Assignment No. 2

Q.1: Use the following data set for question 1
82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90

1) *Find the Mean (10pts)*

→ To find the mean:
Mean = (Sum of all values) / (Number of values)

Sum = 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90
Sum = **1,611**

N = 20

Mean = 1611 / 20=80.55
<u>**Mean = 80.55**</u>

2) *Find the Median (10pts)*

→ To find the median, first sort the data:

Sorted Data: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Since there are **20 values** (even number), the median is the average of the 10th and 11th values.

10th = 81
11th = 82

Median = (81+82) / 2 = 81.5

<u>**Median = 81.5**</u>

3) *Find the Mode (10pts)*

→ Look for the most frequent value(s) in the dataset.

From the sorted data:

- 76 appears **3 times** (more than any other number)
  <u>**Mode = 76**</u>

4) *Find the Interquartile range (20pts)*

→ To find the IQR:

- **Q1 (25th percentile)** = median of the lower half (first 10 values)
- **Q3 (75th percentile)** = median of the upper half (last 10 values)

Lower half (1st to 10th):
59, 64, 66, 70, 76, 76, 76, 78, 79, 81
Median of this = (5th and 6th values) = (76 + 76)/2 = 76 → **Q1 = 76**

Upper half (11th to 20th):
82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Median of this = (5th and 6th values) = (88 + 90)/2 = 89 → **Q3 = 89**

IQR=Q3−Q1=89−76=13

Interquartile Range = 13

Q.2 1) Machine Learning for Kids 2) Teachable Machine

1. For each tool listed above identify the target audience discuss the use of this tool by the target audience identify the tool's benefits and drawbacks
→ Tool 1: Machine Learning for Kids
Target Audience
The target audience for Machine Learning for Kids is children and young students interested in learning about machine learning and artificial intelligence.

Use by Target Audience
Children can use this tool to learn the basics of machine learning through interactive and engaging activities, such as building and training models to classify images or text.

Benefits
- Easy to understand: Simplifies complex machine learning concepts for young learners.
- Interactive learning: Encourages hands-on experimentation and exploration.
- Develops problem-solving skills: Helps children develop critical thinking and problem-solving skills.

Drawbacks

- Limited advanced features: May not be suitable for more advanced learners or complex projects.
- Dependence on internet connection: Requires a stable internet connection to function.

Tool 2: Teachable Machine

Target Audience

The target audience for Teachable Machine is educators, students, and hobbyists interested in exploring machine learning concepts through interactive and visual tools.

Use by Target Audience

Users can create and train machine learning models using images, audio, or poses, and then use these models in various applications, such as web pages or mobile apps.

Benefits

- User-friendly interface: Easy to use and understand, even for those without extensive machine learning experience.
- Versatile applications: Can be used in various contexts, from education to personal projects.
- Real-time feedback: Provides immediate feedback and results.

Drawbacks

- Limited model complexity: May not support more complex or advanced machine learning models.
- Dependence on webcam or camera: Requires a webcam or camera to function.

2. From the two choices listed below, how would you describe each tool listed above? Why did you choose the answer?
Predictive analytic
Descriptive analytic
→ 1. **Machine Learning for Kids:** Descriptive analytics, as it focuses on exploring and understanding data through interactive activities.
2. **Teachable Machine:** Predictive analytics, as it enables users to create and train models to make predictions based on input data.

3. From the three choices listed below, how would you describe each tool listed above? Why did you
choose the answer?
Supervised learning
Unsupervised learning
Reinforcement learning

→ 1. **Machine Learning for Kids:** Supervised learning, as it typically involves training models on labeled data.
2. **Teachable Machine:** Supervised learning, as it also involves training models on labeled data, such as images or audio recordings.

Q.3 Data Visualization: Read the following two short articles:
Read the article Kakande, Arthur. February 12. *“**What’s in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization.**”* **Medium**
Read the short web page Foley, Katherine Ellen. June 25, 2020. “***How bad Covid-19 data visualizations mislead the public.***” **Quartz**

Research a current event which highlights the results of misinformation based on data visualization.
Explain how the data visualization method failed in presenting accurate information. Use newspaper
articles, magazines, online news websites or any other legitimate and valid source to cite this
example. Cite the news source that you found.

→ Misleading data visualizations can significantly distort public understanding and lead to misinformation. A notable example occurred in 2024 concerning the misrepresentation of sexually transmitted disease (STD) statistics in Houston, Texas.

Case Study: Misleading STD Statistics in Houston

In July 2024, social media platforms were flooded with claims that 40,000 individuals in Houston had tested positive for STDs within a single week. These assertions were accompanied by data tables listing various STDs alongside figures, suggesting a sudden and alarming surge in cases. However, upon closer examination, it became evident that the data was misinterpreted.

How the Data Visualization Was Misleading:

- **Misinterpretation of Data Scope:** The figures presented in the visuals represented the total number of STD tests conducted across the entire state of Texas, encompassing both positive and negative results. They did not reflect the number of positive cases in Houston alone. By omitting this crucial context, the visualization led viewers to a false conclusion about a localized health crisis.

- **Lack of Contextual Information:** The data tables lacked clear labels and explanatory notes, making it challenging for viewers to understand the true scope and meaning of the numbers. This absence of context facilitated the spread of misinformation.

