels=['Good', 'Bad'],
yticklabels=['Good', 'Bad'], fmt='.0
```

Out[29]: <AxesSubplot:>



In [30]:
```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, preds)
print('F1 score: %f' % f1)
```

```
F1 score: 0.766082
```

In [31]:
```
from sklearn.metrics import precision_score, recall_score
print('Precision is: ', precision_score(y_test, preds))
print('Recall is: ', recall_score(y_test, preds))
```

```
Precision is:  0.7705882352941177
Recall is:  0.7616279069767442
```

In [32]:
```
from sklearn.metrics import confusion_matrix tn, fp, fn, tp =
confusion_matrix(y_test, preds).ravel() print('True negatives: ', tn, '\nFalse
positives: ', fp, '\nFalse negatives: ', f
```

```
True negatives:  109
False positives:  39
False negatives:  41
True Positives:  131
```

Conclusion : Here we have performed Logistic Regression from Scratch.

PRACTICAL-7

AIM: KNN for Classification

Lets load data

In [1]:
```
import pandas as pd data = pd.read_csv("../input/beginners-classification-
dataset/classification.csv" print(data)
```

```
           age   interest  success
0     23.657801  18.859917      0.0
1     22.573729  17.969223      0.0
2     32.553424  29.463651      0.0
3     6.718035  25.704665      1.0
4     14.401919  16.770856      0.0 ..          ...         ...      ...
292  27.697220  18.799309      0.0
293  15.150959  72.000352      1.0
294  22.264378  68.453459      1.0
295  25.677420  90.118212      1.0
296  21.215594  48.265520      1.0

[297 rows x 3 columns]
```

**Why you want to apply classification on selected dataset? **

Dataset is having age and interest from which we can predict the success So in this case weather person will be successfull or not so i think KNN fits better.

How many total observations in data?

297 total observations

How many independent variables?

There are two independent variables Age and Interest

Which is dependent variable?

Success is dependent variable

Which are most useful variable in classification? Prove using correlation

Here there are two independent variable where intrest is most useful for classification of success which is shown below by correlation.

Out[2]:

[2]: data.corr()

	age	interest	success
age	1.000000	0.142876	0.173307
interest	0.142876	1.000000	0.760703
success	0.173307	0.760703	1.000000

Data Preparation

Finding Missing Values

Out[3]:

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

	age	interest	success
0	23.657801	18.859917	0.0
1	22.573729	17.969223	0.0
2	32.553424	29.463651	0.0
3	6.718035	25.704665	1.0
4	14.401919	16.770856	0.0

In [4]:
```
data.isnull().sum()
```

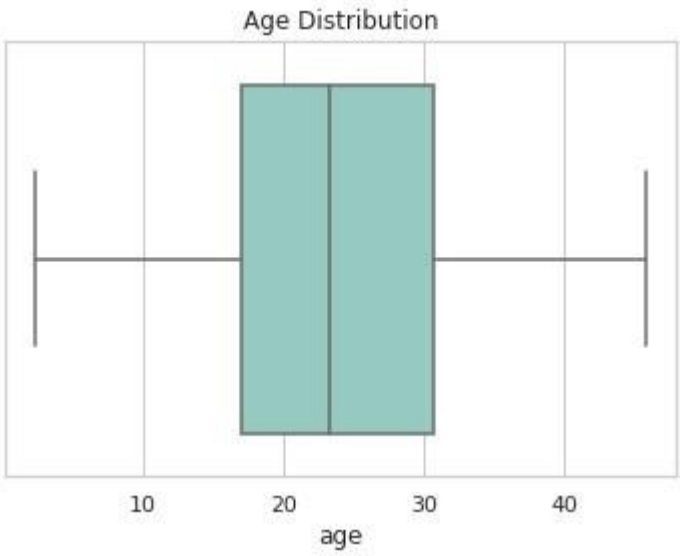
Out[4]:
```
age         0
interest    0
success     0
dtype: int64
```

Age Column

[5]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="whitegrid")
sns.boxplot(x="age", data=data, palette="Set3")
plt.title("Age Distribution")
```

```
[8]: data.interest.min(), data.interest.max()
```

Out[5]: Text(0.5, 1.0, 'Age Distribution')



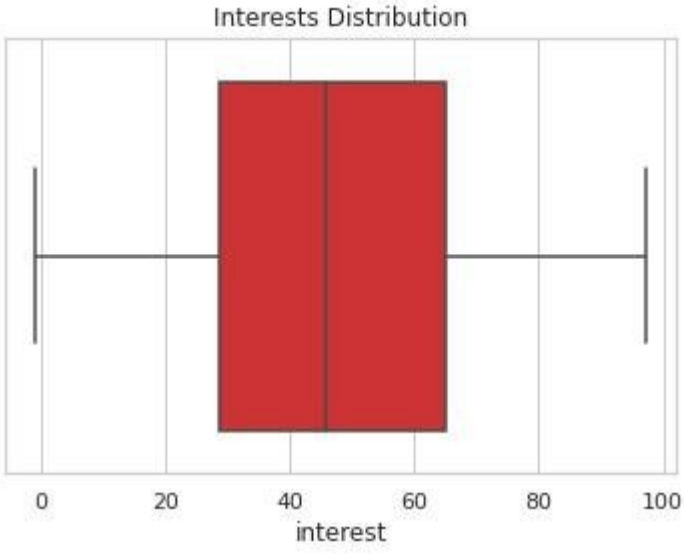

```
In [6]: data.age.min(), data.age.max()
```

Out[6]: (2.3475599373326848, 45.773728755936816)

Interest Column

```
In [7]: sns.set_theme(style="whitegrid")
sns.boxplot(x="interest", data=data, palette="Set1")
plt.title("Interests Distribution")
```

Out[7]: Text(0.5, 1.0, 'Interests Distribution')



Out[8]: (-0.8528003102534427, 97.1755075384403)

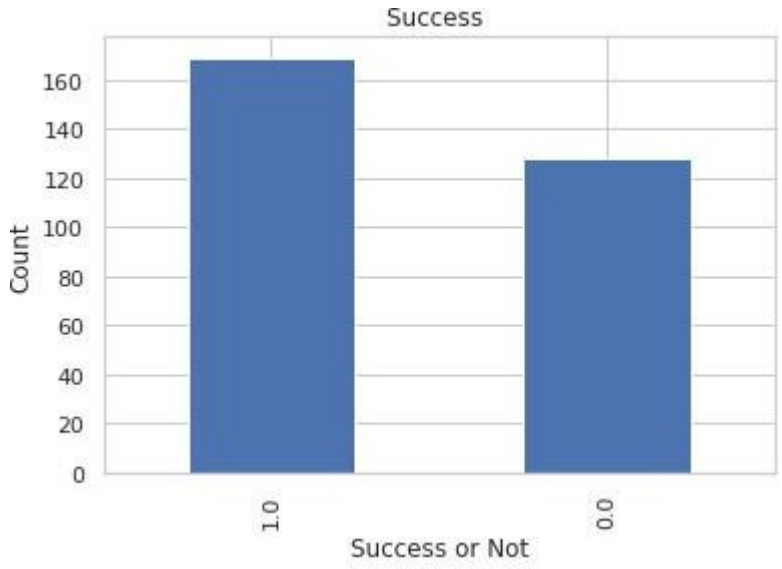
Success

```
In [9]: data.success.unique()
```

Out[9]: array([0., 1.])

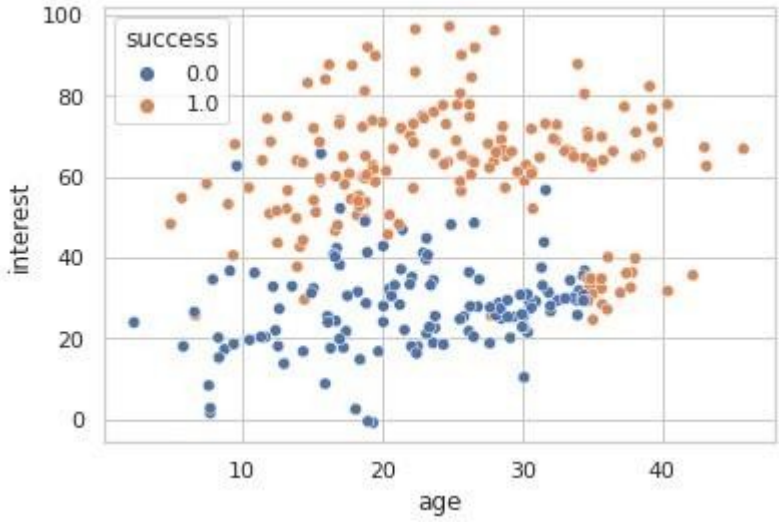
In [10]:
```
data.success.value_counts().plot(kind='bar')
plt.xlabel("Success or Not")
plt.ylabel("Count")
plt.title("Success ")
```

Out[10]: Text(0.5, 1.0, 'Success ')



In [11]:
```
sns.scatterplot(data=data, x="age", y="interest", hue="success")
```

Out[11]: <AxesSubplot:xlabel='age', ylabel='interest'>



Model Preparation and Prediction

Implement KNN using sklearn api

In [12]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

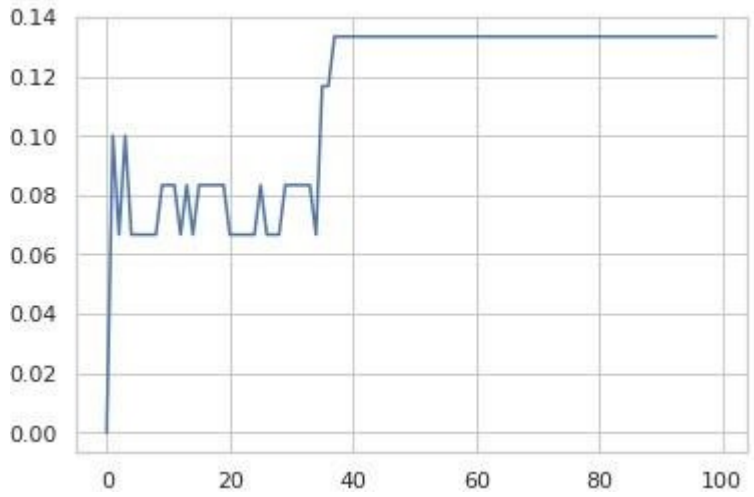
In [13]:
```
X = data[["age","interest"]]
Y = data["success"]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_
```

In [14]:
```
error_rates = []
import numpy as np
```

In [15]:
```
for i in np.arange(1, 101):

    new_model = KNeighborsClassifier(n_neighbors = i)

    new_model.fit(X, Y)

    new_predictions = new_model.predict(X_test)

    error_rates.append(np.mean(new_predictions != y_test))
```

In [16]:
```
plt.plot(error_rates)
```

Out[16]: [<matplotlib.lines.Line2D at 0x7f89b5cf3ed0>]



In [17]:
```
neigh = KNeighborsClassifier(n_neighbors=5)
neigh.fit(X_train, y_train)
```

Out[17]: KNeighborsClassifier()

In [18]:
```
y_pred = neigh.predict(X_test)
y_pred
```

Out[18]:
```
array([1., 0., 0., 0., 0., 1., 1., 1., 0., 1., 0., 0., 1., 1., 1., 1., 0.,
       0., 0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 1., 1., 0., 1., 1., 0.,
       1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 0., 1.,
       1., 0., 1., 1., 1., 1., 0., 1., 0.])
```

In [19]:
```
accuracy_score(y_test,y_pred)
```

Out[19]:
```
0.9333333333333333
```

Quantify goodness of your model and discuss steps taken for improvement

Here we have achieved accuracy of 93%. When value of K was 2 we were getting 85% and by increasing value of K to 5 we got 93%.

