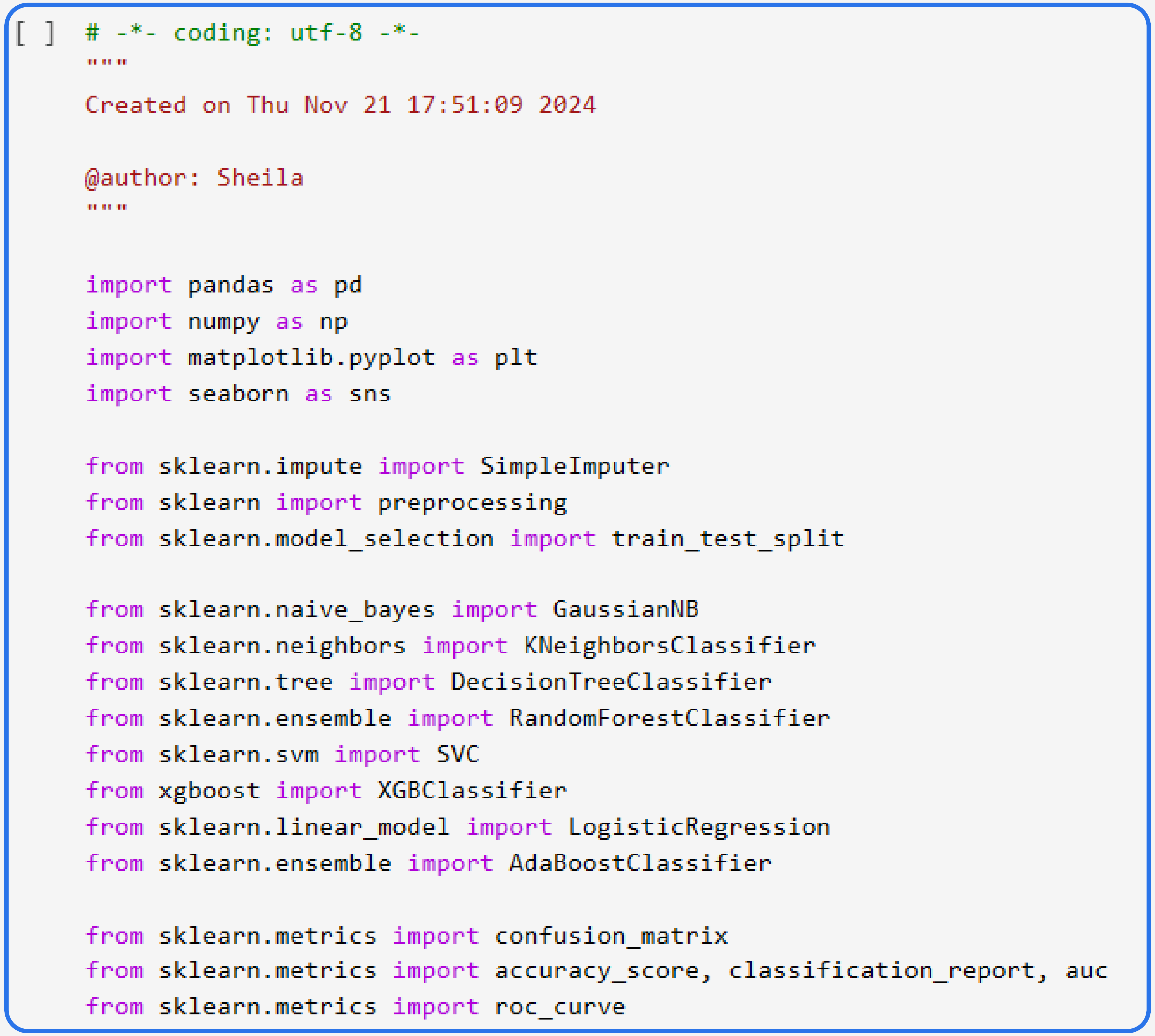
CMSC 128.2

Refactoring

Code

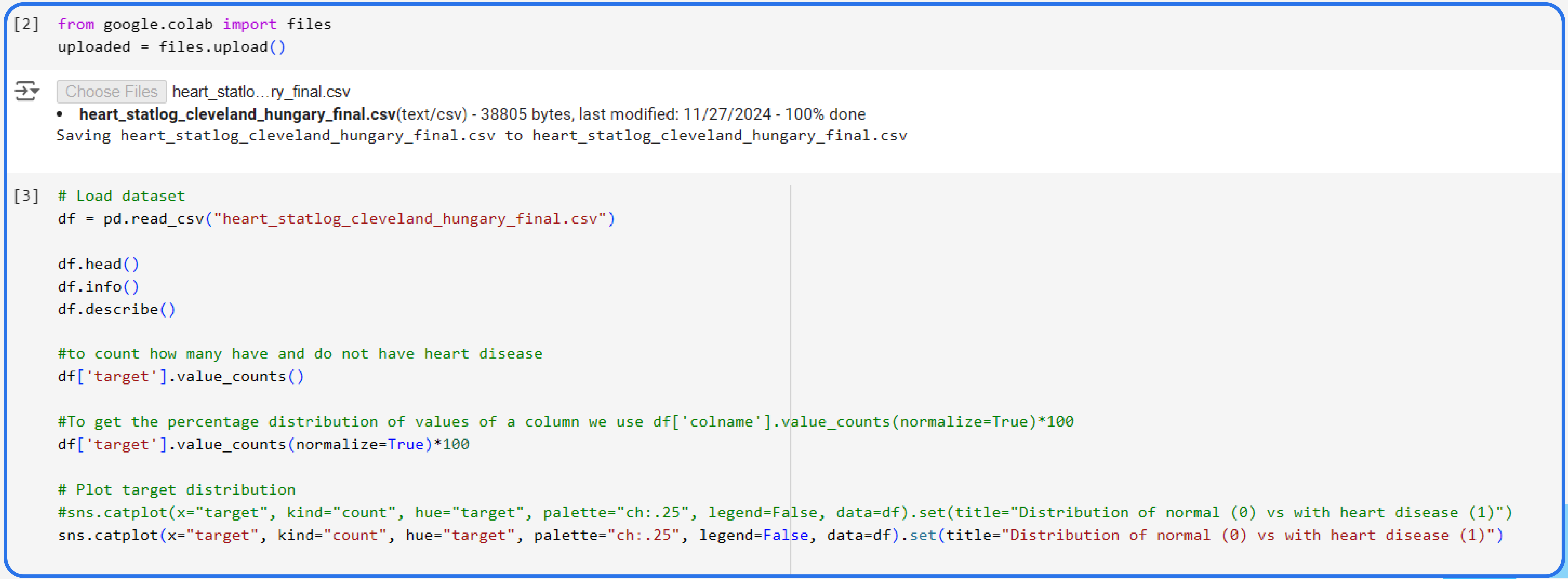
Antonino, Dorado, Layug, Santos, Trani

Setup & Imports



```
[ ]  # -*- coding: utf-8 -*-
     """
     Created on Thu Nov 21 17:51:09 2024

     @author: Sheila
     """


     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns

     from sklearn.impute import SimpleImputer
     from sklearn import preprocessing
     from sklearn.model_selection import train_test_split

     from sklearn.naive_bayes import GaussianNB
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.svm import SVC
     from xgboost import XGBClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.ensemble import AdaBoostClassifier

     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import accuracy_score, classification_report, auc
     from sklearn.metrics import roc_curve
```

Load and Explore Dataset



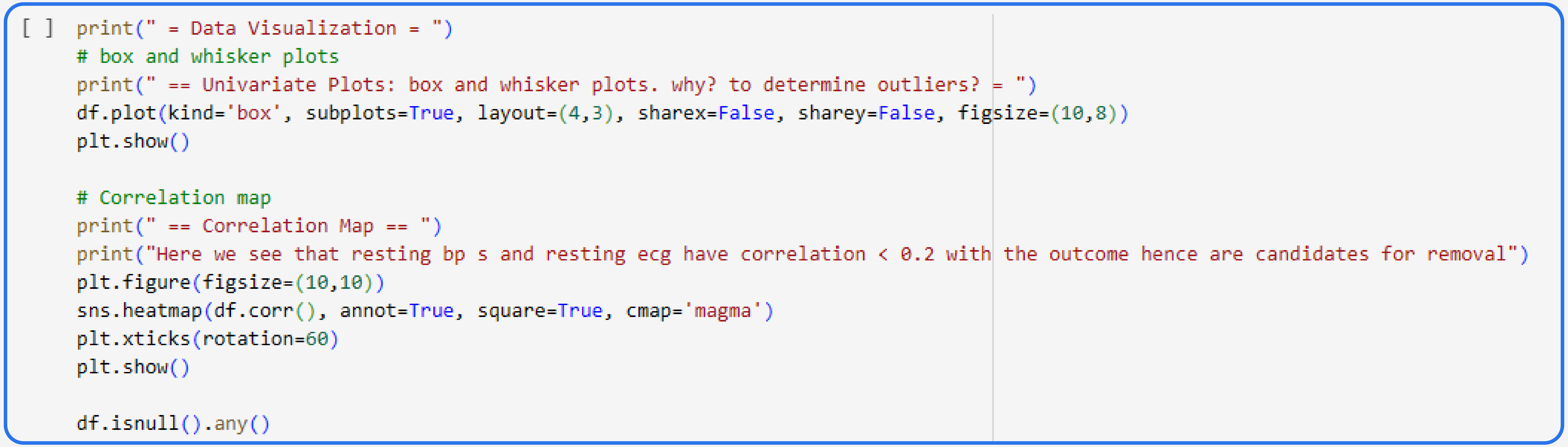
```
[2] from google.colab import files
    uploaded = files.upload()
```

```
Choose Files heart_statlo…ry_final.csv
• heart_statlog_cleveland_hungary_final.csv(text/csv) - 38805 bytes, last modified: 11/27/2024 - 100% done
Saving heart_statlog_cleveland_hungary_final.csv to heart_statlog_cleveland_hungary_final.csv
```

```
[3] # Load dataset
    df = pd.read_csv("heart_statlog_cleveland_hungary_final.csv")

    df.head()
    df.info()
    df.describe()

    #to count how many have and do not have heart disease
    df['target'].value_counts()

    #To get the percentage distribution of values of a column we use df['colname'].value_counts(normalize=True)*100
    df['target'].value_counts(normalize=True)*100

    # Plot target distribution
    #sns.catplot(x="target", kind="count", hue="target", palette="ch:.25", legend=False, data=df).set(title="Distribution of normal (0) vs with heart disease (1)")
    sns.catplot(x="target", kind="count", hue="target", palette="ch:.25", legend=False, data=df).set(title="Distribution of normal (0) vs with heart disease (1)")
```

Data Visualization

Original Code



```
[ ]  print(" = Data Visualization = ")
     # box and whisker plots
     print(" == Univariate Plots: box and whisker plots. why? to determine outliers? = ")
     df.plot(kind='box', subplots=True, layout=(4,3), sharex=False, sharey=False, figsize=(10,8))
     plt.show()

     # Correlation map
     print(" == Correlation Map == ")
     print("Here we see that resting bp s and resting ecg have correlation < 0.2 with the outcome hence are candidates for removal")
     plt.figure(figsize=(10,10))
     sns.heatmap(df.corr(), annot=True, square=True, cmap='magma')
     plt.xticks(rotation=60)
     plt.show()

     df.isnull().any()
```

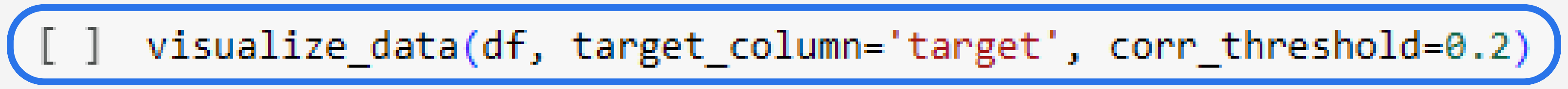
Data visualization

Changes made

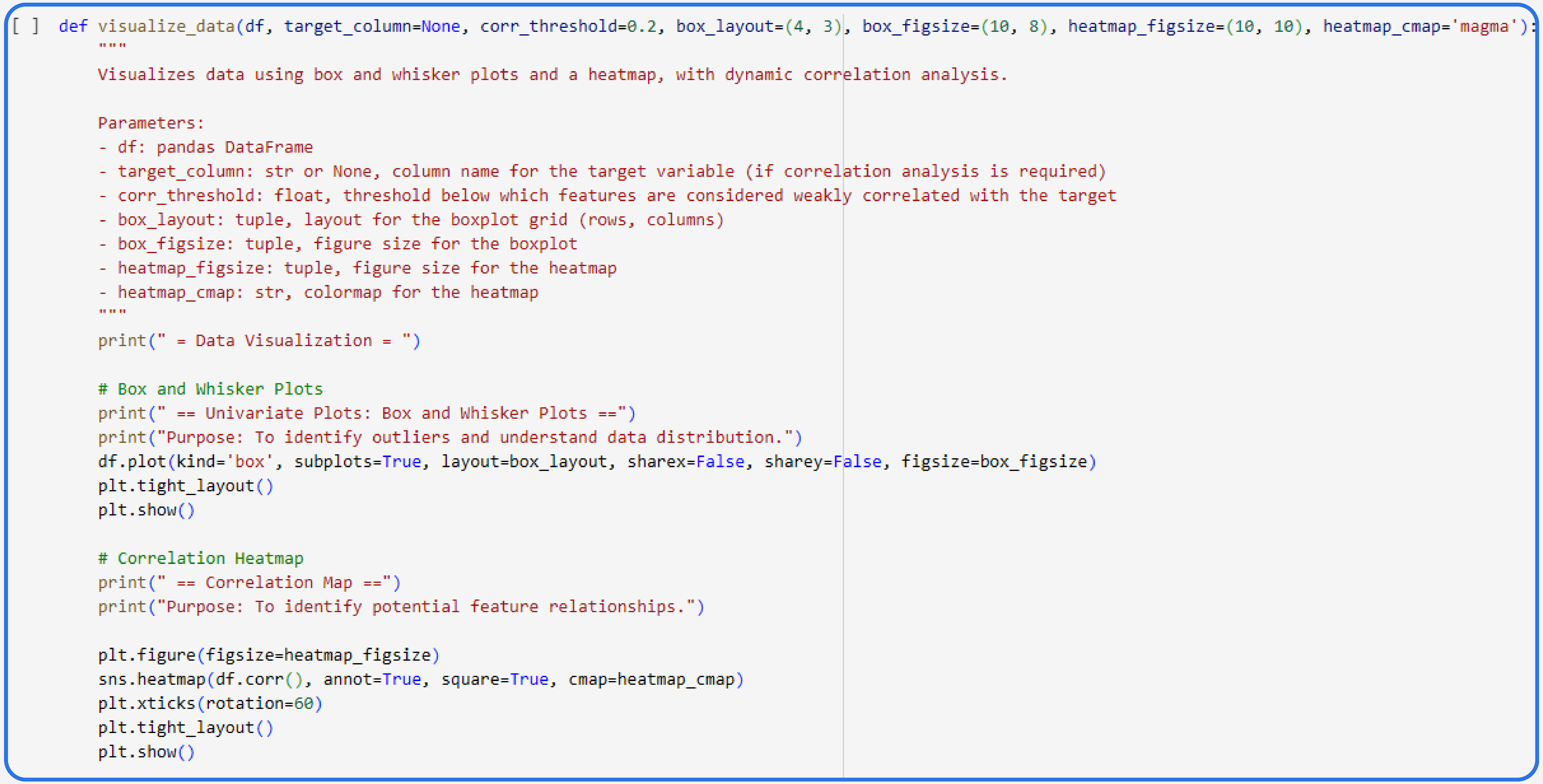
- Code was put inside a function
  - visualize_data(df, target_column='target', corr_threshold=0.2)
- No more hardcoded values
- Correlation threshold and target can be provided during a function call
- Automatically prints features with < 0.2 (weak) correlation with target

