FAZAIA BILQUIS COLLEGE OF EDUCATION FOR WOMEN PAF NUR KHAN RAWALPINDI

Lab Manual

Data Science

Submitted By:

Areeba Shahzadi(44)

BSCS 7th A

Submitted To:

Ms. Sidra Kayani

Date of Submission:

17-01-2024

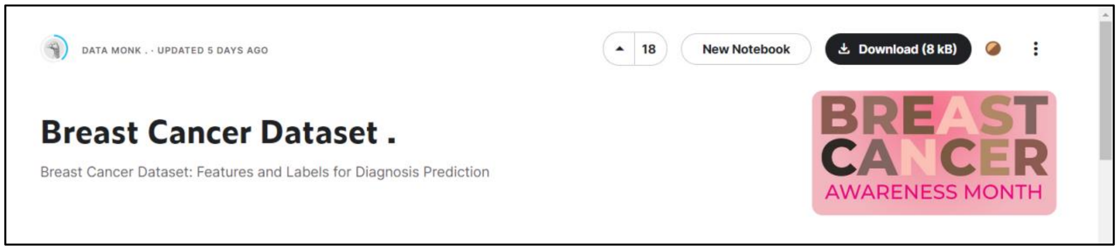
Table of Content

LAB #1

Task1: Dataset take which repository?

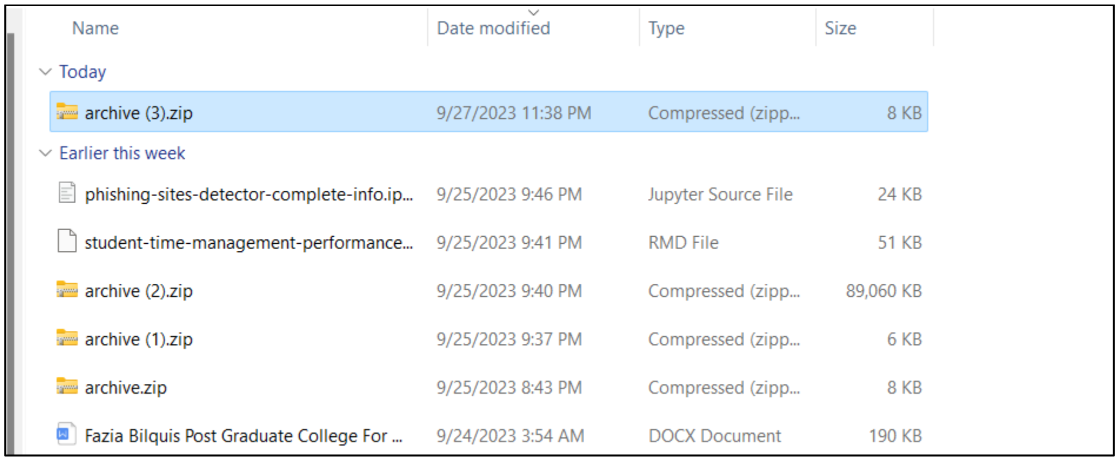
- **Data.gov** https://data.gov/
- **Kaggle** https://www.kaggle.com/datasets
- **UCI Machine Learning Repository** https://archive.ics.uci.edu/datasets

Task2: Dataset download in kaggle?



Breast Cancer Dataset .

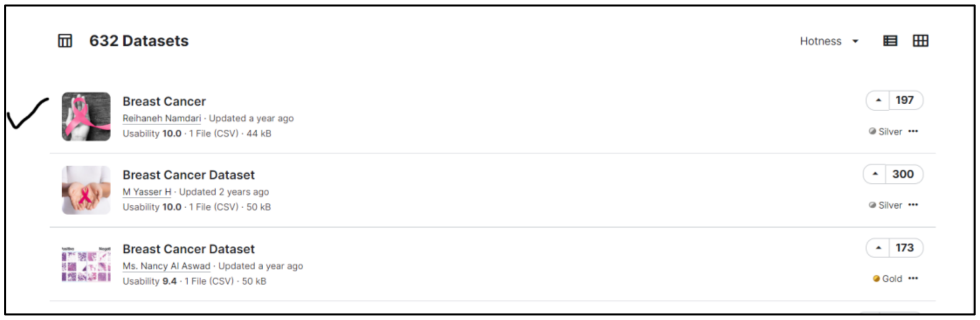
Breast Cancer Dataset: Features and Labels for Diagnosis Prediction



Name	Date modified	Type	Size
Today			
archive (3).zip	9/27/2023 11:38 PM	Compressed (zipp...	8 KB
Earlier this week			
phishing-sites-detector-complete-info.ip...	9/25/2023 9:46 PM	Jupyter Source File	24 KB
student-time-management-performance...	9/25/2023 9:41 PM	RMD File	51 KB
archive (2).zip	9/25/2023 9:40 PM	Compressed (zipp...	89,060 KB
archive (1).zip	9/25/2023 9:37 PM	Compressed (zipp...	6 KB
archive.zip	9/25/2023 8:43 PM	Compressed (zipp...	8 KB
Fazia Bilquis Post Graduate College For ...	9/24/2023 3:54 AM	DOCX Document	190 KB

Task3: Dataset and upload in colab?

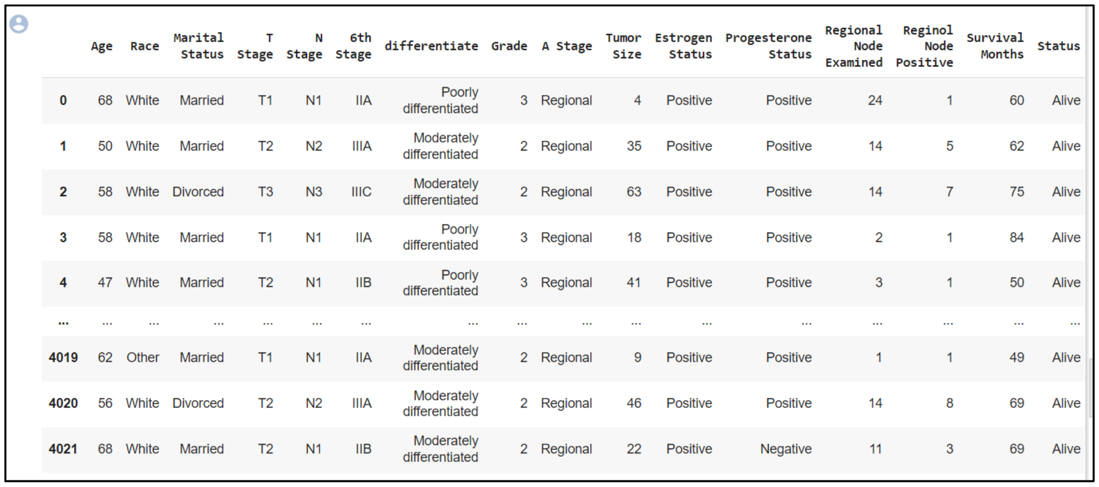
Breast cancer dataset



632 Datasets

Hotness

Breast Cancer Reihaneh Namdari · Updated a year ago Usability 10.0 · 1 File (CSV) · 44 kB	197	Silver
Breast Cancer Dataset M Yasser H · Updated 2 years ago Usability 10.0 · 1 File (CSV) · 50 kB	300	Silver
Breast Cancer Dataset Ms. Nancy Al Aswad · Updated a year ago Usability 9.4 · 1 File (CSV) · 50 kB	173	Gold



	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estrogen Status	Progesterone Status	Regional Node Examined	Reginol Node Positive	Survival Months	Status
0	68	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	4	Positive	Positive	24	1	60	Alive
1	50	White	Married	T2	N2	IIIA	Moderately differentiated	2	Regional	35	Positive	Positive	14	5	62	Alive
2	58	White	Divorced	T3	N3	IIIC	Moderately differentiated	2	Regional	63	Positive	Positive	14	7	75	Alive
3	58	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	18	Positive	Positive	2	1	84	Alive
4	47	White	Married	T2	N1	IIB	Poorly differentiated	3	Regional	41	Positive	Positive	3	1	50	Alive
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4019	62	Other	Married	T1	N1	IIA	Moderately differentiated	2	Regional	9	Positive	Positive	1	1	49	Alive
4020	56	White	Divorced	T2	N2	IIIA	Moderately differentiated	2	Regional	46	Positive	Positive	14	8	69	Alive
4021	68	White	Married	T2	N1	IIB	Moderately differentiated	2	Regional	22	Positive	Negative	11	3	69	Alive

Task 4: Basic python libraries?

1. NumPy: NumPy is the fundamental package for scientific computing with Python. It provides support for arrays and matrices, along with mathematical functions to operate on these arrays efficiently.

Syntax import numpy as np

2. Pandas: Pandas is a powerful library for data manipulation and analysis. It introduces data structures like DataFrames and Series, making it easy to work with structured data.

Syntax import pandas as pd

3. Matplotlib: Matplotlib is a popular library for creating static, animated, and interactive visualizations in Python.

Syntax import matplotlib.pyplot as plt

4. Seaborn: Seaborn is built on top of Matplotlib and provides a high-level interface for creating informative and attractive statistical graphics.

Syntax import seaborn as sns

5. Scikit-Learn: Scikit-Learn is a machine learning library that offers a wide range of machine learning algorithms for classification, regression, clustering, and more.

Syntax from sklearn import <module>

6. SciPy: SciPy is an open-source library used for mathematics, science, and engineering. It builds on NumPy and provides additional functionality for optimization, integration, interpolation, and more.

Syntax import scipy

7. Statsmodels: Statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models.

Syntax import statsmodels.api as sm

8. Random: The random module is part of Python's standard library and is used for generating random numbers, selecting random items from sequences, and more.

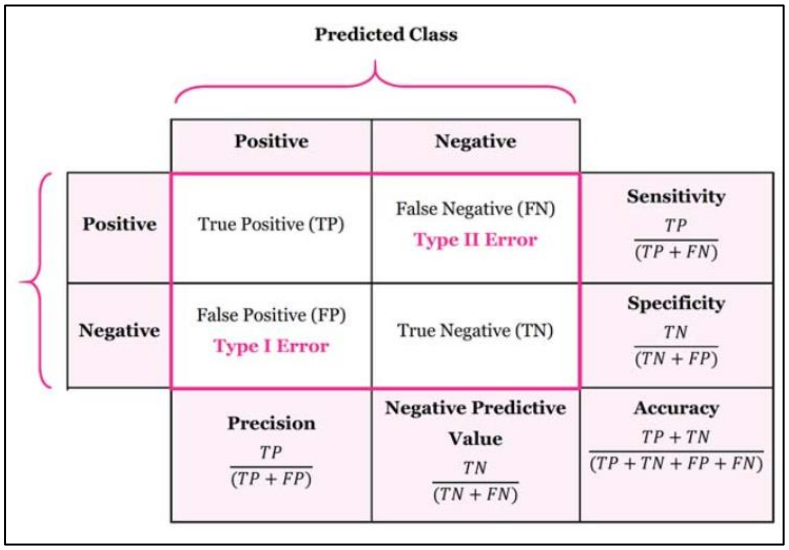
Syntax import random

9. Requests: The requests library allows you to make HTTP requests in Python. It's commonly used for web scraping and interacting with APIs.

Syntax import requests

Task 5: Why Confusion Matrix used?

A confusion matrix represents the prediction summary in matrix form. It shows how many prediction are correct and incorrect per class. It helps in understanding the classes that are being confused by model as other class.

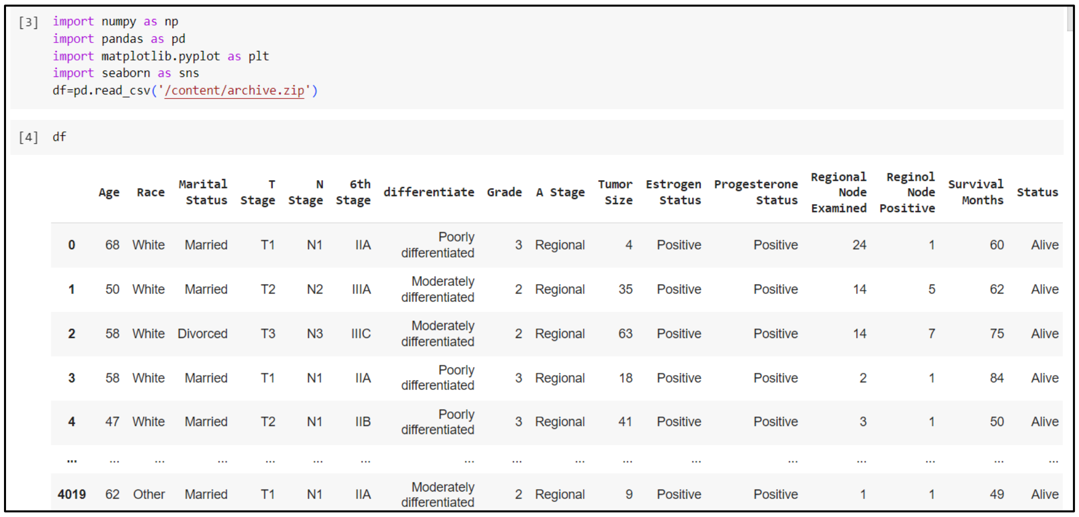


Precision	Precision is a metric used to evaluate the performance of a classification model. The ability of a classification model to identify only the relevant data points.	$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives}+\text{False Positives}}$
Accuracy	Accuracy is the number of correctly predicted data points out of all the data points.	$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$
Recall	The ability of a model to find all the relevant cases within a data set.	$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives}+\text{False Negatives}}$
Sensitivity	Sensitivity is a measure of how well a machine learning model can detect positive instances. It is also known as the true positive rate (TPR) or recall.	$\text{Sensitivity (Recall)} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)}+\text{False Negatives (FN)}}$
Specificity	Specificity is the ratio of true negatives to all negative outcomes.	$\text{Specificity} = \frac{\text{True Negatives (TN)}}{\text{True Negatives (TN)}+\text{False Positives (FP)}}$
F1 score	F1 score is a machine learning evaluation metric that measures a model's accuracy. It combines the precision and recall scores of a model.	$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$

Lab#2

Task#1What do you About Data set?

The term data set refers to a file that contains one or more records. The record is the basic unit of information used by a program running on z/OS. Any named group of records is called a data set.



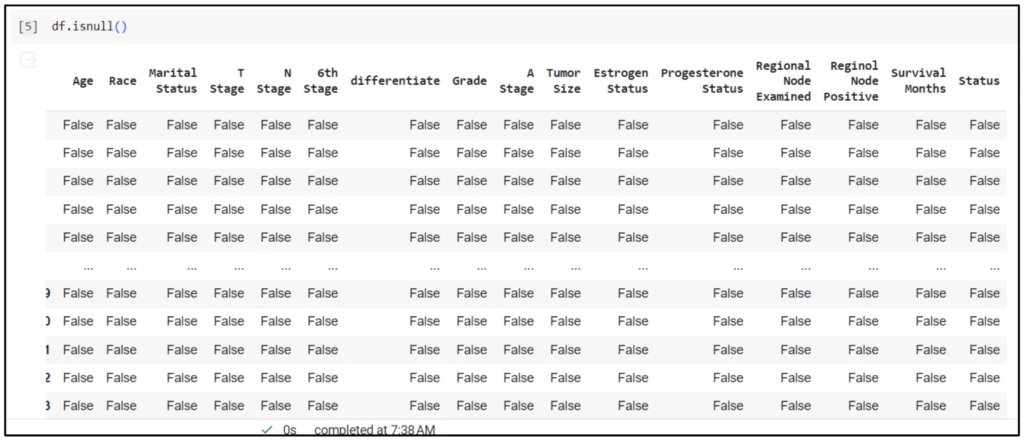
```
[3] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    df=pd.read_csv('/content/archive.zip')
```

```
[4] df
```

	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estrogen Status	Progesterone Status	Regional Node Examined	Reginol Node Positive	Survival Months	Status
0	68	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	4	Positive	Positive	24	1	60	Alive
1	50	White	Married	T2	N2	IIIA	Moderately differentiated	2	Regional	35	Positive	Positive	14	5	62	Alive
2	58	White	Divorced	T3	N3	IIIC	Moderately differentiated	2	Regional	63	Positive	Positive	14	7	75	Alive
3	58	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	18	Positive	Positive	2	1	84	Alive
4	47	White	Married	T2	N1	IIB	Poorly differentiated	3	Regional	41	Positive	Positive	3	1	50	Alive
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4019	62	Other	Married	T1	N1	IIA	Moderately differentiated	2	Regional	9	Positive	Positive	1	1	49	Alive

6. Is null function():

To check column value is null or not.



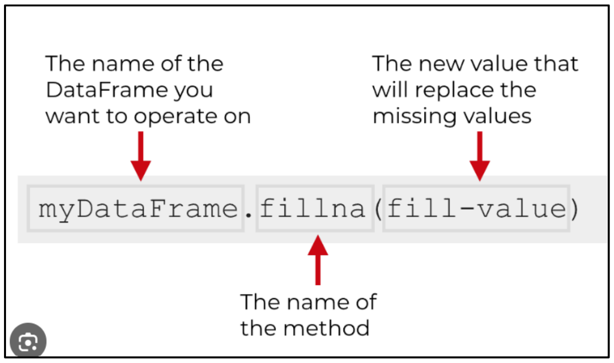
```
[5] df.isnull()
```

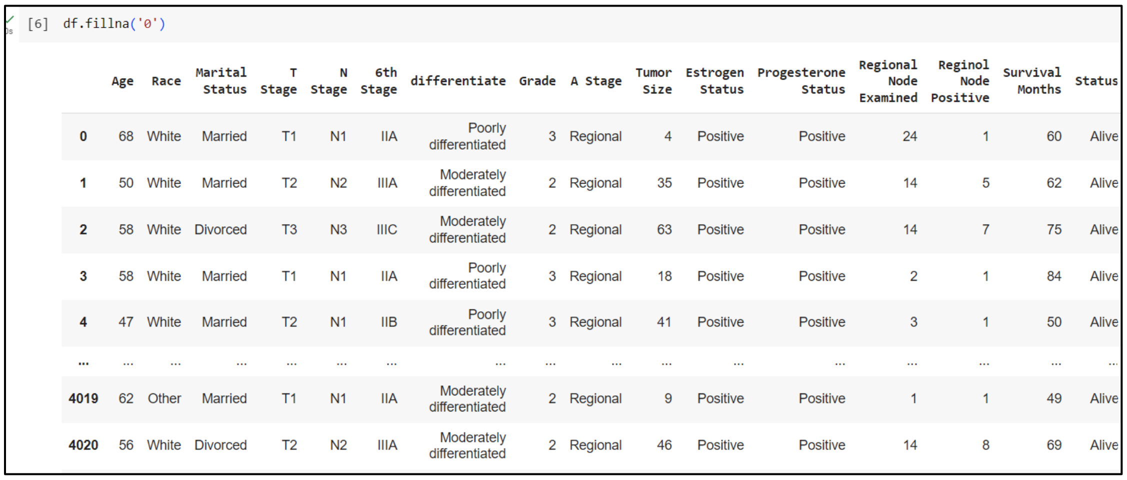
	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estrogen Status	Progesterone Status	Regional Node Examined	Reginol Node Positive	Survival Months	Status
	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

✓ 0s completed at 7:38 AM

7. Fillna function()

Those whose value is null so fill this column.


The name of the DataFrame you want to operate on
The new value that will replace the missing values
myDataFrame.fillna(fill-value)
The name of the method

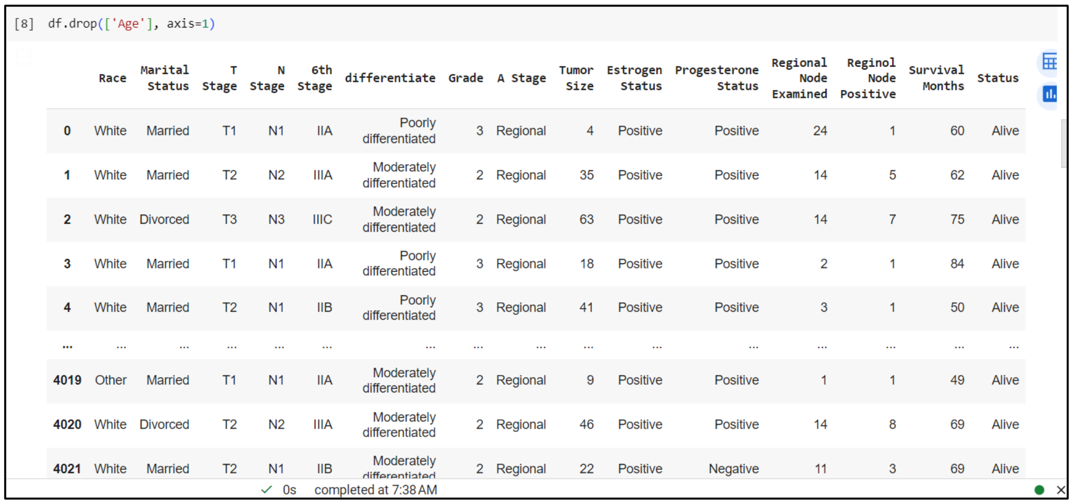


```
[6] df.fillna('0')
```

	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estrogen Status	Progesterone Status	Regional Node Examined	Reginol Node Positive	Survival Months	Status
0	68	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	4	Positive	Positive	24	1	60	Alive
1	50	White	Married	T2	N2	IIIA	Moderately differentiated	2	Regional	35	Positive	Positive	14	5	62	Alive
2	58	White	Divorced	T3	N3	IIIC	Moderately differentiated	2	Regional	63	Positive	Positive	14	7	75	Alive
3	58	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	18	Positive	Positive	2	1	84	Alive
4	47	White	Married	T2	N1	IIB	Poorly differentiated	3	Regional	41	Positive	Positive	3	1	50	Alive
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4019	62	Other	Married	T1	N1	IIA	Moderately differentiated	2	Regional	9	Positive	Positive	1	1	49	Alive
4020	56	White	Divorced	T2	N2	IIIA	Moderately differentiated	2	Regional	46	Positive	Positive	14	8	69	Alive

8. Drop column

Drop a column if you want



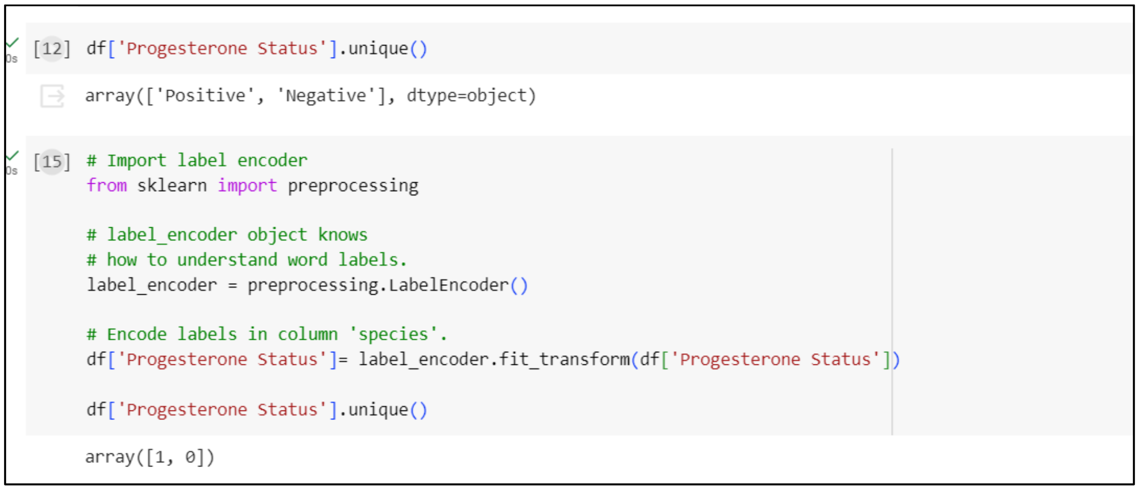
```
[8]  df.drop(['Age'], axis=1)
```

	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estrogen Status	Progesterone Status	Regional Node Examined	Reginol Node Positive	Survival Months	Status
0	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	4	Positive	Positive	24	1	60	Alive
1	White	Married	T2	N2	IIIA	Moderately differentiated	2	Regional	35	Positive	Positive	14	5	62	Alive
2	White	Divorced	T3	N3	IIIC	Moderately differentiated	2	Regional	63	Positive	Positive	14	7	75	Alive
3	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	18	Positive	Positive	2	1	84	Alive
4	White	Married	T2	N1	IIB	Poorly differentiated	3	Regional	41	Positive	Positive	3	1	50	Alive
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4019	Other	Married	T1	N1	IIA	Moderately differentiated	2	Regional	9	Positive	Positive	1	1	49	Alive
4020	White	Divorced	T2	N2	IIIA	Moderately differentiated	2	Regional	46	Positive	Positive	14	8	69	Alive
4021	White	Married	T2	N1	IIB	Moderately differentiated	2	Regional	22	Positive	Negative	11	3	69	Alive

