



Introduction to Machine Learning

Phase One

Name	ID
AbdulRahman AlSaeed Ahmed (Breastcancer)	19P7808
Seif Eldin Amr Mostafa Yassine (Iphone)	20P2006
Sherwet Mohamed Khalil Barakat (Loan)	20P8105

Contents

Breast Cancer Diagnosis:	3
• Data Loading and Preprocessing	3
• Training and Testing Split	3
• Classifier Training and Initial Testing.	3
• Hyperparameter Tuning for KNN	3
• Results Visualization	3
• Evaluation and Analysis	3
Loan Dataset:	14
• Initialize, import and reading of CSV dataset	14
• Dropping first two columns	14
• Applying Label Encoding	16
• Scaling ranging from -1 to 1	18
• Display the shapes of the training and testing sets after splitting them.	19
• The Reason of using the final parameters	21
iPhone Purchase:	23
• Loading the file	23
• Before preprocessing	23
• Preprocessing stage	23
• Training stage:	24
• Testing stage	25
• Screenshots of Output:	26
• Final Values:	27

Breast Cancer Diagnosis:

- **Data Loading and Preprocessing:**

We initiated our project by loading the `breast_cancer_diagnosis.csv` dataset using the Pandas library. Our preprocessing involved removing columns that would not contribute to the classifiers' performance – specifically, the 'id' and 'Unnamed: 32' columns. Subsequently, we encoded the 'diagnosis' column to reflect binary values, with 'M' (malignant) as 1 and 'B' (benign) as 0, preparing the data for the next stage of our analysis.

- **Training and Testing Split:**

Our team split the pre-processed data into a training set (70%) and a testing set (30%) using Scikit learn's `train_test_split` function. This division was crucial for training our models and evaluating their predictive performance on unseen data.

- **Classifier Training and Initial Testing:**

The Naïve Bayes classifier was trained using the `GaussianNB` implementation, chosen for its suitability for continuous input features. The initial testing yielded an impressive accuracy of 94%. The KNN classifier began with an initial parameter of $k=5$ neighbours, reflecting a commonly chosen starting point, and achieved an initial accuracy of 96%.

- **Hyperparameter Tuning for KNN:**

Our team conducted hyperparameter tuning on the KNN classifier to explore a range of k values from 1 to 20. This process aimed to optimize the model by finding the k value that afforded the highest accuracy on our test dataset.

- **Results Visualization:**

We plotted the accuracy of the KNN classifier against the number of neighbours, resulting in a visual representation that clearly showed how the choice of k influenced performance. The plot indicated that accuracy peaked at several points, with diminishing returns as k increased beyond a certain threshold.

- **Evaluation and Analysis:**

After rigorous testing, our analysis showed that while both classifiers performed well, the Naïve Bayes classifier displayed slightly superior accuracy. The KNN classifier's performance varied with different k values, suggesting that an optimal k value exists that balances bias and variance to prevent underfitting or overfitting.

Python Code:

- Initiation of dataset and Reading the CSV:

The image shows two separate sessions of Python code execution in Google Colab. Both sessions are titled "BreastCancer.ipynb".

Session 1 (Top):

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.neighbors import KNeighborsClassifier
6 from sklearn.model_selection import train_test_split
7 from sklearn.naive_bayes import GaussianNB
8 import seaborn as sns
9 from sklearn.preprocessing import LabelEncoder
```

[68]: 1 df1 = pd.read_csv('breast_cancer_diagnosis.csv')
2 df1.head()

	id	diagnosis	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	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.	0.	0.	0.	18.55	15.95	132.95	1001.05	0.11840	0.27760	0.	0.	0.	0.	18.55	15.95	132.95	1001.05	0.11840	0.27760	0.	0.	0.	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.08474	0.07864	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.08474	0.07864	0.	0.	0.	0.
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.10960	0.15990	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.10960	0.15990	0.	0.	0.	0.

Disk 81.68 GB available

Session 2 (Bottom):

```
7 from sklearn.naive_bayes import GaussianNB
8 import seaborn as sns
9 from sklearn.preprocessing import LabelEncoder
```

[68]: 1 df1 = pd.read_csv('breast_cancer_diagnosis.csv')
2 df1.head()

	id	diagnosis	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	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.	0.	0.	0.	18.55	15.95	132.95	1001.05	0.11840	0.27760	0.	0.	0.	0.	18.55	15.95	132.95	1001.05	0.11840	0.27760	0.	0.	0.	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.08474	0.07864	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.08474	0.07864	0.	0.	0.	0.
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.10960	0.15990	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.10960	0.15990	0.	0.	0.	0.
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.	0.	0.	0.	18.55	15.95	132.95	1326.05	0.14250	0.28390	0.	0.	0.	0.	18.55	15.95	132.95	1326.05	0.14250	0.28390	0.	0.	0.	0.
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.10030	0.13280	0.	0.	0.	0.	21.55	19.36	132.95	1326.05	0.10030	0.13280	0.	0.	0.	0.

5 rows × 33 columns

Disk 81.68 GB available

The screenshot shows a Google Colab notebook titled "BreastCancer.ipynb". The code cell [68] contains the command `df1.columns.astype`. The output shows the definition of the `astype` method from the `pandas.core.indexes.base.Index` class, which creates an Index with values cast to dtypes. The code cell [70] contains the command `df1 = df1.drop(columns= ['id' , 'Unnamed: 32'] , axis=1)` followed by `df1.head()`. The output displays the first five rows of the dataset, which has 5 rows and 33 columns. The columns listed are diagnosis, radius_mean, texture_mean, perimeter_mean, area_mean, smoothness_mean, compactness_mean, concavity_mean, point, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31. The "Files" sidebar shows a folder named "sample_data" containing "breast_cancer_d...". The status bar at the bottom indicates "0s completed at 3:57PM".

```
[68]: df1.columns.astype
pandas.core.indexes.base.Index.astype
def astype(dtype, copy: bool=True):
    /usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py
Create an Index with values cast to dtypes.

The class of a new Index is determined by dtype. When conversion is
impossible, a TypeError exception is raised.

[70]: df1 = df1.drop(columns= ['id' , 'Unnamed: 32'] , axis=1)
       df1.head()

diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  compactness_mean  concavity_mean  point
0          M           17.99         10.38        122.80     1001.0      0.11840        0.27760      0.3001
1          M           20.57         17.77        132.90     1326.0      0.08474        0.07864      0.0869
2          M           19.69         21.25        130.00     1203.0      0.10960        0.15990      0.1974
3          M           11.42         20.38        77.58       386.1      0.14250        0.28390      0.2414
4          M           20.29         14.34        135.10      1297.0      0.10030        0.13280      0.1980
```

- Using Label Encoder:

The screenshot shows two nearly identical code snippets in a Google Colab notebook titled "BreastCancer.ipynb". Both snippets perform the same operations: importing the LabelEncoder module and applying it to the 'diagnosis' column of the DataFrame. The difference lies in the target variable used for fitting the encoder.

Code Snippet 1 (Top):

```
[71]: 1 LabelEncoder = LabelEncoder()
2 df1['diagnosis'] = LabelEncoder.fit_transform(df1['diagnosis'])
```

Code Snippet 2 (Bottom):

```
[71]: 1 LabelEncoder = LabelEncoder()
2 df1['diagnosis'] = LabelEncoder.fit_transform(df1['diagnosis'])

[72]: 1 df1.head()
```

Data Preview:

The output of both code snippets is a preview of the first few rows of the DataFrame, showing the columns: 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', and 'concave points_worst'. The values for these columns are identical to those shown in the first snippet, which used 'concave points_worst' as the target variable.

radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst
25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654
24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860
23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430
14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575
22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625

The screenshot shows a Google Colab notebook titled "BreastCancer.ipynb". The notebook has two code cells:

```
[71]: 1 LabelEncoder = LabelEncoder()
2 df1['diagnosis'] = LabelEncoder.fit_transform(df1['diagnosis'])

[72]: 1 df1.head()
```

Cell [71] encodes the 'diagnosis' column using a LabelEncoder. Cell [72] displays the first five rows of the DataFrame.

Output of Cell [71]:

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	diagnosis
0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	1	100.500	1
1	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	1	156.500	1
2	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	1	175.300	1
3	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	1	184.500	1
4	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	1	201.000	1

Output of Cell [72]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	points_mean	symmetry_mean	perimeter_worst	area_worst	diagnosis
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	1	100.500	1
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	1	156.500	1
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	1	175.300	1
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	1	184.500	1
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	1	201.000	1

Disk usage: 81.68 GB available

- Checking if there is any error:

```

symmetry_mean      0
fractal_dimension_mean 0
radius_se          0
texture_se          0
perimeter_se        0
area_se             0
smoothness_se       0
compactness_se      0
concavity_se        0
concave_points_se   0
symmetry_se         0
fractal_dimension_se 0
radius_worst        0
texture_worst        0
perimeter_worst     0
area_worst           0
smoothness_worst    0
compactness_worst   0
concavity_worst     0
concave_points_worst 0
symmetry_worst      0
fractal_dimension_worst 0
dtype: int64

```

0s completed at 3:57PM