Can we use KNN for regression also? Why / Why not?

KNN algorithm can be used for both classification and regression problems. The KNN algorithm uses ‘feature similarity’ to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set.

Discuss drawbacks of algorithms such as KNN.

*Accuracy depends on the quality of the data

*With large data, the prediction stage might be slow

*Sensitive to the scale of the data and irrelevant features

*Require high memory – need to store all of the training data

*Given that it stores all of the training, it can be computationally expensive

Conclusion: here we have performed KNN for classification.

PRACTICAL 8

AIM: SVM for Classification

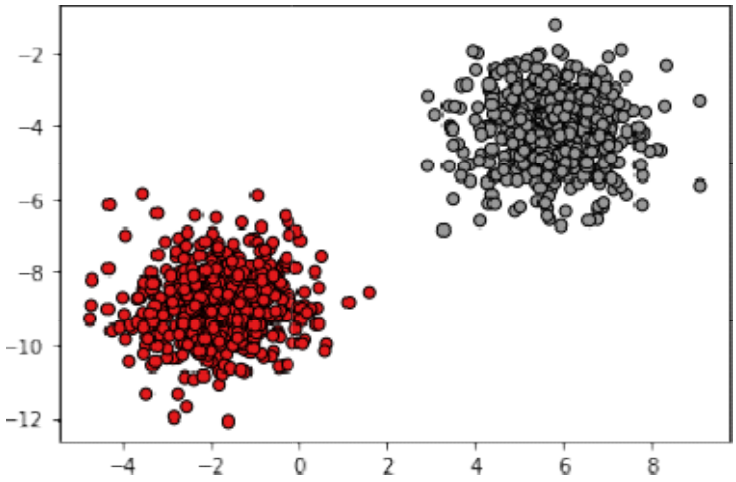
a. SVM Plot using Seaborn

1. Use make blob with different values to create synthetic data.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_blobs
import seaborn as sns
```

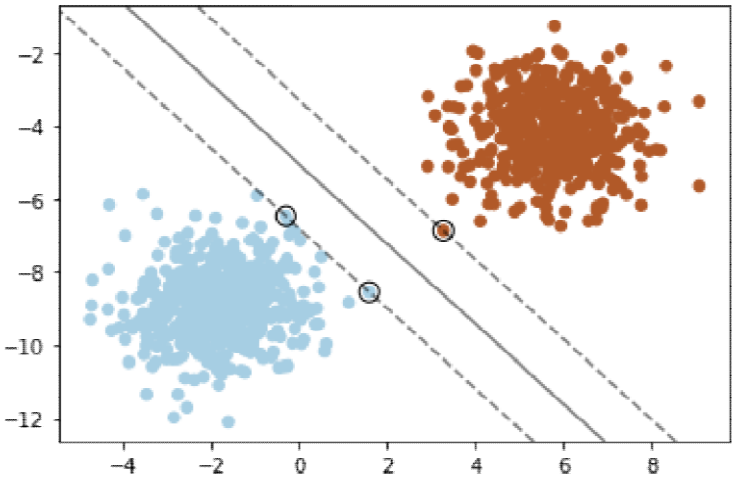
In [2]:
```
X, y = make_blobs(n_samples=1000,centers=2,n_features=2,random_state=40)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1, edgecolor="k")
print(X.shape,y.shape)
```

```
(1000, 2) (1000,)
```



2. Experiment with different types of synthetic data using SVM classification (C and Kernel).

In [3]:
```
# fit the model, don't regularize for illustration purposes
clf = SVC(kernel="linear", C=15)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(
    XX, YY, Z, colors="k", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "-", '
)
# plot support vectors
ax.scatter(
    clf.support_vectors_[:, 0],
    clf.support_vectors_[:, 1],
    s=100,
    linewidth=1,
    facecolors="none",
    edgecolors="k",
)
plt.show()
```



3.Find the dataset which is considered to be real observations.

4.Apply SVM to classify selected dataset.

In [4]:
```
df = pd.read_csv("/content/transfusion.csv")
df.shape
```

Out[4]: (748, 5)

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

Out[5]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007
0	2	50	12500	98	1
1	0	13	3250	28	1
2	1	16	4000	35	1
3	2	20	5000	45	1
4	1	24	6000	77	0

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

```
<class 'pandas.core.frame.DataFrame'> RangeIndex:
748 entries, 0 to 747
Data columns (total 5 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          ----------------------  -----
  0Recency (months)                                 748 non-null    int64
  1Frequency (times)                                748 non-null    int64
  2Monetary (c.c. blood)                            748 non-null    int64
  3Time (months)                                    748 non-null    int64
 4whether he/she donated blood in March 2007  748 non-null    int64 dtypes:
 int64(5) memory usage: 29.3 KB
```

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

Out[7]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007
count	748.000000	748.000000	748.000000	748.000000	748.000000
mean	9.506684	5.514706	1378.676471	34.282086	0.237968
std	8.095396	5.839307	1459.826781	24.376714	0.426124
min	0.000000	1.000000	250.000000	2.000000	0.000000
25%	2.750000	2.000000	500.000000	16.000000	0.000000
50%	7.000000	4.000000	1000.000000	28.000000	0.000000
75%	14.000000	7.000000	1750.000000	50.000000	0.000000

max	74.000000	50.000000	12500.000000	98.000000	1.000000

In [8]:
```
from sklearn.model_selection import train_test_split
X = df.iloc[:,0:4]
Y = df.iloc[:,-1]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random
```

In [9]:
```
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
X_train = Scaler.fit_transform(X_train)
X_test = Scaler.transform(X_test)
```

In [10]:
```
from sklearn.svm import SVC
classifier = SVC(kernel='rbf',random_state=0)
classifier.fit(X_train,Y_train)
```

Out[10]:
```
SVC(random_state=0)
```

In [11]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
y_pred = classifier.predict(X_test) print(confusion_matrix(Y_test,y_pred))
```

```
[[168   5]
 [ 44   8]]
```

In [12]:
```
accuracy = accuracy_score(Y_test,y_pred)
print(accuracy)
```

```
0.7822222222222223
```

5. Discuss analysis of synthetic dataset, iris dataset and one dataset of your choice.

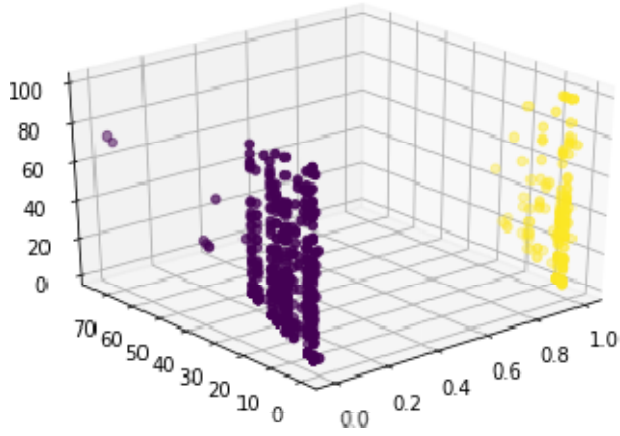
Above the discussion of synthetic dataset and one dataset that is related to your choice

i.e.Transfusion_blood has been discussed.

6. Use matplotlib for plotting data.

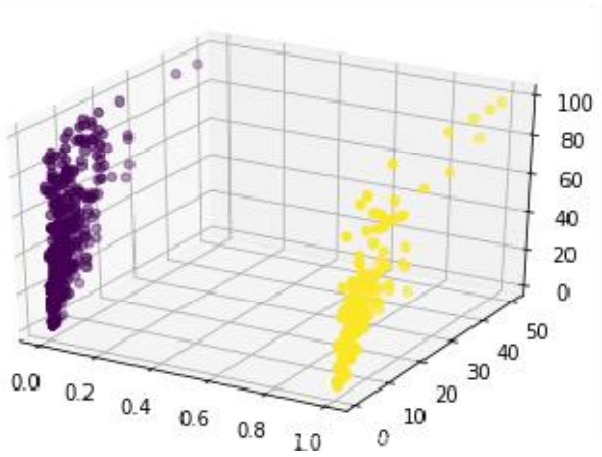
In [13]:

```
X.head()
ax = plt.axes(projection='3d')
ax.scatter3D(Y,X['Recency (months)'] ,X['Time (months)'] , c=Y, cmap='viridis')
ax.view_init(30, 230)
```



In [15]:

```
X.head()
ax = plt.axes(projection='3d')
ax.scatter3D(Y,X['Frequency (times)'] ,X['Time (months)'] , c=Y, cmap='viridis')
plt.show()
```



Conclusion: Here we have performed SVM using seaborn.

