Participant

Title= PhD Student\ Name= "Azeem Javed"\ email = "azeem.*mmg*@gmail.com"\ whatsapp = "0092306-5626700"

DAY_0

10 Important Steps To Become a Data Scientist

1. Learn the Basic MAthematics.
2. Learn the Basic Statistics.
3. Learn a programming language Python or R.
4. Data wrangling and Data Cleaning.
5. Data Visulaization.
6. Machine/Deep Learning Algorithms.
7. linux/conda/GIT enviornments.
8. Open source databases.
    - GIThub
    - Kaggle.com
9. Domain specific data science projects
10. Motivation and Good Pc

BASICS (Before Starting)

```
**02_operators**
```

- spaces are meaningless in operators
- sequences of operators run will always be like (pemdas (parenthesis exponents multiply divide addition subtraction)
- (sequence will be left to right mdas)

```
**03_Strings**
```

1. any text in excel (single/double commas or 3 qutation marks are strings)
2. print("hello world")
3. print("we are lreaning python with ammar")
4. print('test for single quotes')
5. print("test for double quotes")
6. print('''test for triple quotes''')
7. print(" what's up ?")

```
**04_Comments**
```

- print("how are you?") #ctrl+/ to comment out
- print ("we are learning python with ammar")
- print(2+6)

```
**05_ variables**
```

- variables are objects containig specific values
- x=5 (example numeric variable)
- print(x)
- y="we are learning pyton dr" #string variable
- print(y)
- x=123
- print(x) x=x+123+67 print(x)\ **Finding types of variables**
  - type(x)
  - print(type(x))
  - print(type(y))

```
Rules to assign  a variable or a object in a variables
```

- variable should contain letters numbers or underscores
- do not start with numbers (ex=2y)
- spaces are not allowed
- do not use python a key word in function (help google "key words in python")
- short and descriptive variable names
- case sensitivity (lower and upper case) always use lower case leters\ fruit_basket=8\ fruit_basket="mangoes"
- del fruit_basket to delete something\ print(type(fruit_basket))\ print(fruit_basket)

06_input_variables

- input function using 3 ways

In [ ]:

```
# simple function
fruit_basket="mangoes"
print(fruit_basket)

# input function

fruit_basket=input("which is your fruit")
print(fruit_basket)

# input function 2nd stage
name=input("what is your name")
greetings="hello"
print(greetingd, name)

# another way of stage 2 function

name=input("what is your name")
print("hello", name)

###3rd stage input function
name= input("what is your name")
```

```
age=input("how old are you")
greeting="hello"
print(greeting, name, ",you are still young")
```

07_Conditional_Logical_Operators

- logical operators are true/false//yes/no/1/0/
- equal to ==
- not equal to !=
- less than <
- greater than >
- less than and equal to <=
- greater than and equal to >=

Methods of using operators

In [ ]:

```
#one way to put operators in code
print(4==4)
print(4!=4)
print(4>3)
print(3<6)
print(3<=6)
# 2nd wayinput fnctn and logical operator
age_at_school=5
hamid_age=input("how old is hamid") # input function
hamid_age=int(hamid_age)
print(type(hamid_age))
print(hamid_age==age_at_school) #logical operator
```

In [ ]:

```
# applications of logical operators
# brother to school example

hamid_age=4
age_at_school=5
print(hamid_age==age_at_school)
```

In [ ]:

```
# 2nd wayinput fnctn and logical operator
age_at_school=5
hamid_age=input("how old is hamid") # input function
hamid_age=int(hamid_age)
print(type(hamid_age))
print(hamid_age==age_at_school) #logical operator
```

08_Type_Conversion

- x= 10 #integer
- y=10.2 #float =integ*float
- z="hello" #string

In [ ]:

```
# implicit type conversion
x=34
```

```
y=5
x=x+y
print(x,"type of x is:", type(y))
```

In [ ]:

```
# explicit type conversion
age=input("what is your age")
age=int(age)
print(age,type(int(age)))
```

In [ ]:

```
#name
name=input("what is your name")
print(name, type(str(name)))
```

09_If_Else_Elif

In [ ]:

```
# required age at school
hamid_age=2
required_age_at_school=5
```

In [ ]:

```
# can/can't go to school
if hamid_age==required_age_at_school:
    print("hamid can join school")
elif hamid_age > required_age_at_school:
    print("hamid should join h scjool")
elif hamid_age==2:
    print("care baby")

else:
    print("hamid can not go to school or should join high class")
```

10_Functions

In [ ]:

```
# direct writing
from typing import Text

print("Azeem ")
print("Azeem ")
print("Azeem ")
print("Azeem ")
print("Azeem ")
print("Azeem ")
```

In [ ]:

```
## defining a Function (First way)

def print_cds():
    print("Azeem ")
    print("Azeem ")
    print("Azeem ")
print_cds()
```

In [ ]:

```
# defining a function (second way)
```

```
def print_cds():
    Text= "is it changed "
    print(Text)
    print(Text)
    print(Text)
print_cds()
```

In [ ]:

```
# defining a function (third way)
def print_cds(Text):
    print(Text)
    print(Text)
    print(Text)
print_cds("Azeem")
```

In [ ]:

```
# defining a function with if elif else statement

def school_calculator(age, text):
    if age==5:
        print("hamid can join the school")
    elif age>5:
        print("hamid should go tohigherschool")
    else:
        print("hamid is a baby")
school_calculator(2, "hamid")
```

In [ ]:

```
# defining a future  function (advance)
def future_age(age):
    new_age=age+20
    return new_age
    print(new_age)
future_predicted_age= future_age(5)
print(future_predicted_age)
```

11_Loops

In [ ]:

```
# 1- while loop
x=0
while (x<=   5):
    print(x)
    x=x+1
```

In [ ]:

```
# 2- For Loop
for x in range(5,11):
    print(x)
```

In [ ]:

```
# 3- Arrays
days= ["mon", "tue", "wed", "thrus", "fri",
"sat", "sun" ]
for d in days:
    # if (d=="fri"):break #(loop stops)
    if (d=="fri"): continue  # skips
    print(d)
```

12_import_Libraries

In [ ]:
```
# if you want to print the valu of pi
import math
print ("the value of pi is", math.pi)
import statistics
x=[150,250,350,450,550]
print(statistics.mean(x))
# important libraries numpy, pandas
```

13_Trouble_Shooting

- there are two types errors
- syntax error
- easy to remove also called as syntax error

14_Important_Information

- installation
- command_line tool
- vs code and p[ython
- first line of code
- operation
- comments
- variables
- type conversions
- if, elif, else

Data Visulaization Week

- steps involved in data visualization

In [ ]:
```
# import library
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="ticks", color_codes= True)
```

In [ ]:
```
# importing datasets /you can also import your own data(in this step)
kashti = sns.load_dataset("titanic")
print(kashti)
```

In [ ]:
```
# plot basic design(4)
P= sns.countplot(x="sex", data= kashti)
plt.show()
```

In [ ]:
```
# step 5
P= sns.countplot(x="sex", data= kashti, hue="embark_town")
```

```
# step 6 titels
P.set_title("First Graph of life using Python")
plt.show()
```

- Countplot

In [ ]:

```
# Data Visulization using meeting generated file
import pandas as pd
doctors=pd.read_csv("data_chilla.csv")
print(doctors)
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(style="ticks", color_codes=True)
p=sns.countplot(x="Gender", hue="Location", data=doctors)
p.set_title("is plot is correct")
plt.xlabel("population")
plt.ylabel("numbers")
plt.show()
```

- catplot

In [ ]:

```
# catplot
import seaborn as sns
import matplotlib.pyplot as plt
sns. set_theme(style="ticks", color_codes=True)
titanic = sns.load_dataset("titanic")
print(titanic)
sns.catplot(x="sex", y="fare", hue="adult_male", kind="bar", data=titanic)
plt.show()
```

- scatter plot

In [ ]:

```
# scatter plot
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="ticks", color_codes=True)
titanic = sns.load_dataset("titanic")
g=sns.FacetGrid(titanic, row="sibsp", hue="class")
g=(g.map(plt.scatter, "age", "fare").add_legend())
plt.show()
```

- lineplot

In [ ]:

```
#line plot from collected data set
import pandas as pd
homework=pd.read_csv("Chilla_data2_for_plots.csv")
print(homework)
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(6,4))
sns.lineplot(x="Age", y="Gender", data=homework,)
plt.title("Line Plot Representing Ages of participants")
plt.xlabel("Ages")
plt.ylabel("male/female")
```

```
sns.set_style("darkgrid") #, {"grid.color": ".9", "grid.linestyle": ":"})
plt.xlim(2)
plt.ylim(1)
plt.show()
```