✓ 0s completed at 7:38 AM

9. Import label encoder function

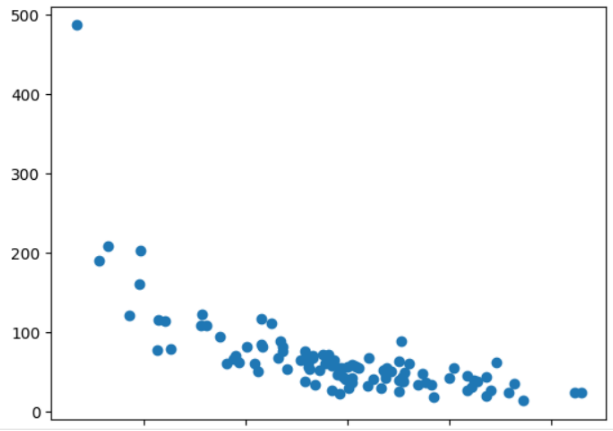
LabelEncoder is a function that is used to encode categorical data into numerical values. It is a part of the scikit-learn library, which is a popular machine learning library in Python. The LabelEncoder function assigns a unique numerical value to each category in a categorical variable.



```
[12] df['Progesterone Status'].unique()

     array(['Positive', 'Negative'], dtype=object)

[15] # Import label encoder
     from sklearn import preprocessing

     # label_encoder object knows
     # how to understand word labels.
     label_encoder = preprocessing.LabelEncoder()

     # Encode labels in column 'species'.
     df['Progesterone Status']= label_encoder.fit_transform(df['Progesterone Status'])

     df['Progesterone Status'].unique()

     array([1, 0])
```

10. Training and Testing

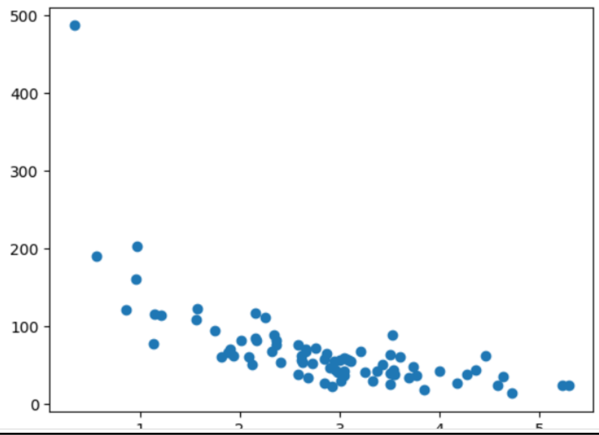
```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

plt.scatter(x, y)
plt.show()
```

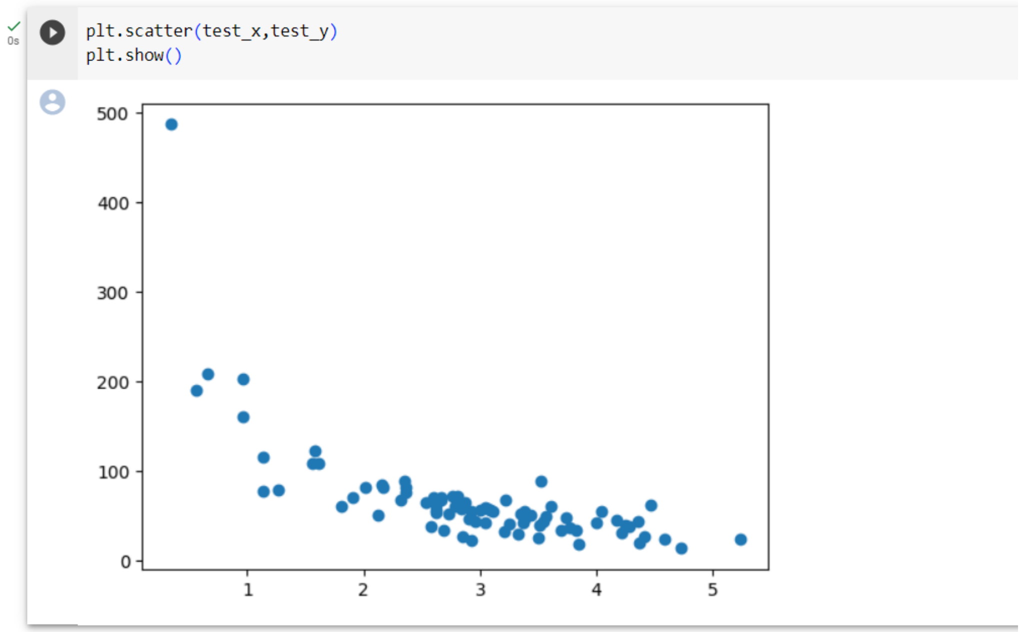


✓ 0s completed at 7:38 AM

```
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]
```

```
plt.scatter(train_x, train_y)
plt.show()
```





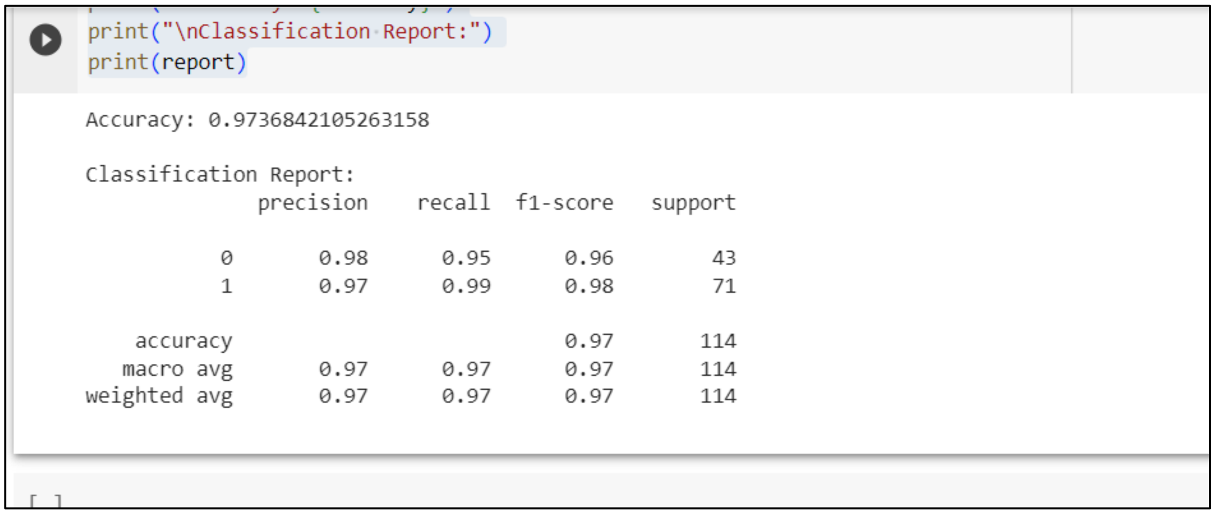
```
plt.scatter(test_x,test_y)
plt.show()
```

LAB 3

Task 1: Apply any algo or technique on your Dataset And justify them.

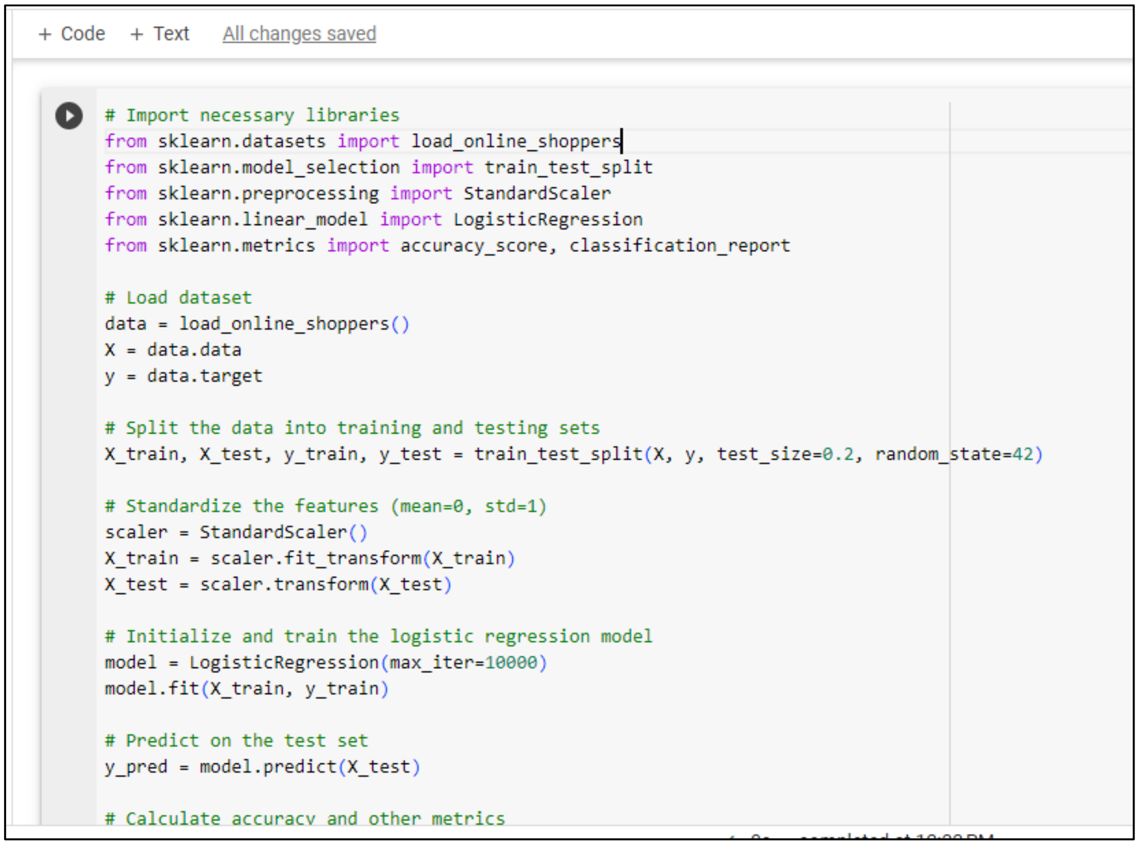
```
# Import necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
# Load the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Standardize the features (mean=0, std=1)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Initialize and train the logistic regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)
# Predict on the test set
y_pred = model.predict(X_test)
# Calculate accuracy and other metrics
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("\nClassification Report:")
print(report)
```

Output:

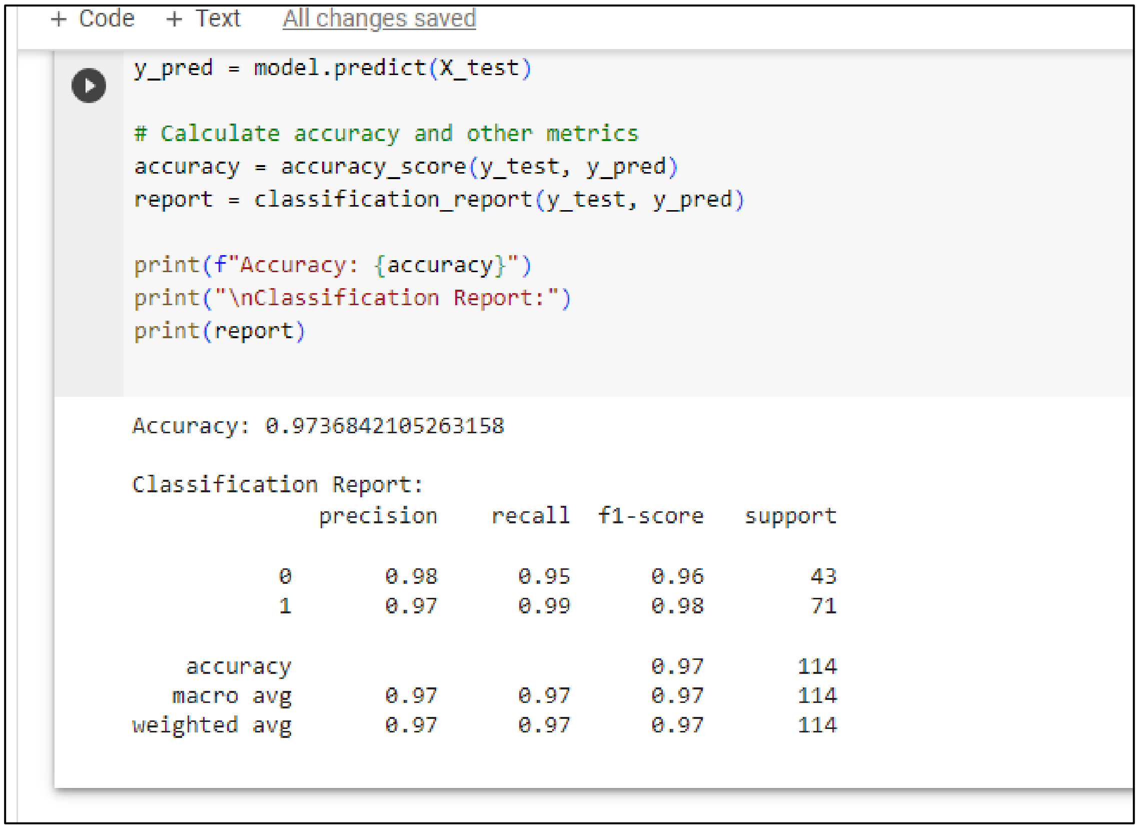


```
print("\nClassification Report:")
print(report)
```

```
Accuracy: 0.9736842105263158

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```



```
# Import necessary libraries
from sklearn.datasets import load_online_shoppers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
data = load_online_shoppers()
X = data.data
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features (mean=0, std=1)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the logistic regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate accuracy and other metrics
```



```python
y_pred = model.predict(X_test)

# Calculate accuracy and other metrics
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("\nClassification Report:")
print(report)
```

```
Accuracy: 0.9736842105263158

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

Lab#4

Task#1

Deep analysis of dataset discuss the number of feature ,datatypes of the data including the full description of your data?

About Dataset

Description:

Breast cancer is the most common cancer amongst women in the world. It accounts for 25% of all cancer cases, and affected over 2.1 Million people in 2015 alone. It starts when cells in the breast begin to grow out of control. These cells usually form tumors that can be seen via X-ray or felt as lumps in the breast area.

The key challenges against it's detection is how to classify tumors into malignant (cancerous) or benign(non cancerous). We ask you to complete the analysis of classifying these tumors using machine learning (with SVMs) and the Breast Cancer Wisconsin (Diagnostic) Dataset.

Acknowledgements:

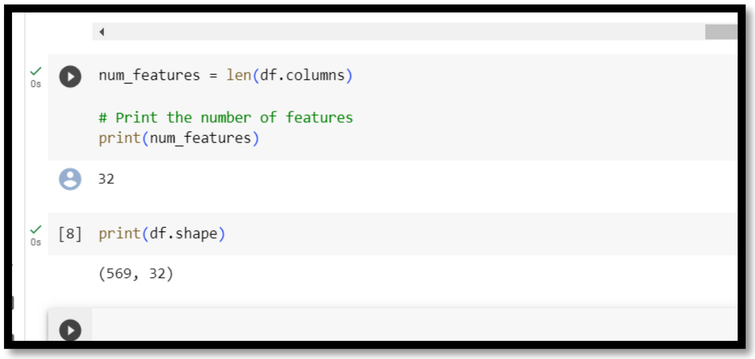
This dataset has been referred from Kaggle.

Objective:

- Understand the Dataset & cleanup (if required).
- Build classification models to predict whether the cancer type is Malignant or Benign.
- Also fine-tune the hyperparameters & compare the evaluation metrics of various classification algorithms.

Task#2

Features check?



```
num_features = len(df.columns)

# Print the number of features
print(num_features)
```

```
32
```

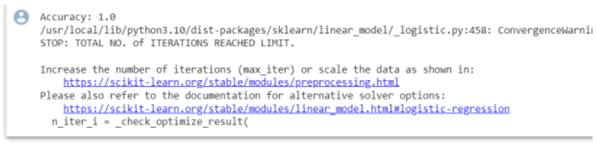
```
[8] print(df.shape)
```

```
(569, 32)
```

Task#3 Apply Logistic Regression?

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Split the data into training and test sets
X = df.drop('id', axis=1)
y = df['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Create a logistic regression model
model = LogisticRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

output:



```
Accuracy: 1.0
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Confusion matrix:

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

# Calculate F1 score
f1_score = f1_score(y_test, y_pred)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate recall
recall = recall_score(y_test, y_pred)

# Print the results
print("F1 score:", f1_score)
print("Accuracy:", accuracy)
print("Recall:", recall)
```

output:

```
F1 score: 1.0
Accuracy: 1.0
Recall: 1.0
```

Task#4 Discuss the Description of all Algorithm?

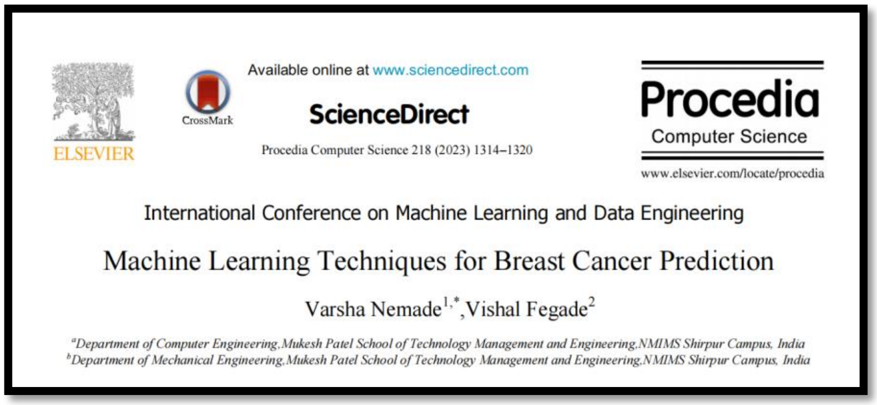
1. **Logistic Regression:** It is an extension of linear regression model and used for classification problem. It provides the probabilities for two possible outcomes. In health care field this method is useful for prediction of likelihood of disease or illness.
2. **KNN:** It uses all training data to classify new data point based on the similarity. To assign label to new data point it calculate its distance from different label point and finally it find nearest neighbour. • SVM: This classification technique not required any prior distribution knowledge for classification. It uses hyper plane to classify data points

3. **Naive Bayes:** This techniques based on Bayes theorem. It performs prediction based on probability.

4. **Decision Tree:** In this technique instances are classified using features value. For splitting it uses Gini Index, Information Gain. Leaf node indicates the label.

Task#5

Research Paper:



Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 218 (2023) 1314–1320

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Machine Learning and Data Engineering

Machine Learning Techniques for Breast Cancer Prediction

Varsha Nemade[1,*], Vishal Fegade[2]

[a]Department of Computer Engineering, Mukesh Patel School of Technology Management and Engineering, NMIMS Shirpur Campus, India
[b]Department of Mechanical Engineering, Mukesh Patel School of Technology Management and Engineering, NMIMS Shirpur Campus, India

Lab#5

Task 1:Dataset

This dataset provided for users whose they want to develop their neural network practices based on numerical dataset like this one. This dataset collected from many more clients.

Task 2:Different algorithm apply on dataset.

There are a variety of machine learning algorithms that can be used for migraine classification. Some of the most popular algorithms include:

- **Support vector machines (SVMs)**: SVMs are a type of supervised learning algorithm that can be used for both classification and regression tasks. SVMs work by finding a hyperplane that separates the data into two classes with the largest possible margin.
- **Random forests**: Random forests are an ensemble learning algorithm that combines the predictions of multiple decision trees to make a final prediction. Random forests are known for their robustness and accuracy, and they are often used for classification tasks.
- **Neural networks**: Neural networks are a type of machine learning algorithm that is inspired by the structure and function of the human brain. Neural networks can be trained to perform a variety of tasks, including classification, regression, and natural language processing.

The best algorithm to use for migraine classification will depend on your specific dataset and needs. However, the three algorithms listed above are a good starting point.

Here are some examples of studies that have used different machine learning algorithms for migraine classification:

- **Support vector machines:** One study used SVMs to classify migraine with aura (MwA) and migraine without aura (MO) patients using MRI data. The SVM classifier achieved an accuracy of 97% for MwA patients and 98% for MO patients.
- **Random forests:** Another study used random forests to classify migraine patients into different subtypes based on their symptoms. The random forest classifier achieved an accuracy of 85%.
- **Neural networks:** A third study used neural networks to classify migraine patients into different subtypes based on their EEG data. The neural network classifier achieved an accuracy of 90%.

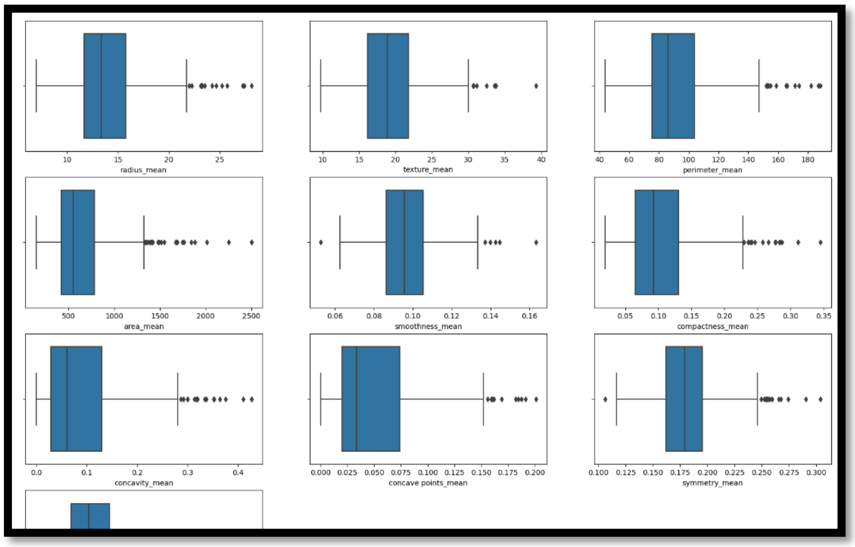
It is important to note that the performance of any machine learning algorithm will depend on the quality and quantity of the training data. Therefore, it is important to have a large and well-labeled dataset of migraine patients in order to train an effective machine learning model for migraine classification.

