Predictive Modeling of Diabetes: A Machine Learning Approach

Importing Dataset and Libraries

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

!pip install ydata-profiling
from ydata_profiling import ProfileReport

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE

from sklearn import tree
from sklearn.tree import plot_tree
from sklearn.tree import export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_predict
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score
```

```
  Preparing metadata (setup.py) ... done
Collecting phik<0.13,>=0.11.1 (from ydata-profiling)
  Downloading phik-0.12.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.6 kB)
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (2.32.3)
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.66.6)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.13.2)
Collecting multimethod<2,>=1.4 (from ydata-profiling)
  Downloading multimethod-1.12-py3-none-any.whl.metadata (9.6 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.14.4)
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.4.1)
Collecting imagehash==4.3.1 (from ydata-profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: wordcloud>=1.9.3 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.9.4)
Collecting dacite>=1.8 (from ydata-profiling)
  Downloading dacite-1.8.1-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: numba<1,>=0.56.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.60.0)
Collecting PyWavelets (from imagehash==4.3.1->ydata-profiling)
  Downloading pywavelets-1.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.0 kB)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling) (11.0.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling) (3.0.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling) (2.8.2)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba<1,>=0.56.0->ydata-profiling) (0.43.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling) (2024.2)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-packages (from phik<0.13,>=0.11.1->ydata-profiling) (1.4.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling) (0.7.0)
Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling) (2.23.4)
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (2024.8.30)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels<1,>=0.13.2->ydata-profiling) (1.0.1)
```

```
Downloading pywavelets-1.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 77.9 MB/s eta 0:00:00
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) ... done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=27081 sha256=5c1c3b118cfb088be8d90217d845f0cc6bc949a88fca5b2dfb0362b9b79e88
```

```python
from google.colab import files
uploaded = files.upload()
```

Choose Files Dataset of Diabetes .csv

- **Dataset of Diabetes .csv**(text/csv) - 49511 bytes, last modified: 10/4/2024 - 100% done

```python
db = pd.read_csv("Dataset of Diabetes .csv")
db
```

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	200	454317	M	71	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	Y
996	671	876534	M	31	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	Y
997	669	87654	M	30	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	Y
998	99	24004	M	38	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	Y
999	248	24054	M	54	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	Y

1000 rows × 14 columns

Next steps: Generate code with db | View recommended plots | New interactive sheet

Stage 1: Data Preparation

Data Formatting

```python
#Renaming the columns to remove capitalization, correct spelling and avoid programming constraints with the libraries being used. This allows for overall

db.columns = ['id', 'no_patient', 'gender', 'age', 'urea', 'cr',
              'hba1c', 'chol', 'tg', 'hdl',
              'ldl', 'vldl', 'bmi', 'db_class']

db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id          1000 non-null   int64
 1   no_patient  1000 non-null   int64
 2   gender      1000 non-null   object
 3   age         1000 non-null   int64
 4   urea        1000 non-null   float64
 5   cr          1000 non-null   int64
 6   hba1c       1000 non-null   float64
 7   chol        1000 non-null   float64
 8   tg          1000 non-null   float64
 9   hdl         1000 non-null   float64
 10  ldl         1000 non-null   float64
 11  vldl        1000 non-null   float64
 12  bmi         1000 non-null   float64
 13  db_class    1000 non-null   object
dtypes: float64(8), int64(4), object(2)
memory usage: 109.5+ KB
```

Data Cleaning

```python
#checking for missing values

```

```
print(db.isnull().sum())
```

```
id            0
no_patient    0
gender        0
age           0
urea          0
cr            0
hba1c         0
chol          0
tg            0
hdl           0
ldl           0
vldl          0
bmi           0
db_class      0
dtype: int64
```

```
#checking for duplicate values in the id column

duplicate_id = db['id'].duplicated().sum()
duplicate_id
```

```
200
```

```
#identifying the duplicate id vlaues

duplicate_num_id = db['id'].value_counts()
duplicate_num_id = duplicate_num_id[duplicate_num_id > 1]
print(duplicate_num_id)
```

```
id
76     2
108    2
57     2
26     2
69     2
      ..
150    2
49     2
144    2
145    2
147    2
Name: count, Length: 200, dtype: int64
```

```
#Random sample #1 - Checking if the ids are duplicate values for the id column or duplicate patient records (i.e. the both observations have the same val
#Since the other attributes have different values in each observation, this might be considered as a duplicate value for the id column only.

display(db[db['id'] == 76])
```

	id	no_patient	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	db_class
249	76	9903	M	73	4.3	79	6.0	5.3	1.4	1.5	3.2	0.6	27.0	Y
910	76	8978	M	60	5.4	64	10.4	3.8	1.5	0.8	2.3	0.6	31.0	Y

```
#Random sample #2 - Checking if the ids are duplicate values only or duplicate patient record (i.e. the both observations have the same values for each c
#Confirming that the duplicate ids can be consideres as duplicate values for id column only.

display(db[db['id'] == 150])
```

	id	no_patient	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	db_class
88	150	45382	F	40	6.3	79	4.9	4.3	0.8	0.8	1.8	1.2	22.0	N
184	150	34287	F	42	2.8	39	4.6	4.7	2.5	1.3	2.4	1.1	25.0	Y

```
#checking for duplicate values in the patient number column

duplicate_no_patient = db['no_patient'].duplicated().sum()
duplicate_no_patient
```

```
39
```

```
#identifying the duplicate patient numbers

duplicate_num_patient = db['no_patient'].value_counts()
duplicate_num_patient = duplicate_num_patient[duplicate_num_patient > 1]
print(duplicate_num_patient)
```

```
no_patient
454316    19
856        2
87654      2
71741      2
```

```
34290      2
14389      2
34517      2
48362      2
45646      2
44835      2
24033      2
34514      2
23972      2
34325      2
234        2
45370      2
34516      2
34518      2
2345       2
34515      2
34545      2
345        2
Name: count, dtype: int64
```

