# DIABETES PREDICTION

USING MACHINE LEARNING

# INTRODUCTION

Diabetes is a serious condition where high blood sugar levels can cause significant health problems. We have a dataset from the National Institute of Diabetes and Digestive and Kidney Diseases, which includes health information from Pima Indian women aged 21 and older. This dataset contains key factors like number of pregnancies, BMI, insulin levels, and age. Our goal is to develop a machine learning model to accurately predict whether a patient has diabetes based on these factors.

# 1) EXPLORATORY DATA ANALYSIS PT.1

```
#Installation of required libraries
import numpy as np
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error, r2_score, roc_auc_score, roc_curve, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import KFold
import warnings
warnings.simplefilter(action = "ignore")
```

```
#Reading the dataset
df = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
```

```
# The first 5 observation units of the data set were accessed.
df.head()
```

```
# The size of the data set was examined. It consists of 768 observation units and 9 variables.
df.shape
```

```
#Feature information
df.info()
```

```
# Descriptive statistics of the data set accessed.
df.describe([0.10,0.25,0.50,0.75,0.90,0.95,0.99]).T
```

```
# The distribution of the Outcome variable was examined.
df["Outcome"].value_counts()*100/len(df)
```

```
# The classes of the outcome variable were examined.
df.Outcome.value_counts()
```

```
# The histagram of the Age variable was reached.
df["Age"].hist(edgecolor = "black");
```

```
print("Max Age: " + str(df["Age"].max()) + " Min Age: " + str(df["Age"].min()))
```

## Explanation:

- Load packages.
- Load dataset.
- Show first 5 rows.
- Display dimensions.
- Describe features.
- Summary statistics.
- Check target distribution.
- Count target classes.
- Plot age distribution.
- Print age range.

Azmah

# PT.2

```
# Histogram and density graphs of all variables were accessed.
fig, ax = plt.subplots(4,2, figsize=(16,16))
sns.distplot(df.Age, bins = 20, ax=ax[0,0])
sns.distplot(df.Pregnancies, bins = 20, ax=ax[0,1])
sns.distplot(df.Glucose, bins = 20, ax=ax[1,0])
sns.distplot(df.BloodPressure, bins = 20, ax=ax[1,1])
sns.distplot(df.SkinThickness, bins = 20, ax=ax[2,0])
sns.distplot(df.Insulin, bins = 20, ax=ax[2,1])
sns.distplot(df.DiabetesPedigreeFunction, bins = 20, ax=ax[3,0])
sns.distplot(df.BMI, bins = 20, ax=ax[3,1])
```

```
df.groupby("Outcome").agg({"Pregnancies":"mean"})
```

```
df.groupby("Outcome").agg({"Age":"mean"})
```

```
df.groupby("Outcome").agg({"Age":"max"})
```

```
df.groupby("Outcome").agg({"Insulin": "mean"})
```

```
df.groupby("Outcome").agg({"Insulin": "max"})
```

```
df.groupby("Outcome").agg({"Glucose": "mean"})
```

```
df.groupby("Outcome").agg({"Glucose": "max"})
```

```
df.groupby("Outcome").agg({"BMI": "mean"})
```

```
# The distribution of the outcome variable in the data was examined and visualized.
f,ax=plt.subplots(1,2,figsize=(18,8))
df['Outcome'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('target')
ax[0].set_ylabel('')
ax[1].set_title('Outcome')
plt.show()
```

```
# Access to the correlation of the data set was provided. What kind of relationship is examined between the variables.
# If the correlation value is> 0, there is a positive correlation. While the value of one variable increases, the value of the other variable also increases.
# Correlation = 0 means no correlation.
# If the correlation is <0, there is a negative correlation. While one variable increases, the other variable decreases.
# When the correlations are examined, there are 2 variables that act as a positive correlation to the Salary dependent variable.
# These variables are Glucose. As these increase, Outcome variable increases.
df.corr()
```

```
# Correlation matrix graph of the data set
f, ax = plt.subplots(figsize= [20,15])
sns.heatmap(df.corr(), annot=True, fmt=".2f", ax=ax, cmap = "magma" )
ax.set_title("Correlation Matrix", fontsize=20)
plt.show()
```

## Explanation:

- Plot histograms and density plots for key variables.
- Compute mean and max values by "Outcome".
- Visualize "Outcome" distribution with pie and count plots.
- Show and visualize correlation matrix.

