BOOSTING ALGORITHMS

To diagnostically predict whether a patient has diabetes

ENSEMBLE LEARNING

Ensemble learning combines several base algorithms to form one optimised predictive algorithm.

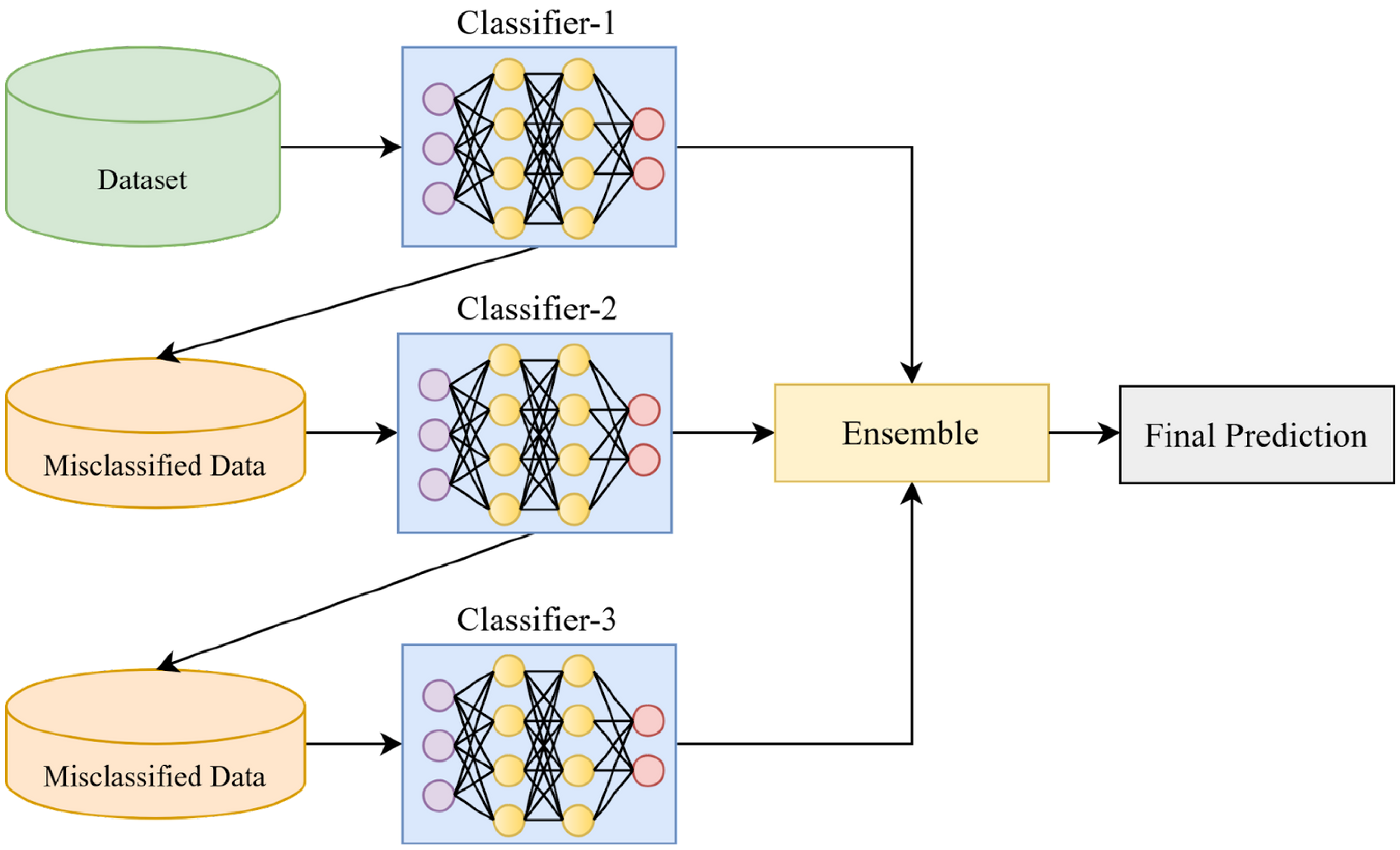
Can be divided into two groups:

- Sequential Learners (AdaBoost)
- Parallel Learners (Random Forest)

Can be used for:

- Decrease Variance (Bagging)
- Decrease Bias (Boosting)
- Improve Predictions (Stacking)

SEQUENTIAL LEARNERS



Types of Boosting Algorithms:

- Adaboost (Adaptive Boosting
- Gradient Tree Boosting
- XGBoost

BOOSTING

Boosting algorithm tries to build a strong learner (predictive model) from the mistakes of several weak learners.

1

2

Step 1:
Assign initial weights

Step 2:
Create a decision stump for each variable

Step 3:
Assign more weights to incorrectly classified samples

Step 4:
Iterate from Step 2 until max iteration level has been reached

3

4

PSEUDO-CODE

```
Initially set uniform example weights.

for Each base learner do:
Train base learner with a weighted sample.
Test base learner on all data.
Set learner weight with a weighted error.
Set example weights based on ensemble predictions.
end for
```

