



Pandit Deendayal Energy University

School of Technology

**Department of Computer Science and
Engineering**

Lab Record

Course: Machine Learning Lab

Course Code: 20CP401P

Program: B-Tech.

A.Y.: 2022-2023

Name of Student: Priyank Rana

Roll No.: 19BCP099

Course Instructor: Dr. Sonam Nahar

INDEX

| SR. NO | EXPERIMENTS | Faculty Sign. |
|-------------------|--|----------------------|
| 1 | Python Practice Assignment | |
| 2 | Assignment on Exploratory data analysis | |
| 3 | Implement of K-Nearest Neighbor Algorithm | |
| 4 | Implement of Logistic Regression Algorithm | |
| 5 | Perform handwritten digit classification using logistic regression | |
| 6 | Perform handwritten digit classification using SVM and neural network model. | |
| 7 | Course Project | |

Name - Priyank Rana

Roll no -19BCP099

Branch -Computer Engineering

Subject: Machine Learning Lab

Division: 2 (G3 Batch)


MACHINE LEARNING LAB - 1 (August 2022, Week -1)

Python Practice

Q1. Write a function that converts a decimal number to binary number.

```
decimal = int(input("Enter a decimal number \n"))
binary = 0
c = 0
temp = decimal
#find binary value using while loop
while(temp > 0):
    binary = ((temp%2)*(10**c)) + binary
    temp = int(temp/2)
    c += 1

print("Binary of {x} is: {y}".format(x=decimal,y=binary))
```

 Enter a decimal number
232
Binary of 232 is: 11101000

Q2. Write a function to compute the sigmoid of a vector of values.

A sigmoid of a real number x is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$

For example: Input: $[x_1, x_2, x_3, x_4]$

Output: $[\sigma(x_1), \sigma(x_2), \sigma(x_3), \sigma(x_4)]$

```
import numpy as np
vector = np.array([5,2,1,3])
print(1 / (1 + np.exp((-1*vector))))

[0.99330715 0.88079708 0.73105858 0.95257413]
```

Q3. Write a function to compute the derivative of the sigmoid function with respect to its input x . Here, the x is a vector.

The derivative of a sigmoid is defined as: $\sigma(x)(1 - \sigma(x))$

TRY TO PROVE THIS IN YOUR COPY

```
def sigmoid(x):
    return (1/(1+np.exp(-x)))
def derivativeOfSigmoid(x):
    return sigmoid(x)*(1-sigmoid(x))
print(derivativeOfSigmoid(vector))

[0.00664806 0.10499359 0.19661193 0.04517666]
```

Q4. Write a function to normalize the rows of a matrix. After applying this function to an input matrix x of size $m \times n$, each row of x should be a vector of unit length.

```
#Answer4
matrix = np.matrix("3 4; 12 5")
for row in matrix:
    for elem in row:
        print(elem / np.linalg.norm(row))

[[0.6 0.8]]
[[0.92307692 0.38461538]]
```

Q5. Write a program which can map() and filter() to make a list whose elements are cube of even numbers in a list.

```
def even(x):
    if x%2:
        return False
    return True
def cube(x):
    return x*x*x
print(list(map(cube,list(filter(even,vector)))))

[8]
```

Q6. consider the marks list of class students given two lists

Students =

['student1','student2','student3','student4','student5','student6','student7','student8','student9','student10']

Marks = [45, 78, 12, 14, 48, 43, 45, 98, 35, 80]

from the above two lists the Student[0] got Marks[0], Student[1] got Marks[1] and so on

your task is to print the name of students

a. Who got top 5 ranks, in the descending order of marks

b. Who got least 5 ranks, in the increasing order of marks

d. Who got marks between >25th percentile <75th percentile, in the increasing order of marks

Ex 1:

```
Students=['student1','student2','student3','student4','student5','student6','student7','student8','student9','student10']
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
```

a.

```
student8 98
student10 80
student2 78
student5 48
student7 47
```

b.

```
student3 12
student4 14
student9 35
student6 43
student1 45
```

c.

```
student9 35
student6 43
student1 45
student7 47
student5 48
```

```
Students = ['student1','student2','student3','student4','student5','student6','student7','student8','student9','student10']
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
data = []
for i in range(len(Marks)):
    data.append((Marks[i],Students[i]))
l = len(data)
data = sorted(data)
print('a.',data[:5])
print('b.',data[-5:])
print('c.',data[int(l/4):int(3*l/4)])
```

```
a. [(98, 'student8'), (80, 'student10'), (78, 'student2'), (48, 'student5'), (47, 'student7')]
b. [(12, 'student3'), (14, 'student4'), (35, 'student9'), (43, 'student6'), (45, 'student1')]
c. [(35, 'student9'), (43, 'student6'), (45, 'student1'), (47, 'student7'), (48, 'student5')]
```

Q7. consider you have given n data points in the form of list of tuples like $S=[(x_1,y_1),(x_2,y_2), (x_3,y_3),(x_4,y_4),(x_5,y_5),\dots,(x_n,y_n)]$ and a point $P=(p,q)$

your task is to find 5 closest points(based on cosine distance) in S from P <brs (x,y) and (p,q) is

defined as $\cos^{-1}\left(\frac{(x \cdot p + y \cdot q)}{\sqrt{(x^2 + y^2)} \cdot \sqrt{(p^2 + q^2)}}\right)$ > cosine distance between two point

Ex:

$S = [(1,2),(3,4),(-1,1),(6,-7),(0,6),(-5,-8),(-1,-1),(6,0),(1,-1)]$
 $P = (3,-4)$

Output:

$(6,-7)$
 $(1,-1)$
 $(6,0)$
 $(-5,-8)$
 $(-1,-1)$

```
import math
def cosDist(x,y,p,q):
    return math.acos(((x*p) + (y*q))/(math.sqrt(x*x + y*y) * math.sqrt(p*p + q*q)))
S= [(1,2),(3,4),(-1,1),(6,-7),(0,6),(-5,-8),(-1,-1),(6,0),(1,-1)]
P= (3,-4)
DistanceList = []
for x,y in S:
    DistanceList.append((cosDist(x,y,P[0],P[1]),(x,y)))
DistanceList.sort()
n=5
for dList in DistanceList[:n]:
    print(dList[1])

(6, -7)
(1, -1)
(6, 0)
(-5, -8)
(-1, -1)
```

Q8: Given two sentences S1, S2 You will be given a list of lists, each sublist will be of length 2 i.e. $[[x,y],[p,q],[l,m],[r,s]]$ consider its like a matrix of n rows and two columns

- the first column Y will contain interger values
- the second column Y_{score} will be having float values

Your task is to find the value of

$$f(Y, Y_{score}) = -1 * \frac{1}{n} \sum_{foreach Y, Y_{score} pair} (Y \log_{10}(Y_{score}) + (1 - Y) \log_{10}(1 - Y_{score}))$$

here n is the number of rows in the matrix

Ex:

```
[[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1, 0.9], [1, 0.8]]
```

output:

```
0.4243099
```

$$\frac{-1}{8} \cdot ((1 \cdot \log_{10}(0.4) + 0 \cdot \log_{10}(0.6)) + (0 \cdot \log_{10}(0.5) + 1 \cdot \log_{10}(0.5)) + \dots + (1 \cdot \log_{10}($$

```
from math import log
arr = [[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1, 0.9], [1, 0.8]]
def f(arr):
    n = len(arr)
    sum = 0
    for y,ys in arr:
        sum = sum + ((-1)*((y*log(ys,10))+((1-y)*log(1-ys,10))))/n
    return sum
print(f(arr))

0.42430993457031635
```

[Colab paid products](#) - [Cancel contracts here](#)



▼ Assignment 2

Problem Statement : Perform Exploratory data analysis on Cance Survival Dataset

Download Haberman Cancer Survival dataset from Kaggle. You may have to create a Kaggle account to donwload data. (<https://www.kaggle.com/gilsousa/habermans-survival-data-set>) or you can also run the below cell and load the data directly.

```
import pandas as pd
df = pd.read_csv('haberman.csv')
df.head()
```

| | 30 | 64 | 1 | 1.1 |
|---|----|----|----|-----|
| 0 | 30 | 62 | 3 | 1 |
| 1 | 30 | 65 | 0 | 1 |
| 2 | 31 | 59 | 2 | 1 |
| 3 | 31 | 65 | 4 | 1 |
| 4 | 33 | 58 | 10 | 1 |

▼ 1.1 Analyze high level statistics of the dataset: number of points, numer of features, number of classes, data-points per class.

- You have to write all of your observations in Markdown cell with proper formatting.
- Do not write your observations as comments in code cells.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
plt.style.use('fivethirtyeight')
import numpy as np
from statsmodels import robust
```

```
haberman=pd.read_csv("haberman.csv",names=['age', 'operation_year', 'axil_nodes', 'surviva
haberman.head()
```



| | age | operation_year | axil_nodes | survival_status |
|---|-----|----------------|------------|-----------------|
| 0 | 30 | 64 | 1 | 1 |
| 1 | 30 | 62 | 3 | 1 |
| 2 | 30 | 65 | 0 | 1 |

```
print(haberman.shape)
haberman["survival_status"].value_counts()
print (haberman.columns)

(306, 4)
Index(['age', 'operation_year', 'axil_nodes', 'survival_status'], dtype='object')
```

```
haberman["survival_status"].value_counts()
```

```
1    225
2     81
Name: survival_status, dtype: int64
```

```
print(haberman["survival_status"].unique())
```

```
[1 2]
```

```
print(haberman.groupby("survival_status").count())
```

```
survival_status  age  operation_year  axil_nodes
1                225                225         225
2                 81                 81          81
```

306 instances of data and 4 attributes are present as can see by `haberman.shape`. The dtype in columns means data type which is object type for all columns present in data. Similarly you can also find the dtype of feature and class attribute and can find how many number of data points belongs to 1 class. you can conclude that there are 225 patients out of 306 were survived more than 5 years and only 81 patients survived less than 5 years

▼ 1.2 - Explain the objective of the problem.

(The objective for a problem can be defined as a brief explanation of problem that you are trying to solve using the given dataset)

Through this exploratory data analysis we are trying to conclude or predict survival status of patients who undergone from surgery. The attributes of this Haberman Dataset are:-

Number of Axillary nodes(Lymph Nodes)

Age

Operation Year

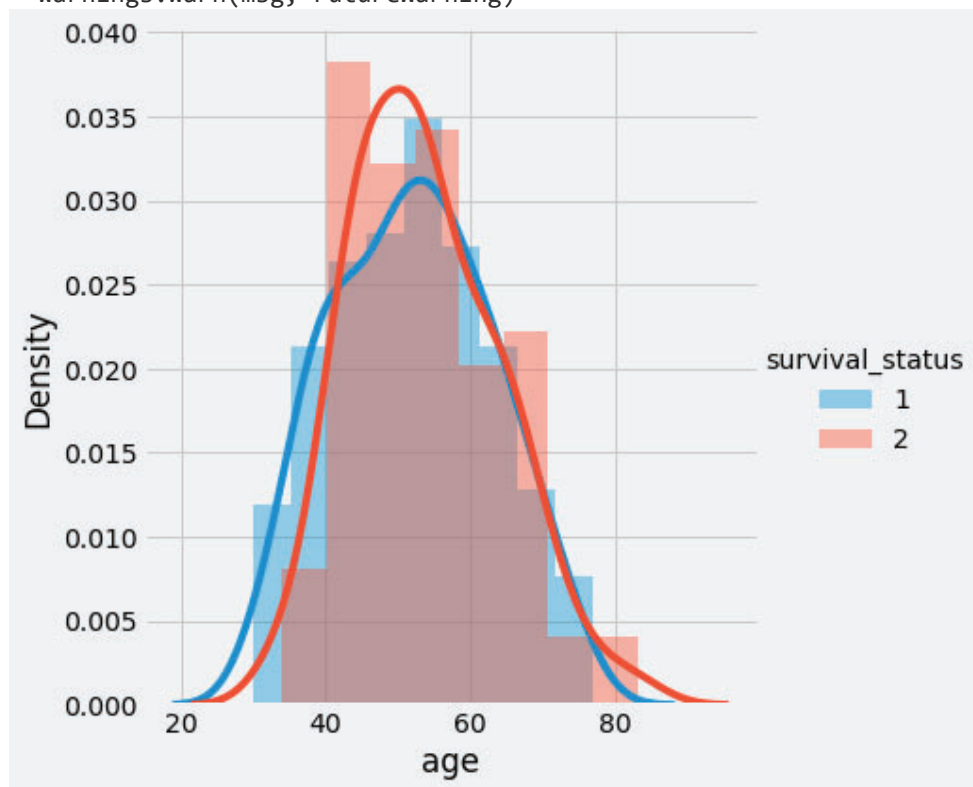
Survival Status

▼ 1.3 Perform Univariate analysis - Plot PDF, CDF, Boxplot.

- Plot the required charts to understand which feature are important for classification.
- Make sure that you add titles, legends and labels for each and every plots.
- Do write observations/inference for each plot.

