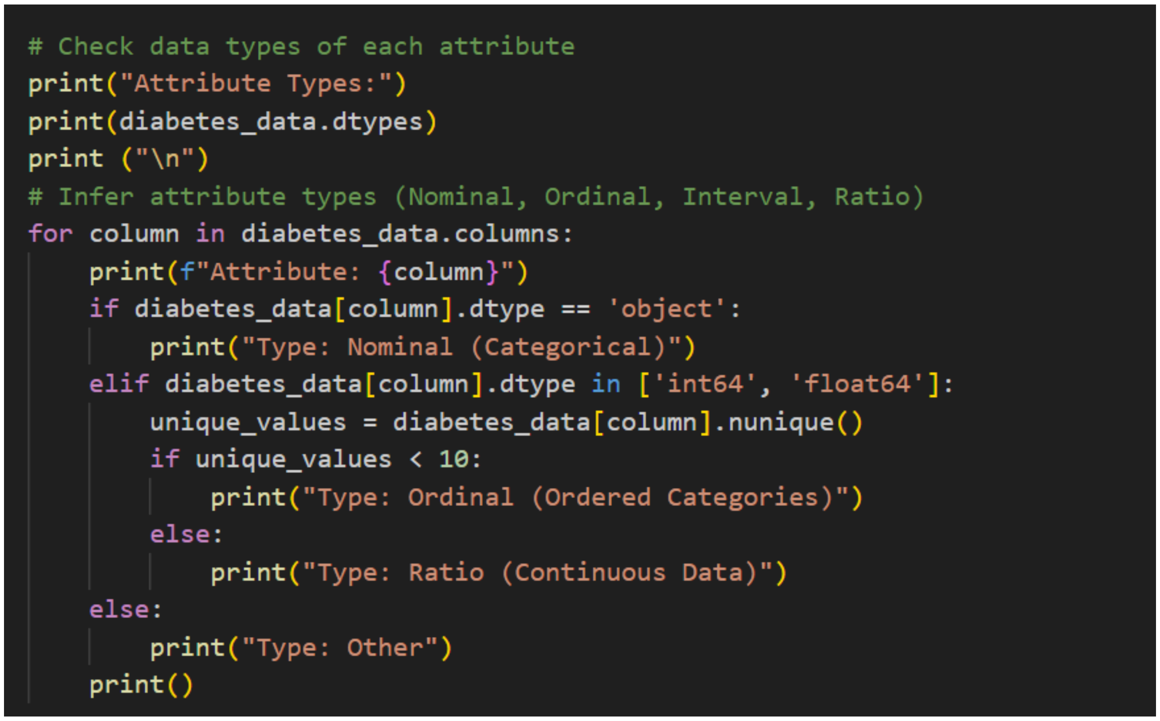
BA report

Objective



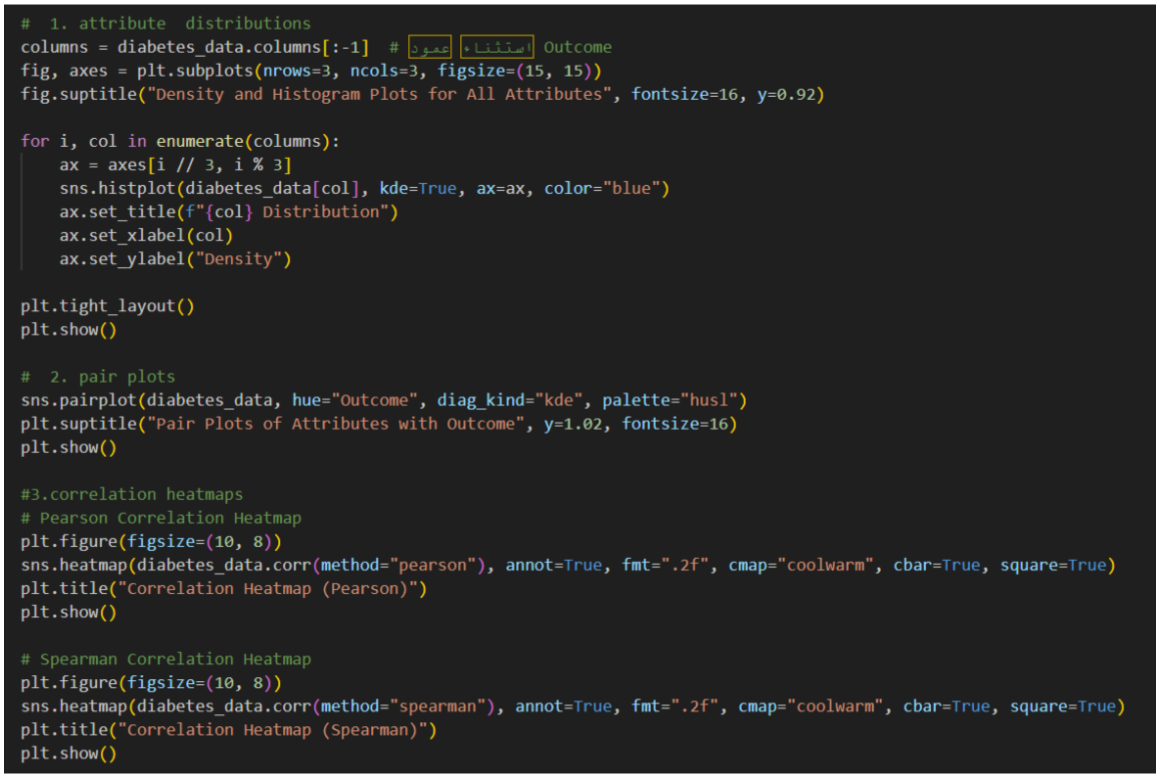
```python
# Check data types of each attribute
print("Attribute Types:")
print(diabetes_data.dtypes)
print ("\n")
# Infer attribute types (Nominal, Ordinal, Interval, Ratio)
for column in diabetes_data.columns:
    print(f"Attribute: {column}")
    if diabetes_data[column].dtype == 'object':
        print("Type: Nominal (Categorical)")
    elif diabetes_data[column].dtype in ['int64', 'float64']:
        unique_values = diabetes_data[column].nunique()
        if unique_values < 10:
            print("Type: Ordinal (Ordered Categories)")
        else:
            print("Type: Ratio (Continuous Data)")
    else:
        print("Type: Other")
    print()
```