```
#Random sample #1 - Checking if the patient number are duplicate values only or duplicate patient record (i.e. the both observations have the same values
#Since the other columns have different values in each observation, this might be duplicate values for the patient number column only.

display(db[db['no_patient'] == 45646])
```

	id	no_patient	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	db_class
936	111	45646	F	63	4.7	55	12.60	7.9	5.0	1.3	2.1	0.7	35.0	Y
938	113	45646	F	55	2.1	23	9.96	4.1	4.2	1.2	1.4	1.3	29.0	Y

```
#Random sample #2 - Checking if the patient number are duplicate values only or duplicate patient record (i.e. the both observations have the same values
#Confirming that the duplicate patient numbers can be considered as duplicate values for the patient column only.

display(db[db['no_patient'] == 34290])
```

	id	no_patient	gender	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	db_class
34	699	34290	F	47	5.6	67	5.1	6.5	1.5	0.9	4.9	0.7	23.0	N
260	510	34290	M	73	4.3	79	6.9	5.3	1.4	1.5	3.2	0.6	28.0	Y

```
#checking the gender column

gender_frequency = db['gender'].value_counts()
print(gender_frequency)
```

```
gender
M    565
F    434
f      1
Name: count, dtype: int64
```

```
#Converting the lowercase f to uppercase and updating the gender_frequency variable

db['gender'] = db['gender'].replace({'f': 'F'})
gender_frequency = db['gender'].value_counts()
print(gender_frequency)
```

```
gender
M    565
F    435
Name: count, dtype: int64
```

```
#checking the class column

class_frequency = db['db_class'].value_counts()
print(class_frequency)
```

```
db_class
Y     840
N     102
P      53
Y       4
N       1
Name: count, dtype: int64
```

```
#There is a space trailing after some of the Y and N values (eg: 'Y ' and 'N '). This is causing the data to be read as different values. Removing the sp

db['db_class'] = db['db_class'].replace({'N ': 'N', 'Y ': 'Y'})
class_frequency = db['db_class'].value_counts()
print(class_frequency)
```

```
db_class
Y    844
```

```
N    103
P     53
Name: count, dtype: int64
```

Stage 2: Exploratory Data Analysis

Statistical Summary

```
db.describe()
```

	id	no_patient	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	
count	1000.000000	1.000000e+03	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.0
mean	340.500000	2.705514e+05	53.528000	5.124743	68.943000	8.281160	4.862820	2.349610	1.204750	2.609790	1.854700	29.5
std	240.397673	3.380758e+06	8.799241	2.935165	59.984747	2.534003	1.301738	1.401176	0.660414	1.115102	3.663599	4.9
min	1.000000	1.230000e+02	20.000000	0.500000	6.000000	0.900000	0.000000	0.300000	0.200000	0.300000	0.100000	19.0
25%	125.750000	2.406375e+04	51.000000	3.700000	48.000000	6.500000	4.000000	1.500000	0.900000	1.800000	0.700000	26.0
50%	300.500000	3.439550e+04	55.000000	4.600000	60.000000	8.000000	4.800000	2.000000	1.100000	2.500000	0.900000	30.0
75%	550.250000	4.538425e+04	59.000000	5.700000	73.000000	10.200000	5.600000	2.900000	1.300000	3.300000	1.500000	33.0
max	800.000000	7.543566e+07	79.000000	38.900000	800.000000	16.000000	10.300000	13.800000	9.900000	9.900000	35.000000	47.7

The average age of the patients in the dataset is 53.53, suggeting the population is mostly middle age. The creatinine variable has a large standard deviation (59.98) and a large range (from 6 to 800), suggesting the presence of extreme values.

Correlation Analysis

```
#Using Panda Profiling Report to perform analysis on the dataset. The output is hidden to ensure the complete download of the IPYNB file, and saving the
#When the output is showing the file cuts off at this code block

report = ProfileReport(db)
report
```

Show hidden output

Report confirms there are no missing values or duplicate rows within the dataset.

Both HbA1c and Cholesterol exhibit a normal bell shaped distribution, indicating a relatively even spread of data around the mean. The other features appear skewed, suggesting that their data distributions are more concentrated around certain values, and the presence of outliers.

Cholesterol, triglycerides, HLDL and LDL have variances below 2, with the lowest being 0.44 for the HLDL variable. This suggests that these variables exhibit small spread with most of the values being close to the mean.

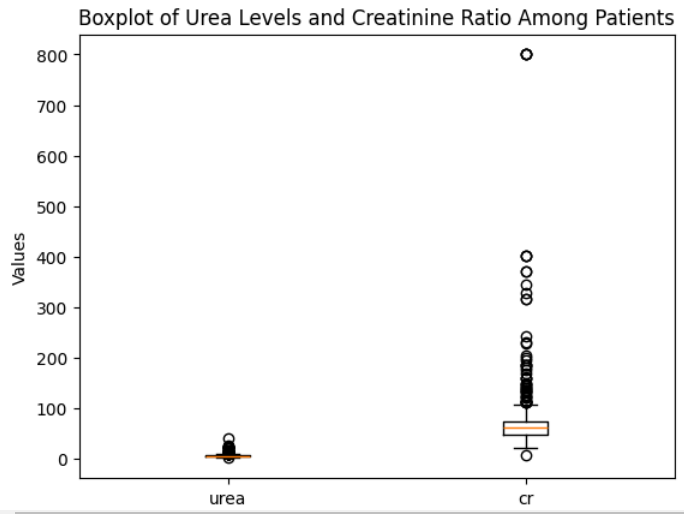
There are several correlations obsevered among the variable. Triglycerides(tg) has a high positive correlation with vldl (0.599), and creatinine has a high positive correlation with urea (0.568). BMI exhibits moderately positive correlation with a coefficient of 0.417, suggesting elevaated HbA1c levels are associated with BMI.

The target variable, db_class, has a strong positive correlation with bmi (0.549) and hba1c (0.662). This suggests that both bmi and hba1c are influential and important in determining the likelihood of diabetes.

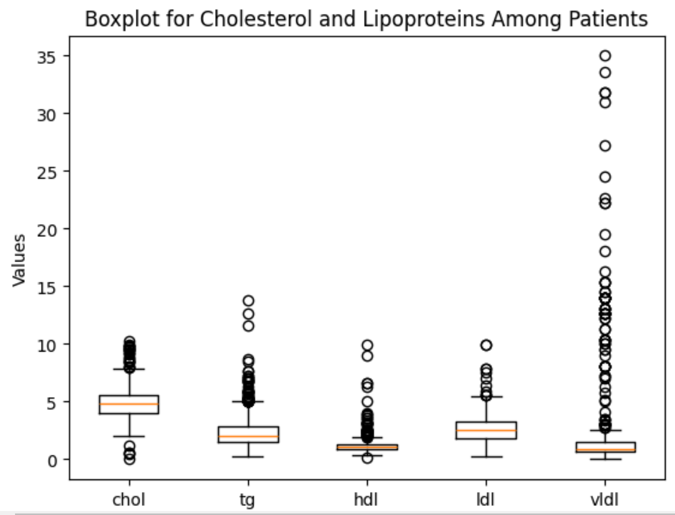
There is a significant class imbalance with 844 individuals classified as diabetic, 103 as non-diabetic, and only 53 as pre-diabetic. This 51.5% imbalance may affect the performance of the predictive algorithms leading to bias towards the majority class.

Data Visualization