```
import seaborn as sns
sns.FacetGrid(haberman,hue="survival_status",size=6).map(sns.distplot,"age").add_legend()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
warnings.warn(msg, FutureWarning)
```



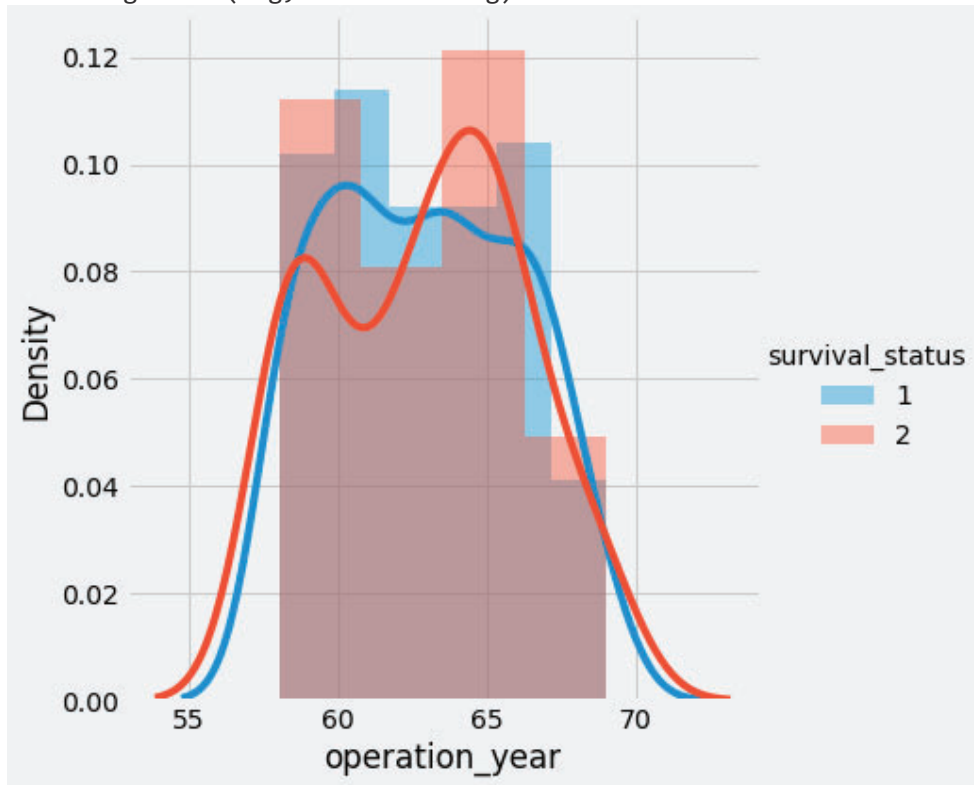
For age range from 30–80 the status of survival and death is same. So, using this datapoint we cannot make any sort of inferences.

```
sns.FacetGrid(haberman,hue="survival_status",size=6).map(sns.distplot,"operation_year").ad
plt.show()
```

```

/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `si:
warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
warnings.warn(msg, FutureWarning)

```



Similar here we cannot predict anything with these histograms as there is equal number of density in each data point. Even the PDF of both classification overlap on each other.

```

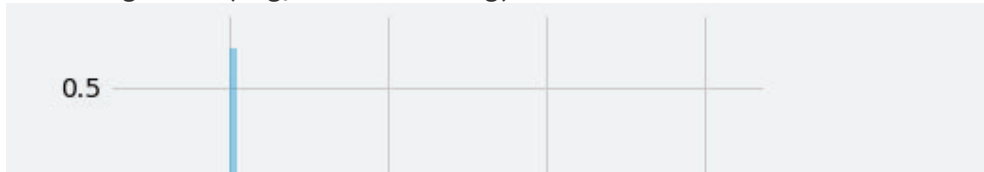
sns.FacetGrid(haberman, hue="survival_status", size=6).map(sns.distplot, "axil_nodes").add_le
plt.show()

```

```

/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `si:
warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
warnings.warn(msg, FutureWarning)

```



It has been observed that people survive long if they have less axillary nodes detected and vice versa but still it is hard to classify but this is the best data you can choose among all. So, I accept the PDF of Axillary nodes and can conclude below result

```

if(AxillaryNodes≤0)

Patient= Long survival

else if(AxillaryNodes≥0 && Axillary nodes≤3.5(approx))

Patient= Long survival chances are high

else if(Axillary nodes ≥3.5)

Patient = Short survival

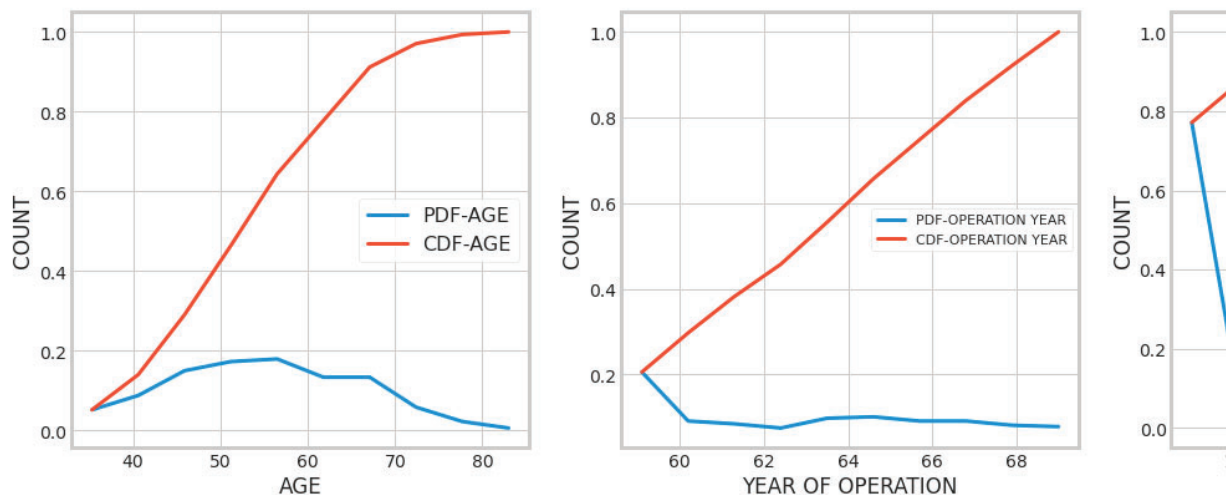

##haberman
plt.figure(figsize=(20,6))
plt.subplot(131) ##(1=no. of rows, 3= no. of columns, 1=1st figure,2,3,4 boxes)
counts,bin_edges=np.histogram(haberman["age"],bins=10,density=True)
pdf=counts/sum(counts)
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf,linewidth=3.0)
plt.plot(bin_edges[1:],cdf,linewidth=3.0)
plt.ylabel("COUNT")
plt.xlabel('AGE')
plt.title('')
plt.legend(['PDF-AGE', 'CDF-AGE'], loc = 5,prop={'size': 16})

plt.subplot(132)
counts,bin_edges=np.histogram(haberman["operation_year"],bins=10,density=True)
pdf=counts/sum(counts)
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf,linewidth=3.0)
plt.plot(bin_edges[1:],cdf,linewidth=3.0)
plt.ylabel("COUNT")
plt.xlabel('YEAR OF OPERATION')
plt.title('')

```

```
plt.legend(['PDF-OPERATION YEAR', 'CDF-OPERATION YEAR'], loc = 5,prop={'size': 11})

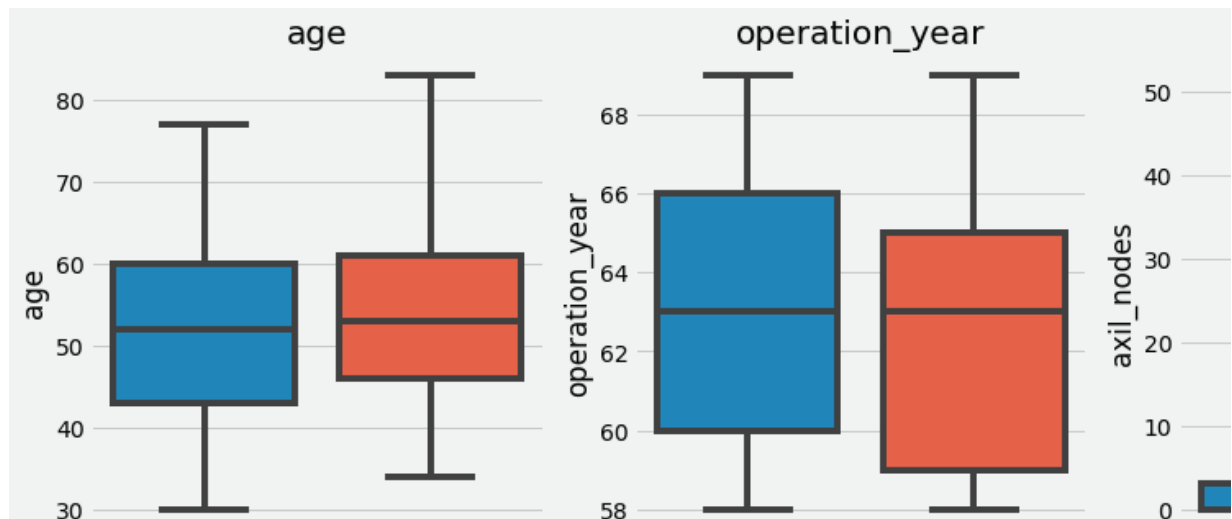
plt.subplot(133)
counts,bin_edges=np.histogram(haberman["axil_nodes"],bins=10,density=True)
pdf=counts/sum(counts)
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf,linewidth=3.0)
plt.plot(bin_edges[1:],cdf,linewidth=3.0)
plt.ylabel("COUNT")
plt.xlabel('AXIL NODES')
plt.title(' ')
plt.legend(['PDF-AXIL NODES', 'CDF-AXIL NODES'], loc = 5,prop={'size': 16})
plt.show()
```



From above graphs there is a 85% chance of long survival if number of axillary nodes detected are < 5. Also you can see as number of axillary nodes increases survival chances also reduces means it is clearly observed that 80% – 85% of people have good chances of survival if they have less no of auxillary nodes detected

Long survival observation is same but in Short survival nearly 55% of people who have nodes less than 5 and there are nearly 100% of people in short survival if nodes are > 40