Task#3

Implementation the visualization method specifically according to your dataset?

```
numerical_features = ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean',
'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']

fig, axes = plt.subplots(4, 3, figsize=(20, 15))
fig.delaxes(axes[3, 1])
fig.delaxes(axes[3, 2])

for i in range(0, len(numerical_features)):
sns.boxplot(ax=axes[i // 3, i % 3], x = df[numerical_features[i]])
```

output:



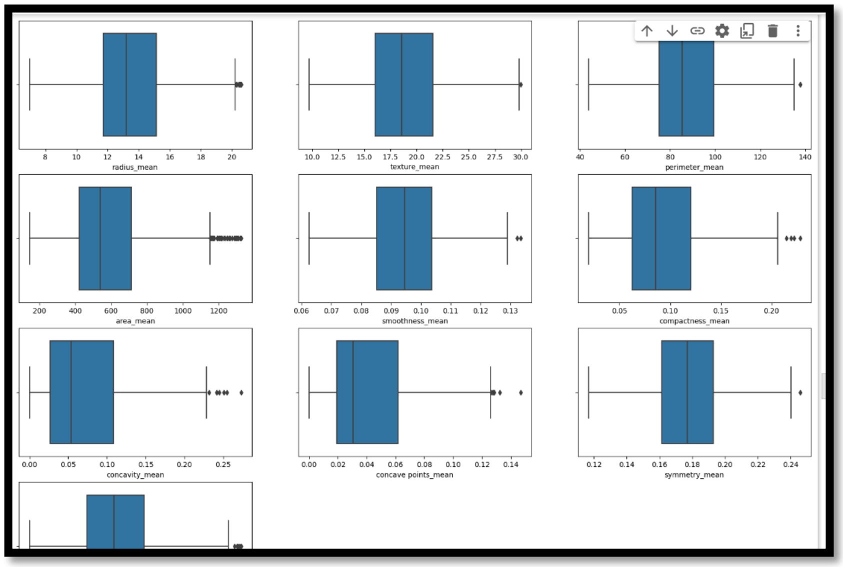
Task#4

Detect outlier?

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis = 1)]

df.shape
```

Now again visualization to check is there any outlier in dataset?

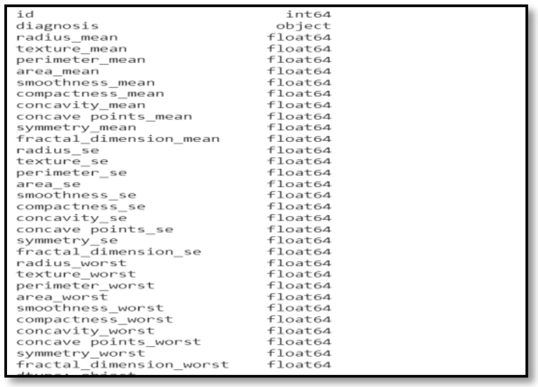
```
fig, axes = plt.subplots(4, 3, figsize=(20, 15))
fig.delaxes(axes[3, 1])
fig.delaxes(axes[3, 2])

for i in range(0, len(numerical_features)):
sns.boxplot(ax=axes[i // 3, i % 3], x = df[numerical_features[i]])
```



Task#5

What is the type of dataset numerical or categorical?

```
# Print the column details
df.dtypes
```



```
id                             int64
diagnosis                     object
radius_mean                  float64
texture_mean                 float64
perimeter_mean               float64
area_mean                    float64
smoothness_mean              float64
compactness_mean             float64
concavity_mean               float64
concave points_mean          float64
symmetry_mean                float64
fractal_dimension_mean       float64
radius_se                    float64
texture_se                   float64
perimeter_se                 float64
area_se                      float64
smoothness_se                float64
compactness_se               float64
concavity_se                 float64
concave points_se            float64
symmetry_se                  float64
fractal_dimension_se         float64
radius_worst                 float64
texture_worst                float64
perimeter_worst              float64
area_worst                   float64
smoothness_worst             float64
compactness_worst            float64
concavity_worst              float64
concave points_worst         float64
symmetry_worst               float64
fractal_dimension_worst      float64
```

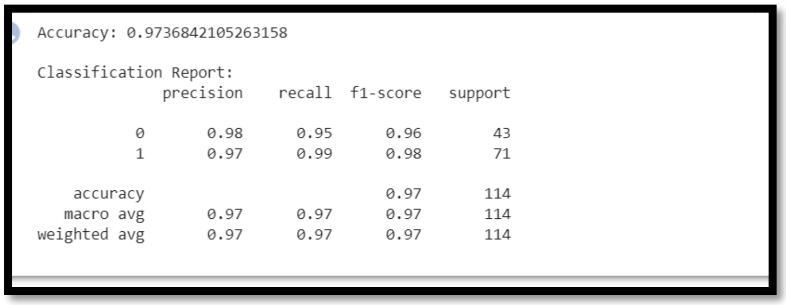
Task#6

Difference between labelled and un labelled data?

Un labelled data

```
# Import necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features (mean=0, std=1)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the logistic regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate accuracy and other metrics
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("\nClassification Report:")
print(report)
```

output:



```
Accuracy: 0.9736842105263158

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

Labelled dataset:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import accuracy_score

knn = KNeighborsRegressor(n_neighbors=10, metric = 'minkowski', p = 2)
knn.fit(x_train, y_train)

models.loc['KNN', 'train_acc'] = accuracy_score(y_pred=np.rint(knn.predict(x_train)),
y_true=y_train)
from sklearn.metrics import f1_score, accuracy_score, recall_score

# Calculate F1 score
f1_score = f1_score(y_test, y_pred)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate recall
recall = recall_score(y_test, y_pred)

# Print the results
print("F1 score:", f1_score)
print("Accuracy:", accuracy)
print("Recall:", recall)
```

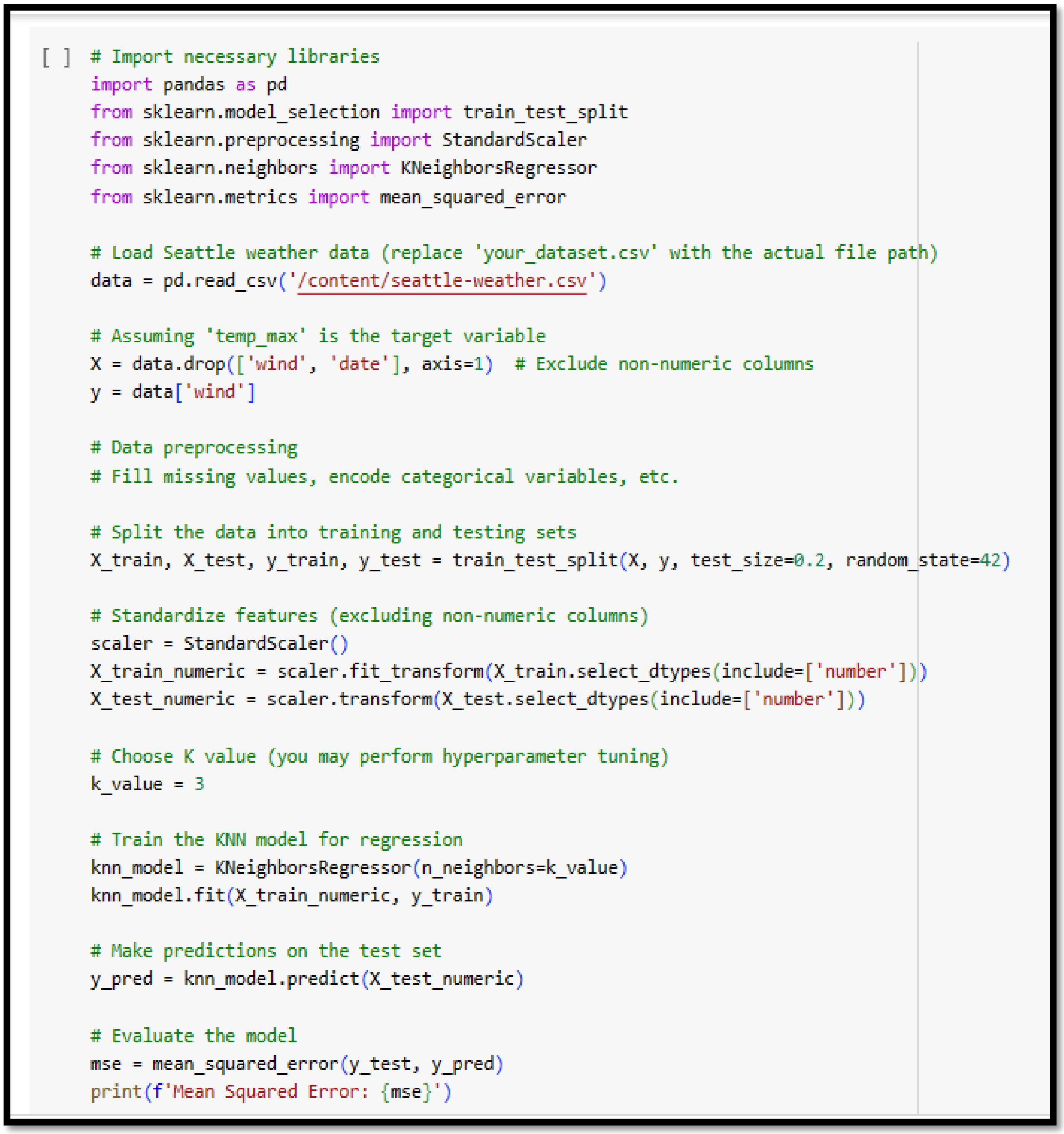
output:

```
F1 score: 0.979020979020979
Accuracy: 0.9736842105263158
Recall: 0.9859154929577465
```

Lab#6&7

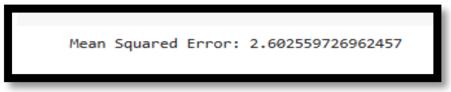
Task#01

Implementing a KNN on your given dataset ?



```
[ ] # Import necessary libraries
    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.neighbors import KNeighborsRegressor
    from sklearn.metrics import mean_squared_error

    # Load Seattle weather data (replace 'your_dataset.csv' with the actual file path)
    data = pd.read_csv('/content/seattle-weather.csv')

    # Assuming 'temp_max' is the target variable
    X = data.drop(['wind', 'date'], axis=1)  # Exclude non-numeric columns
    y = data['wind']

    # Data preprocessing
    # Fill missing values, encode categorical variables, etc.

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Standardize features (excluding non-numeric columns)
    scaler = StandardScaler()
    X_train_numeric = scaler.fit_transform(X_train.select_dtypes(include=['number']))
    X_test_numeric = scaler.transform(X_test.select_dtypes(include=['number']))

    # Choose K value (you may perform hyperparameter tuning)
    k_value = 3

    # Train the KNN model for regression
    knn_model = KNeighborsRegressor(n_neighbors=k_value)
    knn_model.fit(X_train_numeric, y_train)

    # Make predictions on the test set
    y_pred = knn_model.predict(X_test_numeric)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    print(f'Mean Squared Error: {mse}')
```

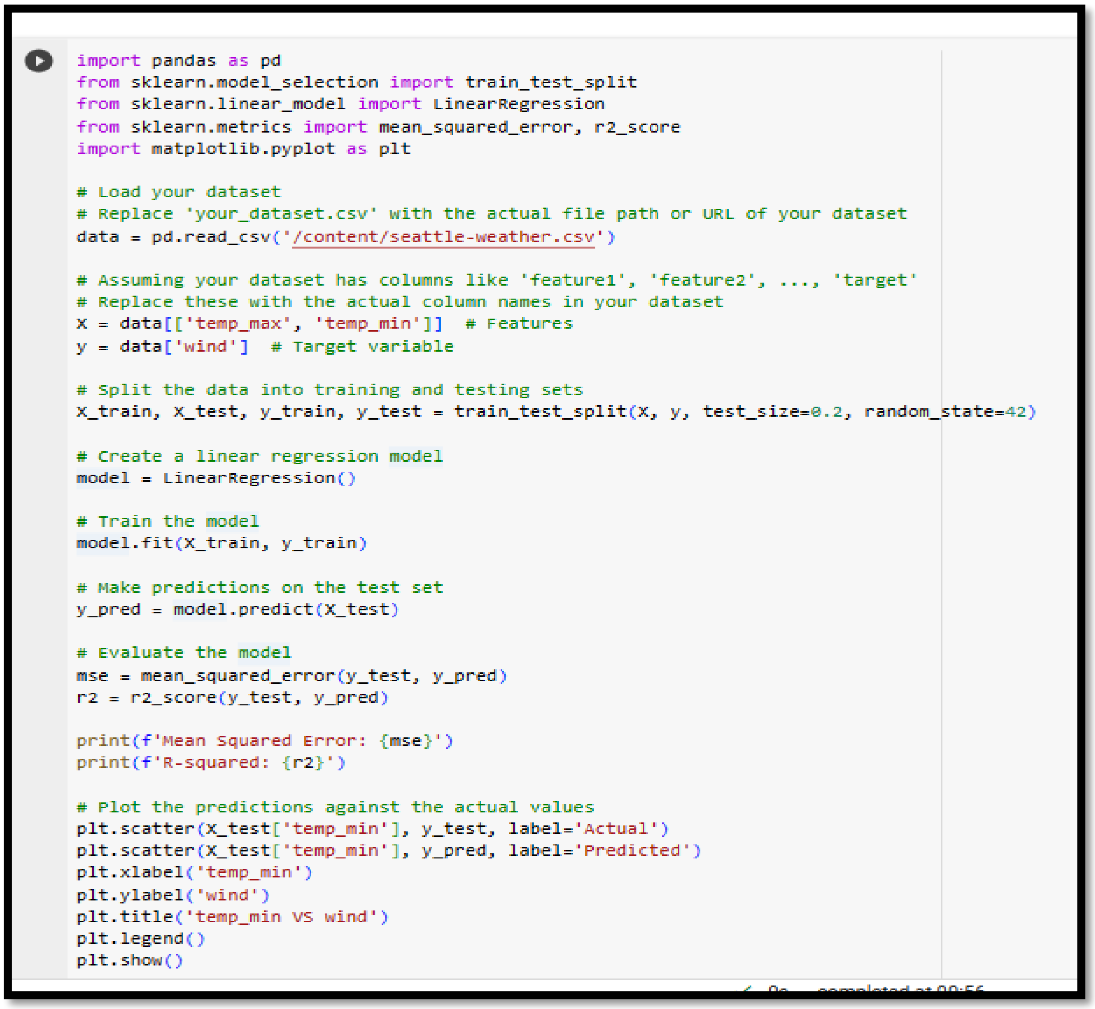
OUTPUT



```
Mean Squared Error: 2.602559726962457
```

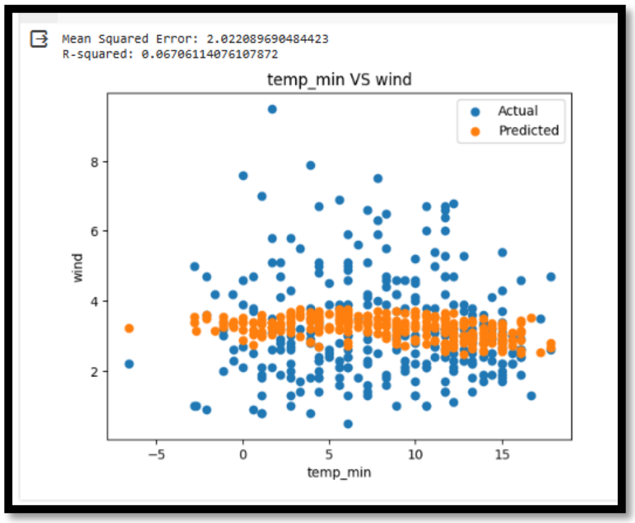
Task#02

Implement a Linear Regression on your given Dateset?



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load your dataset
# Replace 'your_dataset.csv' with the actual file path or URL of your dataset
data = pd.read_csv('/content/seattle-weather.csv')

# Assuming your dataset has columns like 'feature1', 'feature2', ..., 'target'
# Replace these with the actual column names in your dataset
X = data[['temp_max', 'temp_min']]  # Features
y = data['wind']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Plot the predictions against the actual values
plt.scatter(X_test['temp_min'], y_test, label='Actual')
plt.scatter(X_test['temp_min'], y_pred, label='Predicted')
plt.xlabel('temp_min')
plt.ylabel('wind')
plt.title('temp_min VS wind')
plt.legend()
plt.show()
```

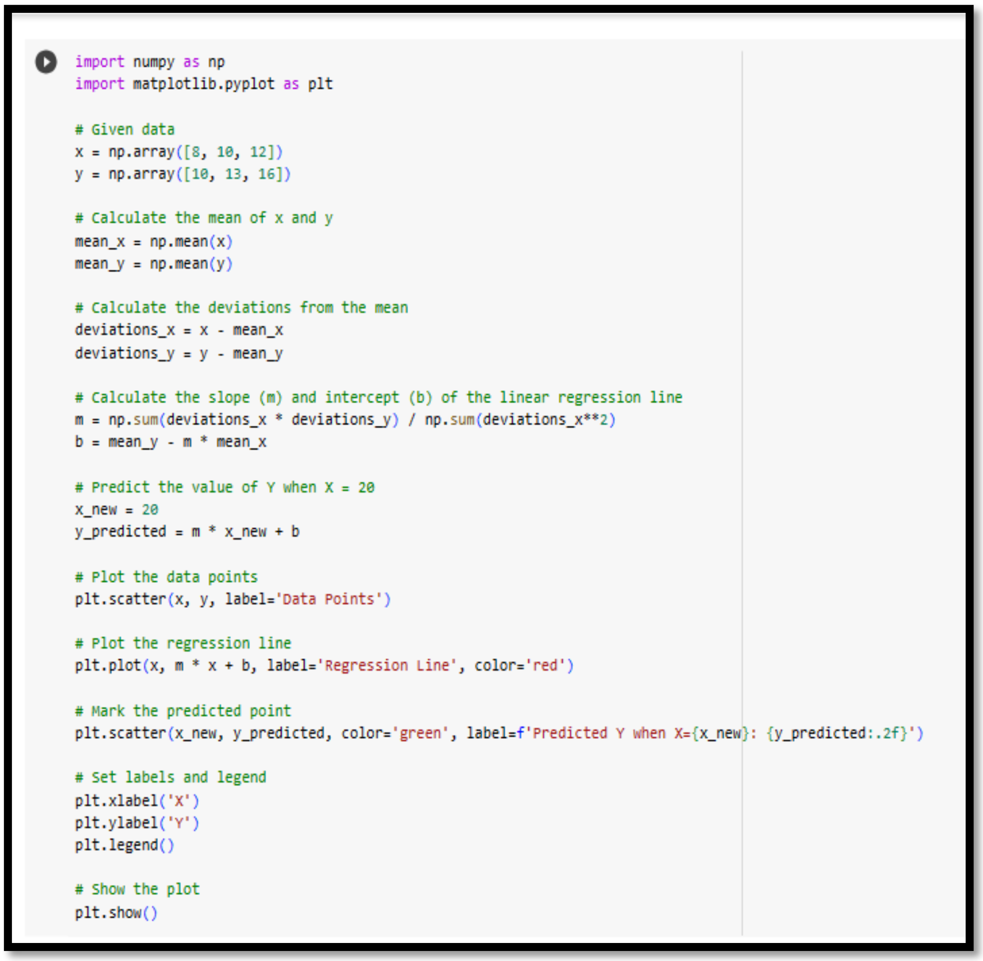
OUTPUT:



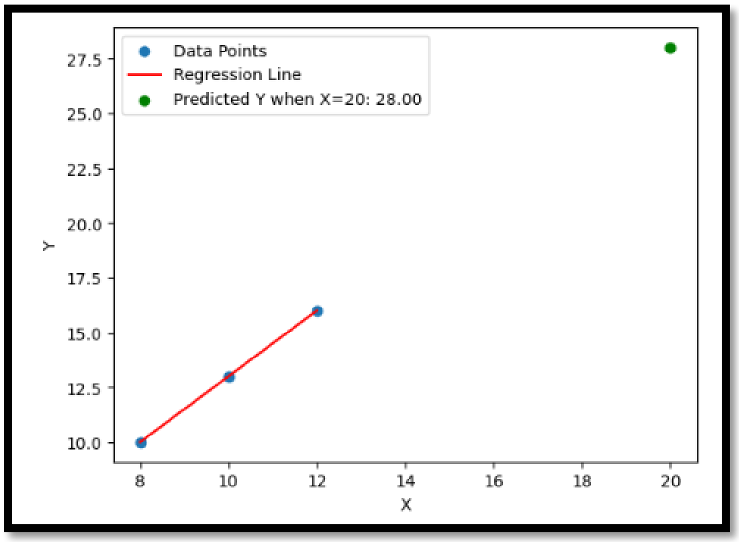
```
Mean Squared Error: 2.022089690484423
R-squared: 0.06706114076107872
```

Task#03

Implementation of given dataset Linear Regression?



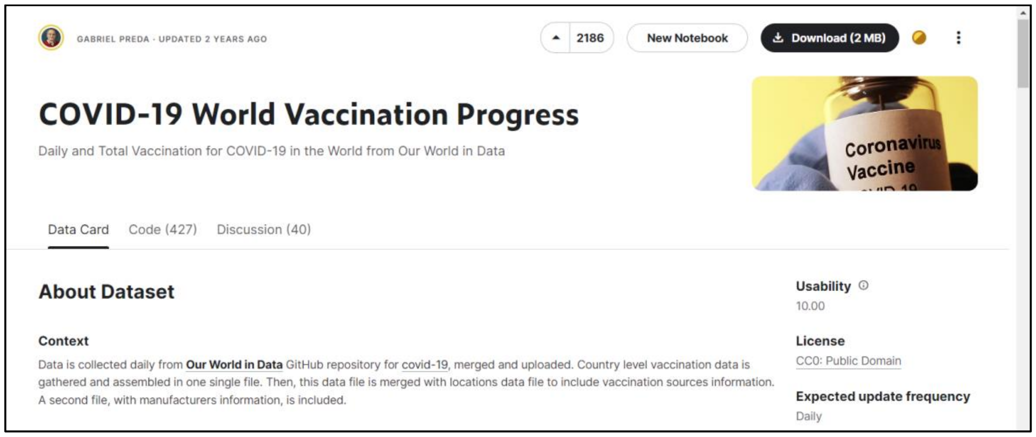
```python
import numpy as np
import matplotlib.pyplot as plt

# Given data
x = np.array([8, 10, 12])
y = np.array([10, 13, 16])

# Calculate the mean of x and y
mean_x = np.mean(x)
mean_y = np.mean(y)

# Calculate the deviations from the mean
deviations_x = x - mean_x
deviations_y = y - mean_y

# Calculate the slope (m) and intercept (b) of the linear regression line
m = np.sum(deviations_x * deviations_y) / np.sum(deviations_x**2)
b = mean_y - m * mean_x

# Predict the value of Y when X = 20
x_new = 20
y_predicted = m * x_new + b

# Plot the data points
plt.scatter(x, y, label='Data Points')

# Plot the regression line
plt.plot(x, m * x + b, label='Regression Line', color='red')

# Mark the predicted point
plt.scatter(x_new, y_predicted, color='green', label=f'Predicted Y when X={x_new}: {y_predicted:.2f}')

# Set labels and legend
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()

# Show the plot
plt.show()
```



Data Points
Regression Line
Predicted Y when X=20: 28.00
Y
X

<u>**Lab Task 8**</u>

Task#01 Take small Dataset to kaggle?



COVID-19 World Vaccination Progress

Daily and Total Vaccination for COVID-19 in the World from Our World in Data

About Dataset

Context

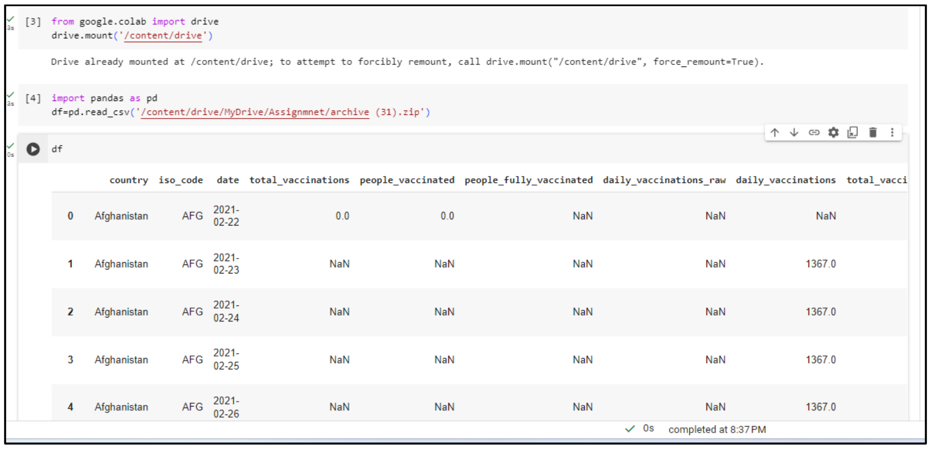
Data is collected daily from **Our World in Data** GitHub repository for covid-19, merged and uploaded. Country level vaccination data is gathered and assembled in one single file. Then, this data file is merged with locations data file to include vaccination sources information. A second file, with manufacturers information, is included.