SVM

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables. SVM can be of two types:

Linear SVM:
Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
Non-linear SVM:
Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

SEABORN

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC
from sklearn.datasets import make_blobs
from sklearn import datasets
```

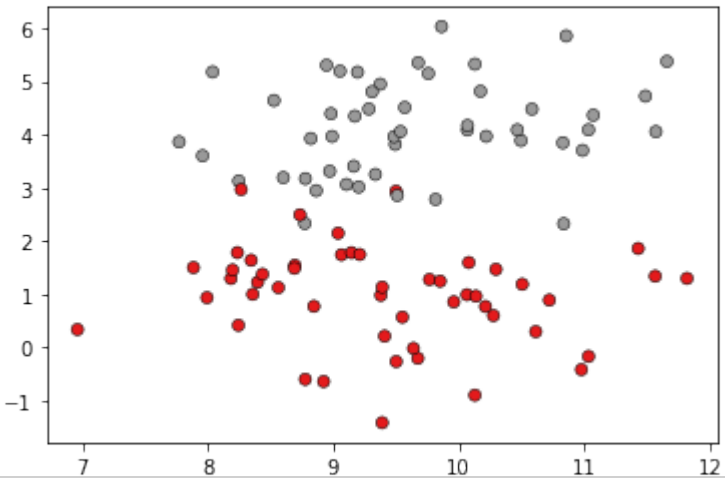
make_blobs:

The make_blobs() function can be used to generate blobs of points with a Gaussian distribution. You can control how many blobs to generate and the number of samples to generate, as well as a host of other properties.

```
# n_samples - it is the total number of points equally divided among clusters.
# centers - The number of centers to generate, or the fixed center locations. If
# 3 centers are generated.
 # make_blobs with random_state is 4 and n_samples is
100.
```

```
# Scatterplot - scatterplot is used to show the relationship between x and y can
# and style parameters.These parameters control what visual semantics are used to
# to three dimensions independently by using all three semantic types, but this s
# Using redundant semantics (i.e. both hue and style for the same variable) can b

X, y = make_blobs(n_samples=100, centers=2, random_state=4)
sns.scatterplot(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1, edgecolor="k")
print(X.shape,y.shape)
```

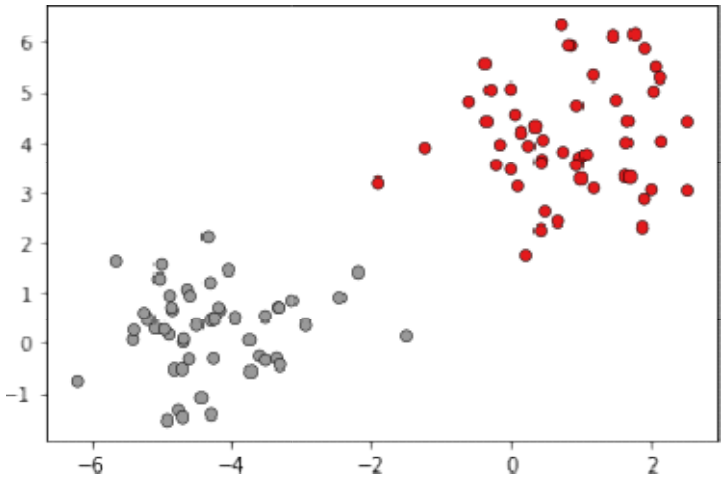
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
FutureWarning

(100, 2) (100,)
```



In [ ]:

```
# make_blobs with random_state is 3
X, y = make_blobs(n_samples=100, centers=2, random_state=3)
sns.scatterplot(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1, edgecolor="k")
print(X.shape,y.shape)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
FutureWarning (100, 2) (100,)
```

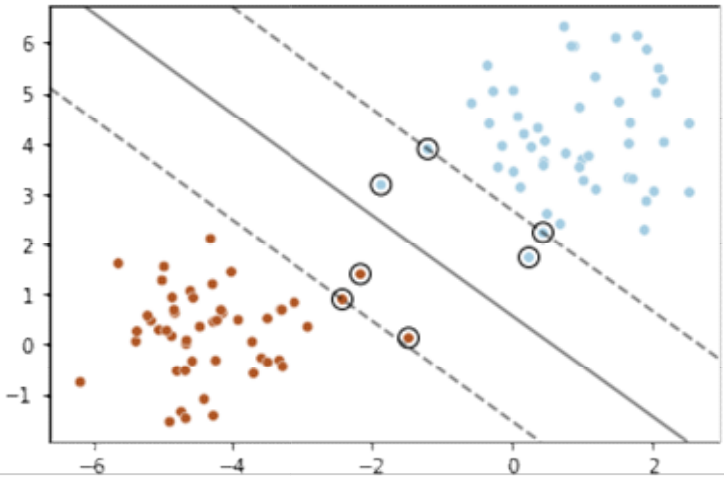


kernel

SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts nonseparable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate.

```
# linear kernel
# fit the model, don't regularize for illustration purposes
clf = SVC(kernel="linear", C=0.10)
clf.fit(X, y)

sns.scatterplot(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(
    XX, YY, Z, colors="k", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "-", '
)
# plot support vectors
ax.scatter(
    clf.support_vectors_[:, 0],
    clf.support_vectors_[:, 1],
    s=100,
    linewidth=1,
    facecolors="none",
    edgecolors="k",
)
plt.show()
```

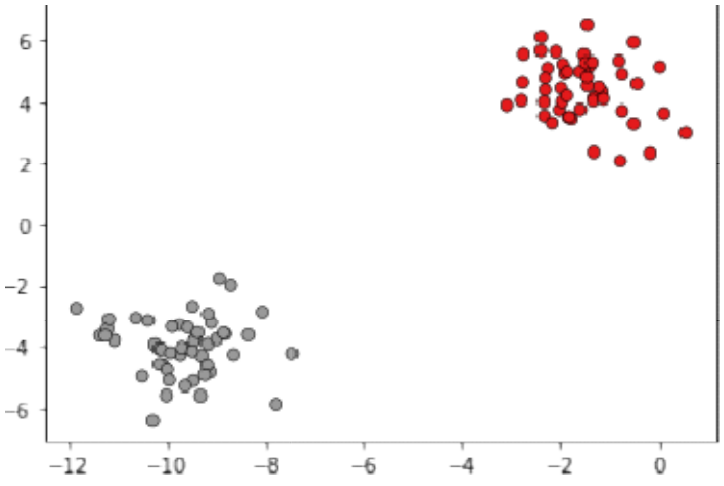
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



In [ ]:
```
# make_blobs with random_state is 1
X, y = make_blobs(n_samples=100, centers=2, random_state=1)
sns.scatterplot(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1, edgecolor="k")
print(X.shape,y.shape)
```

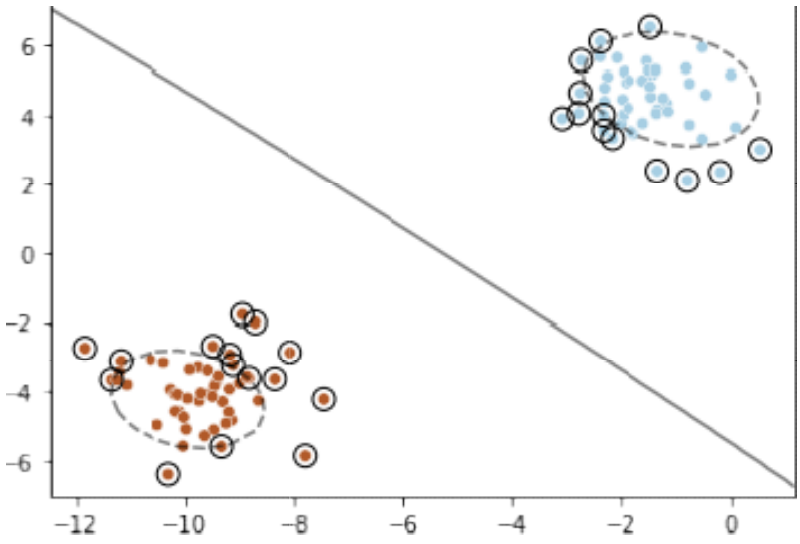
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
FutureWarning
```

```
(100, 2) (100,)
```



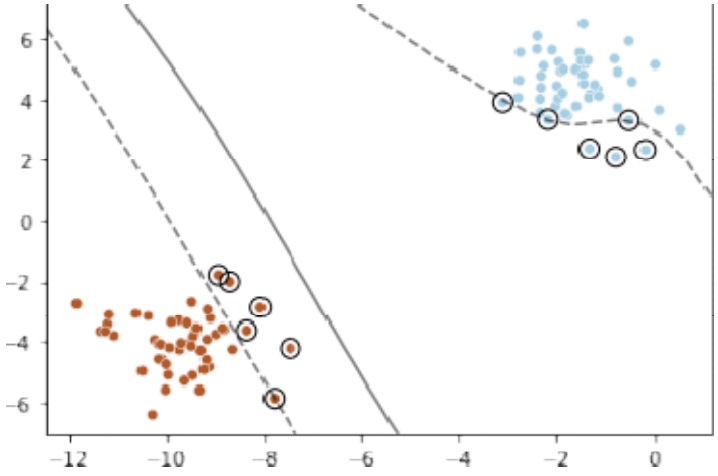
```
# rbf kernel
# fit the model, don't regularize for illustration purposes
clf = SVC(kernel="rbf", C=0.10)
clf.fit(X, y)

sns.scatterplot(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(
    XX, YY, Z, colors="k", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "-", '
)
# plot support vectors
ax.scatter(
    clf.support_vectors_[:, 0],
    clf.support_vectors_[:, 1],
    s=100,
    linewidth=1,
    facecolors="none",
    edgecolors="k",
)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```


6
4
2
0
−2
−4
−6
−12
−10
−8
−6
−4
−2
0

```
# Polynomial kernel
# fit the model, don't regularize for illustration purposes
clf = SVC(kernel="poly", C=0.10)
clf.fit(X, y)

sns.scatterplot(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(
    XX, YY, Z, colors="k", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "-", "
)
# plot support vectors
ax.scatter(
    clf.support_vectors_[:, 0],
    clf.support_vectors_[:, 1],
    s=100,
    linewidth=1,
    facecolors="none",
    edgecolors="k",
)
plt.show()
```