The provided Python script focuses on analyzing and preprocessing two datasets: a diabetes dataset and a market dataset. The goals include attribute analysis, visualization, handling missing data, feature engineering, and implementing association rule mining for the market dataset.

Section A: Diabetes Data Analysis:

1. **Attribute Types Identification**
   - The script identifies the types of attributes (Nominal, Ordinal, Ratio) in the diabetes dataset based on the data types and unique values of columns.
   - Outputs the attribute types along with their inferred nature (categorical or continuous).

2. **Statistical Summaries :**

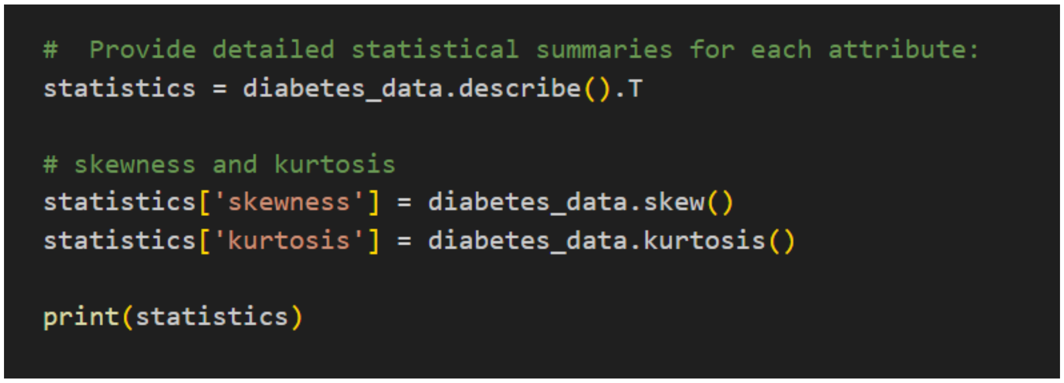


```python
#  1. attribute  distributions
columns = diabetes_data.columns[:-1]  # عمود استثناء Outcome
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
fig.suptitle("Density and Histogram Plots for All Attributes", fontsize=16, y=0.92)

for i, col in enumerate(columns):
    ax = axes[i // 3, i % 3]
    sns.histplot(diabetes_data[col], kde=True, ax=ax, color="blue")
    ax.set_title(f"{col} Distribution")
    ax.set_xlabel(col)
    ax.set_ylabel("Density")

plt.tight_layout()
plt.show()

#  2. pair plots
sns.pairplot(diabetes_data, hue="Outcome", diag_kind="kde", palette="husl")
plt.suptitle("Pair Plots of Attributes with Outcome", y=1.02, fontsize=16)
plt.show()

#3.correlation heatmaps
# Pearson Correlation Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(diabetes_data.corr(method="pearson"), annot=True, fmt=".2f", cmap="coolwarm", cbar=True, square=True)
plt.title("Correlation Heatmap (Pearson)")
plt.show()

# Spearman Correlation Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(diabetes_data.corr(method="spearman"), annot=True, fmt=".2f", cmap="coolwarm", cbar=True, square=True)
plt.title("Correlation Heatmap (Spearman)")
plt.show()
```

- Generates detailed statistics (mean, median, skewness, kurtosis) for each attribute to provide an overview of data distributions.

