**Title: Comprehensive Report on Diabetes Prediction Using Machine Learning**

**1. Introduction**

Diabetes Mellitus is a group of metabolic disorders characterized by high blood sugar levels over a prolonged period. It is one of the leading causes of death globally and can lead to serious complications like heart disease, kidney failure, and blindness if not detected and managed early. The purpose of this project is to build a reliable machine learning model to predict whether a patient is likely to develop diabetes based on medical diagnostic measurements.

This project uses the Pima Indians Diabetes Dataset, which is a widely recognized dataset in the machine learning community. It contains information collected from female patients of Pima Indian heritage who are at least 21 years old.

**2. Dataset Description**

The dataset comprises 768 instances and 9 columns (8 features and 1 target). The attributes include:

| Feature | Description |
| --- | --- |
| Pregnancies | Number of times the patient has been pregnant |
| Glucose | Plasma glucose concentration after a 2-hour oral glucose tolerance test |
| BloodPressure | Diastolic blood pressure (mm Hg) |
| SkinThickness | Triceps skin fold thickness (mm) |
| Insulin | 2-Hour serum insulin (mu U/ml) |
| BMI | Body mass index (weight in kg/(height in m)^2) |
| DiabetesPedigreeFunction | A function that scores the likelihood of diabetes based on family history |
| Age | Patient’s age in years |
| Outcome | Target variable (0 = non-diabetic, 1 = diabetic) |

**3. Data Preprocessing**

Real-world medical datasets often contain anomalies or missing values, which need to be addressed to avoid bias in predictions.

* **Zero Value Handling**:
  + Fields such as Glucose, BloodPressure, SkinThickness, Insulin, and BMI cannot realistically have zero values.
  + These zeros were replaced with NaN using:
  + data[columns] = data[columns].replace(0, np.nan)
* **Missing Value Imputation**:
  + All NaN values were filled with the median of each respective column:
  + data.fillna(data.median(), inplace=True)
* **Feature Scaling**:
  + Used StandardScaler from sklearn.preprocessing to standardize features:
  + scaler = StandardScaler()  
    X\_scaled = scaler.fit\_transform(X)
  + This is especially helpful for models like SVM and KNN.
* **Train-Test Split**:
  + Dataset was split into 80% training and 20% testing data using:
  + from sklearn.model\_selection import train\_test\_split  
    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

**4. Model Training**

We explored several classification algorithms:

* **Logistic Regression**:
  + A linear model ideal for binary classification:
  + from sklearn.linear\_model import LogisticRegression  
    model = LogisticRegression()  
    model.fit(X\_train, y\_train)
* **Random Forest Classifier**:
  + An ensemble technique that uses multiple decision trees:
  + from sklearn.ensemble import RandomForestClassifier  
    rf\_model = RandomForestClassifier()  
    rf\_model.fit(X\_train, y\_train)
* **Support Vector Machine (SVM)**:
  + Good for high-dimensional spaces:
  + from sklearn.svm import SVC  
    svm\_model = SVC(probability=True)  
    svm\_model.fit(X\_train, y\_train)
* **K-Nearest Neighbors (KNN)**:
  + Simple method based on distance:
  + from sklearn.neighbors import KNeighborsClassifier  
    knn\_model = KNeighborsClassifier()  
    knn\_model.fit(X\_train, y\_train)

**5. Model Evaluation**

To assess model performance:

* **Accuracy Score**:
  + Overall correctness:
  + from sklearn.metrics import accuracy\_score  
    accuracy = accuracy\_score(y\_test, y\_pred)
* **Confusion Matrix**:
  + Compares predicted vs actual values:
  + from sklearn.metrics import confusion\_matrix  
    print(confusion\_matrix(y\_test, y\_pred))
* **Classification Report**:
  + Gives precision, recall, and F1-score:
  + from sklearn.metrics import classification\_report  
    print(classification\_report(y\_test, y\_pred))
* **ROC-AUC Curve**:
  + Measures classifier quality:
  + from sklearn.metrics import roc\_auc\_score  
    y\_probs = rf\_model.predict\_proba(X\_test)[:,1]  
    roc\_auc = roc\_auc\_score(y\_test, y\_probs)

**6. Visualization and Insights**

* **Confusion Matrix Heatmap**:
* import seaborn as sns  
  sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True)
* **ROC Curve**:
* from sklearn.metrics import roc\_curve  
  fpr, tpr, \_ = roc\_curve(y\_test, y\_probs)
* **Feature Importance (from Random Forest)**:
* importance = rf\_model.feature\_importances\_

Top features influencing diabetes prediction:

1. Glucose
2. BMI
3. Age
4. Diabetes Pedigree Function
5. Insulin

**7. Model Deployment and Real-Time Prediction**

* Save the model and scaler:
* import joblib  
  joblib.dump(rf\_model, 'diabetes\_model.pkl')  
  joblib.dump(scaler, 'scaler.pkl')
* Predict for new input:
* new\_data = [[2, 120, 70, 25, 79, 28.5, 0.25, 35]]  
  new\_data\_scaled = scaler.transform(new\_data)  
  prediction = rf\_model.predict(new\_data\_scaled)

**8. Conclusion and Future Work**

This project successfully demonstrates how machine learning can help predict diabetes early using easily available clinical features.

**Improvements for future versions:**

* Fine-tuning with GridSearchCV
* Adding cross-validation
* Creating a web app using Streamlit
* Adding deep learning (e.g., with TensorFlow)
* Interpretable ML using SHAP/ELI5

This project can be deployed in health centers for preliminary screening of patients.