Vishal Kompella

Building a Data Mining System, From Data Cleaning to Model Evaluation.

Date of Project -August 6th, 2023.

For this project, the Diabetes Dataset has been used from Kaggle. The link is provided at the end.

Each row represents an individual data point with various features such as Insulin, Glucose, Blood Pressure etc., and each data point is associated with a class label ("Outcome") indicating whether the individual has diabetes i.e., 1, or not as 0.

There are a total of 768 rows and 9 columns in this dataset.

The given dataset contains information about different individuals, where each row represents a specific person. Each column in the dataset represents a particular attribute or characteristic of the individuals. Here's a brief summary of each attribute:

1**.Pregnancies**: Number of times the individual has been pregnant.

2.**Glucose**: Plasma glucose concentration, measured in mg/dL.

3. **BloodPressure**: Diastolic blood pressure, measured in mm Hg.

4.**SkinThickness**: Triceps skinfold thickness, measured in mm.

5.**Insulin**: 2-Hour serum insulin level, measured in mu U/ml.

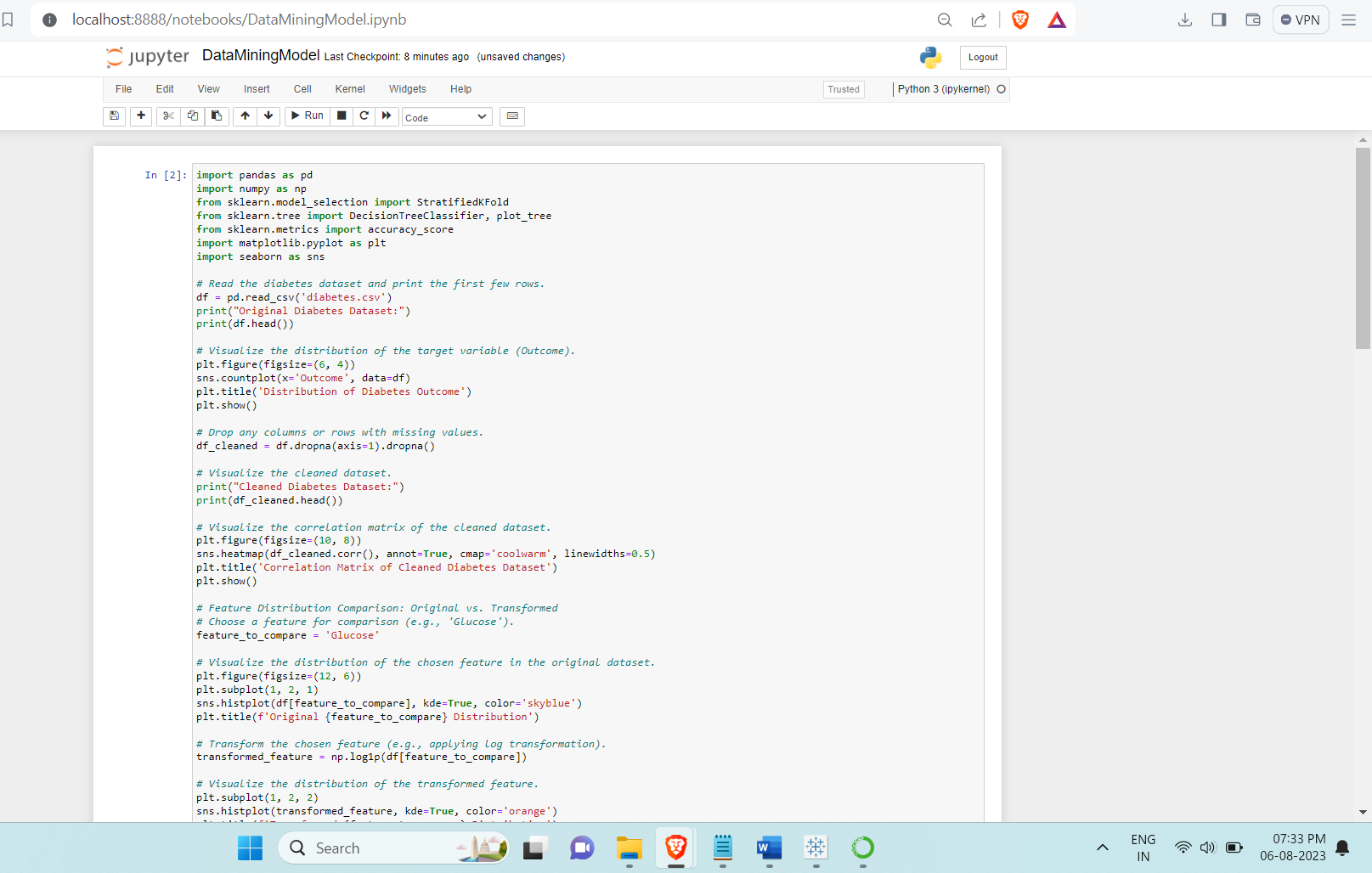
6.**BMI**: It is the Body mass index of the individual.