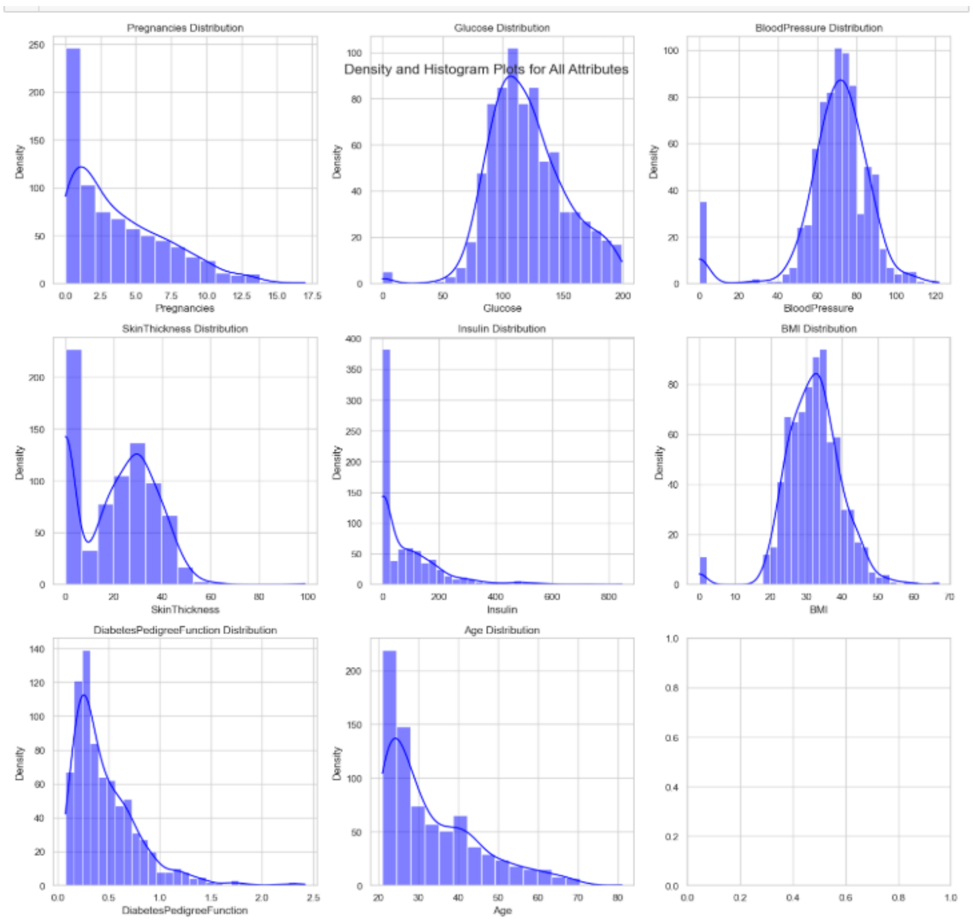
3. Visualizations :

- **Attribute Distributions:** Histograms and density plots for continuous variables.