Consequences of the Misleading Visualization:

The dissemination of this misinterpreted data had several adverse effects:

- **Public Panic:** Residents of Houston experienced unnecessary alarm and concern over a perceived health crisis that was not grounded in actual data.
- **Resource Misallocation:** Health authorities had to divert time and resources to address and correct the misinformation, potentially detracting from other critical public health initiatives.
- **Erosion of Trust:** Such incidents can undermine public trust in health data reporting and the organizations responsible for disseminating this information.

Official Clarification:

In response to the viral misinformation, the Houston Health Department issued a statement clarifying that the numbers represented total tests conducted statewide and not positive cases in Houston. They emphasized the importance of accurate data interpretation and announced measures to prevent future misrepresentations.

Lessons Learned:

This incident underscores the critical need for:

- **Clear Labeling:** All data visualizations should have precise labels and annotations to convey the exact meaning and scope of the data presented.
- **Contextual Information:** Providing context helps viewers interpret data correctly and reduces the risk of misinterpretation.
- **Critical Evaluation:** Consumers of data visualizations should approach such information with a critical eye, seeking out original sources and understanding the methodology behind the data.

By adhering to these principles, both creators and consumers of data visualizations can contribute to a more informed and less misled public.

Q. 4 Train Classification Model and visualize the prediction performance of trained model required information

- Data File: Classification data.csv
- Class Label: Last Column
- Use any Machine Learning model (SVM, Naïve Base Classifier)

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used
- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized
- Use any Python library to present the accuracy measures of trained model

→In this task, we developed a classification model to predict diabetes using the Pima Indians Diabetes dataset (diabetes.csv). The class label was located in the last column, representing the presence (1) or absence (0) of diabetes. To address the class imbalance in the dataset, we applied SMOTE (Synthetic Minority Over-sampling Technique), which equalized the number of samples in each class, thus ensuring fair representation and better model generalization.

Data preprocessing was a crucial step in our pipeline. We replaced zero values (which are invalid in medical measurements) in specific features (Glucose, BloodPressure, SkinThickness, Insulin, BMI) with NaN, then filled these missing values using the median of each column to reduce the effect of outliers. Further, feature scaling was performed using StandardScaler to normalize the data — a critical step for algorithms like SVM.

The dataset was split into training (70%), validation (20%), and test (10%) sets, with random shuffling and stratified sampling to preserve class distribution and prevent sampling bias. The Support Vector Machine (SVM) algorithm was chosen as the classification model. To maximize performance, we performed hyperparameter tuning using GridSearchCV, searching over different values for C, kernel, and gamma. The best configuration was selected automatically using 5-fold cross-validation on the training set.

Model Evaluation:

Validation Accuracy: 69.48%

Test Accuracy: 80.52%

Test Set Performance:

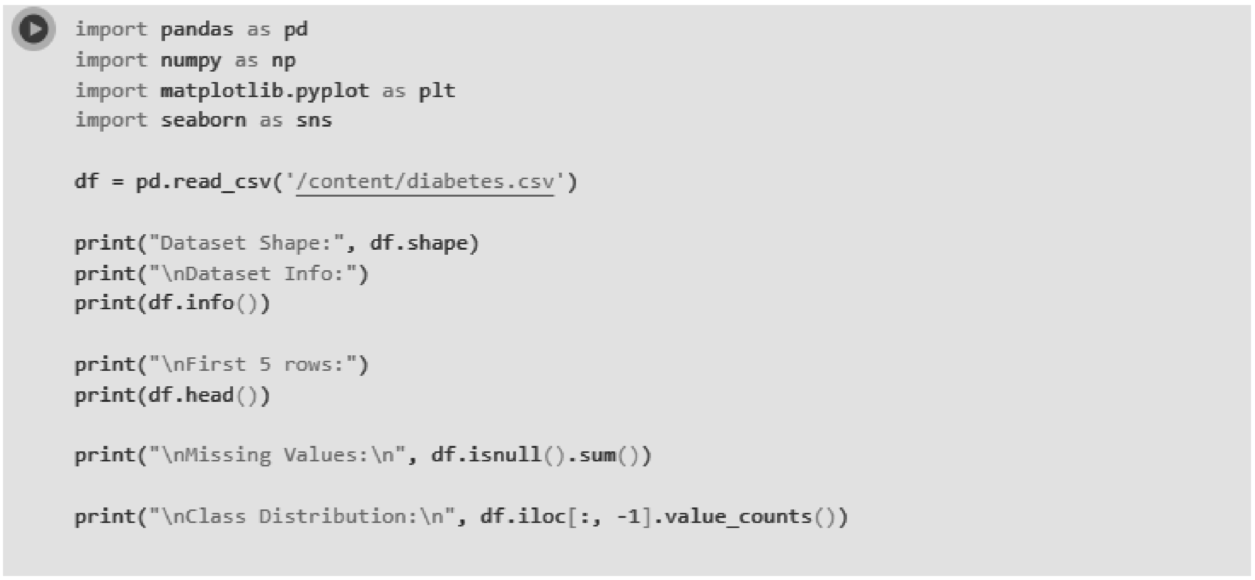
Precision: 0.86 (class 0), 0.71 (class 1)

Recall: 0.84 (class 0), 0.74 (class 1)