Usability
10.00

License
CC0: Public Domain

Expected update frequency
Daily

Task#2 Upload dataset on colab and perform PreProcessing steps?



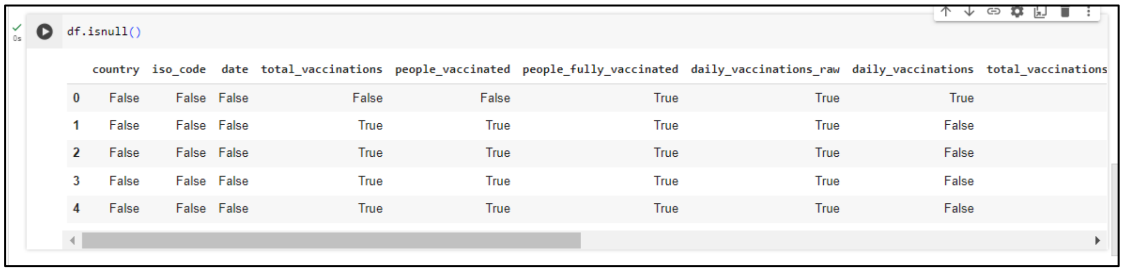
```
[3] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[4] import pandas as pd
    df=pd.read_csv('/content/drive/MyDrive/Assignmnet/archive (31).zip')

    df
```

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vacci
0	Afghanistan	AFG	2021-02-22	0.0	0.0	NaN	NaN	NaN	
1	Afghanistan	AFG	2021-02-23	NaN	NaN	NaN	NaN	1367.0	
2	Afghanistan	AFG	2021-02-24	NaN	NaN	NaN	NaN	1367.0	
3	Afghanistan	AFG	2021-02-25	NaN	NaN	NaN	NaN	1367.0	
4	Afghanistan	AFG	2021-02-26	NaN	NaN	NaN	NaN	1367.0	

✓ 0s completed at 8:37 PM

Task#3 Pre Processing steps?

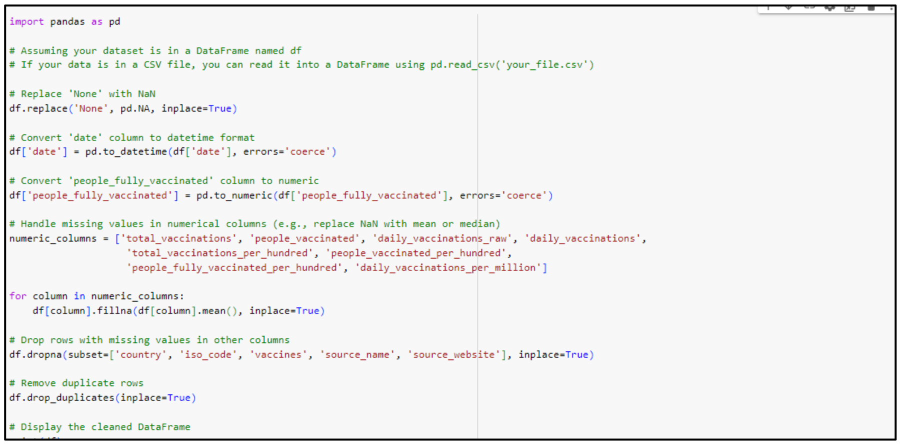
1. Handling Missing Values .



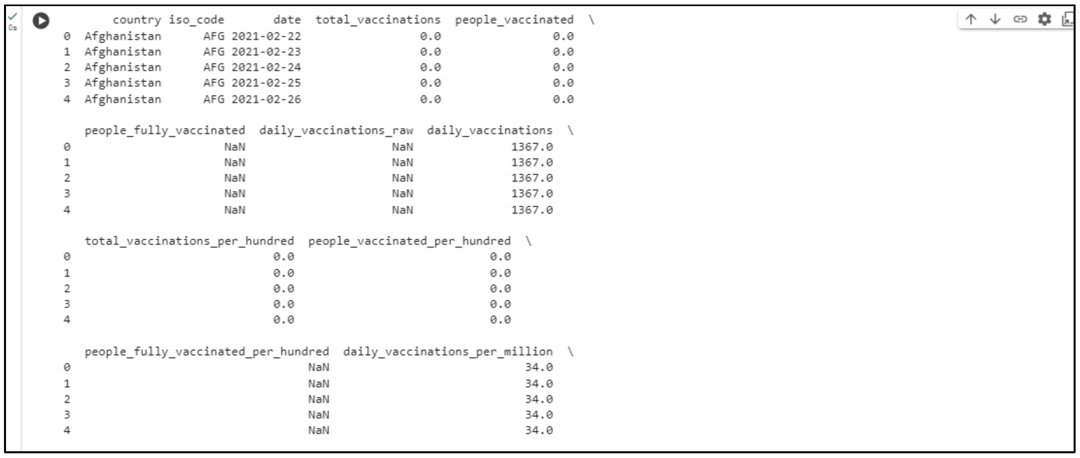
```python
df.isnull()
```

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vaccinations
0	False	False	False	False	False	True	True	True	
1	False	False	False	True	True	True	True	False	
2	False	False	False	True	True	True	True	False	
3	False	False	False	True	True	True	True	False	
4	False	False	False	True	True	True	True	False	

2. Data Cleaning .

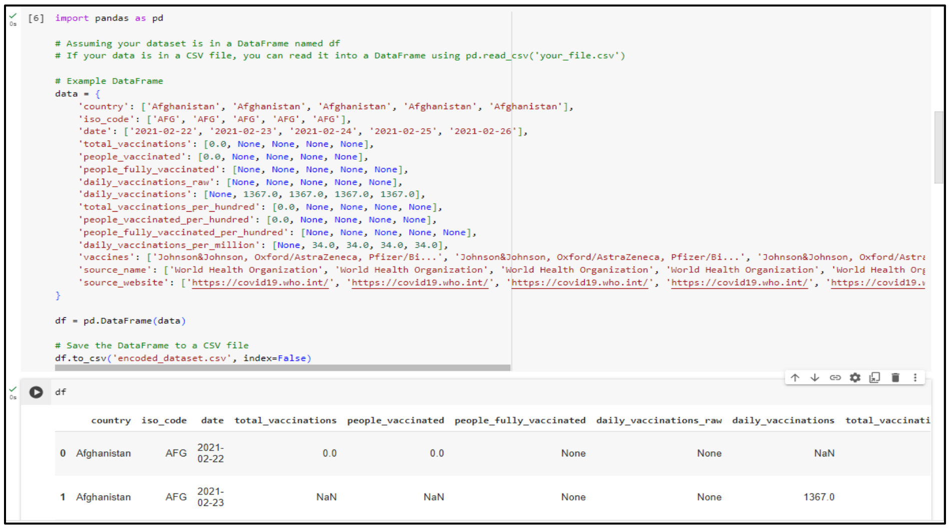


```python
import pandas as pd

# Assuming your dataset is in a DataFrame named df
# If your data is in a CSV file, you can read it into a DataFrame using pd.read_csv('your_file.csv')

# Replace 'None' with NaN
df.replace('None', pd.NA, inplace=True)

# Convert 'date' column to datetime format
df['date'] = pd.to_datetime(df['date'], errors='coerce')

# Convert 'people_fully_vaccinated' column to numeric
df['people_fully_vaccinated'] = pd.to_numeric(df['people_fully_vaccinated'], errors='coerce')

# Handle missing values in numerical columns (e.g., replace NaN with mean or median)
numeric_columns = ['total_vaccinations', 'people_vaccinated', 'daily_vaccinations_raw', 'daily_vaccinations',
                   'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred',
                   'people_fully_vaccinated_per_hundred', 'daily_vaccinations_per_million']

for column in numeric_columns:
    df[column].fillna(df[column].mean(), inplace=True)

# Drop rows with missing values in other columns
df.dropna(subset=['country', 'iso_code', 'vaccines', 'source_name', 'source_website'], inplace=True)

# Remove duplicate rows
df.drop_duplicates(inplace=True)

# Display the cleaned DataFrame
```



```
       country iso_code       date  total_vaccinations  people_vaccinated  \
0  Afghanistan      AFG 2021-02-22                 0.0                0.0
1  Afghanistan      AFG 2021-02-23                 0.0                0.0
2  Afghanistan      AFG 2021-02-24                 0.0                0.0
3  Afghanistan      AFG 2021-02-25                 0.0                0.0
4  Afghanistan      AFG 2021-02-26                 0.0                0.0

   people_fully_vaccinated  daily_vaccinations_raw  daily_vaccinations  \
0                      NaN                     NaN              1367.0
1                      NaN                     NaN              1367.0
2                      NaN                     NaN              1367.0
3                      NaN                     NaN              1367.0
4                      NaN                     NaN              1367.0

   total_vaccinations_per_hundred  people_vaccinated_per_hundred  \
0                             0.0                            0.0
1                             0.0                            0.0
2                             0.0                            0.0
3                             0.0                            0.0
4                             0.0                            0.0

   people_fully_vaccinated_per_hundred  daily_vaccinations_per_million  \
0                                  NaN                            34.0
1                                  NaN                            34.0
2                                  NaN                            34.0
3                                  NaN                            34.0
4                                  NaN                            34.0
```

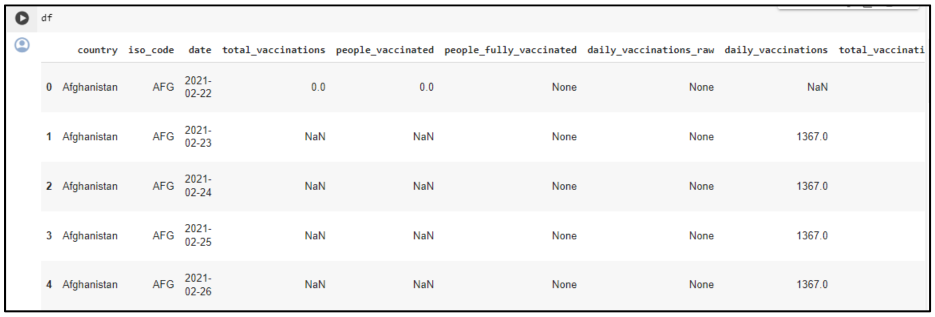
3. Handling Categorical Data .



```
[6] import pandas as pd

    # Assuming your dataset is in a DataFrame named df
    # If your data is in a CSV file, you can read it into a DataFrame using pd.read_csv('your_file.csv')

    # Example DataFrame
    data = {
        'country': ['Afghanistan', 'Afghanistan', 'Afghanistan', 'Afghanistan', 'Afghanistan'],
        'iso_code': ['AFG', 'AFG', 'AFG', 'AFG', 'AFG'],
        'date': ['2021-02-22', '2021-02-23', '2021-02-24', '2021-02-25', '2021-02-26'],
        'total_vaccinations': [0.0, None, None, None, None],
        'people_vaccinated': [0.0, None, None, None, None],
        'people_fully_vaccinated': [None, None, None, None, None],
        'daily_vaccinations_raw': [None, None, None, None, None],
        'daily_vaccinations': [None, 1367.0, 1367.0, 1367.0, 1367.0],
        'total_vaccinations_per_hundred': [0.0, None, None, None, None],
        'people_vaccinated_per_hundred': [0.0, None, None, None, None],
        'people_fully_vaccinated_per_hundred': [None, None, None, None, None],
        'daily_vaccinations_per_million': [None, 34.0, 34.0, 34.0, 34.0],
        'vaccines': ['Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...', 'Johnson&Johnson, Oxford/AstraZeneca, Pfizer/Bi...', 'Johnson&Johnson, Oxford/Astra
        'source_name': ['World Health Organization', 'World Health Organization', 'World Health Organization', 'World Health Organization', 'World Health Org
        'source_website': ['https://covid19.who.int/', 'https://covid19.who.int/', 'https://covid19.who.int/', 'https://covid19.who.int/', 'https://covid19.w
    }

    df = pd.DataFrame(data)

    # Save the DataFrame to a CSV file
    df.to_csv('encoded_dataset.csv', index=False)
```

```
df
```

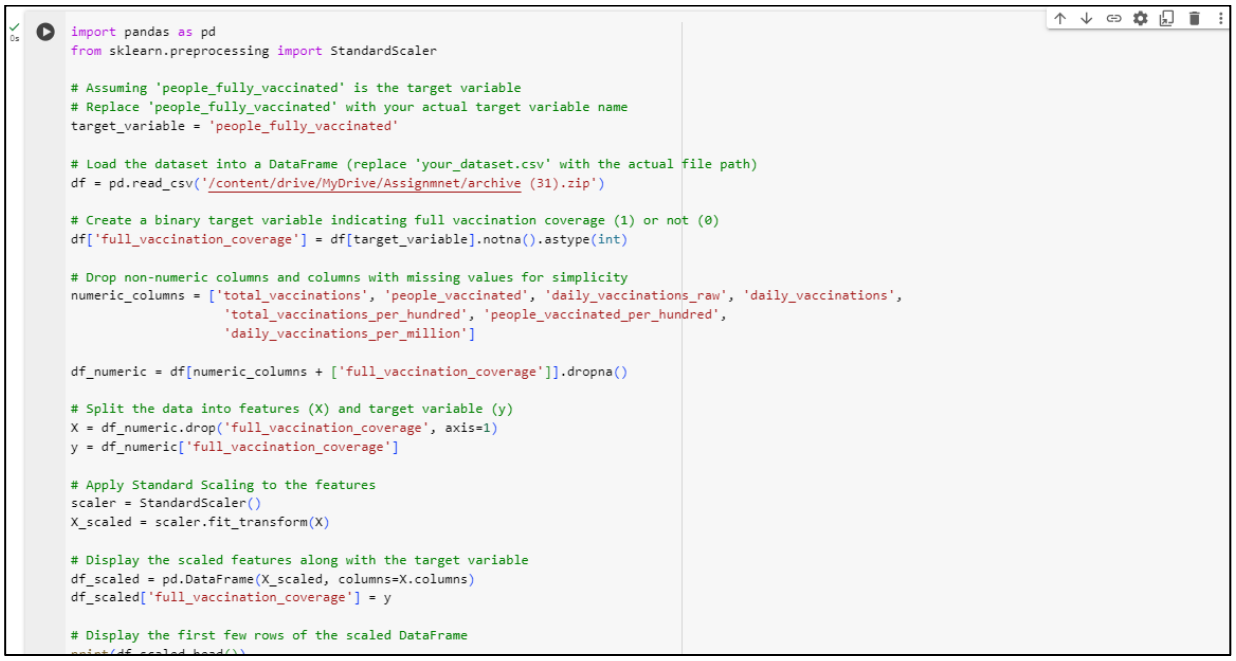
	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vaccinati
0	Afghanistan	AFG	2021-02-22	0.0	0.0	None	None	NaN	
1	Afghanistan	AFG	2021-02-23	NaN	NaN	None	None	1367.0	



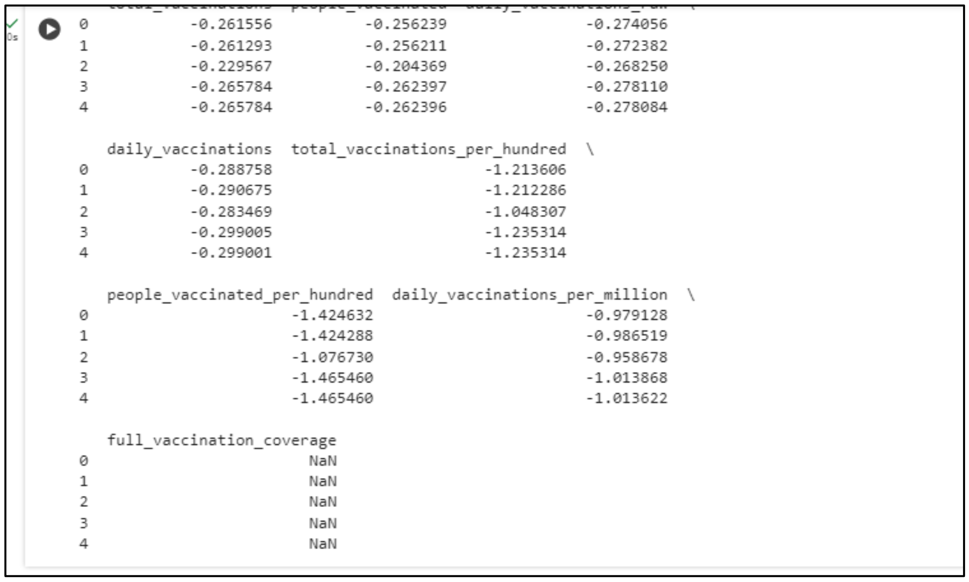
```
df
```

	country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vaccinati
0	Afghanistan	AFG	2021-02-22	0.0	0.0	None	None	NaN	
1	Afghanistan	AFG	2021-02-23	NaN	NaN	None	None	1367.0	
2	Afghanistan	AFG	2021-02-24	NaN	NaN	None	None	1367.0	
3	Afghanistan	AFG	2021-02-25	NaN	NaN	None	None	1367.0	
4	Afghanistan	AFG	2021-02-26	NaN	NaN	None	None	1367.0	

4. Data Scaling .

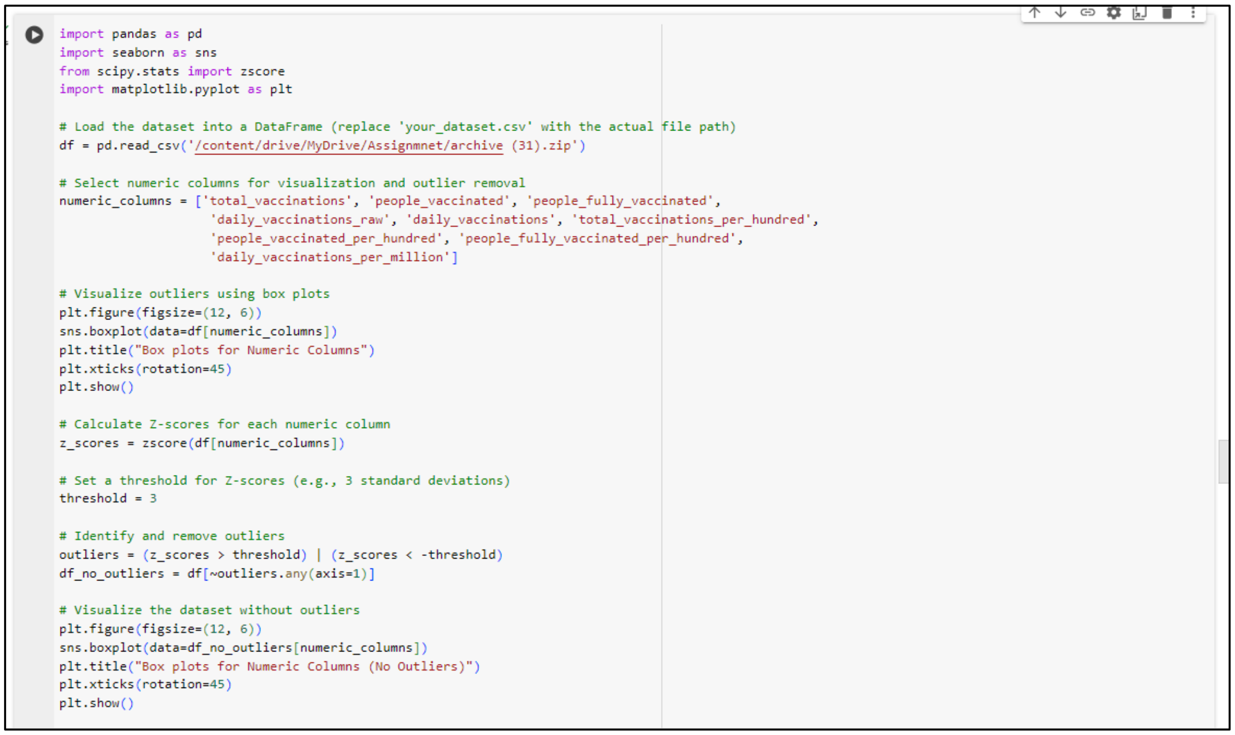


```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Assuming 'people_fully_vaccinated' is the target variable
# Replace 'people_fully_vaccinated' with your actual target variable name
target_variable = 'people_fully_vaccinated'

# Load the dataset into a DataFrame (replace 'your_dataset.csv' with the actual file path)
df = pd.read_csv('/content/drive/MyDrive/Assignmnet/archive (31).zip')

# Create a binary target variable indicating full vaccination coverage (1) or not (0)
df['full_vaccination_coverage'] = df[target_variable].notna().astype(int)

# Drop non-numeric columns and columns with missing values for simplicity
numeric_columns = ['total_vaccinations', 'people_vaccinated', 'daily_vaccinations_raw', 'daily_vaccinations',
                   'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred',
                   'daily_vaccinations_per_million']

df_numeric = df[numeric_columns + ['full_vaccination_coverage']].dropna()

# Split the data into features (X) and target variable (y)
X = df_numeric.drop('full_vaccination_coverage', axis=1)
y = df_numeric['full_vaccination_coverage']

# Apply Standard Scaling to the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Display the scaled features along with the target variable
df_scaled = pd.DataFrame(X_scaled, columns=X.columns)
df_scaled['full_vaccination_coverage'] = y

# Display the first few rows of the scaled DataFrame
```

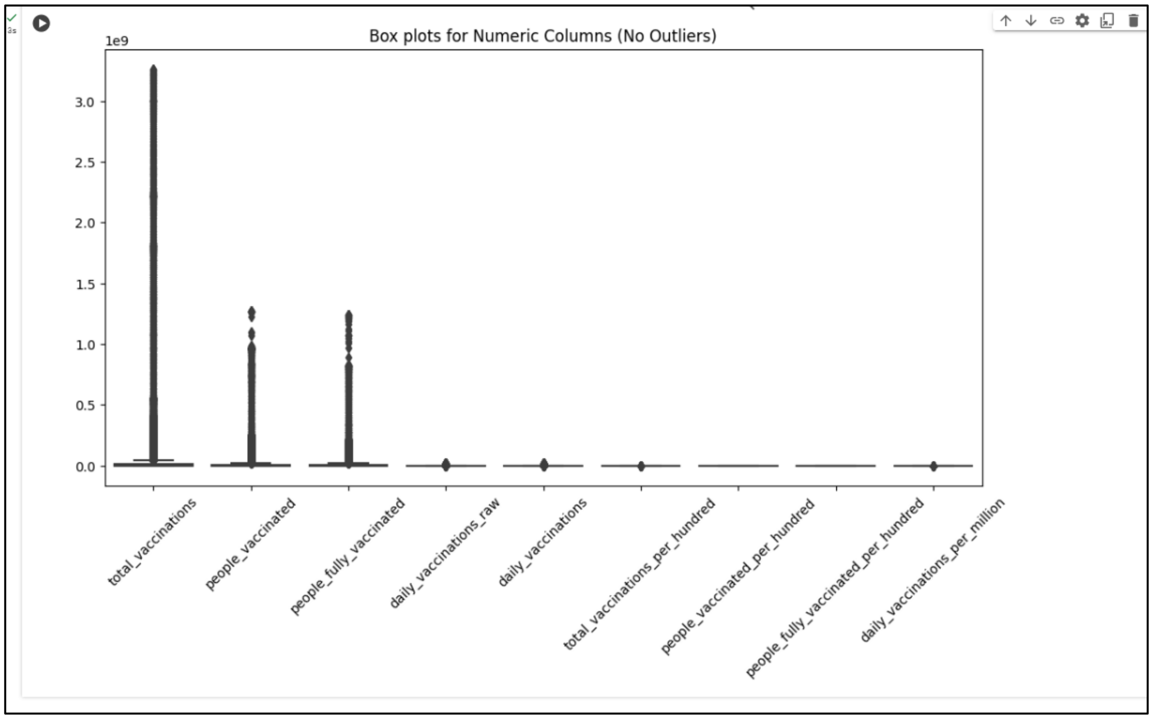


```
   0         -0.261556           -0.256239                -0.274056
   1         -0.261293           -0.256211                -0.272382
   2         -0.229567           -0.204369                -0.268250
   3         -0.265784           -0.262397                -0.278110
   4         -0.265784           -0.262396                -0.278084

   daily_vaccinations  total_vaccinations_per_hundred  \
0           -0.288758                       -1.213606
1           -0.290675                       -1.212286
2           -0.283469                       -1.048307
3           -0.299005                       -1.235314
4           -0.299001                       -1.235314

   people_vaccinated_per_hundred  daily_vaccinations_per_million  \
0                      -1.424632                       -0.979128
1                      -1.424288                       -0.986519
2                      -1.076730                       -0.958678
3                      -1.465460                       -1.013868
4                      -1.465460                       -1.013622

   full_vaccination_coverage
0                        NaN
1                        NaN
2                        NaN
3                        NaN
4                        NaN
```