Plotly Express

- interactive plots

In [ ]:

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species")
fig.show()
```

In [ ]:

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species", marginal_
           marginal_x="box", trendline="ols", template="simple_white")
fig.show()
```

In [ ]:

```
import plotly.express as px
data_canada = px.data.gapminder().query("country == 'Canada'")
fig = px.bar(data_canada, x='year', y='pop')
fig.show()
```

- for data visualization we can also use seaborn library gallery

Basic Data structures in python

1. Tuple
2. list
3. Dictionaries
4. Set

1. Tuple

- orderd colection of elements
- enclosed in () round braces
- different elements can be stored
- once element are stored you can change them (unmutable)
- cannot use replace function

1. list

- orderd collection of elements
- enclosed in [ ] square brackets /braces
- mutateable you can chamge the values

1. Dictionaries

- an unordered collection of element\
  `1- key 2- value`
- always used with curly braces/bravckets
- mutateable

4-sets

- unordered and unindexed
- curely braces are used
- no duplicates allowed

Next Week

- Libraries

NumPy (Numerical Python)

- Basics

In [ ]:
```
# pip install numpy
import numpy as np
a = np.array([5,5,5,5,5])
a
type(a)
```

In [ ]:
```
#ists of arrays
a =([[2, 2,2], [2,2,2],[2,2,2]])
```

In [ ]:
```
import numpy as np
a = np.array([1,2,3,4,5])
```

In [ ]:
```
b = np.zeros(5)
c = np.ones(3)
e = np.arange(2, 60, 5)
c= np.arange(24).reshape(2,3,4)
```

In [ ]:
```
# methods of creating arrays
food= np.array(['pakora', 'samosa', 'raita'])
price=([5,5,5,])
```

In [ ]:
```
type(price)
type(food)
```

In [ ]:
```
price[0:]
food[0:]
```

Array Functions

In [ ]:
```
a= np.array([10,3,4.4,6,9,7])
a.sort()
a
```

In [ ]:
```
b=np.array([9,0.8,56,43,2,])
c=np.concatenate((a,b))
c
```

2-D arrays

- concatenation

In [ ]:
```
a=np.array([[1,2,3,4,],[5,6,7,8,]])
b=np.array([[5,6,3,1], [3,5,4,2]])
c=np.concatenate((a,b), axis=1)
c
```

In [ ]:
```
# finding dimensions of
d=np.array([[[1,2,3,4,],[5,6,7,8,], [6,7,8,9]]])
d.ndim
```

In [ ]:
```
# converting 1-D into 2-D
a= np.array([10,3,4.4,6,9,7,8])
b=a[np.newaxis,:]
b
```

Mark Down Language complete

1- Headings

how to hove headings in mark down files?

Heading 1

Heading 2

Heading 3

Heading 4

2- Block of words

This is a normal Text in mark down.

> this is a block of special text
>
> this is second line

3- Line breaks

press enter to line break.\ or you can put a back slash.\ this is 40 days long courseof data science with python ka chilla Python_ka_chilla_with_ammar.\ this is a second line

4- combine two things

block of words and headings

> ## heading 2

face of text

ittalic *italic* we can also use symbols single underscore \ bold **bold** \ we can also use symbols double underscore bold and italic ***bold and italic***

> write in comments about Bold and ittalic with underscores

adding bullet points or lists

- day-1
- day-2
- day-3
- day-4
- day-5
  - day-5a
    - any sublist
  - day-5b
- day-6

  > using ***or+**

- one
- one

numbring of list

1. day1
2. day2
3. day3

4. day 4
5. day5

    A. day5a
        a. day5b
            i. day5c
6. day6

7- line breaks or page breaks

this is page 1.

---

this is page 2.

8-links and hyper links

hyper link https://mail.google.com/mail/u/0/#inbox click here

this link is here

heading 9- images and figures with link

the pic i wnt to add is here: pic

> how to comment out a markdown line and its short cuts? online picture: azeem

10- adding codes or code blocks

to print a string name use `print("azeem")` to add block of code

this code shows color according to syntax(just add name of language)

```
x=5+6
y= 3-2
z= x+y
print(z)
```

11- adding tables

species	petal length	sepal length
virginica	18.9	445.7
satosa	18.9	445.7

species	petal length	sepal length
hgj	18.9	445.7
virginica	18.9	445.7
virginica	18.9	445.7

12- table of content

headings\ block of words\ text face\ tables

13- install extension

*_sample text_*

Day_12 PANDAS

- (Pannel Data Analysis)
- used for data manipulation and data analysis

- Time line
- 1. 2008 development started
- 1. 2009 become open source
- 1. 1st Edition of *python for data analysis published* in 2012
- 1. 2015 Pandas become a NumFOCUS sponserd project
- 1. First in person core developer sprint 2018

- we use because we can easily manipulate and analyse data
- to tidy up our data and make our data usefull # Types of DATA Analysis
- Descriptive Analytics : what happend?
- Diagnostic Analytics : why happend?
- Predicive Analytics : what is likely to happen in future
- Prescriptive Analytics : what is the best course of action to take?

Different terms

- Data frame - worksheet
- series - columns
- index - row headings first column of a row
- Row - row
- NaN - empty cells missing values

Merits

- python library

- data analysis and manipulation
- data science
- machine learning
- if yourwork invove data you must use it
- its free
- load, prepare, read , model and analyze
- join,merge, filter, reshape
- data from different data bases and join in one
- anything with data
- datafeames= pandas data strucutres
- can downoad data in excel

Pandas Practice

In [ ]:

```
# importing libraries
import pandas as pd
import numpy as np

# object creation
s = pd.Series([1,3, np.nan,5, 7, 8,9])
dates = pd.date_range("20130101", periods=20)
```

In [ ]:

```
df = pd.DataFrame(np.random.randn(20, 4), index=dates, columns=list("ABCD"))
df
```

In [ ]:

```
df2 = pd.DataFrame(
    {
        "A":1.0,
        "B":pd.Timestamp("20221101"),
        "C":pd.Series(1, index=list(range(4)), dtype="float32"),
        "D":np.array([3] * 4, dtype="int32"),
        "E":pd.Categorical(["girl", "women", "girl", "women"]),
        "F":"females"
    }
)
df2.dtypes
```

In [ ]:

```
## how to view data in pandas
df.head(2)
df.index
df.tail(2)
```

In [ ]:

```
# convert data to numpy
df2.to_numpy()

#converting to arrays
df.to_numpy()
```

In [ ]:

```
#describing
df.describe()
```

```
# to transpose  data
df2.T

## sorting to save
df.sort_index(axis=0, ascending=True)
df.sort_values(by="C", ascending=True)

# selection filtartion
df["A"]
## indexing row wise selection
df[0:]
```

In [ ]:

```
## selecting a specific data
df.loc[dates[15]]
## using multiple labels
df.loc[:, ["A", "B"]]
# data selection at very specific point
df.loc["20130102":"20130104", ["A", "B","C"]]

# list inside a list
df.loc[["20130102","20130104"], ["A", "B"]]
df.loc["20130102", ["A", "B","C"]]

# selection at secific index
df.at[dates[0], "A"]
```

In [ ]:

```
# indexing on base of position
df.iloc[3:10]
df.iloc[0:10, :]
df.iloc[0:15,0:3]
df.iloc[:,0:2]
```

In [ ]:

```
# indexing with bolean operators
df[df["A"]>0]
df[df["A"]>1.5]
df[df>0] # missing values function

# indexing with bolean operators
df[df["A"]>0]
df[df["A"]>1.5]
df[df>0] # missing values function
```

In [ ]:

```
# concatenation or addition(adding another column)
df2["E"] = ["one","one", "two","three", "four", "three",
"one","one", "two","three", "four", "three","one","one", "two","three", "four", "t
 "four", "three"]
df2

df2["mean"]=[1.2,2.3,4.0,1.3,3.2,1.2,2.3,4.0,1.3,3.2,1.2,2.3,4.0,1.3,3.2,1.2,2.3,4
df2

## finally updating df2
df2["new"]=[1.2,2.3,4.0,1.3,3.2,1.2,2.3,4.0,1.3,3.2,1.2,2.3,4.0,1.3,3.2,1.2,2.3,4.
df2
df2 = df2.iloc[:, 0:4]
df2
```

EDA (EDA Exploratory Data Analysis)

- this will show how can we use EDA in Python

three inmportant steps to keep in mind are:

1. Understand the data
2. Clean the data
3. find the relation ship between data

- Hands on practice of titanic dataset

In [ ]:

```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
# creating object
kashti= sns.load_dataset('titanic')

# saving the dataset to local repository
kashti.to_csv('kashti.csv')

# getting all information of dataset
kashti.info
ks=kashti # creating another object from previus object
```