CASE STUDY

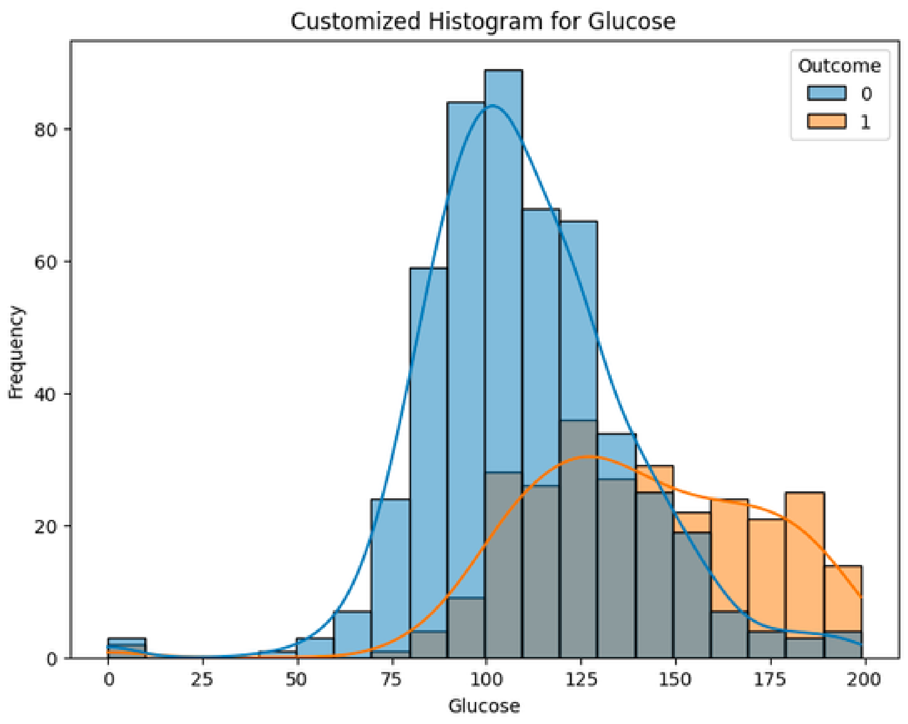
To diagnostically predict whether a patient has diabetes

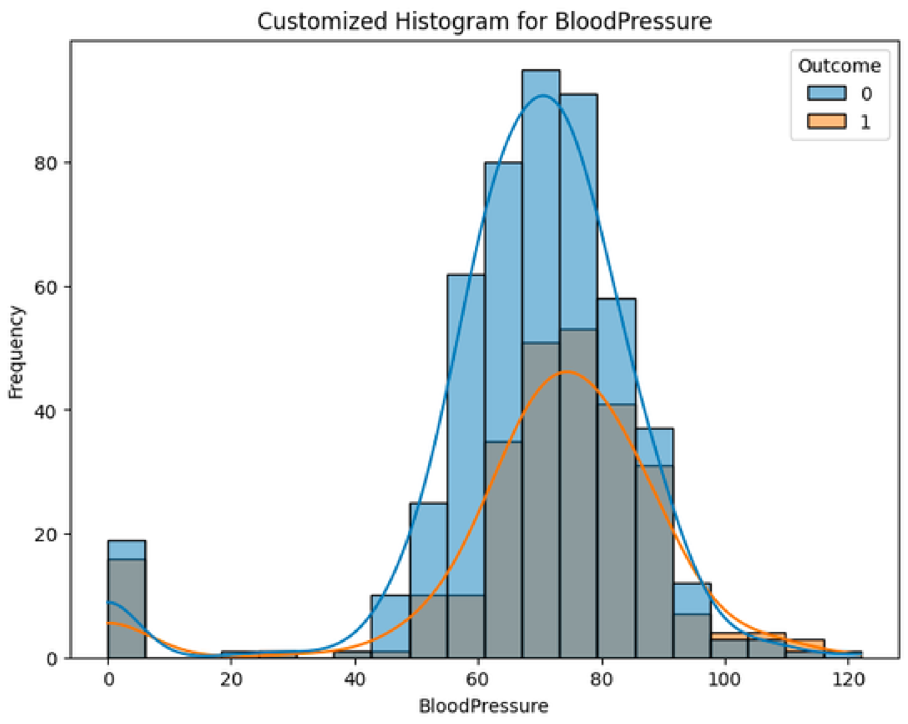
DATASET

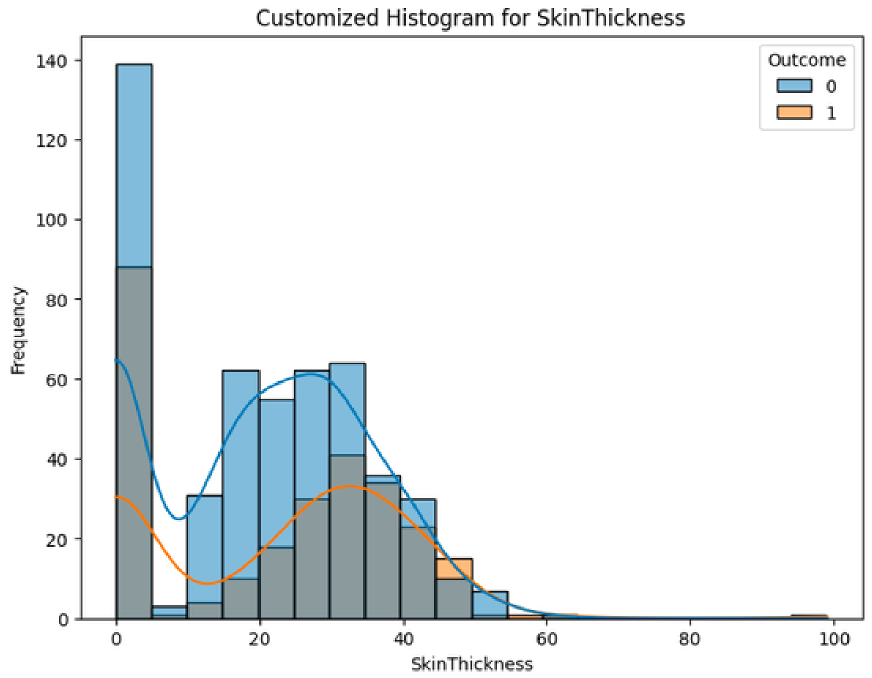
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