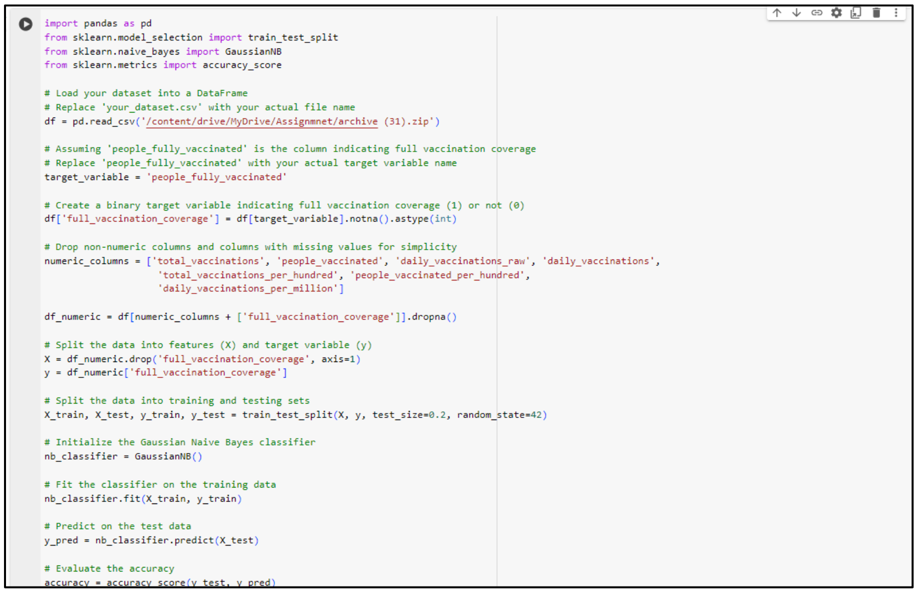
5. Otliers remove



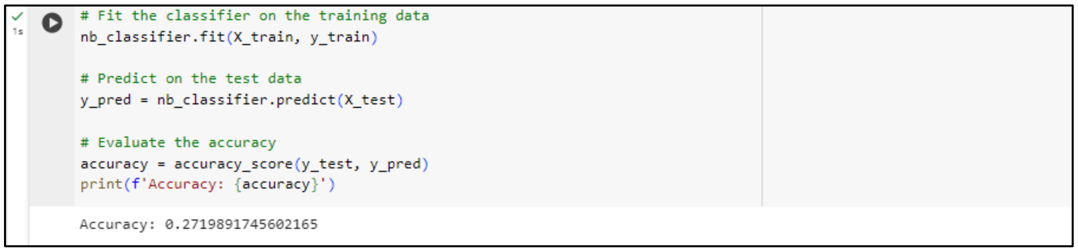
```python
import pandas as pd
import seaborn as sns
from scipy.stats import zscore
import matplotlib.pyplot as plt

# Load the dataset into a DataFrame (replace 'your_dataset.csv' with the actual file path)
df = pd.read_csv('/content/drive/MyDrive/Assignmnet/archive (31).zip')

# Select numeric columns for visualization and outlier removal
numeric_columns = ['total_vaccinations', 'people_vaccinated', 'people_fully_vaccinated',
                   'daily_vaccinations_raw', 'daily_vaccinations', 'total_vaccinations_per_hundred',
                   'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred',
                   'daily_vaccinations_per_million']

# Visualize outliers using box plots
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[numeric_columns])
plt.title("Box plots for Numeric Columns")
plt.xticks(rotation=45)
plt.show()

# Calculate Z-scores for each numeric column
z_scores = zscore(df[numeric_columns])

# Set a threshold for Z-scores (e.g., 3 standard deviations)
threshold = 3

# Identify and remove outliers
outliers = (z_scores > threshold) | (z_scores < -threshold)
df_no_outliers = df[~outliers.any(axis=1)]

# Visualize the dataset without outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_no_outliers[numeric_columns])
plt.title("Box plots for Numeric Columns (No Outliers)")
plt.xticks(rotation=45)
plt.show()
```



Task#3 Apply naive Gaussian code in Python?



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load your dataset into a DataFrame
# Replace 'your_dataset.csv' with your actual file name
df = pd.read_csv('/content/drive/MyDrive/Assignmnet/archive (31).zip')

# Assuming 'people_fully_vaccinated' is the column indicating full vaccination coverage
# Replace 'people_fully_vaccinated' with your actual target variable name
target_variable = 'people_fully_vaccinated'

# Create a binary target variable indicating full vaccination coverage (1) or not (0)
df['full_vaccination_coverage'] = df[target_variable].notna().astype(int)

# Drop non-numeric columns and columns with missing values for simplicity
numeric_columns = ['total_vaccinations', 'people_vaccinated', 'daily_vaccinations_raw', 'daily_vaccinations',
                   'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred',
                   'daily_vaccinations_per_million']

df_numeric = df[numeric_columns + ['full_vaccination_coverage']].dropna()

# Split the data into features (X) and target variable (y)
X = df_numeric.drop('full_vaccination_coverage', axis=1)
y = df_numeric['full_vaccination_coverage']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

# Predict on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
```

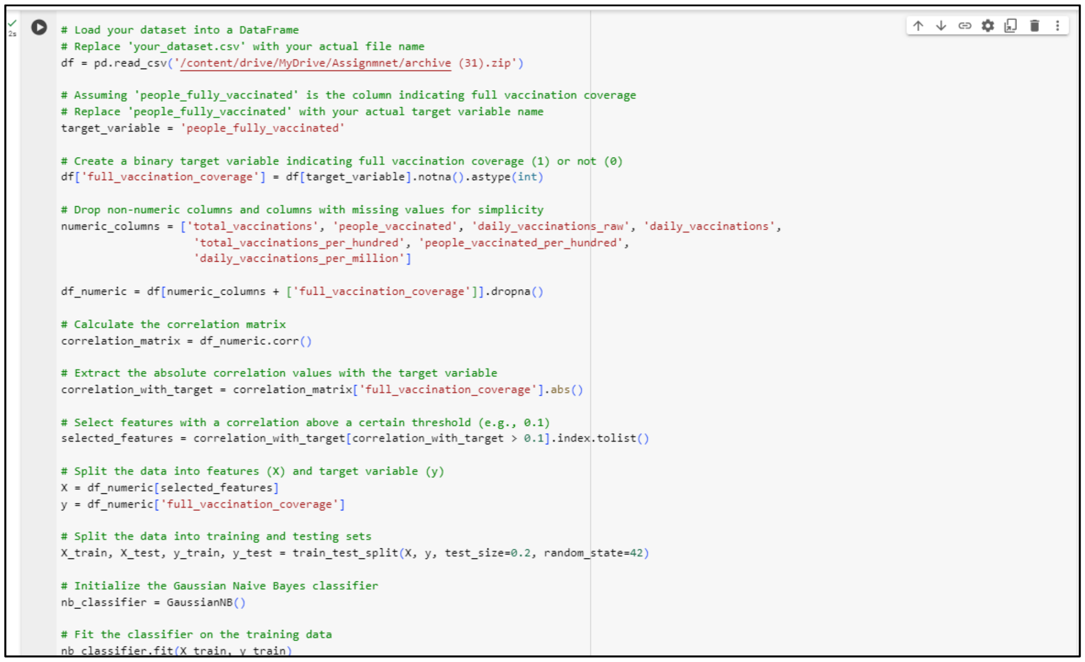


```
# Fit the classifier on the training data
nb_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

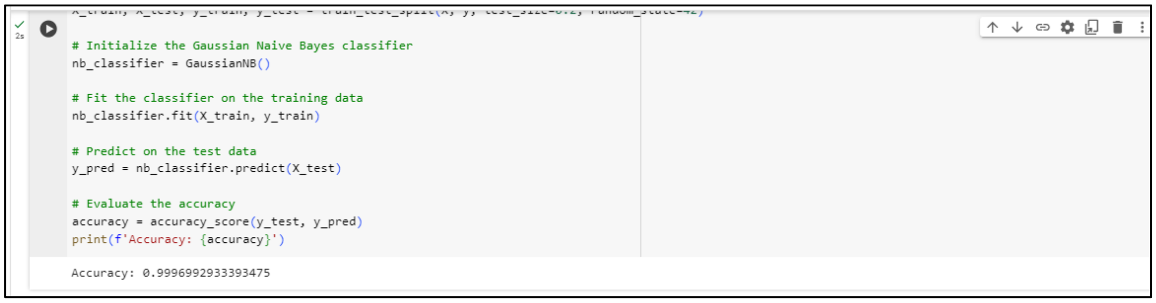
```
Accuracy: 0.2719891745602165
```

Task#4 Apply Correlation Cofficient and select the features to check which give higher accuracy?

When I apply to full dataset So over all accuracy is given below



```
# Load your dataset into a DataFrame
# Replace 'your_dataset.csv' with your actual file name
df = pd.read_csv('/content/drive/MyDrive/Assignmnet/archive (31).zip')

# Assuming 'people_fully_vaccinated' is the column indicating full vaccination coverage
# Replace 'people_fully_vaccinated' with your actual target variable name
target_variable = 'people_fully_vaccinated'

# Create a binary target variable indicating full vaccination coverage (1) or not (0)
df['full_vaccination_coverage'] = df[target_variable].notna().astype(int)

# Drop non-numeric columns and columns with missing values for simplicity
numeric_columns = ['total_vaccinations', 'people_vaccinated', 'daily_vaccinations_raw', 'daily_vaccinations',
                   'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred',
                   'daily_vaccinations_per_million']

df_numeric = df[numeric_columns + ['full_vaccination_coverage']].dropna()

# Calculate the correlation matrix
correlation_matrix = df_numeric.corr()

# Extract the absolute correlation values with the target variable
correlation_with_target = correlation_matrix['full_vaccination_coverage'].abs()

# Select features with a correlation above a certain threshold (e.g., 0.1)
selected_features = correlation_with_target[correlation_with_target > 0.1].index.tolist()

# Split the data into features (X) and target variable (y)
X = df_numeric[selected_features]
y = df_numeric['full_vaccination_coverage']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

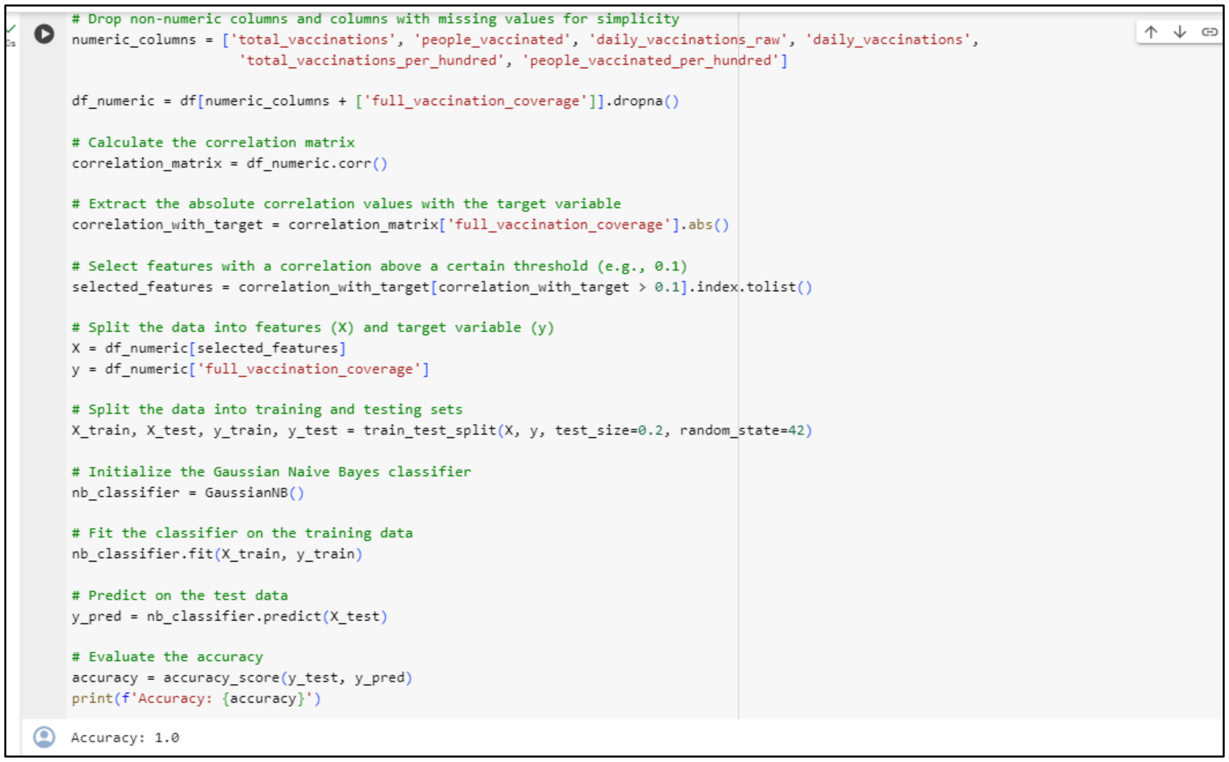


```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

# Predict on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9996992933393475
```

Now I select features and check accuracy?

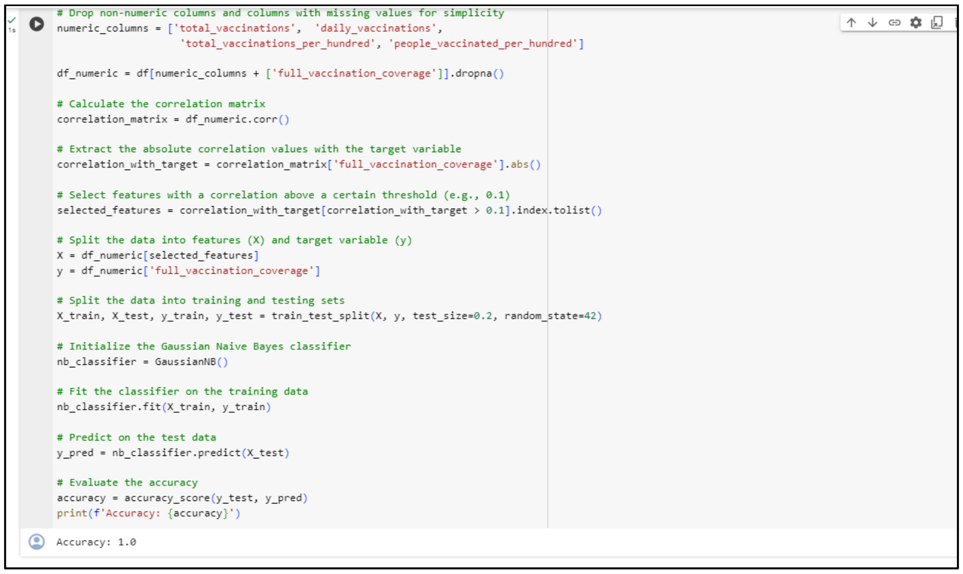
Step#1



```python
# Drop non-numeric columns and columns with missing values for simplicity
numeric_columns = ['total_vaccinations', 'people_vaccinated', 'daily_vaccinations_raw', 'daily_vaccinations',
                    'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred']

df_numeric = df[numeric_columns + ['full_vaccination_coverage']].dropna()

# Calculate the correlation matrix
correlation_matrix = df_numeric.corr()

# Extract the absolute correlation values with the target variable
correlation_with_target = correlation_matrix['full_vaccination_coverage'].abs()

# Select features with a correlation above a certain threshold (e.g., 0.1)
selected_features = correlation_with_target[correlation_with_target > 0.1].index.tolist()

# Split the data into features (X) and target variable (y)
X = df_numeric[selected_features]
y = df_numeric['full_vaccination_coverage']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

# Predict on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 1.0
```

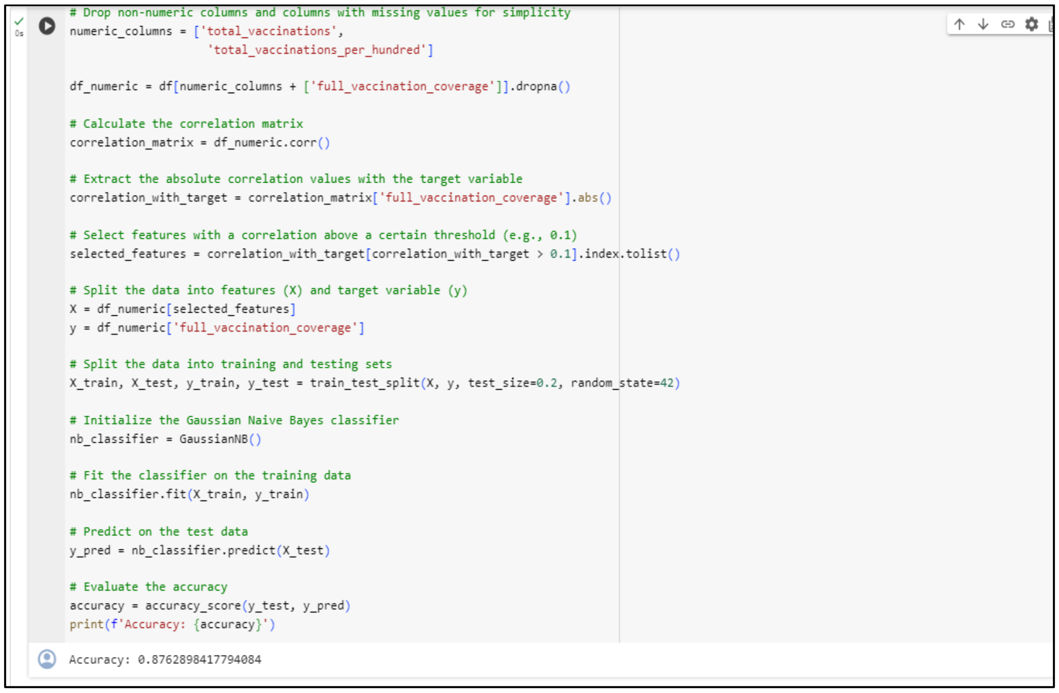
Step#2



```python
# Drop non-numeric columns and columns with missing values for simplicity
numeric_columns = ['total_vaccinations',  'daily_vaccinations',
                    'total_vaccinations_per_hundred', 'people_vaccinated_per_hundred']

df_numeric = df[numeric_columns + ['full_vaccination_coverage']].dropna()

# Calculate the correlation matrix
correlation_matrix = df_numeric.corr()

# Extract the absolute correlation values with the target variable
correlation_with_target = correlation_matrix['full_vaccination_coverage'].abs()

# Select features with a correlation above a certain threshold (e.g., 0.1)
selected_features = correlation_with_target[correlation_with_target > 0.1].index.tolist()

# Split the data into features (X) and target variable (y)
X = df_numeric[selected_features]
y = df_numeric['full_vaccination_coverage']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

# Predict on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 1.0
```

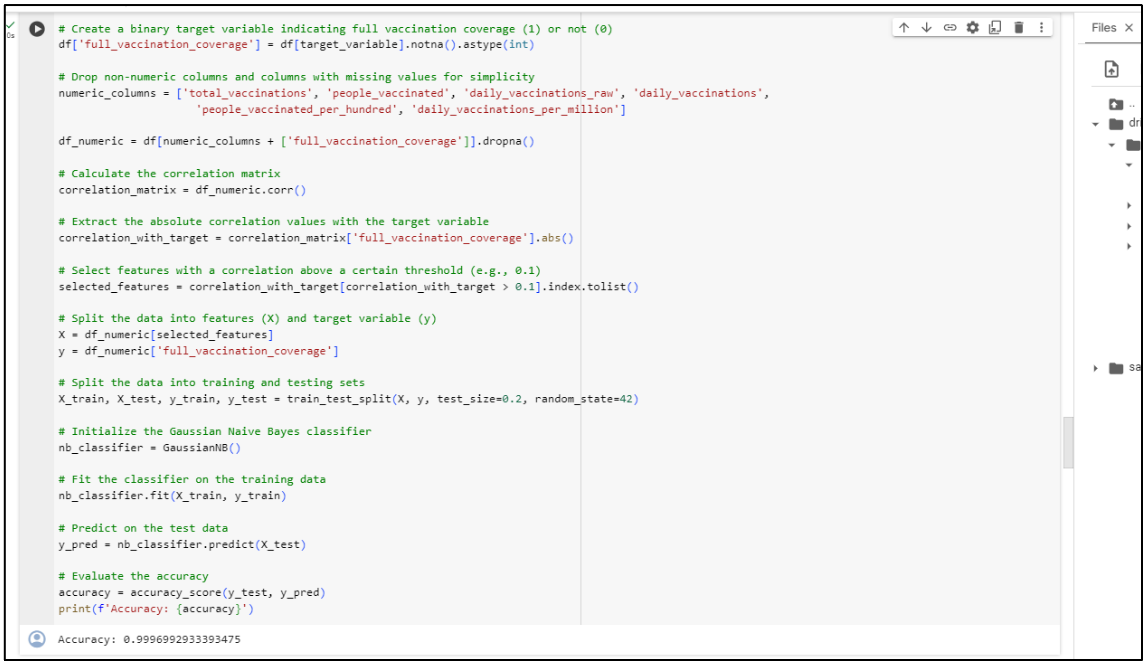
Step#3



```
# Drop non-numeric columns and columns with missing values for simplicity
numeric_columns = ['total_vaccinations',
                     'total_vaccinations_per_hundred']

df_numeric = df[numeric_columns + ['full_vaccination_coverage']].dropna()

# Calculate the correlation matrix
correlation_matrix = df_numeric.corr()

# Extract the absolute correlation values with the target variable
correlation_with_target = correlation_matrix['full_vaccination_coverage'].abs()

# Select features with a correlation above a certain threshold (e.g., 0.1)
selected_features = correlation_with_target[correlation_with_target > 0.1].index.tolist()

# Split the data into features (X) and target variable (y)
X = df_numeric[selected_features]
y = df_numeric['full_vaccination_coverage']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

# Predict on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.8762898417794084
```

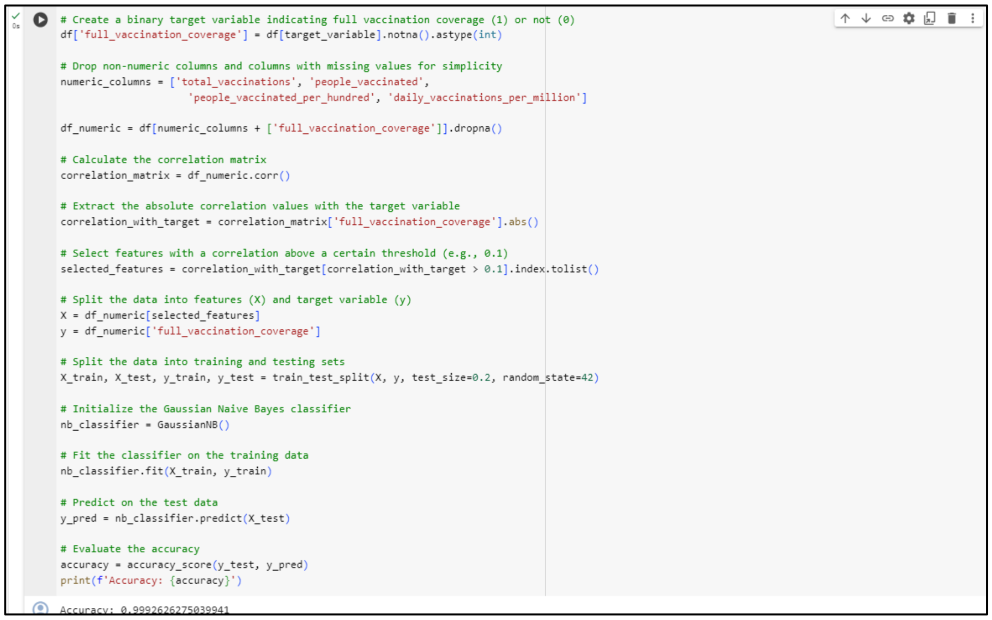
Step#4