Data Visualization

Refactored Code



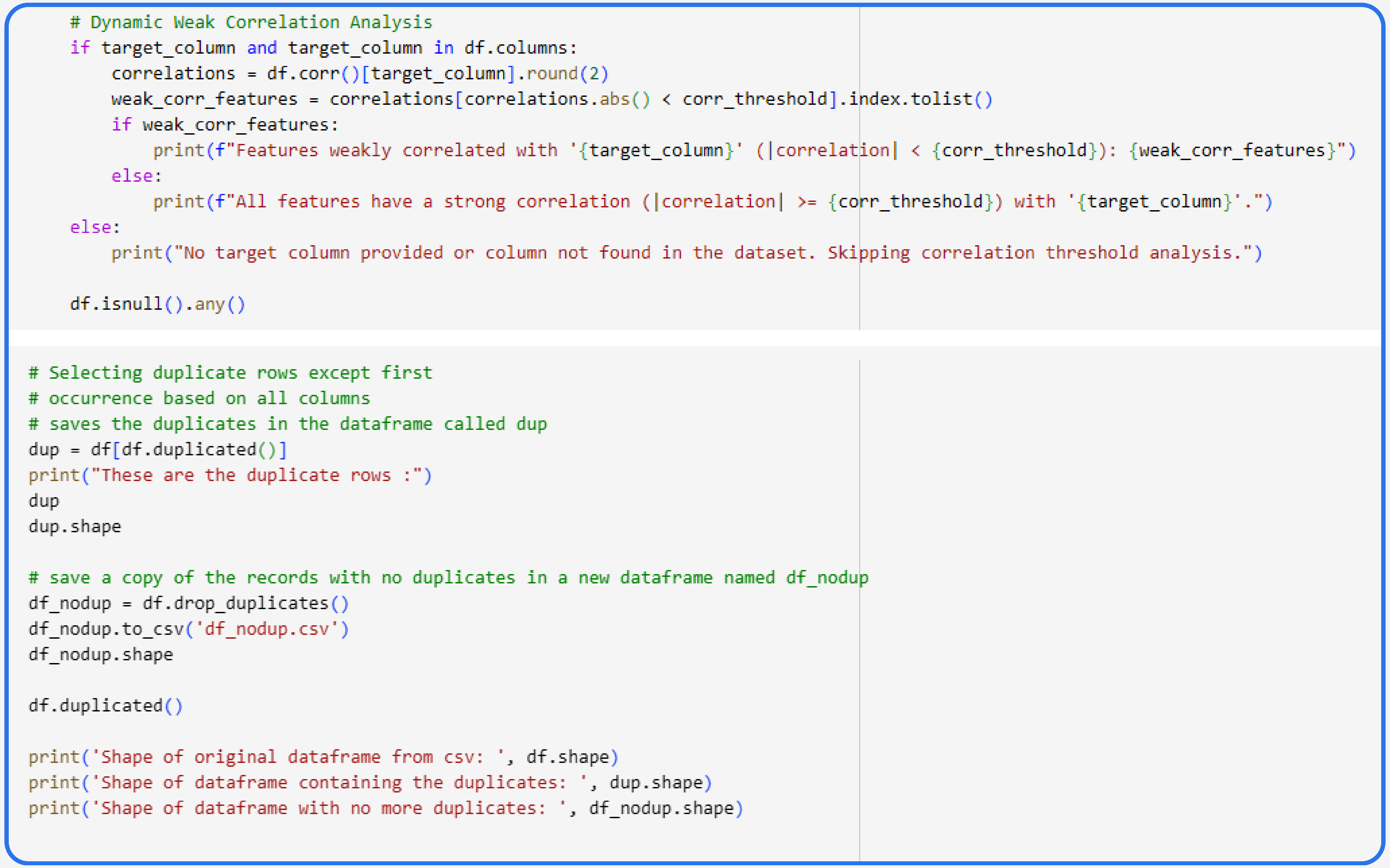
```
[ ]  visualize_data(df, target_column='target', corr_threshold=0.2)
```



```
[ ]  def visualize_data(df, target_column=None, corr_threshold=0.2, box_layout=(4, 3), box_figsize=(10, 8), heatmap_figsize=(10, 10), heatmap_cmap='magma'):
         """
         Visualizes data using box and whisker plots and a heatmap, with dynamic correlation analysis.

         Parameters:
         - df: pandas DataFrame
         - target_column: str or None, column name for the target variable (if correlation analysis is required)
         - corr_threshold: float, threshold below which features are considered weakly correlated with the target
         - box_layout: tuple, layout for the boxplot grid (rows, columns)
         - box_figsize: tuple, figure size for the boxplot
         - heatmap_figsize: tuple, figure size for the heatmap
         - heatmap_cmap: str, colormap for the heatmap
         """
         print(" = Data Visualization = ")

         # Box and Whisker Plots
         print(" == Univariate Plots: Box and Whisker Plots ==")
         print("Purpose: To identify outliers and understand data distribution.")
         df.plot(kind='box', subplots=True, layout=box_layout, sharex=False, sharey=False, figsize=box_figsize)
         plt.tight_layout()
         plt.show()

         # Correlation Heatmap
         print(" == Correlation Map ==")
         print("Purpose: To identify potential feature relationships.")

         plt.figure(figsize=heatmap_figsize)
         sns.heatmap(df.corr(), annot=True, square=True, cmap=heatmap_cmap)
         plt.xticks(rotation=60)
         plt.tight_layout()
         plt.show()
```

Data Visualization

Refactored Code

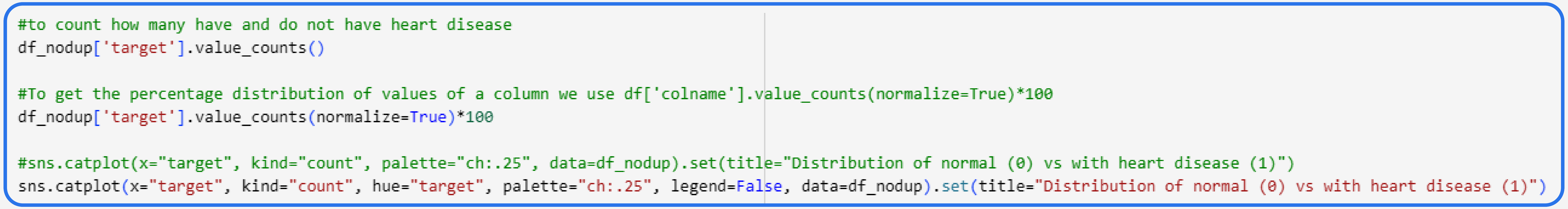


```python
    # Dynamic Weak Correlation Analysis
    if target_column and target_column in df.columns:
        correlations = df.corr()[target_column].round(2)
        weak_corr_features = correlations[correlations.abs() < corr_threshold].index.tolist()
        if weak_corr_features:
            print(f"Features weakly correlated with '{target_column}' (|correlation| < {corr_threshold}): {weak_corr_features}")
        else:
            print(f"All features have a strong correlation (|correlation| >= {corr_threshold}) with '{target_column}'.")
    else:
        print("No target column provided or column not found in the dataset. Skipping correlation threshold analysis.")

    df.isnull().any()
```

```python
# Selecting duplicate rows except first
# occurrence based on all columns
# saves the duplicates in the dataframe called dup
dup = df[df.duplicated()]
print("These are the duplicate rows :")
dup
dup.shape

# save a copy of the records with no duplicates in a new dataframe named df_nodup
df_nodup = df.drop_duplicates()
df_nodup.to_csv('df_nodup.csv')
df_nodup.shape

df.duplicated()

print('Shape of original dataframe from csv: ', df.shape)
print('Shape of dataframe containing the duplicates: ', dup.shape)
print('Shape of dataframe with no more duplicates: ', df_nodup.shape)
```

Rerun EDA on the dataframe with no duplicates

Original Code



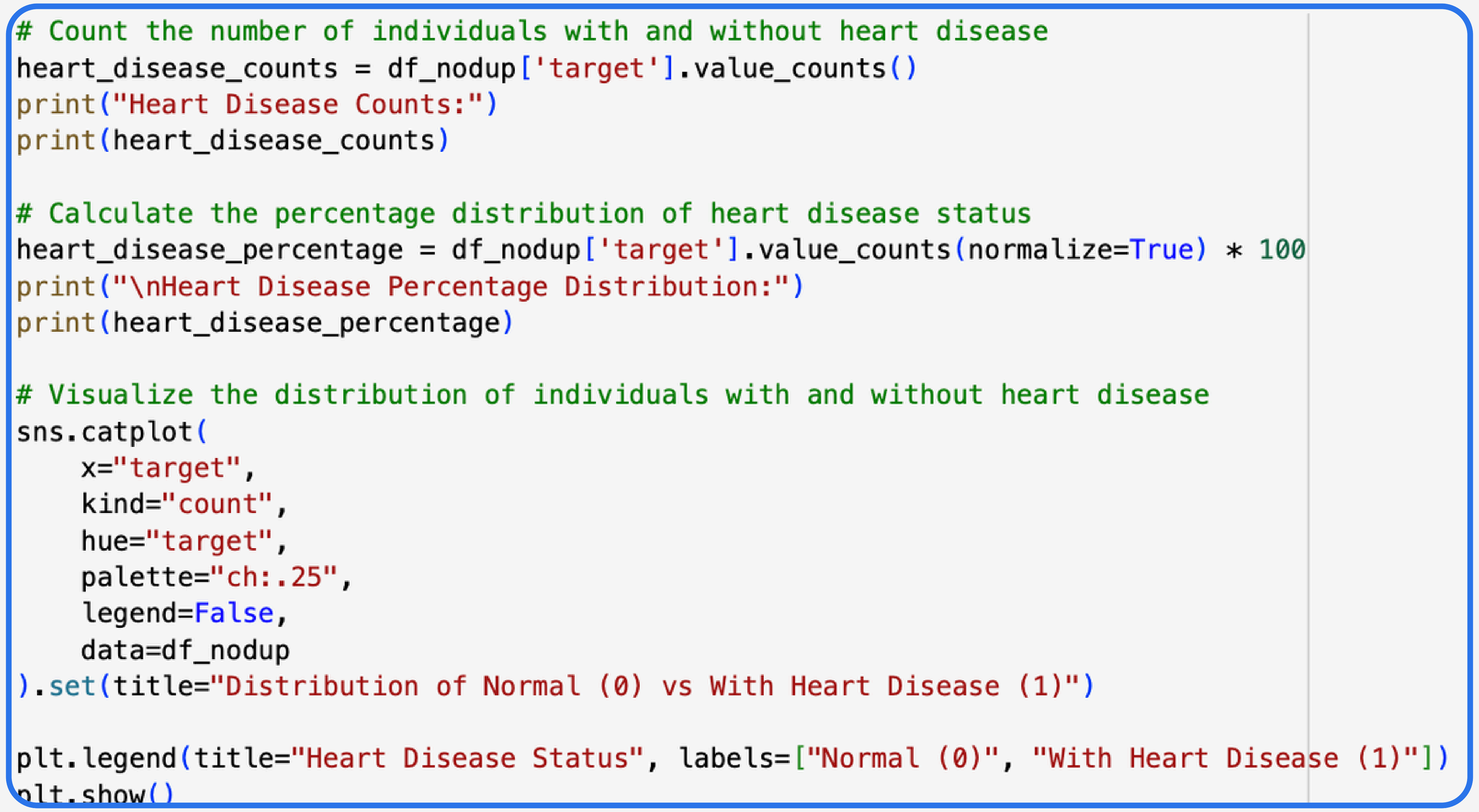
```
#to count how many have and do not have heart disease
df_nodup['target'].value_counts()

#To get the percentage distribution of values of a column we use df['colname'].value_counts(normalize=True)*100
df_nodup['target'].value_counts(normalize=True)*100

#sns.catplot(x="target", kind="count", palette="ch:.25", data=df_nodup).set(title="Distribution of normal (0) vs with heart disease (1)")
sns.catplot(x="target", kind="count", hue="target", palette="ch:.25", legend=False, data=df_nodup).set(title="Distribution of normal (0) vs with heart disease (1)")
```

Rerun EDA on the dataframe with no duplicates

Refactored Code

Changes made

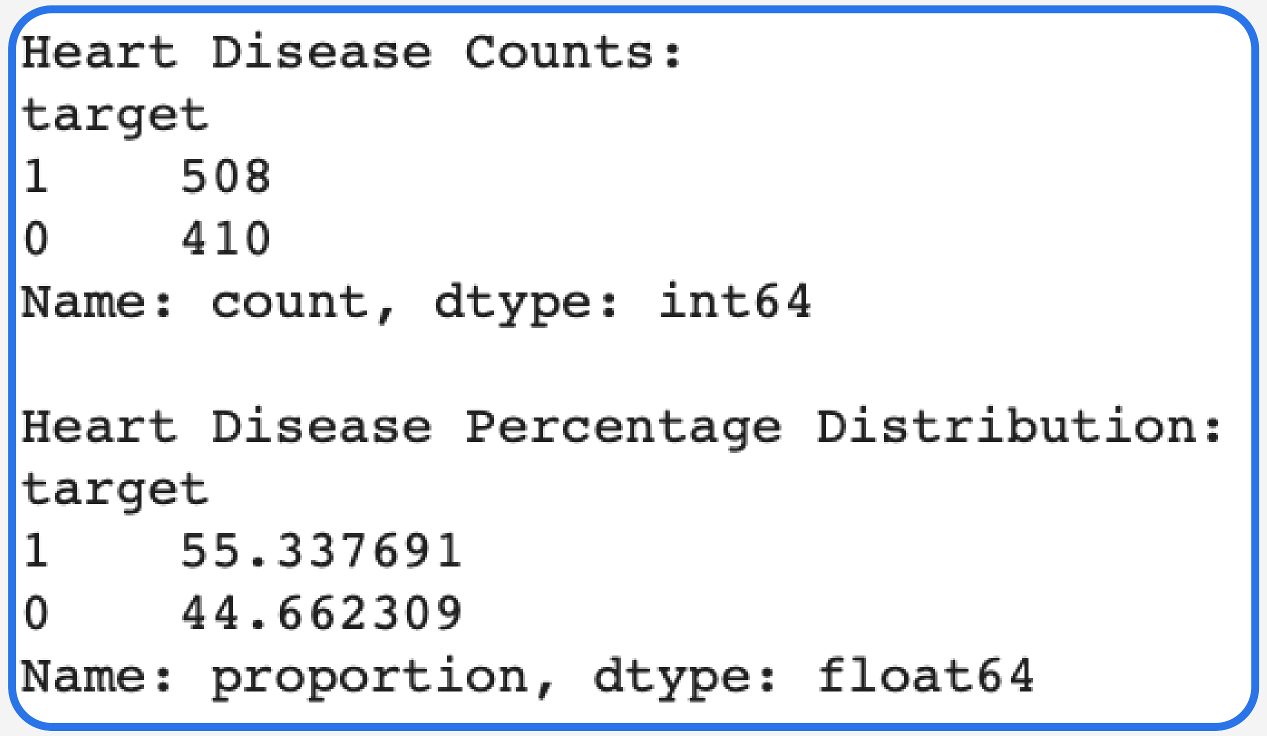
- Exact number of heart disease counts is displayed
- Percentage is displayed
- Labels were added to the graph



