Project Overview:

In this project, I took the lead in building a machine learning system capable of predicting whether a person has diabetes or not. The core of our implementation revolves around the Support Vector Machine (SVM) algorithm, and all the coding is done in Python.

Understanding the Problem:

Before diving into coding, I thoroughly understood the problem statement. Our objective was to leverage medical data, including blood glucose levels, insulin levels, and more, to predict whether an individual is diabetic or non-diabetic.

Implementation with SVM:

I delved into the SVM model, a pivotal supervised learning algorithm. The algorithm works by plotting the data in a graph and finding a hyperplane that effectively separates diabetic and non-diabetic cases.

Workflow Breakdown:

- Data Preprocessing:

  - Actively involved in preprocessing the data, analyzing it, and ensuring it was suitable for training the model. Standardization of data was a key step to bring uniformity to the diverse medical attributes.

- Data Splitting:

  - Executed the splitting of data into training and testing sets. This allowed us to train the model on one set and evaluate its accuracy on another.

- Model Training:

  - Took charge of feeding the training data into the SVM model. Worked on fine-tuning the model to effectively classify diabetic and non-diabetic cases.

- Accuracy Assessment:

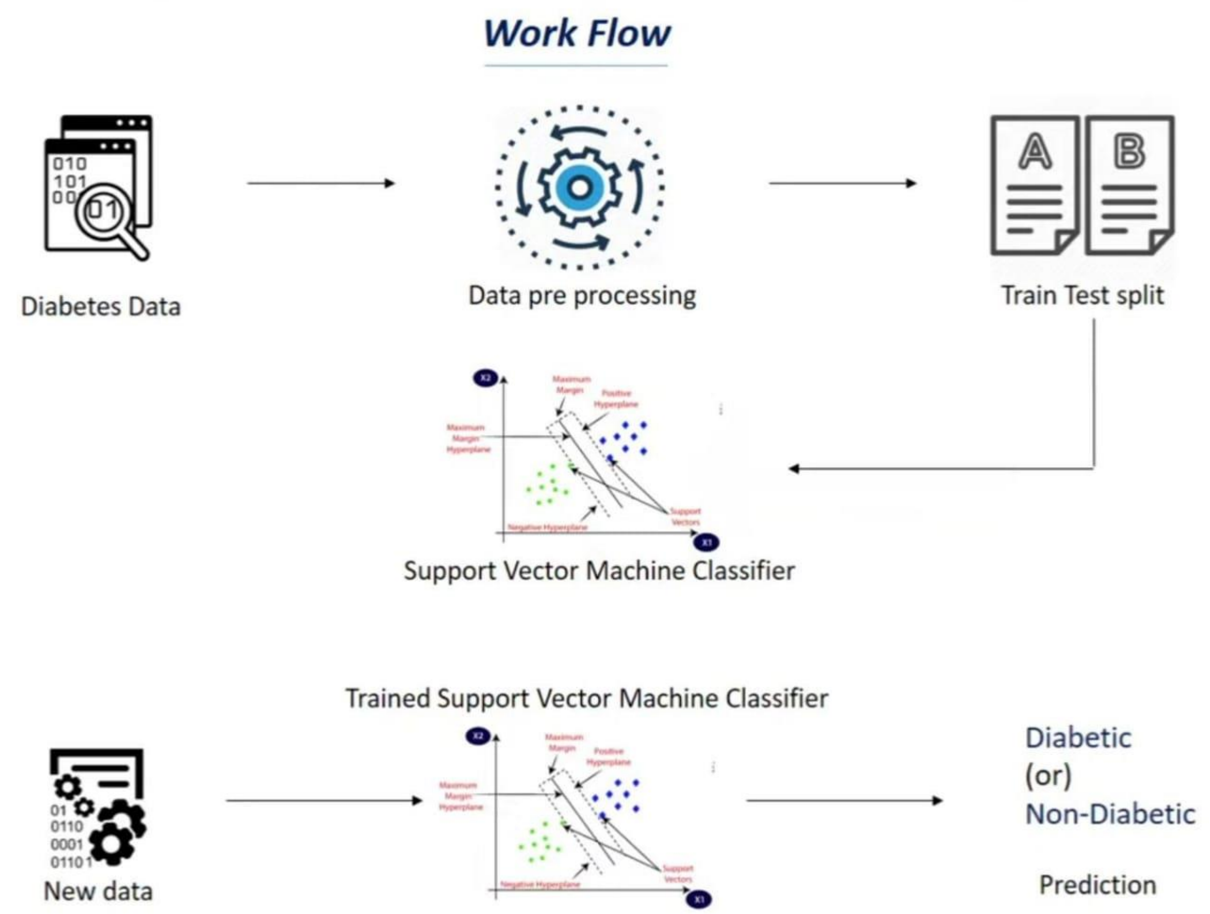
  - Actively participated in assessing the accuracy score of our trained model. This involved evaluating its performance on unseen data to ensure robust predictions.