```
#Observing the values in Urea and Creatinine columns

plt.boxplot(db[['urea','cr']])
plt.title('Boxplot of Urea Levels and Creatinine Ratio Among Patients')

plt.xticks([1, 2], ['urea','cr'])
plt.ylabel("Values")

plt.show()
```



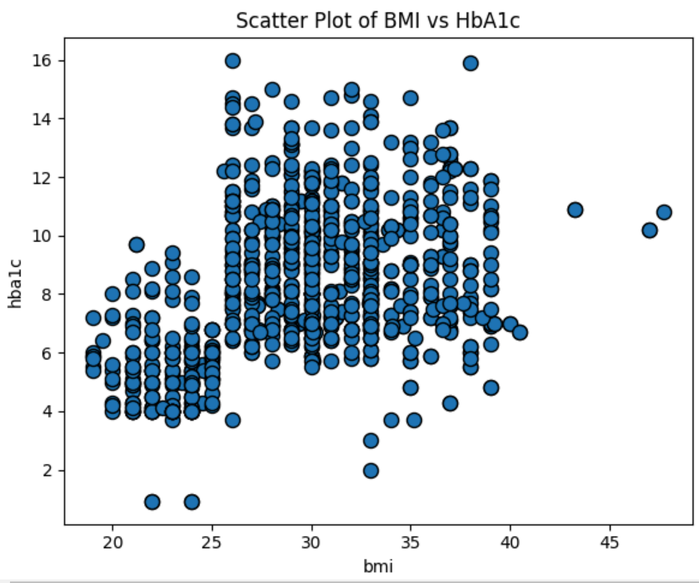
```
#Observing the values for the cholesterol and lipoproteins columns

plt.boxplot(db[['chol','tg','hdl','ldl','vldl']])
plt.title('Boxplot for Cholesterol and Lipoproteins Among Patients')

plt.xticks([1, 2, 3, 4, 5], ['chol','tg','hdl','ldl','vldl'])
plt.ylabel("Values")

plt.show()
```



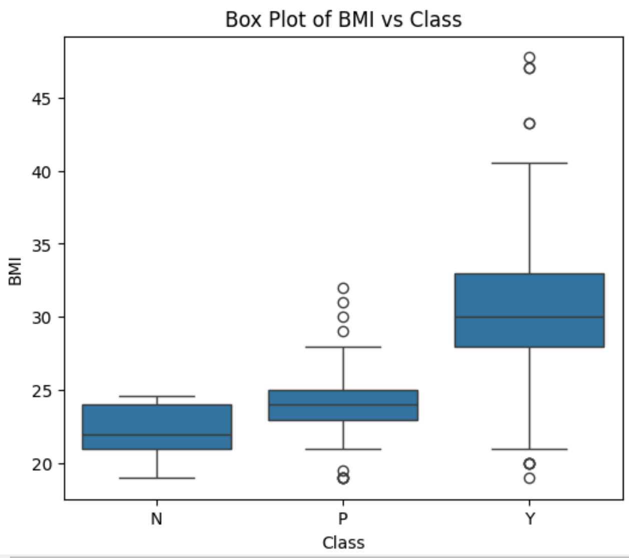
There are noticeable outliers present in the cholesterol, lipoprotein, urea, and creatinine attributes. This suggests that several patients may have extreme values for these attributes.

Urea and creatinine ratio are clinical measurements used to assess kidney impairment. While cholestrol, the lipoproteins and trtriglycerides levels are used to determine cardiovascular damage. These outliers therefore suggests there may be patients with kidney and cardiovascular damage.

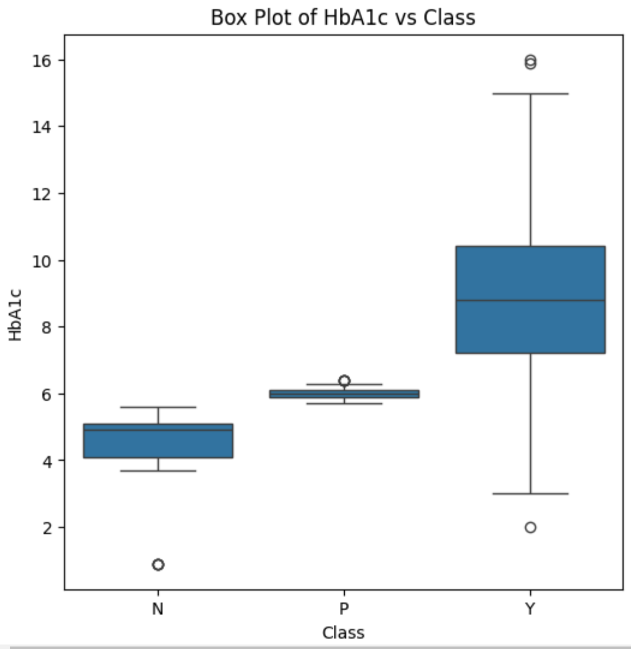
```
#Graph showing relationship between the BMI and Glusoce attributes