**Azmah**

# 2) DATA PREPROCESSING

## 2.1) Missing Observation Analysis

```
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
```

```
df.head()
```

```
# Now, we can look at where are missing values
df.isnull().sum()
```

```
# Have been visualized using the missingno library for the visualization of missing observations.
# Plotting
import missingno as msno
msno.bar(df);
```

```
# The missing values will be filled with the median values of each variable.
def median_target(var):
    temp = df[df[var].notnull()]
    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_index()
    return temp
```

```
# The values to be given for incomplete observations are given the median value of people who are not sick and the median values of people who are sick.
columns = df.columns
columns = columns.drop("Outcome")
for i in columns:
    median_target(i)
    df.loc[(df['Outcome'] == 0 ) & (df[i].isnull()), i] = median_target(i)[i][0]
    df.loc[(df['Outcome'] == 1 ) & (df[i].isnull()), i] = median_target(i)[i][1]
```

```
df.head()
```

```
# Missing values were filled.
df.isnull().sum()
```

Navya

# 2) DATA PREPROCESSING

## 2.2) Outlier Observation Analysis

```
# In the data set, there were asked whether there were any outlier observations compared to the 25%
and 75% quarters.
# It was found to be an outlier observation.
for feature in df:

    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1- 1.5*IQR
    upper = Q3 + 1.5*IQR

    if df[(df[feature] > upper)].any(axis=None):
        print(feature,"yes")
    else:
        print(feature, "no")
```

```
# The process of visualizing the Insulin variable with boxplot method was done. We find the outlier
observations on the chart.
import seaborn as sns
sns.boxplot(x = df["Insulin"]);
```

```
#We conduct a stand alone observation review for the Insulin variable
#We suppress contradictory values
Q1 = df.Insulin.quantile(0.25)
Q3 = df.Insulin.quantile(0.75)
IQR = Q3-Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
df.loc[df["Insulin"] > upper,"Insulin"] = upper
```

```
import seaborn as sns
sns.boxplot(x = df["Insulin"]);
```

Navya

# 2) DATA PREPROCESSING

## 2.3) Local Outlier Factor (LOF)

```
# We determine outliers between all variables with the LOF method
from sklearn.neighbors import LocalOutlierFactor
lof =LocalOutlierFactor(n_neighbors= 10)
lof.fit_predict(df)
```

```
df_scores = lof.negative_outlier_factor_
np.sort(df_scores)[0:30]
```

```
#We choose the threshold value according to lof scores
threshold = np.sort(df_scores)[7]
threshold
```

```
#We delete those that are higher than the threshold
outlier = df_scores > threshold
df = df[outlier]
```

```
# The size of the data set was examined.
df.shape
```

Navya

# 3) FEATURE ENGINEERING

```
NewBMI = pd.Series(["Underweight", "Normal", "Overweight", "Obesity 1", "Obesity 2", "Obes
df["NewBMI"] = NewBMI #Removed extra indent
df.loc[df["BMI"] < 18.5, "NewBMI"] = NewBMI[0]
df.loc[(df["BMI"] > 18.5) & (df["BMI"] <= 24.9), "NewBMI"] = NewBMI[1]
df.loc[(df["BMI"] > 24.9) & (df["BMI"] <= 29.9), "NewBMI"] = NewBMI[2]
df.loc[(df["BMI"] > 29.9) & (df["BMI"] <= 34.9), "NewBMI"] = NewBMI[3]
df.loc[(df["BMI"] > 34.9) & (df["BMI"] <= 39.9), "NewBMI"] = NewBMI[4]
df.loc[df["BMI"] > 39.9 ,"NewBMI"] = NewBMI[5]
```

```
df.head()
```