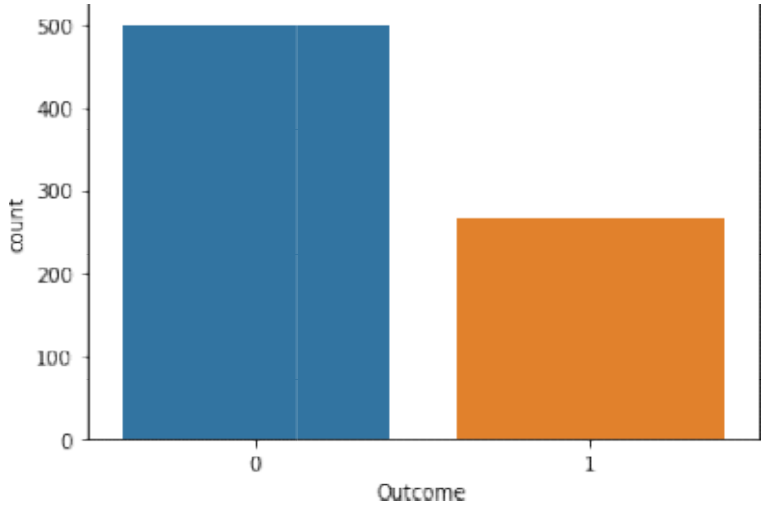
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

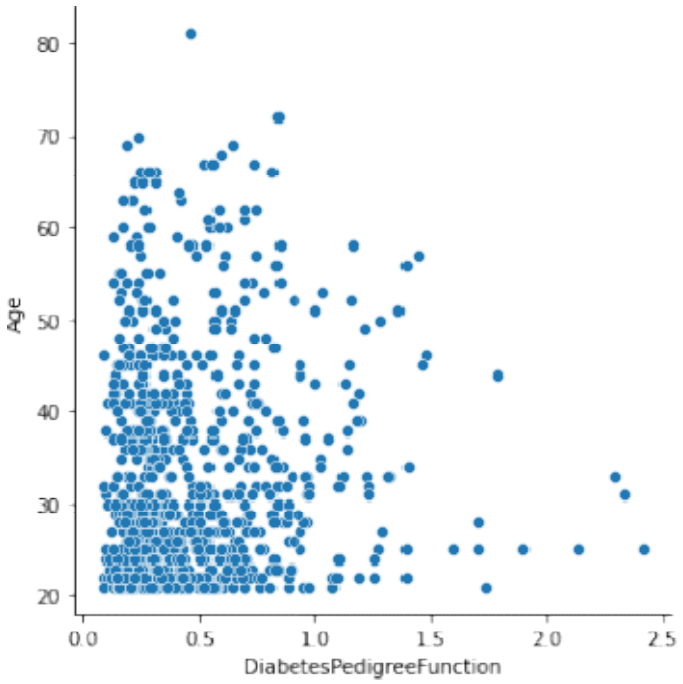


```
#  Apply SVM to classify selected dataset.
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
import seaborn as sns
```

In [ ]:
```
data = pd.read_csv("diabetes2.csv")
data.head(5)

# Countplot Shows the counts of observations in each categorical bin using bars.
sns.countplot(x="Outcome",data=data)

# This plot shows the relationship between two variables in the tips dataset usin
sns.relplot(data=data,x="DiabetesPedigreeFunction",y="Age")
```

Out[10]: <seaborn.axisgrid.FacetGrid at 0x7f56045803d0>





```
x= data.iloc[:, [0,1,2,3,4,5,6,7]].values
y= data.iloc[:, -1].values
```

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [ ]:
```
from sklearn.svm import SVC
# "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

Out[13]:
```
SVC(kernel='linear', random_state=0)
```

In [ ]:
```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

In [ ]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

Out[15]:
```
array([[117,  13],
       [ 25,  37]])
```

In [ ]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[16]:
```
0.8020833333333334
```

In [ ]:
```
from sklearn.metrics import classification_report
report = classification_report (y_test, y_pred)
print (report)
```

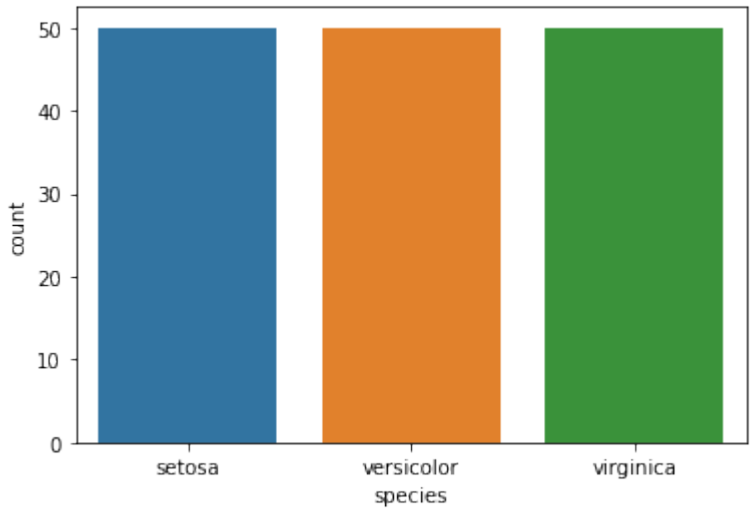
```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       130
           1       0.74      0.60      0.66
62

    accuracy                           0.80       192
macro avg       0.78      0.75      0.76       192
weighted avg       0.80      0.80      0.80       192
```

In [ ]:
```
# SVM for Iris Dataset
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#Define the col names
colnames=["sepal_length", "sepal_width","petal_length","petal_width","species"]
#Read the dataset
dataset = pd.read_csv("iris.csv", header = None, names= colnames )
dataset1= dataset.iloc[1: , : ]

#We can use countplot to check numbers of species in dataset.
sns.countplot(x="species",data=dataset1)

# sns.scatterplot(x="species",y="sepal_length",data=dataset)
#Data
dataset.head()
```

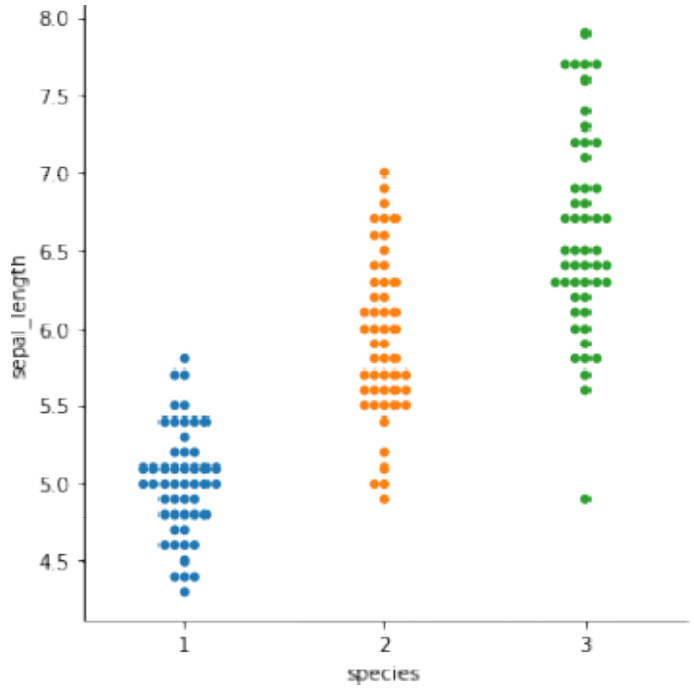
Out[19]:

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



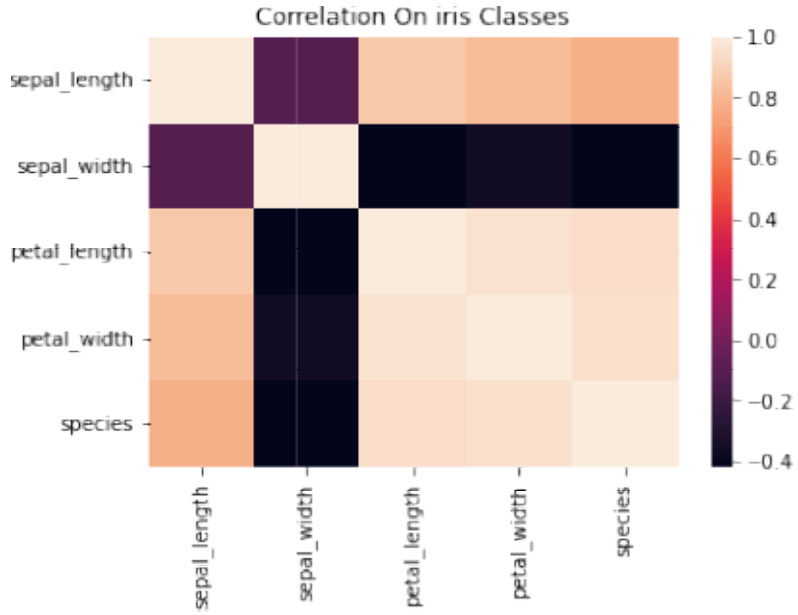

```
df = pd.read_csv("iris.csv")
varlist=['species'] def
binary_map(x):
        return x.map({'setosa': 1, 'versicolor':2,'virginica':3})
df[varlist] = df[varlist].apply(binary_map)
X = df.iloc[:,:-1].values
Y = df.iloc[:,-1:].values
df.tail()

# Catplot is used to show figure-level interface for drawing categorical plots
on sns.catplot(data=df, kind="swarm", x="species", y="sepal_length")
```

Out[20]: <seaborn.axisgrid.FacetGrid at 0x7f5601c75850>




```
plt.figure(1)
sns.heatmap(df.corr())
plt.title('Correlation On iris Classes')
```

Out[21]: Text(0.5, 1.0, 'Correlation On iris Classes')



In

In [ ]:
```
X = df.iloc[:,:-1]
y = df.iloc[:, -1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, randc
```

```
#Create the SVM model
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
#Fit the model for the data

classifier.fit(X_train, y_train)
```

```
#Make the prediction
y_pred = classifier.predict(X_test)
```

[ ]:
```
fr m sklearn.metrics import confusion_matrix cm = confusion_matrix(y_test, y_pred)
print(c )
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, c
print("Accuracy: {:.2f} %".format(accuracies.mean()*100)) print("Standard
Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
Accuracy: 98.18 %
Standard Deviation: 3.64 %
```

Conclusion: Here we have performed SVM with seaborn.

PRACTICAL-9

AIM:
Implementing Neural Net for classification .