```
figure, axes = plt.subplots(1, 3, figsize=(15, 5))
for idx, feature in enumerate(list(haberman.columns)[: -1]):
    mystr="" + feature
    sns.boxplot( x='survival_status', y=feature, data=haberman, ax=axes[idx]).set_title(my
plt.show()
```



Inference : In above box whiskers 25th percentile and 50th percentile are nearly same for Long survive(status=1) and threshold for it is 0 to 7. Also, for short survival(status=2) there are 50th percentile of nodes are nearly same as long survive(status=1) 75th percentile.

So,if nodes between 0–7 have chances of error as short survival(status=2) plot is also lies in it. That is 50% error for Short survival status(status=2).

There are most of point above 10 lies in Short survival (status=2).

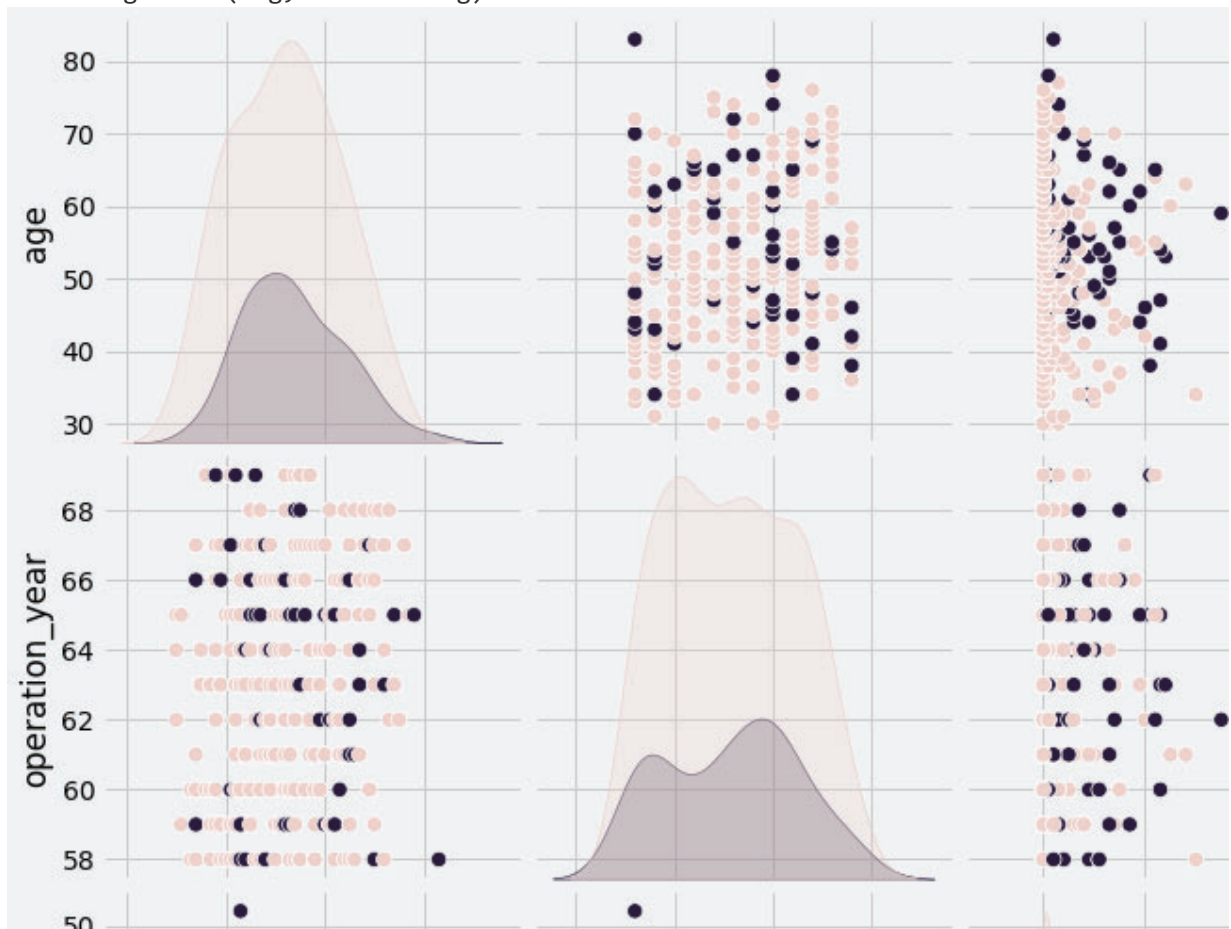
▼ 1.4 Perform Bivariate analysis - Plot 2D Scatter plots and Pair plots

- Plot the required Scatter plots and Pair plots of different features to see which combination of features are useful for clasification task
- Make sure that you add titles, legends and labels for each and every plots.
- Do write observations/inference for each plot.

```
plt.close() ## close previous show()
```

```
sns.pairplot(haberman,hue="survival_status",vars=["age","operation_year","axil_nodes"],size=10,plt.show())
```

/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:2076: UserWarning: The `s`:
warnings.warn(msg, UserWarning)

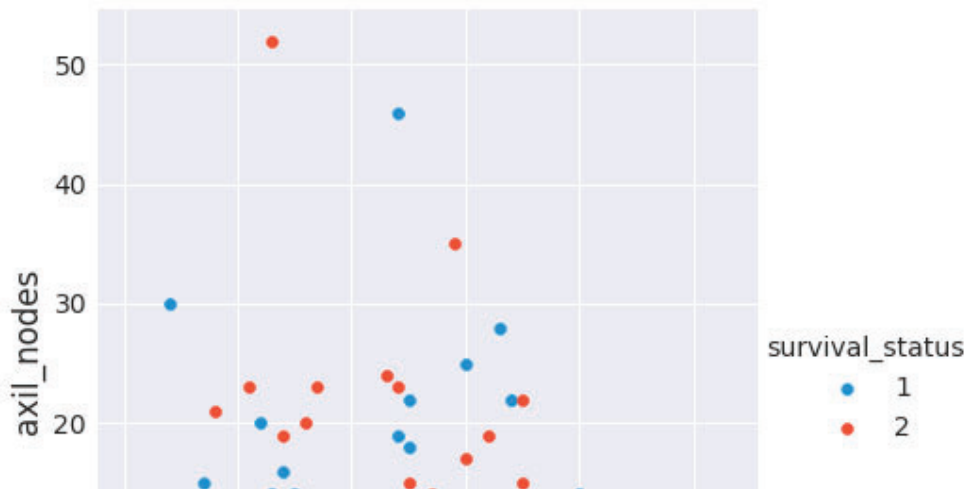


Inference Graphs 1,5,9 represent the probability density of age, year, nodes respectively. The 4th scatter plot represents a plot between year and age. It is difficult to make inference due to a lot of overlapping. The 7th scatter plot represents a plot between nodes and age. It is difficult to make any inference but the pink(status=1) dots are clustered towards 0 nodes. The upper half of scatter plot that is graph 2,3,6 are nothing but 90 deg rotated version of the bottom 3 graphs[4,7,8].



```
#age,axil_nodes better view
sns.set_style("darkgrid")
sns.FacetGrid(haberman, hue="survival_status", size=6) \
    .map(plt.scatter, "age", "axil_nodes") \
    .add_legend()
plt.show()
```

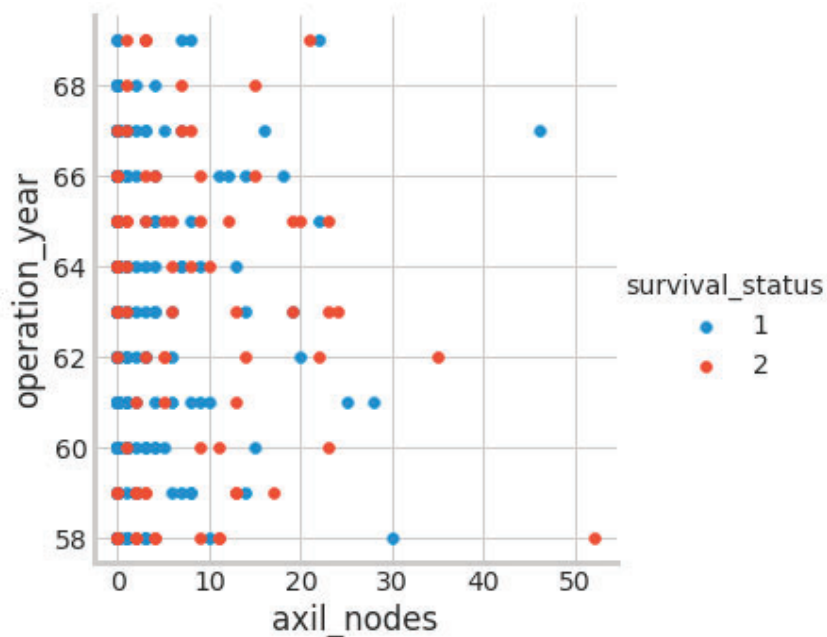
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `si:
warnings.warn(msg, UserWarning)



```
## AXIL NODES <> OPERATION YEAR better view
sns.set_style("whitegrid");
sns.FacetGrid(haberman, hue="survival_status", size=5) \
    .map(plt.scatter, "axil_nodes", "operation_year") \
    .add_legend();

plt.show();
```

/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `si:
warnings.warn(msg, UserWarning)



▼ 1.5 Summarize your final conclusions of the Exploration

- You can describe the key features that are important for the Classification task.
- Try to quantify your results i.e. while writing observations include numbers, percentages, fractions etc.
- Write a brief of your exploratory analysis in 3-5 points

When data is imbalanced and data(independent variables) are not greatly contributing to deciding class attribute better data analysis and visualization techniques are required.

Haberman Cancer Survival Dataset is imbalanced and independent attributes relevance to the class attribute is not great when analyzed using the above techniques. Univariate analysis(CDF) provides information about the distribution of attributes across the range for each class in the class attribute, irrespective of how much the dataset is balanced/imbalanced.

[Colab paid products](#) - [Cancel contracts here](#)



▼ Assignment 3

Problem Statement : Implement of K-Nearest Neighbors Algorithm

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import scipy.spatial
from collections import Counter
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
%matplotlib inline
```

```
!wget https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0
```



```
--2022-11-15 17:31:06-- https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.108.133
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.108.133|:443...
HTTP request sent, awaiting response... 200 OK
Length: 3975 (3.9K) [text/plain]
Saving to: 'iris.csv'
```

```
iris.csv          100%[=====>]    3.88K  --.-KB/s    in 0s
```

```
2022-11-15 17:31:06 (54.7 MB/s) - 'iris.csv' saved [3975/3975]
```



```
df = pd.read_csv("iris.csv")
df.head()
```

| | sepal.length | sepal.width | petal.length | petal.width | variety |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Setosa |

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

```

scaler = StandardScaler()
scaler.fit(x)
scaled_features = scaler.transform(x)

```

```

df_feat = pd.DataFrame(scaled_features, columns=df.columns[:-1])
df_feat.head()

```

| | sepal.length | sepal.width | petal.length | petal.width |
|---|--------------|-------------|--------------|-------------|
| 0 | -0.900681 | 1.019004 | -1.340227 | -1.315444 |
| 1 | -1.143017 | -0.131979 | -1.340227 | -1.315444 |
| 2 | -1.385353 | 0.328414 | -1.397064 | -1.315444 |
| 3 | -1.506521 | 0.098217 | -1.283389 | -1.315444 |
| 4 | -1.021849 | 1.249201 | -1.340227 | -1.315444 |

```

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

```

```

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
    test_size=0.20, random_state= 42)

```

```

class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def distance(self, X1, X2):
        distance = scipy.spatial.distance.euclidean(X1, X2)

    def predict(self, X_test):
        final_output = []
        for i in range(len(X_test)):
            d = []
            votes = []
            for j in range(len(X_train)):
                dist = scipy.spatial.distance.euclidean(X_train[j], X_test[i])
                d.append([dist, j])
            d.sort()
            d = d[0:self.k]
            for d, j in d:
                votes.append(y_train[j])
            # print(votes)
            # print(Counter(votes).most_common(1))
            ans = Counter(votes).most_common(1)[0][0]
            final_output.append(ans)

```

```
        return final_output

    def accuracy(self, predictions, y_test):
        accuracy = ((predictions == y_test).sum() / len(y_test)) * 100
        return round(accuracy, 2)

num_neighbours = [1,3,5,7,9,11,13,15,17,19,21,23,25]
training_accuracy = []
training_error_rate = []
validation_accuracy = []
validation_error_rate = []

for i in num_neighbours:
    my_knn = KNN(i)
    my_knn.fit(X_train, y_train)

    # For Training data
    prediction = my_knn.predict(X_train)
    training_accuracy.append(my_knn.accuracy(prediction, y_train))
    training_error_rate.append(np.mean(prediction != y_train))

    # For Validation data
    prediction = my_knn.predict(X_val)
    validation_accuracy.append(my_knn.accuracy(prediction, y_val))
    validation_error_rate.append(np.mean(prediction != y_val))

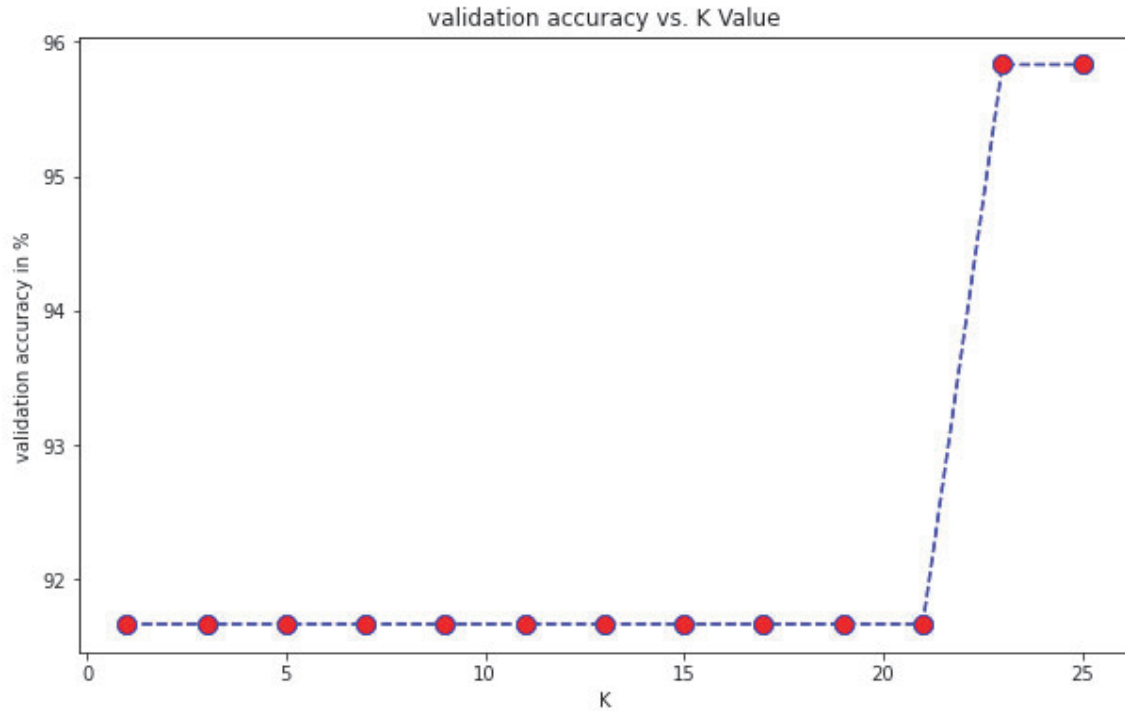
plt.figure(figsize=(10,6))
plt.plot(num_neighbours, training_accuracy, 'b--', marker='o', markerfacecolor='red', mark
plt.title('training accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('training accuracy in %')
```

```
Text(0, 0.5, 'training accuracy in %')
```