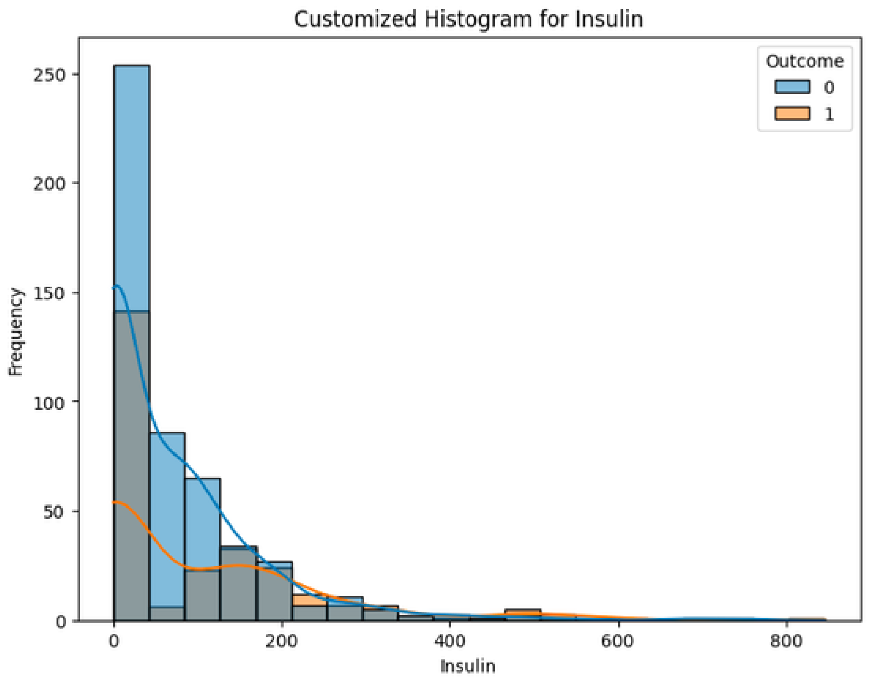
768 rows × 9 columns

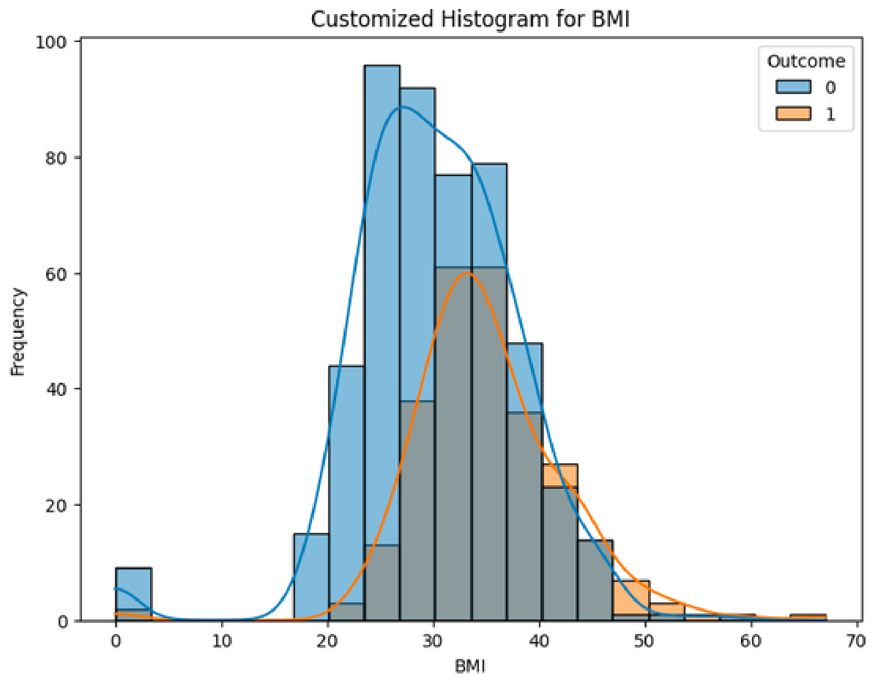
HISTOGRAMS

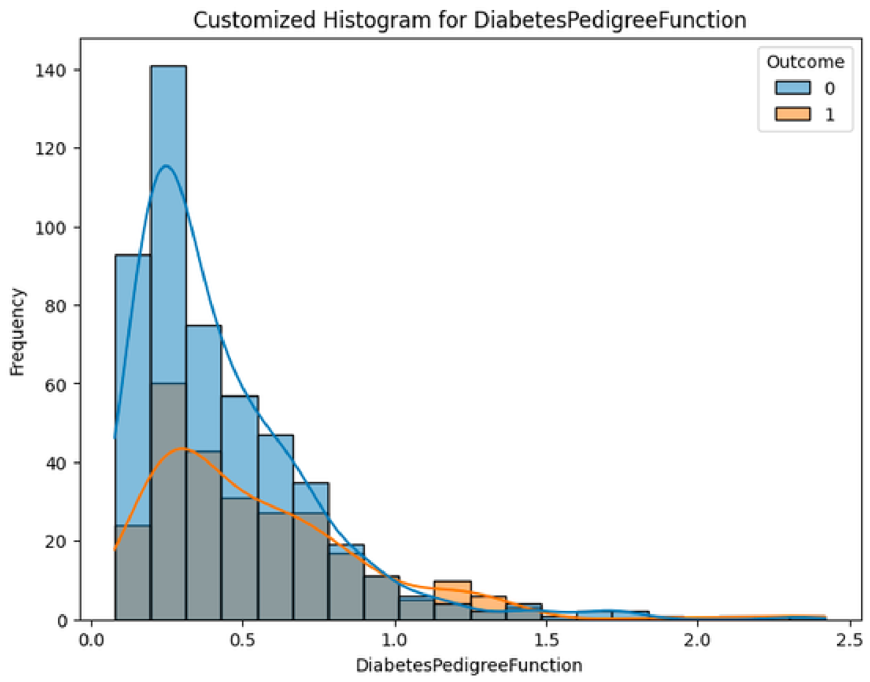

Customized Histogram for Glucose
Outcome
0
1
Frequency
Glucose


Customized Histogram for BloodPressure
Outcome
0
1
Frequency
BloodPressure

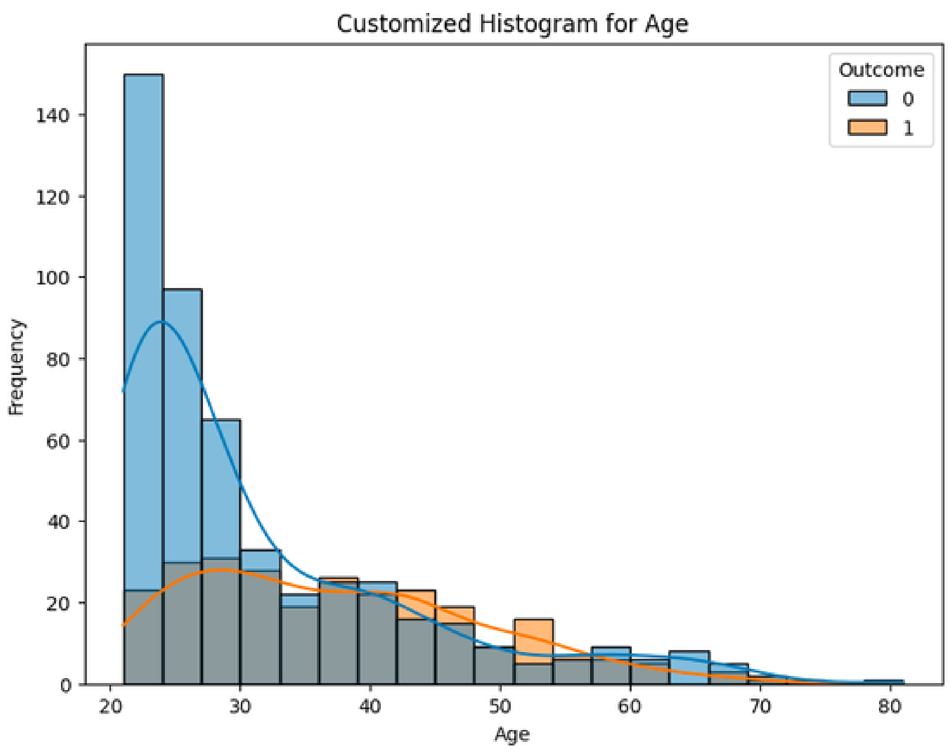