In [ ]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.utils import to_categorical
```

In [ ]:
```
import tensorflow as tf

# Loading the dataset
mnist = tf.keras.datasets.mnist

# Divide into training and test dataset
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnis
t.npz)
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

In [ ]:
```
import tensorflow as tf

# Loading the dataset
mnist = tf.keras.datasets.mnist

# Divide into training and test dataset
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

In [ ]:
```
X_train_digit = x_train.reshape(60000, 784)
X_test_digit = x_test.reshape(10000, 784)
```

In [ ]:
```
y_train_digit = to_categorical(y_train, num_classes=10)

y_test_digit = to_categorical(y_test, num_classes=10)
```

In [ ]:
```
# Build the model
# 3 layers, 2 layers with 64 neurons + ReLu activation function
# l layer with 10 neuron and softmax function (Maximum entropy)
model = Sequential()
model.add(Dense(64,activation='relu',input_dim = 784))
model.add(Dense(64,activation='relu'))
model.add(Dense(10,activation='softmax'))
```

4

In [ ]:
```
# Complie the model
model.compile(
    optimizer = 'adam',
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
    )
```

In [ ]:
```
model.fit(X_train_digit, y_train_digit, batch_size=100, epochs=5)
```

```
600/600 [==============================] - 2s 3ms/step - loss: 0.1314 - accurac
y: 0.9602 Epoch 4/5
600/600 [==============================] - 2s 3ms/step - loss: 0.1028 - accurac
y: 0.9687 Epoch 5/5
600/600 [==============================] - 2s 3ms/step - loss: 0.0858 - accurac
y: 0.9738
```

Out[70]: <keras.callbacks.History at 0x7f686aa96310>

In [ ]:
```
# Evaluate the model
model.evaluate(
    X_test_digit,
    y_test_digit
)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1012 - accurac
y: 0.9679
```

Out[74]: [0.10118868947029114, 0.9678999781608582]

In [ ]:
```
indices = np.random.randint(100,size=25)
```

In [ ]:
```
# predict on the first 5 test images
predictions = model.predict(X_test_digit[indices,:])
# Print model predictions
print(np.argmax(predictions, axis = 1))
```

```
[2 6 4 0 6 9 7 4 1 7 2 5 5 5 7 2 1 0 9 5 0 9 4 3 3]
```

Conclusion: Here we have performed **Neural Net for classification of hand written digits**

PRACTICAL-10

AIM:

Implement Convolutional neural network for hand written digits classification. Tune it and compare it with practical 8.

In [ ]:
```
# Import necessary libraries
import numpy as np import
pandas as pd
from tensorflow.keras.datasets import
mnist from matplotlib import pyplot as plt
import matplotlib.pyplot as plt from
keras.models import Sequential from
tensorflow.keras import datasets from
keras.layers import Dense import
tensorflow as tf
from tensorflow.keras.utils import to_categorical from tensorflow.keras.models
import Sequential from tensorflow.keras.layers import Dense, Dropout,
Activation, Flatten, Conv2D,
```

In [ ]:
```
# load dataset
(trainX, trainY), (testX, testY) = mnist.load_data()
# reshape dataset to have a single channel trainX =
trainX.reshape((trainX.shape[0], 28, 28, 1)) testX =
testX.reshape((testX.shape[0], 28, 28, 1))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnis
t.npz)
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

In [ ]:
```
from keras.layers import Conv2D, MaxPool2D,Flatten, Dense, Dropout
from keras.models import Sequential
```

In [ ]:
```
def create_model():
model = Sequential()
    model.add(Conv2D(filters = 32, kernel_size = (3,3), activation = 'relu',
inpu     model.add(Conv2D(filters = 32, kernel_size = (5,5), activation =
'relu'))     model.add(MaxPool2D(pool_size = (2,2)))
model.add(Dropout(0.25))
    model.add(Conv2D(filters = 64, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPool2D(pool_size = (2,2), strides = (2,2)))
model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.25))
    model.add(Dense(10, activation = 'softmax'))
return model
```

In [ ]:
```
model = create_model()
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
['
```

In [ ]:
```
import tensorflow as tf

# Loading the dataset
mnist = tf.keras.datasets.mnist

# Divide into training and test dataset
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

In [ ]:
```
x_train = tf.keras.utils.normalize(x_train, axis = 1)
x_test = tf.keras.utils.normalize(x_test, axis = 1)
```

In [ ]:
```
IMG_SIZE=28
x_trainr = np.array(x_train).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
x_testr = np.array(x_test).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,

# Creating the network
model = Sequential()

### First Convolution Layer
# 64 -> number of filters, (3,3) -> size of each kernal,
model.add(Conv2D(64, (3,3), input_shape = x_trainr.shape[1:])) # For first layer
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

### Second Convolution Layer
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

### Third Convolution Layer
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

### Fully connected layer 1
model.add(Flatten())
model.add(Dense(64))
model.add(Activation("relu"))

### Fully connected layer 2
model.add(Dense(32))
model.add(Activation("relu"))

### Fully connected layer 3, output layer must be equal to number of classes
model.add(Dense(10))
model.add(Activation("softmax"))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 64)        640
 activation (Activation)     (None, 26, 26, 64)        0
 max_pooling2d_2 (MaxPooling  (None, 13, 13, 64)       0
 2D)
 conv2d_4 (Conv2D)           (None, 11, 11, 64)        36928
 activation_1 (Activation)   (None, 11, 11, 64)        0
 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)
 conv2d_5 (Conv2D)           (None, 3, 3, 64)          36928
 activation_2 (Activation)   (None, 3, 3, 64)          0
 max_pooling2d_4 (MaxPooling  (None, 1, 1, 64)         0
 2D)
 flatten_1 (Flatten)         (None, 64)                0
 dense_2 (Dense)             (None, 64)                4160
 activation_3 (Activation)   (None, 64)                0
 dense_3 (Dense)             (None, 32)                2080
 activation_4 (Activation)   (None, 32)                0
 dense_4 (Dense)             (None, 10)                330
 activation_5 (Activation)   (None, 10)                0
=================================================================
Total params: 81,066
Trainable params: 81,066
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=[
model.fit(x_trainr, y_train, epochs=5, validation_split = 0.3)
```

```
Epoch 1/5
1313/1313 [==============================] - 76s 57ms/step - loss: 0.3894 - acc
uracy: 0.8749 - val_loss: 0.1518 - val_accuracy: 0.9536
Epoch 2/5
1313/1313 [==============================] - 68s 52ms/step - loss: 0.1210 - acc
uracy: 0.9629 - val_loss: 0.1072 - val_accuracy: 0.9671
Epoch 3/5
1313/1313 [==============================] - 68s 52ms/step - loss: 0.0863 - acc
uracy: 0.9734 - val_loss: 0.0872 - val_accuracy: 0.9738
Epoch 4/5
1313/1313 [==============================] - 68s 52ms/step - loss: 0.0676 - acc
uracy: 0.9790 - val_loss: 0.0962 - val_accuracy: 0.9694
Epoch 5/5
1313/1313 [==============================] - 68s 52ms/step - loss: 0.0556 - acc
uracy: 0.9824 - val_loss: 0.0801 - val_accuracy: 0.9765
```

Out[49]: <keras.callbacks.History at 0x7fb3f7f41050>

In [ ]:
```
# Evaluating the accuracy on the test data test_loss,
test_acc = model.evaluate(x_testr, y_test) print("Test
```

```
Loss on 10,000 test samples", test_loss) print("Test
Accuracy on 10,000 test samples", test_acc)
```

```
313/313 [==============================] - 4s 13ms/step - loss: 0.0766 - accura
cy: 0.9779
Test Loss on 10,000 test samples 0.07658391445875168
Test Accuracy on 10,000 test samples 0.9779000282287598
```

In practical 7 we get an accuracy of 96.78% when we use ANN but when we use CNN for classification in practical 8 we get an accuracy of 97.7% which is more than practical 7.

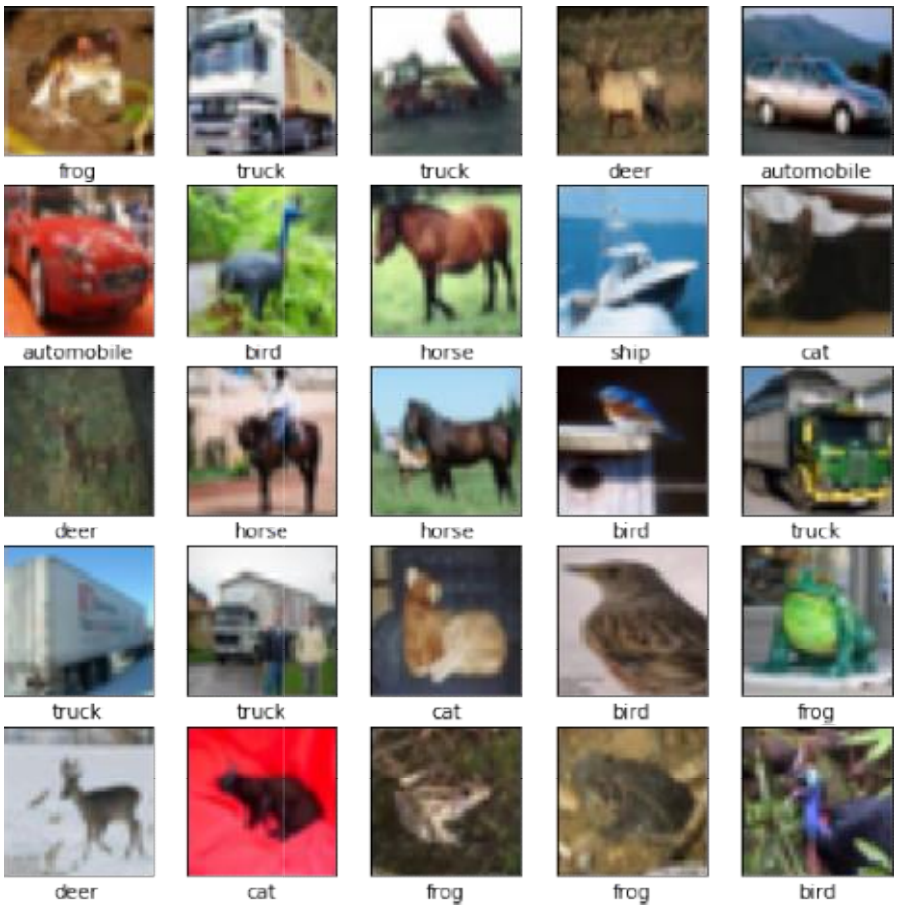
Apply Convolutional neural network on image classification data of your choice and write all steps for hyper parameter optimization. (use Keras library)

In [ ]:
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
(https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170500096/170498071 [==============================] - 13s 0us/step
170508288/170498071 [==============================] - 13s 0us/step
```