training accuracy vs. K Value

```
plt.figure(figsize=(10,6))
plt.plot(num_neighbours, validation_accuracy, 'b--', marker='o', markerfacecolor='red', ma
plt.title('validation accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('validation accuracy in %')
```

```
Text(0, 0.5, 'validation accuracy in %')
```



```
# CV error
plt.figure(figsize=(10,6))
plt.plot(num_neighbours, training_error_rate, 'b--', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Training Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Training Error Rate')
```

Text(0, 0.5, 'Training Error Rate')

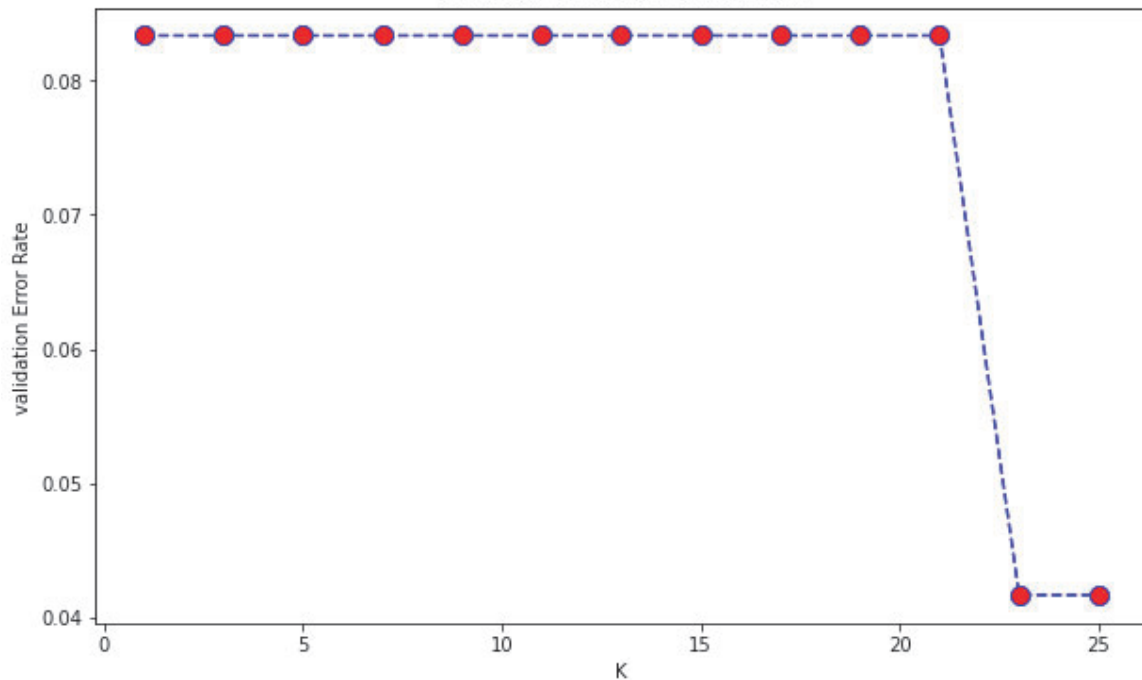
Training Error Rate vs. K Value



```
plt.figure(figsize=(10,6))
plt.plot(num_neighbours,validation_error_rate,'b--', marker='o',
        markerfacecolor='red', markersize=10)
plt.title('validation Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('validation Error Rate')
```

Text(0, 0.5, 'validation Error Rate')

validation Error Rate vs. K Value

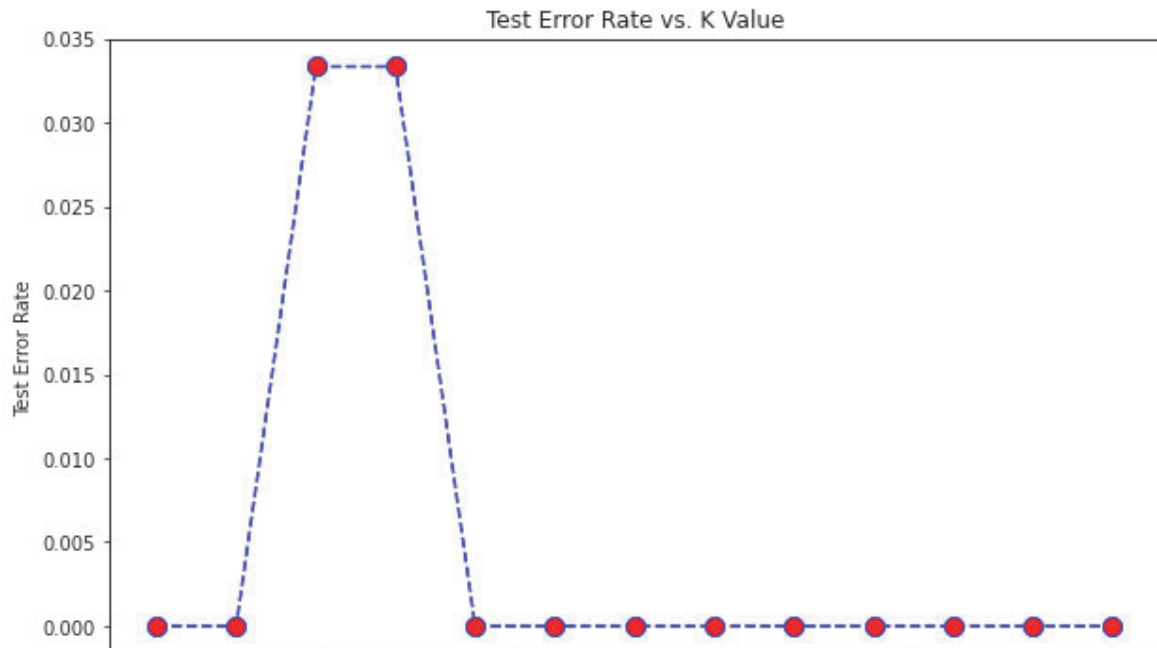


```
test_error_rate = []
```

```
for i in num_neighbours:
    my_knn = KNN(i)
    my_knn.fit(X_train, y_train)
    prediction = my_knn.predict(X_test)
    test_error_rate.append(np.mean(prediction != y_test))
```

```
plt.figure(figsize=(10,6))
plt.plot(num_neighbours,test_error_rate,color='blue', linestyle='dashed', marker='o',
        markerfacecolor='red', markersize=10)
plt.title('Test Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Test Error Rate')
```

Text(0, 0.5, 'Test Error Rate')



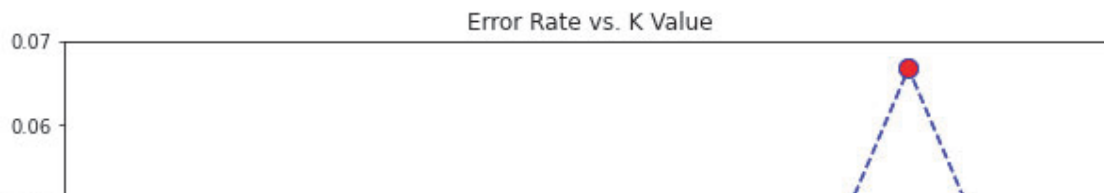
```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(x, y, test_size=0.30, random_state=101)
```

```
neighbours_list = [1,3,5,7,9,11,13,15,17,19,21,23,25]  
error_rate = []
```

```
for i in neighbours_list:  
    knn = KNeighborsClassifier(n_neighbors=i)  
    knn.fit(X_Train, Y_Train)  
    pred_i = knn.predict(X_Test)  
    error_rate.append(np.mean(pred_i != Y_Test))
```

```
plt.figure(figsize=(10,6))  
plt.plot(neighbours_list,error_rate,color='blue', linestyle='dashed', marker='o',  
        markerfacecolor='red', markersize=10)  
plt.title('Error Rate vs. K Value')  
plt.xlabel('K')  
plt.ylabel('Error Rate')
```

Text(0, 0.5, 'Error Rate')



NOW WITH K=3

```
knn = KNeighborsClassifier(n_neighbors=3)
```

```
knn.fit(X_Train, Y_Train)
```

```
pred = knn.predict(X_Test)
```

```
print('Confusion matrix with K=3\n')
```

```
print(confusion_matrix(Y_Test,pred))
```

```
print('\nclassification report with K=3\n')
```

```
print(classification_report(Y_Test,pred))
```

```
print("Accuracy with k=3 is", accuracy_score(Y_Test,pred)*100)
```

Confusion matrix with K=3

```
[[13  0  0]
 [ 0 20  0]
 [ 0  0 12]]
```

classification report with K=3

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Setosa | 1.00 | 1.00 | 1.00 | 13 |
| Versicolor | 1.00 | 1.00 | 1.00 | 20 |
| Virginica | 1.00 | 1.00 | 1.00 | 12 |
| accuracy | | | 1.00 | 45 |
| macro avg | 1.00 | 1.00 | 1.00 | 45 |
| weighted avg | 1.00 | 1.00 | 1.00 | 45 |

Accuracy with k=3 is 100.0

[Colab paid products](#) - [Cancel contracts here](#)



▼ Assignment 4

Problem statement: Implement of Logistic Regression Algorithm

Import Libraries

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

Load the data

```
data = load_breast_cancer() #refer: http://scikit-learn.org/stable/modules/generated/sklearn

# data with features
X = data.data

# data class labels
y = data.target

print(X.shape,X[:1])

(569, 30) [[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]]

print(y.shape,y)

(569,) [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 0 1
 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 0]
```

```

0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 0 0 1 1
1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0
0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1
1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1
0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1
1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 0
1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 0 0 0 0 0 1]

```

Print the number of data points, number of features and number of classes in the given data set.

```

features = len(X[0])
print("Data point ",len(X))
print("features ",features)
print("number of class",len(set(y)))

```

```

Data point 569
features 30
number of class 2

```

Splitting data into Train and test sets with Stratified Sampling using train_test_split()

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
print(len(X_train),len(X_test),len(y_train),len(y_test))

```

```

455 114 455 114

```

Data Preprocessing using column standardisation. Use sklearn.preprocessing.StandardScaler().