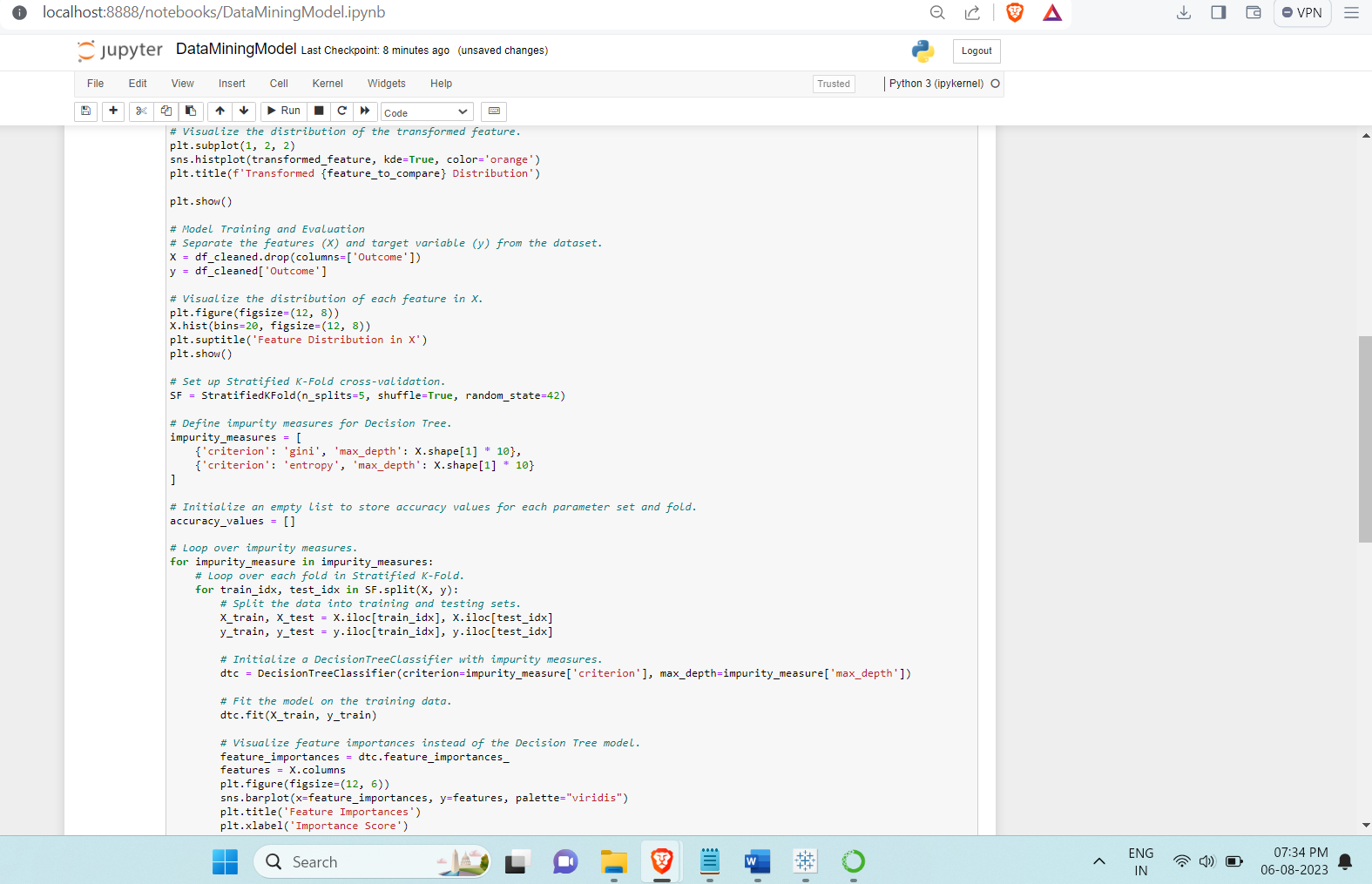
**DiabetesPedigreeFunction**: A function that scores the likelihood of diabetes based on family history.