```

[73] 1 df1.isna().sum()
diagnosis          0
radius_mean         0
texture_mean        0
perimeter_mean      0
area_mean            0
smoothness_mean     0
compactness_mean    0
concavity_mean      0
concave_points_mean 0
symmetry_mean        0
fractal_dimension_mean 0
radius_se           0
texture_se           0
perimeter_se         0
area_se              0
smoothness_se        0
compactness_se       0
concavity_se         0
concave_points_se    0
symmetry_se          0
fractal_dimension_se 0
radius_worst         0

```

0s completed at 3:57PM

- Diagnosis using Mean:

```

colab.google          BreastCancer.ipynb - Colab      ML Project.ipynb - Colab      main.ipynb - Colab      Loan_dataset.ipynb - Colab      +
colab.research.google.com/drive/13cMnS2aYki8tXEmfm3MYAUMWLSqp58E#scrollTo=xPXRiRwZyv3v&uniquifier=1
Gmail YouTube Maps SIS, Faculty of Engin... Ain Shams Universit... Instagram CESS 2025 - Google... Computer 2021-202... LectureVideos - Go...
All Bookmarks

BreastCancer.ipynb
File Edit View Insert Runtime Tools Help All changes saved
RAM Disk Colab AI
+ Code + Text
{x}
df1.groupby('diagnosis').mean()
radius_mean texture_mean perimeter_mean area_mean smoothness_mean compactness_mean concavity_mean points_mean
diagnosis
0 12.146524 17.914762 78.075406 462.790196 0.092478 0.080085 0.046058 0.0
1 17.462830 21.604906 115.365377 978.376415 0.102898 0.145188 0.160775 0.0
2 rows × 30 columns
[75]: 1 scaler = StandardScaler()
2 scaled_features_cancer1 = scaler.fit_transform(df1.drop('diagnosis', axis=1))
3 df1_feat1 = pd.DataFrame(scaled_features_cancer1)
[76]: 1 X1 = df1_feat1
2 y1 = df1['diagnosis']

0s completed at 3:57 PM
Disk 81.68 GB available

```

- Train Test Cell:

```

colab.google          BreastCancer.ipynb - Colab      ML Project.ipynb - Colab      main.ipynb - Colab      Loan_dataset.ipynb - Colab      +
colab.research.google.com/drive/13cMnS2aYki8tXEmfm3MYAUMWLSqp58E#scrollTo=xPXRiRwZyv3v&uniquifier=1
Gmail YouTube Maps SIS, Faculty of Engin... Ain Shams Universit... Instagram CESS 2025 - Google... Computer 2021-202... LectureVideos - Go...
All Bookmarks

BreastCancer.ipynb
File Edit View Insert Runtime Tools Help All changes saved
RAM Disk Colab AI
+ Code + Text
Train Test Cell
{x}
[77]: 1 X1_train , X1_test , y1_train , y1_test = train_test_split(X1,y1 , train_size= 0.8 , random_state= 100 )
Pairplot
[78]: 1 sns.pairplot(df1 , hue= 'diagnosis' )


```

- Naïve Bayes Classifier:

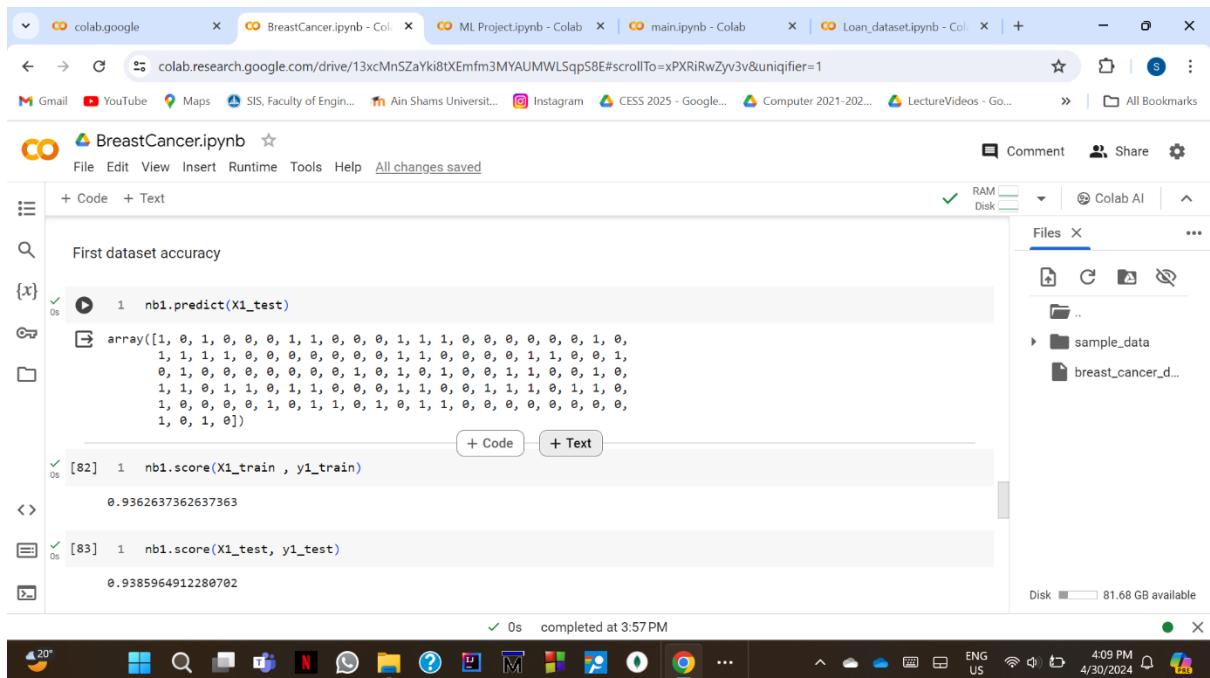
```

colab.google          BreastCancer.ipynb - Colab      ML Project.ipynb - Colab      main.ipynb - Colab      Loan_dataset.ipynb - Colab      +
colab.research.google.com/drive/13cMnS2aYki8tXEmfm3MYAUMWLSqp58E#scrollTo=xPXRiRwZyv3v&uniquifier=1
Gmail YouTube Maps SIS, Faculty of Engin... Ain Shams Universit... Instagram CESS 2025 - Google... Computer 2021-202... LectureVideos - Go...
All Bookmarks

BreastCancer.ipynb
File Edit View Insert Runtime Tools Help All changes saved
RAM Disk Colab AI
+ Code + Text
Naive Bayes Classifier
{x}
[79]: 1 nb1 = GaussianNB()
nb1
[80]: 1 nb1.fit(X1_train , y1_train)
nb1
GaussianNB()
GaussianNB()


```

- First dataset Accuracy:



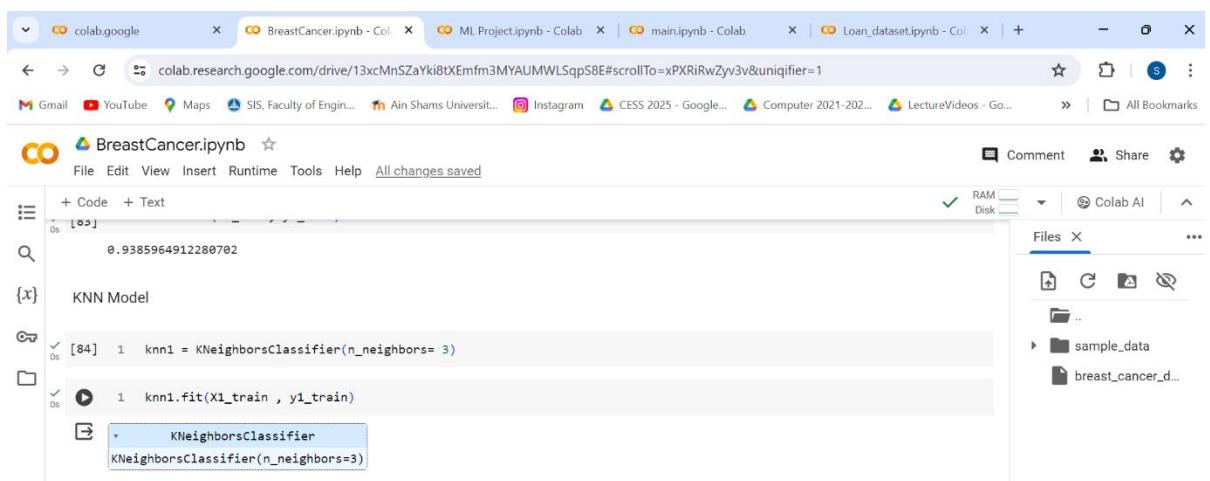
The screenshot shows a Google Colab notebook titled "BreastCancer.ipynb". The code cell [82] contains the command `nb1.score(X1_train, y1_train)` which returns the value 0.9362637362637363. The code cell [83] contains the command `nb1.score(X1_test, y1_test)` which returns the value 0.9385964912280702. The sidebar on the right shows a "Files" section with a "sample_data" folder containing a "breast_cancer_d..." file.