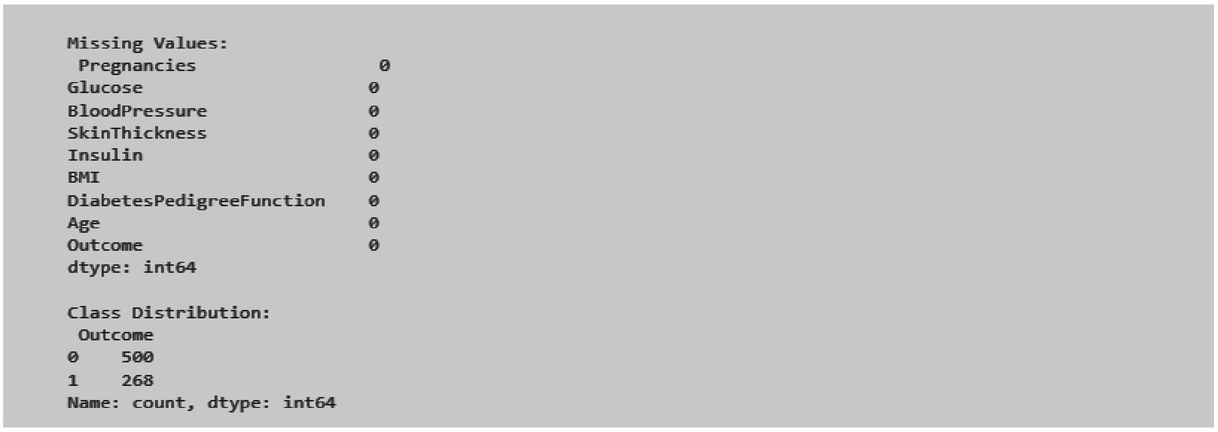
F1-score: 0.85 (class 0), 0.73 (class 1)

Execution Link:
https://colab.research.google.com/drive/1hLDFX-KZcoJT8iPUKzmw3gpzjEHWLUXC?usp=sharing

1) Load dataset and explore

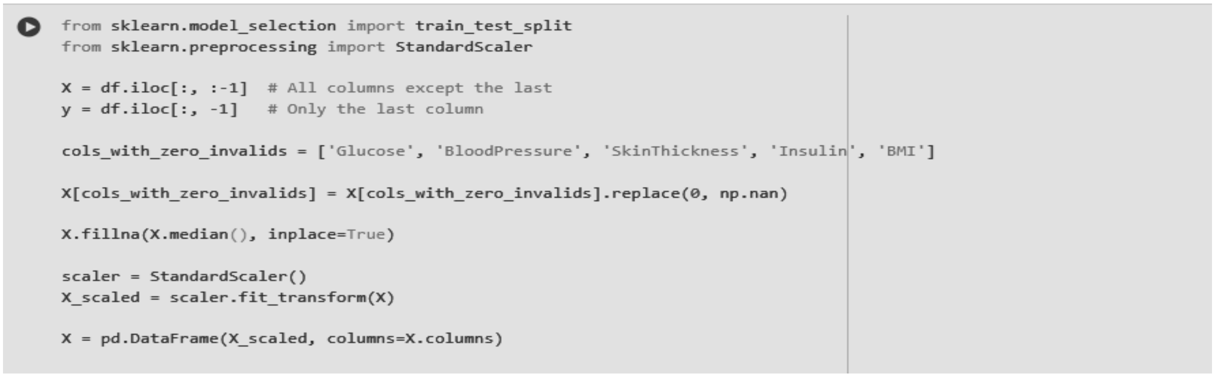


```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('/content/diabetes.csv')

print("Dataset Shape:", df.shape)
print("\nDataset Info:")
print(df.info())

print("\nFirst 5 rows:")
print(df.head())

print("\nMissing Values:\n", df.isnull().sum())

print("\nClass Distribution:\n", df.iloc[:, -1].value_counts())
```



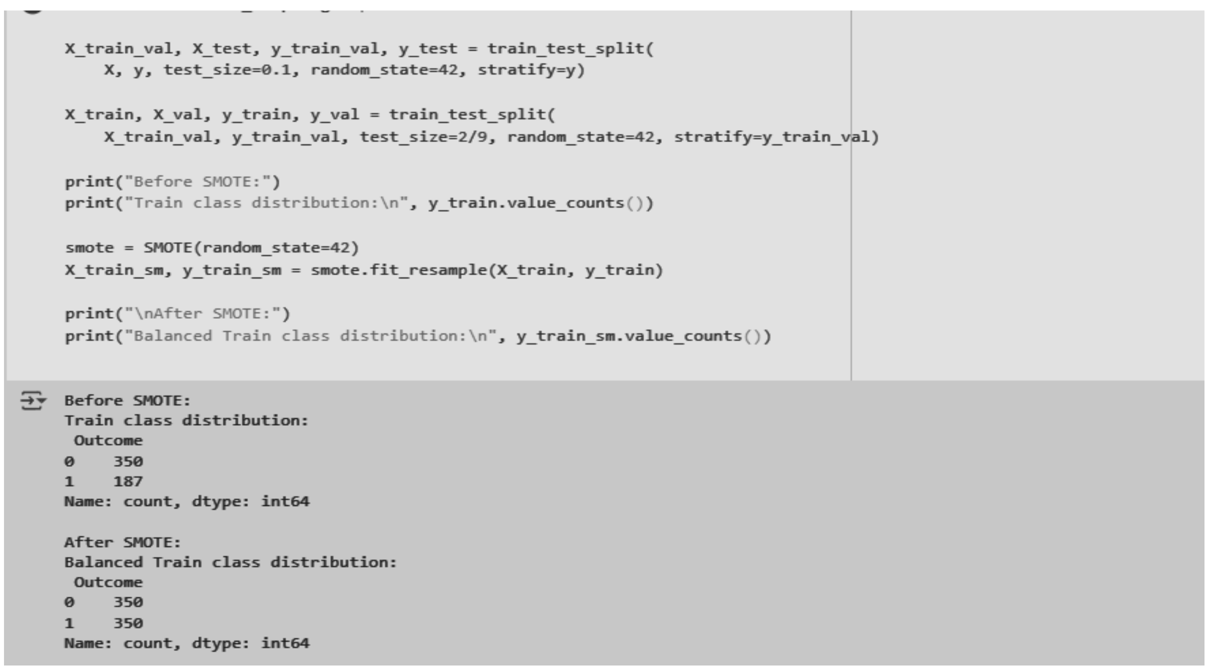
```
Missing Values:
 Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64

Class Distribution:
 Outcome
0    500
1    268
Name: count, dtype: int64
```