In [ ]:
```
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    #which is why we need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```




```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential"
```

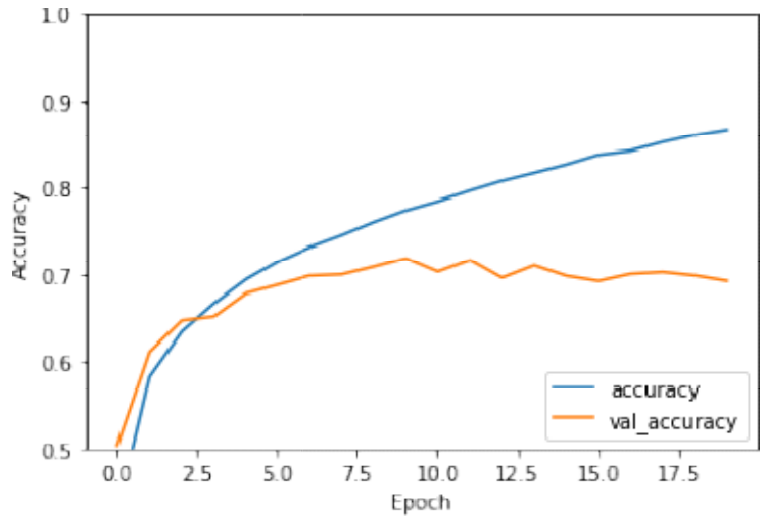
```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896
 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)
 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928
 flatten (Flatten)           (None, 1024)              0
 dense (Dense)               (None, 64)                65600
 dense_1 (Dense)             (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True
metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=20,
validation_data=(test_images, test_labels))
```

```
Epoch 1/20
1563/1563 [==============================] - 17s 5ms/step - loss: 1.5657 - ac
curacy: 0.4289 - val_loss: 1.4035 - val_accuracy: 0.5046
Epoch 2/20
1563/1563 [==============================] - 7s 4ms/step - loss: 1.1758 - acc
uracy: 0.5834 - val_loss: 1.0962 - val_accuracy: 0.6119
Epoch 3/20
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0394 - acc
uracy: 0.6350 - val_loss: 1.0012 - val_accuracy: 0.6476
Epoch 4/20
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9472 - acc
uracy: 0.6671 - val_loss: 0.9865 - val_accuracy: 0.6521
Epoch 5/20
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8756 - acc
uracy: 0.6944 - val_loss: 0.9202 - val_accuracy: 0.6785
Epoch 6/20
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8152 - acc
uracy: 0.7154 - val_loss: 0.9075 - val_accuracy: 0.6884
Epoch 7/20
1563/1563 [==============================] - 7s 5ms/step - loss: 0.7648 - acc
uracy: 0.7320 - val_loss: 0.8708 - val_accuracy: 0.6986
Epoch 8/20
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7186 - acc
uracy: 0.7456 - val_loss: 0.8851 - val_accuracy: 0.7001
Epoch 9/20
1563/1563 [==============================] - 8s 5ms/step - loss: 0.6799 - acc
uracy: 0.7609 - val_loss: 0.8560 - val_accuracy: 0.7092
Epoch 10/20
```

In [ ]:
```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label =
'val_accuracy') plt.xlabel('Epoch') plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,
test_labels,
verbose=2)
```

```
313/313 - 1s - loss: 1.1769 - accuracy: 0.6926 - 1s/epoch - 4ms/step
```



In [ ]:
```
print('Test Accuracy is',test_acc)
```

```
Test Accuracy is 0.6926000118255615
```

Conclusion: Here we have performed CNN for Image Classification.

Practical 11

AIM:
Unsupervised Learning PCA for improving effectiveness and efficiency of ML task.

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
import os
```

In [2]:
```
iris_data = pd.read_csv("../input/Iris.csv",index_col='Id')
```

In [3]:
```
iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), object(1) memory
usage: 7.0+ KB
```

Out[4]:

In [4]:
```
iris_data.describe()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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

Out[5]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species

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

1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

In [6]:
```
## Label encoding since the algorithms we are going to use do not take non numeri
iris_dat .Species.replace({'Iris-setosa':0,'Iris-versicolor':1, 'Iris-virginica':
```

In [7]:
```
iris_data.head()
```

Out[7]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	0
2	4.9	3.0	1.4	0.2	0
3	4.7	3.2	1.3	0.2	0
4	4.6	3.1	1.5	0.2	0
5	5.0	3.6	1.4	0.2	0

In [14]:
```
X = iris_data.drop(['Species'],axis=1)
y = iris_data.Species
```

In [18]:
```
from sklearn.decomposition import PCA
pca = PCA()
X_new = pca.fit_transform(X)
```

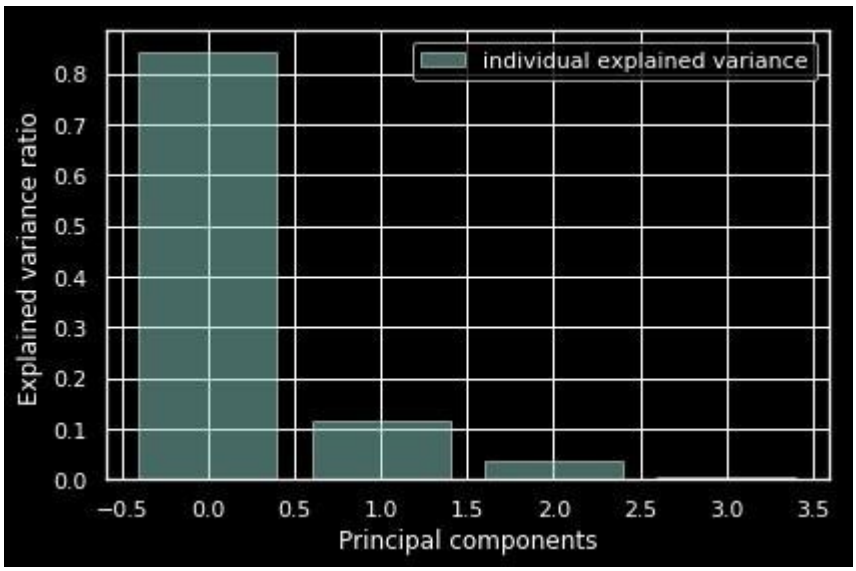
In [19]:
```
pca.get_covariance()
```

Out[19]:
```
array([[ 0.05290845, -0.00454496,  0.05996621,  0.05982683],       [-
  0.00454496,  0.03263959, -0.02271983, -0.02048285],
       [ 0.05996621, -0.02271983,  0.08943348,  0.09155279],
       [ 0.05982683, -0.02048285,  0.09155279,  0.1011136 ]])
```

In [20]:
```
explained_variance=pca.explained_variance_ratio_
explained_variance
```

Out[20]:
```
array([0.84141901, 0.11732474, 0.03490564, 0.00635061])
```

In [21]:
```
with plt.style.context('dark_background'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(4), explained_variance, alpha=0.5, align='center',
            label='individual explained variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



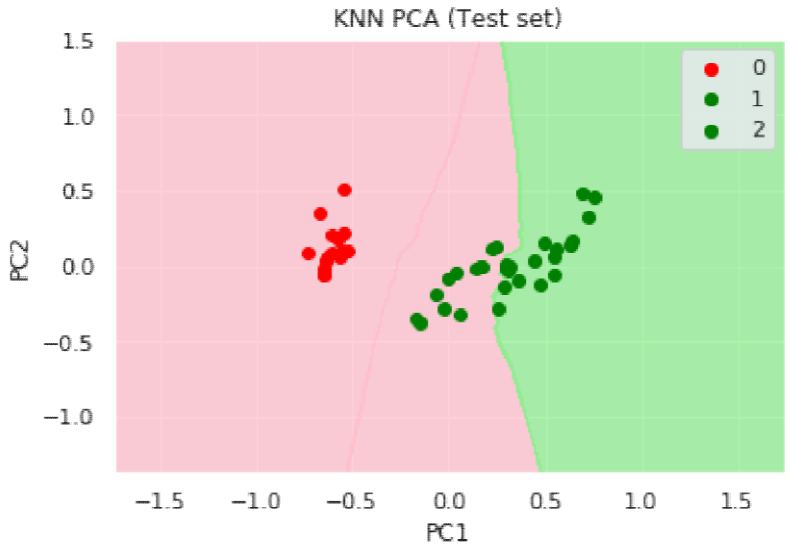
In [22]:
```
pca=PCA(n_components=3)
X_new=pca.fit_transform(X)
```

In [23]:
```
X_train_new, X_test_new, y_train, y_test = train_test_split(X_new, y, test_size =
```

In [24]:
```
knn_pca = KNeighborsClassifier(7)
knn_pca.fit(X_train_new,y_train)
print("Train score after PCA",knn_pca.score(X_train_new,y_train),"%")
print("Test score after PCA",knn_pca.score(X_test_new,y_test),"%")
```

```
Train score after PCA 0.9619047619047619 %
Test score after PCA 0.9777777777777777 %
```

In [25]:
```
# Visualising the Test set results
classifier = knn_pca
from matplotlib.colors import ListedColormap
X_set, y_set = X_test_new, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel(),np.zeros
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('KNN PCA (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```



Conclusion : Here we have performed **Unsupervised Learning PCA for improving effectiveness and efficiency of ML task.**

PRACTICAL 12

Mini Project Object detection

In [3]:

```
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common
libraries from tqdm import
tqdm import os, cv2, random
import numpy as np import
pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (20.0,
10.0)
# import some common detectron2 utilities
from detectron2 import model_zoo from
detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer from
detectron2.data import DatasetCatalog, MetadataCatalog
from detectron2.structures import BoxMode
```

```
def cv2_imshow(im):
    im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
plt.figure(), plt.imshow(im), plt.axis('off');
```

```
def denormalize_bboxes(bboxes, height, width):
    """Denormalize bounding boxes in format of (xmin, ymin, xmax,
ymax)."""     bboxes[:, [0, 2]] = bboxes[:, [0, 2]] * width     bboxes[:,
[1, 3]] = bboxes[:, [1, 3]] * height     return np.round(bboxes)
 def
get_detectron_dicts(annot_df):
    """
    Create Detectron2's standard dataset from an annotation file.