In [ ]:

```
# datasets view
ks.head()
ks.tail()

# to check rows and columns numbers
ks.shape
# to get all statisical info
ks.describe()

# to obtain unique values
ks.nunique()

# to get the column names
ks.columns

# example unique (in a apecific column)
ks['who'].unique()

#  example unique (in mutiple column)
pd.unique(ks['survived'])
pd.unique(ks[['who', 'class','alone']].values.ravel('K'))
# numpy function to find uniques in multiple columns
np.unique(ks[['who', 'class','sex']].values)
```

In [ ]:

```
# cleaning and filtering the data (to find missing values)
ks.isnull()
```

```
ks.isnull().sum()

# removing missing data or cleaning data
ks.drop(['deck'],axis=1)
# remember to make new object after cleaning the data
ks_clean=ks.drop(['deck'],axis=1)
ks_clean.head() # after cleaning view

ks_clean.isnull().sum()
ks_clean.shape
```

In [ ]:

```
# dropping all missing values
ks_clean.dropna()
# view after cleaning
ks_clean.isnull().sum()
# counting of specic information in a column
ks_clean['age'].value_counts()
```

- above data is cleaned
- down is to find out outliers
- making the plots and normalization of cleaned data

In [ ]:

```
ks_clean.columns
sns.boxplot(x='sex',y='age', data=ks_clean)
```

In [ ]:

```
sns.boxplot(y='age', data=ks_clean)
```

In [ ]:

```
sns.distplot(ks_clean['age'])
```

In [ ]:

```
sns.distplot(ks_clean['age'])
```

In [ ]:

```
ks_clean['age'].mean()
```

In [ ]:

```
sns.boxplot(y='age', data=ks_clean)
```

In [ ]:

```
ks_clean.hist()
```

In [ ]:

```
pd.value_counts(ks_clean['class']).plot.bar()
```

In [ ]:

```
ks_clean.groupby(['sex', 'class']).mean().plot.bar()
```

In [ ]:

```
ks.groupby(['sex', 'class']).mean().plot.bar()
```

In [ ]:

```
ks1=kashti
kashti
```

In [ ]:
```
ks_clean.corr()
```

In [ ]:
```
sns.heatmap(corr_ks_clean)
```

In [ ]:
```
sns.heatmap(corr_ks_clean,annot=True)
```

In [ ]:
```
sns.relplot(x='age', y='fare', data=ks_clean)
```

In [ ]:
```
sns.relplot(x='age', y='fare', hue='sex', data=ks_clean)
```

In [ ]:
```
sns.catplot(x='sex', y='age', hue='sex', data=ks_clean, kind='box')
```

In [ ]:
```
#log transformation
sns.distplot(ks_clean['fare'])
ks_clean['fare_log']=np.log(ks_clean['fare'])
```

In [ ]:
```
ks_clean.head()
```

In [ ]:
```
sns.boxplot(x='sex', y='fare', hue='sex', data=ks_clean)
```

In [ ]:
```
sns.boxplot(x='sex', y='fare_log', hue='sex', data=ks_clean)
```

Research Design in Data Science

- Choosing a right statistical method

1. Main Objectives
    - this will enhance your learning
    - choosing a right statistical method
    - Do's and Dont's of statistics
    - Reliable resuts
    - Paper revisions with Proof of statistical test(wh question)
    - Making data visulaization
    - Interpreting the results

1. there are two types of tests
    - Parametric tests
        - more reliable results
        - First we have to meet the assumptions
            - Non para metric tests
            - less reliable results
            - calculates the ranks of the data

    - no need to make assumptions

Steps For Applying/Selecting a Specific Statstical Test

1. Step 1 Normality test

- tests to be used
    1. Shapiro wilk test
        - Specific Reliable
    2. Kolmogorov Test
        - General (Less reliable)

1. Step 2 Homogeniety test
    - the variance of the variable in data are equal

- test tobe used are\ ***LEVENE'S TESTS***

1. Step 3 Purpose of data analysis
    - know the purpose of your research question
    - different examples

Two types of Purposes

The purpose of statistical test must be clear

1. Comparison
    - also called as differences
2. Relationship
    - connection

- if purpose is comparison then you at least have two groups to compare\ (examples) Male vs Female
- control group vs treatment groups
- grouping individuals by color preferences etc

- if the purpose is relationship (finding a connection)
- examples
- connection/correaltion/causation / prediction

1. Data Type Produced by desired purposes
    - know the type of your data you are working with

- There are two types of data

1. Categorical Data
    - qualtative

    - non numerical meaning
    - represnted in texts e.g charactors and factors
2. Continuous Data
    - quantitative
    - Numercal
    - Mostly represented in numbers
    - e.g numerical variables int and float

Note

- if you work with quality of data then data must be categorical
- if you work with rhe quantity of data then data must be continuous

1. Choosing an appropriate statistical test

- choose a statistical test from three main families
    1. Chi squared test family
        - purpose is comparison and data used is categorcical type
        - only (Chi-Squared)
    2. t-Test/ANOVA
        - purpose: comparisons
        - data : categorical and continuous
        - t-test
    3. Correlation
        - purpose: relationship
        - data: continuous only (correlation)

when and where to use?

1. Chi squared test family # Two Types
    A. chi square test of homogeniety
    B. chi square test of independence

- when to use ?
- nothing effects this
- can be used with any number levels of groups\ ***but you must remember the purpose and data type***

when and where to use?

1. test/ANOVA
    A. one sample t-test(for one sample group with a known mwan)
    B. Two Sample t-test:
        - Un-paired t-Test (Two different Groups)
        - Paired t- Test (Same Group Twice)
2. ANOVA (Analysis of Variance) 3+ levels or groups are involved
    - one way ANOVA (Even one of group is significant you will get significant results,

```
but does'nt tell you which one:,)
```

- Two Way ANOVA
- Repeated measures of ANOVA (3+ paired groups, scale up of Pairedt-Test)

when and where to use?

1. Correlation
   - purpose:relationship
   - data:continuous only (correlation) Types
     A. Pearson's Correlation (one Independent and one dependent variable)
     B. Regression: (one Independent and one dependent variable)

- Correlation: tell us how closly connected two variables are?
- ""is food a predictor of weight gain""
- Regression: Tells us a specific mathematicall equation that describe the relationship.
- (this helps us to find the data points not measured yet)
- e.g. missing values can be predictedlike this

Assumption about your data (these tests trusts you)

important notes

- your data is normally distributed
- or follow a gaussian distribution
- non reliable results
- if you do not follow assumption and breaks the trusts of three families , they will not happy with you

if the assumption not met your data is normal

- check normality
- check homogeniety

Statistical tests in python (Hands on Training)

Shapiro Wilk Test

- Tests whether a data has a gaussian distributions\ **Assumptions**
  1. Observations in each sample are independent and identically distributed (iid)
  2. Interpretation
     - HO sample has a gaussian normal distribution (null hypothesis)
     - H1 the sample does not have gaussian normal distribution (alternative hypothesis)

In [ ]:

```
# import libraries
```

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

In [ ]:

```
# ks= sns.load_dataset('titanic')  ### for practice
from scipy.stats import shapiro
data= [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
# pvalue results rejects null hypothesis
shapiro(data)
```

In [ ]:

```
# if we add shapiro test we can directly see p value
from scipy.stats import shapiro
data= [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
stat,p = shapiro(data)
print(stat)
## to print any string we can use
print('stat=', stat)
## to print any string we can use
print('p=', p)
# print(p)
```

In [ ]:

```
#another way to obtain statistical test and p values
#(but this method code is hard to remember)
print('stat=%.3f, p=%.3f' % (stat,p))     # just like respectively
```

In [ ]:

```
# can use different statement according to our need
if p > 0.05:
    print('data is normal')
else:
    print('data is not normal') # if p value is less than 0.5 than data is not nor
```

- Check the normality of data by data visulaization

In [ ]:

```
# loading kashti data set (sorce code)
ks= sns.load_dataset('titanic')
ks['age'].hist()
sns.boxplot(ks['age'])
```

- Normality check after droping empty values

In [ ]:

```
# normality test (to make our data normal)  (source code)
from scipy.stats import shapiro
ks = ks.dropna()
stat, p=shapiro(ks['age'])
if p > 0.5:
    print('Probably normally distributed')
else:
    print('Probably not normally distributed')
```

In [ ]:

```
# boxplot after droping empty values (normalized data)
sns.boxplot(ks['age'])
```

- normalization of "Fare"

In [ ]:
```
## for fare
# Normality test
from scipy.stats import shapiro
ks=ks.dropna()
sns.boxplot(ks['fare'])
stat,p = shapiro(ks['fare'])
if p > 0.05:
    print('the data is normal')
else:
    print('the data is not normal')
```