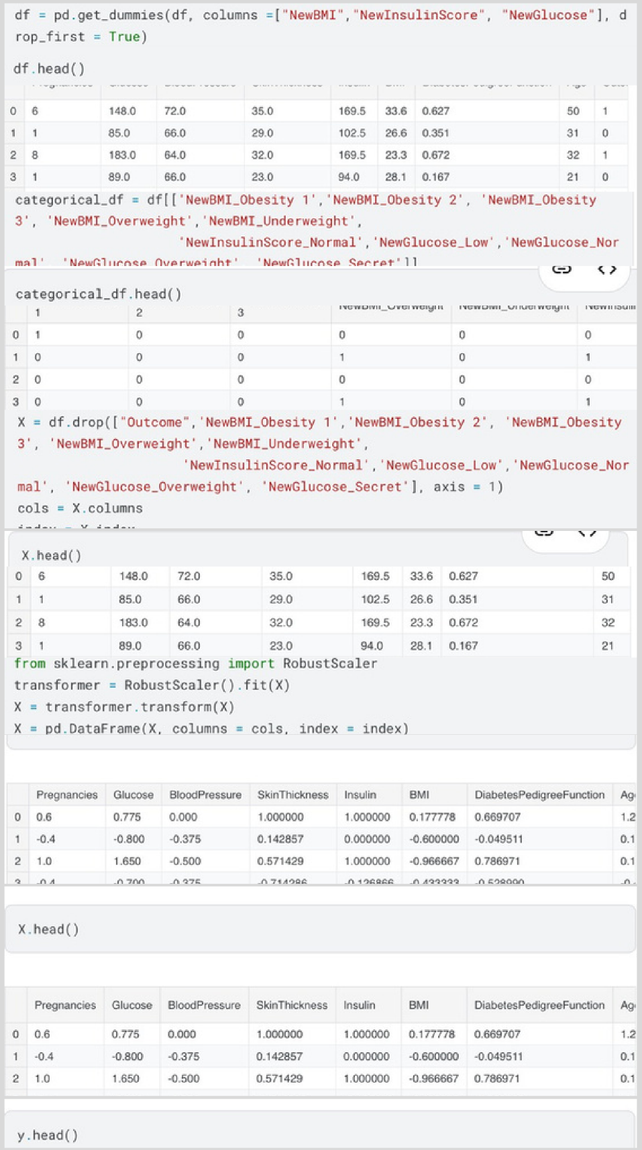
| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedig |
|---|---|---|---|---|---|---|
| 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | |
| 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | |
| 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | |
| 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | |
| 0 | 137.0 | 40.0 | 35.0 | 168.0 | 43.1 | |

- Missing values are handled by replacing them with medians.
- Outliers are removed.
- New features are created from BMI, Insulin, and Glucose.
- Categorical features are one-hot encoded.
- Numerical features are scaled.

Aarav

# 4) ONE HOT ENCODING

- Standardizing variables improves model performance.
- Common standardization methods include:
  - Normalize
  - MinMax
  - Robust
  - Scale



```
df = pd.get_dummies(df, columns =["NewBMI","NewInsulinScore", "NewGlucose"], d
rop_first = True)

df.head()
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 | 0 |

```
categorical_df = df[['NewBMI_Obesity 1','NewBMI_Obesity 2', 'NewBMI_Obesity
3', 'NewBMI_Overweight','NewBMI_Underweight',
                     'NewInsulinScore_Normal','NewGlucose_Low','NewGlucose_Nor
mal', 'NewGlucose_Overweight', 'NewGlucose_Secret']]

categorical_df.head()
```

| | 1 | 2 | 3 | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 1 |

```
X = df.drop(["Outcome",'NewBMI_Obesity 1','NewBMI_Obesity 2', 'NewBMI_Obesity
3', 'NewBMI_Overweight','NewBMI_Underweight',
                     'NewInsulinScore_Normal','NewGlucose_Low','NewGlucose_Nor
mal', 'NewGlucose_Overweight', 'NewGlucose_Secret'], axis = 1)
cols = X.columns