Customized Histogram for SkinThickness
Outcome
0
1
Frequency
SkinThickness


Customized Histogram for Insulin
Outcome
0
1
Frequency
Insulin

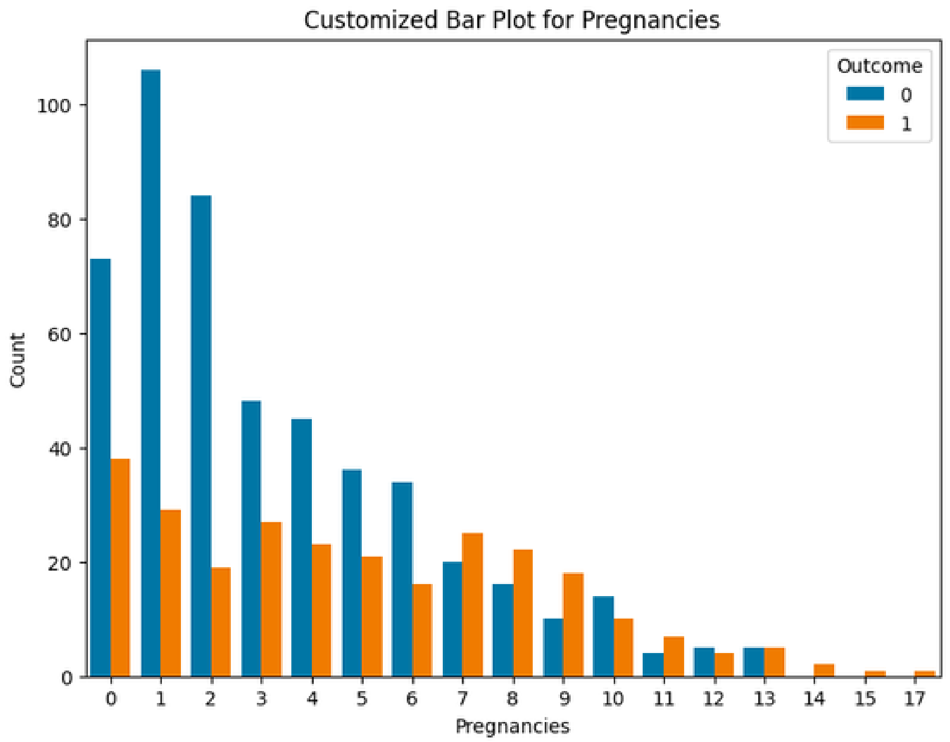

Customized Histogram for BMI
Outcome
0
1
Frequency
BMI


Customized Histogram for DiabetesPedigreeFunction
Outcome
0
1
Frequency
DiabetesPedigreeFunction

