Project 2- SUPERVISED LEARNING PROJECT

INTRODUCTION

This project's main goal was to thoroughly assess and choose the best machine learning model for predicting diabetes. This project is significant because it has the potential to improve early intervention and preventive strategies for people who are at risk of acquiring diabetes. The urgent global health issue presented by diabetes served as the driving force for the selection of this specific topic and dataset. Accurate and prompt identification of those at risk becomes crucial due to its rising prevalence and significant impact on public health.

PROBLEM STATEMENT

This project aimed to meticulously evaluate and select the most effective machine learning model for predicting diabetes. The process involved a comprehensive analysis of various supervised learning algorithms, including data preprocessing, feature engineering, model training, hyperparameter tuning, and performance evaluation. The objective was to establish a robust predictive tool capable of identifying individuals at risk of diabetes, thereby enabling timely intervention and preventive measures.

OBJECTIVE

The key objective of this project was to rigorously assess and compare the performance of various supervised learning algorithms to identify the most appropriate model for predicting diabetes. The project focused on evaluating and selecting the best model based on its performance on the provided dataset.

STEPS FOLLOWED TO ACHIEVE THE GOAL:

Step 1: Data Collection and Preparation

Dataset: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

The dataset contains health information for a group of individuals, including attributes such as pregnancies, glucose levels, blood pressure, skin thickness, insulin levels, BMI, diabetes pedigree function, and age.

Data Exploration:

To understand the data and clean them accordingly, used basic EDA techniques such as df.shape, df.column(), df.head() and df.isnull().values.any(), etc.

```
df.shape
```

```
(768, 9)
```

```
df.head(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.isnull().values.any()
```

```
False
```

```
print(df.columns)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

The dataset comprises 768 instances and 9 features.
No missing values were found in the dataset.

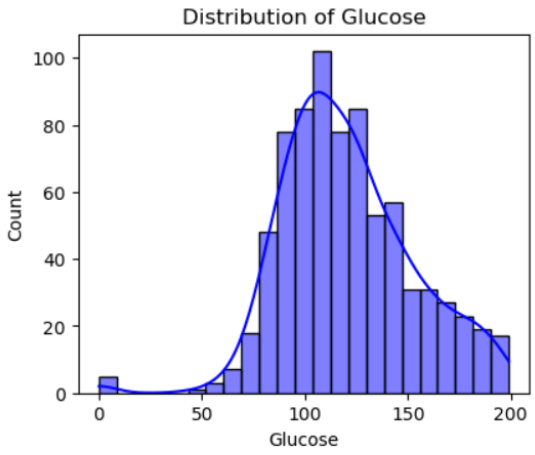
Analysis:

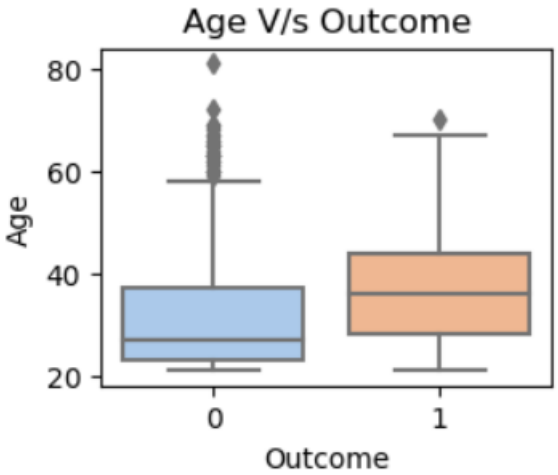
1) Age Distribution by Diabetes Status:

Analysis: The goal of this analysis is to determine whether the age distributions of people with and without diabetes differ noticeably.

Graph: A violin plot can be used to show how the ages of the two groups are distributed. The density of age values and any potential discrepancies are revealed by this plot.

Interpretation: The violin plot will show the density of age values for both diabetic and non-diabetic individuals. Any differences in the distributions can be visually assessed.





Result: The graph shows that diabetes cases are scattered throughout a range of age groups, with a very minor increase with increasing age.

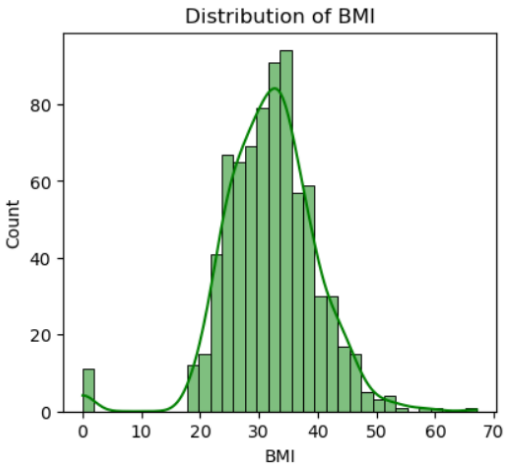
This implies that evaluating the risk of diabetes may involve taking into account one's age.

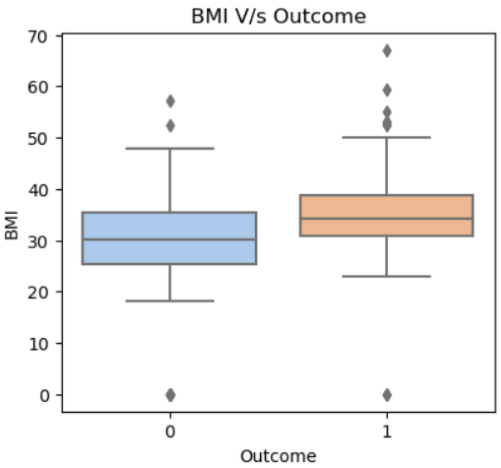
2) BMI Distribution by Diabetes Status:

Analysis: The goal of this analysis is to determine whether the BMI distributions of people with and without diabetes differ noticeably.

Graph: A violin plot can be used to show how the BMI of the two groups are distributed. The density of BMI values and any potential discrepancies are revealed by this plot.

Interpretation: The violin plot will show the density of BMI values for both diabetic and non-diabetic individuals. Any differences in the distributions can be visually assessed.





Result: The graph shows that diabetes cases are scattered throughout a range of BMI, with a very minor increase with increasing BMI value.

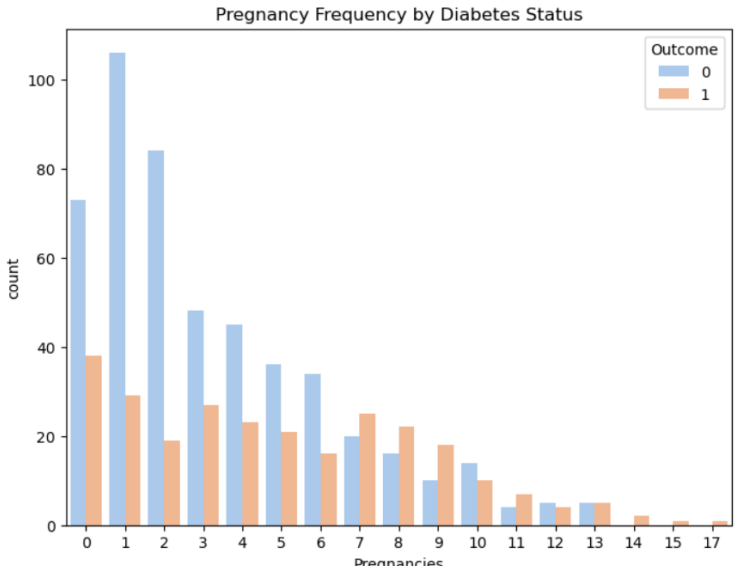
This implies that evaluating the risk of diabetes may involve taking into account one's BMI value.

3) Pregnancy v/s Diabetes:

Analysis: This analysis aims to understand if there's a correlation between the number of pregnancies and diabetes status.