2- Correlation test

1. Pearson's Correlation Coefficient
   - Tests whether two samples/variables have a linear relationship

- **Assumptions**
  1. Observations in each sample are independent and identically distributed (iid)
  2. observation in each sample are normally distributed
  3. oobservation in each sample have the same variance
  4. Interpretations
     - HO:the two samples are independent.(not related)
     - H1:there is dependency in variables (related)

In [ ]:
```
# # Example of the Pearson's Correlation test (python example code)
from scipy.stats import pearsonr
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.86
data2 = [0.353, 3.517, 0.125, -7.545, -0.555, -1.536, 3.350, -1.578, -3.537, -1.57
stat, p = pearsonr(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))  ## don't use this mushkil wali command
if p > 0.05:
  print('Probably independent')
else:
  print('Probably dependent')
```

1. Spearson's Rank Correlation test (non parametric analysis)
   - Tests whether two samples/variables have a monotonic relationship

- **Assumptions**
  1. Observations in each sample are independent and identically distributed (iid)
  2. oobservation can be ranked
  3. Interpretations
     - HO:the two samples are independent. (not related)
     - H1:there is dependency in variables (related)

In [ ]:
```
#examples of spears man correlation test
from scipy.stats import spearmanr
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.86
data2 = [0.353, 3.517, 0.125, -7.545, -0.555, -1.536, 3.350, -1.578, -3.537, -1.57
stat, p = spearmanr(data1, data2)
```

```
stat, p = spearmanr(ks['age] , ks['fare])  ## to adopt
print('stat=%.3f, p=%.3f' % (stat, p))  ## don't use this mushkil wali command
if p > 0.05:
  print('Probably independent(no correlation')
else:
  print('Probably dependent(correlation exists')
```

3- Chi squared test

- test whether two categorical variables are related or independent
- **Assumptions**
- observations used in calculations of contingency table are independent.25 or more examples in each cell of the contigency table.
- interpretations
  - HO: the two samples are independent
  - H1: there is dependency between the samples

In [ ]:

```
# examples of chi squared test
from scipy.stats import chi2_contingency
table = [[10,20,30], [6,9,17]]
stat,p,dof,expected=chi2_contingency(table)
# print('stat=', stat)
print(stat)
print(p)
if p > 0.05:
  print('Probably independent')
else:
  print('Probably dependent')
```

3- Parametric Statistical Hypothesis tests

1.student's t test

- tests whether the means of two independent variables are significantly different
- **Assumptions**
- 1. Observations in each sample are independent and identically distributed (iid)
  2. observation in each sample are normally distributed
  3. oobservation in each sample have the same variance
  4. Interpretations
     - HO:the means of samples are equal
     - H1:the means of samples are not/ unequal

In [ ]:

```
# Example of the Student's t-test
from scipy.stats import ttest_ind
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.86
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075, -0.1
stat, p = ttest_ind(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably the same dataset')
else:
  print('Probably different data set')
```

1. Paired student's t test

- tests whether the means of two paired samples are significantly equal
- **Assumptions**
- 1. Observations in each sample are independent and identically distributed (iid)
  2. observation in each sample are normally distributed
  3. oobservation in each sample have the same variance
  4. observations acrosseach sample are paired
  5. Interpretations
     - HO:the means of samples are equal
     - H1:the means of samples are not/ unequal

In [ ]:

```
# Example of the Paired Student's t-test
from scipy.stats import ttest_rel
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.86
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075, -0.1
stat, p = ttest_rel(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably the same distribution')
else:
  print('Probably different distributions')
```

4. Analysis of Variance ANOVA

- tests whether the means of two or more independent samples are significantly different
- **Assumptions**
  1. Observations in each sample are independent and identically distributed (iid)
  2. observation in each sample are normally distributed
  3. oobservation in each sample have the same variance
  4. Interpretations
     - HO:the means of samples are equal
     - H1:one or more means of samples are not/ unequal

In [ ]:

```
# Example of the \ANOVA
from scipy.stats import f_oneway
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.86
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075, -0.1
data3 =  [-0.208,0.696,0.928,-1.148,-0.213,0.229,0.137,0.269,-0.870,-1.204]
stat, p = f_oneway (data1, data2, data3)
# print('stat=%.3f, p=%.3f' % (stat, p))
print(stat)
print(p)
if p > 0.05:
  print('Probably the same distribution')
else:
  print('Probably different distribution')
```

Machine Learning Week

There are four Types of Machine Learning

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-Supervised machine learning
4. Reinforcement Learning

1. **supervised learning types**\ a. classification (categorical)\ b. regression (numerical)
   - general work flow is below
   - --input data----->training model and learning patterns---->prediction---->output and reports

Algorithms used in supervised learning (Keep learning by self)

- Logistic Regression
- K-Nearest neighbour (k-nn)
- Support vector machine (svm)
- Kernel svm
- naive bayes
- Decision tree classification
- Random Forest classification

Random Forest classification\ The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Decision tree classification\ Decision Tree - Classification. Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.\ Decision trees can handle both categorical and numerical data

naive bayes\ In statistics, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong independence assumptions between the features. They are among the simplest Bayesian network models, but coupled with kernel density estimation, they can achieve higher accuracy levels.

Kernel svm\ SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example linear, nonlinear, polynomial, **radial basis function (RBF)**, and sigmoid.

Logistic Regression\ Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is ***dichotomous***, which means there would be only two possible classes.\ In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).\ Mathematically, a logistic regression model predicts P(Y=1) as a

function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

Support vector machine\ In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis.\ SVMs are one of the most robust prediction methods, being based on statistical learning frameworks

K-Nearest neighbour (k-nn)\ In statistics, the k-nearest neighbors algorithm (k-NN) is a non-parametric classification method used for classification and regression. In both cases, the input consists of the k closest training examples in a data set

1. Second Type of Machine Learnning **unsupervised Learning/clustring**
    - No supervision
    - no teacher
    - self learning
    - find patterns by it self

- --no input data----->training model and learning patterns---->prediction---->output and reports

Unsupervised Learning algorithms

- K- Means clustring
- Hierarchical clustring
- Probablistic clustring

Probablistic clustring\ In probabilistic clustering the assignment of points to clusters is "soft", in the sense that the membership of a data point x in a cluster Ck is given as a probability, denoted by pk(x).\ These are subjective probabilities, indicating strength of belief in the event in question.

Hierarchical clustring\ Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.

K- Means clustring\ Kmeans algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group.\ The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

3. **Semi supervised**

- some labelled data and some unlabelled data **mixture of 1 and 2**
    - some data is labelled whilw some is not
    - we use both supervised and unsupervised algorithms (usually used for empty values)

1. # **Reinforcement learning** (Self learning)

- Hit and trial learn
- Learn from mistakes
- reward and punishment rules
- prediction based on reward and punishment
- depends on feedback
- --->input data(but not a learning data)---->training model and learning data by itself

reinforcement Learning algorithms

- Model free reinforcement learning
  1. Policy optimization
  2. Q-learning
- Model based reinforcement learning
  1. learn the model (model based approach)
  2. given the model (system based approach)

Model based reinforcement learning\ Reinforcement learning systems can make decisions in one of two ways. In the model-based approach, a system uses a predictive model of the world to ask questions of the form “what will happen if I do x?” to choose the best x1.

Model free reinforcement learning\ In reinforcement learning (RL), a model-free algorithm (as opposed to a model-based one) is an algorithm which does not use the transition probability distribution (and the reward function) associated with the Markov decision process (MDP), which, in RL, represents the problem to be solved.\ **Policy optimization**\ Policy optimization methods are centered around the policy, the function that maps the agent's state to its next action. These methods view reinforcement learning as a numerical optimization problem where we optimize the expected reward with respect to the policy's parameters.\ **Q-learning**\ Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.