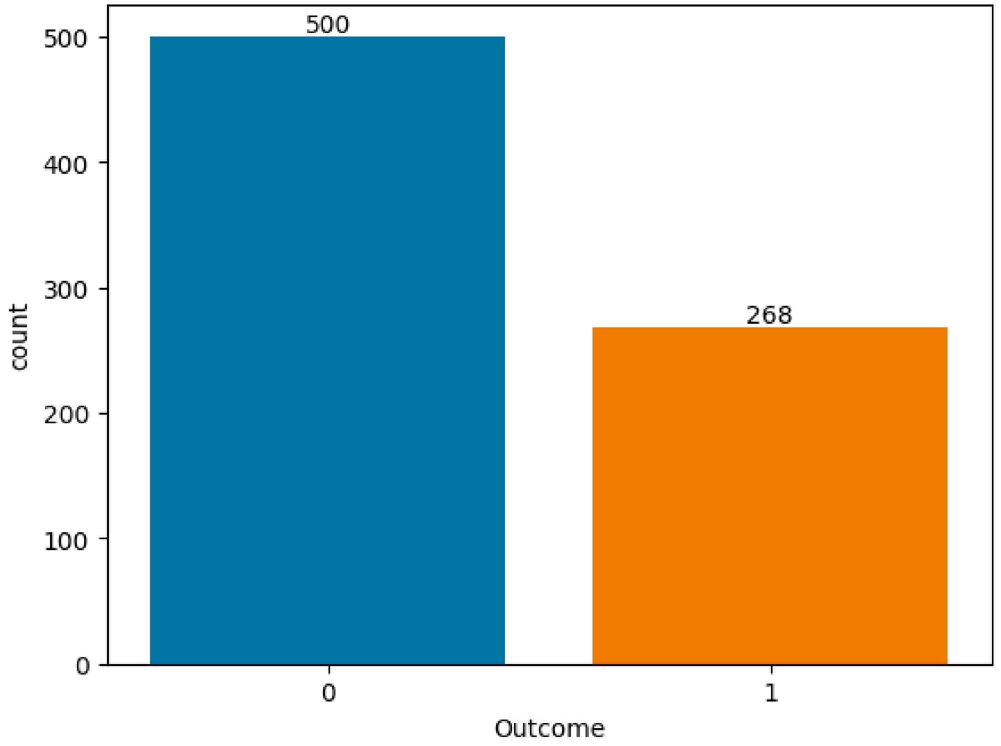
HISTOGRAMS



Customized Histogram for Age
Outcome
0
1
Frequency
0
20
40
60
80
100
120
140
20
30
40
50
60
70
80
Age



Customized Bar Plot for Pregnancies
Outcome
0
1
Count
0
20
40
60
80
100
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17
Pregnancies



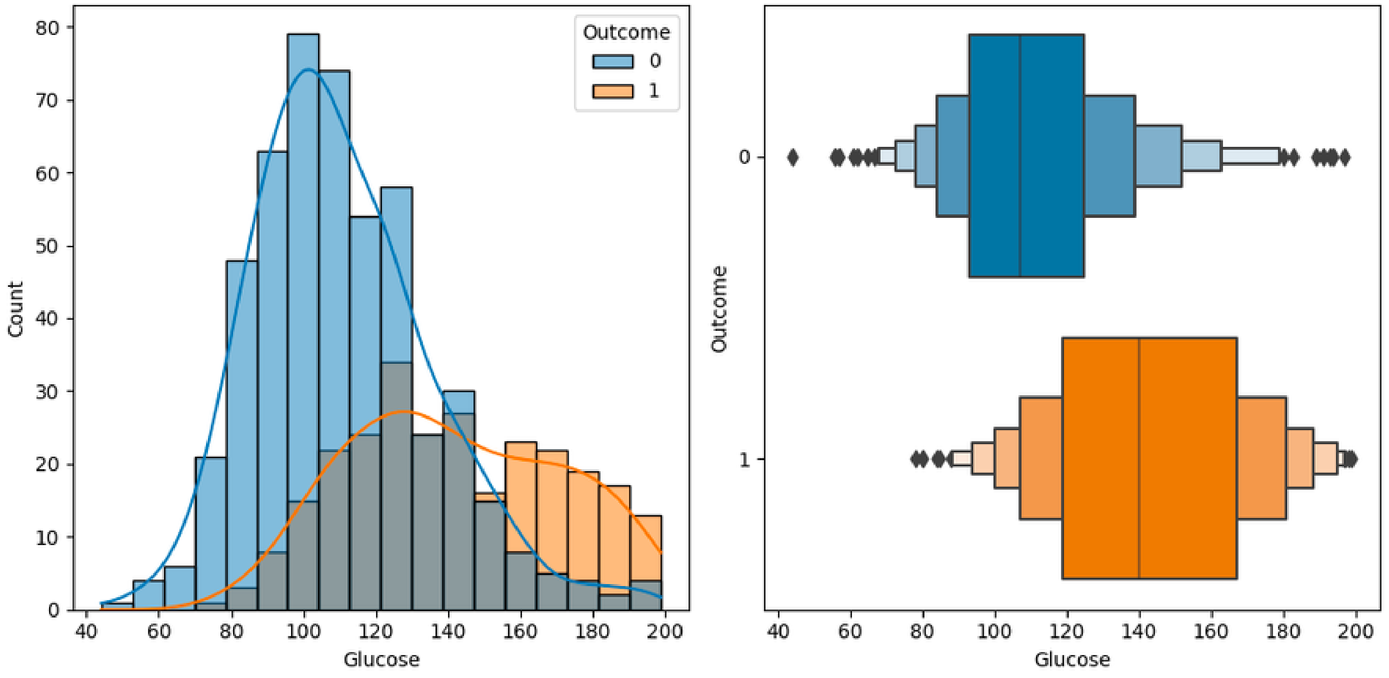
500
268
count
0
100
200
300
400
500
0
1
Outcome