X.head()
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 6 | 148.0 | 72.0 | 35.0 | 169.5 | 33.6 | 0.627 | 50 |
| 1 | 1 | 85.0 | 66.0 | 29.0 | 102.5 | 26.6 | 0.351 | 31 |
| 2 | 8 | 183.0 | 64.0 | 32.0 | 169.5 | 23.3 | 0.672 | 32 |
| 3 | 1 | 89.0 | 66.0 | 23.0 | 94.0 | 28.1 | 0.167 | 21 |

```
from sklearn.preprocessing import RobustScaler
transformer = RobustScaler().fit(X)
X = transformer.transform(X)
X = pd.DataFrame(X, columns = cols, index = index)
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Ag |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.6 | 0.775 | 0.000 | 1.000000 | 1.000000 | 0.177778 | 0.669707 | 1.2 |
| 1 | -0.4 | -0.800 | -0.375 | 0.142857 | 0.000000 | -0.600000 | -0.049511 | 0.1 |
| 2 | 1.0 | 1.650 | -0.500 | 0.571429 | 1.000000 | -0.966667 | 0.786971 | 0.1 |

```
X.head()
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Ag |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.6 | 0.775 | 0.000 | 1.000000 | 1.000000 | 0.177778 | 0.669707 | 1.2 |
| 1 | -0.4 | -0.800 | -0.375 | 0.142857 | 0.000000 | -0.600000 | -0.049511 | 0.1 |
| 2 | 1.0 | 1.650 | -0.500 | 0.571429 | 1.000000 | -0.966667 | 0.786971 | 0.1 |

```
y.head()
```

Rudraksh

# 5) BASE MODELS

```
# Validation scores of all base models

models = []
models.append(('LR', LogisticRegression(random_state = 12345)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state = 12345)))
models.append(('RF', RandomForestClassifier(random_state = 12345)))
models.append(('SVM', SVC(gamma='auto', random_state = 12345)))
models.append(('XGB', GradientBoostingClassifier(random_state = 12345)))
models.append(("LightGBM", LGBMClassifier(random_state = 12345)))

# evaluate each model in turn
results = []
names = []
```

```
for name, model in models:


        cv_results = cross_val_score(model, X, y, cv = 10, scoring= "accuracy")
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

# boxplot algorithm comparison
fig = plt.figure(figsize=(15,10))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

## Explanation:

- Initialize classifiers.
- Evaluate the models.
- Print accuracy stats.
- Show boxplot comparison.

# 6) MODEL TUNING

## 1) Random Forests Tuning

Tuning a RandomForestClassifier involves defining a parameter grid (`n_estimators`, `max_features`, `min_samples_split`, `max_depth`), using GridSearchCV to find the best combination, training the final model with these optimal parameters, evaluating it with 10-fold cross-validation, and visualizing feature importance using the `feature_importances_` attribute and a bar plot.

## 3) XGBoost Tuning

To tune a GradientBoostingClassifier with GridSearchCV, define a parameter grid for `learning_rate`, `min_samples_split`, `max_depth`, `subsample`, and `n_estimators`. Use GridSearchCV to find the best parameters and train the final model with them. Evaluate the model using 10-fold cross-validation and plot feature importance to visualize each feature's contribution.

## 2) LightGBM Tuning

To tune a LightGBM model using GridSearchCV, define parameters (`learning_rate`, `n_estimators`, `max_depth`) and create an LGBMClassifier. Use GridSearchCV to find the best parameters, train the model with these, and perform 10-fold cross-validation to evaluate performance. Finally, visualize feature importance with a bar plot.

Navya

# 7) COMPARISON OF FINAL MODELS