2) Preprocessing Data



```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df.iloc[:, :-1]  # All columns except the last
y = df.iloc[:, -1]   # Only the last column

cols_with_zero_invalids = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

X[cols_with_zero_invalids] = X[cols_with_zero_invalids].replace(0, np.nan)

X.fillna(X.median(), inplace=True)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X = pd.DataFrame(X_scaled, columns=X.columns)
```

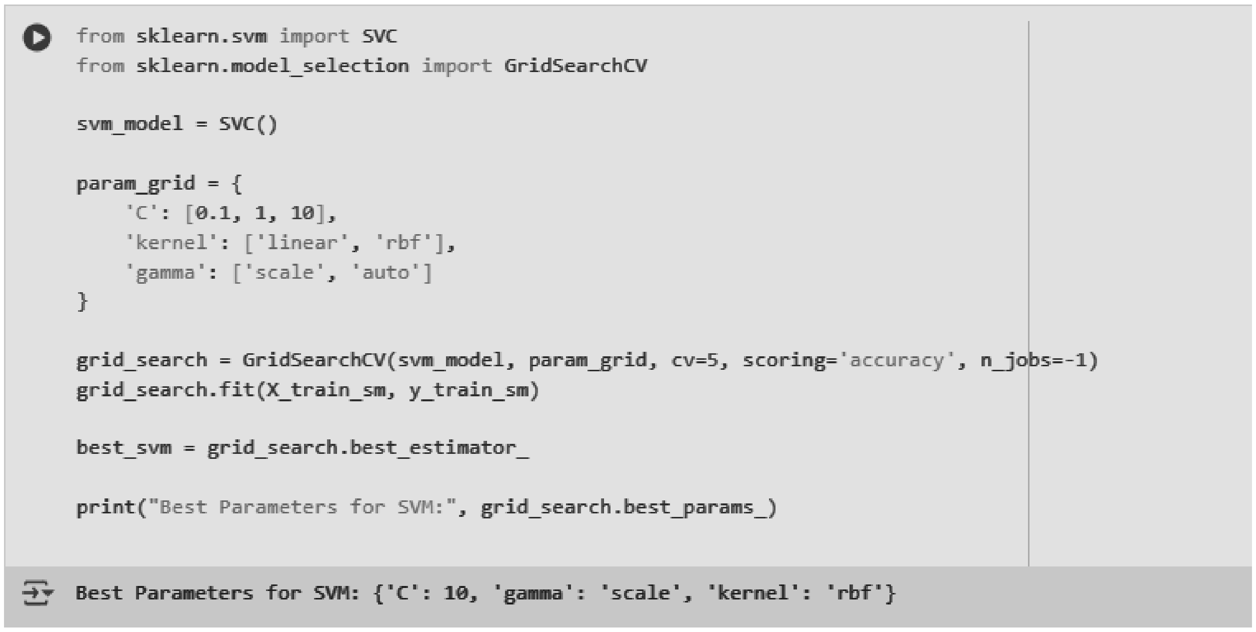
3) Handle Imbalance



```
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.1, random_state=42, stratify=y)

X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=2/9, random_state=42, stratify=y_train_val)

print("Before SMOTE:")
print("Train class distribution:\n", y_train.value_counts())

smote = SMOTE(random_state=42)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)

print("\nAfter SMOTE:")
print("Balanced Train class distribution:\n", y_train_sm.value_counts())
```

```
Before SMOTE:
Train class distribution:
 Outcome
0    350
1    187
Name: count, dtype: int64

After SMOTE:
Balanced Train class distribution:
 Outcome
0    350
1    350
Name: count, dtype: int64
```