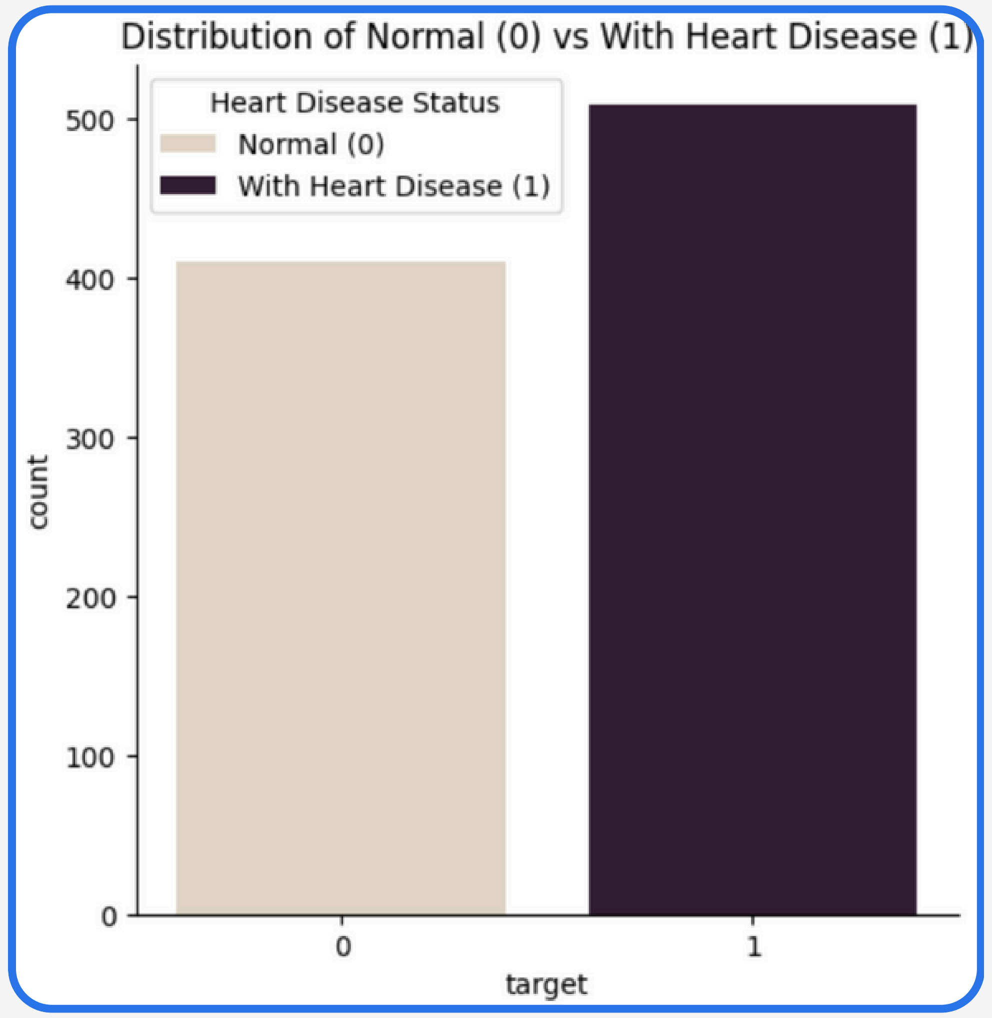
```python
# Count the number of individuals with and without heart disease
heart_disease_counts = df_nodup['target'].value_counts()
print("Heart Disease Counts:")
print(heart_disease_counts)

# Calculate the percentage distribution of heart disease status
heart_disease_percentage = df_nodup['target'].value_counts(normalize=True) * 100
print("\nHeart Disease Percentage Distribution:")
print(heart_disease_percentage)

# Visualize the distribution of individuals with and without heart disease
sns.catplot(
    x="target",
    kind="count",
    hue="target",
    palette="ch:.25",
    legend=False,
    data=df_nodup
).set(title="Distribution of Normal (0) vs With Heart Disease (1)")

plt.legend(title="Heart Disease Status", labels=["Normal (0)", "With Heart Disease (1)"])
plt.show()
```

Rerun EDA on the dataframe with no duplicates

Output

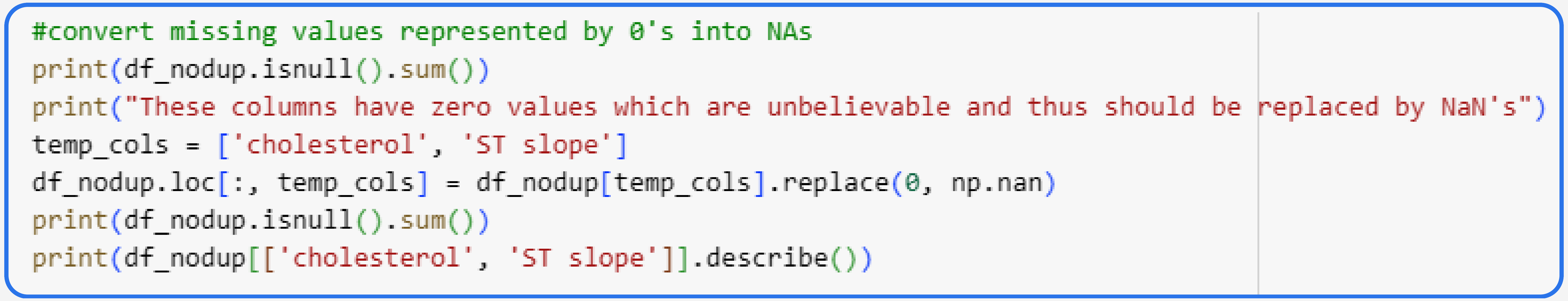


```
Heart Disease Counts:
target
1     508
0     410
Name: count, dtype: int64

Heart Disease Percentage Distribution:
target
1     55.337691
0     44.662309
Name: proportion, dtype: float64
```



Handling Missing Values

Original Code



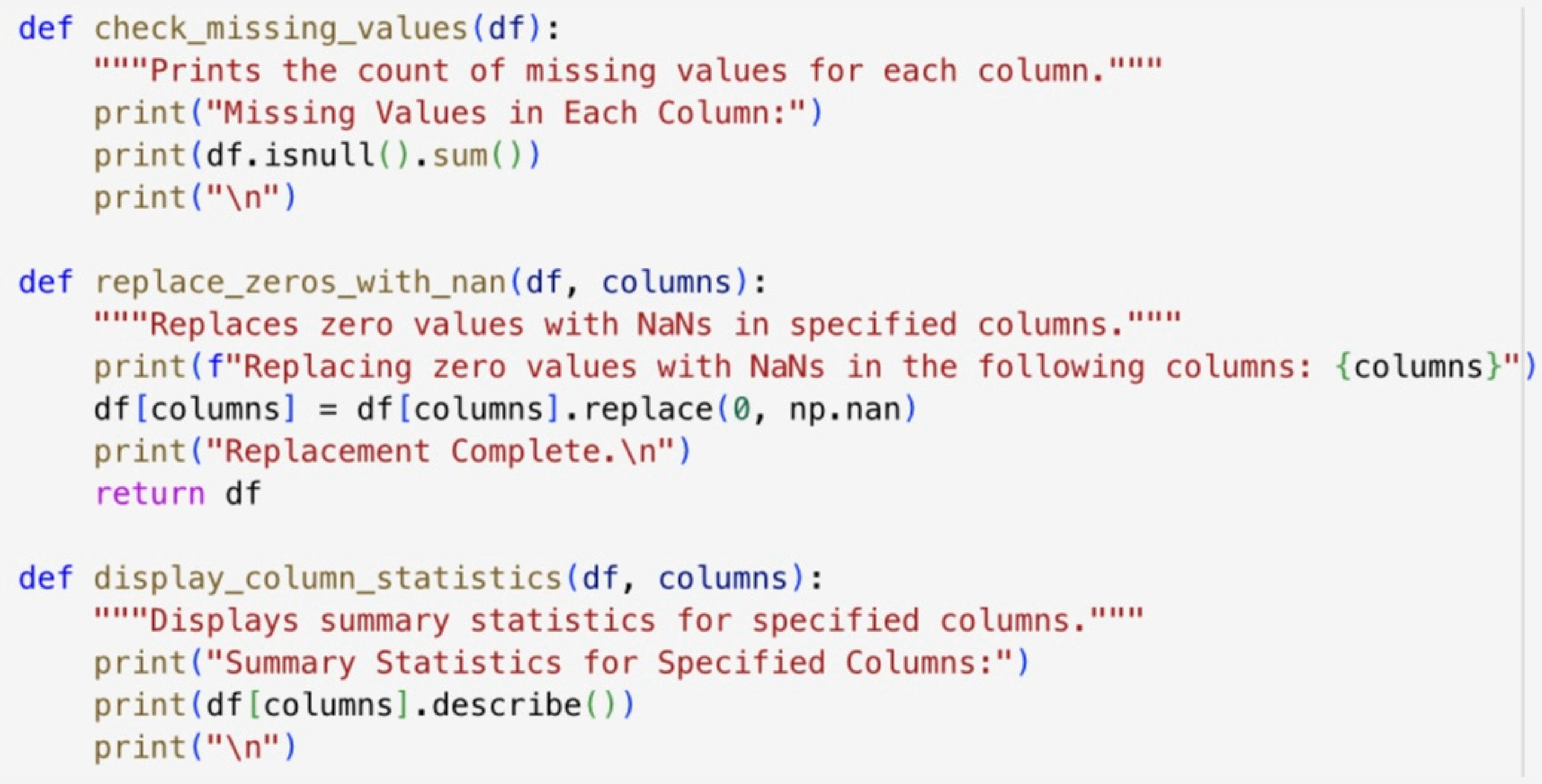
```
#convert missing values represented by 0's into NAs
print(df_nodup.isnull().sum())
print("These columns have zero values which are unbelievable and thus should be replaced by NaN's")
temp_cols = ['cholesterol', 'ST slope']
df_nodup.loc[:, temp_cols] = df_nodup[temp_cols].replace(0, np.nan)
print(df_nodup.isnull().sum())
print(df_nodup[['cholesterol', 'ST slope']].describe())
```

Handling missing values

Refactored Code

Changes made

- Extract method
    - Turning the method into subfunctions



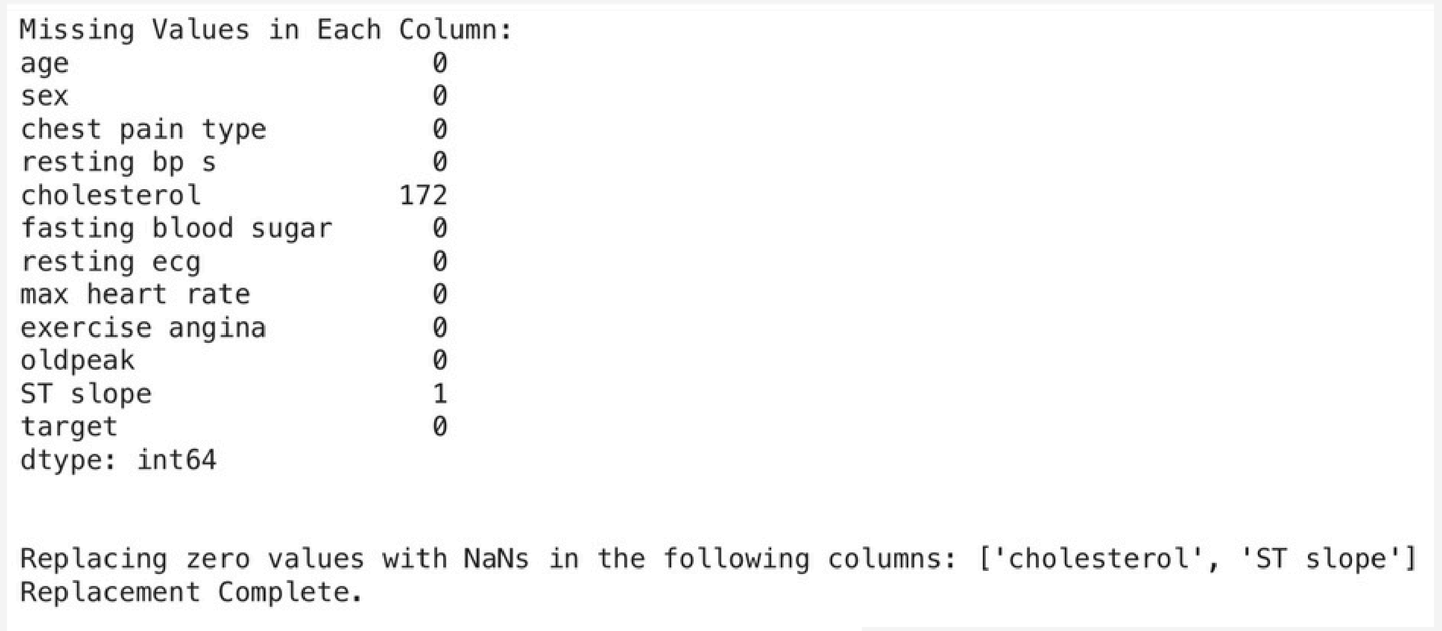
```python
def check_missing_values(df):
    """Prints the count of missing values for each column."""
    print("Missing Values in Each Column:")
    print(df.isnull().sum())
    print("\n")

def replace_zeros_with_nan(df, columns):
    """Replaces zero values with NaNs in specified columns."""
    print(f"Replacing zero values with NaNs in the following columns: {columns}")
    df[columns] = df[columns].replace(0, np.nan)
    print("Replacement Complete.\n")
    return df

def display_column_statistics(df, columns):
    """Displays summary statistics for specified columns."""
    print("Summary Statistics for Specified Columns:")
    print(df[columns].describe())
    print("\n")
```

Handling missing values

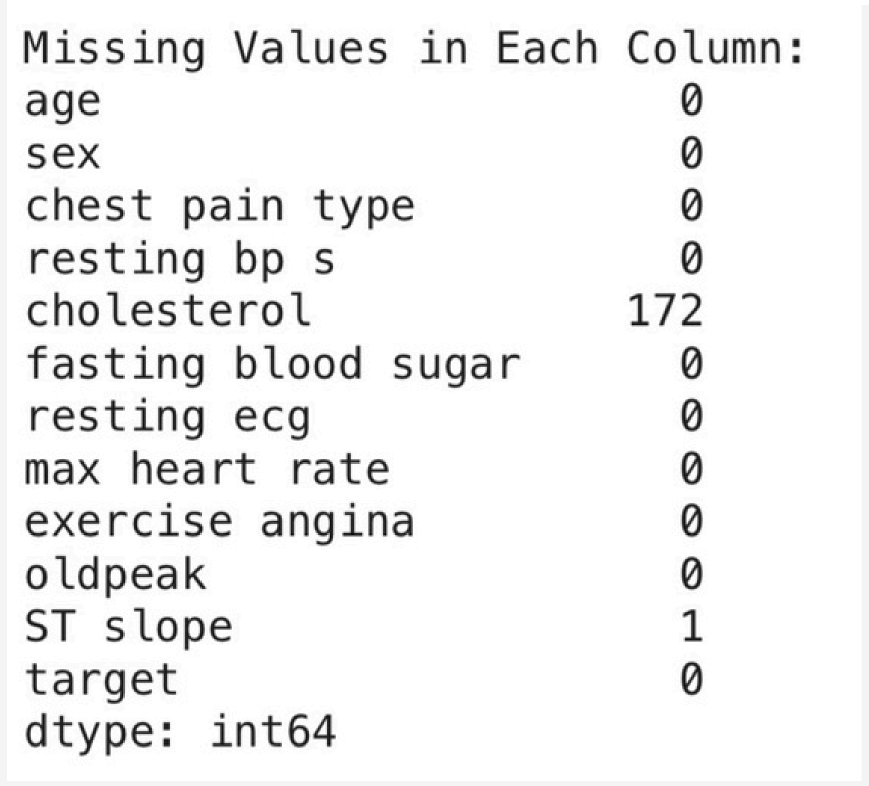
Refactored Code

Changes made

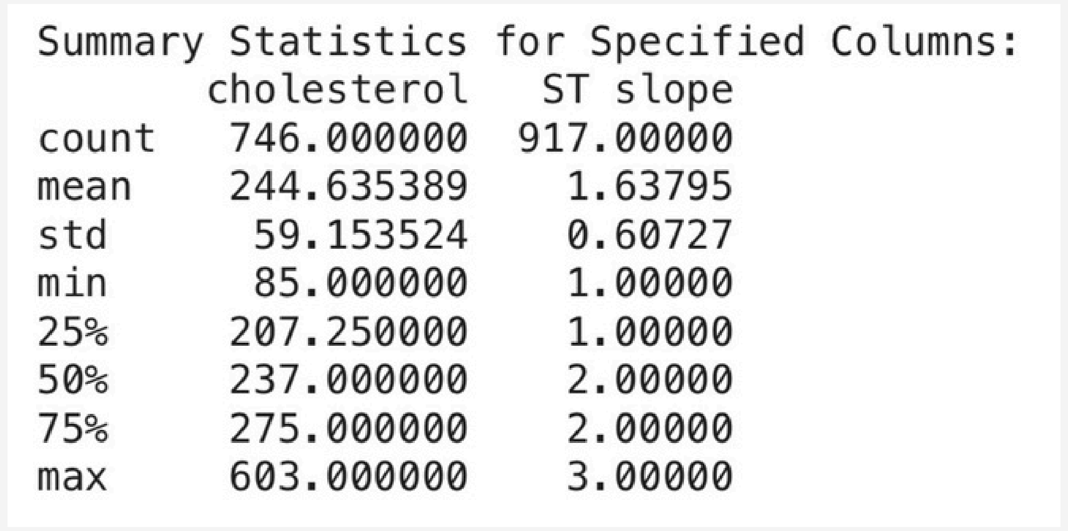
- Extract method
  - Turning the method into subfunctions



```
Missing Values in Each Column:
age                      0
sex                      0
chest pain type          0
resting bp s             0
cholesterol            172
fasting blood sugar      0
resting ecg              0
max heart rate           0
exercise angina          0
oldpeak                  0
ST slope                 1
target                   0
dtype: int64


Replacing zero values with NaNs in the following columns: ['cholesterol', 'ST slope']
Replacement Complete.
```