```
# Create a binary target variable indicating full vaccination coverage (1) or not (0)
df['full_vaccination_coverage'] = df[target_variable].notna().astype(int)

# Drop non-numeric columns and columns with missing values for simplicity
numeric_columns = ['total_vaccinations', 'people_vaccinated', 'daily_vaccinations_raw', 'daily_vaccinations',
                     'people_vaccinated_per_hundred', 'daily_vaccinations_per_million']

df_numeric = df[numeric_columns + ['full_vaccination_coverage']].dropna()

# Calculate the correlation matrix
correlation_matrix = df_numeric.corr()

# Extract the absolute correlation values with the target variable
correlation_with_target = correlation_matrix['full_vaccination_coverage'].abs()

# Select features with a correlation above a certain threshold (e.g., 0.1)
selected_features = correlation_with_target[correlation_with_target > 0.1].index.tolist()

# Split the data into features (X) and target variable (y)
X = df_numeric[selected_features]
y = df_numeric['full_vaccination_coverage']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

# Predict on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9996992933393475
```

Step#5



```
# Create a binary target variable indicating full vaccination coverage (1) or not (0)
df['full_vaccination_coverage'] = df[target_variable].notna().astype(int)

# Drop non-numeric columns and columns with missing values for simplicity
numeric_columns = ['total_vaccinations', 'people_vaccinated',
                   'people_vaccinated_per_hundred', 'daily_vaccinations_per_million']

df_numeric = df[numeric_columns + ['full_vaccination_coverage']].dropna()

# Calculate the correlation matrix
correlation_matrix = df_numeric.corr()

# Extract the absolute correlation values with the target variable
correlation_with_target = correlation_matrix['full_vaccination_coverage'].abs()

# Select features with a correlation above a certain threshold (e.g., 0.1)
selected_features = correlation_with_target[correlation_with_target > 0.1].index.tolist()

# Split the data into features (X) and target variable (y)
X = df_numeric[selected_features]
y = df_numeric['full_vaccination_coverage']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

# Predict on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9992626275039941
```

Lab Task 9

Task#01

Difference between Feature Selection and Feature Extraction.

Feature selection and feature extraction are two important techniques used in machine learning and data analysis to reduce the dimensionality of data and improve the performance of machine learning models.

Feature selection involves selecting a subset of the most relevant and informative features from the original set of features. This can be done using various methods, such as:

- **Filter methods:** These methods score each feature based on its individual properties, such as variance or correlation with the target variable. Features with low scores are then removed.
- **Wrapper methods:** These methods evaluate the performance of a machine learning model on different subsets of features. The subset that produces the best performance is then selected.
- **Embedded methods:** These methods select features as part of the machine learning model training process. For example, L1 regularization (LASSO) and L2 regularization (Ridge) can be used to select features by penalizing the coefficients of less important features.

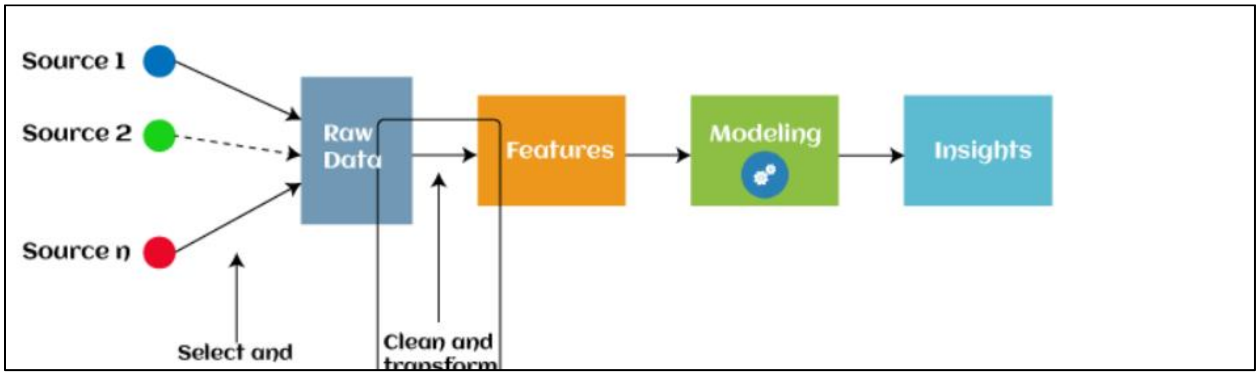
Feature extraction, on the other hand, involves transforming the original set of features into a new set of features that are more informative and discriminative. This can be done using various techniques, such as:

- **Principal component analysis (PCA):** This technique transforms the original features into a new set of orthogonal components called principal components. The principal components are ordered by their variance, so the first few components capture most of the information in the original data.
- **Linear discriminant analysis (LDA):** This technique finds a linear combination of the original features that best discriminates between two or more classes.
- **Independent component analysis (ICA):** This technique finds a set of statistically independent components from the original features. ICA is often used for blind source separation, such as separating audio signals from a mixture of sources.

Feature selection and feature extraction are both powerful techniques that can improve the performance of machine learning models. However, the choice of technique depends on the specific problem and dataset.

Task#02
Illustrate all the techniques of Feature engineering.

Feature engineering is a crucial step in machine learning, where raw data is transformed into features that are more informative and suitable for modeling. It involves various techniques to improve the quality and effectiveness of the data used for training machine learning algorithms. Here are some commonly used feature engineering techniques:



Feature Selection: Selecting the most relevant and informative features from the available dataset. This can be done through statistical measures like correlation analysis, mutual information, or feature importance scores from machine learning models.

Feature Transformation: Applying mathematical transformations to enhance the features' distribution or create new features. Common transformations include log transformation, square root, binning, and one-hot encoding for categorical variables.

Feature Discretization: Converting continuous features into discrete categories or bins. This can be useful for reducing dimensionality, handling non-linear relationships, and improving model interpretability.

Feature Scaling: Normalizing the features to have a consistent scale, ensuring that they are all treated equally by the machine learning algorithm. Common scaling techniques include min-max scaling, standard scaling, and decimal scaling.

Feature Interaction: Creating new features by combining or multiplying existing features. This can capture non-linear relationships and interactions between features that may not be evident in the original dataset.

Feature Reduction: Reducing the number of features while preserving the most important information. Techniques like principal component analysis (PCA), singular value decomposition (SVD), and linear discriminant analysis (LDA) can be used for dimensionality reduction.

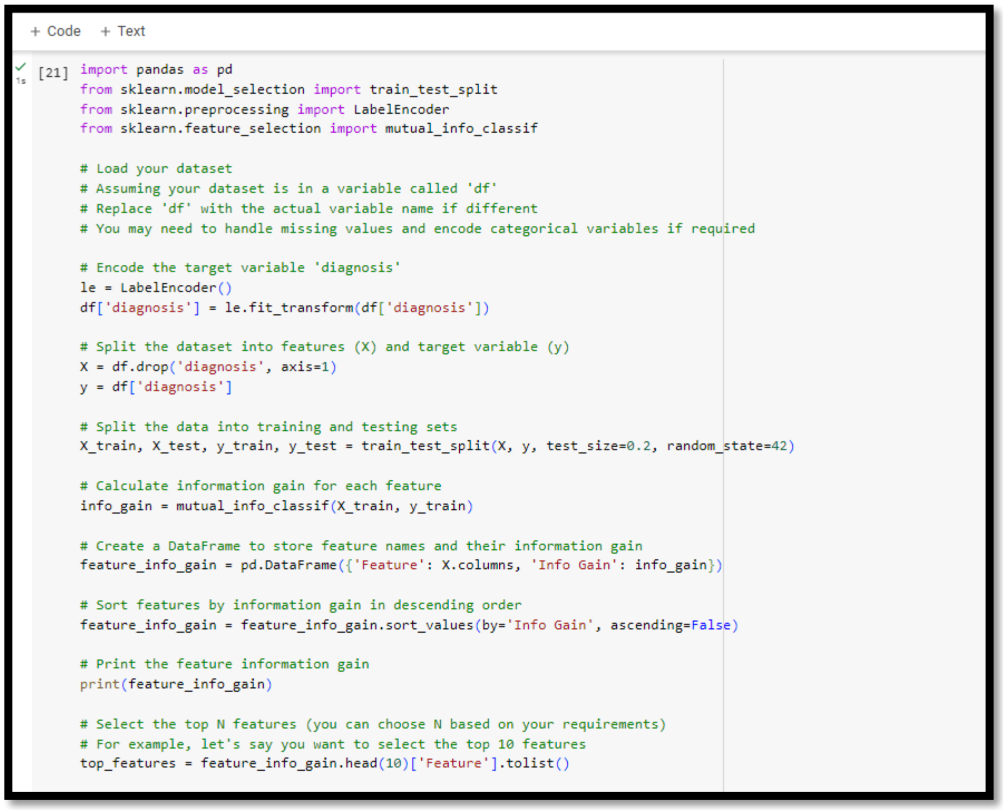
Feature Encoding: Encoding categorical variables into numerical form for machine learning algorithms that can only handle numerical data. One-hot encoding, label encoding, and binary encoding are commonly used encoding techniques.

By applying these feature engineering techniques, you can improve the performance and interpretability of machine learning models, leading to better predictions and decision-making.

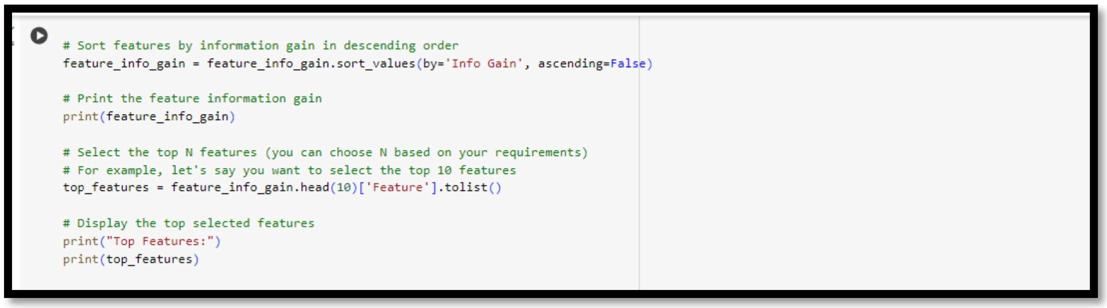
Task#03

Implement the different types Filter Methods on your respective dataset.

Information Gain

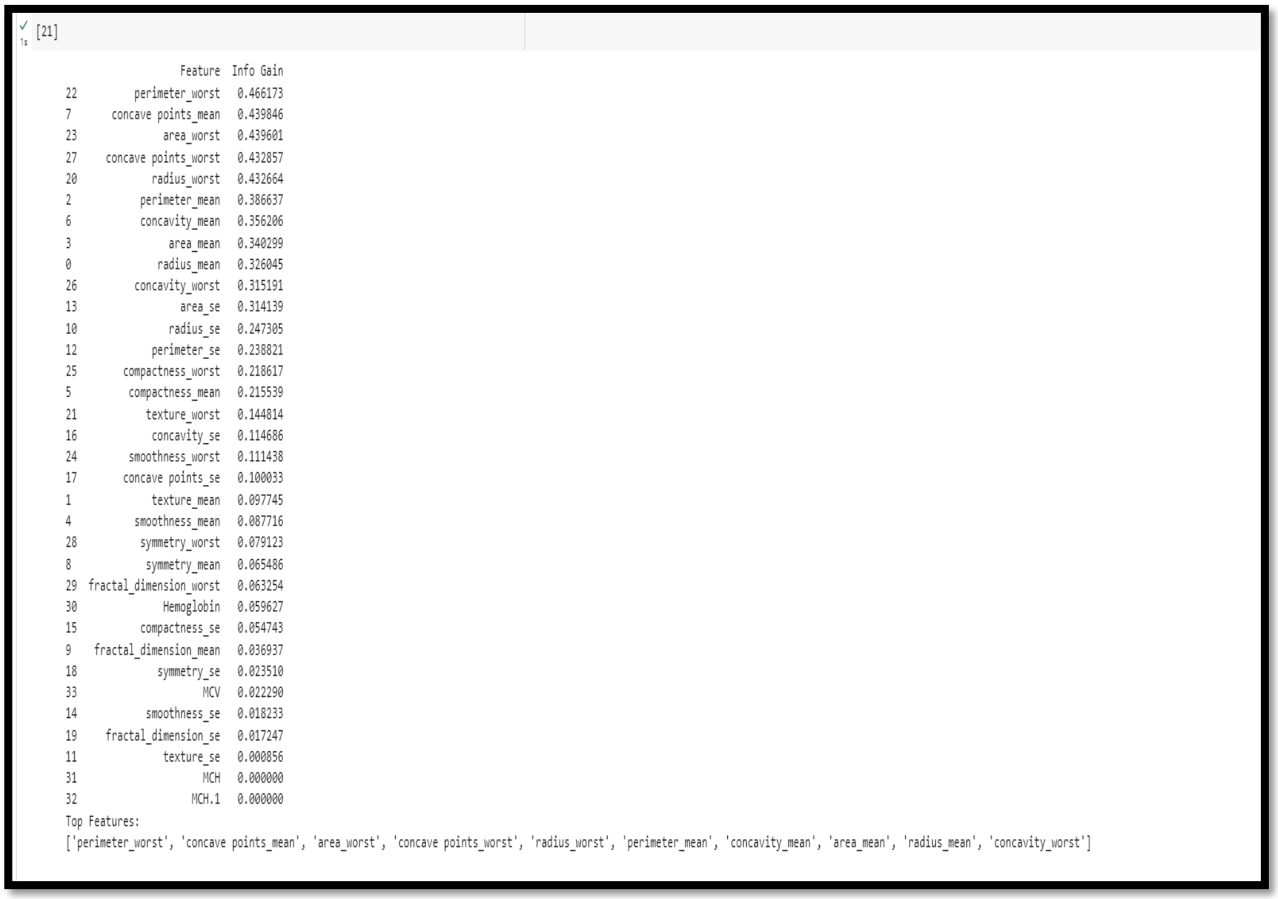


```
[21] import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import LabelEncoder
     from sklearn.feature_selection import mutual_info_classif

     # Load your dataset
     # Assuming your dataset is in a variable called 'df'
     # Replace 'df' with the actual variable name if different
     # You may need to handle missing values and encode categorical variables if required

     # Encode the target variable 'diagnosis'
     le = LabelEncoder()
     df['diagnosis'] = le.fit_transform(df['diagnosis'])

     # Split the dataset into features (X) and target variable (y)
     X = df.drop('diagnosis', axis=1)
     y = df['diagnosis']

     # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     # Calculate information gain for each feature
     info_gain = mutual_info_classif(X_train, y_train)

     # Create a DataFrame to store feature names and their information gain
     feature_info_gain = pd.DataFrame({'Feature': X.columns, 'Info Gain': info_gain})

     # Sort features by information gain in descending order
     feature_info_gain = feature_info_gain.sort_values(by='Info Gain', ascending=False)

     # Print the feature information gain
     print(feature_info_gain)

     # Select the top N features (you can choose N based on your requirements)
     # For example, let's say you want to select the top 10 features
     top_features = feature_info_gain.head(10)['Feature'].tolist()
```



```
# Sort features by information gain in descending order
feature_info_gain = feature_info_gain.sort_values(by='Info Gain', ascending=False)

# Print the feature information gain
print(feature_info_gain)

# Select the top N features (you can choose N based on your requirements)
# For example, let's say you want to select the top 10 features
top_features = feature_info_gain.head(10)['Feature'].tolist()

# Display the top selected features
print("Top Features:")
print(top_features)
```

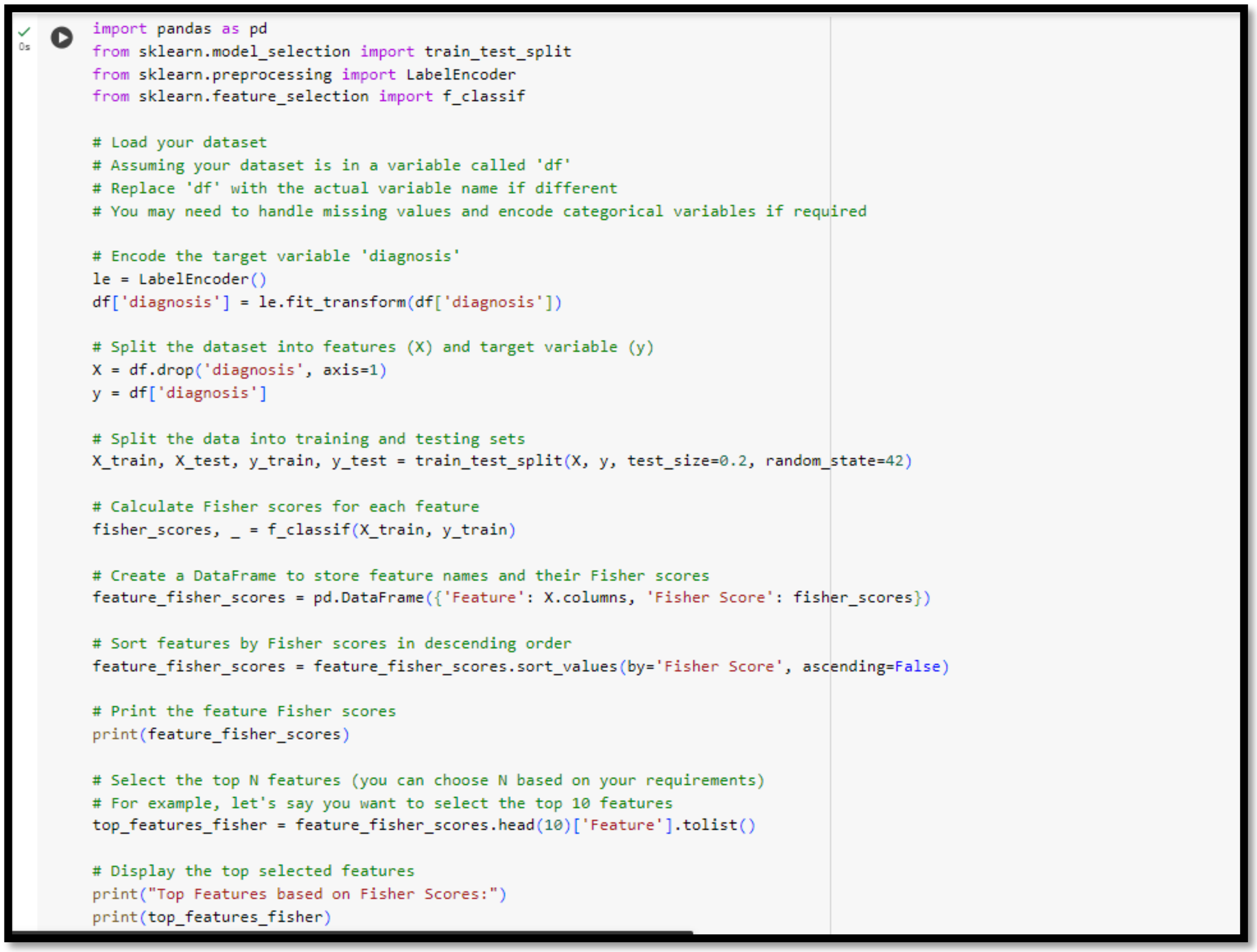
Output:



```
[21]

                   Feature  Info Gain
22         perimeter_worst   0.466173
7      concave points_mean   0.439846
23              area_worst   0.439601
27    concave points_worst   0.432857
20            radius_worst   0.432664
2           perimeter_mean   0.386637
6           concavity_mean   0.356206
3                area_mean   0.340299
0              radius_mean   0.326045
26         concavity_worst   0.315191
13                 area_se   0.314139
10               radius_se   0.247305
12            perimeter_se   0.238821
25       compactness_worst   0.218617
5         compactness_mean   0.215539
21           texture_worst   0.144814
16            concavity_se   0.114686
24        smoothness_worst   0.111438
17       concave points_se   0.100033
1             texture_mean   0.097745
4          smoothness_mean   0.087716
28          symmetry_worst   0.079123
8            symmetry_mean   0.065486
29 fractal_dimension_worst   0.063254
30              Hemoglobin   0.059627
15          compactness_se   0.054743
9   fractal_dimension_mean   0.036937
18             symmetry_se   0.023510
33                     MCV   0.022290
14           smoothness_se   0.018233
19    fractal_dimension_se   0.017247
11              texture_se   0.000856
31                     MCH   0.000000
32                   MCH.1   0.000000
Top Features:
['perimeter_worst', 'concave points_mean', 'area_worst', 'concave points_worst', 'radius_worst', 'perimeter_mean', 'concavity_mean', 'area_mean', 'radius_mean', 'concavity_worst']
```

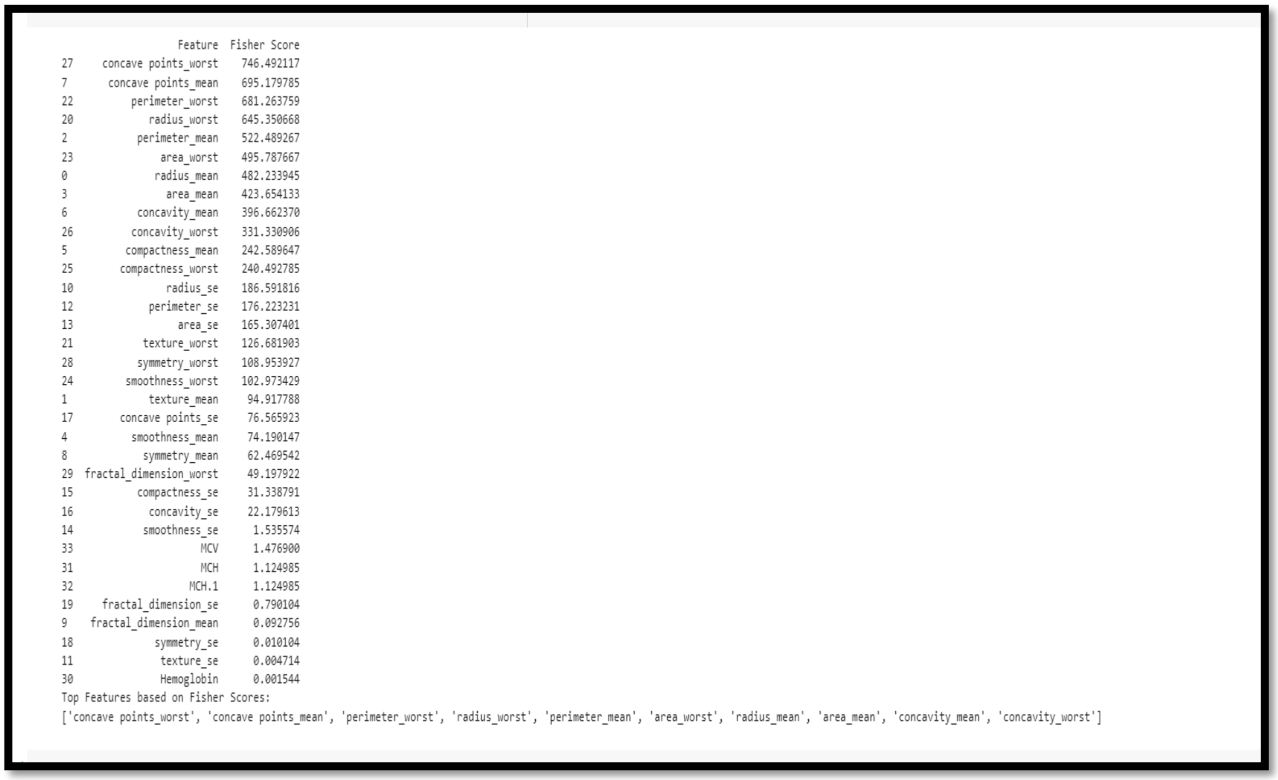
Task#04

When I apply Fisher Score



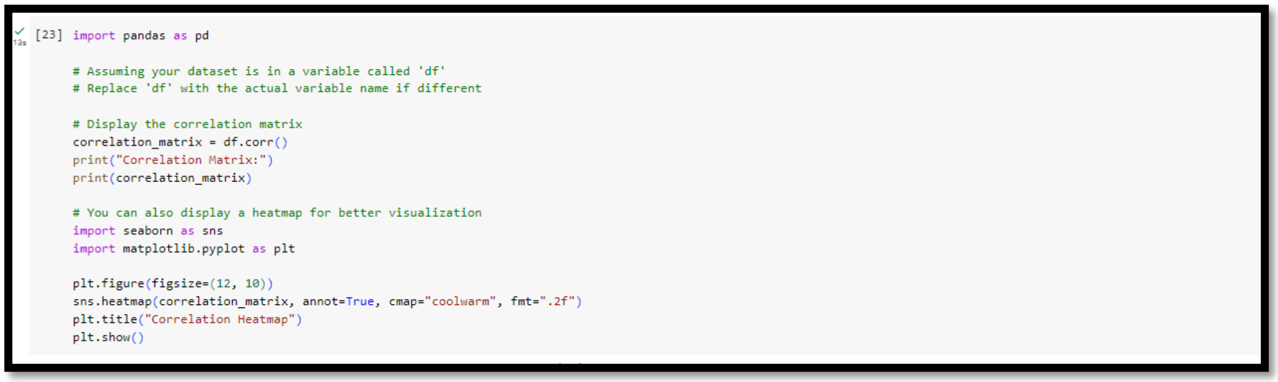
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import f_classif

# Load your dataset
# Assuming your dataset is in a variable called 'df'
# Replace 'df' with the actual variable name if different
# You may need to handle missing values and encode categorical variables if required

# Encode the target variable 'diagnosis'
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis'])

# Split the dataset into features (X) and target variable (y)
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Calculate Fisher scores for each feature
fisher_scores, _ = f_classif(X_train, y_train)

# Create a DataFrame to store feature names and their Fisher scores
feature_fisher_scores = pd.DataFrame({'Feature': X.columns, 'Fisher Score': fisher_scores})

# Sort features by Fisher scores in descending order
feature_fisher_scores = feature_fisher_scores.sort_values(by='Fisher Score', ascending=False)

# Print the feature Fisher scores
print(feature_fisher_scores)

# Select the top N features (you can choose N based on your requirements)
# For example, let's say you want to select the top 10 features
top_features_fisher = feature_fisher_scores.head(10)['Feature'].tolist()

# Display the top selected features
print("Top Features based on Fisher Scores:")
print(top_features_fisher)
```