```

nb1.predict(X1_test)
array([1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0])
[82]: nb1.score(X1_train, y1_train)
0.9362637362637363
[83]: nb1.score(X1_test, y1_test)
0.9385964912280702

```

- KNN Model:



The screenshot shows a Google Colab notebook titled "BreastCancer.ipynb". The code cell [84] contains the command `knn1 = KNeighborsClassifier(n_neighbors= 3)` followed by `knn1.fit(X1_train, y1_train)`. A tooltip for the `KNeighborsClassifier` class is visible. The sidebar on the right shows a "Files" section with a "sample_data" folder containing a "breast_cancer_d..." file.

```

knn1 = KNeighborsClassifier(n_neighbors= 3)
knn1.fit(X1_train, y1_train)

```

- Dataset Accuracy:

The screenshot shows a Google Colab notebook titled "BreastCancer.ipynb". The code cell [86] contains the command `knn1.predict(X1_test)` followed by its output, which is a large array of binary values (0s and 1s). The code cell [87] contains the command `knn1.score(X1_train, y1_train)` with the output `0.9846153846153847`. The code cell [88] contains the command `knn1.score(X1_test, y1_test)` with the output `0.9649122807017544`. The sidebar on the right shows the "Files" section with a folder named "sample_data" containing "breast_cancer_d...". The bottom status bar indicates "Disk 81.68 GB available". The taskbar at the bottom shows various application icons.

```
[86]: knn1.predict(X1_test)
array([1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0])
[87]: knn1.score(X1_train, y1_train)
0.9846153846153847
[88]: knn1.score(X1_test, y1_test)
0.9649122807017544
```

Another Python Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset

data = pd.read_csv('/mnt/data/breast_cancer_diagnosis.csv')

# Preprocessing
data_cleaned = data.drop(['id', 'Unnamed: 32'], axis=1)
data_cleaned['diagnosis'] = data_cleaned['diagnosis'].map({'M': 1, 'B': 0})

# Splitting the data
features = data_cleaned.drop('diagnosis', axis=1)
target = data_cleaned['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3,
random_state=42)

# Naïve Bayes Classifier
```

```

nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_predictions = nb_classifier.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print(f"Naïve Bayes Accuracy: {nb_accuracy:.2f}")

# KNN Classifier with initial k=5

knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print(f"Initial KNN Accuracy (k=5): {knn_accuracy:.2f}")

# Hyperparameter Tuning for KNN

k_values = range(1, 21)
knn_accuracies = []
for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    knn_predictions = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, knn_predictions)
    knn_accuracies.append(accuracy)

# Plotting the KNN accuracies

plt.figure(figsize=(10, 6))
plt.plot(k_values, knn_accuracies, marker='o', linestyle='-', color='b')
plt.title('KNN Accuracy vs. Number of Neighbors')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

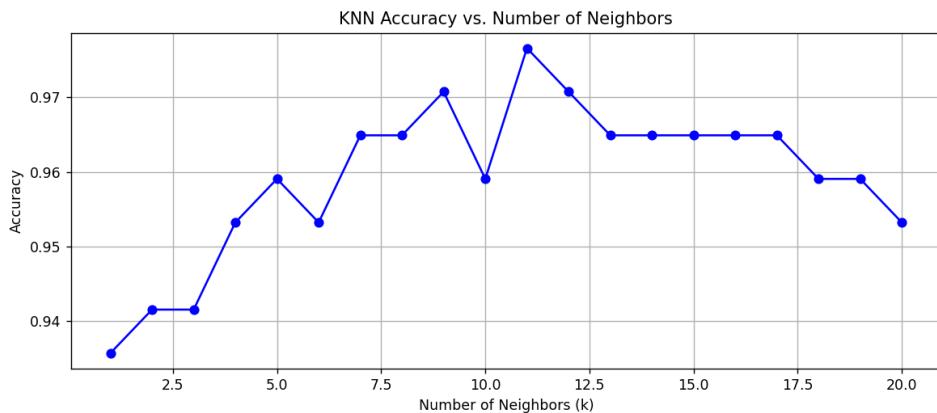
```

Results:

```
(c) Microsoft Corporation. All rights reserved.  
C:\Users\dell>cd Desktop  
  
C:\Users\dell\Desktop>python breastcancer.py  
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xef in position 2: invalid continuation byte  
  
C:\Users\dell\Desktop>python breastcancer.py  
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xef in position 2: invalid continuation byte  
  
C:\Users\dell\Desktop>python breastcancer.py  
Naïve Bayes Accuracy: 0.94  
Initial KNN Accuracy (k=5): 0.96
```

```
Naïve Bayes Accuracy: 0.94  
Initial KNN Accuracy (k=5): 0.96
```

Figure 1

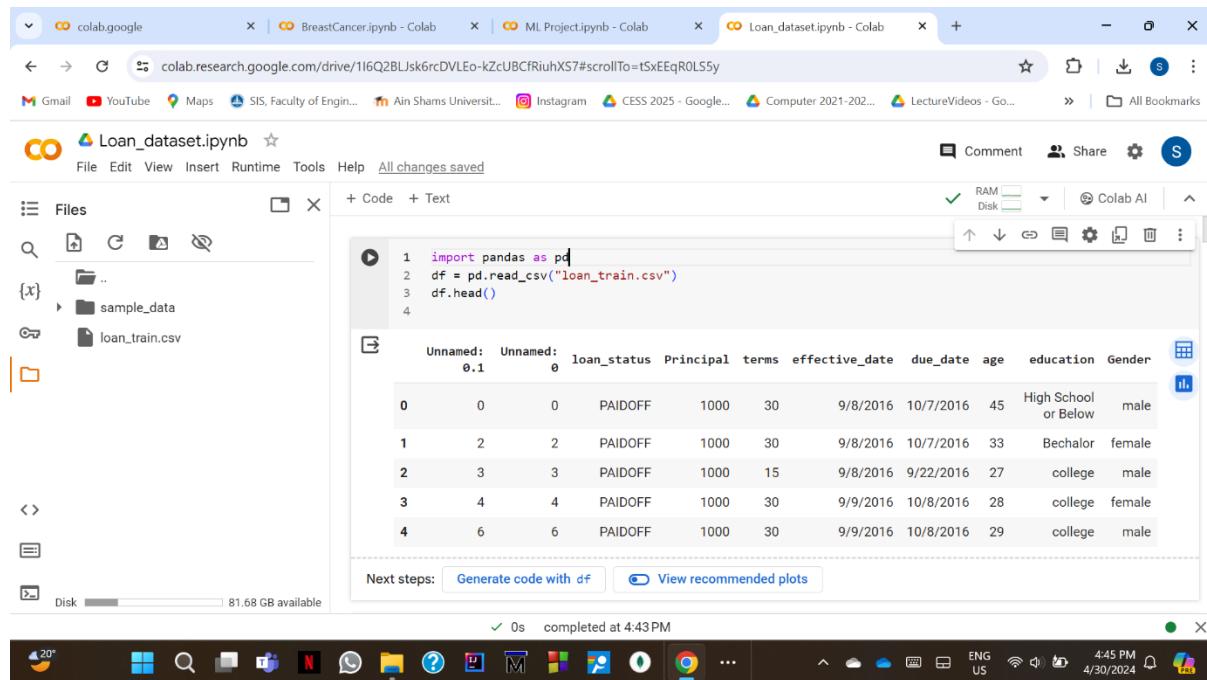


- Final Values of the Hyperparameters:**

After evaluating the KNN model with various k values from 1 to 20, we observed the highest and most stable accuracies around k=5. Although lower k values also provided high accuracy, they can make the model overly sensitive to the training data, potentially capturing noise and anomalies that do not generalize well to new data. Thus, k=5 was chosen as it offers a good balance between accuracy and model generalizability. This value ensures robust performance while minimizing the risk of overfitting, making it suitable for a reliable diagnostic tool.