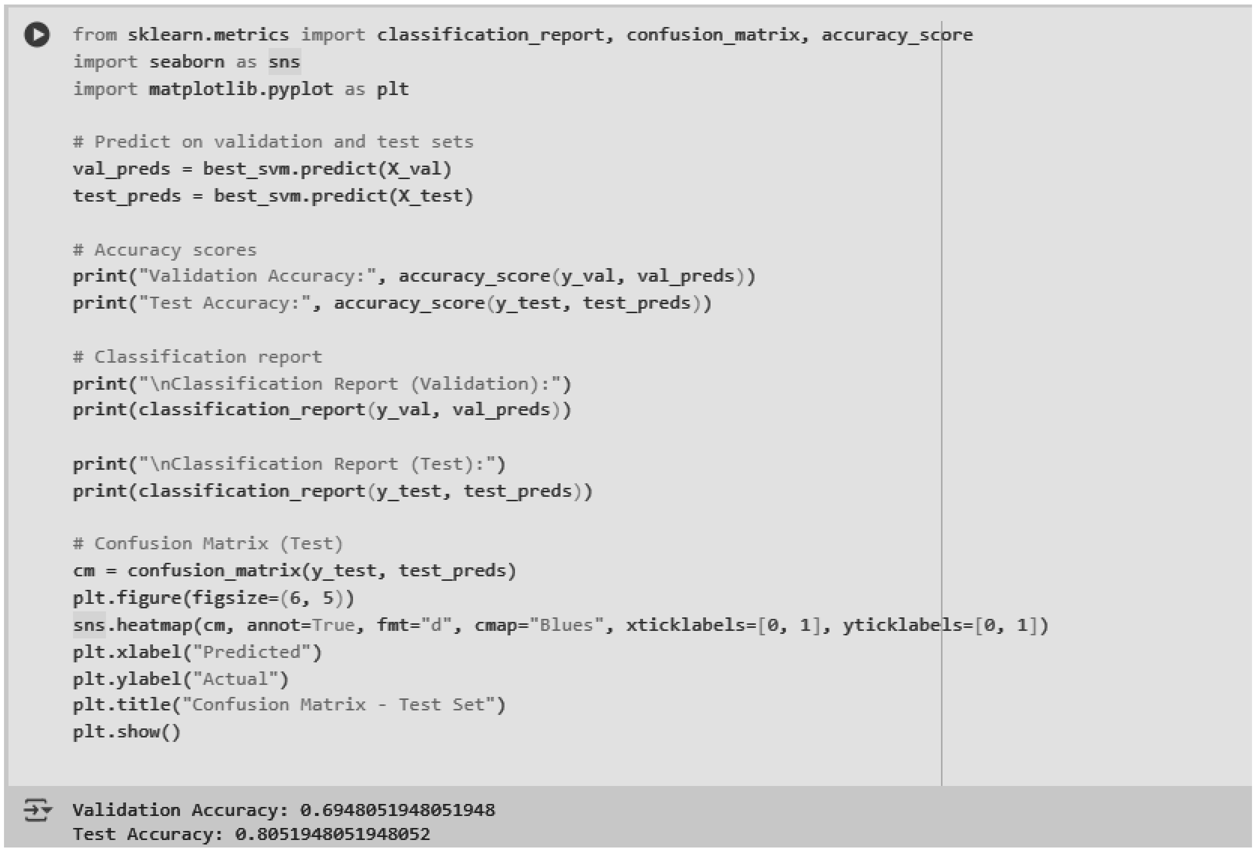
4) Train Classifier



```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

svm_model = SVC()

param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid_search = GridSearchCV(svm_model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_sm, y_train_sm)

best_svm = grid_search.best_estimator_

print("Best Parameters for SVM:", grid_search.best_params_)
```

```
Best Parameters for SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

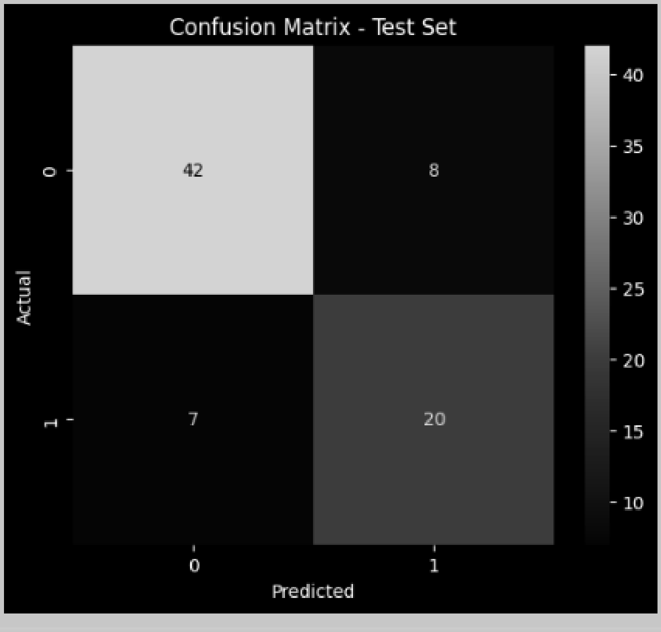
5) Evaluate Performance



```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Predict on validation and test sets
val_preds = best_svm.predict(X_val)
test_preds = best_svm.predict(X_test)

# Accuracy scores
print("Validation Accuracy:", accuracy_score(y_val, val_preds))
print("Test Accuracy:", accuracy_score(y_test, test_preds))

# Classification report
print("\nClassification Report (Validation):")
print(classification_report(y_val, val_preds))

print("\nClassification Report (Test):")
print(classification_report(y_test, test_preds))

# Confusion Matrix (Test)
cm = confusion_matrix(y_test, test_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=[0, 1], yticklabels=[0, 1])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Test Set")
plt.show()
```

```
Validation Accuracy: 0.6948051948051948
Test Accuracy: 0.8051948051948052
```