Args:
        annot_df (pd.DataFrame): annotation dataframe.
    Return:
        dataset_dicts (list[dict]): List of annotation dictionaries for
Detectron2.
    """
    # Get image ids
    img_ids = annot_df["ImageID"].unique().tolist()

    dataset_dicts = []     for
img_id in tqdm(img_ids):
        file_name = f'images/{img_id}.jpg'         if
not os.path.exists(file_name): continue
height, width = cv2.imread(file_name).shape[:2]

record = {}
        record['file_name'] = file_name
record['image_id'] = img_id
```

1. Register Datasets

In [4]:

```
        record['height'] = height
record['width'] = width

        # Extract bboxes from annotation file
        bboxes = annot_df[['XMin', 'YMin', 'XMax', 'YMax']][annot_df['ImageID']
== i         bboxes = denormalize_bboxes(bboxes, height, width)         class_ids
= annot_df[['ClassID']][annot_df['ImageID'] == img_id].values

        annots = []         for i, bbox in
enumerate(bboxes.tolist()):
            annot = {
                "bbox": bbox,
                "bbox_mode": BoxMode.XYXY_ABS,
                "category_id": int(class_ids[i]),
            }
            annots.append(annot)

        record["annotations"] = annots
dataset_dicts.append(record)     return
dataset_dicts
```

In [5]:
```
# Specify target classes and create `class2id` dict
target_classes = ['Camera', 'Tripod'] class2id = {k:
v for v, k in enumerate(target_classes)}
print(class2id)
```

```
{'Camera': 0, 'Tripod': 1}
```

In [6]:
```
# Load target annotations
train_df = pd.read_csv("train-annotations-bbox-truncated.csv")
val_df = pd.read_csv("validation-annotations-bbox-
truncated.csv")
# Register dataset with Detectron2
print("Registering Datasets...")
DatasetCatalog.register("camera_tripod_train", lambda d=train_df:
get_detectron_dict
MetadataCatalog.get("camera_tripod_train").set(thing_classes=target_classes)
DatasetCatalog.register("camera_tripod_val", lambda d=val_df:
get_detectron_dicts(d)
MetadataCatalog.get("camera_tripod_val").set(thing_classes=target_classes)
print("Done!")

# Get metadata. It helps show class labels when we visualize bounding boxes
camera_tripod_metadata = MetadataCatalog.get("camera_tripod_train")
```

```
Registering      Datasets...
Done!
```

```
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
```

2. Train

In [7]:

In [8]:
```
run_name = 'exp-1' if
os.path.exists(run_name):
    !rm -rf $run_name

# Set up model and training configurations cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
Detection/faster_rcnn_R_50_FPN_3 cfg.MODEL.WEIGHTS =
model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FP
```

```
cfg.DATASETS.TRAIN = ("camera_tripod_train",)
cfg.DATASETS.TEST = ("camera_tripod_val",)

# Training hyperparameters
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE =
128 cfg.SOLVER.BASE_LR = 5e-4
cfg.SOLVER.MAX_ITER = 300
cfg.SOLVER.WARMUP_ITERS = 100
cfg.SOLVER.STEPS = (200,)

# Specify class number
cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(target_classes)
cfg.MODEL.RETINANET.NUM_CLASSES = len(target_classes) # if using RetinaNet

cfg.DATALOADER.NUM_WORKERS = 6
cfg.OUTPUT_DIR = run_name
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

# Set up trainer
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

```
[07/14 15:42:06 d2.engine.defaults]: Model:
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (top_block): LastLevelMaxPool()
    (bottom_up): ResNet(
      (stem): BasicStem(
        (conv1): Conv2d(
          3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
      )
      (res2): Sequential(
        (0):            BottleneckBlock(
(shortcut): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv1): Conv2d(
            64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
```

```
              )
              (conv3): Conv2d(
                64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
                (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
              )
            )
            (1): BottleneckBlock(
              (conv1): Conv2d(
                256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
                (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
              )
              (conv2): Conv2d(
                64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
                (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
              )
              (conv3): Conv2d(
                64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
                (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
              )
            )
            (2): BottleneckBlock(
              (conv1): Conv2d(
                256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
                (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
              )
              (conv2): Conv2d(
                64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
                (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
              )
              (conv3): Conv2d(
                64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
                (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
              )
)
          )
        )
        (res3): Sequential(
          (0):            BottleneckBlock(
(shortcut): Conv2d(
              256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
              (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
            )
            (conv1): Conv2d(
              256, 128, kernel_size=(1, 1), stride=(2, 2), bias=False
              (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
            )
            (conv2): Conv2d(
              128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
              (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
            )
            (conv3): Conv2d(
              128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
              (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
)
          )
          (1): BottleneckBlock(
            (conv1): Conv2d(
              512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
              (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
            )
```

```
      (conv2): Conv2d(
        128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
      )
      (conv3): Conv2d(
        128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
      )
    )
    (2): BottleneckBlock(
      (conv1): Conv2d(
        512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
      )
      (conv2): Conv2d(
        128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
)
      (conv3): Conv2d(
        128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
)
    )
    (3): BottleneckBlock(
      (conv1): Conv2d(
        512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
      )
      (conv2): Conv2d(
        128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
      )
      (conv3): Conv2d(
        128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
      )
    )
  )
  (res4): Sequential(
    (0):            BottleneckBlock(
(shortcut): Conv2d(
        512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False
        (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
      )
      (conv1): Conv2d(
        512, 256, kernel_size=(1, 1), stride=(2, 2), bias=False
        (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
      )
      (conv2): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
      )
      (conv3): Conv2d(
        256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
        (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
      )
    )
    (1): BottleneckBlock(
      (conv1): Conv2d(
```

```
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
)
  )
  (2): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (3): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
)
  )
  (4): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (5): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
```

```
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
)
        )
      )
      (res5): Sequential(
        (0):            BottleneckBlock(
(shortcut): Conv2d(
            1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
          (conv1): Conv2d(
            1024, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
)
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
)
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
)
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
)
        )
      )
    )
```

```
  )
  (proposal_generator): RPN(
    (anchor_generator): DefaultAnchorGenerator(
      (cell_anchors): BufferList()
    )
    (rpn_head): StandardRPNHead(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (objectness_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
      (anchor_deltas): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (roi_heads): StandardROIHeads(
    (box_pooler): ROIPooler(
      (level_poolers): ModuleList(
        (0): ROIAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0,
alig ned=True)
        (1): ROIAlign(output_size=(7, 7), spatial_scale=0.125, sampling_ratio=0,
ali gned=True)
        (2): ROIAlign(output_size=(7, 7), spatial_scale=0.0625, sampling_ratio=0,
al igned=True)
        (3): ROIAlign(output_size=(7, 7), spatial_scale=0.03125, sampling_ratio=0,
a ligned=True)
      )
    )
    (box_head): FastRCNNConvFCHead(
      (fc1): Linear(in_features=12544, out_features=1024, bias=True)
      (fc2): Linear(in_features=1024, out_features=1024, bias=True)
    )
    (box_predictor): FastRCNNOutputLayers(
      (cls_score): Linear(in_features=1024, out_features=3, bias=True)
      (bbox_pred): Linear(in_features=1024, out_features=8, bias=True)
)
  )
)
100%|▒▒▒▒▒▒▒▒▒▒▒▒| 6091/6091 [01:41<00:00, 59.82it/s]
[07/14 15:43:48 d2.data.build]: Removed 0 images with no usable annotations. 6091
im ages left.

[07/14 15:43:48 d2.data.build]: Distribution of instances among all 2 categories:
|  category  | #instances   |  category  | #instances   |
|:-----------:|:--------------|:-----------:|:--------------|
|   Camera   | 6404         |   Tripod   | 1446         |
|            |              |            |              |
|   total    | 7850         |            |              |
[07/14 15:43:48 d2.data.common]: Serializing 6091 elements to byte tensors and conca
tenating them all ...
[07/14 15:43:48 d2.data.common]: Serialized dataset takes 1.57 MiB
[07/14 15:43:48 d2.data.detection_utils]: TransformGens used in training:
[ResizeSho rtestEdge(short_edge_length=(640, 672, 704, 736, 768, 800),
max_size=1333, sample_st yle='choice'), RandomFlip()] [07/14 15:43:48
d2.data.build]: Using training sampler TrainingSampler
Unable to load 'roi_heads.box_predictor.cls_score.weight' to the model due to
incomp atible shapes: (81, 1024) in the checkpoint but (3, 1024) in the model!
Unable to load 'roi_heads.box_predictor.cls_score.bias' to the model due to
incompat ible shapes: (81,) in the checkpoint but (3,) in the model!
Unable to load 'roi_heads.box_predictor.bbox_pred.weight' to the model due to
incomp atible shapes: (320, 1024) in the checkpoint but (8, 1024) in the model!
```

```
Unable to load 'roi_heads.box_predictor.bbox_pred.bias' to the model due to
incompat ible shapes: (320,) in the checkpoint but (8,) in the model!
[07/14 15:43:49 d2.engine.train_loop]: Starting training from iteration 0
[07/14 15:43:55 d2.utils.events]:  eta: 0:01:26  iter: 19  total_loss: 1.440  loss_c
ls: 0.956  loss_box_reg: 0.518  loss_rpn_cls: 0.016  loss_rpn_loc:  0.013  time:
0.31 34  data_time: 0.0256  lr: 0.000095  max_mem: 4959M
[07/14 15:44:01 d2.utils.events]:  eta: 0:01:21  iter: 39  total_loss: 0.922  loss_c
ls: 0.442  loss_box_reg: 0.458  loss_rpn_cls: 0.015  loss_rpn_loc:  0.010  time:
0.31 32  data_time: 0.0097  lr: 0.000195  max_mem: 4959M
[07/14 15:44:08 d2.utils.events]:  eta: 0:01:15  iter: 59  total_loss: 0.853  loss_c
ls: 0.345  loss_box_reg: 0.474  loss_rpn_cls: 0.012  loss_rpn_loc:  0.010  time:
0.31 43  data_time: 0.0102  lr: 0.000295  max_mem: 4959M
[07/14 15:44:14 d2.utils.events]:  eta: 0:01:09  iter: 79  total_loss: 0.724  loss_c
ls: 0.272  loss_box_reg: 0.464  loss_rpn_cls: 0.011  loss_rpn_loc:  0.007  time:
0.31 37  data_time: 0.0095  lr: 0.000395  max_mem: 4959M
[07/14 15:44:20 d2.utils.events]:  eta: 0:01:03  iter: 99  total_loss: 0.806  loss_c
ls: 0.214  loss_box_reg: 0.540  loss_rpn_cls: 0.008  loss_rpn_loc:  0.009  time:
0.31 38  data_time: 0.0094  lr: 0.000495  max_mem: 4959M
[07/14 15:44:26 d2.utils.events]:  eta: 0:00:56  iter: 119  total_loss: 0.691  loss_
cls: 0.184  loss_box_reg: 0.483  loss_rpn_cls: 0.014  loss_rpn_loc: 0.009  time:
0.3 128  data_time: 0.0104  lr: 0.000500  max_mem: 4959M
[07/14 15:44:33 d2.utils.events]:  eta: 0:00:50  iter: 139  total_loss: 0.644  loss_
cls: 0.200  loss_box_reg: 0.444  loss_rpn_cls: 0.009  loss_rpn_loc: 0.007  time:
0.3 128  data_time: 0.0096  lr: 0.000500  max_mem: 4959M
[07/14 15:44:39 d2.utils.events]:  eta: 0:00:44  iter: 159  total_loss: 0.543  loss_
cls: 0.152  loss_box_reg: 0.390  loss_rpn_cls: 0.003  loss_rpn_loc: 0.009  time:
0.3 121  data_time: 0.0094  lr: 0.000500  max_mem: 4959M
[07/14 15:44:45 d2.utils.events]:  eta: 0:00:37  iter: 179  total_loss: 0.473  loss_
cls: 0.110  loss_box_reg: 0.334  loss_rpn_cls: 0.005  loss_rpn_loc: 0.011  time:
0.3 126  data_time: 0.0096  lr: 0.000500  max_mem: 4959M
[07/14 15:44:51 d2.utils.events]:  eta: 0:00:31  iter: 199  total_loss: 0.458  loss_
cls: 0.121  loss_box_reg: 0.321  loss_rpn_cls: 0.009  loss_rpn_loc: 0.010  time:
0.3 130  data_time: 0.0094  lr: 0.000500  max_mem: 4959M
[07/14 15:44:58 d2.utils.events]:  eta: 0:00:25  iter: 219  total_loss: 0.454  loss_
cls: 0.116  loss_box_reg: 0.322  loss_rpn_cls: 0.004  loss_rpn_loc: 0.007  time:
0.3 134  data_time: 0.0099  lr: 0.000050  max_mem: 4959M
[07/14 15:45:04 d2.utils.events]:  eta: 0:00:19  iter: 239  total_loss: 0.526  loss_
cls: 0.125  loss_box_reg: 0.360  loss_rpn_cls: 0.006  loss_rpn_loc: 0.007  time:
0.3 143  data_time: 0.0091  lr: 0.000050  max_mem: 4959M
[07/14 15:45:11 d2.utils.events]:  eta: 0:00:12  iter: 259  total_loss: 0.492  loss_
cls: 0.115  loss_box_reg: 0.347  loss_rpn_cls: 0.006  loss_rpn_loc: 0.008  time:
0.3 148  data_time: 0.0089  lr: 0.000050  max_mem: 4959M
 [07/14 15:45:17 d2.utils.events]:  eta: 0:00:06  iter: 279  total_loss: 0.439
loss_ cls: 0.110  loss_box_reg: 0.311  loss_rpn_cls: 0.002  loss_rpn_loc: 0.010
                                    time: 0.3
151  data_time: 0.0096  lr: 0.000050  max_mem: 4959M
100%|▒▒▒▒▒▒▒▒▒▒▒▒| 166/166 [00:02<00:00, 64.52it/s]
[07/14 15:45:27 d2.data.build]: Distribution of instances among all 2 categories:
|  category  | #instances   |  category  | #instances   |
|:----------:|:-------------|:----------:|:-------------|
|   Camera   | 153          |   Tripod   | 40           |
|            |              |            |              |
|   total    | 193          |            |              |
[07/14 15:45:27 d2.data.common]: Serializing 166 elements to byte tensors and concat
enating them all ...
[07/14 15:45:27 d2.data.common]: Serialized dataset takes 0.04 MiB
WARNING [07/14 15:45:27 d2.engine.defaults]: No evaluator found. Use `DefaultTraine
r.test(evaluators=)`, or implement its `build_evaluator` method.
```

```
 [07/14 15:45:27 d2.utils.events]:  eta: 0:00:00  iter: 299  total_loss: 0.478
 loss_ cls: 0.158  loss_box_reg: 0.325  loss_rpn_cls: 0.008  loss_rpn_loc: 0.006
                                      time: 0.3
154  data_time: 0.0095  lr: 0.000050  max_mem: 4959M
[07/14 15:45:27 d2.engine.hooks]: Overall training speed: 297 iterations in 0:01:33
 (0.3165 s / it)
[07/14 15:45:27 d2.engine.hooks]: Total training time: 0:01:37 (0:00:03 on hooks)
```

```
from detectron2.evaluation import COCOEvaluator

evaluator = COCOEvaluator("camera_tripod_val", cfg, False,
output_dir=cfg.OUTPUT_DIR trainer.test(cfg=cfg,
             model=trainer.model,
evaluators=evaluator)
```