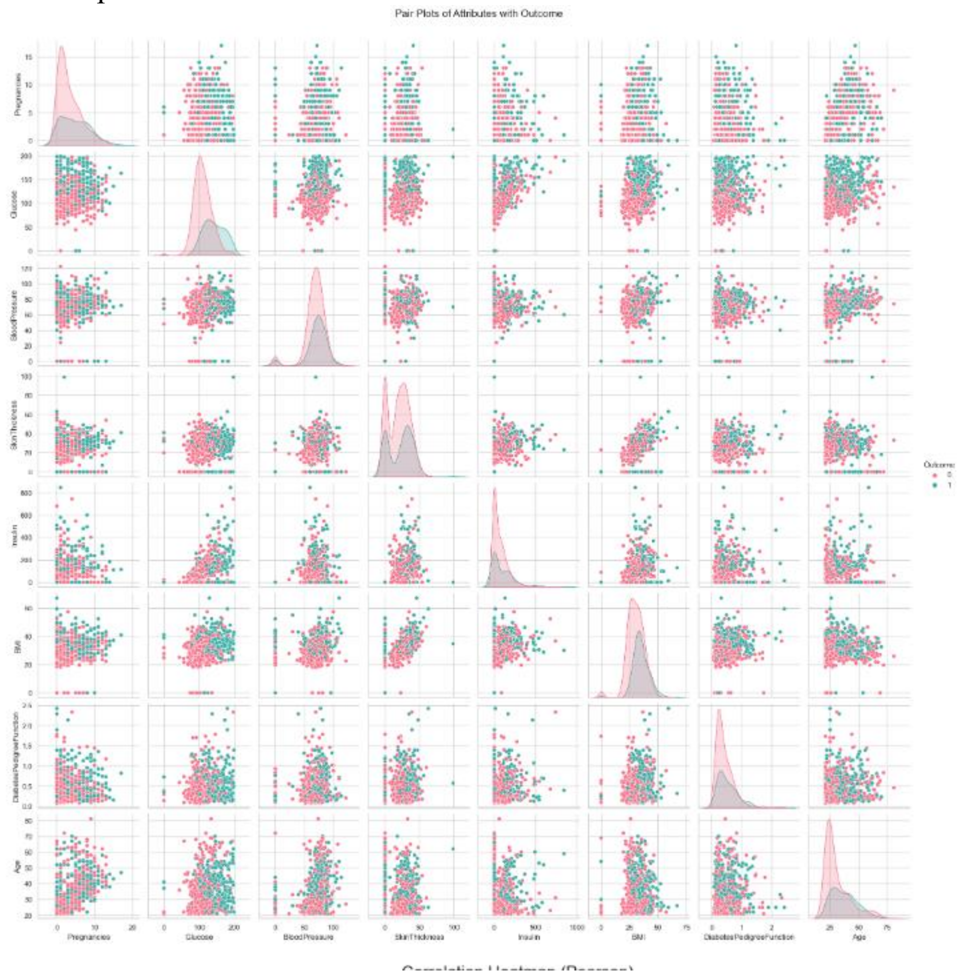


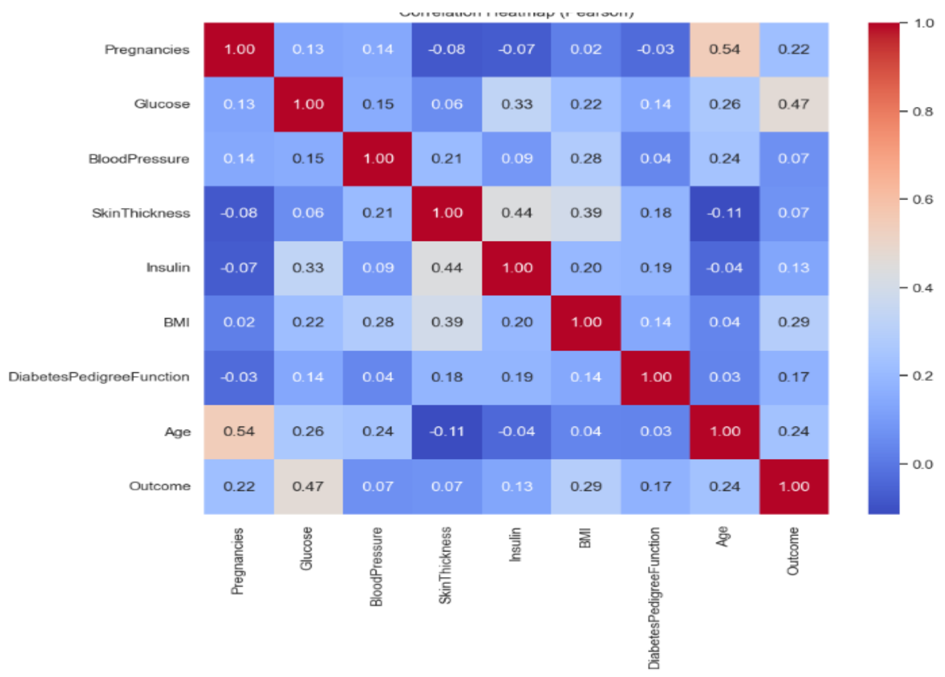
```python
#  Provide detailed statistical summaries for each attribute:
statistics = diabetes_data.describe().T

# skewness and kurtosis
statistics['skewness'] = diabetes_data.skew()
statistics['kurtosis'] = diabetes_data.kurtosis()

print(statistics)
```

- **Pair Plots:** Visualizing pairwise relationships among attributes colored by the diabetes outcome.



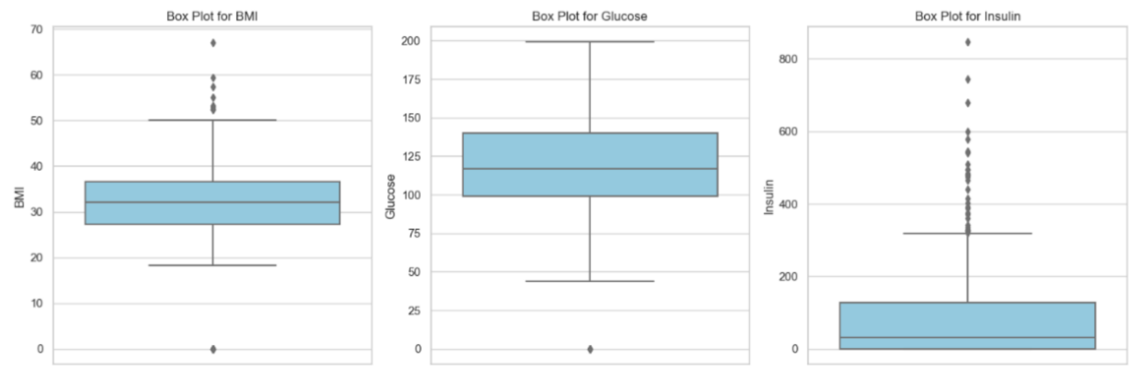
- **Correlation Heatmaps:** Pearson and Spearman correlation matrices for feature relationships.