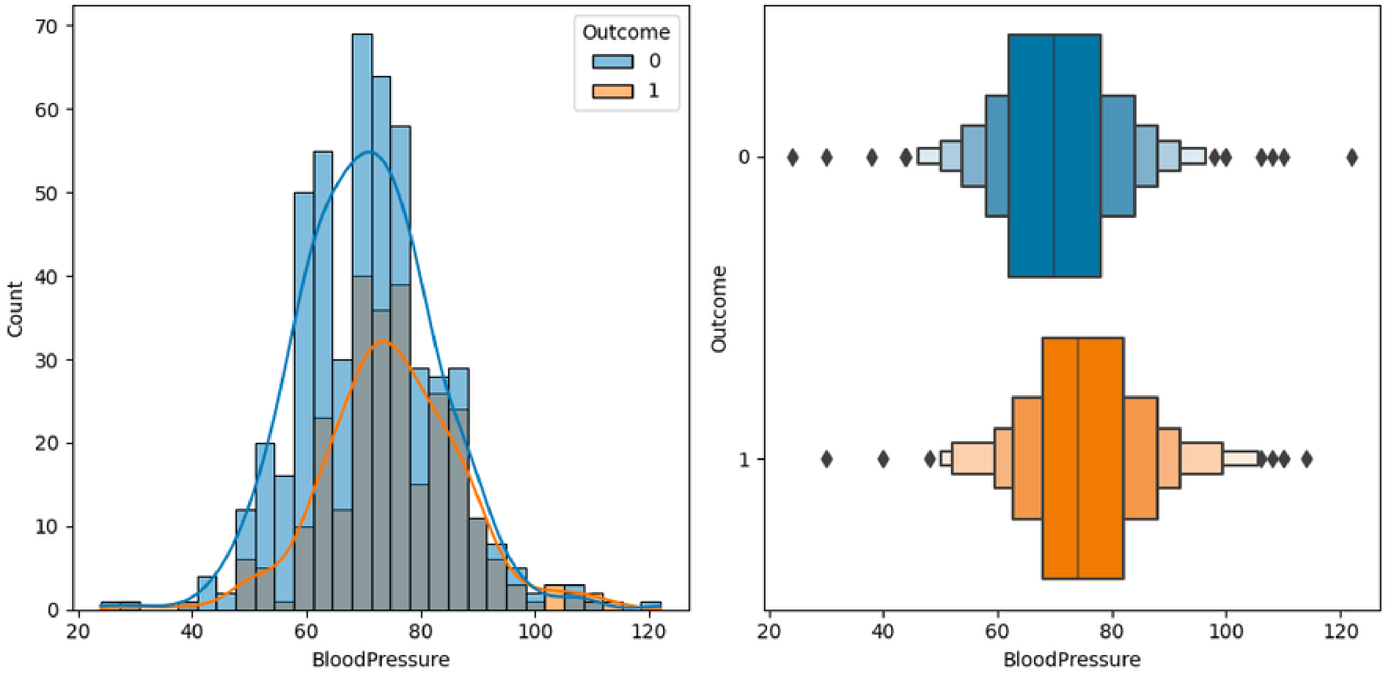
REPLACE INCORRECT DATA

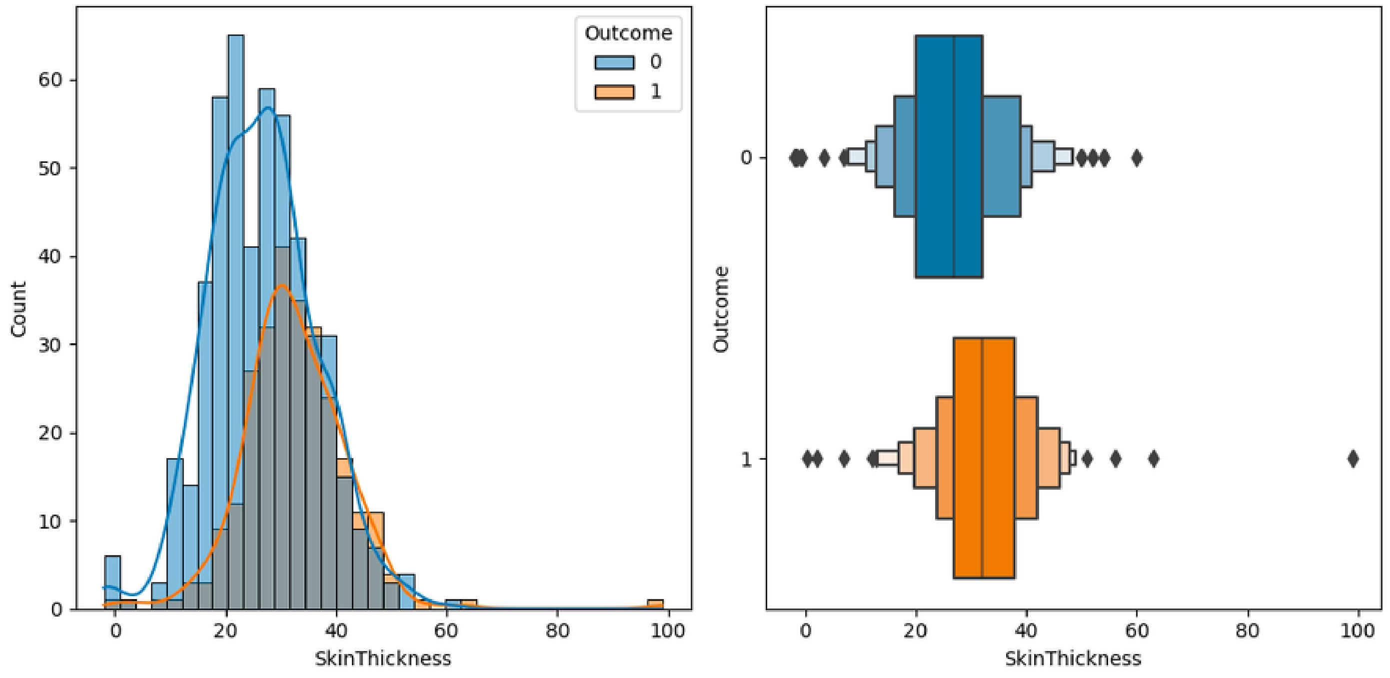
```python
def predict_with_regression(data, target_variable, predictor_variables):
    # Split the dataset into two parts: one with non-zero values and one with zero values
    training_data = data[data[target_variable] != 0]
    prediction_data = data[data[target_variable] == 0]

    # Separate the training data into features (X) and the target variable (y)
    X_train = training_data[predictor_variables]
    y_train = training_data[target_variable]

    # Create a Linear Regression model and fit it to the training data
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Use the trained model to predict values for the zero entries in the prediction set
    X_pred = prediction_data[predictor_variables]
    predicted_values = model.predict(X_pred)

    # Replace the zero values in the original dataset with the predicted values
    data.loc[data[target_variable] == 0, target_variable] = predicted_values

    return data
```