plt.figure(figsize=(6, 5))
plt.scatter(db['bmi'], db['hba1c'], s=75, edgecolor='k')

plt.title('Scatter Plot of BMI vs HbA1c')
plt.xlabel('bmi')
plt.ylabel('hba1c')
plt.tight_layout()
plt.show()
```



```
#Graph showing relationship between the BMI and Class attributes

plt.figure(figsize=(6, 5))
sns.boxplot(x='db_class', y='bmi', data=db)
plt.title('Box Plot of BMI vs Class')
plt.xlabel('Class')
plt.ylabel('BMI')
plt.show()
```



```
#Graph showing relationship between the HbA1c and Class attributes

plt.figure(figsize=(6, 6))
sns.boxplot(x='db_class', y='hba1c', data=db)
plt.title('Box Plot of HbA1c vs Class')
plt.xlabel('Class')
plt.ylabel('HbA1c')
plt.show()
```



Stage 3: Data Preprocessing

Label Encoding

```
#Coverting the categorical attributes to numerical variables to make compatible for the classifiers.

db_encode=db.copy()

label_encoder = LabelEncoder()

db_encode['gender'] = label_encoder.fit_transform(db_encode['gender'])
db_encode['db_class'] = label_encoder.fit_transform(db_encode['db_class'])

db_encode.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   id          1000 non-null   int64  
 1   no_patient  1000 non-null   int64  
 2   gender      1000 non-null   int64  
 3   age         1000 non-null   int64  
 4   urea        1000 non-null   float64
 5   cr          1000 non-null   int64  
 6   hba1c       1000 non-null   float64
 7   chol        1000 non-null   float64
 8   tg          1000 non-null   float64
 9   hdl         1000 non-null   float64
 10  ldl         1000 non-null   float64
 11  vldl        1000 non-null   float64
 12  bmi         1000 non-null   float64
 13  db_class    1000 non-null   int64  
dtypes: float64(8), int64(6)
memory usage: 109.5 KB
```

```
#Compared the count with db_class count performed during cleaning to ensure consistency and understand the new format.
#Class Attributes are now: 0 = Not Diabetic (previously N) ; 1 = Pre-Diabetic (previously P) ; 2 = Diabetic (previously Y)

db_encode['db_class'].value_counts()
```

db_class	count
2	844
0	103
1	53

dtype: int64

Data Standardization

```python
#Separating the original numercial and label encoded categorical variables to standardize the numerical variables only

num_cols = db_encode.drop(['gender', 'db_class'], axis=1).columns
cat_cols = ['gender', 'db_class']

std_scaler = StandardScaler()
db_encode[num_cols] = std_scaler.fit_transform(db_encode[num_cols])


#Combining the numerical and label encoded categorical variables

db_std = pd.concat([db_encode[num_cols], db_encode[cat_cols]], axis=1)
db_std.head(5)
```

	id	no_patient	age	urea	cr	hba1c	chol	tg	hdl	ldl	vldl	bmi	gender	db_class
0	0.672140	-0.074747	-0.401144	-0.144781	-0.382672	-1.334983	-0.509436	-1.035084	1.810756	-1.085457	-0.369958	-1.124622	0	0
1	1.641852	-0.069940	-3.130017	-0.212954	-0.115804	-1.334983	-0.893730	-0.678063	-0.158692	-0.457398	-0.342649	-1.326239	1	0
2	0.330868	-0.065869	-0.401144	-0.144781	-0.382672	-1.334983	-0.509436	-1.035084	1.810756	-1.085457	-0.369958	-1.124622	0	0
3	1.412950	-0.054126	-0.401144	-0.144781	-0.382672	-1.334983	-0.509436	-1.035084	1.810756	-1.085457	-0.369958	-1.124622	0	0
4	0.680463	-0.069939	-2.334096	0.673299	-0.382672	-1.334983	0.028576	-0.963680	-0.613180	-0.547121	-0.397267	-1.729472	1	0

Next steps: Generate code with db_std | View recommended plots | New interactive sheet

Data Splitting

```python
#Splitting the standardized data into train and test set

train_std_set, test_std_set = train_test_split(db_std, train_size = 0.80, random_state = 89)


#Separating the feature variables and the target variable

X_train = train_std_set.drop('db_class', axis=1)
y_train = train_std_set['db_class']