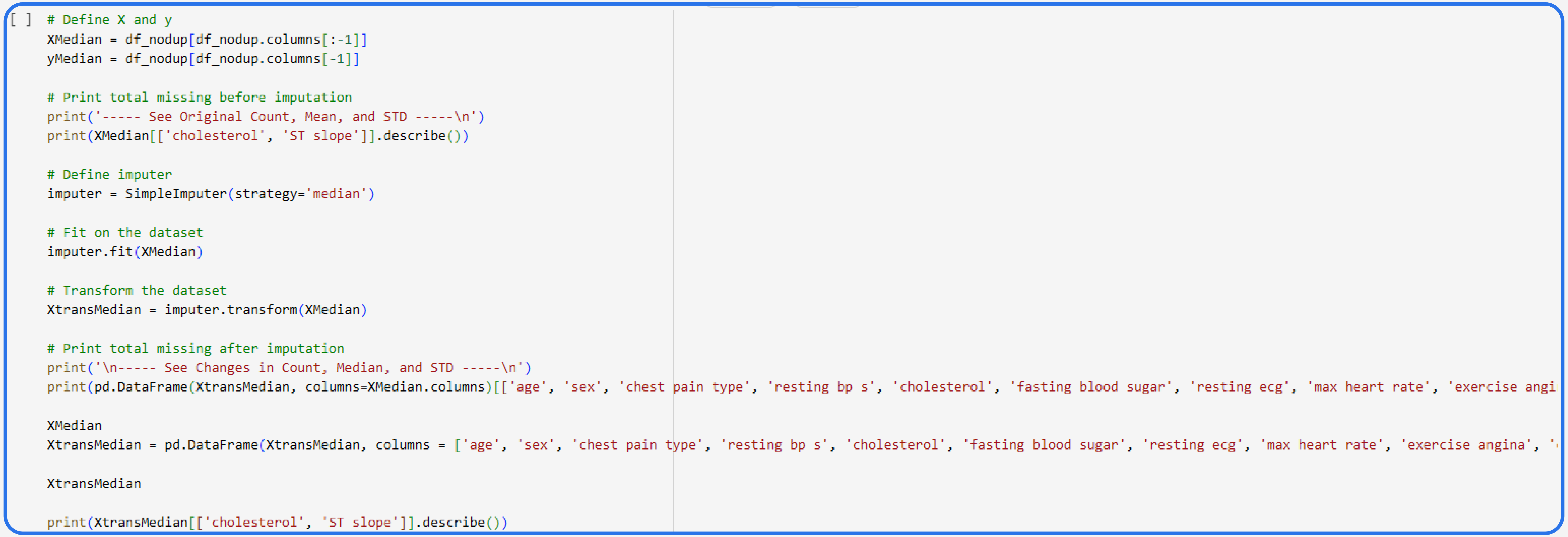
```
Missing Values in Each Column:
age                      0
sex                      0
chest pain type          0
resting bp s             0
cholesterol            172
fasting blood sugar      0
resting ecg              0
max heart rate           0
exercise angina          0
oldpeak                  0
ST slope                 1
target                   0
dtype: int64
```



```
Summary Statistics for Specified Columns:
       cholesterol   ST slope
count   746.000000  917.00000
mean    244.635389    1.63795
std      59.153524    0.60727
min      85.000000    1.00000
25%     207.250000    1.00000
50%     237.000000    2.00000
75%     275.000000    2.00000
max     603.000000    3.00000
```

Imputing the NaN's in Insulin, Skin Thickness, BMI, Blood Pressure, and Glucose using the median

Original Code



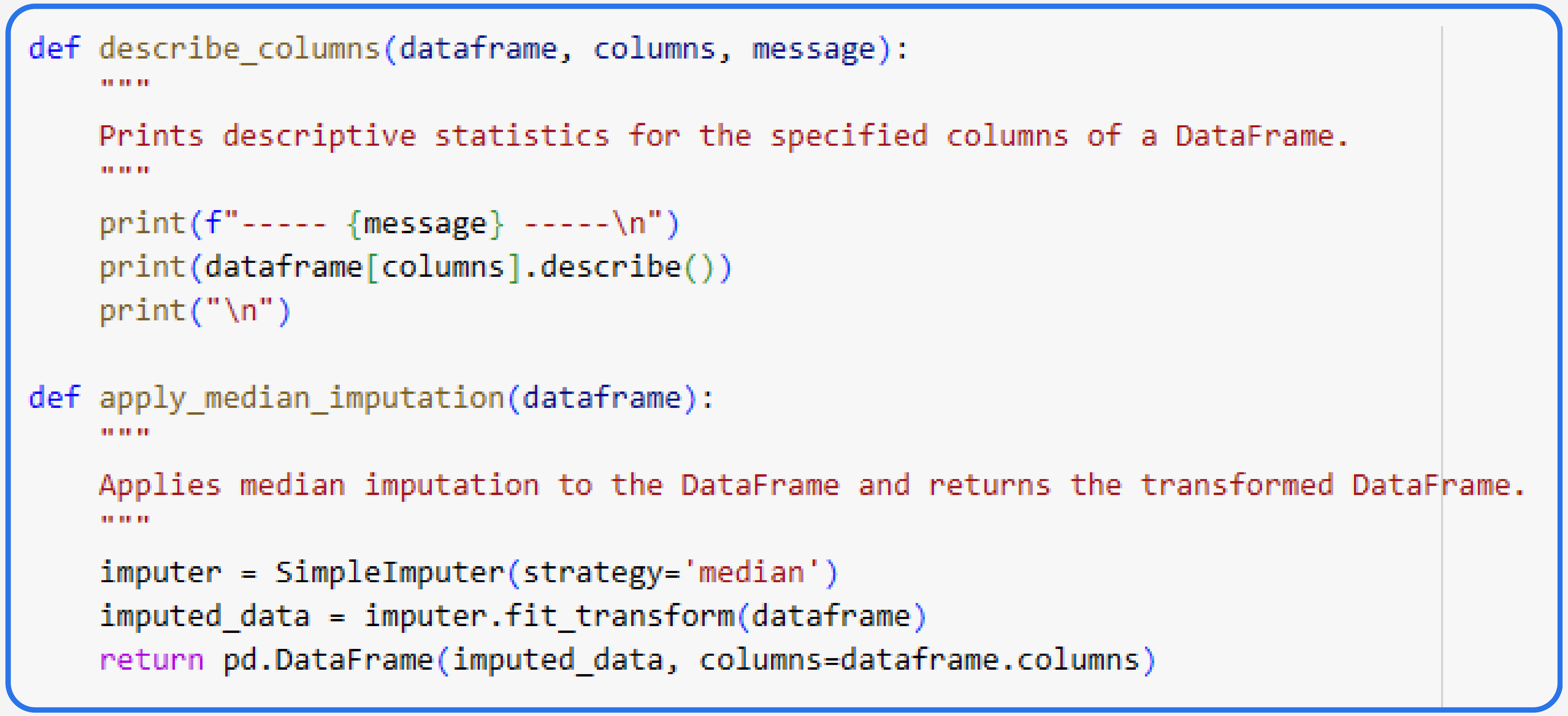
```
[ ]  # Define X and y
     XMedian = df_nodup[df_nodup.columns[:-1]]
     yMedian = df_nodup[df_nodup.columns[-1]]

     # Print total missing before imputation
     print('----- See Original Count, Mean, and STD -----\n')
     print(XMedian[['cholesterol', 'ST slope']].describe())

     # Define imputer
     imputer = SimpleImputer(strategy='median')

     # Fit on the dataset
     imputer.fit(XMedian)

     # Transform the dataset
     XtransMedian = imputer.transform(XMedian)

     # Print total missing after imputation
     print('\n----- See Changes in Count, Median, and STD -----\n')
     print(pd.DataFrame(XtransMedian, columns=XMedian.columns)[['age', 'sex', 'chest pain type', 'resting bp s', 'cholesterol', 'fasting blood sugar', 'resting ecg', 'max heart rate', 'exercise angi

     XMedian
     XtransMedian = pd.DataFrame(XtransMedian, columns = ['age', 'sex', 'chest pain type', 'resting bp s', 'cholesterol', 'fasting blood sugar', 'resting ecg', 'max heart rate', 'exercise angina', '

     XtransMedian

     print(XtransMedian[['cholesterol', 'ST slope']].describe())
```

Imputing the NaN's in Insulin, Skin Thickness, BMI, Blood Pressure, and Glucose using the median

Refactored Code

Changes made

- Extract method
  - Turning the method into subfunctions



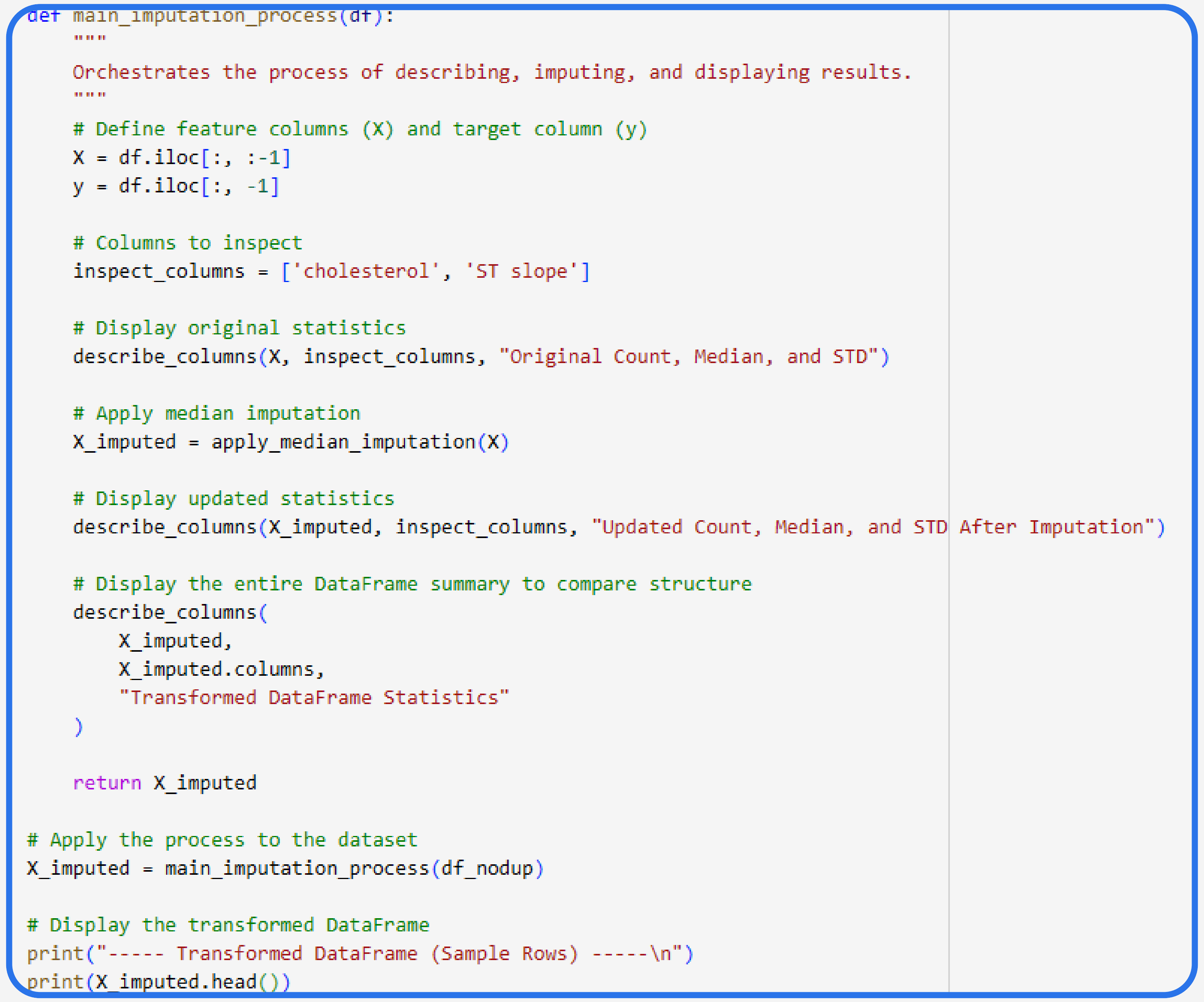
```python
def describe_columns(dataframe, columns, message):
    """
    Prints descriptive statistics for the specified columns of a DataFrame.
    """
    print(f"----- {message} -----\n")
    print(dataframe[columns].describe())
    print("\n")

def apply_median_imputation(dataframe):
    """
    Applies median imputation to the DataFrame and returns the transformed DataFrame.
    """
    imputer = SimpleImputer(strategy='median')
    imputed_data = imputer.fit_transform(dataframe)
    return pd.DataFrame(imputed_data, columns=dataframe.columns)
```

Imputing the NaN's in Insulin, Skin Thickness, BMI, Blood Pressure, and Glucose using the median

Refactored Code

Changes made

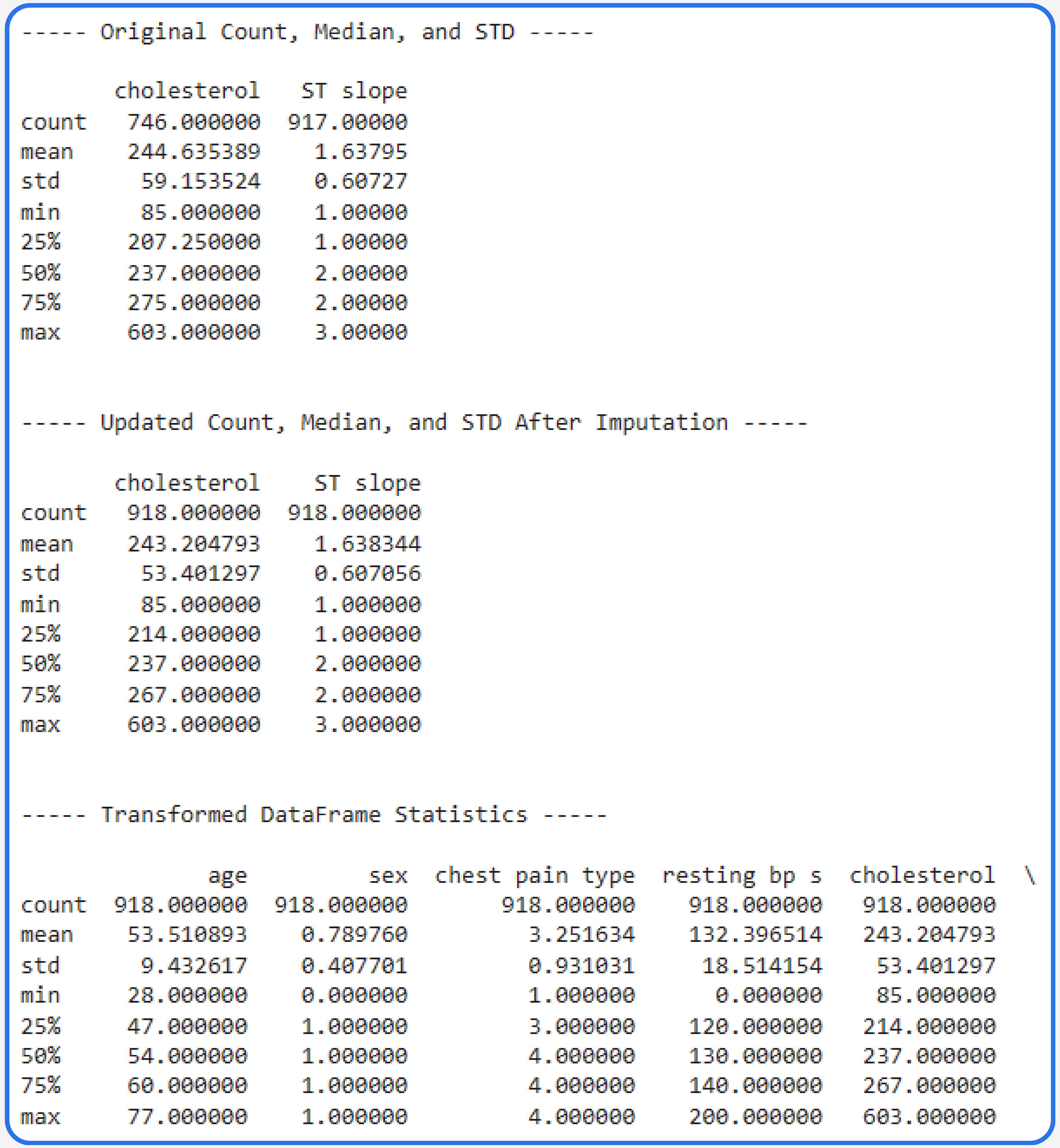
- Printed the step by step process instead of directly updating the full transformed dataframe statistics:
  - Original Count, Median, and STD
  - Updated Count, Median, and STD
  - Full Transformed DataFrame Statistics
  - Transformed DataFrame (Sample Rows)



```python
def main_imputation_process(df):
    """
    Orchestrates the process of describing, imputing, and displaying results.
    """
    # Define feature columns (X) and target column (y)
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]

    # Columns to inspect
    inspect_columns = ['cholesterol', 'ST slope']

    # Display original statistics
    describe_columns(X, inspect_columns, "Original Count, Median, and STD")

    # Apply median imputation
    X_imputed = apply_median_imputation(X)

    # Display updated statistics
    describe_columns(X_imputed, inspect_columns, "Updated Count, Median, and STD After Imputation")

    # Display the entire DataFrame summary to compare structure
    describe_columns(
        X_imputed,
        X_imputed.columns,
        "Transformed DataFrame Statistics"
    )

    return X_imputed

# Apply the process to the dataset
X_imputed = main_imputation_process(df_nodup)

# Display the transformed DataFrame
print("----- Transformed DataFrame (Sample Rows) -----\n")
print(X_imputed.head())
```