- Prediction Process:

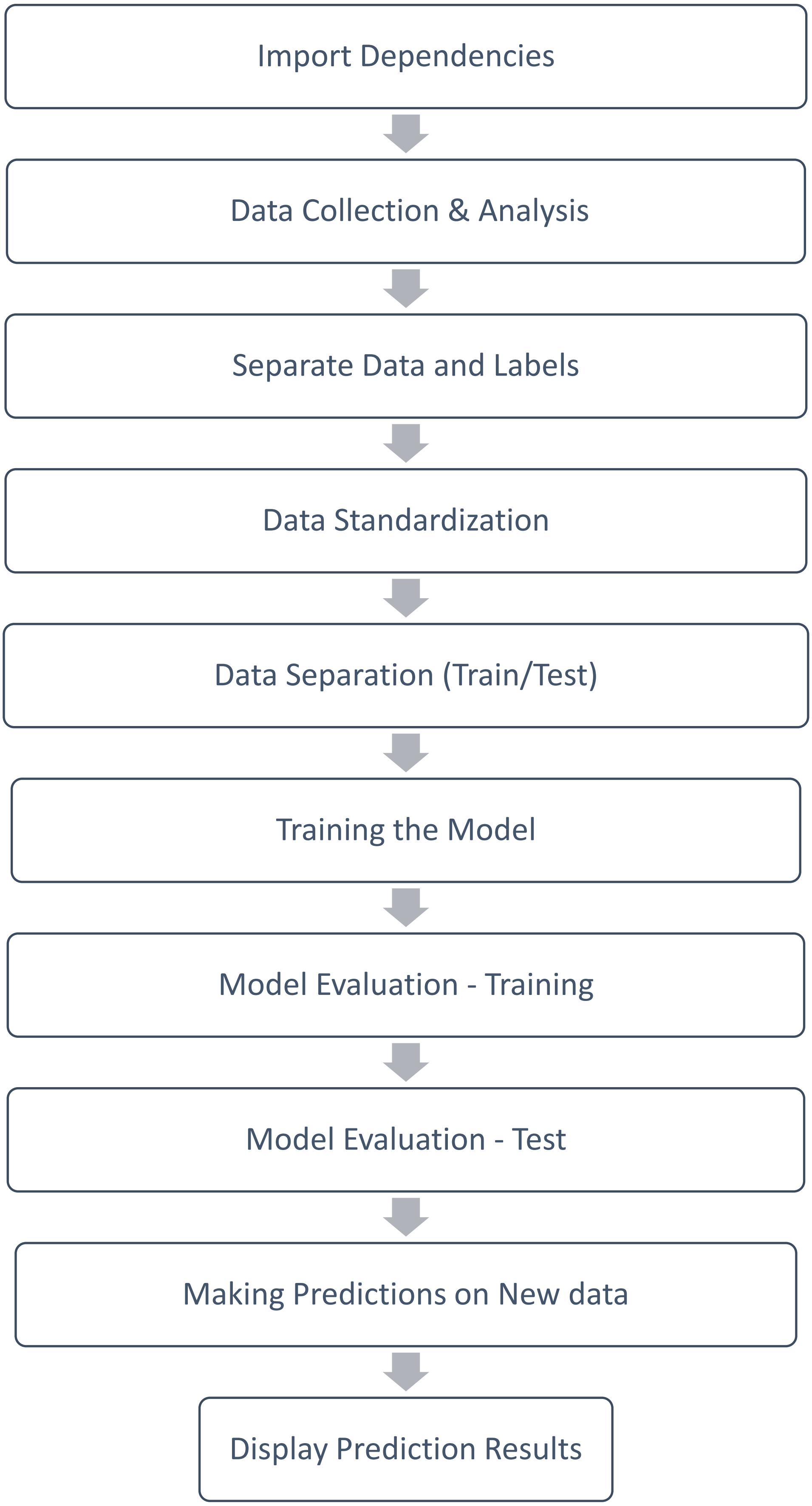
  - Demonstrated the model's capability to predict diabetes status for new patients based on their medical attributes.

Features Utilized:

Utilized various medical features, including BMI, blood glucose levels, and insulin levels, to enhance the predictive power of our model.



Coding Workflow



Importing the Dependencies

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Explanation:

- **numpy** and **pandas** are popular Python libraries for numerical and data manipulation.
- **StandardScaler** from **sklearn.preprocessing** is used for standardizing the data.
- **train_test_split** from **sklearn.model_selection** is used to split the dataset into training and testing sets.
- **svm** is the Support Vector Machine model from **sklearn**.
- **accuracy_score** from **sklearn.metrics** is used to measure the accuracy of the model.

Data Collection and Analysis

```
# loading the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv('diabetes (1).csv')

# printing the first 5 rows of the dataset
diabetes_dataset.head()

# number of rows and Columns in this dataset
diabetes_dataset.shape

# getting the statistical measures of the data
diabetes_dataset.describe()

# Counting the number of diabetic and non-diabetic cases
diabetes_dataset['Outcome'].value_counts()

# Getting the mean values for diabetic and non-diabetic cases
diabetes_dataset.groupby('Outcome').mean()
```

Explanation:

- The dataset is loaded from a CSV file named "diabetes (1) .csv".
- The first five rows of the dataset are printed for visualization.