X_test = test_std_set.drop('db_class', axis=1)
y_test = test_std_set['db_class']
```

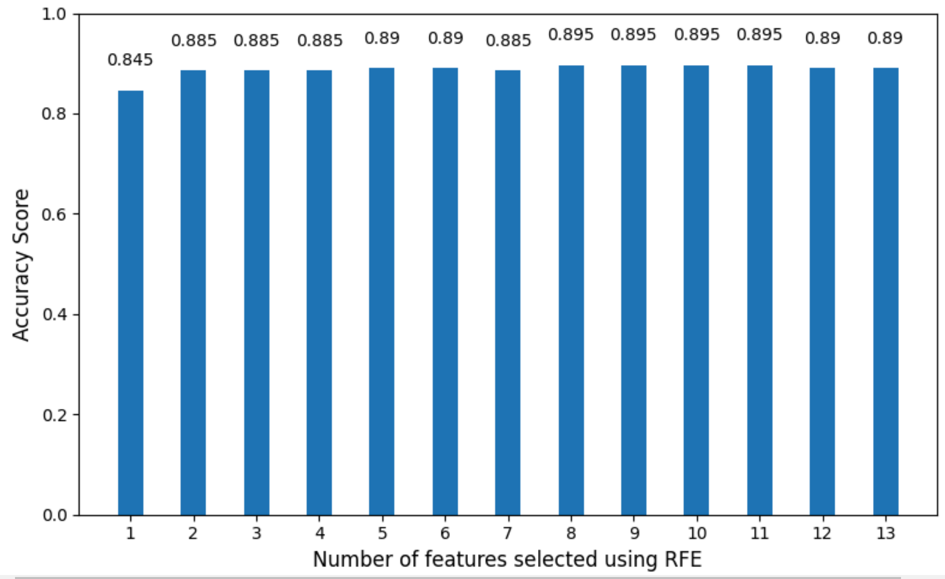
Recursive Feature Selection

```python
#Creating a model to perform recursive feature selection

recursive_model = LogisticRegression(random_state=50)

rfe_selector = RFE(estimator=recursive_model, n_features_to_select=1)
rfe_selector.fit(X_train, y_train)


# Ranking and sorting the the features based on their relevance

feature_ranks = rfe_selector.ranking_

feature_rank_df = pd.DataFrame({'Feature': X_train.columns, 'Rank': feature_ranks})
feature_rank_df.columns = ['Feature', 'Rank']

feature_rank_df = feature_rank_df.sort_values(by='Rank')

print(feature_rank_df)
```

```
       Feature  Rank
5        hba1c     1
11         bmi     2
6         chol     3
12      gender     4
7           tg     5
10        vldl     6
2          age     7
3         urea     8
9          ldl     9
8          hdl    10
4           cr    11
0           id    12
1   no_patient    13
```

```
#Using a loop to iteratively perform recursive feature elimination, removing features one at a time, and calculating the accuracy score at each step

rfe_accuracy_list = []

for k in range(1,14):
  rfe_selector = RFE(estimator=recursive_model, n_features_to_select=k)
  rfe_selector.fit(X_train, y_train)

  selected_features = X_train.columns[rfe_selector.support_]
  X_train_rfe = X_train[selected_features]
  X_test_rfe = X_test[selected_features]

  recursive_model.fit(X_train_rfe, y_train)
  rfe_preds = recursive_model.predict(X_test_rfe)

  accuracy_rfe = round(accuracy_score(y_test, rfe_preds) ,3)

  rfe_accuracy_list.append(accuracy_rfe)
```

```
#Creating chart to show the accuracy score as feature are progressivley removed. This is used to determine the minimum number of features required for op

plt.figure(figsize=(8, 5))
plt.bar(np.arange(1,14), rfe_accuracy_list, width=0.4)

plt.xlabel('Number of features selected using RFE', fontsize=12)
plt.ylabel('Accuracy Score', fontsize=12)

plt.xticks(np.arange(1,14))
plt.ylim(0, 1.0)

for i, v in enumerate(rfe_accuracy_list):
  plt.text(x=i+1, y=v+0.05, s=str(v), ha = 'center')

plt.tight_layout()
```



```
#Updating the number of features to select to 5 and showing the selected fatures

rfe_selector = RFE(estimator=recursive_model, n_features_to_select=8)
rfe_selector.fit(X_train, y_train)

selected_features = X_train.columns[rfe_selector.support_]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['age', 'urea', 'hba1c', 'chol', 'tg', 'vldl', 'bmi', 'gender'], dtype='object')
```

```
#Reducing both the training and test dataset to the five selected features

X_train_rfe = X_train[selected_features]
X_test_rfe = X_test[selected_features]
```

Base Models: Cross Validation and Predicitions

```
#Applying StratifiedKFold for cross validation given the class imbalance in the dataset
#StratifiedKFold ensures that each fold maintains the same proportion of samples for each class and prevents misleading distribution in the validation fo

strat_fold = StratifiedKFold(n_splits=10, random_state=10, shuffle=True)

dc_model = DecisionTreeClassifier(max_depth=5, random_state=10, class_weight = 'balanced')
pred_dc_val = cross_val_predict(dc_model, X_train_rfe, y_train, cv=strat_fold)

lgr_model = LogisticRegression(random_state=10, multi_class='multinomial', class_weight = 'balanced')
pred_lgr_val = cross_val_predict(lgr_model, X_train_rfe, y_train, cv=strat_fold)

knn_model = KNeighborsClassifier(n_neighbors=5, weights='distance')
pred_knn_val = cross_val_predict(knn_model, X_train_rfe, y_train, cv=strat_fold)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
```

Second Level Model: Stacking and Training

```
# Stacking the validation predictions from each base model to form the training set for the second level model

X_seclvl_train = np.column_stack((pred_dc_val, pred_lgr_val, pred_knn_val))
```

```
# Training the second level model, Logistic Regression, with the stacked predictions

seclvl_model = LogisticRegression(random_state=20, multi_class='multinomial', class_weight = 'balanced')
seclvl_model.fit(X_seclvl_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
```

```
LogisticRegression
LogisticRegression(class_weight='balanced', multi_class='multinomial',
                   random_state=20)
```

Final Prediction