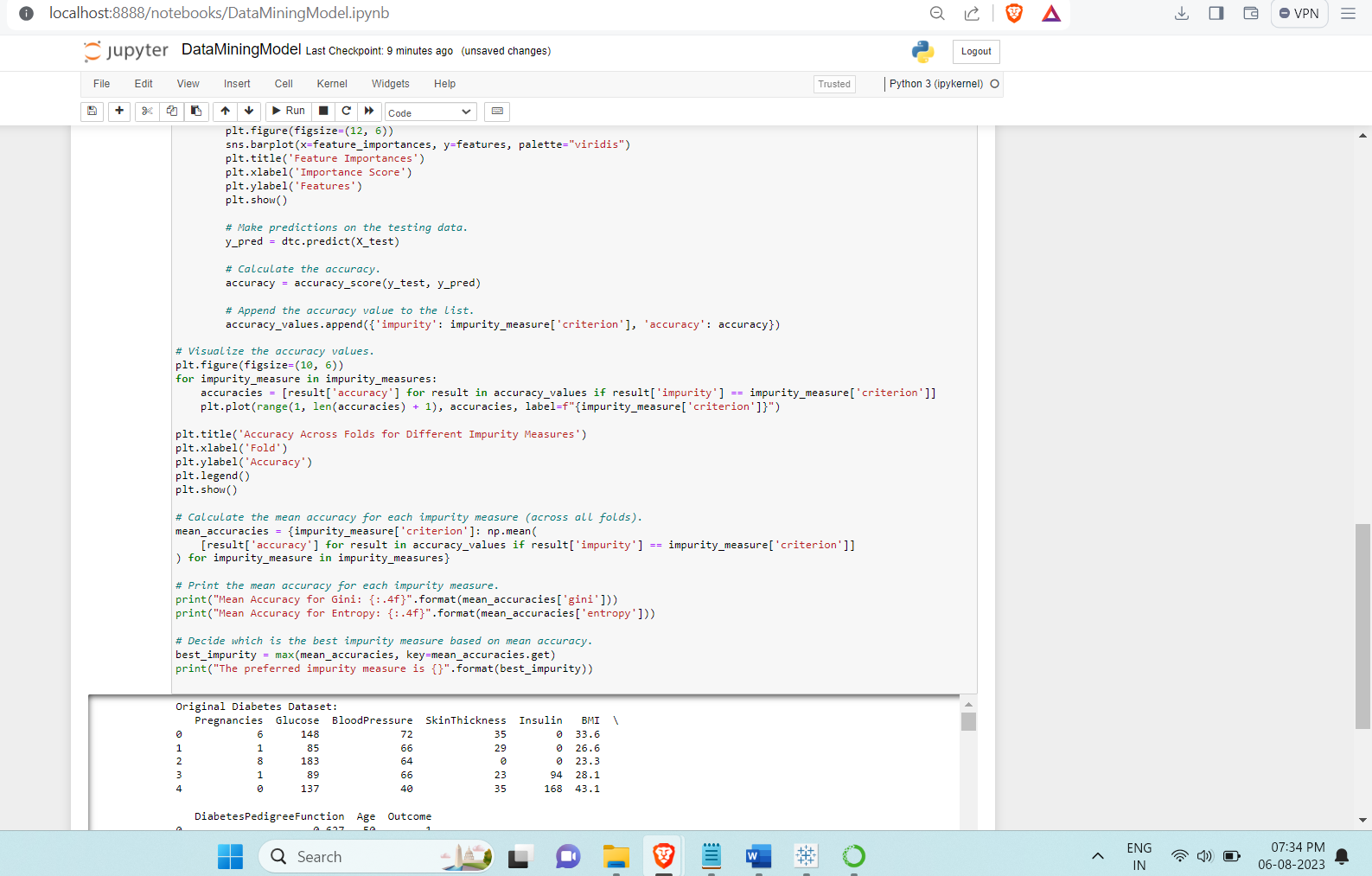
**Age**: Age of the individual in years.

