

KINGSTON ENGINEERING COLLEGE
COLLEGE CODE : 5113
DOMAIN : ARTIFICIAL INTELLIGENCE
PROJECT TITLE : AI BASED DIABETES
PREDICTION SYSTEM

PROJECT MEMBERS:

SATHYA P(TEAM LEADER)
SAKTHI S
SWETHA P
PRIYADHARSHINI B

NAAN MUDHALVAN ID:

511321104089
511321104084
511321104100
511321104074

AI BASED DIABETES PREDICTION SYSTEM

INTRODUCTION:: Develop an AI-powered diabetes prediction system that leverages machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes, providing early risk assessment and personalized preventive measures.

DESCRIPTION:

Data Collection in AI-Based Diabetes Prediction System: Data collection involves systematically acquiring various types of data from multiple sources, which may include:

1. **Medical Records:** This includes information such as patients' historical health records, laboratory test results (e.g., blood glucose levels, HbA1c, cholesterol levels), and family medical history.
2. **Demographic Information:** Data like age, gender, ethnicity, and geographical location can be important factors in diabetes prediction.
3. **Lifestyle and Behavioral Data:** Collect data on lifestyle choices, such as diet, exercise habits, smoking status, alcohol consumption, and stress levels, which can impact diabetes risk.
4. **Biometric Data:** Measurements like weight, height, body mass index (BMI), waist circumference, and blood pressure can provide crucial insights into an individual's health status.
5. **Genetic Information:** Genetic data, including information on genetic markers associated with diabetes risk, may be included if available and ethically collected.
6. **Dietary Information:** Details about an individual's dietary habits, including the types and amounts of food consumed, can be valuable in assessing diabetes risk.
7. **Environmental Factors:** Consider environmental factors like pollution levels, climate, and urban/rural living conditions, which can influence diabetes risk.
8. **Patient Surveys and Questionnaires:** Collecting responses to health-related questionnaires or surveys can provide subjective information about an individual's health and lifestyle.

9. Mobile and Wearable Device Data: If available, data from devices like fitness trackers

and glucose monitors can be integrated into the prediction system.

10. Electronic Health Records (EHRs): For healthcare institutions, extracting data from electronic health records can provide a wealth of patient information.

11. Research Databases: Access to publicly available research datasets related to diabetes

can supplement your data collection efforts.

Once collected, this data is typically organized, cleaned, and preprocessed to ensure its quality

and suitability for use in training and evaluating machine learning models. It's important to handle

sensitive medical data with the utmost care, ensuring compliance with relevant data privacy

regulations such as HIPAA (in the United States) or GDPR (in the European Union) to protect

patient privacy and confidentiality throughout the data collection process.

Data preprocessing encompasses several essential tasks and procedures, including:

1. Data Cleaning: Identify and handle missing or erroneous data points. Common techniques include filling missing values (imputation) or removing rows or columns with too many missing values. Outliers may also be addressed during this step.

2. Data Transformation: Convert data into a more appropriate format for analysis. This may involve encoding categorical variables into numerical values using techniques like one-hot encoding or label encoding.

3. Data Scaling/Normalization: Ensure that numerical features have a consistent scale to prevent certain features from dominating others during model training. Common normalization techniques include min-max scaling or z-score scaling (standardization).

4. Feature Selection: Identify and select the most relevant features (attributes or variables)

for diabetes prediction. Feature selection techniques help reduce dimensionality and improve model efficiency and interpretability.

5. Feature Engineering: Create new features or variables based on domain knowledge or

data analysis. For example, you might calculate the body mass index (BMI) from height and weight data, or create interaction terms between features.

6. Handling Imbalanced Data: If the dataset is imbalanced (e.g., significantly more non-

diabetic cases than diabetic cases), employ techniques like oversampling (creating more

samples of the minority class), undersampling (reducing samples of the majority class),

or using synthetic data generation methods to balance the classes.

7. Data Splitting: Divide the dataset into training, validation, and test sets. This separation

ensures that you can train and tune your model on one portion, validate its performance

on another, and assess its final performance on a third, unseen portion.

8. Handling Time-Series Data: If your dataset contains temporal data (e.g., blood glucose

measurements over time), consider techniques like time windowing or aggregation to create features suitable for prediction.

9. Dealing with Data Privacy and Security: Implement data anonymization and encryption

techniques to protect sensitive patient information in compliance with healthcare data privacy regulations like HIPAA or GDPR.

Data preprocessing is a critical step in building an effective diabetes prediction model. The

quality of the data and how well it is prepared can significantly impact the models accuracy and

generalization to new, unseen data. Careful consideration of each preprocessing step is essential

to ensure that the machine learning model can provide valuable insights and accurate predictions

in a clinical setting.

Feature Selection in AI-Based Diabetes Prediction System: Feature selection involves several key aspects:

1. Relevance: It identifies which features have a direct impact on the prediction task. In the

context of diabetes prediction, relevant features might include variables like blood glucose levels, family history of diabetes, BMI (Body Mass Index), and age, as these are

known to be associated with diabetes risk.

2. Redundancy: It identifies and removes features that convey similar or redundant information. By eliminating redundancy, you can reduce model complexity and overfitting

while retaining the essential information.