```
#Training the base models on all of the training dataset to adequately make predictions on the test set

dc_model.fit(X_train_rfe, y_train)
pred_dc_test = dc_model.predict(X_test_rfe)

lgr_model.fit(X_train_rfe, y_train)
pred_lgr_test = lgr_model.predict(X_test_rfe)

knn_model.fit(X_train_rfe, y_train)
pred_knn_test = knn_model.predict(X_test_rfe)
```

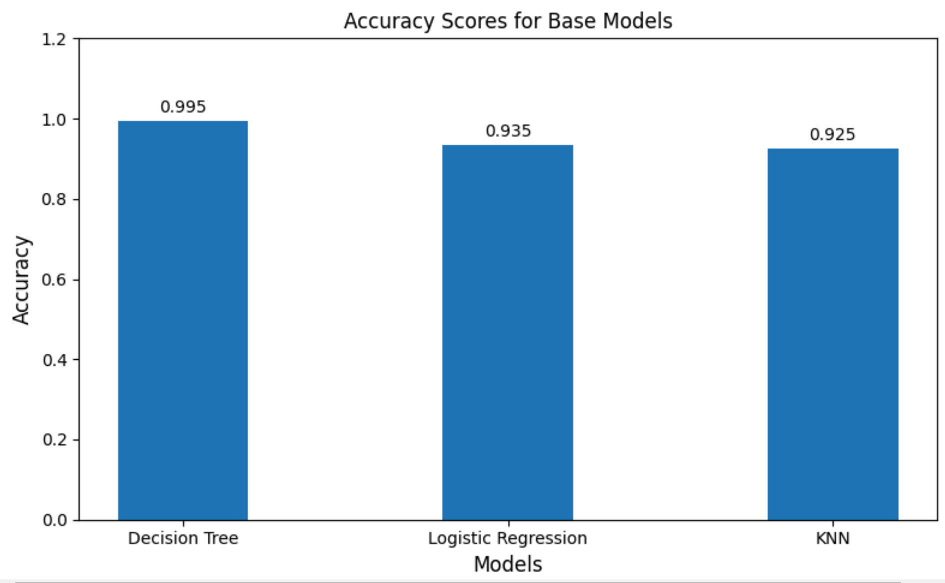
```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be r
  warnings.warn(
```

```
# Stacking the predictions made by the base models to create a new testing dataset for the second level model
X_seclvl_test = np.column_stack((pred_dc_test, pred_lgr_test, pred_knn_test))

# Final prediction made using the second level model
pred_final = seclvl_model.predict(X_seclvl_test)
```

Stage 5: Model Evaluation

Base Models Evaluation

```
#Creating barchart to show the accuracy score among the base models

accuracy_dc = accuracy_score(y_test, pred_dc_test)
accuracy_lgr = accuracy_score(y_test, pred_lgr_test)
accuracy_knn = accuracy_score(y_test, pred_knn_test)

accuracy_base_scores = [accuracy_dc, accuracy_lgr, accuracy_knn]
model_names_final = ['Decision Tree', 'Logistic Regression', 'KNN']

plt.figure(figsize=(8, 5))
plt.bar(model_names_final, accuracy_base_scores, width=0.4)

plt.xlabel('Models', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Accuracy Scores for Base Models', fontsize=12)
plt.ylim(0, 1.2)

for i, v in enumerate(accuracy_base_scores):
    plt.text(i, v + 0.02, f"{v:.3f}", ha='center', fontsize=10)

plt.tight_layout()
plt.show()
```



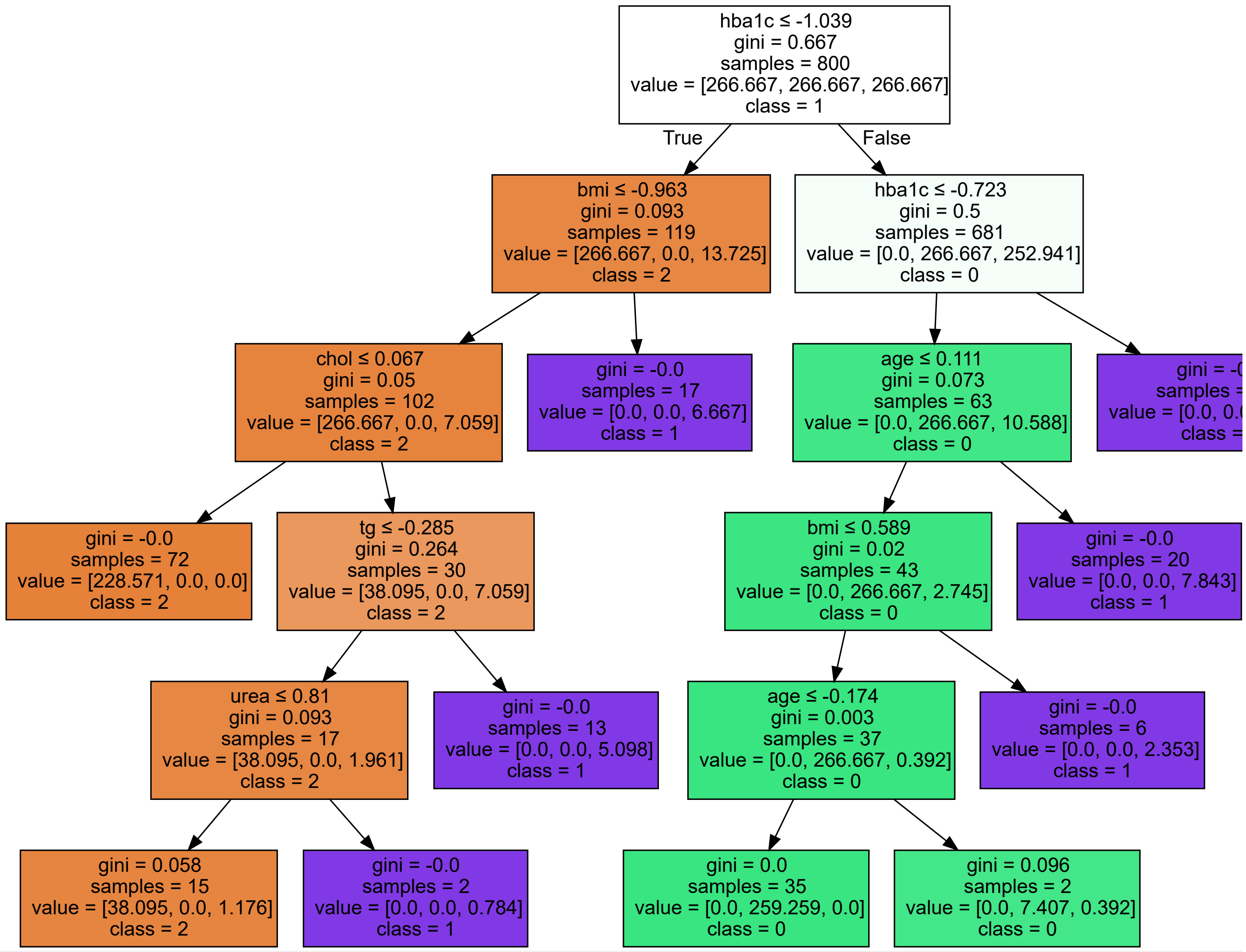
```
cf_dc =confusion_matrix(y_test, pred_dc_test)
print("Confusion Matrix for Decision Tree Base Model:\n")
print(cf_dc)