**Outcome**: The target variable indicating the presence (1) or absence (0) of diabetes for each individual (the class label).

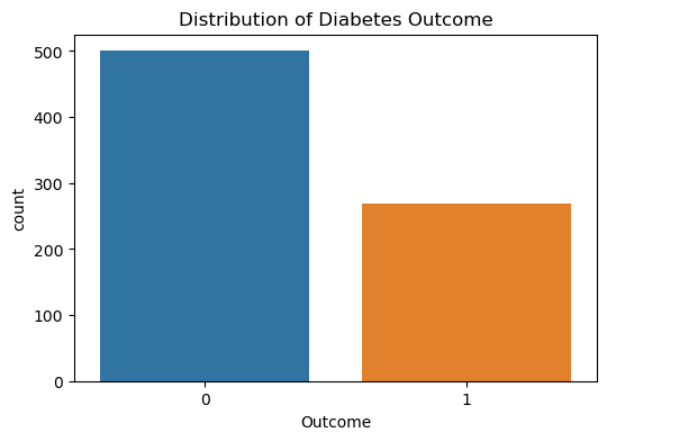
The screenshots for code after running in jupyter notebook is given below:

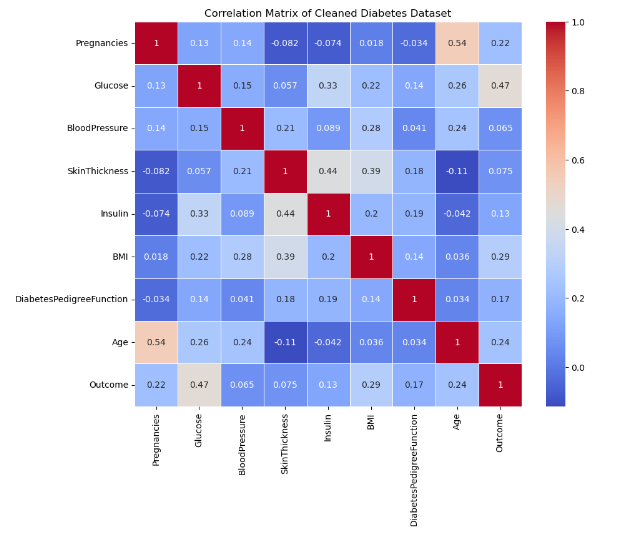


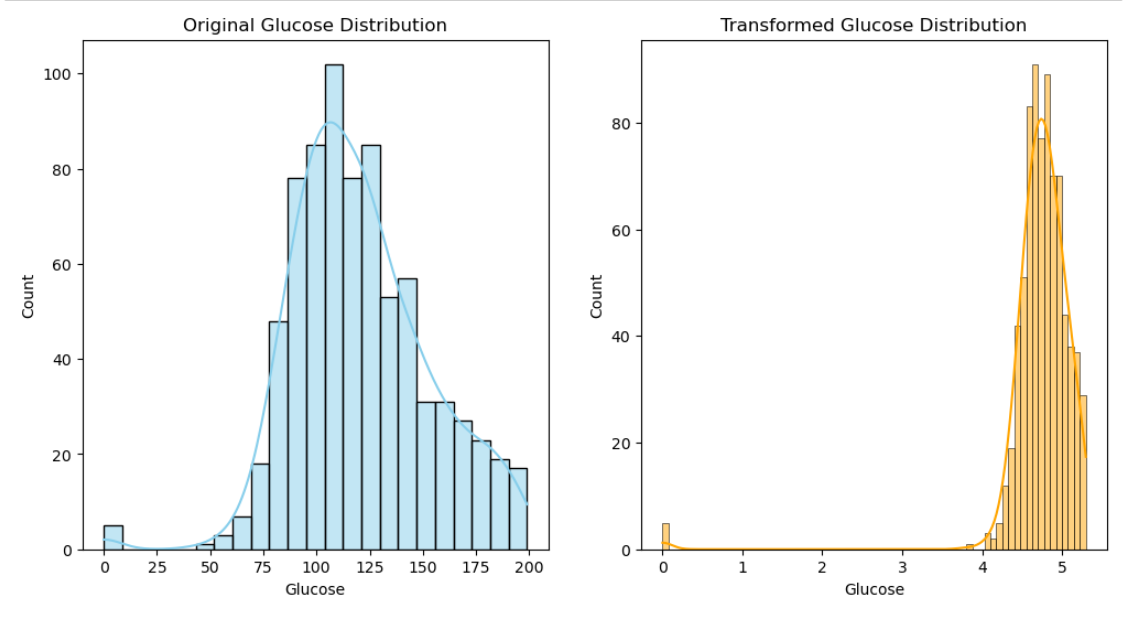


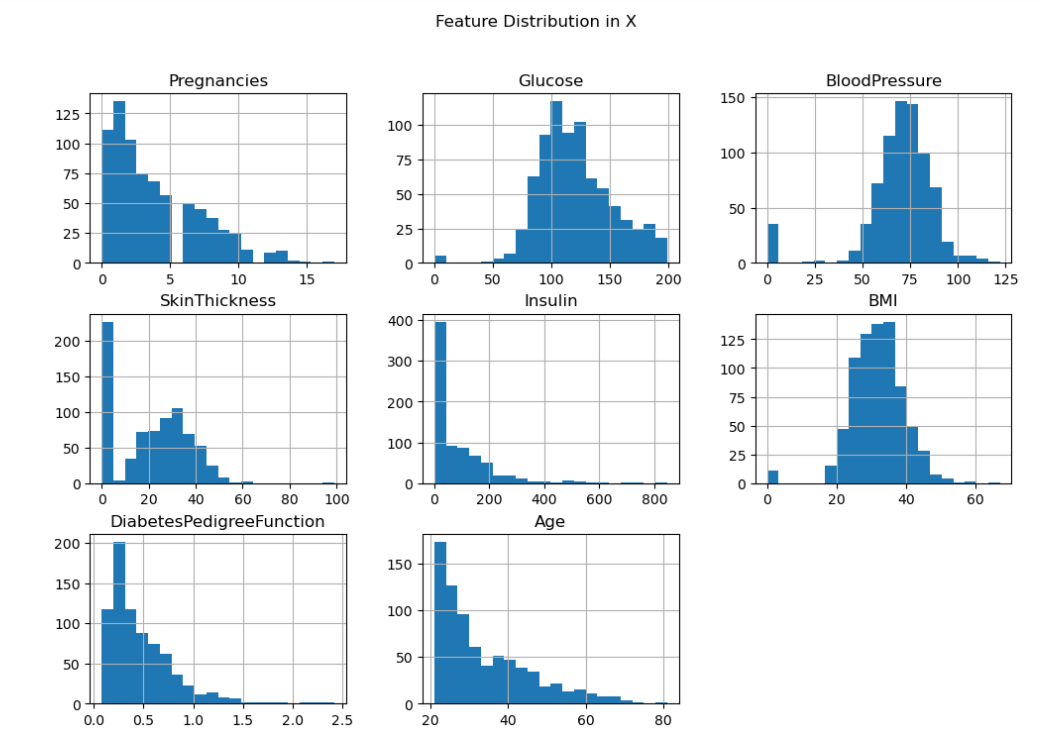


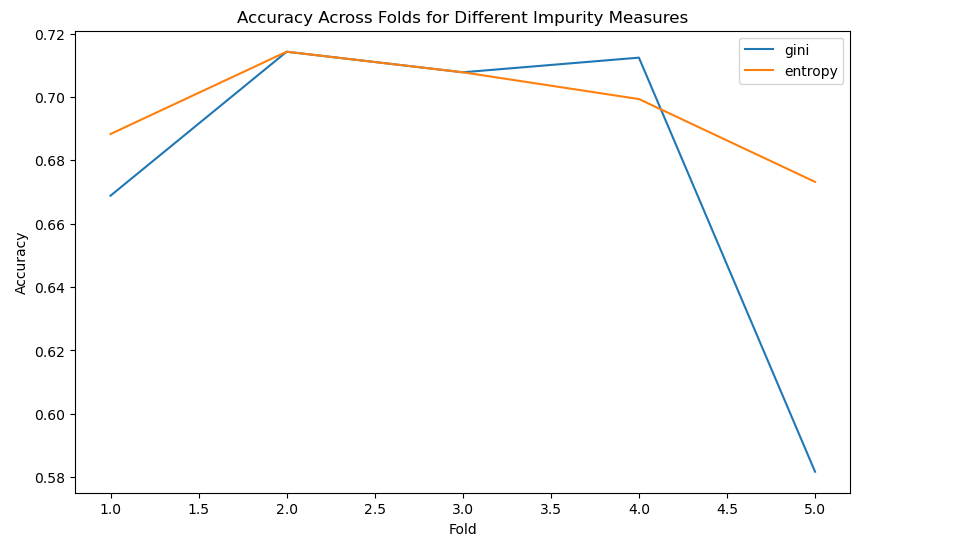
Following are the visualizations highlighted below:











Result:

Mean Gini = 0.688

Mean Entropy = 0.707

**CODE:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import StratifiedKFold

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

import seaborn as sns

# Read the diabetes dataset and print the first few rows.

df = pd.read\_csv('diabetes.csv')

print("Original Diabetes Dataset:")

print(df.head())

# Visualize the distribution of the target variable (Outcome).

plt.figure(figsize=(6, 4))

sns.countplot(x='Outcome', data=df)

plt.title('Distribution of Diabetes Outcome')

plt.show()

# Drop any columns or rows with missing values.

df\_cleaned = df.dropna(axis=1).dropna()

# Visualize the cleaned dataset.

print("Cleaned Diabetes Dataset:")

print(df\_cleaned.head())

# Visualize the correlation matrix of the cleaned dataset.

plt.figure(figsize=(10, 8))

sns.heatmap(df\_cleaned.corr(), annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Matrix of Cleaned Diabetes Dataset')

plt.show()

# Feature Distribution Comparison: Original vs. Transformed

# Choose a feature for comparison (e.g., 'Glucose').

feature\_to\_compare = 'Glucose'

