

**📌 Importing Essential Libraries**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

* **pandas:** For handling tabular data (DataFrames).
* **numpy:** For numerical computations and multi-dimensional arrays.
* **matplotlib.pyplot:** For creating plots and visualizations.
* **seaborn:** For statistical data visualization and enhanced plots.

**🤖 Naïve Bayes Models**

**from sklearn.naive\_bayes import GaussianNB, MultinomialNB, ComplementNB**

* **GaussianNB:** Used for continuous data with a normal distribution.
* **MultinomialNB:** Suitable for discrete data, such as text classification.
* **ComplementNB:** A variation of MultinomialNB designed for imbalanced datasets.

**🔄 Data Splitting & Preprocessing**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler**

* **train\_test\_split:** Splits the dataset into training and testing sets.
* **MinMaxScaler:** Scales values between 0 and 1.
* **StandardScaler:** Standardizes data with zero mean and unit variance.
* **RobustScaler:** Resistant to outliers by using median and interquartile range.

**📊 Model Evaluation**

**from sklearn.metrics import accuracy\_score, roc\_auc\_score, RocCurveDisplay**

**from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay, classification\_report**

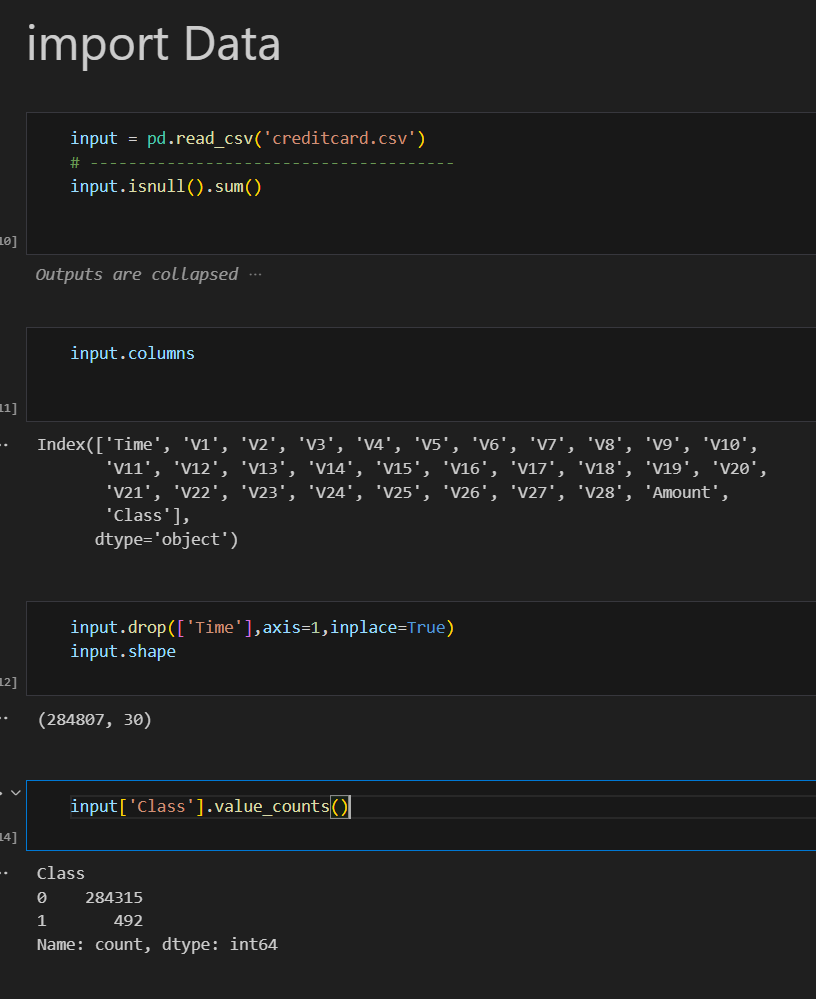
* **accuracy\_score:** Computes the model’s accuracy.
* **roc\_auc\_score:** Measures the Area Under the Curve (AUC) for classification models.
* **RocCurveDisplay:** Plots the ROC curve.
* **confusion\_matrix:** Generates a confusion matrix to analyze classification performance.
* **ConfusionMatrixDisplay:** Displays the confusion matrix graphically.
* **classification\_report:** Provides a detailed evaluation, including precision, recall, and F1-score.