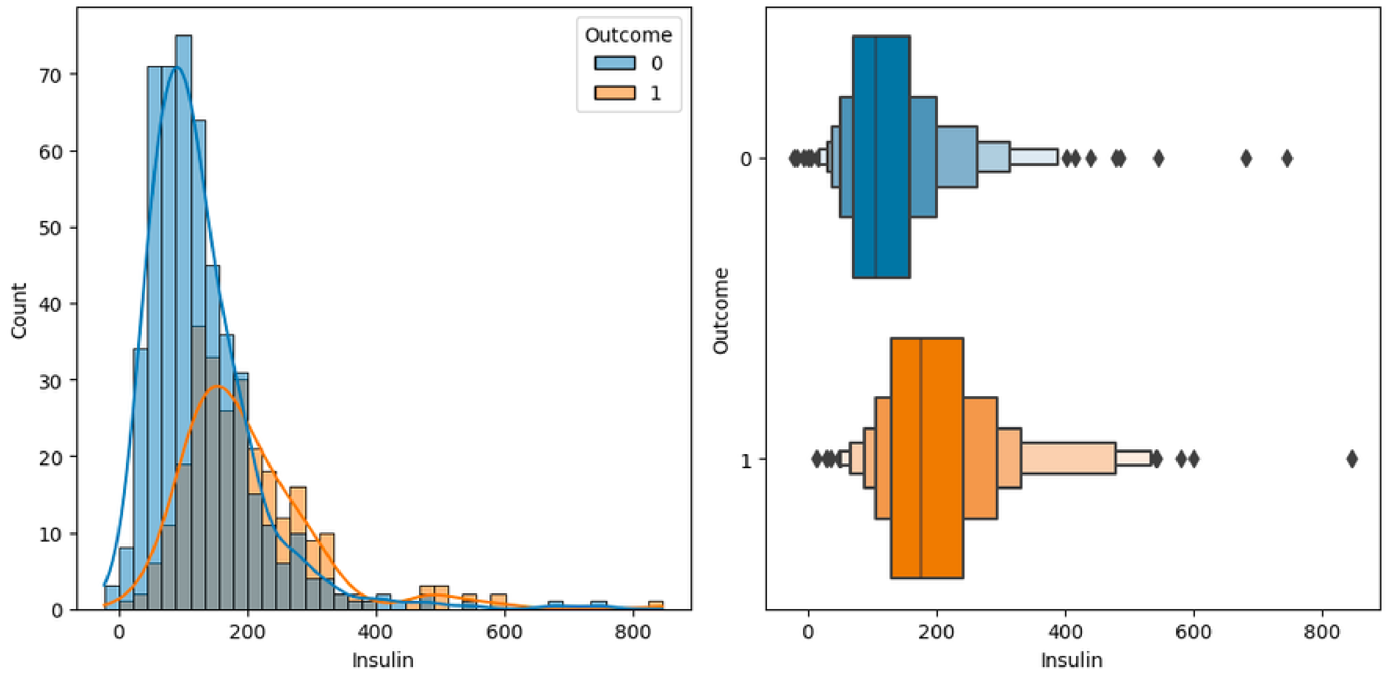
```python
df_1 = predict_with_regression(df_raw, 'Glucose', ['Pregnancies', 'BloodPressure', 'SkinThickness', 'Insulin',
        'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],)
```