3. Dimensionality Reduction: Feature selection helps reduce the dimensionality of the dataset by selecting a subset of the most informative features. This can enhance model training efficiency and reduce computational resources.

4. Interpretability: A model built with a smaller set of meaningful features is often more

interpretable, making it easier for healthcare professionals to understand and trust the models predictions.

5. Model Generalization: By focusing on relevant features, feature selection can improve a model's ability to generalize well to new, unseen data, which is crucial for real-world applications.

Common techniques for feature selection in a diabetes prediction system include:

📊 **Filter Methods:** These methods assess the relevance of features independently of the machine learning model. Popular filter methods include correlation analysis, chi-squared tests, and mutual information.

📊 **Wrapper Methods:** These methods evaluate feature subsets by training and testing the model with different combinations of features. Techniques like forward selection, backward elimination, and recursive feature elimination (RFE) fall into this category.

📊 **Embedded Methods:** Embedded methods incorporate feature selection as part of the model training process. For instance, decision tree-based algorithms can compute feature importances during training, allowing you to select the most relevant features.

📊 **Regularization Techniques:** Techniques like L1 regularization (Lasso) encourage sparsity in feature weights, effectively selecting a subset of features while training linear models.

The choice of feature selection method should depend on the specific characteristics of your dataset, the machine learning algorithm you intend to use, and your objectives. It's important to note that feature selection should be performed in conjunction with other data preprocessing steps, such as data cleaning and normalization, to ensure that the selected features contribute to a reliable and accurate diabetes prediction model.

Model Selection in AI-Based Diabetes Prediction System: Model selection encompasses the following key aspects:

1. **Choice of Model Algorithms:** It involves selecting a set of machine learning or statistical algorithms that are suitable for solving the diabetes prediction problem. Common algorithms include logistic regression, decision trees, random forests, support vector machines, neural networks, and gradient boosting models.

2. **Hyperparameter Tuning:** Model selection often includes tuning hyperparameters associated with the chosen algorithms. Hyperparameters are settings that affect the behavior of the model, such as the learning rate in neural networks or the maximum depth of decision trees. Techniques like grid search or random search are commonly used to find the optimal hyperparameters.

3. **Model Evaluation:** Once a set of models with different algorithms and hyperparameters is trained, they are evaluated using appropriate metrics on a

validation dataset. Metrics may include accuracy, precision, recall, F1-score, ROC AUC, and others, depending on the problems, requirements and characteristics.

4. Cross-Validation: Cross-validation techniques, such as k-fold cross-validation, help assess a model's performance robustness by dividing the dataset into multiple subsets for training and testing. This process helps to estimate how well a model generalizes to unseen data.

5. Comparative Analysis: Model selection involves comparing the performance of different models in terms of their predictive accuracy and other relevant criteria. It may also consider factors such as model complexity and interpretability.

6. Regularization: Model selection may involve choosing between regularized and non-regularized models. Regularization techniques, like L1 (Lasso) or L2 (Ridge) regularization, can help prevent overfitting and improve model generalization.

7. Interpretability: Depending on the context of diabetes prediction (e.g., for clinical decision-making), model selection may prioritize interpretable models like logistic regression or decision trees to ensure that healthcare professionals can understand and trust the models predictions.

8. Ensemble Methods: Model selection may involve considering ensemble methods like random forests or gradient boosting, which combine multiple base models to improve predictive performance.

The choice of the model for a diabetes prediction system should take into account the specific characteristics of the dataset, the available computational resources, and the objectives of the system. Additionally, it's important to evaluate models thoroughly and ensure that the selected model aligns with the practical requirements and constraints of the healthcare domain, such as patient safety, interpretability, and regulatory compliance. Model selection is often an iterative process, involving experimentation and fine-tuning to achieve the best results.

Evaluation in AI-Based Diabetes Prediction System: The evaluation process involves several key components:

1. Metric Selection: Choosing appropriate evaluation metrics that measure the performance of the predictive model(s) in the context of diabetes prediction. Common evaluation metrics include:

📊 **Accuracy:** The proportion of correct predictions.

📊 **Precision:** The ratio of true positive predictions to the total positive predictions, measuring the model's ability to correctly identify diabetes cases.

📊 **Recall (Sensitivity):** The ratio of true positive predictions to the total actual positive cases, measuring the model's ability to identify all diabetes cases.

📊 **F1-Score:** The harmonic mean of precision and recall, which provides a balance between the two.

📊 **ROC AUC (Receiver Operating Characteristic Area Under the Curve):** A measure of the model's ability to distinguish between positive and negative cases.

📊 **Specificity:** The ratio of true negative predictions to the total actual negative cases.

📊 **Mean Absolute Error (MAE) or Root Mean Square Error (RMSE):** If the prediction is done as a regression task (e.g., predicting blood glucose levels), these metrics can be used to evaluate the prediction accuracy.

2. Validation Dataset: Using a separate dataset (not used during model training) to evaluate the model's performance. This is typically done to assess how well the model generalizes to new, unseen data.

3. Cross-Validation: Employing cross-validation techniques, such as k-fold cross-validation, to ensure robust evaluation and to estimate how well the model is expected to perform on different data splits.