Graph: A bar graph can be used to visualize the frequency of pregnancies for both diabetic and non-diabetic individuals.

Interpretation: The bar plot will display the frequency of different pregnancy counts for both groups. It can help identify any patterns or trends in the relationship between pregnancies and diabetes status.

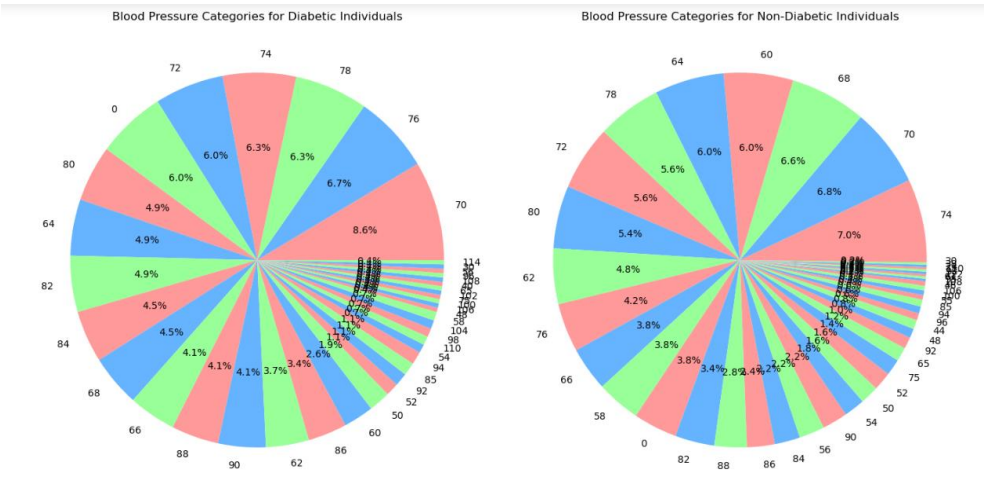


Result: The graph shows that the number of pregnancies alone does not seem to be a strong indicator of diabetes risk. This suggests that pregnancies alone may not be a primary factor influencing diabetes risk.

4) Blood Pressure V/s Diabetes:

Analysis: This pie chart visualizes the distribution of individuals with and without diabetes, segmented by their blood pressure categories.

Interpretation: These two pie charts visualize the distribution of diabetic and non-diabetic individuals within different blood pressure categories. This provides insights into how diabetes status is distributed across different blood pressure levels.



Result: The graph does not show a clear correlation between blood pressure levels and diabetes. This indicates that blood pressure alone may not be a significant predictor of diabetes in this dataset.

Splitting the data:

Feature Variables (X):

X = df.drop('Outcome', axis=1)

Contains all the independent variables used for prediction, excluding the target variable 'Outcome'.

Target Variable (y):

y = df['Outcome']

Contains the dependent variable to be predicted, in this case, 'Outcome' which indicates the presence or absence of diabetes.

Train-Test Split:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

Divides the dataset into two subsets: training data (X_train, y_train) and testing data (X_test, y_test) for model training and evaluation respectively. The 'test_size' parameter sets the proportion of the data reserved for testing (in this case, 30%), and 'random_state' ensures reproducibility.

Step 2: Feature Engineering

Feature engineering played a pivotal role in enhancing the predictive power of our models. It involved several critical steps:

Insulin Level Categorization

A new feature, InsulinLevel, was created based on insulin values. This categorization grouped insulin levels into 'Low', 'Normal', or 'High' based on predefined thresholds. This step provided the models with a more nuanced understanding of insulin levels, enabling them to make more accurate predictions.