Basic Libraries needed to install

- pip install tensorflow
- pip install scikit-learn

Hands on training on ML

simple Linear Regression

- y= a +bx .....> y=beta0 + betaX
- Dependent variable y
- independent variable x
- a= constant
- b= function/slope of x

Machine learning

1. simple learning regression

In [ ]:
```
# step 1 import libraries
import pandas as pd
df=pd.read_csv('ml_data_salary.csv')
df.head()
```

In [ ]:
```
# step 2 splitt data set into training data and testing data
x= df[['YearsExperience']]    ## to make 2d just put doble braces
y= df['Salary']
```

In [ ]:
```
# confirm splitting
x.head()
x.tail()
y.head()
y.tail()
```

In [ ]:
```
# import libraries to train splitted data sets
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_stat
```

In [ ]:
```
# fiting the linear regression model
from sklearn.linear_model import LinearRegression
model=LinearRegression().fit(x_train, y_train)
model
```

In [ ]:
```
#visulaization of fitted model
import matplotlib.pyplot as plt
plt.scatter(x_train, y_train)
# to make line in plot
plt.plot(x_train, model.predict(x_train), color="green")
plt.xlabel("tajurba")
plt.ylabel("tankhwa")
plt.title("train plot")
plt.show()
```

In [ ]:
```
# to make line in plot of test data
plt.scatter(x_test, y_test)
plt.plot(x_test, model.predict(x_test), color="green")
plt.xlabel("tajurba")
plt.ylabel("tankhwa")
plt.title("train plot")
plt.show()
```

In [ ]:
```
# checking the tested and trained model score
print('score of train data=',model.score(x_train, y_train))
print('score of test data=',model.score(x_test, y_test))
```

In [ ]:
```
# Prediction of unknown sample through trained model
model.predict([[100]])
model.predict([[5],[10],[20]])
```

In [ ]:
```
# prediction of test data X_test after training data
model.predict(x_test)
```

In [ ]:
```
# codes for saving the files/plotted data in png,jpeg,pdf,tiff
plt.savefig('PracticeGraph2.tiff', dpi=300)
plt.savefig('PracticeGraph2.png')
plt.savefig('PracticeGraph2.pdf')
```

Multiple Linear Regression

- only difference is it has more than one independent variables
- features/input data/ independent variables
- salary= i1*age+i2*distance+i3*years of experience+b

In [ ]:
```
# import libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
# import your data
df=pd.read_csv("multiple linear regression.csv")
df.head()
```

In [ ]:
```
x= df[['age','distance','YearsExperience']]
y= df['Salary']
```

In [ ]:
```
# creat and fit your model
model=LinearRegression().fit(x,y)
model
```

In [ ]:
```
# finding of model coef, intercept and model scores
model.coef_
model.intercept_
model().score()
```

In [ ]:
```
model.predict([[31.3,77.5,1.1]])
```

- to check the accuracy of model we have to split the data by 20/80 rule.

Decision Tree Classifiers

In [ ]:
```
# import libraries
import pandas as pd
df= pd.read_csv('mldata1.csv')
df.head()
```

In [ ]:
```
# change bolean operator to dummies
df['gender']= df['gender'].replace('Male',1)
```

```
df['gender']= df['gender'].replace('Female',0)
df.tail()
```

In [ ]:

```
# selection of input and output variables
x = df[['weight', 'gender']]
y = df['likeness']
x.head()
```

In [ ]:

```
# machine learning algorithm
from sklearn.tree import DecisionTreeClassifier
model= DecisionTreeClassifier()
# creat and fit our model
model= DecisionTreeClassifier().fit(x, y)
# model.fit(x, y)

# prediction
model.predict([[80,1]])
```

- Measuring the accuracy of the model

In [ ]:

```
# how to measure the accuracy of our model
###split data into test and train (80/20 rule)
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
x_train, x_test, y_train,y_test=train_test_split(x, y, test_size=0.2, random_state

# creat a model
model= DecisionTreeClassifier()
## fitting a model
model.fit(x_train, y_train)
##prediction values
predicted_values = model.predict(x_test)
predicted_values
## checking score
# y_test= actual values
score= accuracy_score(y_test, predicted_values)
score
```

- Training and Saving your Model

In [ ]:

```
## how to train and save your model
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import joblib

model = DecisionTreeClassifier().fit(x,y)

joblib.dump(model, "foodie.joblib")
model_from_joblib = joblib.load('foodie.joblib')
```

- Exporting the model

In [ ]:

```
from sklearn import tree
model= DecisionTreeClassifier().fit(x,y)
tree.export_graphviz(model,
```

```
                    out_file="Gender.dot",
                    feature_names=['age', 'weight','height', 'likeness'],
                    class_names=sorted(y.unique()),
                    label='all',
                    rounded= True,
                    filled= True)
```

importing the saved model for other data

In [ ]:
```
# import libraries
import pandas as pd
df= pd.read_csv('mldata1.csv')
df.head()
```

- Prediction of age using Wieght, height, gender , and likeness

In [ ]:
```
# get dummies from gender and likeness
df['likeness']= df['likeness'].replace('Biryani',1)
df['likeness']= df['likeness'].replace('Pakora',3)
df['likeness']= df['likeness'].replace('Samosa',2)
df['gender']= df['gender'].replace('Male',1)
df['gender']= df['gender'].replace('Female',0)
df.tail()
```

In [ ]:
```
# splitting the datasets
x = df[['weight','height', 'gender', 'likeness']]
y = df['age']
x.head()
```

- importing the saved model

In [ ]:
```
import joblib
model_from_joblib = joblib.load('Gender.joblib')
```

In [ ]:
```
# fitting the exported model
model_from_joblib.fit(x,y)
```

- Prediction through fitted Model

In [ ]:
```
# prediction
model_from_joblib.predict([[80,131,1,3]])
```

In [ ]:
```
# Measuring the score of fitted model
model_from_joblib.score(x,y)
```

K-Nearest Neighbours

K- Nearest Neighbors

- Minkowaski distance
- can also be used for numerical data/regression
- numbers of rooms
- years and maximum ratios

Accuracy checking method

- important
    1. Jaccard Index
    2. F1 Score
    3. Log loss
    4. Manymore
        - Classification accuracy
        - confusion matrix
        - area under curve
        - Mean absolute error
        - mean squared error

How to measure the accuracy of kNN model

- This is source code

In [ ]:

```
# split data into split and train and train (80/20)
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# here by changing accuracy scores you can see different metrics
#precision score
# recall_score
#F1-score
x_train, x_test, y_train, y_test =train_test_split(x,y, test_size=0.2) # 80% trair

# creat a model
model = DecisionTreeClassifier()
# fitting a model
model.fit(x_train, y_train)

predicted_values= model.predict(x_test)
predicted_values

##checking score
#y_test =actual_values

score=accuracy_score(y_test, predicted_values) #here too by changing accuracy scor
score
```

Pros

- training phase is faster
- instance based learning algorithms

- can be used with non linear data # Cons
- Testing Phase is slower
- Costly for memory and computation
- Not suitable for large dimensions # How to Improve
- data wrangling and scaling
- missing values treatment
- normalization on same scale for every thing (-1,0,1)
- Reduce dimensions to improve performance

In [ ]:

```
# actual example
# impoert libraries
import pandas as pd
df=pd.read_csv('mldata1.csv')
df['gender'] = df['gender'].replace('Male',1)
df['gender'] = df['gender'].replace('Female',0)
```

In [ ]:

```
# selection of input and output variables
x=df[['weight', 'gender']]
y=df['likeness']
```

In [ ]:

```
#model and Prediction
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5)

# train models using the training sets
model.fit(x,y)

#predicted output
predicted=model.predict([[70,1]])
predicted
# prediction of values of X
model.predict(x)
```

- Accuracy score check and applying metrics

In [ ]:

```
#metrics for evaluation
# split data into split and train and train (80/20)
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

x_train, x_test, y_train, y_test =train_test_split(x,y, test_size=0.2) # 80% trair
# creat a model
model = KNeighborsClassifier()
# fitting a model
model.fit(x_train, y_train)

predicted_values= model.predict(x_test)
predicted_values

##checking score
#y_test =actual_values

score=accuracy_score(y_test, predicted_values) #here too by changing accuracy scor
print("Accuracy score of our model=", score)
```

Logistic Regression

- Maximum Likely Hood
- Maximum likelihood estimation is a method that determines values for the parameters of a model. The parameter values are found such that they maximise the likelihood that the process described by the model produced the data that were actually observed.