print('\n')

print("Performance Metrics for Decision Tree Base Model:\n")
print(classification_report(y_test, pred_dc_test))
```

```
Confusion Matrix for Decision Tree Base Model:

[[ 19   0   0]
 [  0  16   1]
 [  0   0 164]]
```

```
Performance Metrics for Decision Tree Base Model:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.94      0.97        17
           2       0.99      1.00      1.00       164

    accuracy                           0.99       200
   macro avg       1.00      0.98      0.99       200
weighted avg       1.00      0.99      0.99       200
```

```
class_values = y_train.unique()  # Or, if it's in a DataFrame, use df['Class'].unique()
print("Class Names:", class_values)

# Visualize the Decision Tree
dot_data = tree.export_graphviz(dc_model, out_file=None,
                                feature_names=X_train_rfe.columns,  # Use the column names for features
                                class_names=[str(c) for c in class_values],  # Class names as strings
                                filled=True,  # Color the nodes based on their class
                                special_characters=True)  # Handle special characters in feature names

# Generate and render the graph
graph = graphviz.Source(dot_data, format="png")
graph
```

```
Class Names: [2 0 1]
```



```
cf_lgr =confusion_matrix(y_test, pred_lgr_test)
print("Confusion Matrix for Logisitic Regression Base Model:\n")
print(cf_lgr)

print('\n')

print("Performance Metrics for Logisitic Regression Base Model:\n")
print(classification_report(y_test, pred_lgr_test))
```

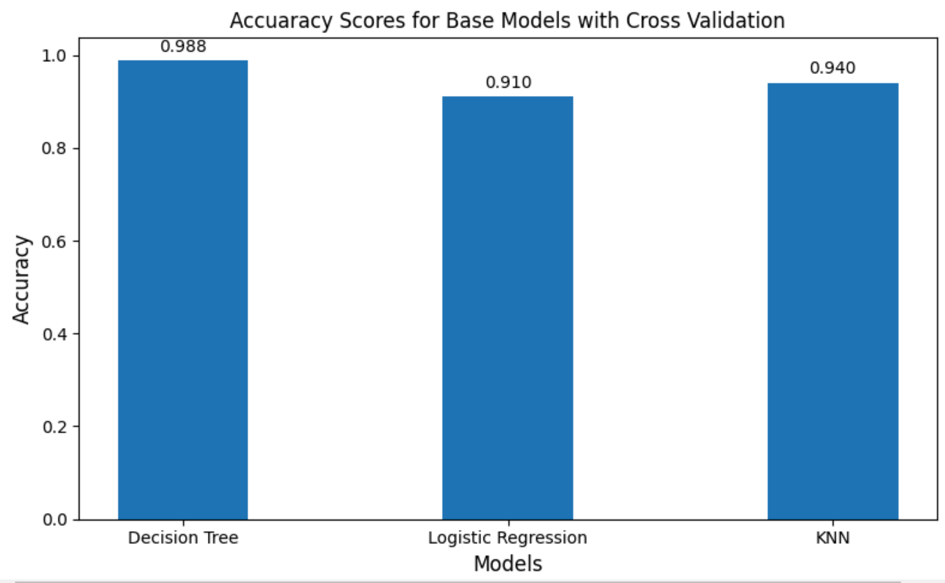
```
Confusion Matrix for Logisitic Regression Base Model:

[[ 17   2   0]
 [  0  17   0]
 [  4   7 153]]


Performance Metrics for Logisitic Regression Base Model:

              precision    recall  f1-score   support

           0       0.81      0.89      0.85        19
           1       0.65      1.00      0.79        17
           2       1.00      0.93      0.97       164

    accuracy                           0.94       200
   macro avg       0.82      0.94      0.87       200
weighted avg       0.95      0.94      0.94       200
```

```python
cf_knn =confusion_matrix(y_test, pred_knn_test)
print("Confusion Matrix for kNN Base Model:\n")
print(cf_knn)

print('\n')

print("Performance Metrics for kNN Base Model:\n")
print(classification_report(y_test, pred_knn_test))
```

```
Confusion Matrix for kNN Base Model:

[[ 15   1   3]
 [  2  12   3]
 [  2   4 158]]


Performance Metrics for kNN Base Model:

              precision    recall  f1-score   support

           0       0.79      0.79      0.79        19
           1       0.71      0.71      0.71        17
           2       0.96      0.96      0.96       164

    accuracy                           0.93       200
   macro avg       0.82      0.82      0.82       200
weighted avg       0.93      0.93      0.93       200
```

Base Model Performance with Cross Validation