Loan Dataset:

- Initialize, import and reading of CSV dataset:



The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains the following Python code:

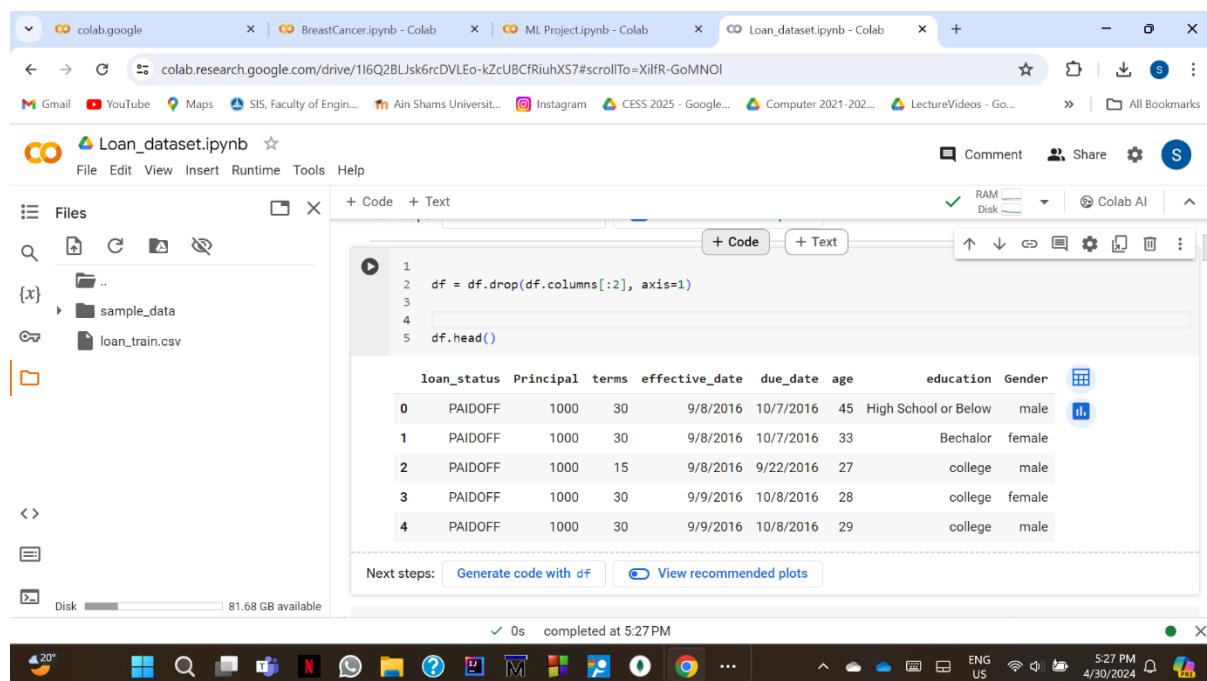
```
1 import pandas as pd
2 df = pd.read_csv("loan_train.csv")
3 df.head()
```

The resulting data frame is displayed below the code:

	Unnamed: 0	loan_status	Principal	terms	effective_date	due_date	age	education	Gender
0	0	PAIDOFF	1000	30	9/8/2016	10/7/2016	45	High School or Below	male
1	2	PAIDOFF	1000	30	9/8/2016	10/7/2016	33	Bachelor	female
2	3	PAIDOFF	1000	15	9/8/2016	9/22/2016	27	college	male
3	4	PAIDOFF	1000	30	9/9/2016	10/8/2016	28	college	female
4	6	PAIDOFF	1000	30	9/9/2016	10/8/2016	29	college	male

Next steps: [Generate code with df](#) [View recommended plots](#)

- Dropping first two columns:



The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains the following Python code:

```
1 df = df.drop(df.columns[:2], axis=1)
2 df.head()
```

The resulting data frame is displayed below the code:

	loan_status	Principal	terms	effective_date	due_date	age	education	Gender
0	PAIDOFF	1000	30	9/8/2016	10/7/2016	45	High School or Below	male
1	PAIDOFF	1000	30	9/8/2016	10/7/2016	33	Bachelor	female
2	PAIDOFF	1000	15	9/8/2016	9/22/2016	27	college	male
3	PAIDOFF	1000	30	9/9/2016	10/8/2016	28	college	female
4	PAIDOFF	1000	30	9/9/2016	10/8/2016	29	college	male

Next steps: [Generate code with df](#) [View recommended plots](#)

- Assuming df is the data frame, display the target variable and the modified data frame:

The screenshot shows two consecutive code cells in a Google Colab notebook titled "Loan_dataset.ipynb".

Code Cell 1:

```

2 target_variable = df.iloc[:, 0]
3 df = df.drop(df.columns[0], axis=1)
4 print("Target Variable:")
5 print(target_variable)
6 print("\nModified DataFrame:")
7 df.head()

```

Output of Code Cell 1:

```

Target Variable:
0 PAIDOFF
1 PAIDOFF
2 PAIDOFF
3 PAIDOFF
4 PAIDOFF
...
341 COLLECTION
342 COLLECTION
343 COLLECTION
344 COLLECTION
345 COLLECTION
Name: loan_status, Length: 346, dtype: object

Modified DataFrame:
   Principal  terms  effective_date  due_date  age  education  Gender
0         1000      30       9/8/2016  10/7/2016  45  High School or Below    male
1         1000      30       9/8/2016  10/7/2016  33      Bachelor    female
2         1000      15       9/8/2016  9/22/2016  27        college    male
3         1000      30       9/9/2016  10/8/2016  28        college    female
4         1000      30       9/9/2016  10/8/2016  29        college    male

```

Code Cell 2:

```

2 PAIDOFF
3 PAIDOFF
4 PAIDOFF
...
341 COLLECTION
342 COLLECTION
343 COLLECTION
344 COLLECTION
345 COLLECTION
Name: loan_status, Length: 346, dtype: object

```

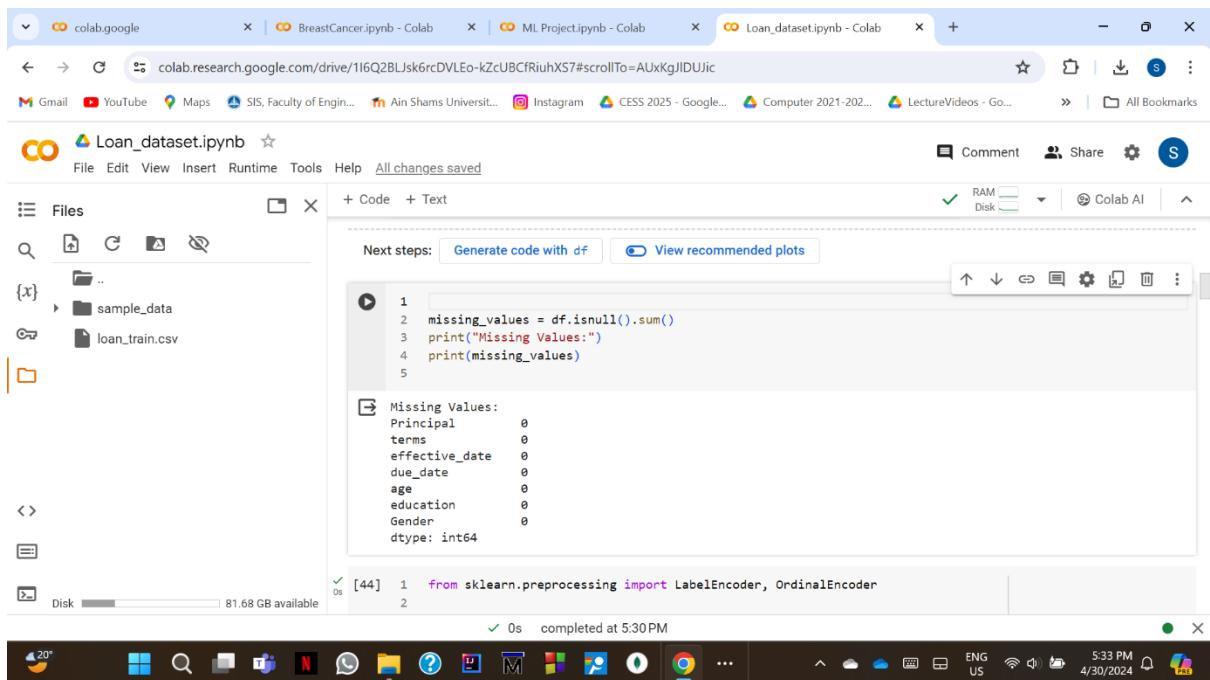
Output of Code Cell 2:

```

PAIDOFF
PAIDOFF
PAIDOFF
...
COLLECTION
COLLECTION
COLLECTION
COLLECTION
COLLECTION
Name: loan_status, Length: 346, dtype: object

```

- Check if there are any missing values in the data frame and display the number of these missing values if there is any:



The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains the following Python code:

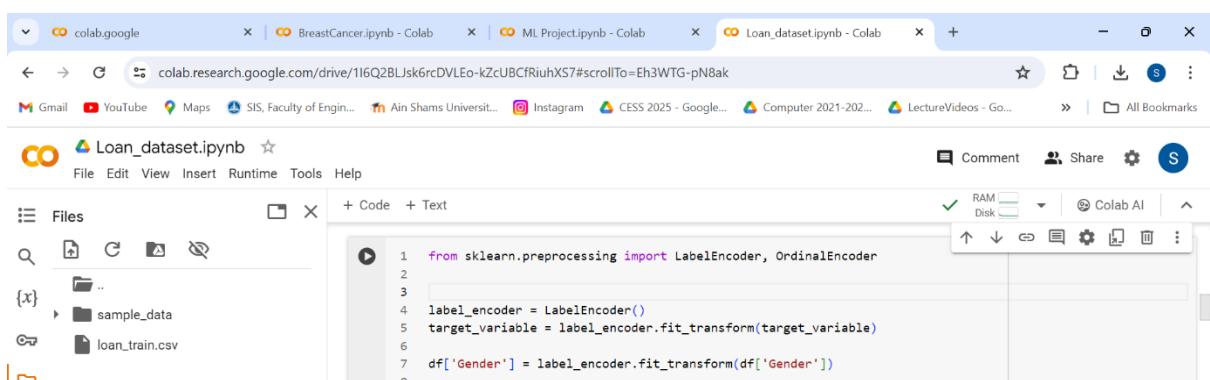
```

1 missing_values = df.isnull().sum()
2 print("Missing Values:")
3 print(missing_values)
4
5
6 Missing Values:
7 Principal      0
8 terms          0
9 effective_date 0
10 due_date       0
11 age            0
12 education      0
13 Gender          0
14 dtype: int64

```

The output cell shows the results of the code execution. The status bar at the bottom indicates "81.68 GB available".