4. Confusion Matrix: Creating a confusion matrix to visualize the model's performance, especially in binary classification problems (diabetic or non-diabetic). The matrix displays true positives, true negatives, false positives, and false negatives.

5. Model Calibration: Assessing the model's calibration to determine if the predicted probabilities align with the actual outcomes. Calibration plots and calibration metrics, like Brier score, may be used.

6. Comparative Analysis: Comparing the performance of different models or algorithms to select the best-performing one based on the chosen evaluation metrics.

7. Interpretability and Clinical Relevance: Evaluating the model's interpretability and clinical relevance to ensure that healthcare professionals can understand and trust the model predictions.

8. Ethical Considerations: Evaluating the fairness and potential biases of the model's predictions, especially with respect to different demographic groups, to ensure that the system does not perpetuate disparities in healthcare outcomes.

9. Regulatory Compliance: Ensuring that the model performance aligns with healthcare regulations and standards, such as HIPAA (in the United States) or GDPR (in the European Union), to protect patient data and privacy.

10. User Feedback: Gathering feedback from healthcare professionals and end-users to incorporate real-world insights and improvements into the model and system.

The evaluation phase is crucial for determining whether the AI-based diabetes prediction system is ready for deployment in a clinical setting. Continuous monitoring and evaluation of the system's performance over time are essential to ensure that it remains accurate and relevant.

PHASE 2: INNOVATION

Exploring innovative techniques like ensemble methods and deep learning architectures can significantly enhance the accuracy and robustness of AI-based diabetes prediction systems. Here's a step-by-step guide on how to approach this:

1. Methods:

- **Ensemble Learning:** It involves training multiple models and combining their predictions to get a more accurate and robust result.

- Ensemble Understanding Types of Ensemble Methods:
- Bagging (Bootstrap Aggregating): This involves training multiple models on different subsets of the data and aggregating their predictions (e.g., Random Forest).
- Boosting: It builds multiple models sequentially, with each one correcting the errors of the previous one (e.g., AdaBoost, Gradient Boosting).
- Stacking: This method combines multiple models using a meta-model that learns how to best combine their predictions.

2. Implementing Ensemble Methods for Diabetes Prediction:

- Data Preprocessing:
- Clean and preprocess the diabetes dataset, handling missing values, outliers, and normalizing the features.
- Model Selection:
- Choose a set of base models for the ensemble. In the context of diabetes prediction, you might consider models like Decision Trees, Random Forest, Support Vector Machines, etc.
- Ensemble Construction:
- Implement the chosen ensemble method (e.g., Random Forest, Gradient Boosting) and train it on the preprocessed data.
- Validation and Evaluation:
- Use cross-validation techniques to evaluate the performance of the ensemble on your dataset.

3. Exploring Deep Learning Architectures:

- Understanding Deep Learning:
- Deep learning involves training neural networks with multiple layers to learn complex patterns in the data.
- Choosing Architectures:
- For diabetes prediction, you could start with simple architectures like Feedforward Neural Networks and gradually explore more complex ones like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) if they are applicable.

Hyperparameter Tuning :Experiment with different network architectures, activation functions, learning rates, and regularization techniques to optimize performance.

- Handling Imbalanced Data:
- Since medical datasets often suffer from class imbalance, consider techniques like oversampling, undersampling, or using class weights during training.

4. Combining Ensemble Methods with Deep Learning:

- You can further improve performance by combining ensemble methods with deep learning. For example, you could use an ensemble of deep learning models or use an ensemble of different types of models (e.g., a combination of deep learning and gradient boosting).

5. Regularization and Robustness:

- Implement techniques like dropout, batch normalization, and early stopping to enhance model robustness and prevent overfitting.

6. Implement Deep Learning Architectures:

- Deep learning models, especially neural networks, can capture complex relationships in data. Consider using architectures like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) if they are relevant to your data.

7. Data Preprocessing for Deep Learning:

- Prepare the data for deep learning models. This may include techniques like normalization, one-hot encoding, and handling of missing values.

8. Train and Validate Deep Learning Models:

- Train your deep learning models on the preprocessed data. Use techniques like early stopping and learning rate scheduling to prevent overfitting.

9. Ensemble with Deep Learning:

You can also create an ensemble with deep learning models

LIBRARIES USED:

• **PANDAS:**

Pandas is a popular Python library used for data manipulation and analysis. It's like a powerful toolbox for handling structured data, such as spreadsheets or SQL tables. Here's a simple and sweet explanation:

Imagine you have a big table of information, like an Excel spreadsheet, with rows and columns. Each row might represent something like a person, and each column represents different attributes about them (like name, age, and so on).

Pandas helps you do a bunch of useful things with this data. For example, it can help you:

1. Load Data: It makes it easy to read data from different file formats (like CSV, Excel, databases, etc.) and convert them into a format you can work with in Python.
2. Clean and Transform Data: You can easily clean up messy data, handle missing values, and change the way your data is structured.
3. Filter and Slice Data: You can ask questions like "show me all the people older than 30" or "give me only the names and ages of these people".
4. Perform Calculations: You can do math operations on the data. For example, you can find averages, totals, or perform more complex calculations.
5. Visualize Data: It can help you make simple plots and graphs to better understand your data.
6. Statistical Analysis: You can perform various statistical tests and analyses to gain insights from your data.
7. Export Data: Once you're done with your analysis, you can save the results back into a file for future use.