```python
#Creating barchart to show the accuracy score among the cross validated base models

acc_dc_val = accuracy_score(y_train, pred_dc_val)
acc_lgr_val = accuracy_score(y_train, pred_lgr_val)
acc_knn_val = accuracy_score(y_train, pred_knn_val)

acc_val = [acc_dc_val, acc_lgr_val, acc_knn_val]
model_names_val = ['Decision Tree', 'Logistic Regression', 'KNN']

plt.figure(figsize=(8, 5))
plt.bar(model_names_val, acc_val, width=0.4)

plt.xlabel('Models', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Accuaracy Scores for Base Models with Cross Validation', fontsize=12)

for i, v in enumerate(acc_val):
    plt.text(i, v + 0.02, f"{v:.3f}", ha='center', fontsize=10)

plt.tight_layout()
plt.show()
```



```
cf_dc_val = confusion_matrix(y_train, pred_dc_val)
print("Confusion Matrix for Decision Tree Base Model with Cross Validation:\n")
print(cf_dc_val)

print('\n')

print("Performance Metrics for Decision Tree Base Model with Cross Validation:\n")
print(classification_report(y_train, pred_dc_val))
```

```
Confusion Matrix for Decision Tree Base Model with Cross Validation:

[[ 81   0   3]
 [  0  34   2]
 [  4   1 675]]


Performance Metrics for Decision Tree Base Model with Cross Validation:

              precision    recall  f1-score   support

           0       0.95      0.96      0.96        84
           1       0.97      0.94      0.96        36
           2       0.99      0.99      0.99       680

    accuracy                           0.99       800
   macro avg       0.97      0.97      0.97       800
weighted avg       0.99      0.99      0.99       800
```

```
cf_lgr_val =confusion_matrix(y_train, pred_lgr_val)
print("Confusion Matrix for Logisitic Regression Base Model with Cross Validation:\n")
print(cf_lgr_val)

print('\n')

print("Performance Metrics for Logisitic Regression Base Model with Cross Validation:\n")
print(classification_report(y_train, pred_lgr_val))
```

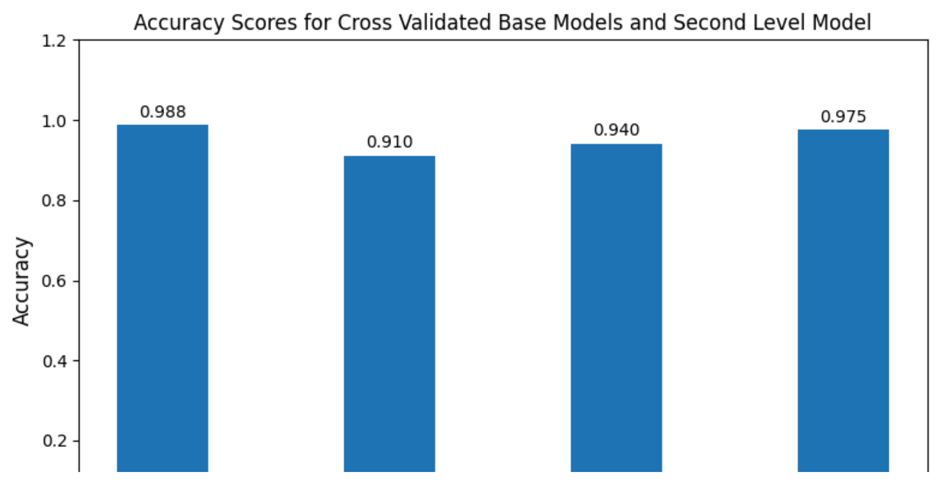
```
Confusion Matrix for Logisitic Regression Base Model with Cross Validation:

[[ 74  10   0]
 [  3  31   2]
 [ 23  34 623]]


Performance Metrics for Logisitic Regression Base Model with Cross Validation:

              precision    recall  f1-score   support

           0       0.74      0.88      0.80        84
           1       0.41      0.86      0.56        36
           2       1.00      0.92      0.95       680

    accuracy                           0.91       800
   macro avg       0.72      0.89      0.77       800
weighted avg       0.94      0.91      0.92       800
```

```
cf_knn_val =confusion_matrix(y_train, pred_knn_val)
print("Confusion Matrix for kNN Base Model with Cross Validation:\n")
print(cf_knn_val)

print('\n')

print("Performance Metrics for kNN Base Model with Cross Validation:\n"
print(classification report(v train, pred knn val))
```

```
Confusion Matrix for kNN Base Model with Cross Validation:

[[ 62   9  13]
 [  7  24   5]
 [  8   6 666]]


Performance Metrics for kNN Base Model with Cross Validation:

              precision    recall  f1-score   support

           0       0.81      0.74      0.77        84
           1       0.62      0.67      0.64        36
           2       0.97      0.98      0.98       680

    accuracy                           0.94       800
   macro avg       0.80      0.79      0.80       800
weighted avg       0.94      0.94      0.94       800
```

Second Level Model Evaluation

```
#Creating barchart to compare the accuracy scores for the base and second level models

accuracy_seclvl = accuracy_score(y_test, pred_final)

accuracy_scores = [acc_dc_val , acc_lgr_val, acc_knn_val, accuracy_seclvl ]
model_names_final = ['Decision Tree', 'Logistic Regression', 'KNN', 'Second-Level Model']


plt.figure(figsize=(8, 5))
plt.bar(model_names_final, accuracy_scores, width=0.4)

plt.xlabel('Models', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Accuracy Scores for Cross Validated Base Models and Second Level Model', fontsize=12)
plt.ylim(0, 1.2)

for i, v in enumerate(accuracy_scores):
    plt.text(i, v + 0.02, f"{v:.3f}", ha='center', fontsize=10)

plt.tight_layout()
plt.show()
```


Accuracy Scores for Cross Validated Base Models and Second Level Model
Accuracy