- **Applying Label Encoding:**



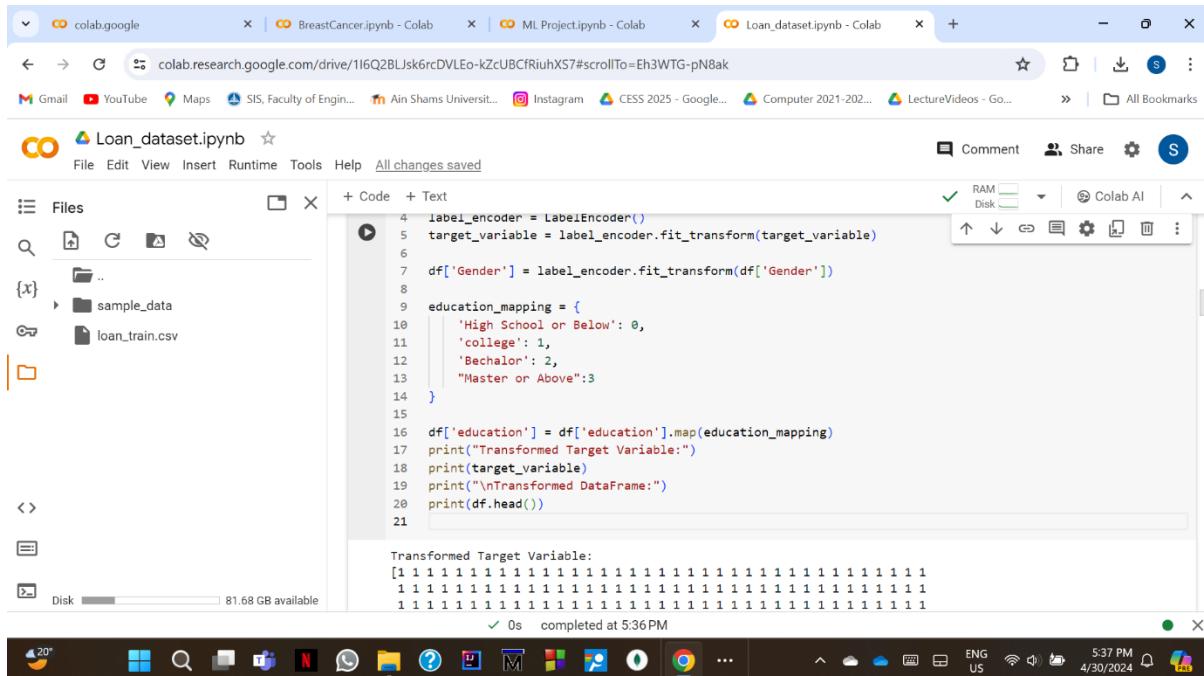
The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains the following Python code:

```

1 from sklearn.preprocessing import LabelEncoder, OrdinalEncoder
2
3
4 label_encoder = LabelEncoder()
5 target_variable = label_encoder.fit_transform(target_variable)
6
7 df['Gender'] = label_encoder.fit_transform(df['Gender'])

```

- Giving education column custom mapping by giving High school or below 0, college 1, bachelor 2 and Master or Above 2, which means that it is better if the education is higher which will mean that he can afford to buy stuff and not having to wait till the due date:



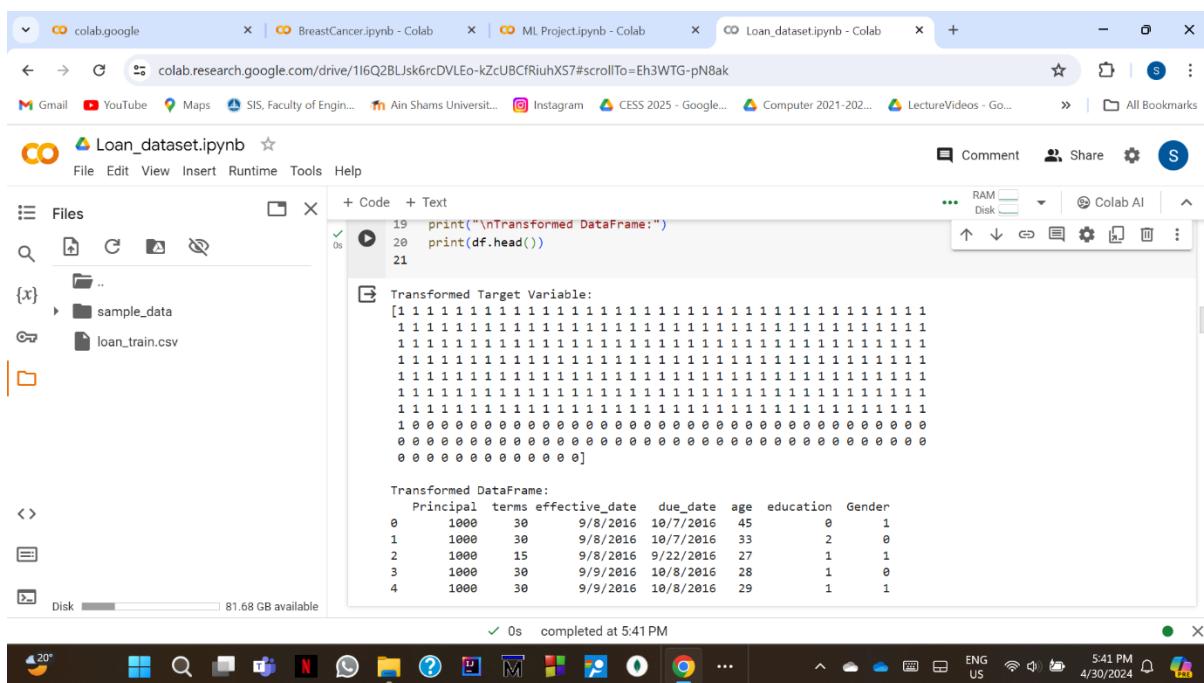
The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains the following Python code:

```

4  label_encoder = LabelEncoder()
5  target_variable = label_encoder.fit_transform(target_variable)
6
7  df['Gender'] = label_encoder.fit_transform(df['Gender'])
8
9  education_mapping = {
10    'High School or Below': 0,
11    'college': 1,
12    'Bachelor': 2,
13    "Master or Above":3
14 }
15
16 df['education'] = df['education'].map(education_mapping)
17 print("Transformed Target Variable:")
18 print(target_variable)
19 print("\nTransformed DataFrame:")
20 print(df.head())
21

```

The output of the code shows the transformed target variable and the first five rows of the transformed DataFrame.



The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains the following Python code:

```

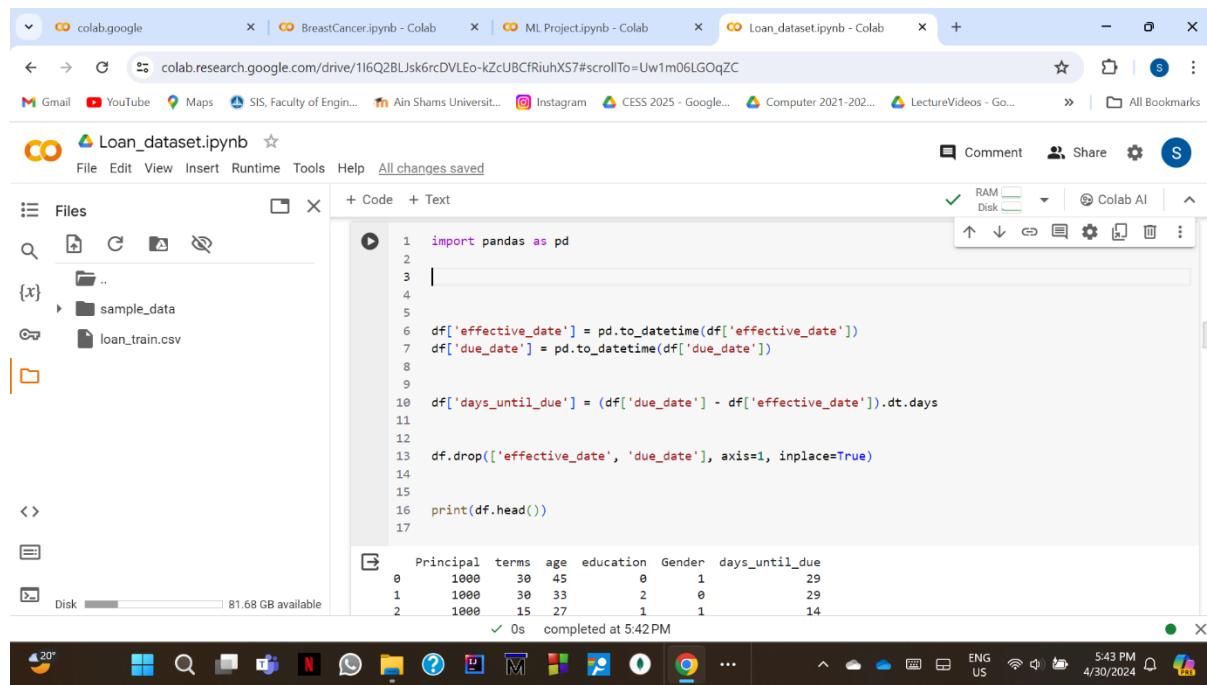
19 print("\nTransformed DataFrame:")
20 print(df.head())
21