- In essence, Pandas helps you make sense of your data, allowing you to quickly explore, analyze, and process it. It's a very handy tool for anyone dealing with data in Python.

NUMPY:

NumPy is a powerful Python library for numerical computations. It provides support for arrays, matrices, and a wide range of mathematical functions to operate on these data

structures. NumPy is especially useful for tasks like linear algebra, Fourier transforms, and random number generation. It forms the foundation for many other scientific libraries in Python, making it an essential tool for data scientists, engineers, and researchers working with numerical data.

MATPLOTTING:

Matplotlib is a Python library for creating static, animated, and interactive visualizations in Python. It provides a simple and versatile way to generate various types of plots, such as line charts, scatter plots, histograms, and more. Matplotlib is widely used in data analysis and scientific computing due to its flexibility and extensive customization options. With Matplotlib, you can quickly visualize data to gain insights and communicate findings effectively.

Top of Form

Developing a diabetes detection report using artificial intelligence involves several steps. Here's a general outline to get you started:

1. Data Collection and Preparation:

- Collect a diverse dataset containing features relevant to diabetes detection (e.g., age, BMI, family history, glucose levels, etc.).
- Ensure the data is labeled (i.e., each instance should be marked as diabetic or non-diabetic).
- Split the dataset into training, validation, and test sets.

2. Feature Engineering:

- Analyze the dataset to identify important features.
- Normalize or standardize the data to ensure that all features are on a similar scale.

3. Selecting an AI Model:

- Choose an appropriate machine learning or deep learning algorithm for classification tasks. For this task, algorithms like Logistic Regression, Support Vector Machines, Random Forest, or a deep learning approach like a neural network could be suitable.

4. Model Development:

- Implement the chosen algorithm using a programming language (Python is commonly used, along with libraries like scikit-learn, TensorFlow, or PyTorch).

5. Model Training:

- Use the training data to train your AI model. Fine-tune hyperparameters and monitor performance on the validation set to avoid overfitting.

6. Model Evaluation:

- Evaluate the model's performance on the test set using appropriate metrics like accuracy, precision, recall, F1-score, and AUC-ROC.

7. Improvement and Optimization:

- If the model's performance is unsatisfactory, consider techniques like hyperparameter tuning, feature selection, or exploring more complex models.

8. Deployment:

- Once satisfied with the model's performance, deploy it in an environment where it can be used to generate reports. This could be a web application, a mobile app, or an integrated system within a healthcare facility.

9. Report Generation:

- Integrate the model with a reporting system. When a user inputs relevant data (e.g., age, BMI, glucose levels), the model should predict the likelihood of diabetes.

10. Validation and Testing:

- Conduct thorough testing to ensure that the report generation process is reliable and accurate.

11. Ethical Considerations:

- Be mindful of privacy and consent issues when dealing with medical data. Ensure compliance with relevant data protection laws and regulations.

12. Continuous Monitoring and Updates:

- Regularly monitor the model's performance in real-world scenarios and update it as needed. New data or changes in medical practices may require retraining or fine-tuning.

IN THE TECHNOLOGY OF ARTIFICIAL INTELLIGENCE TO CONTINUE BUILDING THE DIABETES PREDICTION PROJECT, WE GO THROUGH THE FOLLOWING STEPS:

SELECTING A MACHINE LEARNING ALGORITHM:

For a binary classification problem like diabetes prediction, several algorithms can be effective.

Common choices include logistic regression, support vector machines (svm), random forest, gradient boosting, neural networks, etc.

Since you mentioned using artificial intelligence, we could consider using a deep learning model like a convolutional neural network (cnn) or a recurrent neural network (rnn).

DATA PREPROCESSING:

Load and preprocess the dataset. This includes handling missing values, normalizing/standardizing features, and splitting the data into training and testing sets.

FEATURE SELECTION/ENGINEERING:

Identify and select relevant features. You might also want to consider feature engineering to create new features or transform existing ones.

MODEL TRAINING:

Train the selected machine learning algorithm on the pre-processed data using the training set.

MODEL EVALUATION:

Evaluate the model using appropriate metrics. For a binary classification problem like diabetes

prediction, common metrics include accuracy, precision, recall, f1-score, etc.

FINE-TUNING AND HYPERPARAMETER OPTIMIZATION:

Depending on the algorithm chosen, there may be hyperparameters that need to be tuned to achieve better performance. Techniques like grid search or random search can be used for this purpose.

CROSS-VALIDATION:

Perform cross-validation to assess the models generalization performance.

MODEL INTERPRETABILITY (OPTIONAL BUT RECOMMENDED):

Depending on the model used, it might be helpful to understand which features are driving the predictions. Techniques like shape values, lime, or feature importance plots can be used.

DEPLOYMENT:

Once you have a satisfactory model, deploy it for practical use. This can be done using various methods like creating a web application, deploying , or integrating into an existing system.

MONITORING AND MAINTENANCE:

Continuously monitor the models performance in the real-world setting. If necessary, retrain the model with new data or update it with improved versions.