Imputing the NaN's in Insulin, Skin Thickness, BMI, Blood Pressure, and Glucose using the median

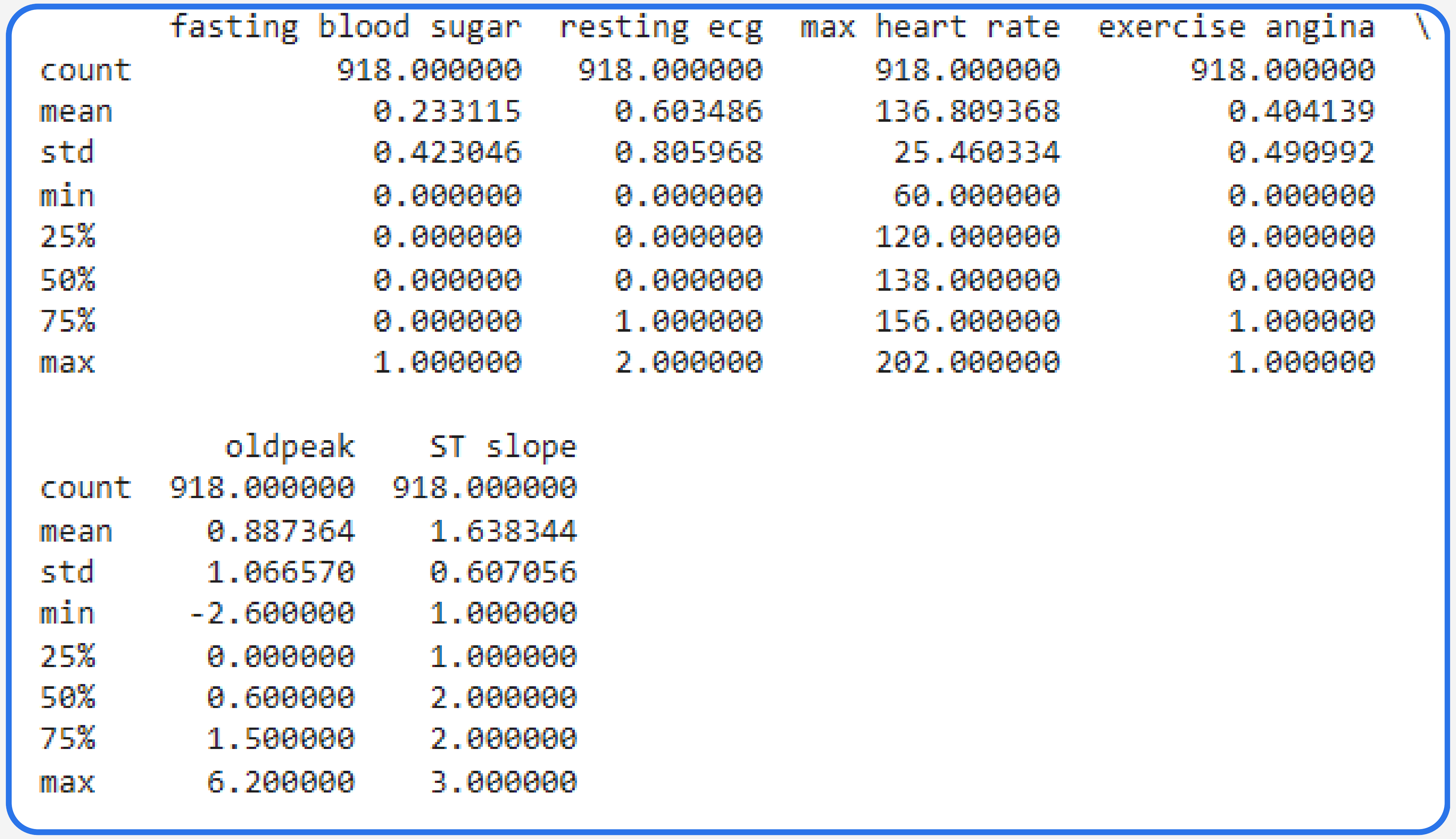
Refactored Code



```
----- Original Count, Median, and STD -----

       cholesterol   ST slope
count   746.000000  917.00000
mean    244.635389    1.63795
std      59.153524    0.60727
min      85.000000    1.00000
25%     207.250000    1.00000
50%     237.000000    2.00000
75%     275.000000    2.00000
max     603.000000    3.00000


----- Updated Count, Median, and STD After Imputation -----

       cholesterol    ST slope
count   918.000000  918.000000
mean    243.204793    1.638344
std      53.401297    0.607056
min      85.000000    1.000000
25%     214.000000    1.000000
50%     237.000000    2.000000
75%     267.000000    2.000000
max     603.000000    3.000000


----- Transformed DataFrame Statistics -----

              age         sex  chest pain type  resting bp s  cholesterol  \
count  918.000000  918.000000       918.000000    918.000000   918.000000
mean    53.510893    0.789760         3.251634    132.396514   243.204793
std      9.432617    0.407701         0.931031     18.514154    53.401297
min     28.000000    0.000000         1.000000      0.000000    85.000000
25%     47.000000    1.000000         3.000000    120.000000   214.000000
50%     54.000000    1.000000         4.000000    130.000000   237.000000
75%     60.000000    1.000000         4.000000    140.000000   267.000000
max     77.000000    1.000000         4.000000    200.000000   603.000000
```

Imputing the NaN's in Insulin, Skin Thickness, BMI, Blood Pressure, and Glucose using the median

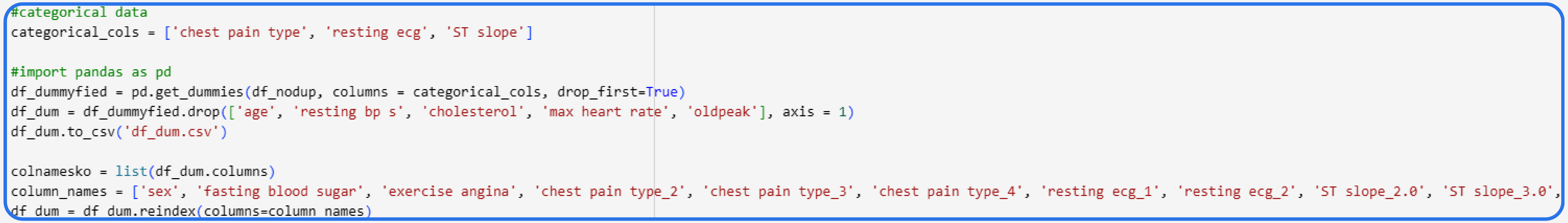
Refactored Code



```
       fasting blood sugar  resting ecg  max heart rate  exercise angina  \
count           918.000000   918.000000      918.000000       918.000000
mean              0.233115     0.603486      136.809368         0.404139
std               0.423046     0.805968       25.460334         0.490992
min               0.000000     0.000000       60.000000         0.000000
25%               0.000000     0.000000      120.000000         0.000000
50%               0.000000     0.000000      138.000000         0.000000
75%               0.000000     1.000000      156.000000         1.000000
max               1.000000     2.000000      202.000000         1.000000

          oldpeak    ST slope
count  918.000000  918.000000
mean     0.887364    1.638344
std      1.066570    0.607056
min     -2.600000    1.000000
25%      0.000000    1.000000
50%      0.600000    2.000000
75%      1.500000    2.000000
max      6.200000    3.000000
```

Preprocessing categorical variables using get_dummies() function of pandas

Original Code



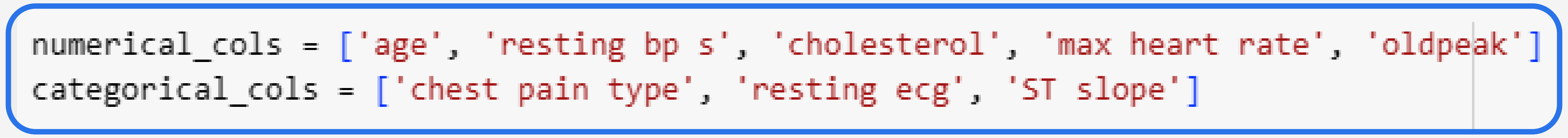
```
#categorical data
categorical_cols = ['chest pain type', 'resting ecg', 'ST slope']

#import pandas as pd
df_dummyfied = pd.get_dummies(df_nodup, columns = categorical_cols, drop_first=True)
df_dum = df_dummyfied.drop(['age', 'resting bp s', 'cholesterol', 'max heart rate', 'oldpeak'], axis = 1)
df_dum.to_csv('df_dum.csv')

colnamesko = list(df_dum.columns)
column_names = ['sex', 'fasting blood sugar', 'exercise angina', 'chest pain type_2', 'chest pain type_3', 'chest pain type_4', 'resting ecg_1', 'resting ecg_2', 'ST slope_2.0', 'ST slope_3.0',
df_dum = df_dum.reindex(columns=column_names)
```

Preprocessing categorical variables using get_dummies() function of pandas

Refactored Code

Changes made

- Refactored variables
  - Preprocessing categorical variables using get_dummies() function of pandas



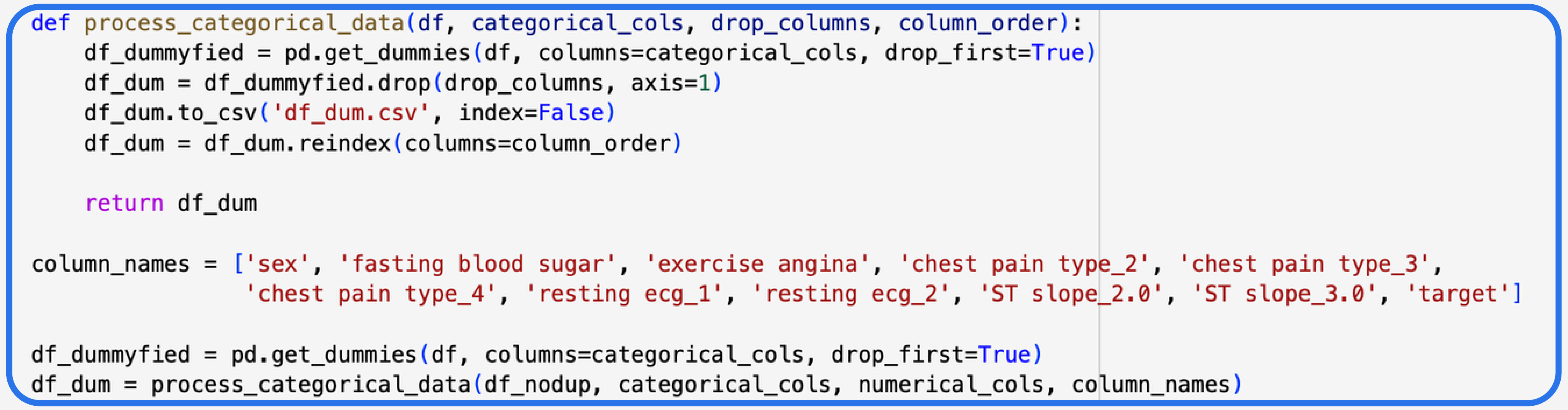
```python
numerical_cols = ['age', 'resting bp s', 'cholesterol', 'max heart rate', 'oldpeak']
categorical_cols = ['chest pain type', 'resting ecg', 'ST slope']
```

Preprocessing categorical variables using get_dummies() function of pandas

Refactored Code

Changes made

- Replace Magic Numbers with Constants
  - Replace ['age', 'resting bp s', 'cholesterol', 'max heart rate', 'oldpeak'] to variable numerical_cols
  - Replace ['chest pain type', 'resting ecg', 'ST slope'] to variable categorical_cols
- Consolidate duplicate code
  - Used a function to implement processing of categorical data into one

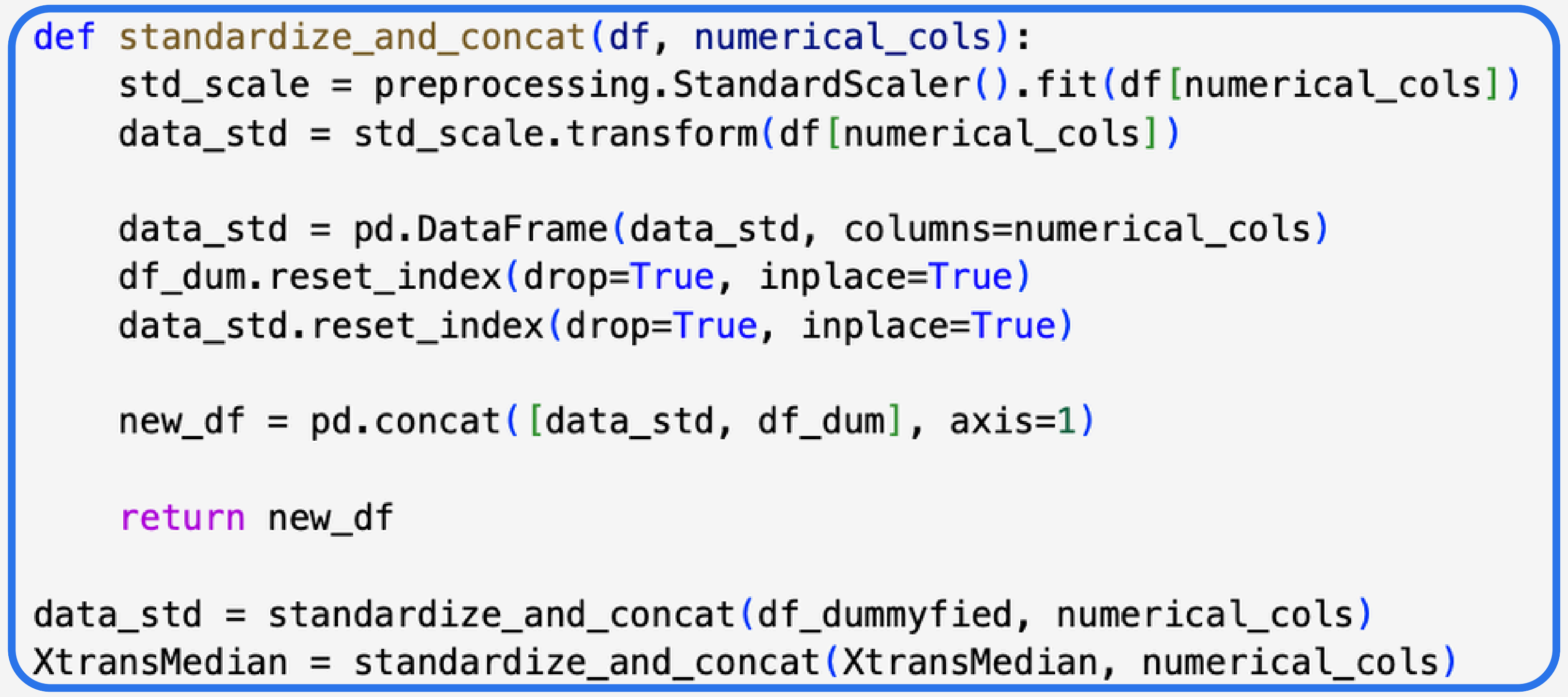


```
def process_categorical_data(df, categorical_cols, drop_columns, column_order):
    df_dummyfied = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
    df_dum = df_dummyfied.drop(drop_columns, axis=1)
    df_dum.to_csv('df_dum.csv', index=False)
    df_dum = df_dum.reindex(columns=column_order)

    return df_dum

column_names = ['sex', 'fasting blood sugar', 'exercise angina', 'chest pain type_2', 'chest pain type_3',
                'chest pain type_4', 'resting ecg_1', 'resting ecg_2', 'ST slope_2.0', 'ST slope_3.0', 'target']

df_dummyfied = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
df_dum = process_categorical_data(df_nodup, categorical_cols, numerical_cols, column_names)
```