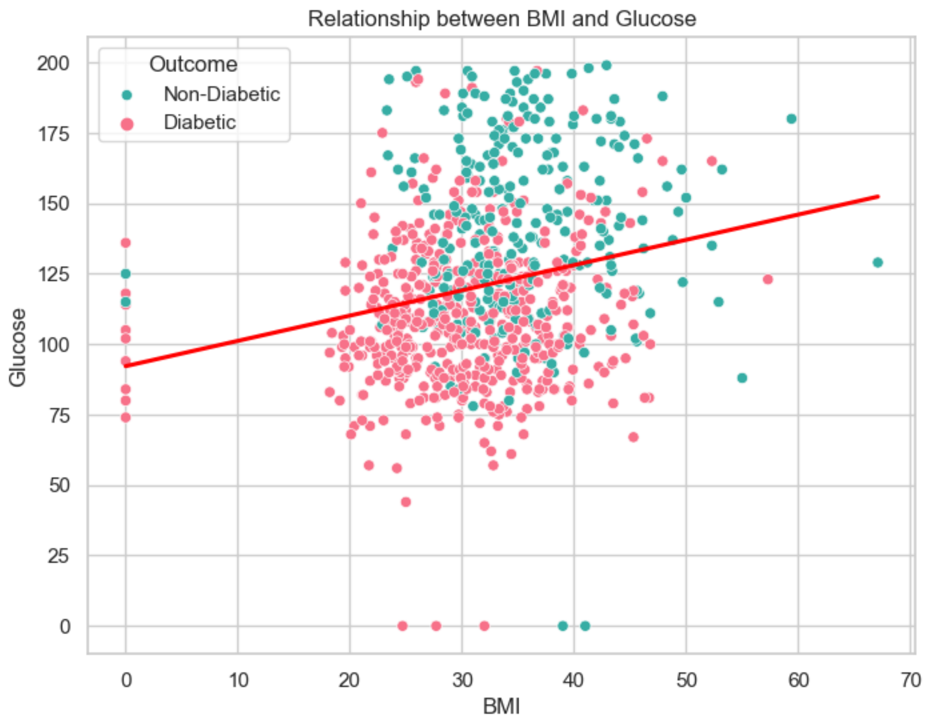
4. **Outlier Analysis :**

- Identifies outliers in specific features (BMI, Glucose, Insulin) using Z-scores and visualized via boxplots.



```
Outliers in BMI: [  9  49  60  81 145 177 371 426 445 494 522 673 684 706]
Outliers in Glucose: [ 75 182 342 349 502]
Outliers in Insulin: [  8  13 111 153 186 220 228 247 286 370 409 415 486 584 645 655 695 753]
```

- Explores relationships between BMI, Glucose, and the diabetes outcome using scatter plots and statistical comparisons.

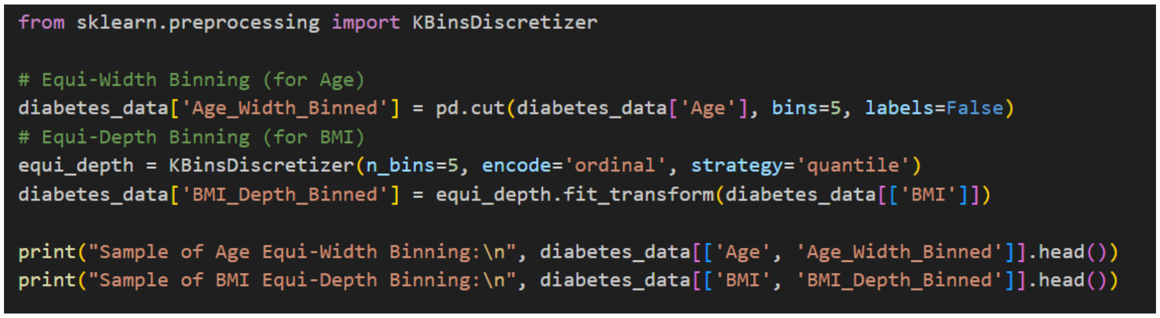


```
Average BMI for Diabetic: 35.14
Average BMI for Non-Diabetic: 30.30
Average Glucose for Diabetic: 141.26
Average Glucose for Non-Diabetic: 109.98
```