```

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled =scaler.transform(X_test)

```

```

print(X_train_scaled[:1])

```

```

[[-0.14529368 -0.32853678 -0.22822099 -0.22899756 -1.18158813 -1.19151412
 -0.85849002 -0.93659608 -0.87266881 -1.07019398 -0.51237252 -0.08604863
 -0.57140924 -0.38025021 -1.06533229 -1.12081446 -0.71361006 -1.08589559
 -0.86874749 -1.01324538 -0.15068304  0.06764607 -0.24214338 -0.24180676
 -1.17486169 -1.1288592  -0.79887857 -0.94506497 -0.93038565 -1.17865791]]

```

Implement Logistic Regression Using Gradient Descent: without using sklearn.

In this algorithm, n is the total number of datapoints in dataset. α is the learning rate to be used in gradient descent. For this work, just fix $\alpha = 0.001$.

The predicted value for data point x is $y_{pred} = \sigma(w^T x + b)$, where σ is a sigmoid function.

ALGORITHM:

- Initialize the weight_vector and intercept term to zeros (Write your code in `def initialize_weights()`)
- Create a loss function (Write your code in `def logloss()`)

$$\text{logloss} = -1 * \frac{1}{n} \sum_{\text{foreach } y_{\text{true}}, y_{\text{pred}}} (y_{\text{true}} \log(y_{\text{pred}}) + (1 - y_{\text{true}}) \log(1 - y_{\text{pred}}))$$
- for each epoch:
 - for each data point say x_i in train:
 - calculate the gradient of loss function w.r.t each weight in weight vector (write your code in `def gradient_dw()`)

$$dw^{(t)} = \frac{1}{n} (x_i (\sigma((w^{(t)})^T x_i + b^t) - y_i))$$
 - Calculate the gradient of the intercept (write your code in `def gradient_db()`)

$$db^{(t)} = \frac{1}{n} (\sigma((w^{(t)})^T x_i + b^t) - y_i)$$
 - Update weights and intercept usign gradient descent

$$w^{(t+1)} \leftarrow w^{(t)} - \alpha(dw^{(t)})$$

$$b^{(t+1)} \leftarrow b^{(t)} - \alpha(db^{(t)})$$
 - predict the output for all test data points with updated weights. (write your function in `def prediction()`)
 - calculate the log loss for train and test data points separately with the updated weights. Store these losses in the lists, `train_loss` and `test_loss`.
 - And if you wish, you can compare the previous train loss and the current train loss, if it is not updating, then you can stop the training
 - return the updated weights, training and test loss lists.

```
n_features = len(X[0])
```

```
def initialize_weights():
    weights = np.zeros((n_features, ))
    bias = 0
    return weights, bias
```

```
def sigmoid(z):
```

```
return 1 / (1 + np.exp(-z))
```

```
def logloss(y_true, y_pred):
```

```
    n = y_true.shape[0]
```

```
    log_sum = 0
```

```
    for i in range(n):
```

```
        log_sum += (y_true[i] * np.log(y_pred[i])) + ((1 - y_true[i]) * np.log(1 - y_pred[i]))
```

```
    return -log_sum / n
```

```
    # you have been given two arrays y_true and y_pred and you have to calculate the loglo
```

w should be a vector of size as input data point. Size of w and dw be same.

```
def gradient_dw(x, y, w, b):
```

```
    dw = np.dot(x, sigmoid(np.dot(w.T, x) + b) - y)
```

```
    return dw
```

#b should be a scalar value

```
def gradient_db(x,y,w,b):
```

```
    db = sigmoid(np.dot(w.T, x) + b) - y
```

```
    return db
```

For the prediction, if activation_value > 0.5 then assign label = 1 else label = 0

```
def predict(x, w, b):
```

```
    predictions = []
```

```
    for point in x:
```

```
        predictions.append(0 if sigmoid(np.dot(w.T, point) + b) < 0.5 else 1)
```

```
    return predictions
```

```
def logistic_regression(epochs=1000, alpha=.001):
```

```
    weights, bias = initialize_weights()
```

```
    n = len(X_train_scaled[0])
```

```
    train_loss = []
```

```
    for epoch in range(epochs):
```

```
        for x, y in zip(X_train_scaled, y_train):
```

```
            weights -= (alpha * gradient_dw(x, y, weights, bias))
```

```
            bias -= (alpha * gradient_db(x, y, weights, bias))
```

```
            loss = logloss(y_train, sigmoid(np.dot(X_train_scaled, weights) + bias))
```

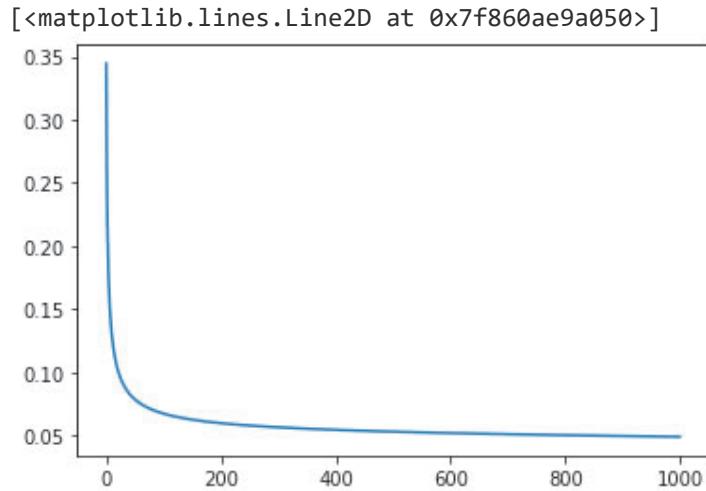
```
            train_loss.append(loss)
```

```
    return weights, bias, train_loss
```

Plot your train and test loss vs epochs. Plot epoch number on X-axis and loss on Y-axis and make sure that the curve is converging

```
weights, bias, train_loss = logistic_regression()
```

```
plt.plot(list(range(1000)), train_loss)
```



BONUS: Train your model with varying values of learning rates say ranging in $[0.1, 0.01, 0.001, 0.0001]$ and plot the performances.

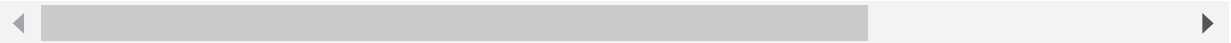
```
y_pred = predict(X_test_scaled, weights, bias)
accuracy = accuracy_score(y_test, y_pred)
```

```
alpha_list = [0.1, 0.01, 0.001, 0.0001]
acc_list = []
```

```
for alpha in alpha_list:
    weights, bias, train_loss = logistic_regression(alpha=alpha)
    y_pred = predict(X_test_scaled, weights, bias)
    accuracy = accuracy_score(y_test, y_pred)
    acc_list.append(accuracy)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: RuntimeWarning: divi
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: RuntimeWarning: inva
```



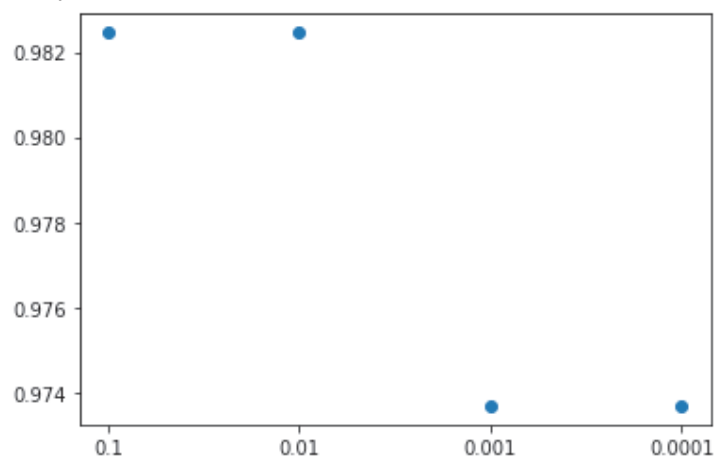
```
print(list((alpha_list, acc_list)))
```

```
[[0.1, 0.01, 0.001, 0.0001], [0.9824561403508771, 0.9824561403508771, 0.973684210526
```



```
plt.scatter([str(x) for x in alpha_list], acc_list)
```

<matplotlib.collections.PathCollection at 0x7f860ae79fd0>



[Colab paid products](#) - [Cancel contracts here](#)



▼ Assignment - 5

Problem Statement - Perform handwritten digit classification using Logistic Regression.

```
import numpy as np
import pandas as pd
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
import matplotlib.pyplot as plt
```

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
df = pd.read_csv("sample_data/mnist_train_small.csv")
```

```
df.head()
```

| | 6 | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | ... | 0.581 | 0.582 | 0.583 | 0.584 | 0.5 |
|---|---|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-------|-------|-------|-------|-----|
| 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | |
| 2 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | |
| 3 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | |
| 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | |

5 rows × 785 columns

```
df.isnull().any()
```

```
6      False
0      False
0.1    False
0.2    False
0.3    False
```



```

...
0.586    False
0.587    False
0.588    False
0.589    False
0.590    False
Length: 785, dtype: bool

```

```
df.describe()
```

| | 6 | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |
|--------------|--------------|---------|---------|---------|---------|---------|---------|---------|---------|
| count | 19999.000000 | 19999.0 | 19999.0 | 19999.0 | 19999.0 | 19999.0 | 19999.0 | 19999.0 | 19999.0 |
| mean | 4.470124 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| std | 2.892807 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| min | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 25% | 2.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 50% | 4.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 75% | 7.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| max | 9.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

8 rows × 785 columns

```

X = df.drop('6', axis=1)
y = df['6']

```

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

```
scaler = StandardScaler()
```

```

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

```

```

import warnings
warnings.filterwarnings("ignore")

```

```
parameters = {'C': [.001, .01, .1, 1, 10, 100]}
```

```

grid = GridSearchCV(estimator=LogisticRegression(), param_grid=parameters, cv=4, verbose=2)
grid.fit(X_train_scaled, y_train)

```

```

Fitting 4 folds for each of 6 candidates, totalling 24 fits
[CV] END .....C=0.001; total time= 8.0s
[CV] END .....C=0.001; total time= 13.4s
[CV] END .....C=0.001; total time= 9.4s
[CV] END .....C=0.001; total time= 5.7s
[CV] END .....C=0.01; total time= 7.2s
[CV] END .....C=0.01; total time= 7.2s

```

```

[CV] END .....C=0.01; total time= 7.3s
[CV] END .....C=0.01; total time= 12.5s
[CV] END .....C=0.1; total time= 8.6s
[CV] END .....C=0.1; total time= 7.2s
[CV] END .....C=0.1; total time= 7.0s
[CV] END .....C=0.1; total time= 7.2s
[CV] END .....C=1; total time= 7.1s
[CV] END .....C=1; total time= 7.4s
[CV] END .....C=1; total time= 8.6s
[CV] END .....C=1; total time= 7.9s
[CV] END .....C=10; total time= 7.1s
[CV] END .....C=10; total time= 7.1s
[CV] END .....C=10; total time= 7.2s
[CV] END .....C=10; total time= 7.7s
[CV] END .....C=100; total time= 7.0s
[CV] END .....C=100; total time= 7.0s
[CV] END .....C=100; total time= 7.0s
[CV] END .....C=100; total time= 7.4s
GridSearchCV(cv=4, estimator=LogisticRegression(),
              param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100]},
              return_train_score=True, verbose=2)

```

grid.cv_results_

```

{'mean_fit_time': array([9.10951942, 8.54638851, 7.4803046 , 7.72054148, 7.25428867,
                          7.06924778]),
 'std_fit_time': array([2.81814117, 2.28359769, 0.62369202, 0.56874718, 0.22690702,
                          0.1547289 ]),
 'mean_score_time': array([0.01549727, 0.01658201, 0.01196879, 0.01458263,
                           0.01414961,
                           0.01395339]),
 'std_score_time': array([0.00485836, 0.00874816, 0.0004257 , 0.00373894,
                           0.00461697,
                           0.00350481]),
 'param_C': masked_array(data=[0.001, 0.01, 0.1, 1, 10, 100],
                          mask=[False, False, False, False, False, False],
                          fill_value='?',
                          dtype=object),
 'params': [{'C': 0.001},
             {'C': 0.01},
             {'C': 0.1},
             {'C': 1},
             {'C': 10},
             {'C': 100}],
 'split0_test_score': array([0.90625, 0.91625, 0.903 , 0.88175, 0.8685 , 0.863 ]),
 'split1_test_score': array([0.90275, 0.91575, 0.8995 , 0.879 , 0.87225, 0.869 ]),
 'split2_test_score': array([0.90125, 0.9155 , 0.90625, 0.88225, 0.872 , 0.86725]),
 'split3_test_score': array([0.89622406, 0.90822706, 0.90247562, 0.87821955,
                              0.8692173 ,
                              0.86246562]),
 'mean_test_score': array([0.90161851, 0.91393176, 0.9028064 , 0.88030489,
                           0.87049183,
                           0.8654289 ]),
 'std_test_score': array([0.0036044 , 0.00330467, 0.00239466, 0.0017265 ,
                           0.00165511,
                           0.00277262]),
 'rank_test_score': array([3, 1, 2, 4, 5, 6], dtype=int32),
 'split0_train_score': array([0.91490958, 0.94507876, 0.97233103, 0.9935828 ,
                              0.9995833 ,