Preprocessing Numeric variables using StandardScaler

Changes made

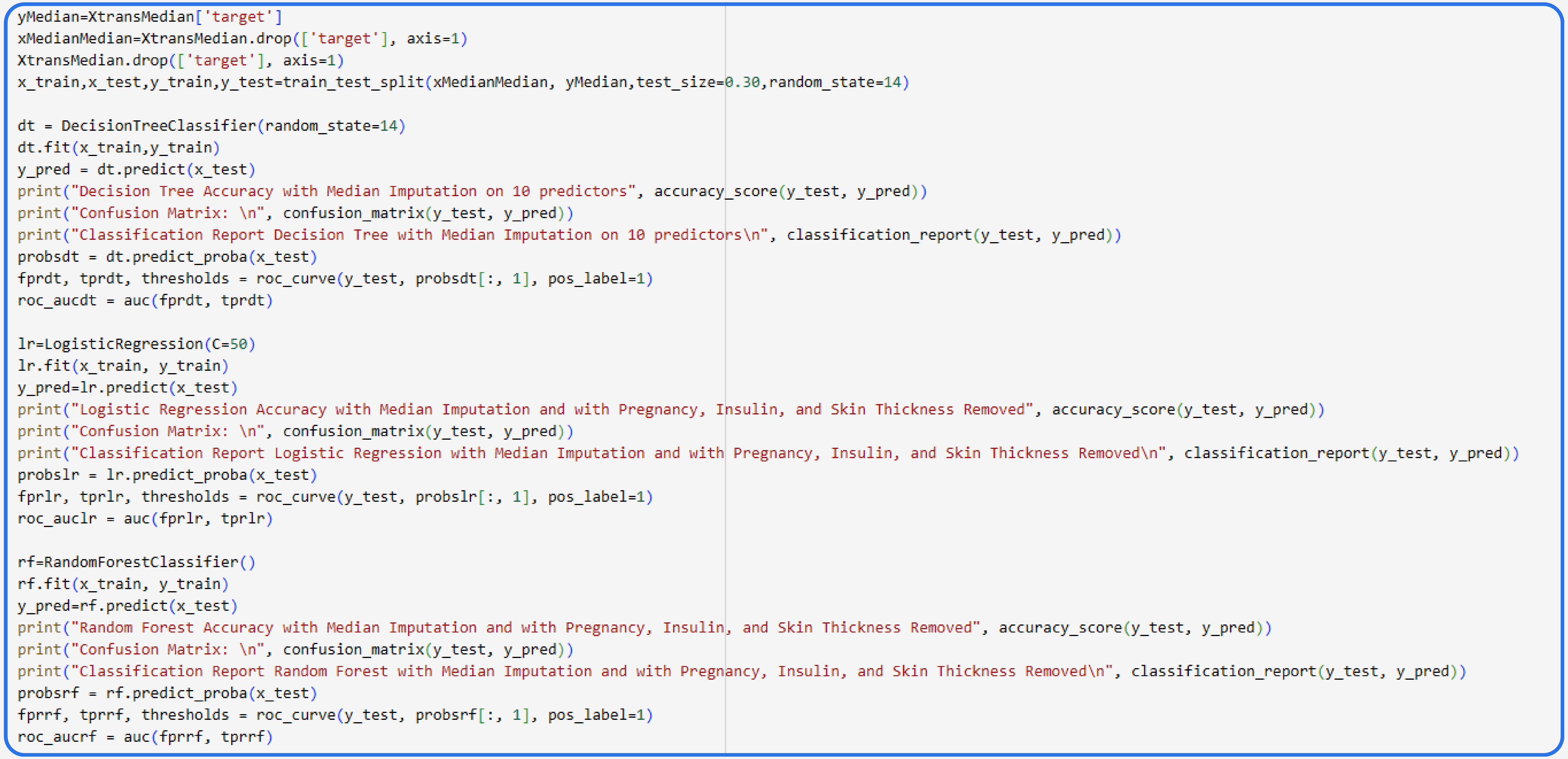
- Replace Magic Numbers with Constants
  - Replace ['age', 'resting bp s', 'cholesterol', 'max heart rate', 'oldpeak'] to variable numerical_cols
- Consolidate Duplicate Code
  - Used a function to implement Preprocessing, etc into one



```python
def standardize_and_concat(df, numerical_cols):
    std_scale = preprocessing.StandardScaler().fit(df[numerical_cols])
    data_std = std_scale.transform(df[numerical_cols])

    data_std = pd.DataFrame(data_std, columns=numerical_cols)
    df_dum.reset_index(drop=True, inplace=True)
    data_std.reset_index(drop=True, inplace=True)

    new_df = pd.concat([data_std, df_dum], axis=1)

    return new_df

data_std = standardize_and_concat(df_dummyfied, numerical_cols)
XtransMedian = standardize_and_concat(XtransMedian, numerical_cols)
```

Performance evaluation

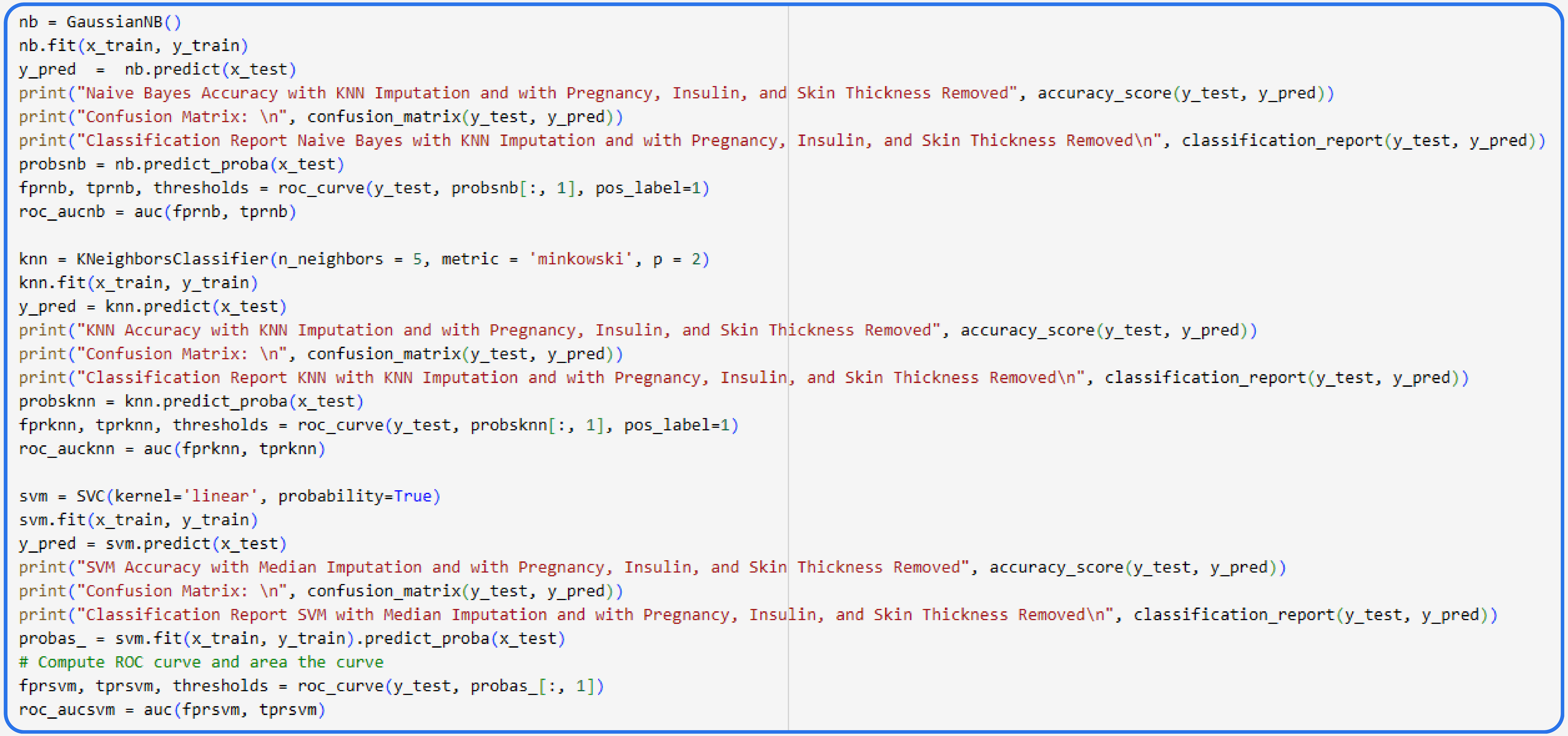
Original Code



```
yMedian=XtransMedian['target']
xMedianMedian=XtransMedian.drop(['target'], axis=1)
XtransMedian.drop(['target'], axis=1)
x_train,x_test,y_train,y_test=train_test_split(xMedianMedian, yMedian,test_size=0.30,random_state=14)

dt = DecisionTreeClassifier(random_state=14)
dt.fit(x_train,y_train)
y_pred = dt.predict(x_test)
print("Decision Tree Accuracy with Median Imputation on 10 predictors", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report Decision Tree with Median Imputation on 10 predictors\n", classification_report(y_test, y_pred))
probsdt = dt.predict_proba(x_test)
fprdt, tprdt, thresholds = roc_curve(y_test, probsdt[:, 1], pos_label=1)
roc_aucdt = auc(fprdt, tprdt)

lr=LogisticRegression(C=50)
lr.fit(x_train, y_train)
y_pred=lr.predict(x_test)
print("Logistic Regression Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report Logistic Regression with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed\n", classification_report(y_test, y_pred))
probslr = lr.predict_proba(x_test)
fprlr, tprlr, thresholds = roc_curve(y_test, probslr[:, 1], pos_label=1)
roc_auclr = auc(fprlr, tprlr)

rf=RandomForestClassifier()
rf.fit(x_train, y_train)
y_pred=rf.predict(x_test)
print("Random Forest Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report Random Forest with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed\n", classification_report(y_test, y_pred))
probsrf = rf.predict_proba(x_test)
fprrf, tprrf, thresholds = roc_curve(y_test, probsrf[:, 1], pos_label=1)
roc_aucrf = auc(fprrf, tprrf)
```

Performance evaluation

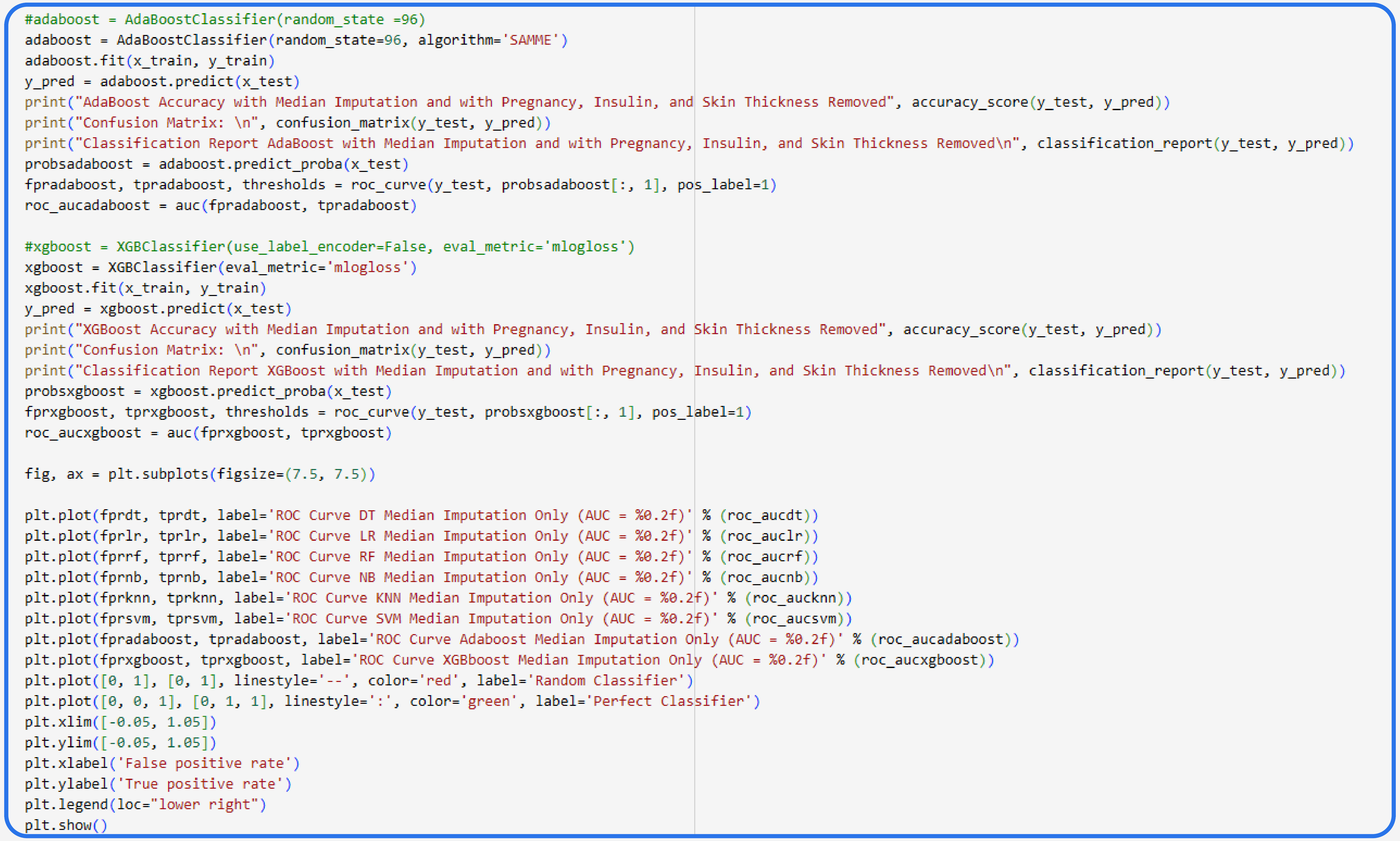
Original Code



```
nb = GaussianNB()
nb.fit(x_train, y_train)
y_pred  =  nb.predict(x_test)
print("Naive Bayes Accuracy with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report Naive Bayes with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed\n", classification_report(y_test, y_pred))
probsnb = nb.predict_proba(x_test)
fprnb, tprnb, thresholds = roc_curve(y_test, probsnb[:, 1], pos_label=1)
roc_aucnb = auc(fprnb, tprnb)

knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
print("KNN Accuracy with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report KNN with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed\n", classification_report(y_test, y_pred))
probsknn = knn.predict_proba(x_test)
fprknn, tprknn, thresholds = roc_curve(y_test, probsknn[:, 1], pos_label=1)
roc_aucknn = auc(fprknn, tprknn)

svm = SVC(kernel='linear', probability=True)
svm.fit(x_train, y_train)
y_pred = svm.predict(x_test)
print("SVM Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report SVM with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed\n", classification_report(y_test, y_pred))
probas_ = svm.fit(x_train, y_train).predict_proba(x_test)
# Compute ROC curve and area the curve
fprsvm, tprsvm, thresholds = roc_curve(y_test, probas_[:, 1])
roc_aucsvm = auc(fprsvm, tprsvm)
```