Section B: Data Preprocessing Techniques

1. Binning :

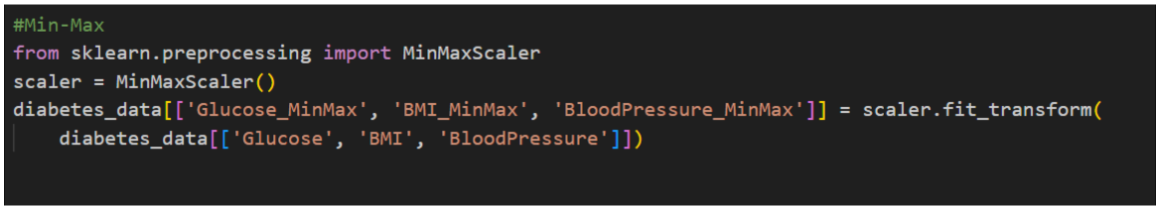
- Implements equi-width and equi-depth binning for features like Age and BMI.
- Creates categorical age groups based on predefined bins.



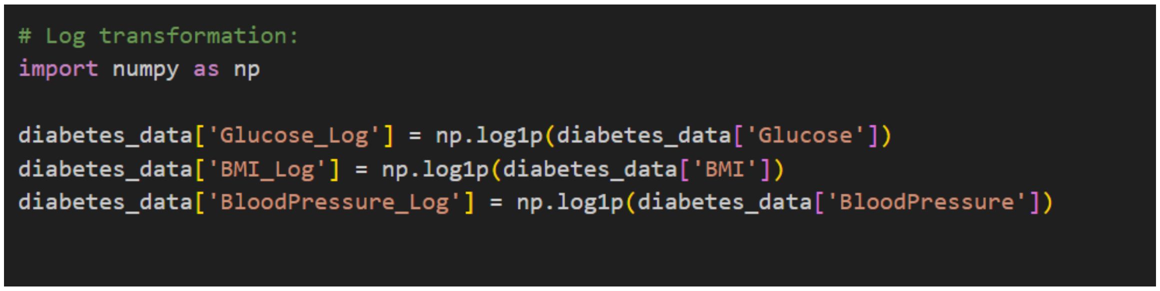
```
from sklearn.preprocessing import KBinsDiscretizer

# Equi-Width Binning (for Age)
diabetes_data['Age_Width_Binned'] = pd.cut(diabetes_data['Age'], bins=5, labels=False)
# Equi-Depth Binning (for BMI)
equi_depth = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
diabetes_data['BMI_Depth_Binned'] = equi_depth.fit_transform(diabetes_data[['BMI']])

print("Sample of Age Equi-Width Binning:\n", diabetes_data[['Age', 'Age_Width_Binned']].head())
print("Sample of BMI Equi-Depth Binning:\n", diabetes_data[['BMI', 'BMI_Depth_Binned']].head())
```

2. Normalization :

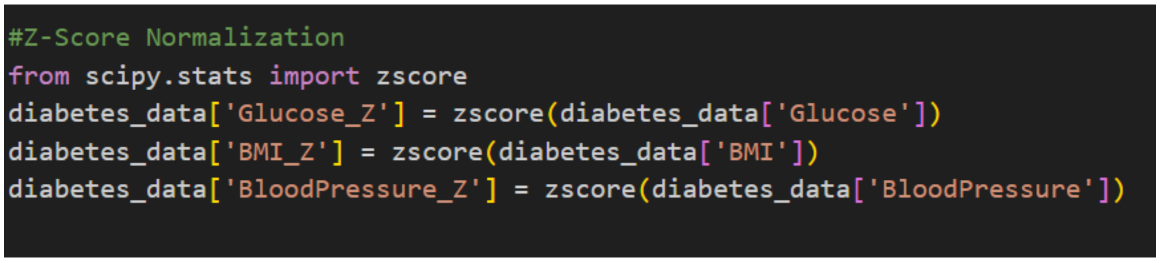
- Applies Min-Max Scaling, Log Transformation, and Z-Score normalization to continuous variables.



```
#Min-Max
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
diabetes_data[['Glucose_MinMax', 'BMI_MinMax', 'BloodPressure_MinMax']] = scaler.fit_transform(
    diabetes_data[['Glucose', 'BMI', 'BloodPressure']])
```