```

```

        1.    ]),
    'split1_train_score': array([0.915993 , 0.94649554, 0.97233103, 0.99324944,
0.99974998,
        1.    ]),
    'split2_train_score': array([0.91574298, 0.94457871, 0.97024752, 0.99074923,
0.99899992,
        1.    ]),
    'split3_train_score': array([0.91808333, 0.94733333, 0.97191667, 0.99391667,
0.99983333,
        1.    ]),
    'mean_train_score': array([0.91618222, 0.94587159, 0.97170656, 0.99287453,
0.99954163,
        1.    ]),
    'std_train_score': array([0.00116861, 0.00109842, 0.00085919, 0.00124952,
0.00032546,
        0.    ])}

```

```
C_vals = [str(x) for x in parameters['C']]
```

```

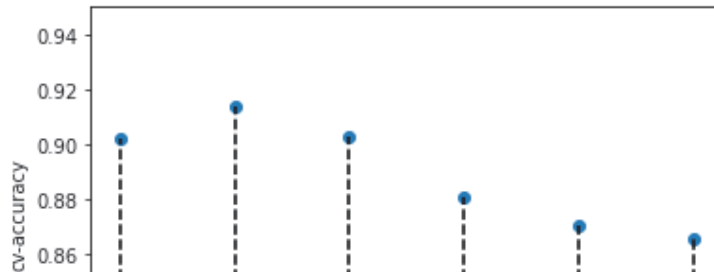
plt.xlabel("C")
plt.ylabel("cv-accuracy")
plt.scatter(C_vals, grid.cv_results_['mean_test_score'])
plt.vlines(C_vals, 0, grid.cv_results_['mean_test_score'], linestyle="dashed")
plt.ylim(0.80,.95)
plt.xticks(C_vals)
plt.show()

```

```

plt.xlabel("C")
plt.ylabel("train-accuracy")
plt.scatter(C_vals, grid.cv_results_['mean_train_score'])
plt.vlines(C_vals, 0, grid.cv_results_['mean_train_score'], linestyle="dashed")
plt.ylim(0.85,1.00)
plt.xticks(C_vals)
plt.show()

```



```
print("Best parameters:", grid.best_params_)
print("Best score:", grid.best_score_)
```

```
Best parameters: {'C': 0.01}
Best score: 0.9139317641910476
```

```
---
```

```
model = LogisticRegression(C=.01)
```

```
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
```

```
accuracy_score(y_test, y_pred)
```

```
0.91675
```

```
0.88 | i i i i i |
```

```
for c in parameters['C']:
```

```
    model = LogisticRegression(C=c)
```

```
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
```

```
    print(c, accuracy_score(y_test, y_pred))
```

```
0.001 0.90775
```

```
0.01 0.91675
```

```
0.1 0.90725
```

```
1 0.89425
```

```
10 0.8845
```

```
100 0.88175
```

[Colab paid products](#) - [Cancel contracts here](#)



▼ Assignment 6

Problem Statement : Perform handwritten digit classification using SVM model, Neural network model and Logistic Regression model

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler

from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay, classification_report

data = load_digits()

# data with features
X = data.data

# data class labels
y = data.target

plt.gray()
plt.matshow(data.images[0])
```

```
<matplotlib.image.AxesImage at 0x7fab7af85290>
<Figure size 432x288 with 0 Axes>
```

X.shape

```
(1797, 64)
```



```
labels, label_count = np.unique(y, return_counts=True)
print(*zip(labels, label_count))
```

```
(0, 178) (1, 182) (2, 177) (3, 183) (4, 181) (5, 182) (6, 181) (7, 179) (8, 174) (9,
```



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



```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

▼ SVM

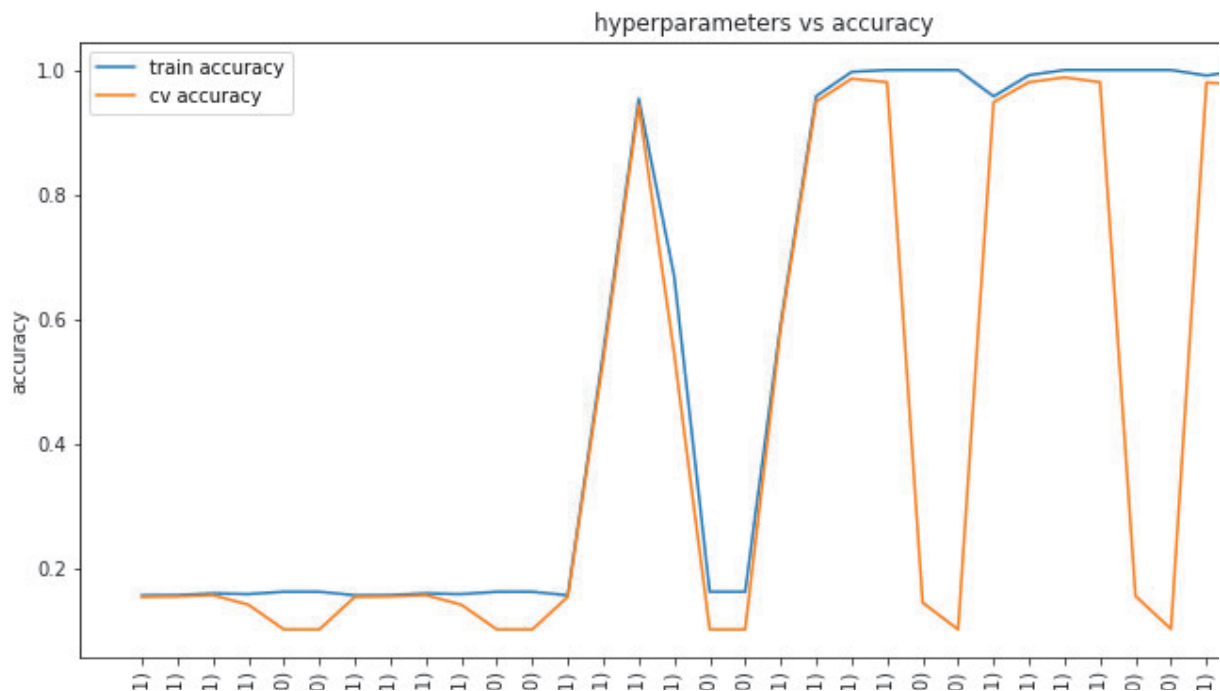
```
svm = SVC()
parameters = {'C': [.001, .01, .1, 1, 10, 100], 'gamma': [.001, .01, .1, 1, 10, 100]}
```

```
svm_grid = GridSearchCV(estimator=svm, param_grid=parameters, n_jobs=-1, cv=5, return_train_score=True)
svm_grid.fit(X_train_scaled, y_train)
```

```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=-1,
              param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                          'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},
              return_train_score=True)
```

```
params_list = [str((x['C'], x['gamma'])) for x in svm_grid.cv_results_['params']]
# params_list
```

```
fig = plt.figure(figsize = (13,6))
plt.title("hyperparameters vs accuracy")
plt.plot(params_list, svm_grid.cv_results_['mean_train_score'], label="train accuracy")
plt.plot(params_list, svm_grid.cv_results_['mean_test_score'], label="cv accuracy")
plt.xlabel("(C, gamma) pair")
plt.ylabel("accuracy")
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



```
print(f"Best score: {svm_grid.best_score_}")
print(f"Best parameters: {svm_grid.best_params_}")
```

```
Best score: 0.9881678281068524
Best parameters: {'C': 10, 'gamma': 0.1}
```

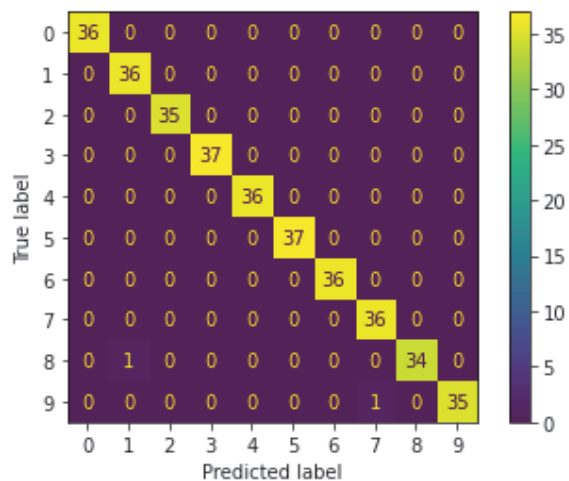
```
svm_pred = svm_grid.predict(X_test_scaled)
```

```
print(f"Accuracy: {accuracy_score(y_test, svm_pred):.4f}")
```

```
Accuracy: 0.9944
```

```
ConfusionMatrixDisplay.from_predictions(y_test, svm_pred, display_labels=svm_grid.classes_)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fab7ab207d0>
```



```
print(classification_report(y_test, svm_pred))
```


| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 1.00 | 1.00 | 1.00 | 36 |
| 1 | 0.97 | 1.00 | 0.99 | 36 |
| 2 | 1.00 | 1.00 | 1.00 | 35 |
| 3 | 1.00 | 1.00 | 1.00 | 37 |
| 4 | 1.00 | 1.00 | 1.00 | 36 |
| 5 | 1.00 | 1.00 | 1.00 | 37 |
| 6 | 1.00 | 1.00 | 1.00 | 36 |
| 7 | 0.97 | 1.00 | 0.99 | 36 |
| 8 | 1.00 | 0.97 | 0.99 | 35 |
| 9 | 1.00 | 0.97 | 0.99 | 36 |
| accuracy | | | 0.99 | 360 |
| macro avg | 0.99 | 0.99 | 0.99 | 360 |
| weighted avg | 0.99 | 0.99 | 0.99 | 360 |

► Neural Network

[] ↳ 8 cells hidden

► Logistic Regression

[] ↳ 8 cells hidden

► Observations

[] ↳ 2 cells hidden

[Colab paid products](#) - [Cancel contracts here](#)



ML LAB PROJECT: LUNG CANCER DETECTION MEMBERS:

MISHAL PATEL(19BCP081)

RUSHABH SHAH(19BCP108)

PRIYANK RANA(19BCP099)

JINIL SHUKLA(19BCP161)

PROBLEM STATEMENT: In this project, looking at the various factors which are being affected and also see the attributes that contribute in the classification of detection of lung cancer in a human body. The lung Cancer found in the body is dependent on many factors which includes 16 different attributes like age, smoking, yellow fingers, alchohol consumptions, and many more. The data will be trained after preprocessing and predictions will be made with different classification algorithms such as Logistic Regression, Random Forest and Support Vector Machine.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
import cufflinks as cf
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)
from tensorflow.keras.layers import BatchNormalization
from keras.backend import dropout
from keras.models import Model, load_model, Sequential
from tensorflow.keras.models import Model
from keras.layers import Input,Dense,Dropout
```



```
df = pd.read_csv("survey lung cancer.csv")
```

DATASET DESCRIPTION:

The Dataset contains 15 different attributes with over more than 250 test cases with the training datasets. The training datasets include multiple attributes like yellow fingers, age, gender, smoking, fatigue, allergy etc. which includes multiple cases for the dataset. The Dataset also contained some unbalanced results and to overcome the situation we will preprocess the data by the method of smote.