3. Evaluation

In [9]:

```
WARNING [07/14 15:45:27 d2.evaluation.coco_evaluation]: json_file was not found in
M etaDataCatalog for 'camera_tripod_val'. Trying to convert it to COCO format ...
[07/14  15:45:27  d2.data.datasets.coco]:  Converting  annotations  of  dataset
'camera_tr ipod_val' to COCO format ...)
100%|□□□□□□□□□□□| 166/166 [00:02<00:00, 65.06it/s]
[07/14 15:45:29 d2.data.datasets.coco]: Converting dataset dicts into COCO format
[07/14 15:45:29 d2.data.datasets.coco]: Conversion finished, #images: 166,
#annotati ons: 193
[07/14 15:45:29 d2.data.datasets.coco]: Caching COCO format annotations at 'exp-
1/ca mera_tripod_val_coco_format.json' ...
100%|□□□□□□□□□□□| 166/166 [00:02<00:00, 65.31it/s]
[07/14 15:45:32 d2.data.common]: Serializing 166 elements to byte tensors and concat
enating them all ...
[07/14  15:45:32  d2.data.common]:  Serialized  dataset  takes  0.04  MiB
[07/14 15:45:32 d2.evaluation.evaluator]: Start inference on 166 images

[07/14 15:45:32 d2.evaluation.evaluator]: Inference done 11/166. 0.0399 s / img.
ETA =0:00:06
[07/14 15:45:37 d2.evaluation.evaluator]: Inference done 136/166. 0.0386 s / img.
ET A=0:00:01
[07/14  15:45:39  d2.evaluation.evaluator]:  Total  inference  time:  0:00:06.442064
(0.04
0013 s / img per device, on 1 devices)
[07/14 15:45:39 d2.evaluation.evaluator]: Total inference pure compute time: 0:00:06
(0.038252 s / img per device, on 1 devices)
[07/14 15:45:39 d2.evaluation.coco_evaluation]: Preparing results for COCO format
 ...
[07/14   15:45:39   d2.evaluation.coco_evaluation]:   Saving   results   to   exp-
1/coco_instanc es_results.json
[07/14 15:45:39 d2.evaluation.coco_evaluation]: Evaluating predictions ...
```

```
Loading and preparing
results... DONE (t=0.00s)
creating index... index
created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.22s).
Accumulating evaluation results...
DONE (t=0.05s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.415
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.703
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.387
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.377
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.426
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 1 ] = 0.448
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.589
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.604
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.433
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.618
[07/14 15:45:39 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 41.476 | 70.260 | 38.666 |  nan  | 37.693 | 42.611 |
[07/14 15:45:39 d2.evaluation.coco_evaluation]: Note that some metrics cannot be
com puted.
[07/14 15:45:39 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|
| Camera     | 55.735 | Tripod     | 27.217 |
[07/14 15:45:39 d2.engine.defaults]: Evaluation results for camera_tripod_val in
csv format:
[07/14 15:45:39 d2.evaluation.testing]: copypaste: Task: bbox
[07/14 15:45:39 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl
[07/14            15:45:39           d2.evaluation.testing]:           copypaste:
41.4764,70.2595,38.6656,nan,37.69
31,42.6113
OrderedDict([('bbox',
```

Out[9]:

```
              {'AP': 41.47643344526002,
               'AP50': 70.25950629106936,
               'AP75':       38.66556641542805,
'APs': nan,
               'APm': 37.693069306930695,
               'APl': 42.61129782599682,
               'AP-Camera': 55.735393725928816,
               'AP-Tripod': 27.217473164591215})])
```

The model achieves 41 mAP after only 300 training steps. That's impressive!

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set the testing threshold for
this m
```

Inference on the validation set

In [18]:

```
predictor = DefaultPredictor(cfg)
```

In [11]:

```
dataset_dicts = get_detectron_dicts(val_df)
```

```
100%|▒▒▒▒▒▒▒▒▒▒▒▒| 166/166 [00:02<00:00, 64.63it/s]
```

In [19]:

```
for d in random.sample(dataset_dicts, 3):
    im = cv2.imread(d["file_name"])     outputs = predictor(im)     v =
Visualizer(im[:, :, ::-1], metadata=camera_tripod_metadata, scale=0.8)
pred = v.draw_instance_predictions(outputs["instances"][:2].to("cpu"))
cv2_imshow(pred.get_image()[:, :, ::-1])     plt.title("Prediction");
```

Prediction

Prediction

Prediction

4.Production

After training our model, we should save the configurations to put the model into production.

In [13]:
```
# Save config with open(os.path.join(cfg.OUTPUT_DIR,
"config.yaml"), "w") as f:     f.write(cfg.dump())
```

Now we can load the trained model and run it on images downloaded from the internet.

In [14]:
```
# Load config
cfg = get_cfg()
cfg.merge_from_file('exp-1/config.yaml')
cfg.MODEL.WEIGHTS = 'exp-1/model_final.pth'

# Set up predictor
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.95
predictor = DefaultPredictor(cfg)
```

In [15]:
```
%%capture
# Download an image
!wget -O "./sample.jpg" https://independenttravelcats.com/wp-
content/uploads/2017/11
```

In [16]:
```
im = cv2.imread("./sample.jpg")
outputs = predictor(im)

# Visualize
v = Visualizer(im[:, :, ::-1], camera_tripod_metadata, scale=0.6)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1]);
```