```

The output of the code shows the transformed DataFrame, which includes columns: Principal, terms, effective_date, due_date, age, education, and Gender. The data is as follows:

	Principal	terms	effective_date	due_date	age	education	Gender
0	1000	30	9/8/2016	10/7/2016	45	0	1
1	1000	30	9/8/2016	10/7/2016	33	2	0
2	1000	15	9/8/2016	9/22/2016	27	1	1
3	1000	30	9/9/2016	10/8/2016	28	1	0
4	1000	30	9/9/2016	10/8/2016	29	1	1

- Calculating the difference between effective date and due date to make a new column which is “days until due” and then display the modified dataframe:



The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains the following Python code:

```

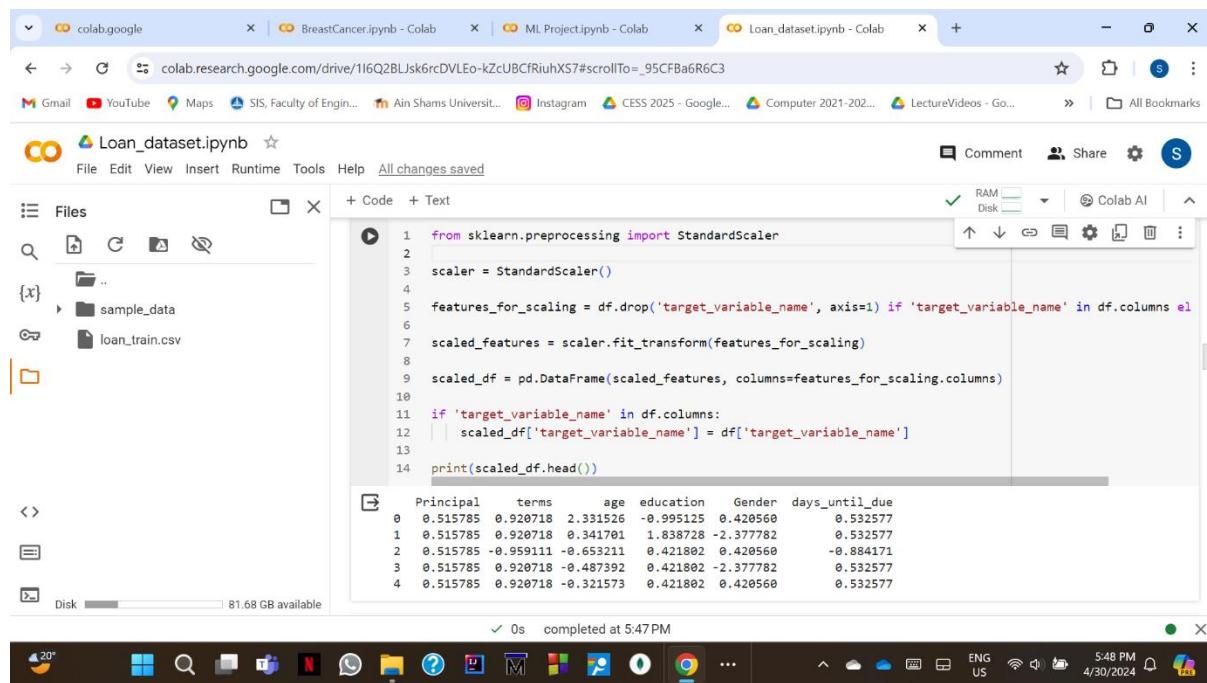
1 import pandas as pd
2
3
4
5
6 df['effective_date'] = pd.to_datetime(df['effective_date'])
7 df['due_date'] = pd.to_datetime(df['due_date'])
8
9
10 df['days_until_due'] = (df['due_date'] - df['effective_date']).dt.days
11
12
13 df.drop(['effective_date', 'due_date'], axis=1, inplace=True)
14
15
16 print(df.head())
17

```

The output cell displays the first few rows of the modified DataFrame:

	Principal	terms	age	education	Gender	days_until_due
0	1000	30	45	0	1	29
1	1000	30	33	2	0	29
2	1000	15	27	1	1	14

- Scaling ranging from -1 to 1:



The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains the following Python code:

```

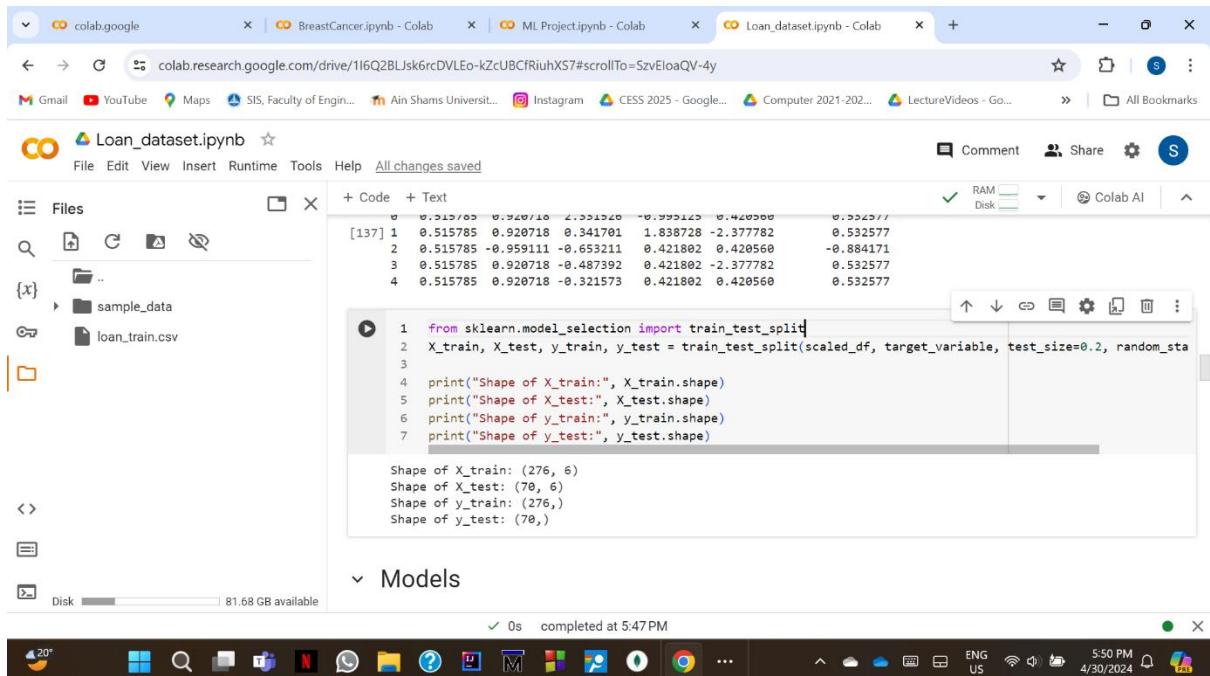
1 from sklearn.preprocessing import StandardScaler
2
3 scaler = StandardScaler()
4
5 features_for_scaling = df.drop('target_variable_name', axis=1) if 'target_variable_name' in df.columns else df
6
7 scaled_features = scaler.fit_transform(features_for_scaling)
8
9 scaled_df = pd.DataFrame(scaled_features, columns=features_for_scaling.columns)
10
11 if 'target_variable_name' in df.columns:
12     scaled_df['target_variable_name'] = df['target_variable_name']
13
14 print(scaled_df.head())

```

The output cell displays the scaled DataFrame:

	Principal	terms	age	education	Gender	days_until_due
0	0.515785	0.920718	2.331526	-0.995125	0.420560	0.532577
1	0.515785	0.920718	0.341701	1.838728	-2.377782	0.532577
2	0.515785	-0.959111	-0.653211	0.421802	0.420560	-0.884171
3	0.515785	0.920718	-0.487392	0.421802	-2.377782	0.532577
4	0.515785	0.920718	-0.321573	0.421802	0.420560	0.532577