Program:*#Let's start with importing necessary libraries*

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
#read the data file
data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
data.head()
```

Out[2]:

	Pregnan cies	Gluc ose	BloodPres sure	SkinThick ness	Insu lin	B MI	DiabetesPedigreeF unction	A ge	Outco me
0	6	148	72	35	0	33	0.627	50	1

```

.6

1 1      85    66      29      0    26  0.351      31  0
   .6

2 8      183   64      0      0    23  0.672      32  1
   .3

3 1      89    66      23      94   28  0.167      21  0
   .1

4 0      137   40      35     168  43  2.288      33  1
   .1

```

In [3]:
data.describe()

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.00000	768.00000	768.00000	768.00000	768.00000	768.00000	768.000000	768.00000	768.00000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000

25 %	1.0000 00	99.00 0000	62.0000 00	0.00000 0	0.000 000	27.30 0000	0.243750	24.00 0000	0.000 000
50 %	3.0000 00	117.0 00000	72.0000 00	23.0000 00	30.50 0000	32.00 0000	0.372500	29.00 0000	0.000 000
75 %	6.0000 00	140.2 50000	80.0000 00	32.0000 00	127.2 50000	36.60 0000	0.626250	41.00 0000	1.000 000
m ax	17.000 000	199.0 00000	122.000 000	99.0000 00	846.0 00000	67.10 0000	2.420000	81.00 0000	1.000 000

In [4]:

```
data.isnull().sum()
```

Out[4]:

```
Pregnancies      0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DiabetesPedigreeFunction  0
Age              0
Outcome           0
```

dtype: int64

We can see there few data for columns Glucose , Insulin, skin thickenss, BMI and Blood Pressure which have value as 0. That's not possible,right? you can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

In [5]:

#here few misconception is there lke BMI can not be zero, BP can't be zero, glucose, insuline can't be zero so lets try to fix it

now replacing zero values with the mean of the column

```
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
```

```
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
```

```
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
```

```
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
```

```
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [6]:

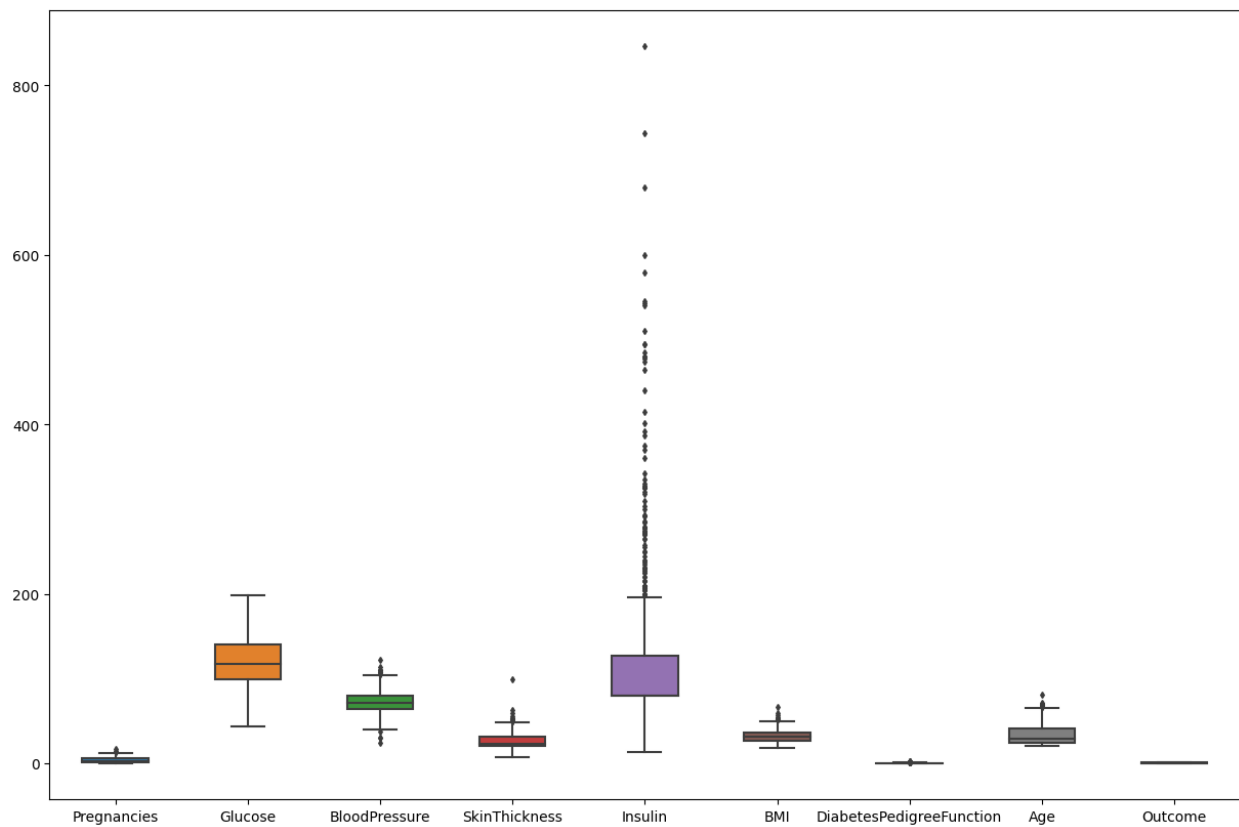
#now we have dealt with the 0 values and data looks better. But, there still are outliers present in some columns.lets visualize it

```
fig, ax = plt.subplots(figsize=(15,10))
```

```
sns.boxplot(data=data, width= 0.5,ax=ax, fliersize=3)
```