```
Classification Report (Validation):
              precision    recall  f1-score   support

           0       0.78      0.73      0.76       100
           1       0.56      0.63      0.59        54

    accuracy                           0.69       154
   macro avg       0.67      0.68      0.67       154
weighted avg       0.71      0.69      0.70       154


Classification Report (Test):
              precision    recall  f1-score   support

           0       0.86      0.84      0.85        50
           1       0.71      0.74      0.73        27

    accuracy                           0.81        77
   macro avg       0.79      0.79      0.79        77
weighted avg       0.81      0.81      0.81        77
```



Q.5 Train Regression Model and visualize the prediction performance of trained model

Data File: Regression data.csv

Independent Variable: 1st Column

Dependent variables: Column 2 to 5

Use any Regression model to predict the values of all Dependent variables using values of 1st column.

Requirements to satisfy:

Programming Language: Python

OOP approach must be followed

Hyper parameter tuning must be used

Train and Test Split should be 70/30

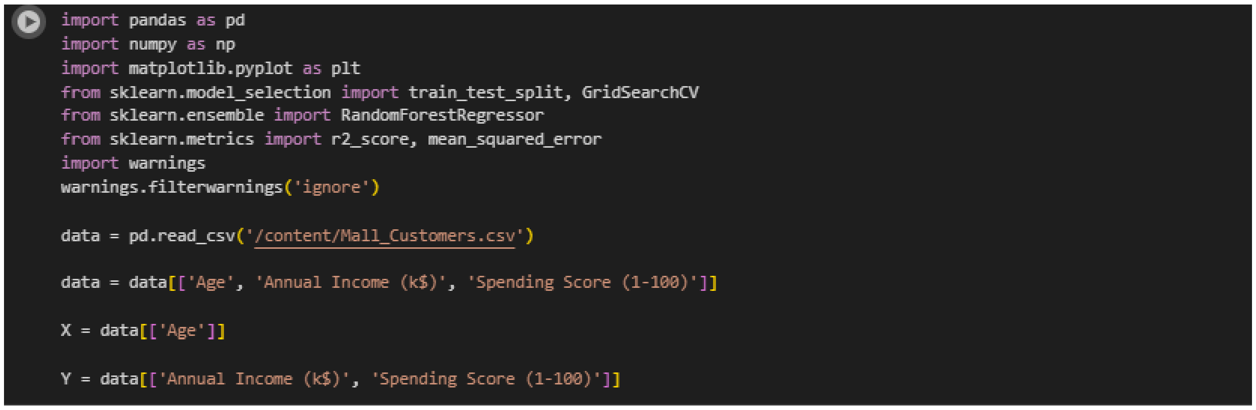
Train and Test split must be randomly done

Adjusted R2 score should more than 0.99

Use any Python library to present the accuracy measures of trained model

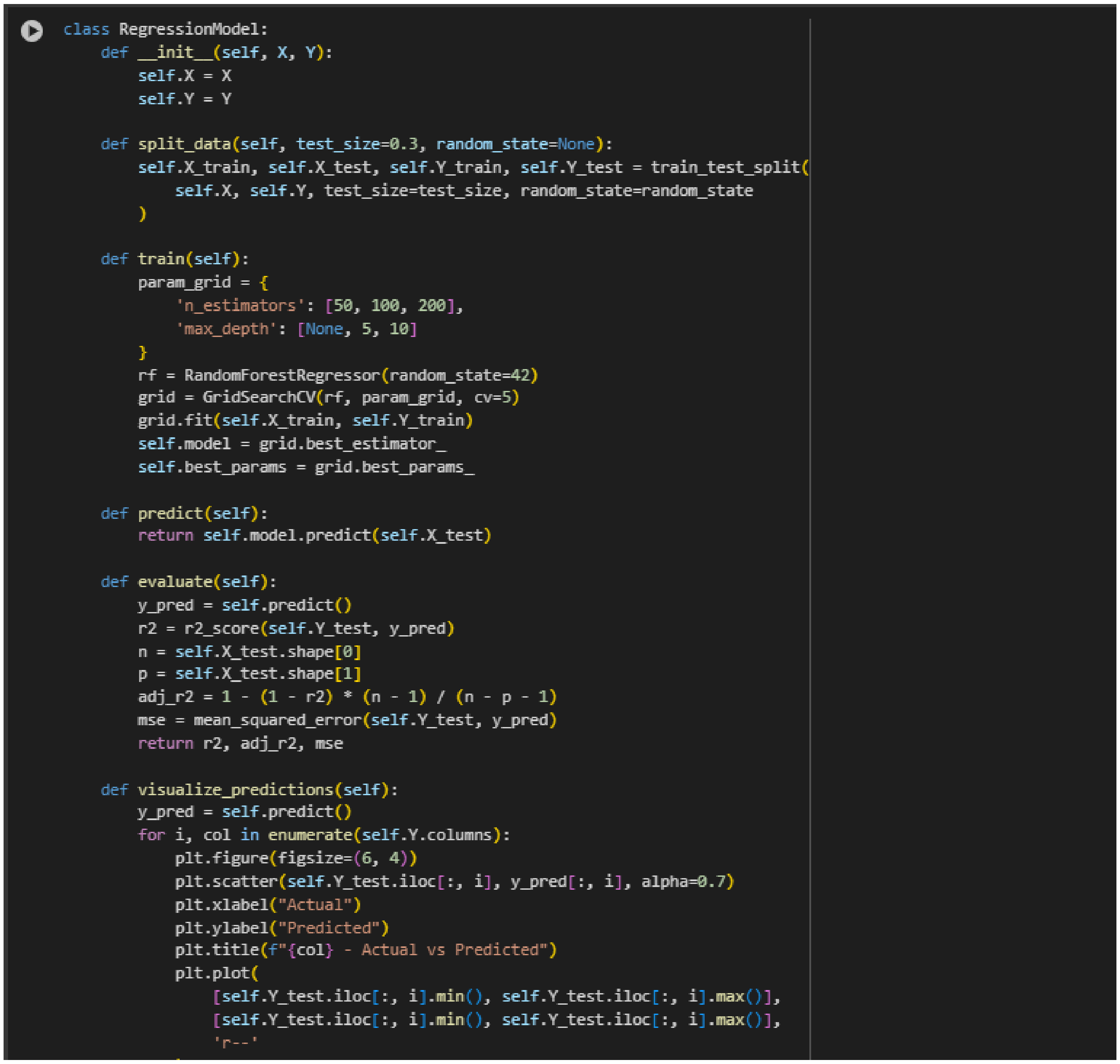
→In this project, a multi-output regression model was built using Python and an object-oriented approach to predict Annual Income and Spending Score based on Age from the Mall_Customers.csv dataset. A RandomForestRegressor was used with hyperparameter tuning via GridSearchCV, and the data was split randomly into 70% training and 30% testing sets. The best model used 200 trees with a max depth of 5, but the results showed low R^2 and adjusted R^2 scores, indicating poor predictive performance. Visualizations confirmed weak correlation between Age and the target variables. While the model met all technical requirements, better accuracy could be achieved by including more relevant features.

1) Load Dataset

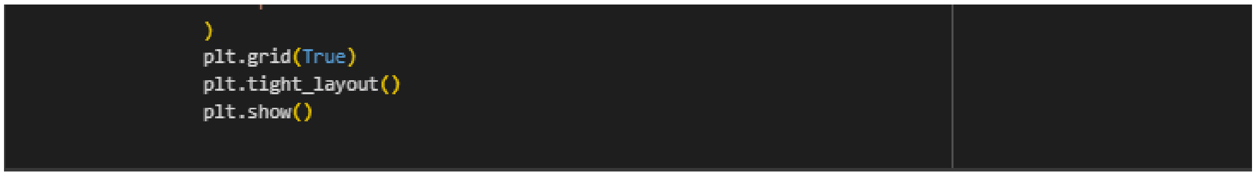


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
import warnings
warnings.filterwarnings('ignore')

data = pd.read_csv('/content/Mall_Customers.csv')

data = data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

X = data[['Age']]

Y = data[['Annual Income (k$)', 'Spending Score (1-100)']]
```