- Hands on Training on Logistic regression

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

In [ ]:
```
# loading datasets built in
from sklearn.datasets import load_digits
digits= load_digits()
```

In [ ]:
```
# input variables/features (x=data)
digits.data.shape
x=digits.data
# means 1797 pictures size 64=8x8
```

In [ ]:
```
# output labels (y=target)
digits.target.shape
y=digits.target
```

- Loading 10 pics

In [ ]:
```
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:10],digits.target[0:10]))
    plt.subplot(1,10,index +1)
    plt.imshow(np.reshape(image,(8,8)),cmap=plt.cm.gray)
    plt.title('Training: %i\n'  % label, fontsize = 20)  ## error to be removed
```

In [ ]:
```
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:10],digits.target[0:10]))
    plt.subplot(1, 10, index + 1)
    plt.imshow(np.reshape(image,(8,8)),cmap=plt.cm.gray)
    plt.title(label, fontsize = 20)
# to get any help info
help(plt)
```

In [ ]:
```
# split the data
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_stat
```

In [ ]:
```
print("Trained input data:=", x_train.shape)
print("Test input data:=", x_test.shape)
```

```
print("Trained out put data:=", y_train.shape)
print("Test out put data:=", y_test.shape)
```

In [ ]:

```
#train the model
from sklearn.linear_model import LogisticRegression
model= LogisticRegression().fit(x_train,y_train)
model
```

In [ ]:

```
model.predict(x_test[0:10])
predictions=model.predict(x_test)
predictions
```

In [ ]:

```
## accuracy test
score=model.score(x_test,y_test)
print('the accuracy score id:', score)
```

- Applying Confusion Matrix

In [ ]:

```
###confusion matrics
from sklearn import metrics
cm =metrics.confusion_matrix(y_test,predictions)
cm
```

- Plotting confusion Matrix

In [ ]:

```
import seaborn as sns
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidth=.5, square=True, cmap='Spectral')
plt.ylabel('Actual outputs');
plt.ylabel('Predicted outputs');
all_sample_title='Accuracy score: {0}'.format(score)
plt.title(all_sample_title, size=15);
```

- Getting Misclassified Images

In [ ]:

```
# getting missed classified labels
import numpy as np
import matplotlib.pyplot as plt
index= 0
misclassifiedIndexes = []
for label, predict in zip(y_test, predictions):
    if label != predict:
        misclassifiedIndexes.append(index)
        index +=1
```

- Plotting Misclassfied Images with labels

In [ ]:

```
# plotting misclassified labels with known labels
plt.figure(figsize=(20,4))
for plotIndex, badIndex in enumerate(misclassifiedIndexes[0:10]):
    plt.subplot(1, 10, plotIndex + 1)
    plt.imshow(np.reshape(x_test[badIndex], (8,8)), cmap=plt.cm.gray)
```

```
    plt.title("Predicted: {}, Actual:{}". format(predictions[badIndex],y_test[labe
```

Decision Tree Classifiers

what are N_Estimators?\ n_estimators : This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes your code slower

Hands On Practice

In [ ]:

```
# load sample datasets
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
df= sns.load_dataset('iris')
df.head()
```

In [ ]:

```
# Defining/selecting dataset
x= df.iloc[: , :-1]
y= df.iloc[: , -1:]
```

- Applying and fitting the Model

In [ ]:

```
# Applying and fitting the Model
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100)
model.fit(x,y)
model.predict([[5,4,2,6]])
```

- splitting data set to find the accuracy of the model

In [ ]:

```
# splitting data set to find the accuracy of the model
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)
predictions= model.predict(x_test)
predictions
```

- accuracy test

In [ ]:

```
# accuracy test
score= model.score(x_test, y_test)
print ('The accuracy score is', score)
```

- importing and applying matrix

In [ ]:

```
from sklearn import metrics
print("Accuracy:", metrics.accuracy_score(y_test, predictions))
# cm Matrix
```

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, predictions)
```

- plotting

In [ ]:
```
import seaborn as sns
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt='.3f', linewidth=.5, square=True, cmap='Spectral')
plt.ylabel('Actual output');
plt.xlabel('Predicted Output');
all_sample_title='Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size=15);
```

- Self work

In [ ]:
```
# load sample datasets
import pandas as pd
import seaborn as sns
import numpy as np
df= sns.load_dataset('iris')
df.head()
```

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

x= df.iloc[: , :-1]
y= df.iloc[: , -1:]
```

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
model = DecisionTreeClassifier().fit(x,y)
plot_tree(model, filled=True)
plt.title('Decision treee trained modekl for iris data')
```

Predictions from trained model

In [ ]:
```
model.predict([[5,4,2,6]])
```

In [ ]:
```
model.predict([[7,7,7,6]])
```

saving the Tree in png/pdf/tiff

In [ ]:
```
# how to save this plot in png, hd , tiff hd qualities
# plt.savefig('decisiontree.png', dpi=300)
plt.savefig('tiff_compressed.tiff', dpi=600, format='tiff',
            facecolor='white', edgecolor='none',
            pil_kwargs={'compression':'tiff_lzw'})
plt.show()
# for help in saving
help(plt.savefig)
```

Checking the model accuracy (20/80)

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,  y_test= train_test_split(x,y, test_size=0.2)
predictions= model.predict(x_test)
predictions
```

In [ ]:
```
score= model.score(x_test, y_test)
print ('The accuracy score is', score)
```

In [ ]:
```
from sklearn import metrics
print("Accuracy:", metrics.accuracy_score(y_test, predictions))
```

In [ ]:
```
import seaborn as sns
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt='.3f', linewidth=.5, square=True, cmap='Spectral')
plt.ylabel('Actual output');
plt.xlabel('Predicted Output');
all_sample_title='Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size=15);
```

Checking the model accuracy (30/70)

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,  y_test= train_test_split(x,y, test_size=0.3)
predictions= model.predict(x_test)
predictions
```

In [ ]:
```
score= model.score(x_test, y_test)
print ('The accuracy score is', score)
from sklearn import metrics
print("Accuracy:", metrics.accuracy_score(y_test, predictions))
```

In [ ]:
```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, predictions)
import seaborn as sns
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt='.3f', linewidth=.5, square=True, cmap='Spectral')
plt.ylabel('Actual output');
plt.xlabel('Predicted Output');
all_sample_title='Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size=15);
```

Checking the model accuracy (10/90)

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,  y_test= train_test_split(x,y, test_size=0.1)
predictions= model.predict(x_test)
predictions
```

In [ ]:
```
score= model.score(x_test, y_test)
print ('The accuracy score is', score)
from sklearn import metrics
print("Accuracy:", metrics.accuracy_score(y_test, predictions))
```

In [ ]:
```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, predictions)
import seaborn as sns
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt='.3f', linewidth=.5, square=True, cmap='Spectral')
plt.ylabel('Actual output');
plt.xlabel('Predicted Output');
all_sample_title='Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size=15);
```

Naive Bayes

Basics concepts of Naive Bayes algorithm

- Naive Bayes is a classification algorithm that works based on the Bayes theorem.
- Bayes theorem is used to find the probability of a hypothesis with given evidence.
  - Bayes theorem helpsus to find the probability of A, given that B occurred.
- A is the hypothesis and B is the evidence.

1. P(B|A) is the probability of B given that A is True.

2. P(A) and P(B) is the independent probabilities of A and B.

importing libraries

In [ ]:
```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# load the data sets
import seaborn as sns
phool= sns.load_dataset('iris')
phool.head()
# input and output
X= phool.iloc[:,:-1]
y= phool.iloc[ : , -1:]
```

- Training and fitting the model

In [ ]:
```
# training and fitting the model
from sklearn.naive_bayes import GaussianNB
model=GaussianNB().fit(X,y)
model
```

- Training to find the Accuracy

In [ ]:
```
# train test split and testing the accuracy
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_s
```

In [ ]:
```
# training the model on training set
from sklearn.naive_bayes import GaussianNB
model = GaussianNB().fit(X_train, y_train)
#making prediction on the testing sets
y_pred= model.predict(X_test)
y_pred
```

- Applying Metrics

In [ ]:
```
from sklearn import metrics
score= metrics.accuracy_score(y_test, y_pred)
print('Gaussian Naive Bayes model accuracy(in %):', metrics.accuracy_score(y_test,
```

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

- Plotting

In [ ]:
```
import seaborn as sns
plt.figure(figsize=(12,12))
sns.heatmap(cm, annot=True, fmt='.3f', linewidths=.5, square=True, cmap='Spectral'
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Gaussian Naive Bayes model accuracy(in %): {0}'.format(score*1
plt.title(all_sample_title, size=15)
```

- **SVM**

- Difference b/w Accuracy, Precision and Recall score
- **Precision**
- is the ratio of correctly predicted positive observations to the total predicted positive observations.
- **Recall**
- Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes
- **Accuracy**
- accuracy is the fraction of predictions our model got right. we can say that accuracy:
- Accuracy = Number of correct predictions Total number of predictions.