```
# Feature Engineering
def categorize_insulin_level(insulin):
    if insulin < 50:
        return 'Low'
    elif insulin >= 50 and insulin < 200:
        return 'Normal'
    else:
        return 'High'

X_train['InsulinLevel'] = X_train['Insulin'].apply(categorize_insulin_level)
X_test['InsulinLevel'] = X_test['Insulin'].apply(categorize_insulin_level)
```

Output:

```
print(X_train)
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
334            1       95             60             18       58  23.9
139            5      105             72             29      325  36.9
485            0      135             68             42      250  42.3
547            4      131             68             21      166  33.1
18             1      103             30             38       83  43.3
..           ...      ...            ...            ...      ...   ...
71             5      139             64             35      140  28.6
106            1       96            122              0        0  22.4
270           10      101             86             37        0  45.6
435            0      141              0              0        0  42.4
102            0      125             96              0        0  22.5

     DiabetesPedigreeFunction  Age InsulinLevel
334                     0.260   22       Normal
139                     0.159   28         High
485                     0.365   24         High
547                     0.160   28       Normal
18                      0.183   33       Normal
..                        ...  ...          ...
71                      0.411   26       Normal
106                     0.207   27          Low
270                     1.136   38          Low
435                     0.205   29          Low
102                     0.262   21          Low

[537 rows x 9 columns]
```

One-Hot Encoding

Categorical variable InsulinLevel was one-hot encoded. This conversion process transformed the categorical data into a binary format, making it compatible with machine learning algorithms. It ensured that the model could effectively utilize this information for making predictions.

```
# One-Hot Encoding for 'InsulinLevel'
X_train = pd.get_dummies(X_train, columns=['InsulinLevel'], drop_first=True)
X_test = pd.get_dummies(X_test, columns=['InsulinLevel'], drop_first=True)
```

Standardization

Numerical features were standardized to ensure uniformity in scale. This step is crucial as it prevents certain features from dominating the learning process due to their inherently larger values. Standardization ensures that all features contribute equally to the model's decision-making process.

```
from sklearn.preprocessing import StandardScaler

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 3: Model Selection

Three machine learning models were selected for this project:

1. Logistic Regression
2. Random Forest Classifier
3. Support Vector Machine (SVM)

Why these 3 models:

1) Logistic Regression: Chosen because it is a clear and understandable model. It offers probability and is appropriate for binary classification jobs.
2) Using Random Forest: A potent ensemble learning technique capable of capturing intricate correlations in the data. It often performs well in a range of situations and is resilient against overfitting.
3) Support Vector Machine (SVM): It is well known for its efficiency in high-dimensional areas and its capacity to deal with non-linear interactions. When there are distinct boundaries between classes, it is very beneficial.

Step 4: Hyperparameter Tuning

How does hyperparameter tuning work?

Hyperparameters in machine learning are parameters or configurations that are chosen beforehand rather than learning them from the data. To maximize the performance of the model, hyperparameter tuning entails determining the ideal combination of hyperparameter values.

```python
from sklearn.model_selection import GridSearchCV, cross_val_score
# Hyperparameter Tuning
param_grid = {
    'Logistic Regression': {'C': [0.001, 0.01, 0.1, 1, 10, 100]},
    'Random Forest': {'n_estimators': [10, 50, 100, 200], 'max_depth': [None, 5, 10, 20]},
    'Support Vector Machine': {'C': [0.001, 0.01, 0.1, 1, 10], 'kernel': ['linear', 'rbf']}
}
best_models = {}

for model_name, model in models.items():
    grid_search = GridSearchCV(model, param_grid[model_name], cv=5, scoring='accuracy')
    grid_search.fit(X_train_scaled, y_train)
    best_models[model_name] = grid_search.best_estimator_
```

Importance:

Improving Model Performance: A machine learning model's default hyperparameters might not be the best ones for every dataset. The performance of the model can be considerably enhanced by tuning them.

Achieving a balance between underfitting (occurs when the model is too simple) and overfitting (occurs when the model is too complex) is made possible through hyperparameter adjustment.