Output:

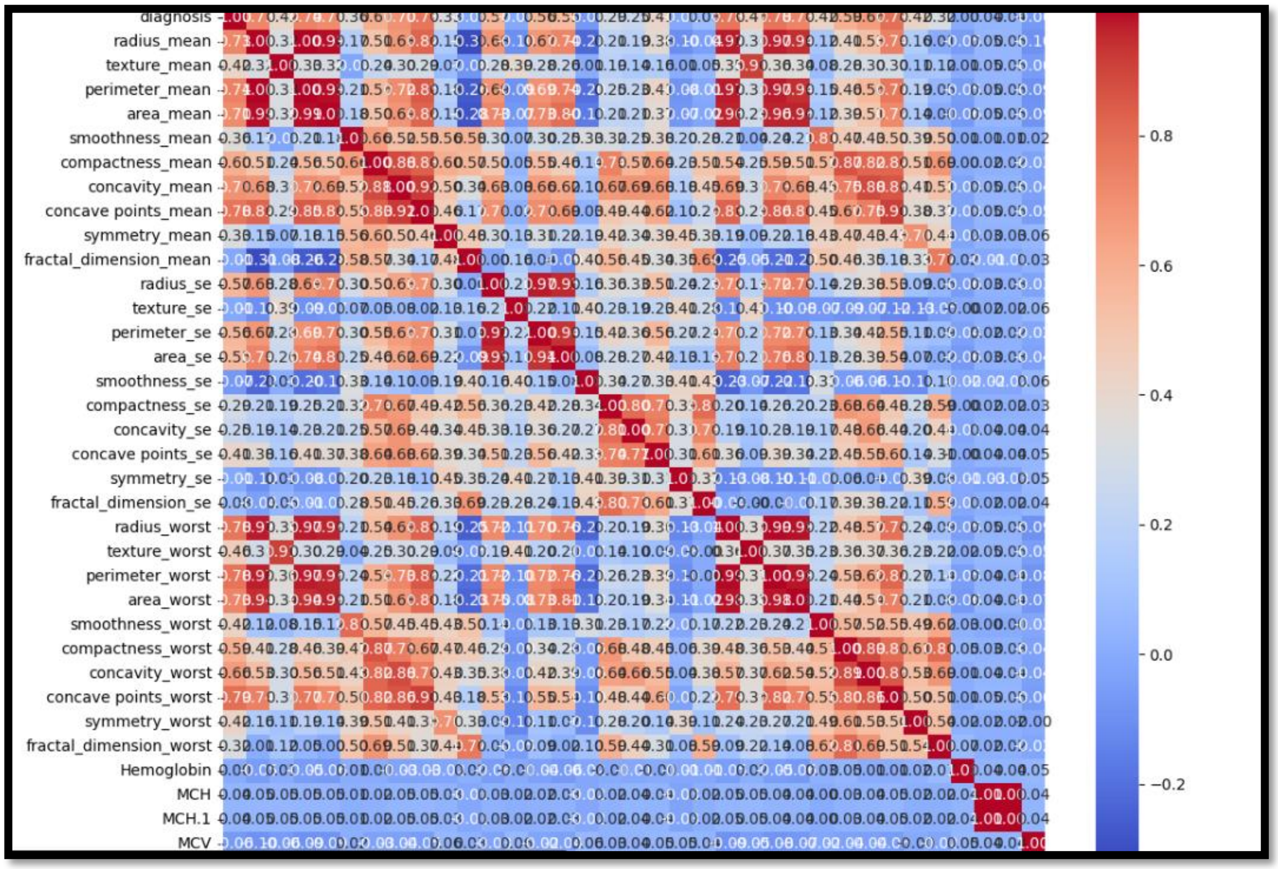


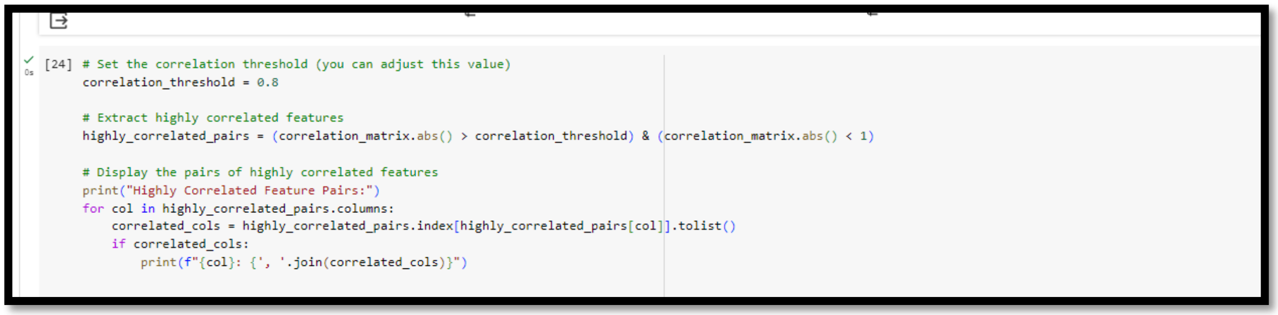
```
                    Feature  Fisher Score
27     concave points_worst    746.492117
7       concave points_mean    695.179785
22          perimeter_worst    681.263759
20             radius_worst    645.350668
2            perimeter_mean    522.489267
23               area_worst    495.787667
0               radius_mean    482.233945
3                 area_mean    423.654133
6            concavity_mean    396.662370
26          concavity_worst    331.330906
5          compactness_mean    242.589647
25        compactness_worst    240.492785
10                radius_se    186.591816
12             perimeter_se    176.223231
13                  area_se    165.307401
21            texture_worst    126.681903
28           symmetry_worst    108.953927
24         smoothness_worst    102.973429
1              texture_mean     94.917788
17        concave points_se     76.565923
4           smoothness_mean     74.190147
8             symmetry_mean     62.469542
29  fractal_dimension_worst     49.197922
15           compactness_se     31.338791
16             concavity_se     22.179613
14            smoothness_se      1.535574
33                      MCV      1.476900
31                      MCH      1.124985
32                    MCH.1      1.124985
19     fractal_dimension_se      0.790104
9    fractal_dimension_mean      0.092756
18              symmetry_se      0.010104
11               texture_se      0.004714
30               Hemoglobin      0.001544
Top Features based on Fisher Scores:
['concave points_worst', 'concave points_mean', 'perimeter_worst', 'radius_worst', 'perimeter_mean', 'area_worst', 'radius_mean', 'area_mean', 'concavity_mean', 'concavity_worst']
```

Correlation Cofficient

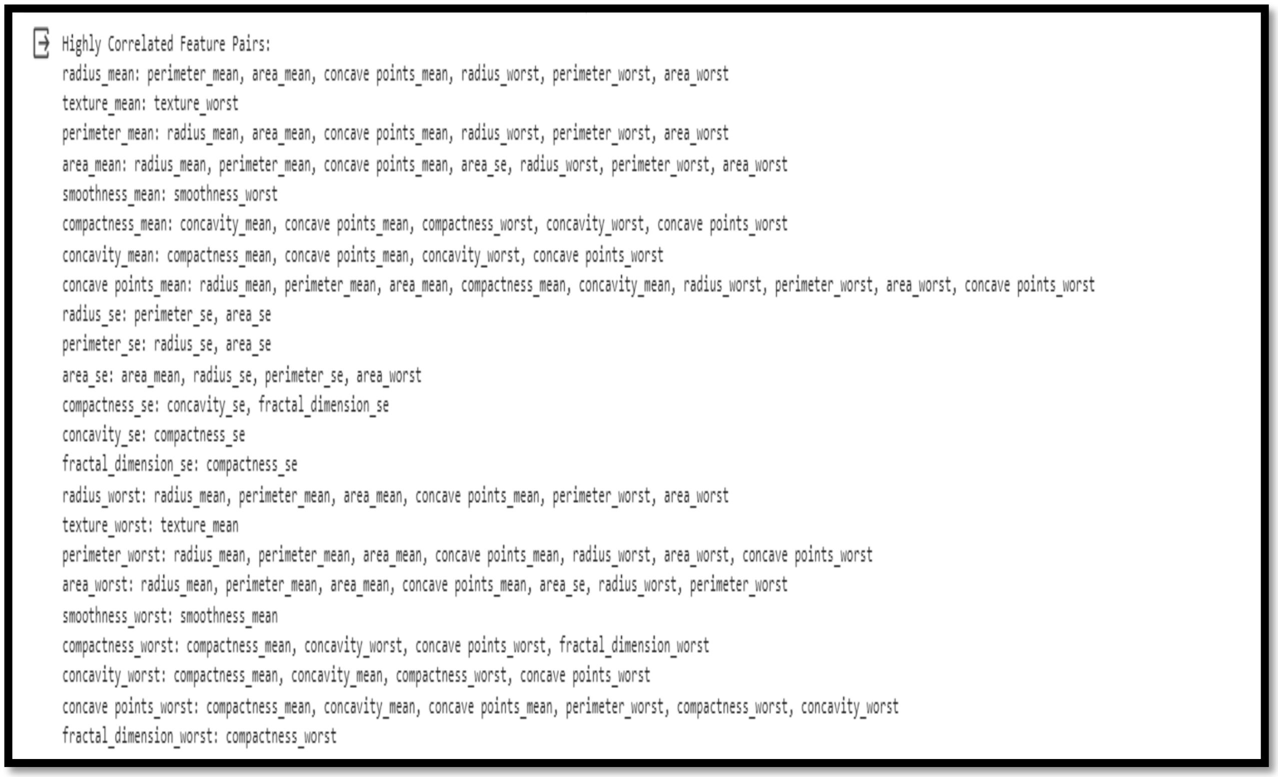


```
[23] import pandas as pd

     # Assuming your dataset is in a variable called 'df'
     # Replace 'df' with the actual variable name if different

     # Display the correlation matrix
     correlation_matrix = df.corr()
     print("Correlation Matrix:")
     print(correlation_matrix)

     # You can also display a heatmap for better visualization
     import seaborn as sns
     import matplotlib.pyplot as plt

     plt.figure(figsize=(12, 10))
     sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
     plt.title("Correlation Heatmap")
     plt.show()
```




```
[24] # Set the correlation threshold (you can adjust this value)
     correlation_threshold = 0.8

     # Extract highly correlated features
     highly_correlated_pairs = (correlation_matrix.abs() > correlation_threshold) & (correlation_matrix.abs() < 1)

     # Display the pairs of highly correlated features
     print("Highly Correlated Feature Pairs:")
     for col in highly_correlated_pairs.columns:
         correlated_cols = highly_correlated_pairs.index[highly_correlated_pairs[col]].tolist()
         if correlated_cols:
             print(f"{col}: {', '.join(correlated_cols)}")
```

Output:

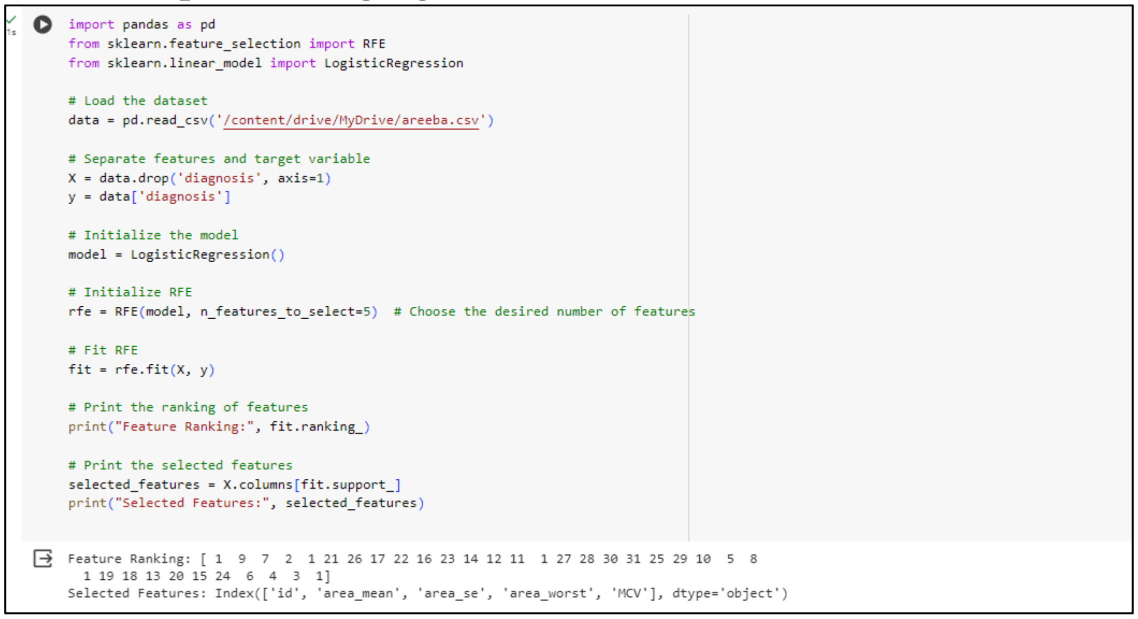


```
Highly Correlated Feature Pairs:
radius_mean: perimeter_mean, area_mean, concave points_mean, radius_worst, perimeter_worst, area_worst
texture_mean: texture_worst
perimeter_mean: radius_mean, area_mean, concave points_mean, radius_worst, perimeter_worst, area_worst
area_mean: radius_mean, perimeter_mean, concave points_mean, area_se, radius_worst, perimeter_worst, area_worst
smoothness_mean: smoothness_worst
compactness_mean: concavity_mean, concave points_mean, compactness_worst, concavity_worst, concave points_worst
concavity_mean: compactness_mean, concave points_mean, concavity_worst, concave points_worst
concave points_mean: radius_mean, perimeter_mean, area_mean, compactness_mean, concavity_mean, radius_worst, perimeter_worst, area_worst, concave points_worst
radius_se: perimeter_se, area_se
perimeter_se: radius_se, area_se
area_se: area_mean, radius_se, perimeter_se, area_worst
compactness_se: concavity_se, fractal_dimension_se
concavity_se: compactness_se
fractal_dimension_se: compactness_se
radius_worst: radius_mean, perimeter_mean, area_mean, concave points_mean, perimeter_worst, area_worst
texture_worst: texture_mean
perimeter_worst: radius_mean, perimeter_mean, area_mean, concave points_mean, radius_worst, area_worst, concave points_worst
area_worst: radius_mean, perimeter_mean, area_mean, concave points_mean, area_se, radius_worst, perimeter_worst
smoothness_worst: smoothness_mean
compactness_worst: compactness_mean, concavity_worst, concave points_worst, fractal_dimension_worst
concavity_worst: compactness_mean, concavity_mean, compactness_worst, concave points_worst
concave points_worst: compactness_mean, concavity_mean, concave points_mean, perimeter_worst, compactness_worst, concavity_worst
fractal_dimension_worst: compactness_worst
```

Wrapper Methods:

Recursive Feature Elimination (RFE):
It involves recursively removing the least important features based on a model's performance until the desired number of features is reached.

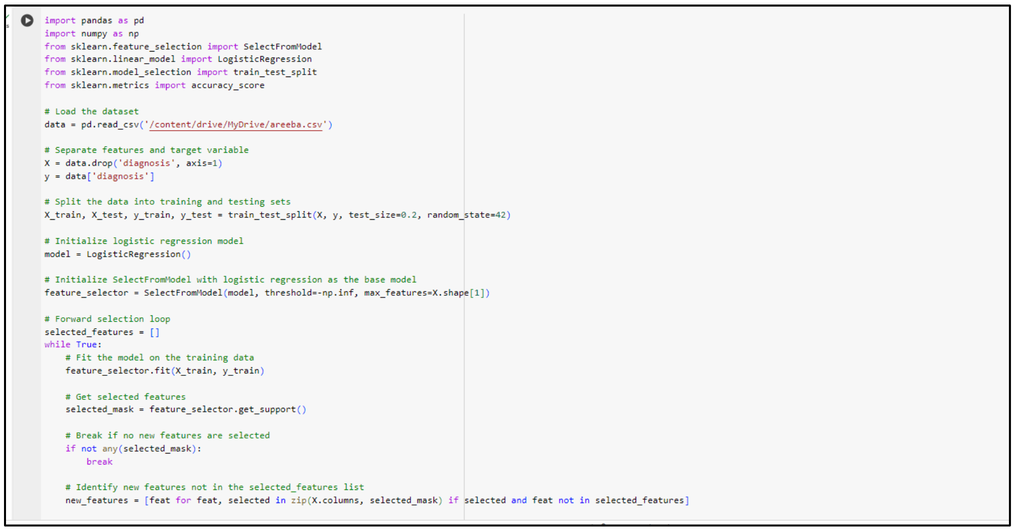
Now lets implement in google colab:



```
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/areeba.csv')

# Separate features and target variable
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']

# Initialize the model
model = LogisticRegression()

# Initialize RFE
rfe = RFE(model, n_features_to_select=5)  # Choose the desired number of features

# Fit RFE
fit = rfe.fit(X, y)

# Print the ranking of features
print("Feature Ranking:", fit.ranking_)

# Print the selected features
selected_features = X.columns[fit.support_]
print("Selected Features:", selected_features)
```

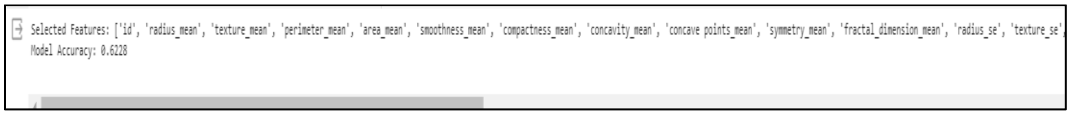
```
Feature Ranking: [ 1  9  7  2  1 21 26 17 22 16 23 14 12 11  1 27 28 30 31 25 29 10  5  8
  1 19 18 13 20 15 24  6  4  3  1]
Selected Features: Index(['id', 'area_mean', 'area_se', 'area_worst', 'MCV'], dtype='object')
```

Forward Selection: It starts with an empty set of features and adds one feature at a time, choosing the feature that provides the best improvement in model performance.



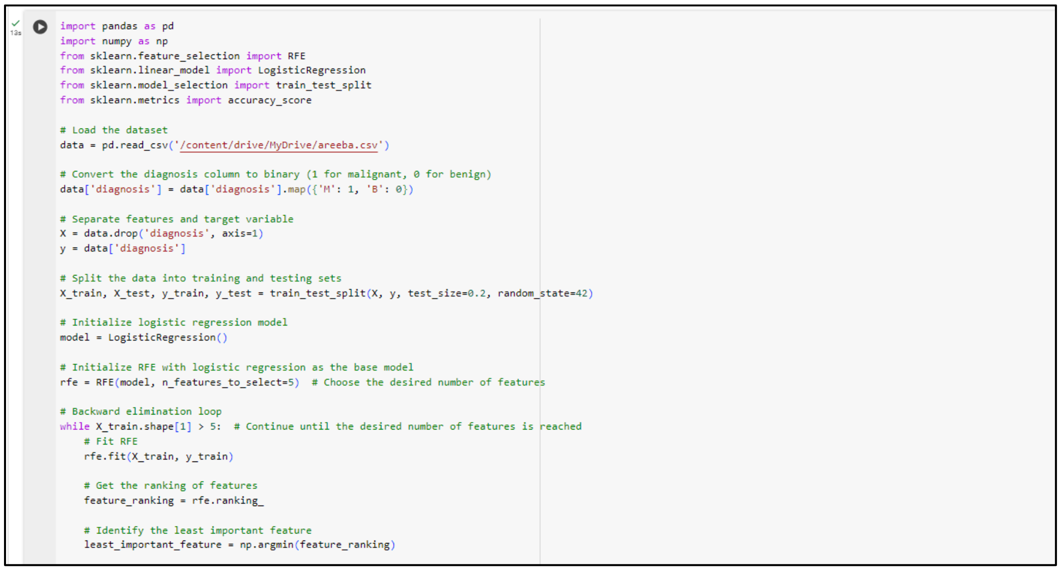
```
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/areeba.csv')

# Separate features and target variable
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize logistic regression model
model = LogisticRegression()

# Initialize SelectFromModel with logistic regression as the base model
feature_selector = SelectFromModel(model, threshold=-np.inf, max_features=X.shape[1])

# Forward selection loop
selected_features = []
while True:
    # Fit the model on the training data
    feature_selector.fit(X_train, y_train)

    # Get selected features
    selected_mask = feature_selector.get_support()

    # Break if no new features are selected
    if not any(selected_mask):
        break

    # Identify new features not in the selected_features list
    new_features = [feat for feat, selected in zip(X.columns, selected_mask) if selected and feat not in selected_features]
```

OUTPUT:

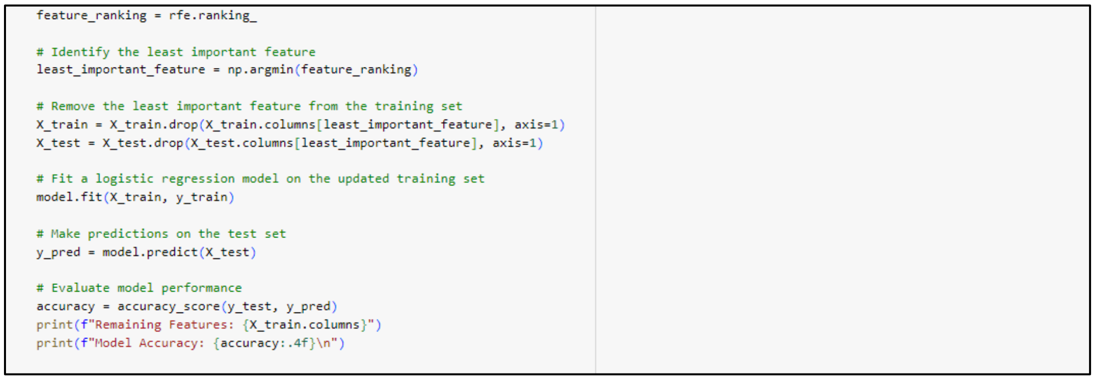


```
Selected Features: ['id', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se',
Model Accuracy: 0.6228
```

Backward Elimination: It starts with all features and removes one at a time, eliminating the least significant features in each iteration.