Performance evaluation

Original Code



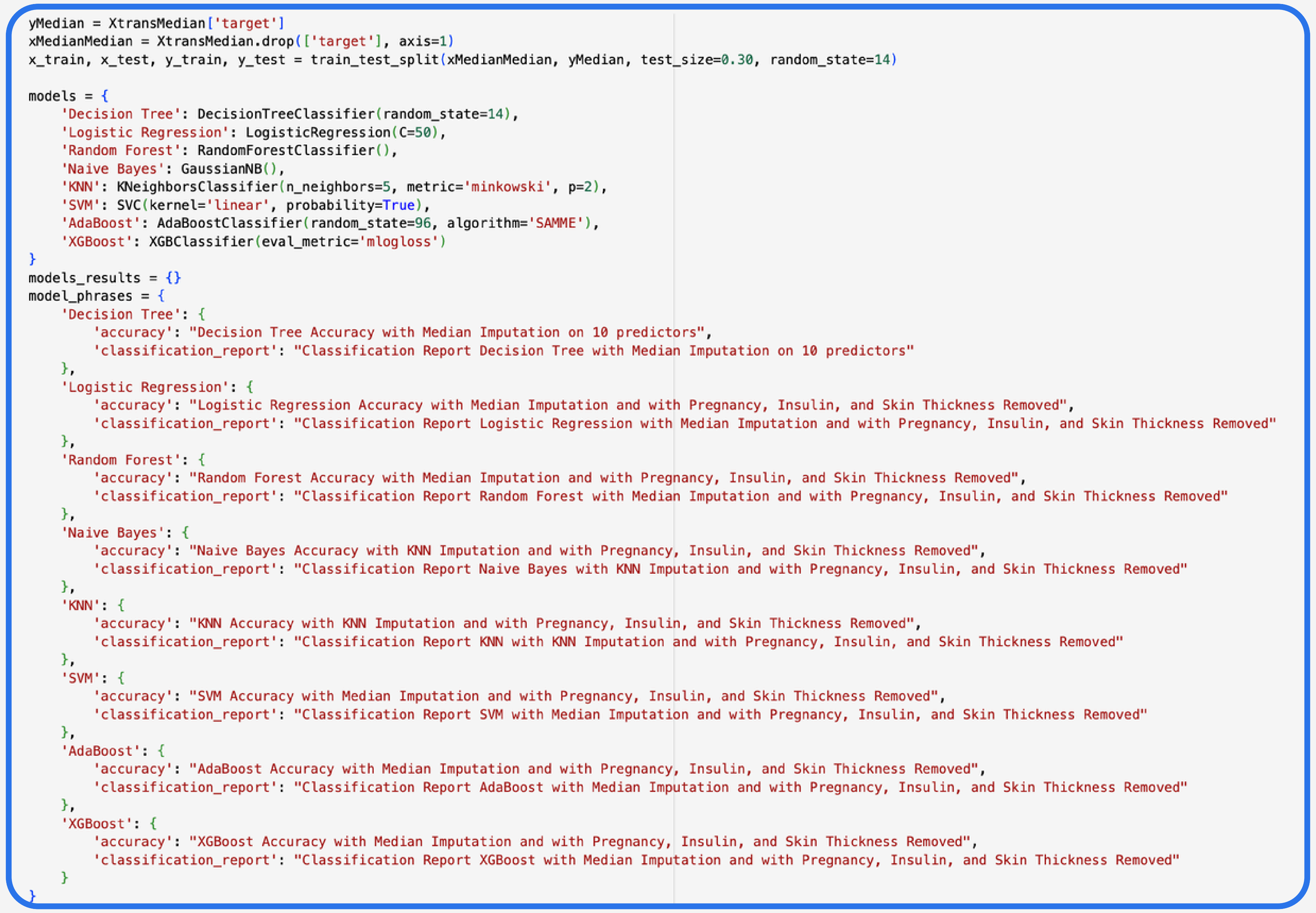
```python
#adaboost = AdaBoostClassifier(random_state =96)
adaboost = AdaBoostClassifier(random_state=96, algorithm='SAMME')
adaboost.fit(x_train, y_train)
y_pred = adaboost.predict(x_test)
print("AdaBoost Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report AdaBoost with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed\n", classification_report(y_test, y_pred))
probsadaboost = adaboost.predict_proba(x_test)
fpradaboost, tpradaboost, thresholds = roc_curve(y_test, probsadaboost[:, 1], pos_label=1)
roc_aucadaboost = auc(fpradaboost, tpradaboost)

#xgboost = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgboost = XGBClassifier(eval_metric='mlogloss')
xgboost.fit(x_train, y_train)
y_pred = xgboost.predict(x_test)
print("XGBoost Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report XGBoost with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed\n", classification_report(y_test, y_pred))
probsxgboost = xgboost.predict_proba(x_test)
fprxgboost, tprxgboost, thresholds = roc_curve(y_test, probsxgboost[:, 1], pos_label=1)
roc_aucxgboost = auc(fprxgboost, tprxgboost)

fig, ax = plt.subplots(figsize=(7.5, 7.5))

plt.plot(fprdt, tprdt, label='ROC Curve DT Median Imputation Only (AUC = %0.2f)' % (roc_aucdt))
plt.plot(fprlr, tprlr, label='ROC Curve LR Median Imputation Only (AUC = %0.2f)' % (roc_auclr))
plt.plot(fprrf, tprrf, label='ROC Curve RF Median Imputation Only (AUC = %0.2f)' % (roc_aucrf))
plt.plot(fprnb, tprnb, label='ROC Curve NB Median Imputation Only (AUC = %0.2f)' % (roc_aucnb))
plt.plot(fprknn, tprknn, label='ROC Curve KNN Median Imputation Only (AUC = %0.2f)' % (roc_aucknn))
plt.plot(fprsvm, tprsvm, label='ROC Curve SVM Median Imputation Only (AUC = %0.2f)' % (roc_aucsvm))
plt.plot(fpradaboost, tpradaboost, label='ROC Curve Adaboost Median Imputation Only (AUC = %0.2f)' % (roc_aucadaboost))
plt.plot(fprxgboost, tprxgboost, label='ROC Curve XGBboost Median Imputation Only (AUC = %0.2f)' % (roc_aucxgboost))
plt.plot([0, 1], [0, 1], linestyle='--', color='red', label='Random Classifier')
plt.plot([0, 0, 1], [0, 1, 1], linestyle=':', color='green', label='Perfect Classifier')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.legend(loc="lower right")
plt.show()
```

Performance evaluation

Refactored Code

Changes made

- Consolidate Duplicate Code
  - Dictionary
    - map training and evaluating each model to its corresponding action, reducing repetitive lines of codes that do the same function.
    - to map print functions



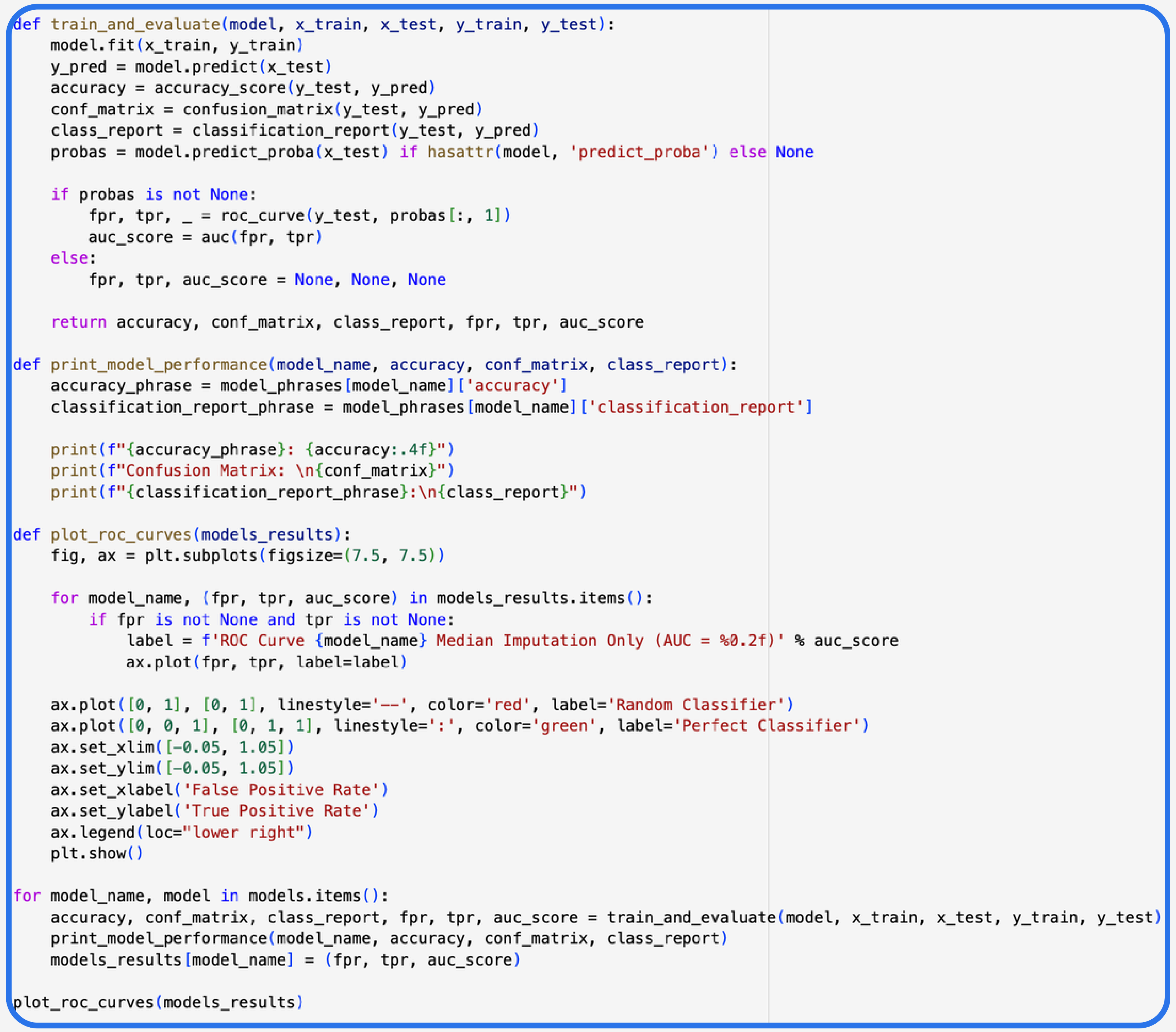
```python
yMedian = XtransMedian['target']
xMedianMedian = XtransMedian.drop(['target'], axis=1)
x_train, x_test, y_train, y_test = train_test_split(xMedianMedian, yMedian, test_size=0.30, random_state=14)

models = {
    'Decision Tree': DecisionTreeClassifier(random_state=14),
    'Logistic Regression': LogisticRegression(C=50),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'KNN': KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2),
    'SVM': SVC(kernel='linear', probability=True),
    'AdaBoost': AdaBoostClassifier(random_state=96, algorithm='SAMME'),
    'XGBoost': XGBClassifier(eval_metric='mlogloss')
}
models_results = {}
model_phrases = {
    'Decision Tree': {
        'accuracy': "Decision Tree Accuracy with Median Imputation on 10 predictors",
        'classification_report': "Classification Report Decision Tree with Median Imputation on 10 predictors"
    },
    'Logistic Regression': {
        'accuracy': "Logistic Regression Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed",
        'classification_report': "Classification Report Logistic Regression with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed"
    },
    'Random Forest': {
        'accuracy': "Random Forest Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed",
        'classification_report': "Classification Report Random Forest with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed"
    },
    'Naive Bayes': {
        'accuracy': "Naive Bayes Accuracy with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed",
        'classification_report': "Classification Report Naive Bayes with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed"
    },
    'KNN': {
        'accuracy': "KNN Accuracy with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed",
        'classification_report': "Classification Report KNN with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed"
    },
    'SVM': {
        'accuracy': "SVM Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed",
        'classification_report': "Classification Report SVM with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed"
    },
    'AdaBoost': {
        'accuracy': "AdaBoost Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed",
        'classification_report': "Classification Report AdaBoost with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed"
    },
    'XGBoost': {
        'accuracy': "XGBoost Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed",
        'classification_report': "Classification Report XGBoost with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed"
    }
}
```

Performance evaluation

Refactored Code

Changes made

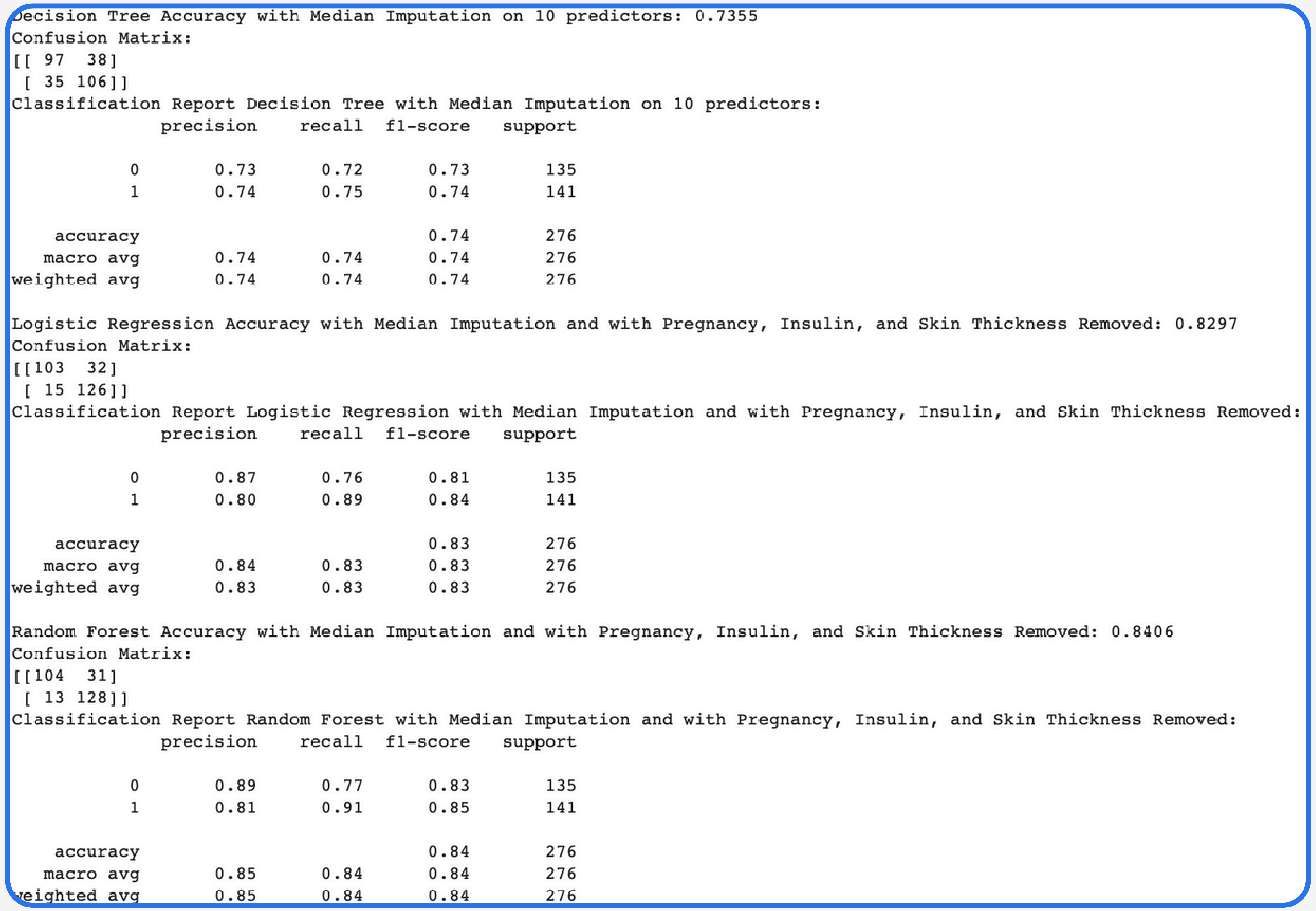
- Consolidate Duplicate Code
  - Used a function to implement Plotting of ROC Curve into one



```python
def train_and_evaluate(model, x_train, x_test, y_train, y_test):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)
    probas = model.predict_proba(x_test) if hasattr(model, 'predict_proba') else None

    if probas is not None:
        fpr, tpr, _ = roc_curve(y_test, probas[:, 1])
        auc_score = auc(fpr, tpr)
    else:
        fpr, tpr, auc_score = None, None, None

    return accuracy, conf_matrix, class_report, fpr, tpr, auc_score

def print_model_performance(model_name, accuracy, conf_matrix, class_report):
    accuracy_phrase = model_phrases[model_name]['accuracy']
    classification_report_phrase = model_phrases[model_name]['classification_report']

    print(f"{accuracy_phrase}: {accuracy:.4f}")
    print(f"Confusion Matrix: \n{conf_matrix}")
    print(f"{classification_report_phrase}:\n{class_report}")

def plot_roc_curves(models_results):
    fig, ax = plt.subplots(figsize=(7.5, 7.5))

    for model_name, (fpr, tpr, auc_score) in models_results.items():
        if fpr is not None and tpr is not None:
            label = f'ROC Curve {model_name} Median Imputation Only (AUC = %0.2f)' % auc_score
            ax.plot(fpr, tpr, label=label)

    ax.plot([0, 1], [0, 1], linestyle='--', color='red', label='Random Classifier')
    ax.plot([0, 0, 1], [0, 1, 1], linestyle=':', color='green', label='Perfect Classifier')
    ax.set_xlim([-0.05, 1.05])
    ax.set_ylim([-0.05, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.legend(loc="lower right")
    plt.show()

for model_name, model in models.items():
    accuracy, conf_matrix, class_report, fpr, tpr, auc_score = train_and_evaluate(model, x_train, x_test, y_train, y_test)
    print_model_performance(model_name, accuracy, conf_matrix, class_report)
    models_results[model_name] = (fpr, tpr, auc_score)

plot_roc_curves(models_results)
```