- Display the shapes of the training and testing sets after splitting them:



The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". In the code editor, the following Python code is written:

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_df, target_variable, test_size=0.2, random_state=42)
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)

```

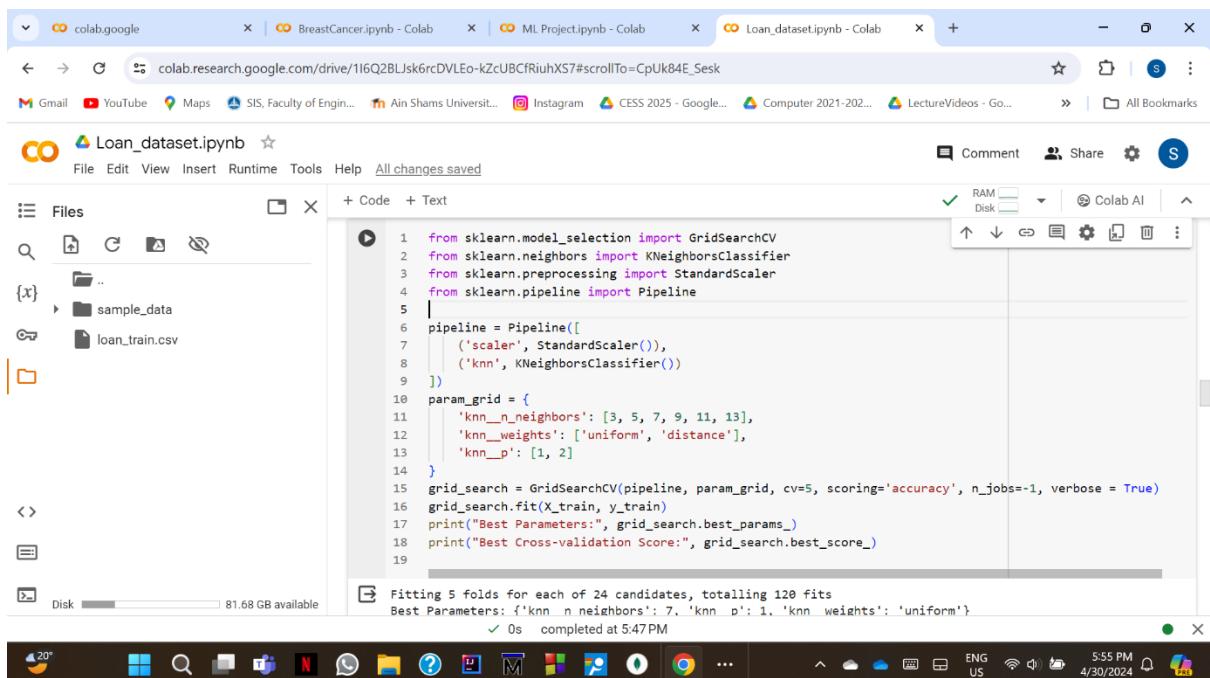
The output of this code is displayed below the code cell:

```

Shape of X_train: (276, 6)
Shape of X_test: (70, 6)
Shape of y_train: (276,)
Shape of y_test: (70,)

```

- KNN Model with Parameters (3,5,7,9,11,13) and its accuracy:



The screenshot shows a Google Colab notebook titled "Loan_dataset.ipynb". In the code editor, the following Python code is written:

```

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])
param_grid = {
    'knn_n_neighbors': [3, 5, 7, 9, 11, 13],
    'knn_weights': ['uniform', 'distance'],
    'knn_p': [1, 2]
}
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose = True)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-validation Score:", grid_search.best_score_)

```

The output of this code is displayed below the code cell:

```

Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'knn_n_neighbors': 7, 'knn_p': 1, 'knn_weights': 'uniform'}
Best Cross-validation Score: 0.8555555555555556

```

A screenshot of a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains Python code for fitting a KNN classifier using GridSearchCV. The output shows the best parameters found: {'knn__n_neighbors': 7, 'knn__p': 1, 'knn__weights': 'uniform'} and a cross-validation score of 0.7101948051948053. A second code cell shows the prediction accuracy on the test set, which is 0.7428571428571429.

```
14 }
15 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose = True)
16 grid_search.fit(X_train, y_train)
17 print("Best Parameters:", grid_search.best_params_)
18 print("Best Cross-validation Score:", grid_search.best_score_)

19
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'knn__n_neighbors': 7, 'knn__p': 1, 'knn__weights': 'uniform'}
Best Cross-validation Score: 0.7101948051948053

1
2 y_pred = grid_search.predict(X_test)
3 from sklearn.metrics import accuracy_score
4
5 accuracy = accuracy_score(y_test, y_pred)
6 print("Accuracy on the test set:", accuracy)
7

Accuracy on the test set: 0.7428571428571429
```

- KNN Model with Parameters (3,5,7) and its accuracy:

A screenshot of a Google Colab notebook titled "Loan_dataset.ipynb". The code cell contains Python code for fitting a KNN classifier using GridSearchCV with a parameter grid ranging from 3 to 7. The output shows the best parameters found: {'knn__n_neighbors': 7, 'knn__p': 1, 'knn__weights': 'uniform'} and a cross-validation score of 0.7101948051948053.

```
2s
2 from sklearn.neighbors import KNeighborsClassifier
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.pipeline import Pipeline
5
6 pipeline = Pipeline([
7     ('scaler', StandardScaler()),
8     ('knn', KNeighborsClassifier())
9 ])
10 param_grid = [
11     {'knn__n_neighbors': [3, 5, 7],
12      'knn__weights': ['uniform', 'distance'],
13      'knn__p': [1, 2]}
14 ]
15 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose = True)
16 grid_search.fit(X_train, y_train)
17 print("Best Parameters:", grid_search.best_params_)
18 print("Best Cross-validation Score:", grid_search.best_score_)

Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Parameters: {'knn__n_neighbors': 7, 'knn__p': 1, 'knn__weights': 'uniform'}
Best Cross-validation Score: 0.7101948051948053
```

```
13     'knn__p': [1, 2]
14 }
15 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose = True)
16 grid_search.fit(X_train, y_train)
17 print("Best Parameters:", grid_search.best_params_)
18 print("Best Cross-validation Score:", grid_search.best_score_)
19
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Parameters: {'knn__n_neighbors': 7, 'knn__p': 1, 'knn__weights': 'uniform'}
Best Cross-validation Score: 0.7101948051948053
```

```
1
2 y_pred = grid_search.predict(X_test)
3 from sklearn.metrics import accuracy_score
4
5 accuracy = accuracy_score(y_test, y_pred)
6 print("Accuracy on the test set:", accuracy)
7
```

Accuracy on the test set: 0.7428571428571429

- **The Reason of using the final parameters:**

We notice that in both cases the model has chosen from the neighbour number 7 and to be uniform and using Manhattan which is number 1. Which means that by increasing numbers or decreasing them the best values to be chosen are 7, uniform and 1.

- Naïve Bayes Classifier and its accuracy:

The screenshot shows the Google Colab interface with a notebook titled "Loan_dataset.ipynb". The code cell contains Python code for setting up a pipeline and performing a grid search:

```

1 from sklearn.model_selection import GridSearchCV
2 from sklearn.naive_bayes import GaussianNB
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.pipeline import Pipeline
5
6 pipeline = Pipeline([
7     ('scaler', StandardScaler()),
8     ('nb', GaussianNB())
9 ])
10 param_grid = {
11 }
12
13
14 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=True)
15
16 grid_search.fit(X_train, y_train)
17
18 print("Best Parameters:", grid_search.best_params_)
19
20 print("Best Cross-validation Score:", grid_search.best_score_)
21
22

```

Below the code cell, the status bar indicates "Executing (18s)". The taskbar at the bottom shows various icons for system functions.

The screenshot shows the Google Colab interface with the same notebook "Loan_dataset.ipynb". The code cell has been executed, and the output shows the best parameters and cross-validation score:

```

18
19 print("Best Parameters:", grid_search.best_params_)
20
21 print("Best Cross-validation Score:", grid_search.best_score_)
22

```

Fitting 5 folds for each of 1 candidates, totalling 5 fits
Best Parameters: {}
Best Cross-validation Score: 0.6051948051948053

Below the output, another code cell is shown:

```

1 |
2 y_pred = grid_search.predict(X_test)
3 from sklearn.metrics import accuracy_score
4
5 accuracy = accuracy_score(y_test, y_pred)
6 print("Accuracy on the test set:", accuracy)
7

```

Accuracy on the test set: 0.5571428571428572

The status bar indicates "completed at 6:09 PM". The taskbar at the bottom shows various icons for system functions.

iPhone Purchase:

- **Loading the file:** we used `pd.read_csv()` to read the file as follows:

```
# Load the dataset  
df = pd.read_csv('Iphone_purchase.csv')
```

- **Before preprocessing:** we ignored the “id” column from consideration:

We used `drop()` function as follows:

```
# Drop the 'User ID' column  
df.drop('User ID', axis=1, inplace=True)
```

- **Preprocessing stage:**

- **Encoding:** Encoding categorical column 'Gender' by `LabelEncoder()` as follows:

```
# Encoding categorical column 'Gender'  
labelEncoder = LabelEncoder()  
labelEncoder.fit(df["Gender"])  
df["Gender"] = labelEncoder.transform(df["Gender"])
```

- **Splitting the data into features (X) and target variable (y) as follows:**

```
# Splitting the data into features (X) and target variable (y)  
X = df[['Gender', 'Age', 'EstimatedSalary']]  
y = df['Purchased']
```

- **Splitting the data into training and testing sets:** using `train_test_split()` with test size 20% of the data as follows:

```
# Splitting 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)
```

- **Feature scaling:** using `StandardScaler()` function on both training and testing inputs as follows:

```
# Feature scaling  
scaler = StandardScaler()  
X_train_scaled = scaler.fit_transform(X_train)  
X_test_scaled = scaler.transform(X_test)
```

- **Training stage:** we are training two classifiers KNN and Naïve Bayes:

- **Naïve Bayes:** using **GaussianNB()** as it is the most suitable technique according to the dataset given, we use the scaled training features “x_train_scaled” and training target variables “y_train” by **fit()** function as follows:

```
# Train the Naïve Bayes Classifier
naive_bayes_classifier = GaussianNB()
naive_bayes_classifier.fit(X_train_scaled, y_train)
```

I thought about using Bernoulli instead of Gaussian distribution, but this type works on binarized features only, which contradicts with our training data.

- **KNN classifier:** I tuned 3 hyperparameters (no. of neighbors, weight of features, metric used). I examined the classifier on ranges of n [1,10] weight [distance, uniform], metric can be (Euclidean or Manhattan). So, we made 3 nested for loop to test all the combinations available as follows:

```
# Train the KNN Classifier with different values of n_neighbors, weights, and metric
for n in range(1, 11): # trying values from 1 to 10 for n_neighbors hyperparameter
    for weight in ['uniform', 'distance']: # trying both 'uniform' and 'distance' for weights hyperparameter
        for metric in ['euclidean', 'manhattan']: # trying only 'euclidean' and 'manhattan' for metric
            knn_classifier = KNeighborsClassifier(n_neighbors=n, weights=weight, metric=metric)
            knn_classifier.fit(X_train_scaled, y_train)
```

- **Testing stage:** we test both classifiers using **`predict()`** function and measure accuracy using **`metrics.accuracy_score()`**

- **Naïve Bayes:** we use the scaled testing data in prediction then we compare testing outputs we have with our predictions to calculate accuracy as follows:

```
# Test the Naïve Bayes Classifier
naive_bayes_predictions = naive_bayes_classifier.predict(X_test_scaled)
naive_bayes_accuracy = accuracy_score(y_test, naive_bayes_predictions)
print("Naïve Bayes Classifier Accuracy:", naive_bayes_accuracy * 100)
print("Naïve Bayes Classifier Predictions on Testing Set:")
print(naive_bayes_predictions)
```

- **KNN:** we use the same arguments as Naïve Bayes.

```
# Test the KNN Classifier
knn_predictions = knn_classifier.predict(X_test_scaled)
knn_accuracy = accuracy_score(y_test, knn_predictions)
accuracy_scores.append(knn_accuracy)

print(f"KNN Classifier Accuracy (n_neighbors={n}, weights={weight}, metric={metric}):", knn_accuracy * 100)
```

Now, we need to track all the tunings of the hyperparameters to get the combination that gives highest accuracy. Thus, we do some extra steps:

- First, initialize variables to keep track of the best KNN classifier and its hyperparameters as follows:

```
# Initialize variables to keep track of the best KNN classifier and its hyperparameters
best_knn_accuracy = 0
best_knn_predictions = None
best_n_neighbors = None
best_weights = None
best_metric = None
accuracy_scores = []
```

We set all the best hyperparameters to the minimum to get the highest records.

- Next, we make an if condition that updates the best hyperparameters once better records came as follows:

```
# Update best KNN classifier and its hyperparameters if current one has higher accuracy
if knn_accuracy > best_knn_accuracy:
    best_knn_accuracy = knn_accuracy
    best_knn_predictions = knn_predictions
    best_n_neighbors = n
    best_weights = weight
    best_metric = metric
```

- Finally, we print the final best tuning of hyperparameters as follows:

```
print("\n Best KNN Classifier Accuracy:", best_knn_accuracy * 100)
print("Hyperparameters for Best KNN Classifier:")
print(" - n_neighbors:", best_n_neighbors)
print(" - weights:", best_weights)
print(" - metric:", best_metric)
print("Best KNN Classifier Predictions on Testing Set:")
print(best_knn_predictions)
```

• Screenshots of Output:

- Naïve Bayes:** No hyperparameters can be tuned in Naïve Bayes classifier except the distribution as using Bernoulli instead of Gaussian but I explained before why such tuning is not good. The output is shown as follows:

```
C:\Users\seifa\pythonProject1\Scripts\python.exe C:\Users\seifa\PycharmProjects\pythonProject1\main.py
Naïve Bayes Classifier Accuracy: 93.75
Naïve Bayes Classifier Predictions on Testing Set:
[1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 0
 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0
 0 0 1 1 0 0]
```

- KNN:** we have 40 combinations of tuned hyperparameters. (neighbours, weight, metric) shown as follows:

The screenshot shows the PyCharm IDE's Run interface with a list of 40 entries. Each entry represents the accuracy of a KNN classifier with specific parameters: (n_neighbors, weights, metric). The parameters are varied across values: n_neighbors (1, 2, 3, 4, 5, 6, 7), weights (uniform, distance), and metric (euclidean, manhattan). The accuracy values range from 83.75 to 95.0.

n_neighbors	weights	metric	Accuracy
1	uniform	euclidean	88.75
1	uniform	manhattan	88.75
1	distance	euclidean	88.75
1	distance	manhattan	88.75
2	uniform	euclidean	83.75
2	uniform	manhattan	85.0
2	distance	euclidean	88.75
2	distance	manhattan	88.75
3	uniform	euclidean	91.25
3	uniform	manhattan	93.75
3	distance	euclidean	88.75
3	distance	manhattan	88.75
4	uniform	euclidean	91.25
4	uniform	manhattan	91.25
4	distance	euclidean	91.25
4	distance	manhattan	91.25
5	uniform	euclidean	92.5
5	uniform	manhattan	92.5
5	distance	euclidean	91.25
5	distance	manhattan	90.0
6	uniform	euclidean	92.5
6	uniform	manhattan	95.0
6	distance	euclidean	92.5
6	distance	manhattan	91.25
7	uniform	euclidean	92.5
7	uniform	manhattan	93.75
7	distance	euclidean	91.25

```
Run: main <pre>KNN Classifier Accuracy (n_neighbors=7, weights=distance, metric=euclidean): 91.25
KNN Classifier Accuracy (n_neighbors=7, weights=distance, metric=manhattan): 92.5
KNN Classifier Accuracy (n_neighbors=8, weights=uniform, metric=euclidean): 93.75
KNN Classifier Accuracy (n_neighbors=8, weights=uniform, metric=manhattan): 93.75
KNN Classifier Accuracy (n_neighbors=8, weights=distance, metric=euclidean): 92.5
KNN Classifier Accuracy (n_neighbors=8, weights=distance, metric=manhattan): 92.5
KNN Classifier Accuracy (n_neighbors=9, weights=uniform, metric=euclidean): 93.75
KNN Classifier Accuracy (n_neighbors=9, weights=uniform, metric=manhattan): 93.75
KNN Classifier Accuracy (n_neighbors=9, weights=distance, metric=euclidean): 93.75
KNN Classifier Accuracy (n_neighbors=9, weights=distance, metric=manhattan): 93.75
KNN Classifier Accuracy (n_neighbors=10, weights=uniform, metric=euclidean): 93.75
KNN Classifier Accuracy (n_neighbors=10, weights=uniform, metric=manhattan): 93.75
KNN Classifier Accuracy (n_neighbors=10, weights=distance, metric=euclidean): 93.75
KNN Classifier Accuracy (n_neighbors=10, weights=distance, metric=manhattan): 93.75</pre>
```

- **Final Values:**

After we tracked all the combinations of hyperparameters, we were able to output the hyperparameters that recorded the highest accuracies with their predictions as follows:

```
Best KNN Classifier Accuracy: 95.0
Hyperparameters for Best KNN Classifier:
- n_neighbors: 6
- weights: uniform
- metric: manhattan
Best KNN Classifier Predictions on Testing Set:
[1 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0
 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0
 1 0 1 1 0 0]

Process finished with exit code 0
|
```

on Control Run Python Packages TODO Python Console Problems Terminal Services

n_neighbors: **6**
weights: **uniform**
metric: **manhattan**