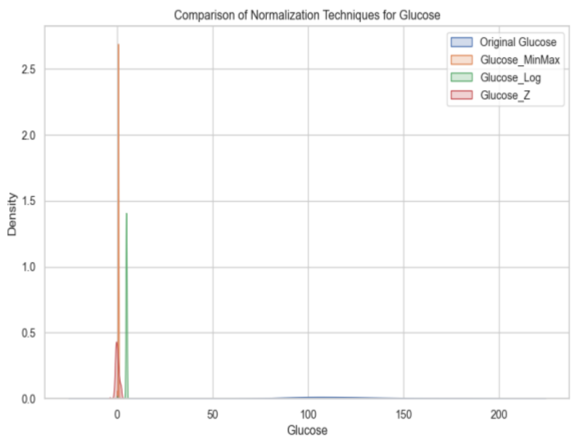


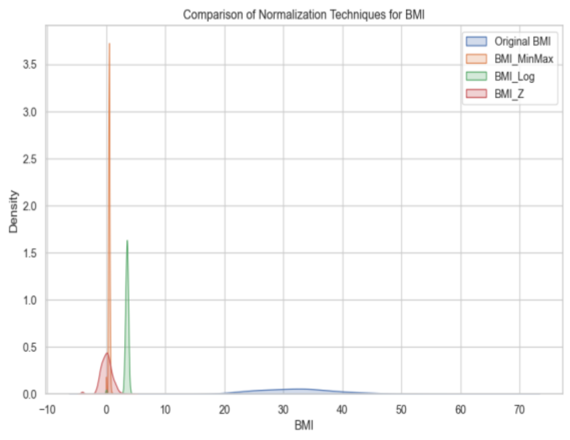
```
# Log transformation:
import numpy as np

diabetes_data['Glucose_Log'] = np.log1p(diabetes_data['Glucose'])
diabetes_data['BMI_Log'] = np.log1p(diabetes_data['BMI'])
diabetes_data['BloodPressure_Log'] = np.log1p(diabetes_data['BloodPressure'])
```

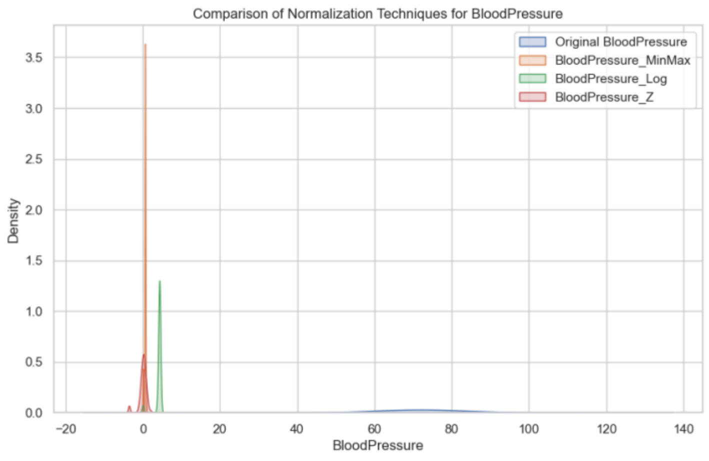


```
#Z-Score Normalization
from scipy.stats import zscore
diabetes_data['Glucose_Z'] = zscore(diabetes_data['Glucose'])
diabetes_data['BMI_Z'] = zscore(diabetes_data['BMI'])
diabetes_data['BloodPressure_Z'] = zscore(diabetes_data['BloodPressure'])
```

- Compares the distributions before and after normalization using density plots.