Performance evaluation

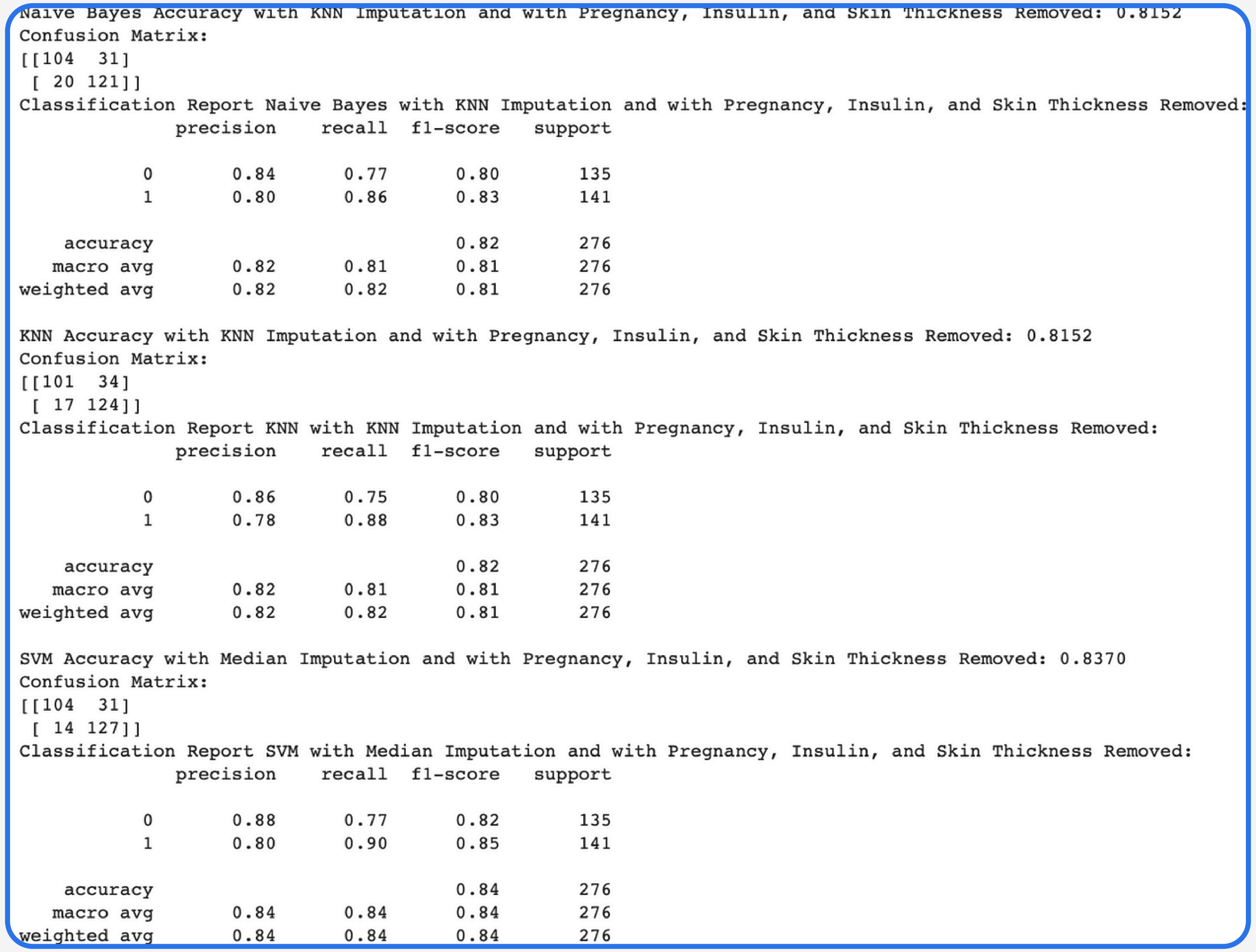
Refactored Code



```
Decision Tree Accuracy with Median Imputation on 10 predictors: 0.7355
Confusion Matrix:
[[ 97  38]
 [ 35 106]]
Classification Report Decision Tree with Median Imputation on 10 predictors:
              precision    recall  f1-score   support

           0       0.73      0.72      0.73       135
           1       0.74      0.75      0.74       141

    accuracy                           0.74       276
   macro avg       0.74      0.74      0.74       276
weighted avg       0.74      0.74      0.74       276

Logistic Regression Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed: 0.8297
Confusion Matrix:
[[103  32]
 [ 15 126]]
Classification Report Logistic Regression with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed:
              precision    recall  f1-score   support

           0       0.87      0.76      0.81       135
           1       0.80      0.89      0.84       141

    accuracy                           0.83       276
   macro avg       0.84      0.83      0.83       276
weighted avg       0.83      0.83      0.83       276

Random Forest Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed: 0.8406
Confusion Matrix:
[[104  31]
 [ 13 128]]
Classification Report Random Forest with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed:
              precision    recall  f1-score   support

           0       0.89      0.77      0.83       135
           1       0.81      0.91      0.85       141

    accuracy                           0.84       276
   macro avg       0.85      0.84      0.84       276
weighted avg       0.85      0.84      0.84       276
```

Performance evaluation

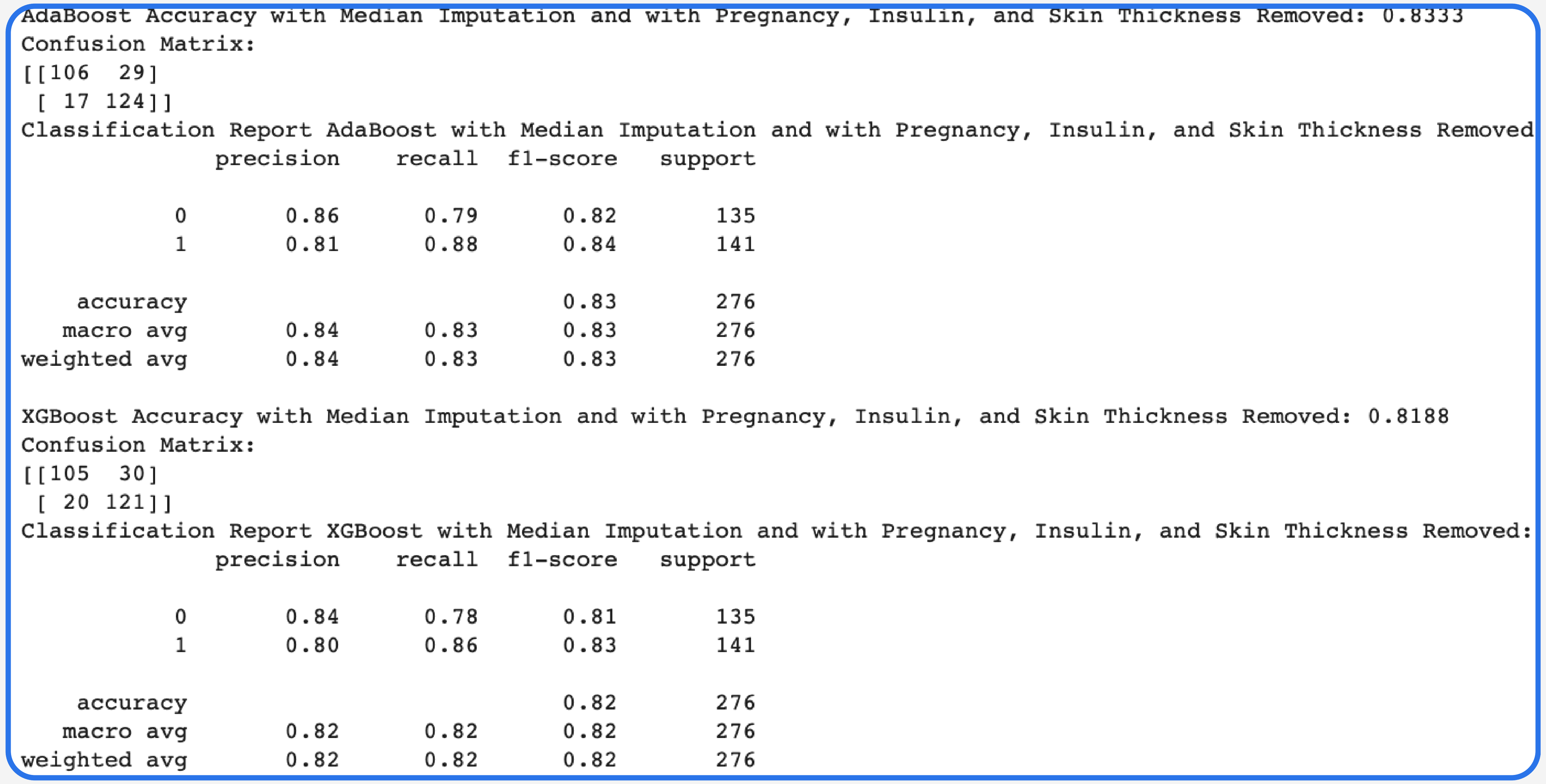
Refactored Code



```
Naive Bayes Accuracy with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed: 0.8152
Confusion Matrix:
[[104  31]
 [ 20 121]]
Classification Report Naive Bayes with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed:
              precision    recall  f1-score   support

           0       0.84      0.77      0.80       135
           1       0.80      0.86      0.83       141

    accuracy                           0.82       276
   macro avg       0.82      0.81      0.81       276
weighted avg       0.82      0.82      0.81       276

KNN Accuracy with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed: 0.8152
Confusion Matrix:
[[101  34]
 [ 17 124]]
Classification Report KNN with KNN Imputation and with Pregnancy, Insulin, and Skin Thickness Removed:
              precision    recall  f1-score   support

           0       0.86      0.75      0.80       135
           1       0.78      0.88      0.83       141

    accuracy                           0.82       276
   macro avg       0.82      0.81      0.81       276
weighted avg       0.82      0.82      0.81       276

SVM Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed: 0.8370
Confusion Matrix:
[[104  31]
 [ 14 127]]
Classification Report SVM with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed:
              precision    recall  f1-score   support

           0       0.88      0.77      0.82       135
           1       0.80      0.90      0.85       141

    accuracy                           0.84       276
   macro avg       0.84      0.84      0.84       276
weighted avg       0.84      0.84      0.84       276
```

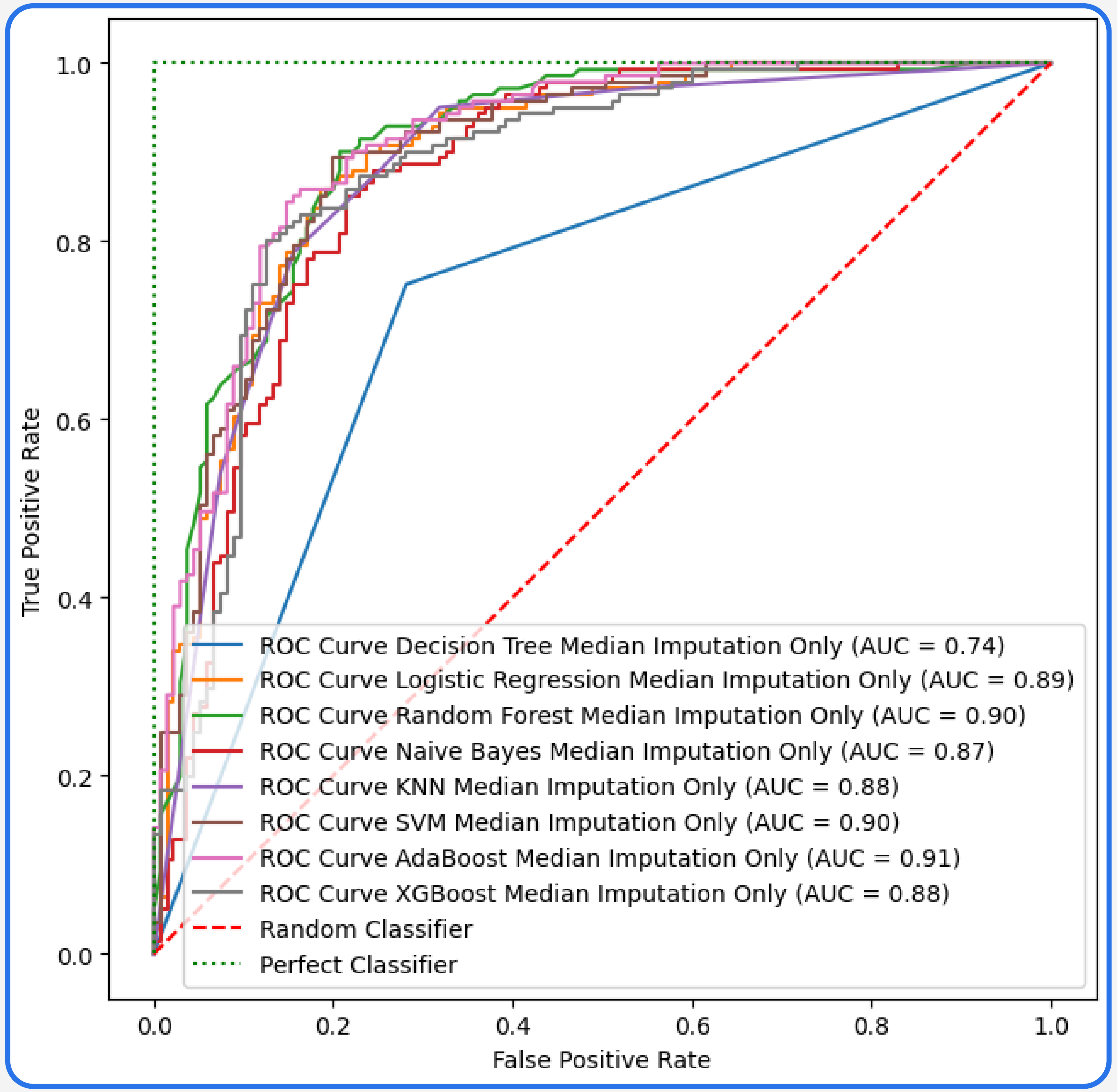
Performance evaluation

Refactored Code


```
AdaBoost Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed: 0.8333
Confusion Matrix:
[[106  29]
 [ 17 124]]
Classification Report AdaBoost with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed
              precision    recall  f1-score   support

           0       0.86      0.79      0.82       135
           1       0.81      0.88      0.84       141

    accuracy                           0.83       276
   macro avg       0.84      0.83      0.83       276
weighted avg       0.84      0.83      0.83       276

XGBoost Accuracy with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed: 0.8188
Confusion Matrix:
[[105  30]
 [ 20 121]]
Classification Report XGBoost with Median Imputation and with Pregnancy, Insulin, and Skin Thickness Removed:
              precision    recall  f1-score   support

           0       0.84      0.78      0.81       135
           1       0.80      0.86      0.83       141

    accuracy                           0.82       276
   macro avg       0.82      0.82      0.82       276
weighted avg       0.82      0.82      0.82       276
```

Performance evaluation

Refactored Code


True Positive Rate
False Positive Rate
ROC Curve Decision Tree Median Imputation Only (AUC = 0.74)
ROC Curve Logistic Regression Median Imputation Only (AUC = 0.89)
ROC Curve Random Forest Median Imputation Only (AUC = 0.90)
ROC Curve Naive Bayes Median Imputation Only (AUC = 0.87)
ROC Curve KNN Median Imputation Only (AUC = 0.88)
ROC Curve SVM Median Imputation Only (AUC = 0.90)
ROC Curve AdaBoost Median Imputation Only (AUC = 0.91)
ROC Curve XGBoost Median Imputation Only (AUC = 0.88)
Random Classifier
Perfect Classifier