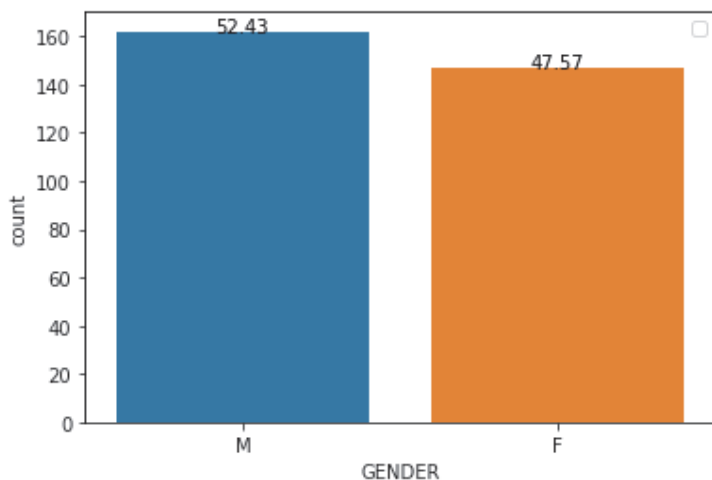
```
df.head()
```

| | GENDER | AGE | SMOKING | YELLOW_FINGERS | ANXIETY | PEER_PRESSURE | CHRONIC DISEASE | FATIGUE | AI |
|---|--------|-----|---------|----------------|---------|---------------|-----------------|---------|----|
| 0 | M | 69 | 1 | 2 | 2 | 1 | 1 | 2 | |
| 1 | M | 74 | 2 | 1 | 1 | 1 | 2 | 2 | |
| 2 | F | 59 | 1 | 1 | 1 | 2 | 1 | 2 | |
| 3 | M | 63 | 2 | 2 | 2 | 1 | 1 | 1 | |
| 4 | F | 63 | 1 | 2 | 1 | 1 | 1 | 1 | |

```
total = float(df.shape[0])
plotting = sns.countplot(x='GENDER', data=df)
for p in plotting.patches:
    height = p.get_height()
    plotting.text(p.get_x() + p.get_width()/2.,
                  height,
                  '{:.2f}'.format((height/total)*100),
                  ha='center')
```

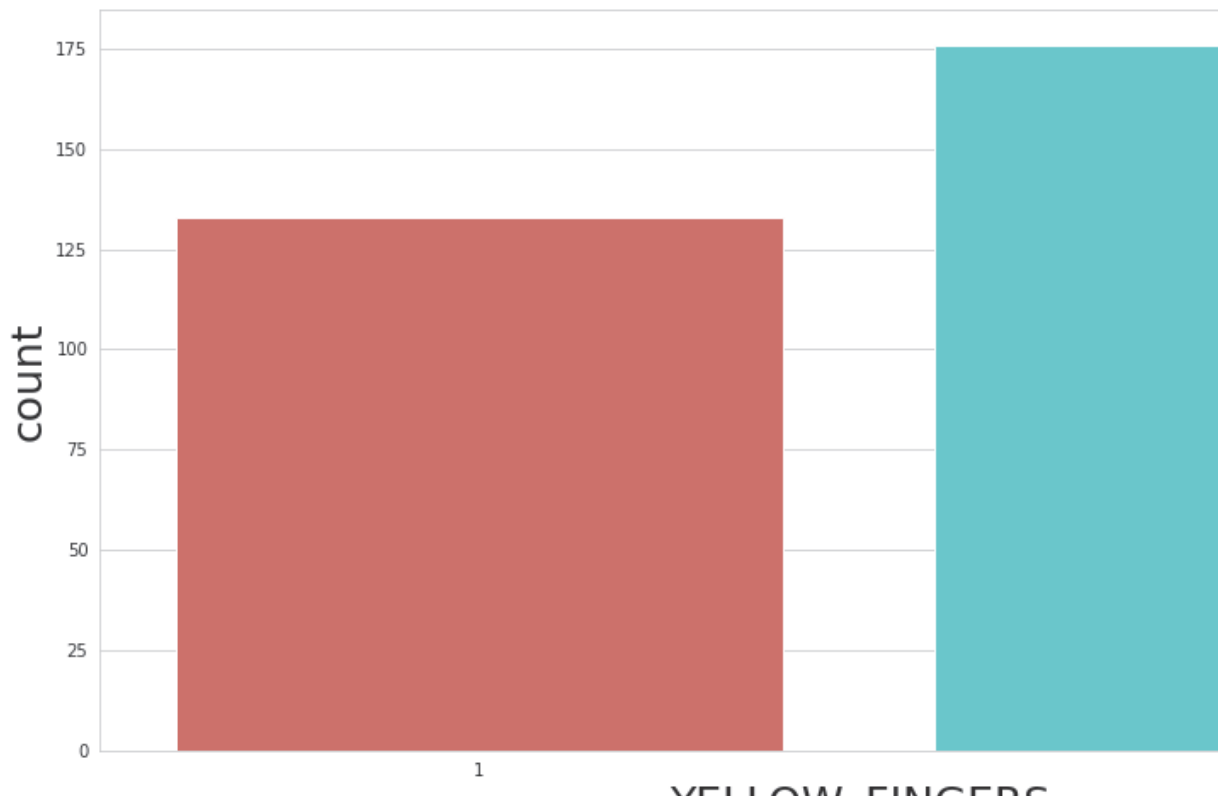
```
plt.legend()
plt.show()
```

WARNING:matplotlib.legend:No handles with labels found to put in legend.



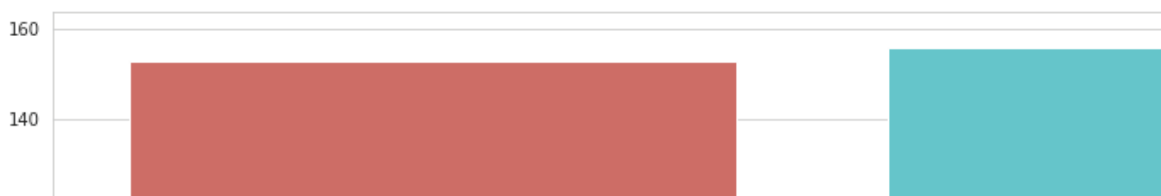
```
plt.figure(figsize=(16,8))
sns.set_style("whitegrid")
plt.title('Grouping People by YELLOW FINGERS ', fontsize=30, fontweight='bold', y=1.05,)
plt.xlabel('YELLOW_FINGERS', fontsize=25)
plt.ylabel('Count', fontsize=25)
sns.countplot(x="YELLOW_FINGERS", data=df, palette="hls");
plt.show()
```

Grouping People by YELLOW FIN



```
plt.figure(figsize=(16,8))
sns.set_style("whitegrid")
plt.title('Grouping People by CHRONIC DISEASE ', fontsize=30, fontweight='bold', y=1.05,)
plt.xlabel('CHRONIC DISEASE', fontsize=25)
plt.ylabel('Count', fontsize=25)
sns.countplot(x="CHRONIC DISEASE", data=df, palette="hls");
plt.show()
```

Grouping People by CHRONIC DI



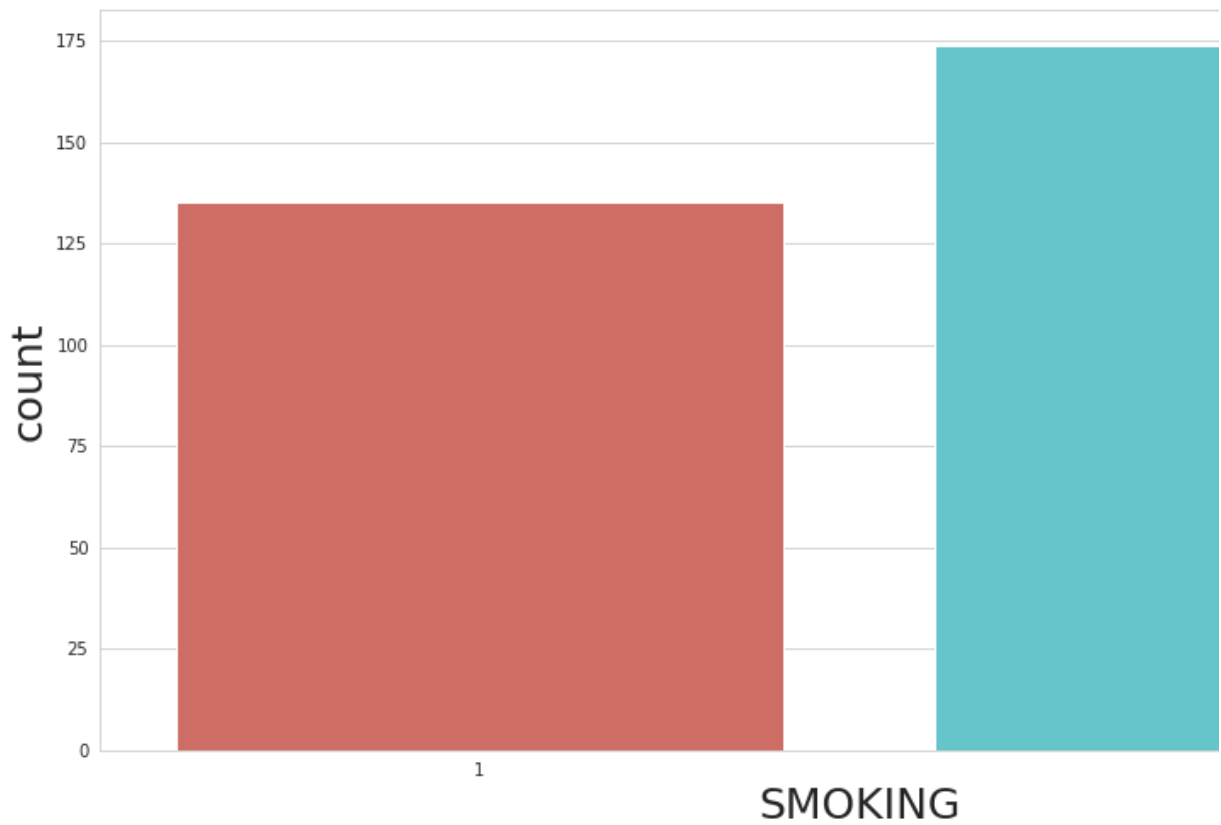
```
plt.figure(figsize=(16,8))
sns.set_style("whitegrid")
plt.title('Grouping People by ALCOHOL CONSUMING ', fontsize=30, fontweight='bold', y=1.05,
plt.xlabel('ALCOHOL CONSUMING',fontsize=25)
plt.ylabel('Count',fontsize=25)
sns.countplot(x="ALCOHOL CONSUMING", data=df, palette="hls");
plt.show()
```

Grouping People by ALCOHOL CON



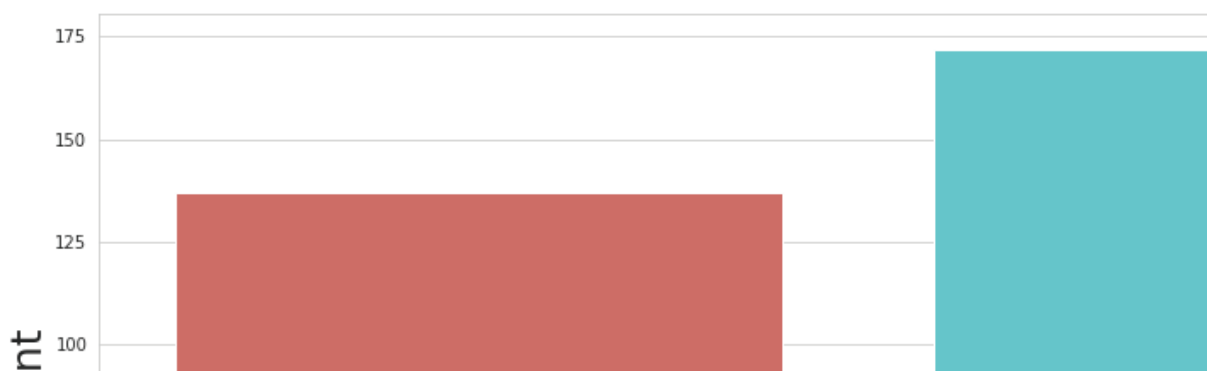
```
plt.figure(figsize=(16,8))
sns.set_style("whitegrid")
plt.title('Grouping People by SMOKING ', fontsize=30, fontweight='bold', y=1.05,)
plt.xlabel('SMOKING',fontsize=25)
plt.ylabel('Count',fontsize=25)
sns.countplot(x="SMOKING", data=df, palette="hls");
plt.show()
```