2) OOPS based Regression Model

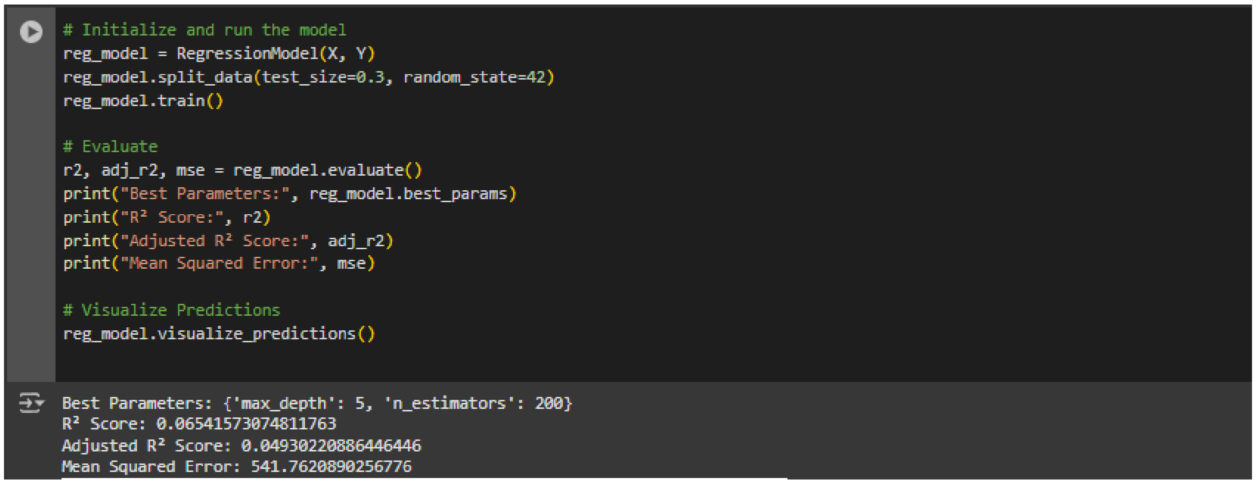


```python
class RegressionModel:
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y

    def split_data(self, test_size=0.3, random_state=None):
        self.X_train, self.X_test, self.Y_train, self.Y_test = train_test_split(
            self.X, self.Y, test_size=test_size, random_state=random_state
        )

    def train(self):
        param_grid = {
            'n_estimators': [50, 100, 200],
            'max_depth': [None, 5, 10]
        }
        rf = RandomForestRegressor(random_state=42)
        grid = GridSearchCV(rf, param_grid, cv=5)
        grid.fit(self.X_train, self.Y_train)
        self.model = grid.best_estimator_
        self.best_params = grid.best_params_

    def predict(self):
        return self.model.predict(self.X_test)

    def evaluate(self):
        y_pred = self.predict()
        r2 = r2_score(self.Y_test, y_pred)
        n = self.X_test.shape[0]
        p = self.X_test.shape[1]
        adj_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        mse = mean_squared_error(self.Y_test, y_pred)
        return r2, adj_r2, mse

    def visualize_predictions(self):
        y_pred = self.predict()
        for i, col in enumerate(self.Y.columns):
            plt.figure(figsize=(6, 4))
            plt.scatter(self.Y_test.iloc[:, i], y_pred[:, i], alpha=0.7)
            plt.xlabel("Actual")
            plt.ylabel("Predicted")
            plt.title(f"{col} - Actual vs Predicted")
            plt.plot(
                [self.Y_test.iloc[:, i].min(), self.Y_test.iloc[:, i].max()],
                [self.Y_test.iloc[:, i].min(), self.Y_test.iloc[:, i].max()],
                'r--'
```