DATA DISTRIBUTIONS AFTER TRANSFORMATIONS

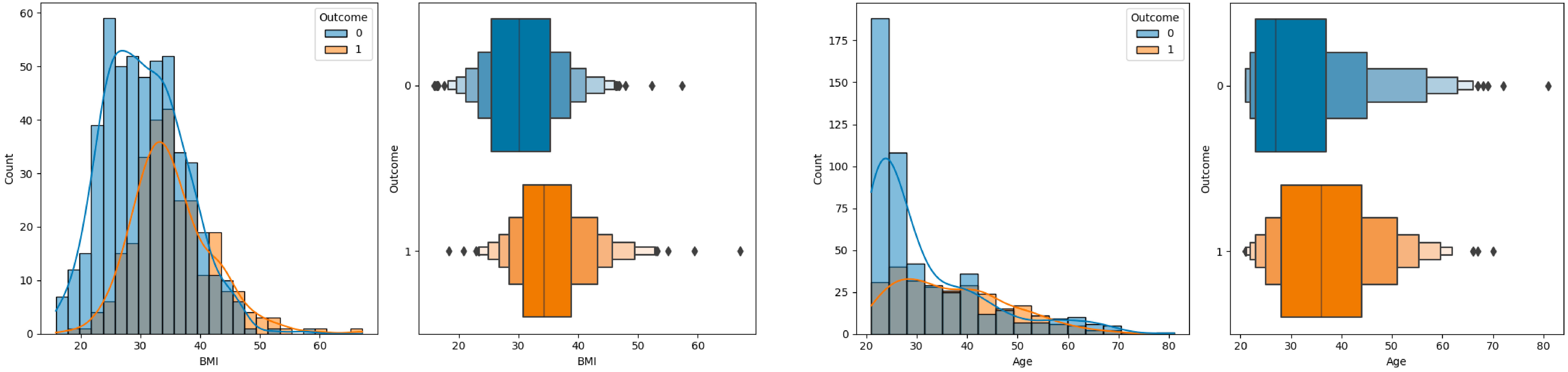








DATA DISTRIBUTIONS AFTER TRANSFORMATIONS



AdaBoost

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Split your data into features (X) and the target variable (y)
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Split your data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

# Create an AdaBoostClassifier
adaboost_classifier = AdaBoostClassifier(n_estimators=1000, random_state=123)

# Train the AdaBoost classifier on the training data
adaboost_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = adaboost_classifier.predict(X_test)

# accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(confusion)
```

```
Accuracy: 0.77
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.84      0.82        96
           1       0.71      0.64      0.67        58

    accuracy                           0.77       154
   macro avg       0.75      0.74      0.75       154
weighted avg       0.76      0.77      0.76       154


Confusion Matrix:
[[81 15]
 [21 37]]
```

AdaBoost
withGridSearchCV

```
from sklearn.model_selection import GridSearchCV

# Create an AdaBoostClassifier
adaboost_classifier = AdaBoostClassifier(n_estimators=1000, random_state=123)


# Define the hyperparameter grid for tuning
param_grid = {
    'n_estimators': [50, 100, 200, 500, 1000],
    'learning_rate': [0.01, 0.1, 0.5, 1.0, 1.5, 2, 2.5, 3.0, 4.0, 5.0]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=adaboost_classifier, param_grid=param_grid, scoring='accuracy', cv=7)

# Perform hyperparameter tuning on the training data
grid_search.fit(X_train, y_train)

# Get the best estimator from the grid search
best_classifier = grid_search.best_estimator_

# Make predictions on the testing data using the best classifier
y_pred = best_classifier.predict(X_test)

# classification report
report = classification_report(y_test, y_pred)

# confusion matrix
confusion = confusion_matrix(y_test, y_pred)

print("Classification Report:")
print(report)

print("\nConfusion Matrix:")
print(confusion)

# Print the best hyperparameters
print("\nBest Hyperparameters:")
print(grid_search.best_params_)
```

```
Accuracy: 0.75
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.82      0.80        96
           1       0.68      0.62      0.65        58

    accuracy                           0.75       154
   macro avg       0.73      0.72      0.73       154
weighted avg       0.74      0.75      0.74       154


Confusion Matrix:
[[79 17]
 [22 36]]

Best Hyperparameters:
{'learning_rate': 0.1, 'n_estimators': 500}
```

XGBoost

```
import xgboost as xgb

# Create an XGBoost classifier
xgb_classifier = xgb.XGBClassifier(random_state = 123)

# Train the XGBoost classifier on the training data
xgb_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = xgb_classifier.predict(X_test)

# accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(confusion)
```

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.81      0.83        96
           1       0.71      0.78      0.74        58

    accuracy                           0.80       154
   macro avg       0.79      0.79      0.79       154
weighted avg       0.80      0.80      0.80       154


Confusion Matrix:
[[78 18]
 [13 45]]
```

XGBoost
withGridSearchCV

```python
# Create an XGBoost classifier
xgb_classifier = xgb.XGBClassifier(random_state = 123)

# Define the hyperparameter grid for tuning
param_grid = {
    'n_estimators': [50, 100, 200, 500, 1000],
    'learning_rate': [0.01, 0.1, 0.5, 1.0, 1.5, 2, 2.5, 3.0, 4.0, 5.0],
    'max_depth': [3, 4, 5, 7, 10, 15]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid, scoring='accuracy', cv=7)

# Perform hyperparameter tuning on the training data
grid_search.fit(X_train, y_train)

# Get the best estimator from the grid search
best_classifier = grid_search.best_estimator_

# Make predictions on the testing data using the best classifier
y_pred = best_classifier.predict(X_test)

# accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(confusion)

# Print the best hyperparameters
print("\nBest Hyperparameters:")
print(grid_search.best_params_)
```

Accuracy: 0.81

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.85	0.85	96
1	0.75	0.72	0.74	58
accuracy			0.81	154
macro avg	0.79	0.79	0.79	154
weighted avg	0.80	0.81	0.80	154

Confusion Matrix:

```
[[82 14]
 [16 42]]
```

Best Hyperparameters:

{'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 500}

COMPARISON

Boosting Method

AdaBoost

Adaptive Boosting

Accuracy

75%

Confustion Matrix

TP	FP
79	17
22	36
FN	TN

XGBoost

Extreme Gradient Boosting

Accuracy

81%

Confustion Matrix

TP	FP
82	14
16	42
FN	TN

SUMMARY

ADABOOST

- Sensitive to noisy data and outliers
- Less parallelizable for large datasets
- May require additional techniques to prevent overfitting

XGBOOST

- Robust and scalable
- Highly parallelizable
- Built-in regularization techniques to help with overfitting
- Usually outperforms AdaBoost

Further reading:

An improved AdaBoost algorithm for identification of lung cancer based on electronic nose.
Lijun Haoa, Gang Huang

https://www.sciencedirect.com/science/article/pii/S240584402300840X

THANK YOU