Grouping People by SMOKING



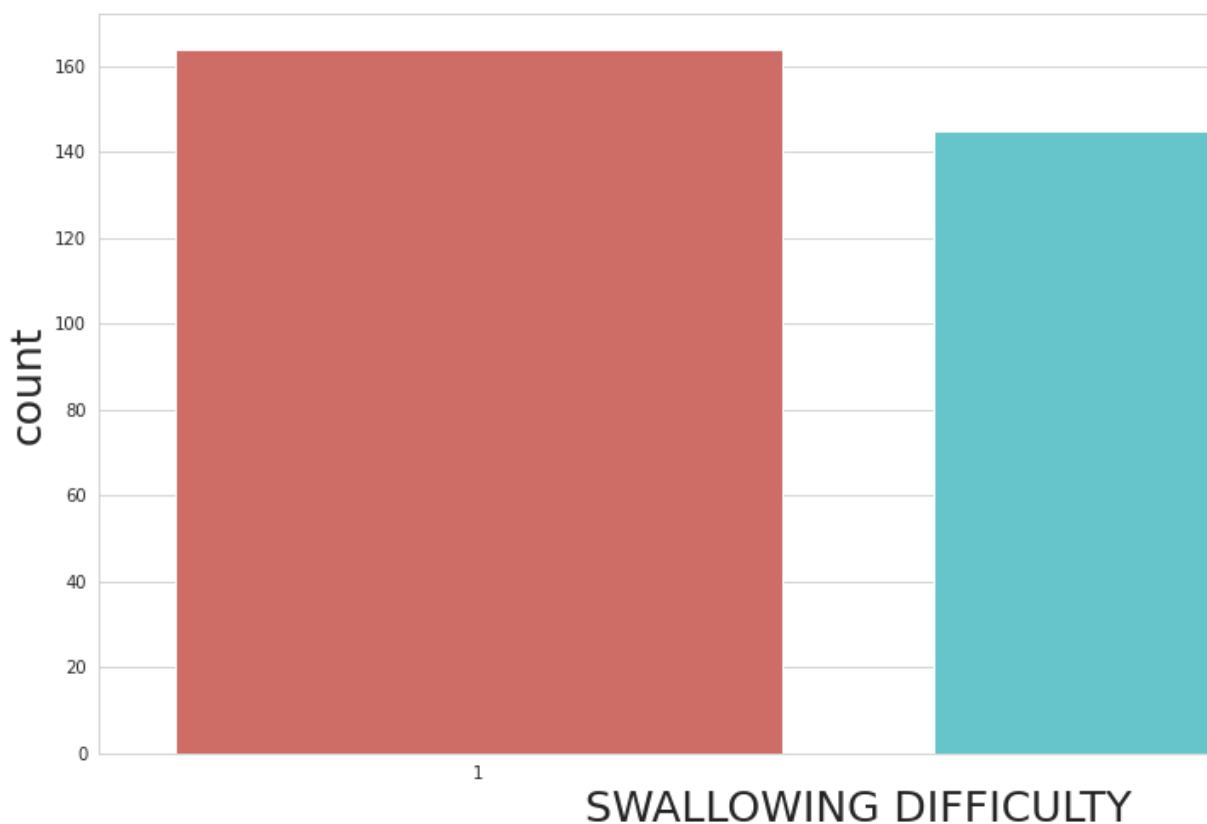
```
plt.figure(figsize=(16,8))
sns.set_style("whitegrid")
plt.title('Grouping People by CHEST PAIN ', fontsize=30, fontweight='bold', y=1.05,)
plt.xlabel('CHEST PAIN',fontsize=25)
plt.ylabel('Count',fontsize=25)
sns.countplot(x="CHEST PAIN", data=df, palette="hls");
plt.show()
```

Grouping People by CHEST P



```
plt.figure(figsize=(16,8))
sns.set_style("whitegrid")
plt.title('Grouping People by SWALLOWING DIFFICULTY ', fontsize=30, fontweight='bold', y=1)
plt.xlabel('SWALLOWING DIFFICULTY', fontsize=25)
plt.ylabel('Count', fontsize=25)
sns.countplot(x="SWALLOWING DIFFICULTY", data=df, palette="hls");
plt.show()
```

Grouping People by SWALLOWING D



```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
```



```

df['GENDER'] = encoder.fit_transform(df['GENDER'])
df['LUNG_CANCER'] = encoder.fit_transform(df['LUNG_CANCER'])

df["GENDER"].unique()

array([1, 0])

x = df.drop(["LUNG_CANCER"], axis = 1)

y = df["LUNG_CANCER"]

x = x.values
y = y.values

```

x

```

array([[ 1, 69,  1, ...,  2,  2,  2],
       [ 1, 74,  2, ...,  2,  2,  2],
       [ 0, 59,  1, ...,  2,  1,  2],
       ...,
       [ 1, 58,  2, ...,  1,  1,  2],
       [ 1, 67,  2, ...,  2,  1,  2],
       [ 1, 62,  1, ...,  1,  2,  1]])

```

y

```

array([1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1])

```

MODELS USED: Our Model includes three different kinds of algos which are logistic regression, Random Forest classifier and Support Vector Machine, with training and test data splatted into 8:2 ratio of the dataset. These models are being implemented with the help of sklearn library and then the confusions matrix is generated. The main reason to use these algos is because the dataset is not continues and for predicting the lung cancer will require the algorithms which are used to classify the cases according to the given attributes.

```
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)

print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))

from imblearn.over_sampling import SMOTE

sm = SMOTE(sampling_strategy=0.5,k_neighbors=5,random_state = 100)
X_train_res, y_train_res = sm.fit_resample(x_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))

    Before OverSampling, counts of label '1': 216
    Before OverSampling, counts of label '0': 31

    After OverSampling, the shape of train_X: (324, 15)
    After OverSampling, the shape of train_y: (324,)

    After OverSampling, counts of label '1': 216
    After OverSampling, counts of label '0': 108

from sklearn.linear_model import LogisticRegression

mode = LogisticRegression()

mode.fit(X_train_res, y_train_res)

    LogisticRegression()

y_pred = mode.predict(x_test)

from sklearn.metrics import accuracy_score

print(accuracy_score(y_test, mode.predict(x_test)))

    0.9516129032258065

from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, y_pred))

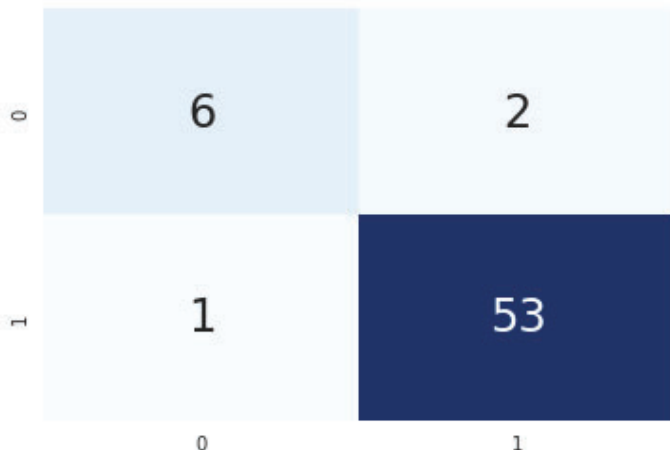
    precision    recall  f1-score   support

```

| | | | | | |
|--------------|---|------|------|------|----|
| | 0 | 0.86 | 0.75 | 0.80 | 8 |
| | 1 | 0.96 | 0.98 | 0.97 | 54 |
| accuracy | | | | 0.95 | 62 |
| macro avg | | 0.91 | 0.87 | 0.89 | 62 |
| weighted avg | | 0.95 | 0.95 | 0.95 | 62 |

```
cm_lr = confusion_matrix(y_test, y_pred)
sns.heatmap(cm_lr, annot=True, cmap="Blues", fmt="d", cbar=False, annot_kws={"size": 24})
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f1e510cf150>



```
from sklearn.ensemble import RandomForestClassifier
```

```
model = RandomForestClassifier(n_estimators = 3, criterion = "entropy")
```

```
model.fit(X_train_res, y_train_res)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=3)
```

```
y_pred = model.predict(x_test)
```

```
print(accuracy_score(y_test, model.predict(x_test)))
```

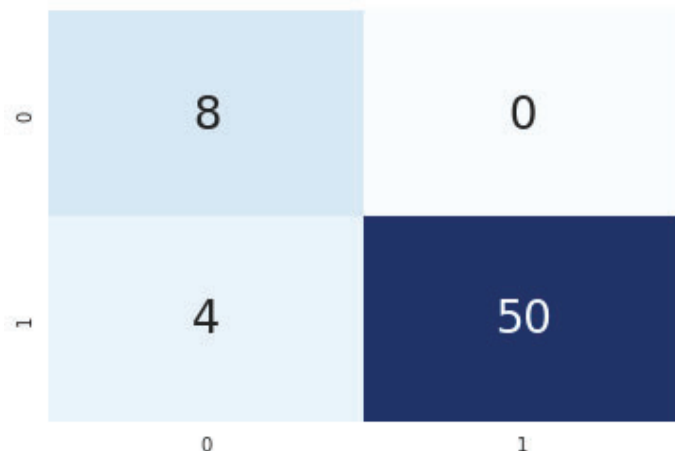
```
0.9354838709677419
```

```
print(classification_report(y_test, y_pred))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.67 | 1.00 | 0.80 | 8 |
| 1 | 1.00 | 0.93 | 0.96 | 54 |
| accuracy | | | 0.94 | 62 |
| macro avg | 0.83 | 0.96 | 0.88 | 62 |
| weighted avg | 0.96 | 0.94 | 0.94 | 62 |

```
cn_rf = confusion_matrix(y_test, y_pred)
sns.heatmap(cn_rf,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"size": 24})
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f1e502ed690>



```
from sklearn import svm
```

```
classifier = svm.SVC(kernel='linear')
```

```
classifier.fit(X_train_res, y_train_res)
```

```
SVC(kernel='linear')
```

```
y_pred = classifier.predict(x_test)
```

```
print(accuracy_score(y_test, classifier.predict(x_test)))
```

```
0.9516129032258065
```

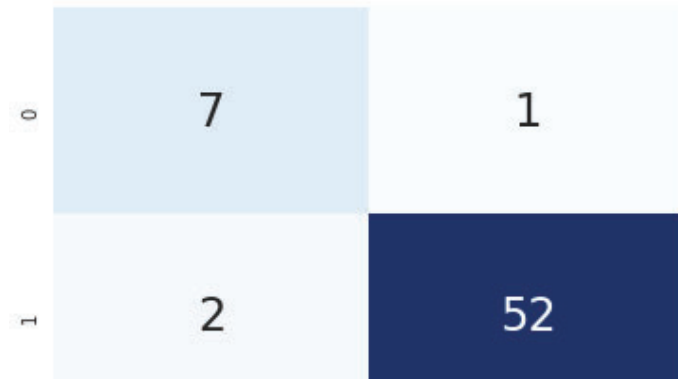
```
print(classification_report(y_test, y_pred))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.78 | 0.88 | 0.82 | 8 |
| 1 | 0.98 | 0.96 | 0.97 | 54 |
| accuracy | | | 0.95 | 62 |
| macro avg | 0.88 | 0.92 | 0.90 | 62 |
| weighted avg | 0.95 | 0.95 | 0.95 | 62 |

```
cn_svm = confusion_matrix(y_test, y_pred)
```

```
sns.heatmap(cn_svm,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"size": 24})
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f1e4da8d390>



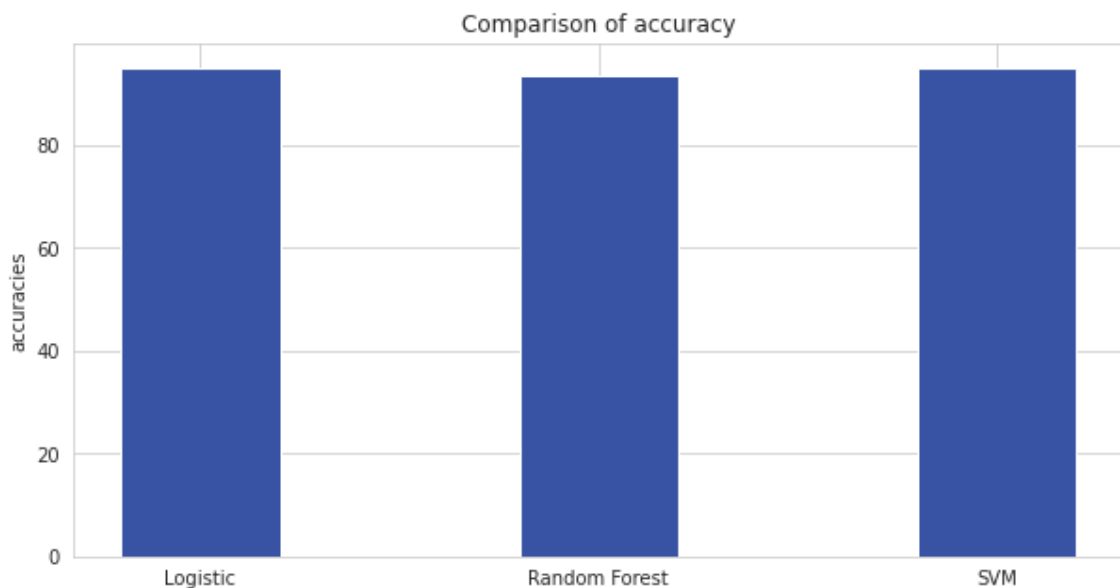
```
import numpy as np
import matplotlib.pyplot as plt

# creating the dataset
data = {'Logistic':95, 'Random Forest':93.5, 'SVM':95
        }
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color = 'blue',
        width = 0.4)

plt.ylabel("accuracies")
plt.title("Comparison of accuracy")
plt.show()
```



[Colab paid products](#) - [Cancel contracts here](#)