Out[6]:

<Axes: >



In [7]:

```
data.head()
```

Out[7]:

	Pregnan cies	Gluc ose	BloodPre ssure	SkinThick ness	Insulin	B MI	DiabetesPedigreeF unction	A g e	Outco me
0	6	148. 0	72.0	35.000000	79.7994 79	33 .6	0.627	50	1
1	1	85.0	66.0	29.000000	79.7994 79	26 .6	0.351	31	0
2	8	183. 0	64.0	20.536458	79.7994 79	23 .3	0.672	32	1
3	1	89.0	66.0	23.000000	94.0000 00	28 .1	0.167	21	0
4	0	137. 0	40.0	35.000000	168.000 000	43 .1	2.288	33	1

In [8]:

```
#segregate the dependent and independent variable
```

```
X = data.drop(columns = ['Outcome'])
```

```
y = data['Outcome']
```

In [9]:

```
# separate dataset into train and test
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)
```

```
X_train.shape, X_test.shape
```

Out[9]:

```
((576, 8), (192, 8))
```

In [10]:

```
import pickle
```

```
##standard Scaling- Standardization
```

```
def scaler_standard(X_train, X_test):
```

```
    #scaling the data
```

```
    scaler = StandardScaler()
```

```
    X_train_scaled = scaler.fit_transform(X_train)
```



```
X_test_scaled = scaler.transform(X_test)
```

```
#saving the model
```

```
file = open('standardScalar.pkl','wb')
```

```
pickle.dump(scaler,file)
```

```
file.close()
```

```
return X_train_scaled, X_test_scaled
```

In [11]:

```
X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```

In [12]:

```
X_train_scaled
```

Out[12]:

```
array([[ 1.50755225, -1.09947934, -0.89942504, ..., -1.45561965,
        -0.98325882, -0.04863985],
       [-0.82986389, -0.1331471 , -1.23618124, ...,  0.09272955,
        -0.62493647, -0.88246592],
       [-1.12204091, -1.03283573,  0.61597784, ..., -0.03629955,
         0.39884168, -0.5489355 ],
       ...,
       [ 0.04666716, -0.93287033, -0.64685789, ..., -1.14021518,
        -0.96519215, -1.04923114],
       [ 2.09190629, -1.23276654,  0.11084355, ..., -0.36604058,
        -0.5075031 ,  0.11812536],
       [ 0.33884418,  0.46664532,  0.78435594, ..., -0.09470985,
         0.51627505,  2.953134 ]])
```

In [13]:

```
log_reg = LogisticRegression()
```

```
log_reg.fit(X_train_scaled,y_train)
```

Out[13]:

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [14]:

```

## Hyperparameter Tuning
## GridSearch CV
from sklearn.model_selection import GridSearchCV
import numpy as np
import warnings
warnings.filterwarnings('ignore')
# parameter grid
parameters = {
    'penalty' : ['l1','l2'],
    'C'      : np.logspace(-3,3,7),
    'solver' : ['newton-cg', 'lbfgs', 'liblinear'],
}

```

In [15]:

```

logreg = LogisticRegression()
clf = GridSearchCV(logreg,          # model
                  param_grid = parameters, # hyperparameters
                  scoring='accuracy',    # metric for scoring
                  cv=10)                # number of folds

clf.fit(X_train_scaled,y_train)

```

Out[15]:

```

GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]),
                         'penalty': ['l1', 'l2'],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear']}],
             scoring='accuracy')

```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [16]:

```

clf.best_params_

```

Out[16]:

```

{'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}

```

In [17]:

```

clf.best_score_

```

Out[17]:

0.763793103448276

let's see how well our model performs on the test data set.

In [18]:

```
y_pred = clf.predict(X_test_scaled)
```

```
accuracy = accuracy_score(y_test,y_pred) accuracy
```

In [19]:

```
conf_mat = confusion_matrix(y_test,y_pred)
```

```
conf_mat
```

Out[19]:

```
array([[117, 13],  
       [ 26, 36]])
```

In [20]:

```
true_positive = conf_mat[0][0]
```

```
false_positive = conf_mat[0][1]
```

```
false_negative = conf_mat[1][0]
```

```
true_negative = conf_mat[1][1]
```

In [21]:

```
Accuracy = (true_positive + true_negative) / (true_positive + false_positive + false_negative +  
true_negative)
```

```
Accuracy
```

Out[21]:

```
0.796875
```

In [22]:

```
Precision = true_positive/(true_positive+false_positive)
```

```
Precision
```

Out[22]:

```
0.9
```

In [23]:

```
Recall = true_positive/(true_positive+false_negative)
```

```
Recall
```

Out[23]:

```
0.8181818181818182
```

In [24]:

$$F1_Score = 2 * (Recall * Precision) / (Recall + Precision)$$

F1_Score

Out[24]:

0.8571428571428572

In [25]:

import pickle

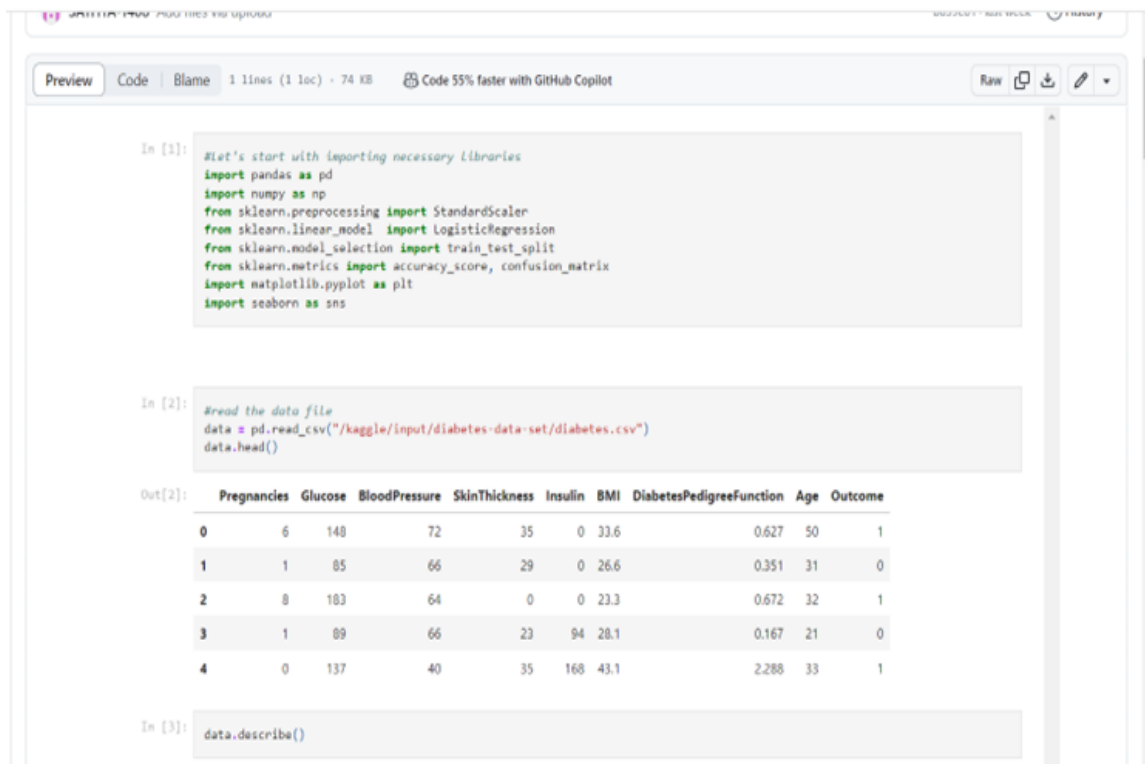
file = open('modelForPrediction.pkl','wb')

pickle.dump(log_reg,file)

file.close()

Screenshots:

1.



```
In [1]: #let's start with importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

In [2]: #read the data file
data = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
data.head()

Out[2]:
```

	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

```
In [3]: data.describe()
```

2.

```
CodeDemo / diabetespredictor-using-logistic-regression.ipynb
Preview Code Blame 1 lines (1 loc) · 74 KB Code 55% faster with GitHub Copilot

In [4]: data.isnull().sum()

Out[4]: Pregnancies      0
        Glucose          0
        BloodPressure    0
        SkinThickness    0
        Insulin          0
        BMI              0
        DiabetesPedigreeFunction  0
        Age              0
        Outcome          0
        dtype: int64

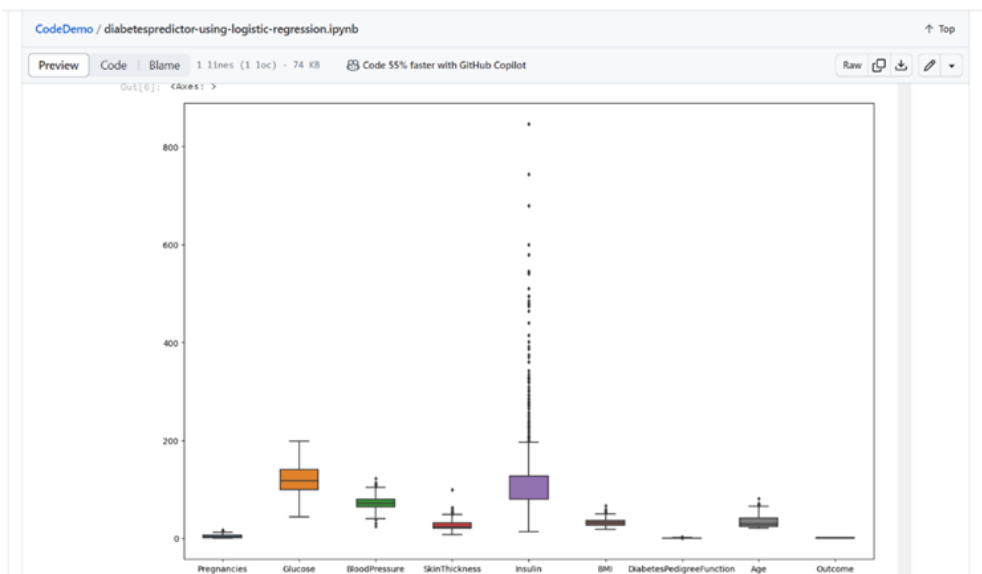
We can see there few data for columns Glucose , Insulin, skin thickness, BMI and Blood Pressure which have value as 0. That's not possible, right? you can do a quick search to see that one cannot have 0 values for these. Let's deal with that, we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