```
from sklearn.linear_model import LogisticRegression

models = []
models.append(('LR', LogisticRegression(random_state = 12345)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state = 12345)))
models.append(('RF', RandomForestClassifier(random_state = 12345)))
models.append(('SVM', SVC(gamma='auto', random_state = 12345)))
models.append(('XGB', GradientBoostingClassifier(random_state = 12345)))
models.append(("LightGBM", LGBMClassifier(random_state = 12345)))
results = []
names = []

for name, model in models:

    kfold = KFold((n_splits = 10, random_state = 12345))
    cv_results = cross_val_score(model, X, y, cv = 10, scoring= "accuracy")
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

The Code compares 7 machine learning models (LR, KNN, CART, RF, SVM, XGB, LightGBM) using 10-fold cross-validation and accuracy as the metric. The results are visualized in a boxplot, allowing you to compare their performance.

Aarav

# 8) REPORTING

Study Aim:
- Develop models to predict diabetes.
- Achieve high validation scores.

Work Done:

1. Data Reading:
   - Read the diabetes dataset.

2. EDA:
   - Checked structure and types.
   - Replaced zeros with NaN.
   - Analyzed statistics.

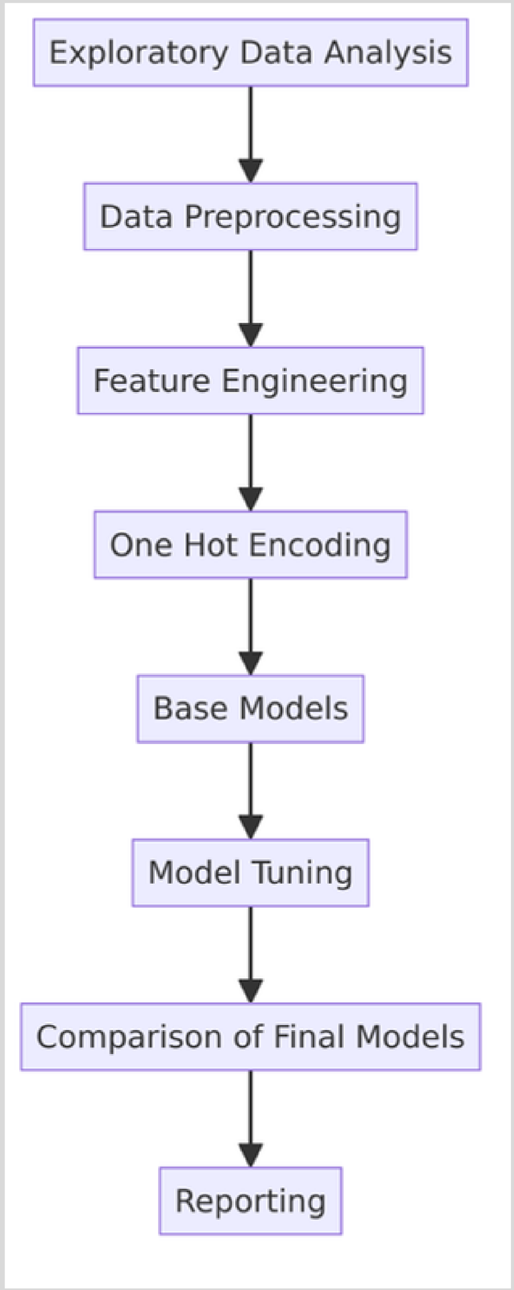
3. Data Preprocessing:
   - Filled NaN with median values.
   - Removed outliers.
   - Standardized variables.

4. Model Building:
   - Used various machine learning models.
   - Calculated Cross Validation Scores.
   - Optimized hyperparameters for better scores.

5. Result:
   - XGBoost with optimized hyperparameters had the best score (0.90).

Rudraksh

# FLOWCHART:




Rudraksh

# CONLCUSION

The code performs an exploratory data analysis on a diabetes dataset, handles missing values and outliers, performs feature engineering, and then compares the performance of different machine learning models to predict diabetes.

## WHAT WE DID:

- Prepared data for modeling.
- Compared machine learning models.
- Evaluated model performance.

Aarav

# THANK YOU

By Navya, Azmah, Rudraksh and Aarav.