- Equation for optimized hyperplane in svm\ $\mathbf{W^TX=0}$

Hands on practice on scikit learn library built in datasets

In [ ]:
```
# import scikit-learn datasets library
```

```
from sklearn import datasets
# load datasets
cancer = datasets.load_breast_cancer()
```

In [ ]:

```
# print the names of13 features
print('Features:', cancer.feature_names)

# print the label type of cancer ('malignant), 'bengin'
print('Labels:', cancer.target_names)
```

In [ ]:

```
# print data (feature) shapes
cancer.data.shape
```

In [ ]:

```
# print the cancer data feattures (top 5 records)
print(cancer.data[0:5])
```

In [ ]:

```
#print the cancer labels (0:malignant, 1:benign)
print(cancer.target)
```

In [ ]:

```
# import train_test_split function
from sklearn.model_selection import train_test_split

# split data into training set and test set
X_train, X_test, y_train, y_test=train_test_split(cancer.data, cancer.target, test
```

In [ ]:

```
# import svm model
from sklearn import svm

# creat a svm classifier
clf=svm.SVC(kernel='linear')  # linear kernel

# train the model using training sets
clf.fit(X_train, y_train)

# predict the response for test datasets
y_pred = clf.predict(X_test)
```

In [ ]:

```
# import sikit learn metrcs module for accuracy calculation
from sklearn import metrics
score= metrics.accuracy_score(y_test,y_pred)
# model accuracy : how often is the classifier is correct
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
```

In [ ]:

```
#model precision: what %age of positive tuples are labelled as such?
print("Precision:", metrics.precision_score(y_test,y_pred))

# model recall: what %age of positive tuples are labelled as such?
print("Recall:", metrics.recall_score(y_test,y_pred))
```

In [ ]:

```
# confusion matrix
from sklearn import metrics
```

```
cm= metrics.confusion_matrix(y_test, y_pred)
cm
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(12,12))
sns.heatmap(cm, annot=True, fmt='.3f', linewidths=.5, square=True, cmap='Spectral'
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'svm model accuracy(in %): {0}'.format(score*100)
plt.title(all_sample_title, size=15)
```

Unsupervised Learning

Importing Libraries

In [ ]:

```
# Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing data set

In [ ]:

```
flower = sns.load_dataset('iris')
print(flower.head())
```

Remove the Target Feature, we will deal with Descriptive features only.

In [ ]:

```
descriptive_features = flower.iloc[:,:-1].values
descriptive_features
```

Finding the optimum number of clusters for k-means classification

The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion. Inertia can be recognized as a measure of how internally coherent clusters are. This is what the KMeans tries to minimize with each iteration.

In [ ]:

```
#  Elbow Method
from sklearn.cluster import KMeans
within_cluster_sum_of_squares = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 1
    kmeans.fit(descriptive_features)
    within_cluster_sum_of_squares.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
```

```
plt.plot(range(1, 11), within_cluster_sum_of_squares)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('within_cluster_sum_of_squares') #within cluster sum of squares
plt.show()
```

Apply K-means Clustering

In [ ]:

```
#Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, r
y_prediction = kmeans.fit_predict(descriptive_features)
```

Visualizing the Clusters

Match Accuracy with Known Labels i.e Species in Iris Dataset

In [ ]:

```
target_feature = flower.iloc[:,-1:]
target_feature
```

Changing categorical data into Numeric Data

Measuring Accuracy

In [ ]:

```
# Measuring score
from sklearn.metrics import accuracy_score
score = accuracy_score(target_feature,y_prediction)
print('Accuracy Score of K-means Classification is:', score)
```

Confusion Matrix

In [ ]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(target_feature,y_prediction)
cm
```

In [ ]:

```
sns.set_style(style='whitegrid')
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot = True,)
plt.ylabel = 'Actual Output'
plt.xlabel = 'Predicted Output'
cm_title = 'Accuracy score : {0}'.format(score)
plt.title(cm_title)
```

K mediods

In [ ]:

```
pip install scikit-learn-extra
```

In [ ]:

```
#Applying kmeans to the dataset / Creating the kmeans classifier
from sklearn_extra.cluster import KMedoids
k_medions = KMedoids(n_clusters=3, metric='euclidean', method='alternate', init='h
y_prediction = k_medions.fit_predict(descriptive_features)
```

In [ ]:

```
y_prediction
```

Visualizing the Clusters

Sepal Features

In [ ]:

```
#Visualising the clusters
plt.scatter(descriptive_features[y_prediction == 0, 0], descriptive_features[y_pre
plt.scatter(descriptive_features[y_prediction == 1, 0], descriptive_features[y_pre
plt.scatter(descriptive_features[y_prediction == 2, 0], descriptive_features[y_pre

#Plotting the centroids of the clusters
plt.scatter(k_medions.cluster_centers_[:, 0], k_medions.cluster_centers_[:,1], mar

plt.legend()
```

Plotting on the basis of Petal Features

In [ ]:

```
#Visualising the clusters
plt.scatter(descriptive_features[y_prediction == 0, 2], descriptive_features[y_pre
plt.scatter(descriptive_features[y_prediction == 1, 2], descriptive_features[y_pre
plt.scatter(descriptive_features[y_prediction == 2, 2], descriptive_features[y_pre

#Plotting the centroids of the clusters
plt.scatter(k_medions.cluster_centers_[:, 2], k_medions.cluster_centers_[:,3], mar


plt.legend()
```

Match Accuracy with Known Labels i.e Species in Iris Dataset

In [ ]:

```
target_feature = flower.iloc[:,-1:]
target_feature
```

Changing categorical data into Numeric Data

In [ ]:

```
target_feature = target_feature.replace('setosa',2)
target_feature = target_feature.replace('virginica',0)
target_feature = target_feature.replace('versicolor',1)
target_feature
```

In [ ]:

```
# Measuring score
from sklearn.metrics import accuracy_score
score = accuracy_score(target_feature,y_prediction)
print('Accuracy Score of K-medoids Clustering is:', score)
```

Confusion Matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(target_feature,y_prediction)
cm
```

In [ ]:
```
sns.set_style(style='whitegrid')
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot = True,)
plt.ylabel = 'Actual Output'
plt.xlabel = 'Predicted Output'
cm_title = 'Accuracy score : {0}'.format(score)
plt.title(cm_title)
```

Dendograms

In [ ]:
```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(descriptive_features, method='ward'))
plt.axhline(y=3, color='b', linestyle='--')
```

Aglomerative Clustering

In [ ]:
```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='war
y_prediction=cluster.fit_predict(descriptive_features)
y_prediction
```

In [ ]:
```
plt.figure(figsize=(10, 7))
plt.scatter(descriptive_features[:,:-3], descriptive_features[:,1:-2], c=cluster.l

plt.show()
```

Getting Target Feature

In [ ]:
```
target_feature = flower.iloc[:,-1:]
target_feature
```

Changing categorical data into Numeric Data

In [ ]:
```
target_feature = target_feature.replace('setosa',1)
target_feature = target_feature.replace('virginica',2)
target_feature = target_feature.replace('versicolor',0)
target_feature
```

In [ ]:
```
# Measuring score
from sklearn.metrics import accuracy_score
```

```
score = accuracy_score(target_feature,y_prediction)
print('Accuracy Score of Aglomerative Clustering is:', score)
```

Confusion Matrix

In [ ]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(target_feature,y_prediction)
cm
```

In [ ]:

```
sns.set_style(style='whitegrid')
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot = True,)
plt.ylabel = 'Actual Output'
plt.xlabel = 'Predicted Output'
cm_title = 'Accuracy score : {0}'.format(score)
plt.title(cm_title)
```

DB SCAN with Data Points in Circular pattern

In [ ]:

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

Create Datapoints in a form of Circle

In [ ]:

```
np.random.seed(42)

# Function for creating datapoints in the form of a circle
def PointsInCircum(r,n=100):
    return [(math.cos(2*math.pi/n*x)*r+np.random.normal(-30,30),math.sin(2*math.pi
```

In [ ]:

```
# Creating data points in the form of a circle
df=pd.DataFrame(PointsInCircum(500,1000))
df=df.append(PointsInCircum(300,700))
df=df.append(PointsInCircum(100,300))

# Adding noise to the dataset
df=df.append([(np.random.randint(-600,600),np.random.randint(-600,600)) for i in r
```

Plotting datapoints

In [ ]:

```
plt.figure(figsize=(10,10))
plt.scatter(df[0],df[1],s=15,color='grey')
plt.title('Dataset',fontsize=20)
plt.ylabel = 'Actual Output'
plt.xlabel = 'Predicted Output'
plt.show()
```

Clustering on the basis of K-Means

In [ ]:

```
from sklearn.cluster import KMeans
```

```
k_means=KMeans(n_clusters=4,random_state=42)
k_means.fit(df[[0,1]])
df['KMeans_labels']=k_means.labels_

# Plotting resulting clusters
colors=['purple','red','blue','green']
plt.figure(figsize=(10,10))
plt.scatter(df[0],df[1],c=df['KMeans_labels'],cmap=matplotlib.colors.ListedColorma
plt.title('K-Means Clustering',fontsize=20)
plt.ylabel = 'Actual Output'
plt.xlabel = 'Predicted Output'
plt.show()
```

Clustering Based on Aglomerative Clustering

In [ ]:

```
from sklearn.cluster import AgglomerativeClustering
model = AgglomerativeClustering(n_clusters=4, affinity='euclidean')
model.fit(df[[0,1]])
df['HR_labels']=model.labels_

# Plotting resulting clusters
plt.figure(figsize=(10,10))
plt.scatter(df[0],df[1],c=df['HR_labels'],cmap=matplotlib.colors.ListedColormap(cc
plt.title('Hierarchical Clustering',fontsize=20)
plt.show()
```

DB Scan method

In [ ]:

```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(df[[0,1]])
distances, indices = nbrs.kneighbors(df[[0,1]])

from sklearn.cluster import DBSCAN
dbscan_opt=DBSCAN(eps=30,min_samples=6)
dbscan_opt.fit(df[[0,1]])

df['DBSCAN_opt_labels']=dbscan_opt.labels_
df['DBSCAN_opt_labels'].value_counts()
# Plotting the resulting clusters
plt.figure(figsize=(10,10))
plt.scatter(df[0],df[1],c=df['DBSCAN_opt_labels'],cmap=matplotlib.colors.ListedCol
plt.title('DBSCAN Clustering',fontsize=20)
plt.show()
```

Neural Networks

Data set for digit number recognition

The data set is made of 8x8 images of digits. We start with loading the dataset.