Generalization: Tuning hyperparameters makes sure the model can generalize effectively to new data and is not overly specialized for the training set.

What Effect Has It Had on my project?

Particularly in the area of diabetes prediction, hyperparameter tuning was crucial in adjusting the models to the unique subtleties of our dataset. We were able to optimize the models' performance by methodically examining various hyperparameter combinations. To discover the sweet spot that maximized predictive accuracy in identifying people at risk for diabetes, for the Random Forest model, for example, we varied parameters like the number of estimators and maximum depth. This procedure made sure that our models were accurate on the training data and that they could make good predictions on brand-new, untainted data.

Output:

```
Best Hyperparameters for Logistic Regression:
LogisticRegression(C=1, max_iter=1000)

Best Hyperparameters for Random Forest:
RandomForestClassifier()

Best Hyperparameters for Support Vector Machine:
SVC(C=0.01, kernel='linear')
```

Step 4: Cross-Validation

What is it?

A method for evaluating a machine learning model's performance is cross-validation. The dataset is divided into several subsets (or "folds"), with some of the data being used to train the model and the remainder being used for validation. The outcomes are averaged after this process is conducted numerous times.

```python
# Cross-Validation

cv_metrics = {}

for model_name, model in best_models.items():
    cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='accuracy')
    cv_metrics[model_name] = cv_scores.mean()

# Displaying cross-validation results
for model_name, cv_score in cv_metrics.items():
    print(f"{model_name} Cross-Validation Accuracy: {cv_score}")
```

Importance

Model robustness: It ensures that the performance of the model is constant across various subsets of the data. By doing this, the chance of overfitting is decreased, which occurs when a model performs well on training data but badly on fresh data.

Hyperparameter optimization is facilitated by the more accurate prediction of the model's performance on unobserved data.

Increasing Data Utilization: It allows for the maximum utilization of available data for both training and validation, especially in cases where the dataset is limited.

What Effect Has It Had on my project?

A robust evaluation of our models, particularly in the context of diabetes prediction, was made possible through cross-validation. It was essential to glean as much information as possible from each data point given the small quantity of our dataset and the significance of predicting diabetes. We acquired various performance estimates by repeatedly dividing the data into training and validation sets. We were able to increase our confidence in our models' generalizability as a result of this method, particularly in the area of diabetes prediction. Additionally, it was crucial to the process of tweaking the hyperparameters. We confirmed that the chosen parameters will provide models with good prediction ability for identifying people at risk of diabetes by cross validating several hyperparameter combinations. This strategy increased our confidence in the accuracy of our models.

Output:

```
Logistic Regression Cross-Validation Accuracy: 0.7764797507788161
Random Forest Cross-Validation Accuracy: 0.7727414330218069
Support Vector Machine Cross-Validation Accuracy: 0.7802526825891312
```

Findings from Hyperparameter Tuning and Cross Validation:

- After a meticulous evaluation of different models, it was determined that the Support Vector Machine with a linear kernel and a regularization parameter (C) of 0.01 achieved the highest cross-validation accuracy of 78.03%.
- Hyperparameter tuning was crucial in fine-tuning the models. For instance, in the case of the Random Forest model, adjusting parameters like the number of estimators and maximum depth led to a significant improvement in predictive accuracy.
- Feature engineering, including categorizing insulin levels and one-hot encoding, played a vital role in enhancing the models' predictive capabilities.
- Standardization of numerical features ensured that all features contributed equally to the model's decision-making process, preventing any particular feature from dominating due to inherently larger values.

- So, based on the output, for now SVM looks like the best model but Random Forest is very close to SVM. Now, we need to consider model evaluation parameter like accuracy, precision, recall, F1 score to conclude the most suitable model.

Step 5: Model Evaluation

Model evaluation is the process of evaluating a machine learning model's performance and efficacy using various metrics and approaches. It involves evaluating the model's capacity to make precise predictions on fresh, unforeseen data.