In [5]: #there few misconception is there like BMI can not be zero, BP can't be zero, glucose, insulin can't be zero so lets try to
        # now replacing zero values with the mean of the column
        data['BMI'] = data['BMI'].replace(0, data['BMI'].mean())
        data['BloodPressure'] = data['BloodPressure'].replace(0, data['BloodPressure'].mean())
        data['Glucose'] = data['Glucose'].replace(0, data['Glucose'].mean())
        data['Insulin'] = data['Insulin'].replace(0, data['Insulin'].mean())
        data['SkinThickness'] = data['SkinThickness'].replace(0, data['SkinThickness'].mean())

In [6]: #now we have dealt with the 0 values and data looks better. But, there still are outliers present in some columns.lets vi
        fig, ax = plt.subplots(figsize=(15,10))
        sns.boxplot(data=data, width=0.5, ax=ax, fliersize=3)

Out[6]: <Axes: >
```

3.



4.

```
CodeDemo / diabetespredictor-using-logistic-regression.ipynb
Preview Code Blame 1 lines (1 loc) · 74 KB Code 55% faster with GitHub Copilot
Raw Copy Edit

#saving the model
file = open('standardScaler.pkl','wb')
pickle.dump(scaler,file)
file.close()

return X_train_scaled, X_test_scaled

In [11]: X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)

In [12]: X_train_scaled

Out[12]: array([[ 1.58755225, -1.09947934, -0.89942504, ..., -1.45561965,
                -0.98325882, -0.04863985],
                [-0.82986389, -0.1331471, -1.23618124, ..., 0.09272955,
                -0.62491647, -0.88246592],
                [-1.12204091, -1.03283573, 0.61597784, ..., -0.03629955,
                0.39884168, -0.5489355 ],
                ...,
                [ 0.04666716, -0.93287833, -0.64685789, ..., -1.14021518,
                -0.96519215, -1.04923114],
                [ 2.09190629, -1.23276654, 0.11084355, ..., -0.36604058,
                -0.5875031, 0.11812536],
                [ 0.33884418, 0.4664532, 0.78435594, ..., -0.09478985,
                0.51627505, 2.953134 ]])

In [13]: log_reg = LogisticRegression()
log_reg.fit(X_train_scaled,y_train)

Out[13]: LogisticRegression()
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

5.

```
CodeDemo / diabetespredictor-using-logistic-regression.ipynb
Preview Code Blame 1 lines (1 loc) · 74 KB Code 55% faster with GitHub Copilot
Raw Copy Edit

In [14]: ## Hyperparameter Tuning
## GridSearch CV
from sklearn.model_selection import GridSearchCV
import numpy as np
import warnings
warnings.filterwarnings('ignore')
# parameter: grid
parameters = {
    'penalty': ['l1','l2'],
    'C': np.logspace(-3,3,7),
    'solver': ['newton-cg', 'lbfgs', 'liblinear'],
}

In [15]: logreg = LogisticRegression()
clf = GridSearchCV(logreg, # model
                  param_grid = parameters, # hyperparameters
                  scoring='accuracy', # metric for scoring
                  cv=10 # number of folds)

clf.fit(X_train_scaled,y_train)

Out[15]: GridSearchCV(cv=10, estimator=LogisticRegression(),
                    param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]),
                    'penalty': ['l1', 'l2'],
                    'solver': ['newton-cg', 'lbfgs', 'liblinear']},
                    scoring='accuracy')

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [16]: clf.best_params_
```

6.



```
CodeDemo / diabetespredictor-using-logistic-regression.ipynb
Preview Code Blame 1 lines (1 loc) · 74 KB Code 55% faster with GitHub Copilot
Raw Copy Download Edit

Out[16]: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}

In [17]: clf.best_score_

Out[17]: 0.763793103448276

let's see how well our model performs on the test data set.

In [18]: y_pred = clf.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred) accuracy

In [19]: conf_mat = confusion_matrix(y_test, y_pred)
conf_mat

Out[19]: array([[117, 13],
               [ 26, 36]])

In [20]: true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]

In [21]: Accuracy = (true_positive + true_negative) / (true_positive + false_positive + false_negative + true_negative)
Accuracy

Out[21]: 0.796875

In [22]: Precision = true_positive / (true_positive + false_positive)
Recall = true_negative / (true_negative + false_negative)
```

CONCLUSION:

Therefore, in this technology we built our project by selecting a machine learning algorithm, training the model, and evaluated its performance by using the given dataset.