In [ ]:

```
import matplotlib.pyplot as plt

# Import datasets, classifiers and performance metrics
from sklearn import datasets, metrics
from sklearn.neural_network import MLPClassifier
```

```
from sklearn import tree

# The digits dataset
digits = datasets.load_digits()
```

Now, let's have a look at some of the first images, stored in the images attribute of the dataset.

- If we were working from image files, we could load them using matplotlib.pyplot.imread. Note that each image must have the same size.
- For these images, we know which digit they represent: it is given in the 'target' of the dataset.

In [ ]:

```
images_and_labels = list(zip(digits.images, digits.target))
plt.figure(figsize=(10,7))
for index, (image, label) in enumerate(images_and_labels[:32]):
    plt.subplot(4, 8, index + 1)
    plt.axis('off')
    plt.imshow(image,cmap=plt.cm.gray_r)
    plt.title('Training: %i' % label)
plt.show()
```

In [ ]:

```
len(images_and_labels)
```

We need to do some preprocessing!

In [ ]:

```
# To apply a classifier on this data, we need to flatten the image, to
# turn the data in a (samples, feature) matrix:
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
```

Next, let's see what our classification algorithms does:

In [ ]:

```
# Create a classifier:

classifier = MLPClassifier(hidden_layer_sizes=(100,))

# We learn the digits on the first half of the digits
classifier.fit(data, digits.target)

# Now predict the value of the digit on the second half:
expected = digits.target
predicted = classifier.predict(data)
```

Lets show the predictions:

In [ ]:

```
classifier
```

In [ ]:

```
n_samples,predicted.size
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
index = 0
misclassified_indicies = []
```

```
for label, predict in zip(expected, predicted):
 if label != predict:
  misclassified_indicies.append(index)
  index +=1
number_of_missclassified_data =len(misclassified_indicies)
print('indicies of missclssified data', misclassified_indicies)
print('number of missclassified pictures :',number_of_missclassified_data )
```

In [ ]:

```
images_and_predictions = list(zip(digits.images[9*(n_samples) // 10:], predicted))
for index, (image, prediction) in enumerate(images_and_predictions[:10]):
    plt.subplot(3, 4, index+1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r)
    plt.title('Prediction: %i' % prediction)

plt.show()
```

Evaluation metrics can be used to compare your classifications:

In [ ]:

```
print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))
```

Splitting into Training and Testing Data

Importing Digits data set from SKLearn

In [ ]:

```
from sklearn.datasets import load_digits

new_digit = load_digits()
type(new_digit)
```

In [ ]:

```
# View Digits data
new_digit.data.shape
```

Data Contains 1797 pictures of size 8x8 i.e 64

Setting Descriptive and Target features (data = descriptive , target = target in this data set)

In [ ]:

```
descriptive_features = new_digit.data
target_feature = new_digit.target

descriptive_features.shape, target_feature.shape
```

Splitting data

In [ ]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(descriptive_features, target_f
```

In [ ]:

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Training Model - Neural Network

10 Hidden Layers

In [ ]:

```
# Create a classifier:

classifier_10 = MLPClassifier(hidden_layer_sizes=(10,))

# We learn the digits on the first half of the digits
classifier_10.fit(x_train, y_train)
```

Predicting test data

In [ ]:

```
# Predicting the Test set results
y_pred = classifier_10.predict(x_test)
```

Accuracy test

In [ ]:

```
score_10 = classifier_10.score(x_test,y_test)
print('acuracy of model is :' , score_10)
```

Training Model - Neural Network - Hiidden Layers = 100

In [ ]:

```
# Create a classifier:

classifier_100 = MLPClassifier(hidden_layer_sizes=(100,))

# We learn the digits on the first half of the digits
classifier_100.fit(x_train, y_train)
```

Predicting test data

In [ ]:

```
# Predicting the Test set results
y_pred = classifier_100.predict(x_test)
```

Accuracy test

In [ ]:

```
score_100 = classifier.score(x_test,y_test)
print('acuracy of model is :' , score_100)
```

Training Model - Neural Network - Hiidden Layers = 1000

In [ ]:

```
# Create a classifier:

classifier_1000 = MLPClassifier(hidden_layer_sizes=(1000,))

# We learn the digits on the first half of the digits
classifier_1000.fit(x_train, y_train)
```

Predicting test data

In [ ]:

```
# Predicting the Test set results
y_pred = classifier_1000.predict(x_test)
```

Accuracy test

In [ ]:

```
score_1000 = classifier_1000.score(x_test,y_test)
print('acuracy of model is :' , score_1000)
```

Training Model - Neural Network - Hiidden Layers = 10000

In [ ]:

```
# Create a classifier:

classifier_10000 = MLPClassifier(hidden_layer_sizes=(10000,))

# We learn the digits on the first half of the digits
classifier_10000.fit(x_train, y_train)
```

Predicting test data

In [ ]:

```
# Predicting the Test set results
y_pred = classifier_10000.predict(x_test)
```

Accuracy test

In [ ]:

```
score_10000 = classifier_10000.score(x_test,y_test)
print('acuracy of model is :' , score_10000)
```

Training Model - Neural Network - Hiidden Layers = 100000

In [ ]:

```
# Create a classifier:

classifier_100000 = MLPClassifier(hidden_layer_sizes=(100000,))

# We learn the digits on the first half of the digits
classifier_100000.fit(x_train, y_train)
```

Predicting test data

In [ ]:
```
# Predicting the Test set results
y_pred = classifier_100000.predict(x_test)
```

Accuracy test

In [ ]:
```
score_100000 = classifier_100000.score(x_test,y_test)
print('acuracy of model is :' , score_100000)
```

In [ ]:
```
x =[10,100,1000,10000,100000]
y =[score_10,score_100,score_1000,score_10000,score_100000]
```

In [ ]:
```
import seaborn as sns
plt.figure(figsize=(14,6))
a =sns.lineplot(x=x, y=y, ci=None)
a.set(xlabel='Number of Hidden Layers', ylabel='Accuracy')
plt.show()
```

Confusion Matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm
```

Creating HEATMAP to understand Confusion matrix

In [ ]:
```
sns.set_style(style='whitegrid')
plt.figure(figsize=(12,10))
sns.heatmap(cm, annot = True,)
plt.ylabel = 'Actual Output'
plt.xlabel = 'Predicted Output'
cm_title = 'Accuracy score : {0}'.format(score_1000)
plt.title(cm_title)
```

Getting Misclassified Labels

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
index = 0
misclassifiedIndexes = []
for label, predict in zip(y_test, y_pred):
 if label != predict:
  misclassifiedIndexes.append(index)
  index +=1
```

In [ ]:
```
index = 0
misclassified_indicies = []
for label, predict in zip(y_test,y_pred):
```

```
    if label != predict:
        #print(index,label,predict) #Debugging
        misclassified_indicies.append(index)
    index = index +1
number_of_missclassified_data =len(misclassified_indicies)
print('indicies of missclssified data', misclassified_indicies)
print('number of missclassified pictures :',number_of_missclassified_data )
```

Plotting Missclassified Lables with Real Lables

In [ ]:

```
sns.set_style(style='whitegrid')
plt.figure(figsize=(20,4))
for plotindex, badindex in enumerate(misclassified_indicies):
    plt.subplot(1,number_of_missclassified_data, plotindex+1)
    plt.imshow(np.reshape(x_test[badindex],(8,8)), cmap=plt.cm.gray)
    plt.title('x:{}, y:{}'.format(y_test[badindex],y_pred[badindex]), fontsize =20
```