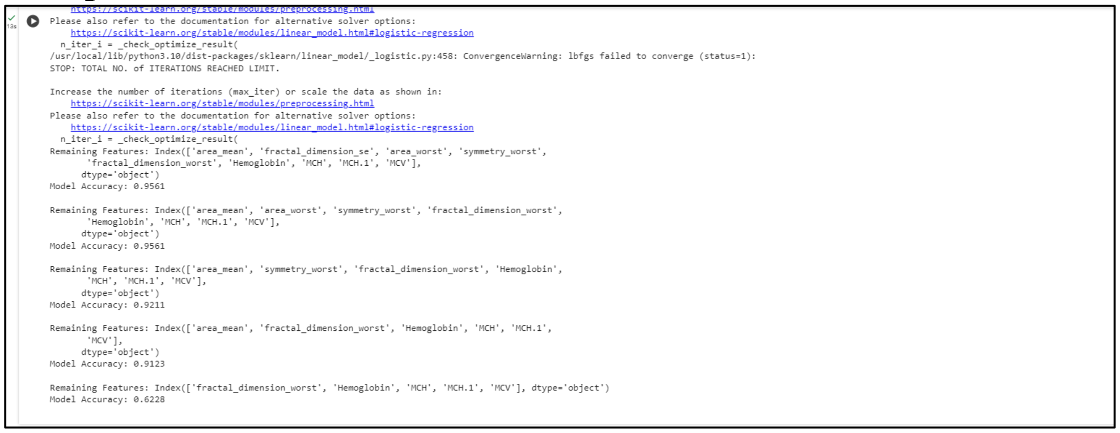


```
import pandas as pd
import numpy as np
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/areeba.csv')

# Convert the diagnosis column to binary (1 for malignant, 0 for benign)
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})

# Separate features and target variable
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize logistic regression model
model = LogisticRegression()

# Initialize RFE with logistic regression as the base model
rfe = RFE(model, n_features_to_select=5)  # Choose the desired number of features

# Backward elimination loop
while X_train.shape[1] > 5:  # Continue until the desired number of features is reached
    # Fit RFE
    rfe.fit(X_train, y_train)

    # Get the ranking of features
    feature_ranking = rfe.ranking_

    # Identify the least important feature
    least_important_feature = np.argmin(feature_ranking)
```



```
feature_ranking = rfe.ranking_

# Identify the least important feature
least_important_feature = np.argmin(feature_ranking)

# Remove the least important feature from the training set
X_train = X_train.drop(X_train.columns[least_important_feature], axis=1)
X_test = X_test.drop(X_test.columns[least_important_feature], axis=1)

# Fit a logistic regression model on the updated training set
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Remaining Features: {X_train.columns}")
print(f"Model Accuracy: {accuracy:.4f}\n")
```

Output:



```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Remaining Features: Index(['area_mean', 'fractal_dimension_se', 'area_worst', 'symmetry_worst',
       'fractal_dimension_worst', 'Hemoglobin', 'MCH', 'MCH.1', 'MCV'],
      dtype='object')
Model Accuracy: 0.9561

Remaining Features: Index(['area_mean', 'area_worst', 'symmetry_worst', 'fractal_dimension_worst',
       'Hemoglobin', 'MCH', 'MCH.1', 'MCV'],
      dtype='object')
Model Accuracy: 0.9561

Remaining Features: Index(['area_mean', 'symmetry_worst', 'fractal_dimension_worst', 'Hemoglobin',
       'MCH', 'MCH.1', 'MCV'],
      dtype='object')
Model Accuracy: 0.9211

Remaining Features: Index(['area_mean', 'fractal_dimension_worst', 'Hemoglobin', 'MCH', 'MCH.1',
       'MCV'],
      dtype='object')
Model Accuracy: 0.9123

Remaining Features: Index(['fractal_dimension_worst', 'Hemoglobin', 'MCH', 'MCH.1', 'MCV'], dtype='object')
Model Accuracy: 0.6228
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Remaining Features: Index(['area_mean', 'fractal_dimension_se', 'area_worst', 'symmetry_worst',
       'fractal_dimension_worst', 'Hemoglobin', 'MCH', 'MCH.1', 'MCV'],
      dtype='object')
Model Accuracy: 0.9561

Remaining Features: Index(['area_mean', 'area_worst', 'symmetry_worst', 'fractal_dimension_worst',
       'Hemoglobin', 'MCH', 'MCH.1', 'MCV'],
      dtype='object')
Model Accuracy: 0.9561

Remaining Features: Index(['area_mean', 'symmetry_worst', 'fractal_dimension_worst', 'Hemoglobin',
       'MCH', 'MCH.1', 'MCV'],
      dtype='object')
Model Accuracy: 0.9211

Remaining Features: Index(['area_mean', 'fractal_dimension_worst', 'Hemoglobin', 'MCH', 'MCH.1',
       'MCV'],
      dtype='object')
Model Accuracy: 0.9123

Remaining Features: Index(['fractal_dimension_worst', 'Hemoglobin', 'MCH', 'MCH.1', 'MCV'], dtype='object')
Model Accuracy: 0.6228
```

Lab Task 10

Task#1

Q:1 What's the risk with tuning hyper parameters using a test dataset? What are the challenges and best practices of hyper parameter tuning?

Tuning hyperparameters using a test dataset is a common practice in machine learning, but it can lead to overfitting and biased results. Here's why:

Overfitting: The primary risk of tuning hyperparameters on a test dataset is overfitting. Overfitting occurs when a model learns the specific patterns and noise in the test data too closely, making it perform well on that particular dataset but poorly on new, unseen data. This is because the model becomes too specialized to the test set and loses its ability to generalize to other data.

Biased Results: Using the test dataset for hyperparameter tuning can introduce bias into the evaluation process. Since the model is trained and evaluated on the same data, it may perform better on that specific dataset than it would on new data. This bias can lead to an overly optimistic assessment of the model's performance and make it challenging to compare different models fairly.

Reduced Generalization: The goal of hyperparameter tuning is to find the best set of parameters that will enable the model to perform well on unseen data. By tuning hyperparameters on the test dataset, you are essentially "fitting" the model to that specific data, which reduces its ability to generalize to new situations and data distributions.

Suggestion sShow:

To avoid these issues, it's recommended to use a separate validation set for hyperparameter tuning. The validation set should be a representative sample of the data that is distinct from the training and test sets. By tuning hyperparameters on the validation set, you can mitigate the risk of overfitting and obtain a more accurate assessment of the model's performance.

Challenges of Hyper parameter Tuning:

Computational Cost: Trying different combinations of hyperparameters can be computationally expensive, especially for complex models or large datasets.

Overfitting: Tuning hyperparameters can lead to overfitting, where the model performs well on the training data but poorly on new data.

Local Minima: Optimization algorithms used for hyperparameter tuning can get stuck in local minima, leading to suboptimal results.

Lack of Interpretability: It can be challenging to understand how different hyperparameters affect the model's performance, making it difficult to make informed decisions.

Best Practices for Hyper parameter Tuning:

Start with Default Values: Begin with the default values recommended by the model or algorithm you are using.

Use a Structured Approach: Systematically vary one hyperparameter at a time while keeping others constant. This helps identify the impact of each parameter.

Leverage Automation: Use automated tools or libraries for hyperparameter tuning, such as grid search, random search, or Bayesian optimization.

Cross-Validation: Use cross-validation to evaluate the performance of different hyperparameter combinations and avoid overfitting.

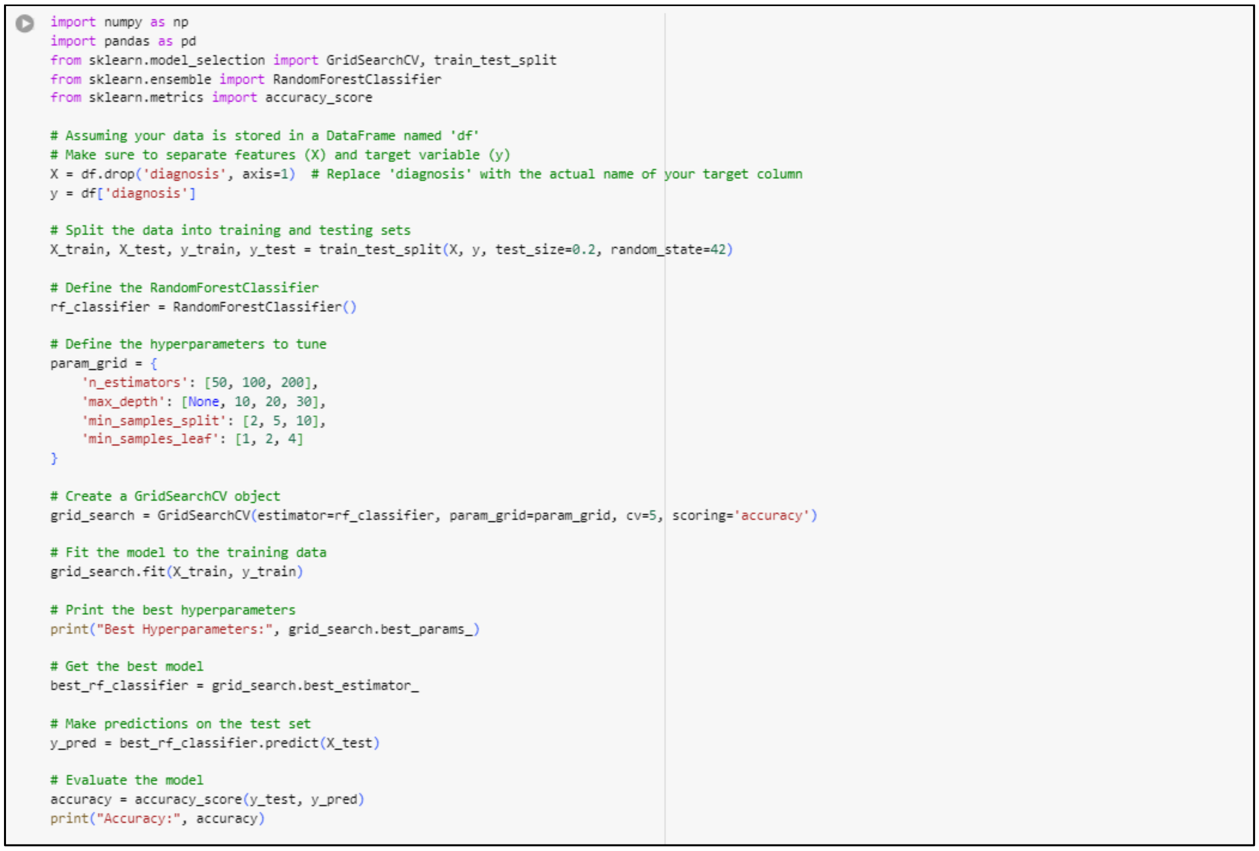
Early Stopping: Implement early stopping to prevent overfitting by terminating the training process when the model's performance on the validation set starts to decline.

Ensemble Methods: Consider using ensemble methods, such as random forests or gradient boosting, which are less sensitive to hyperparameter choice

Transfer Learning: If possible, leverage pre-trained models or transfer learning to reduce the need for extensive hyperparameter tuning.

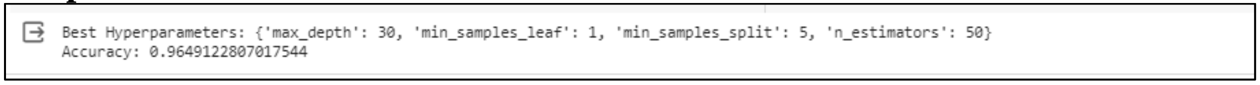
Task#2 Implement Grid Search on:

Random Forest Classifier



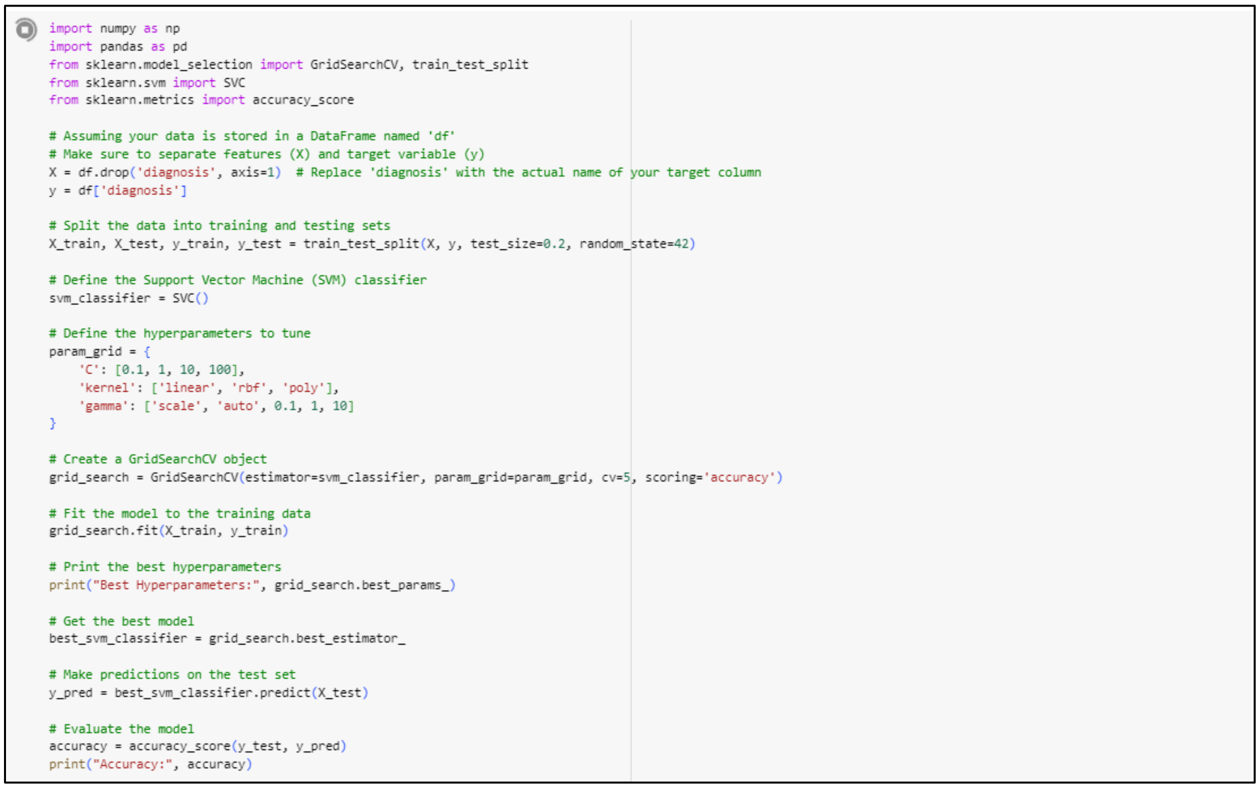
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming your data is stored in a DataFrame named 'df'
# Make sure to separate features (X) and target variable (y)
X = df.drop('diagnosis', axis=1)  # Replace 'diagnosis' with the actual name of your target column
y = df['diagnosis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the RandomForestClassifier
rf_classifier = RandomForestClassifier()

# Define the hyperparameters to tune
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model
best_rf_classifier = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Output:



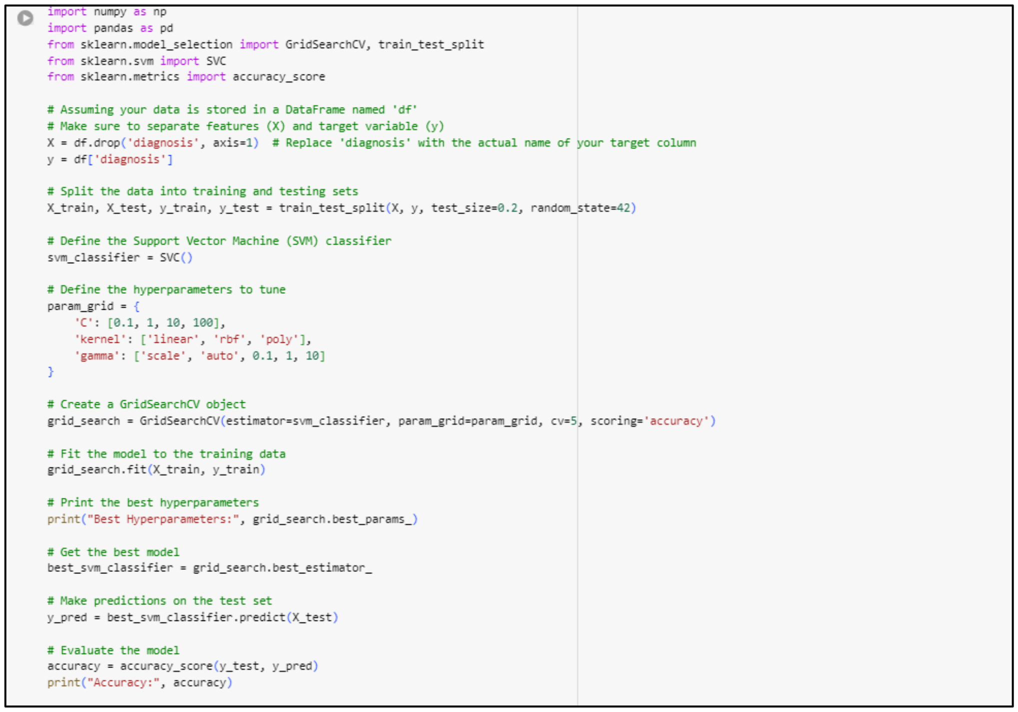
```
Best Hyperparameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 50}
Accuracy: 0.9649122807017544
```

- **Support Vector Machine**



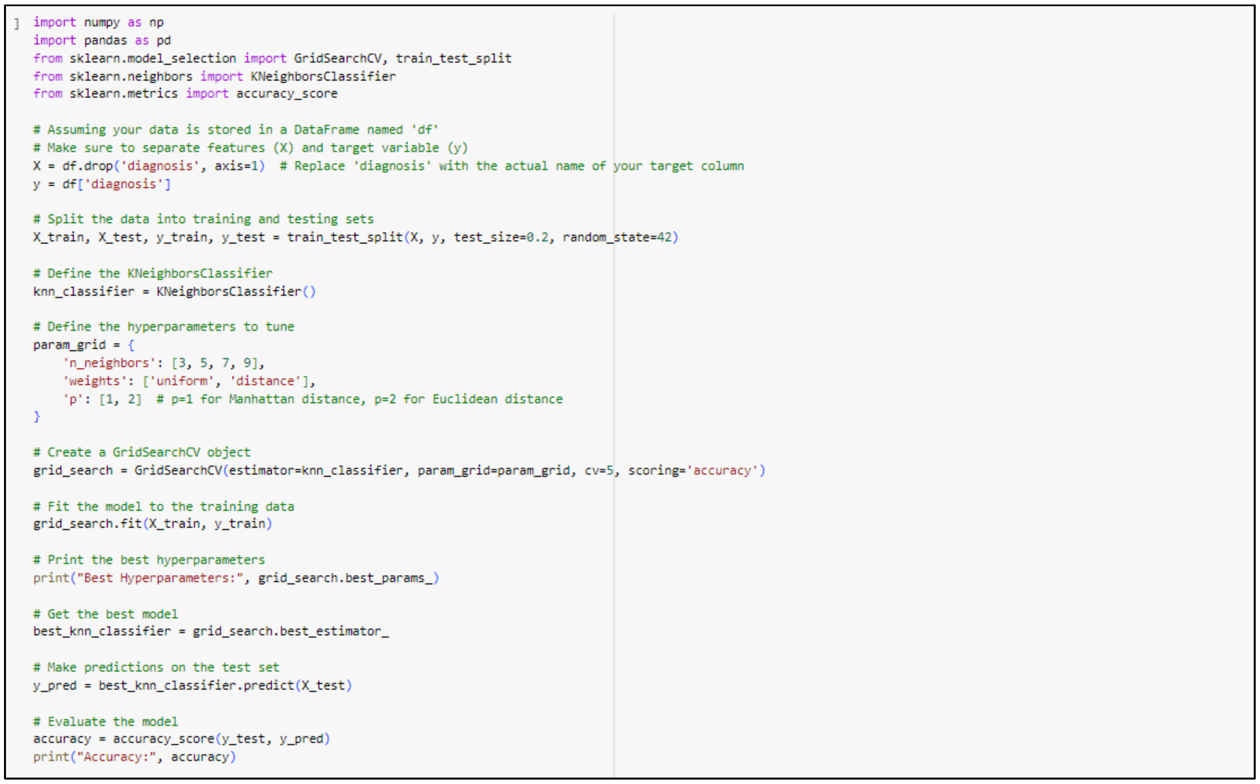
```
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Assuming your data is stored in a DataFrame named 'df'
# Make sure to separate features (X) and target variable (y)
X = df.drop('diagnosis', axis=1)  # Replace 'diagnosis' with the actual name of your target column
y = df['diagnosis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the Support Vector Machine (SVM) classifier
svm_classifier = SVC()

# Define the hyperparameters to tune
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto', 0.1, 1, 10]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=svm_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model
best_svm_classifier = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_svm_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

- **Logistic Regression**



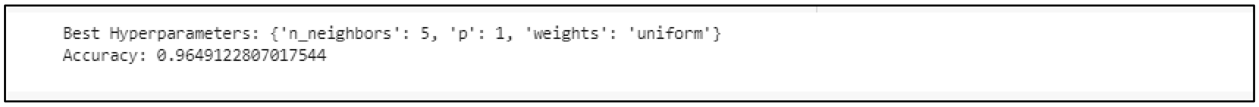
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Assuming your data is stored in a DataFrame named 'df'
# Make sure to separate features (X) and target variable (y)
X = df.drop('diagnosis', axis=1)  # Replace 'diagnosis' with the actual name of your target column
y = df['diagnosis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the Support Vector Machine (SVM) classifier
svm_classifier = SVC()

# Define the hyperparameters to tune
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto', 0.1, 1, 10]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=svm_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model
best_svm_classifier = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_svm_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

- **K-Nearest Neighbor**



```python
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Assuming your data is stored in a DataFrame named 'df'
# Make sure to separate features (X) and target variable (y)
X = df.drop('diagnosis', axis=1)  # Replace 'diagnosis' with the actual name of your target column
y = df['diagnosis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the KNeighborsClassifier
knn_classifier = KNeighborsClassifier()

# Define the hyperparameters to tune
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]  # p=1 for Manhattan distance, p=2 for Euclidean distance
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=knn_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model
best_knn_classifier = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_knn_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Output:



```
Best Hyperparameters: {'n_neighbors': 5, 'p': 1, 'weights': 'uniform'}
Accuracy: 0.9649122807017544
```