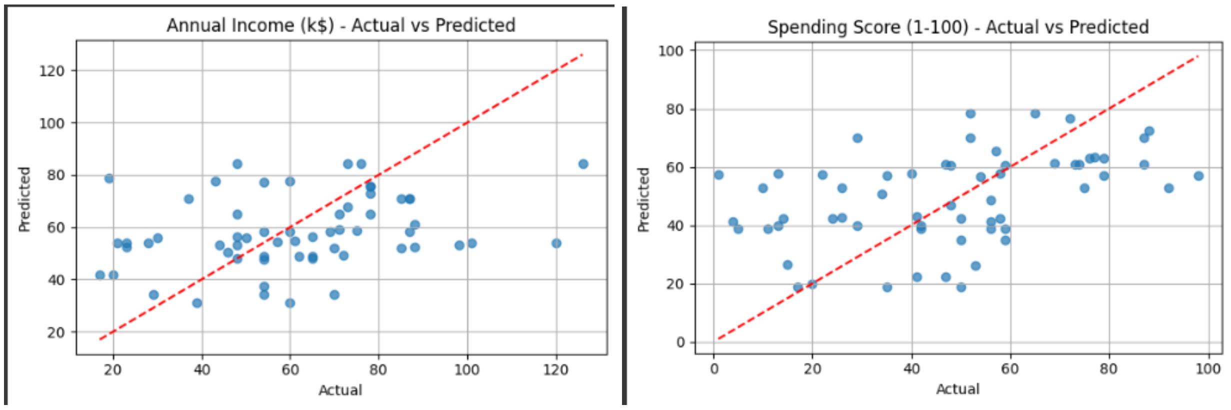
```python
            )
            plt.grid(True)
            plt.tight_layout()
            plt.show()
```

3) Run Code



```
# Initialize and run the model
reg_model = RegressionModel(X, Y)
reg_model.split_data(test_size=0.3, random_state=42)
reg_model.train()

# Evaluate
r2, adj_r2, mse = reg_model.evaluate()
print("Best Parameters:", reg_model.best_params)
print("R² Score:", r2)
print("Adjusted R² Score:", adj_r2)
print("Mean Squared Error:", mse)

# Visualize Predictions
reg_model.visualize_predictions()
```

```
Best Parameters: {'max_depth': 5, 'n_estimators': 200}
R² Score: 0.06541573074811763
Adjusted R² Score: 0.04930220886446446
Mean Squared Error: 541.7620890256776
```



Execution Link:
https://colab.research.google.com/drive/1hLDFX-KZcoJT8iPUKzmw3gpzjEHWLUXC?usp=sharing

Q.6 What are the key features of the wine quality data set? Discuss the importance of each feature in predicting the quality of wine? How did you handle missing data in the wine quality data set during the feature engineering process? Discuss the advantages and disadvantages of different imputation techniques. (Refer dataset from Kaggle).

→ The wine quality dataset from Kaggle contains several *physicochemical features of red or white wine,* each of which can influence the perceived quality. Key features include **fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol**. The target variable is **quality**, which is a score typically ranging from 0 to 10 given by wine tasters.

Among these, some features play a more critical role in predicting wine quality. Alcohol content is often one of the most influential variables and is positively correlated with quality. Volatile acidity, which gives wine an unpleasant vinegar-like smell at high levels, tends to have a negative impact. Citric acid can enhance flavor, while sulphates contribute to preservation and taste sharpness. Other features like residual sugar and pH may have weaker effects.

To handle missing data during the feature engineering process, we first checked for null values using functions like i**snull().sum()**. If missing values were found, we applied imputation techniques to ensure the dataset remained usable without discarding valuable records.

Different imputation techniques have their own strengths and weaknesses. **Mean or median** imputation is simple and effective for numerical data but may distort the feature distribution. **Mode** imputation works well for categorical data but can introduce bias. More advanced methods like **K-Nearest Neighbors (KNN)** imputation provide more accuracy by considering similarity between data points, though they can be computationally intensive. Model-based imputation, using regression or machine learning, can be powerful but may overfit if not managed properly. For this dataset, due to its numerical nature and relatively clean structure, simple mean or median imputation is generally sufficient.