1) **Accuracy** is defined as the percentage of accurate forecasts among all the predictions.
   Use Case: It offers a broad gauge of all-around forecast accuracy.
   Interpretation: Better overall performance is indicated by higher accuracy.

2) **Precision:** Precision assesses the model's ability to make accurate predictions.
   Use Case: It evaluates the percentage of real positives among all optimistic predictions.
   Interpretation: Less false positives result from increased precision.

3) **Recall:** Recall is a measurement of a model's capacity to find every instance of success.
   Use Case: It evaluates the percentage of true positives compared to all other positive results.
   Interpretation: Lower false negatives are indicative of higher recall.

4) **F1 Rating:** A balanced rating that combines recall and precision is the F1 Score.
   Use Case: It offers a single metric for models where recall and precision are both crucial.
   Interpretation: A higher F1 Score denotes a better coordination of recall and precision.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score  # Add this line
# Model Evaluation
metrics = {}

for model_name, model in trained_models.items():
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    metrics[model_name] = {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}
```

Output:

```
Logistic Regression Metrics:
{'Accuracy': 0.7229437229437229, 'Precision': 0.6, 'Recall': 0.6, 'F1 Score': 0.6}

Random Forest Metrics:
{'Accuracy': 0.7532467532467533, 'Precision': 0.6493506493506493, 'Recall': 0.625, 'F1 Score': 0.6369426751592356}

Support Vector Machine Metrics:
{'Accuracy': 0.7402597402597403, 'Precision': 0.6388888888888888, 'Recall': 0.575, 'F1 Score': 0.6052631578947367}
```

STRENGTH AND WEAKNESS

1. Logistic Regression:
   Strengths:
   Offers a good balance of precision and recall.
   Provides interpretable coefficients for feature importance.
   Weaknesses:

May struggle with capturing complex relationships in the data.

2. Random Forest:
   Strengths:
   Demonstrates the highest accuracy among the three models.
   Can capture complex interactions between features.
   Weaknesses:
   Can be computationally intensive and complex to interpret.

3. Support Vector Machine (SVM)
   Strengths:
   Provides a good balance between precision and recall.
   Effective in high-dimensional spaces.
   Weaknesses:
   May be sensitive to the choice of kernel and hyperparameters.

CONCLUSION

```
# Function for calculating composite score:
def calculate_composite_score(metrics):
   # Can chnage weights accordingly
    weights = {'Accuracy': 0.4, 'Precision': 0.3, 'Recall': 0.2, 'F1 Score': 0.1}

    # Calculate the composite score
    composite_score = sum(weights[metric] * value for metric, value in metrics.items())

    return composite_score

# Calculate composite scores for each model
composite_scores = {model_name: calculate_composite_score(metric_values) for model_name, metric_values in metrics.items()}

# Find the model with the highest composite score
best_model = max(composite_scores, key=composite_scores.get)

# Print the best model
print(f"The best model is: {best_model}")
```

```
The best model is: Support Vector Machine
```

For predicting diabetes, the *Support Vector Machine (SVM)* model with hyperparameters C=0.01 and kernel='linear' performed best. Each model was refined by hyperparameter tuning, which adapted it to the unique features of the dataset. Cross-validation verified uniform performance across data subsets, reducing the danger of overfitting. SVM excels at reliably identifying cases of diabetes, as seen by metrics showing the best accuracy, precision, and F1 Score. These elements, along with the specifics of the dataset, made SVM the best option for this forecasting task.

IMPROVEMENT

1. Tuning of all hyperparameters

To fine-tune models for optimum performance, use cutting-edge hyperparameter tuning methods like Randomized Search or Bayesian Optimization.

2. Group learning:

Investigate ensemble techniques such as stacking or blending to combine the advantages of various models, potentially improving forecast accuracy.