- The number of rows and columns in the dataset are printed.
- Statistical measures of the dataset (mean, standard deviation, percentiles, etc.) are displayed.
- The count of diabetic and non-diabetic cases is printed.
- The mean values for diabetic and non-diabetic cases are calculated and displayed.

Separating the data and labels

```
# separating the data and labels
X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
Y = diabetes_dataset['Outcome']
```

Explanation:

- The dataset is split into **x** (features) and **y** (labels). **x** contains all columns except the 'outcome' column, and **y** contains only the 'outcome' column.

Printing Data and Labels

```
print(X)
print(Y)
```

Explanation:

- The separated data (**x**) and labels (**y**) are printed.

Data Standardization

```
scaler = StandardScaler()

scaler.fit(X)

standardized_data = scaler.transform(X)

X = standardized_data
Y = diabetes_dataset['Outcome']
```

Explanation:

- The purpose of data standardization is to bring all features to a common scale. This is important because different features might have different ranges, and it can affect the performance of machine learning models.

- The **StandardScaler** is used to standardize the data, making the mean of each feature 0 and the standard deviation 1.
- The data is first fitted using **scalar.fit(x)**, and then the transformation is applied using **scalar.transform(x)**. The standardized data is then stored back in the variable **x**.

Data Separation (Again) and Printing

```
# Separating data and labels again
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size =
0.2, stratify=Y, random_state=2)

# Printing the shapes of the datasets
print(X.shape, X_train.shape, X_test.shape)
```

Explanation:

- The **train_test_split** function is used to split the standardized data (**x**) and labels (**y**) into training and testing sets.
- **test_size=0.2** indicates that 20% of the data will be used for testing.
- **stratify=y** ensures that the distribution of labels in the original dataset is maintained in the train and test sets.
- **random_state=2** is used for reproducibility.
- The shapes of the resulting datasets are printed.

This section essentially re-splits the data into training and testing sets after standardization.

Training the Model

```
# Creating the SVM classifier
classifier = svm.SVC(kernel='linear')

#training the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

Explanation

- Here, a Support Vector Machine (SVM) classifier is created with a linear kernel. The **x_train** and **y_train** are the training features and labels, respectively. The **fit** method is used to train the SVM model.

Model Evaluation- Accuracy Score on Training Data

```
# accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy score of the training data : ', training_data_accuracy)
```

- Make predictions on the training data using the trained model (**classifier.predict(x_train)**).
- Calculate the accuracy score with **accuracy_score(predicted, y_train)**.

Model Evaluation - Accuracy Score on Test Data:

```
# accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score of the test data : ', test_data_accuracy)
```

- Make predictions on the test data using the trained model (**classifier.predict(x_test)**).
- Calculate the accuracy score with **accuracy_score(predicted, y_test)**.

Making Predictions on New Data:

```
input_data = (5,166,72,19,175,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

Explanation

- Define new input data.
- Reshape and standardize the input data for model compatibility.
- Use the trained model to predict whether the person is diabetic.
- Display a message based on the prediction (diabetic or not).

Results

The accuracy scores provided indicate the performance of your Support Vector Machine (SVM) model on both the training and test datasets.

1. **Accuracy Score on Training Data (0.7866):**
    - The accuracy score on the training data is approximately 78.7%.
    - This means that the SVM model correctly predicted the outcomes for about 78.7% of the instances in the training dataset.
    - It is a measure of how well the model generalizes to the data it was trained on.
2. **Accuracy Score on Test Data (0.7727):**
    - The accuracy score on the test data is approximately 77.3%.
    - This score represents the model's ability to make accurate predictions on new, unseen data (test set).
    - A test accuracy close to the training accuracy suggests that the model is not overfitting (memorizing the training data) and performs reasonably well on new data.

Interpretation:

- The accuracy scores suggest that the SVM model is performing decently on both the training and test datasets.
- It's important to note that accuracy alone might not provide a complete picture, especially in imbalanced datasets where one class significantly outnumbers the other. Precision, recall, and F1-score are other metrics you might want to consider, especially in medical applications where misclassifying certain cases (e.g., false negatives for diabetes prediction) can have serious consequences.
- May also consider further optimization of the model or exploring other evaluation metrics based on the specific requirements and constraints of the project.

Results predictions on New Data

1. **Input Data:**
    - set of input data with the following values: (5, 166, 72, 19, 175, 25.8, 0.587, 51).
    - Converted this data into a format suitable for prediction.
2. **Data Standardization:**
    - Ensured the input data is in the same scale as the data used to train the model.
3. **Prediction:**
    - Used the trained SVM model to predict if the person is diabetic.
    - The model predicted diabetic (1) for the given input.
4. **Result:**
    - Printed the standardized input data and the model's prediction.
    - Concluded that, according to the model, the person is diabetic.