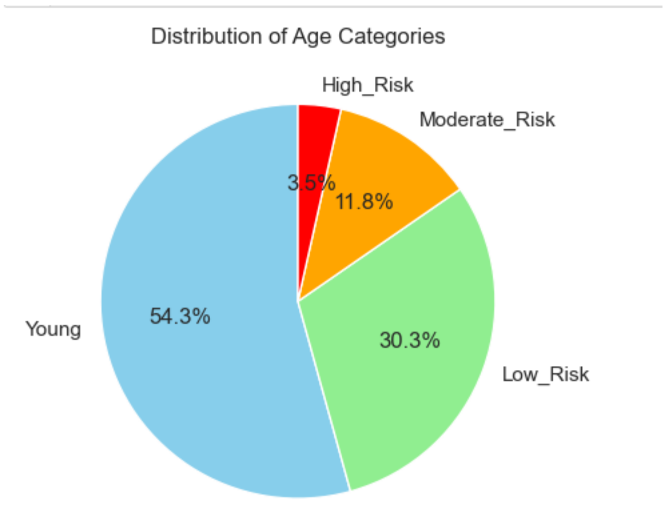






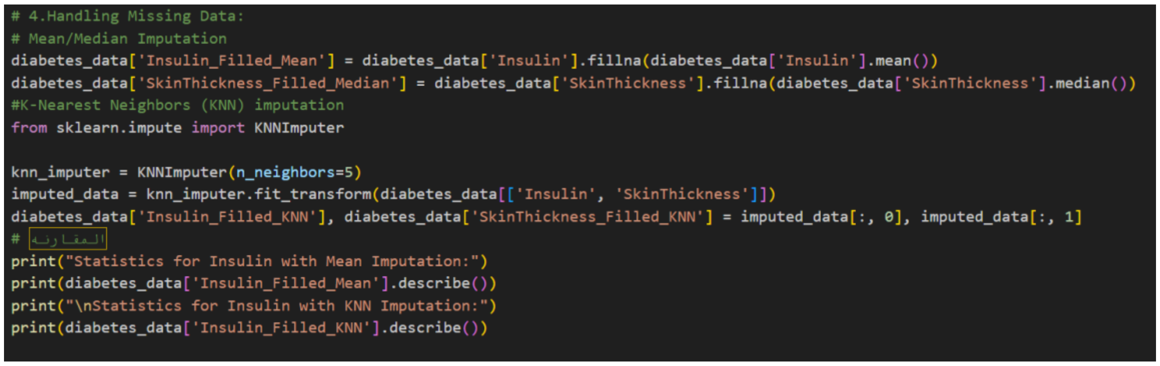
3. Discretization of Age:

- Where Young (18–30)
- Low_Risk (31–45)
- Moderate_Risk (46–60),
- High_Risk (61+).



4. Handling Missing Data in Diabetes Dataset :

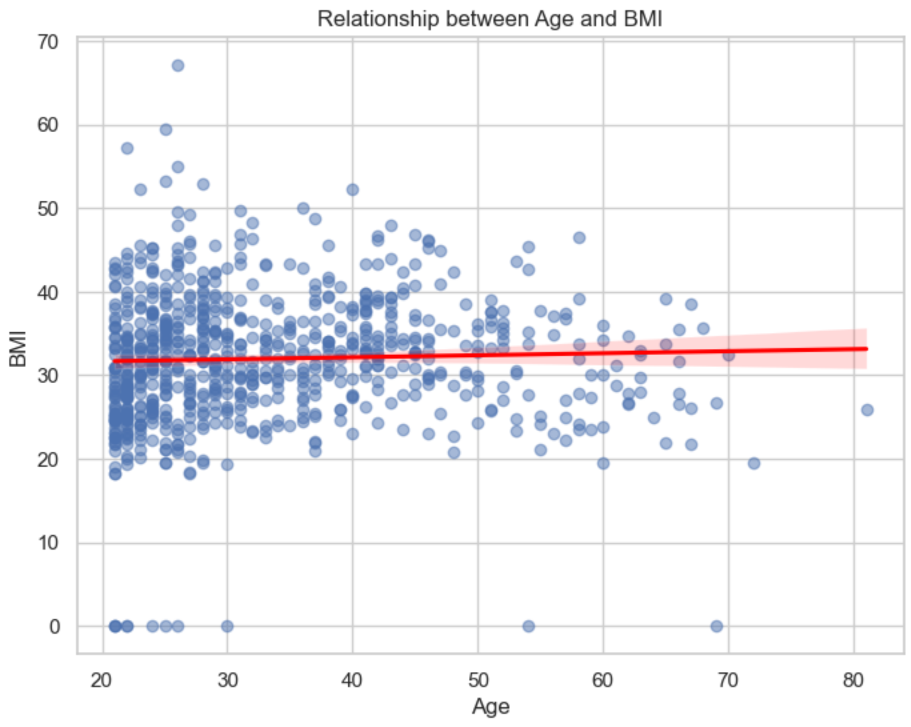
The code addresses missing data in two features, **Insulin** and **SkinThickness**, by applying different imputation techniques and comparing the results. The aim is to fill the missing values using methods that maintain the integrity of the dataset.



```python
# 4.Handling Missing Data:
# Mean/Median Imputation
diabetes_data['Insulin_Filled_Mean'] = diabetes_data['Insulin'].fillna(diabetes_data['Insulin'].mean())
diabetes_data['SkinThickness_Filled_Median'] = diabetes_data['SkinThickness'].fillna(diabetes_data['SkinThickness'].median())
#K-Nearest Neighbors (KNN) imputation
from sklearn.impute import KNNImputer

knn_imputer = KNNImputer(n_neighbors=5)
imputed_data = knn_imputer.fit_transform(diabetes_data[['Insulin', 'SkinThickness']])
diabetes_data['Insulin_Filled_KNN'], diabetes_data['SkinThickness_Filled_KNN'] = imputed_data[:, 0], imputed_data[:, 1]
# المقارنه
print("Statistics for Insulin with Mean Imputation:")
print(diabetes_data['Insulin_Filled_Mean'].describe())
print("\nStatistics for Insulin with KNN Imputation:")
print(diabetes_data['Insulin_Filled_KNN'].describe())
```

5. Feature Engineering :

- Explores the correlation between BMI and Age and visualizes it using regression plots.



ules based on a minimum confidence threshold.