# Visualize the distribution of the chosen feature in the original dataset.

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

sns.histplot(df[feature\_to\_compare], kde=True, color='skyblue')

plt.title(f'Original {feature\_to\_compare} Distribution')

# Transform the chosen feature (e.g., applying log transformation).

transformed\_feature = np.log1p(df[feature\_to\_compare])

# Visualize the distribution of the transformed feature.

plt.subplot(1, 2, 2)

sns.histplot(transformed\_feature, kde=True, color='orange')

plt.title(f'Transformed {feature\_to\_compare} Distribution')

plt.show()

# Model Training and Evaluation

# Separate the features (X) and target variable (y) from the dataset.

X = df\_cleaned.drop(columns=['Outcome'])

y = df\_cleaned['Outcome']

# Visualize the distribution of each feature in X.

plt.figure(figsize=(12, 8))

X.hist(bins=20, figsize=(12, 8))

plt.suptitle('Feature Distribution in X')

plt.show()

# Set up Stratified K-Fold cross-validation.

SF = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

# Define impurity measures for Decision Tree.

impurity\_measures = [

{'criterion': 'gini', 'max\_depth': X.shape[1] \* 10},

{'criterion': 'entropy', 'max\_depth': X.shape[1] \* 10}

]

# Initialize an empty list to store accuracy values for each parameter set and fold.

accuracy\_values = []

# Loop over impurity measures.

for impurity\_measure in impurity\_measures:

# Loop over each fold in Stratified K-Fold.

for train\_idx, test\_idx in SF.split(X, y):

# Split the data into training and testing sets.

X\_train, X\_test = X.iloc[train\_idx], X.iloc[test\_idx]

y\_train, y\_test = y.iloc[train\_idx], y.iloc[test\_idx]

# Initialize a DecisionTreeClassifier with impurity measures.

dtc = DecisionTreeClassifier(criterion=impurity\_measure['criterion'], max\_depth=impurity\_measure['max\_depth'])

# Fit the model on the training data.

dtc.fit(X\_train, y\_train)

# Visualize feature importances instead of the Decision Tree model.

feature\_importances = dtc.feature\_importances\_

features = X.columns

plt.figure(figsize=(12, 6))

sns.barplot(x=feature\_importances, y=features, palette="viridis")

plt.title('Feature Importances')

plt.xlabel('Importance Score')

plt.ylabel('Features')

plt.show()

# Make predictions on the testing data.

y\_pred = dtc.predict(X\_test)

# Calculate the accuracy.

accuracy = accuracy\_score(y\_test, y\_pred)

# Append the accuracy value to the list.

accuracy\_values.append({'impurity': impurity\_measure['criterion'], 'accuracy': accuracy})

# Visualize the accuracy values.

plt.figure(figsize=(10, 6))

for impurity\_measure in impurity\_measures:

accuracies = [result['accuracy'] for result in accuracy\_values if result['impurity'] == impurity\_measure['criterion']]

plt.plot(range(1, len(accuracies) + 1), accuracies, label=f"{impurity\_measure['criterion']}")

plt.title('Accuracy Across Folds for Different Impurity Measures')

plt.xlabel('Fold')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Calculate the mean accuracy for each impurity measure (across all folds).

mean\_accuracies = {impurity\_measure['criterion']: np.mean(

[result['accuracy'] for result in accuracy\_values if result['impurity'] == impurity\_measure['criterion']]

) for impurity\_measure in impurity\_measures}

# Print the mean accuracy for each impurity measure.

print("Mean Accuracy for Gini: {:.4f}".format(mean\_accuracies['gini']))

print("Mean Accuracy for Entropy: {:.4f}".format(mean\_accuracies['entropy']))

# Decide which is the best impurity measure based on mean accuracy.

best\_impurity = max(mean\_accuracies, key=mean\_accuracies.get)

print("The preferred impurity measure is {}".format(best\_impurity))

**References**

Link to the Dataset: <https://www.kaggle.com/datasets/ashishkumarjayswal/diabetes-dataset>