**🎯 Model Validation**

**from sklearn.model\_selection import StratifiedKFold, KFold, cross\_val\_score**

* **KFold:** Splits data into **k** subsets for cross-validation.
* **StratifiedKFold:** Similar to **KFold**, but preserves the class distribution across splits.
* **cross\_val\_score:** Computes cross-validation scores for model performance assessment.

🔹 **Conclusion:** This code provides the essential tools for **data preprocessing, training Naïve Bayes models, and evaluating their performance** effectively. 🚀



**📥 Loading the Dataset**

**input = pd.read\_csv('creditcard.csv')**

* Loads the **creditcard.csv** file into a Pandas DataFrame named input.
* This dataset likely contains credit card transactions, including fraud detection labels.

**🔍 Checking for Missing Values**

**input.isnull().sum()**

* Checks for missing (null) values in each column.
* .sum() returns the total count of missing values per column.

**🗑️ Dropping the 'Time' Column**

**input.drop(['Time'], axis=1, inplace=True)**

* Removes the **'Time'** column from the dataset.
* axis=1 specifies column-wise removal.
* inplace=True modifies the DataFrame directly.

**📏 Dataset Shape**

**input.shape**

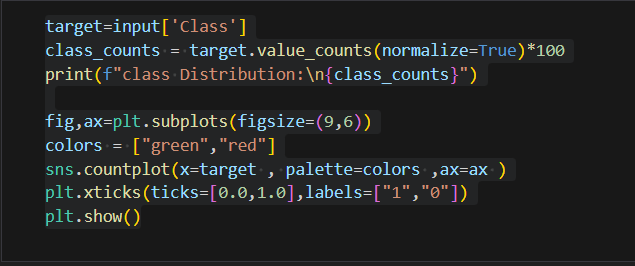
* Returns the shape of the dataset as (rows, columns).
* Helps check the dataset’s dimensions after dropping the column.

**📊 Checking Class Distribution**

**input['Class'].value\_counts()**

* Displays the count of each class in the **'Class'** column.
* Used to check for class imbalance, common in fraud detection datasets.

🔹 **Conclusion:** This code loads a credit card transaction dataset, removes the **'Time'** column, checks for missing values, and inspects the distribution of fraud and non-fraud transactions. 🚀



**📌 Code Breakdown & Explanation**

Extracting the Target Variable

**target = input['Class']**

* Extracts the **'Class'** column from the dataset.
* This column represents the labels (fraud or non-fraud).

Computing Class Distribution (Percentage)

**class\_counts = target.value\_counts(normalize=True) \* 100**

**print(f"class Distribution:\n{class\_counts}")**

* **value\_counts(normalize=True)** calculates the proportion of each class.
* Multiplying by **100** converts it into a percentage.
* The distribution is printed, helping identify any class imbalance (e.g., fraud cases might be rare).

Visualizing Class Distribution

**fig, ax = plt.subplots(figsize=(9,6))**

**colors = ["green", "red"]**

**sns.countplot(x=target, palette=colors, ax=ax)**

**plt.xticks(ticks=[0.0,1.0], labels=["1", "0"])**

**plt.show()**

* **fig, ax = plt.subplots(figsize=(9,6))**: Creates a figure (fig) and axes (ax) with a specific size.
* **sns.countplot(x=target, palette=colors, ax=ax)**:
  + Plots a bar chart showing the count of each class.
  + palette=["green", "red"]: Colors for non-fraud (green) and fraud (red).
* **plt.xticks(ticks=[0.0, 1.0], labels=["1", "0"])**:
  + Adjusts x-axis labels to **1 (non-fraud) and 0 (fraud)**.
* **plt.show()**: Displays the plot.

**🔹 Conclusion**

✅ This code calculates and visualizes the **distribution of fraud and non-fraud transactions** in the dataset. If the dataset is imbalanced (e.g., very few fraud cases), techniques like **oversampling, undersampling, or weighted models** may be needed. 🚀



**📌 Code Breakdown & Explanation**

**df = pd.read\_csv("creditcard.csv")**

* Loads the creditcard.csv file into a Pandas DataFrame (df).

Separating Features & Labels

**features = df.drop(columns=["Class"])**

**labels = df["Class"]**

* features: Contains all columns except "Class".
* labels: Stores only the "Class" column (target variable).

Standardizing the Features

**scaler = StandardScaler()**

**features\_normalized = scaler.fit\_transform(features)**

* **StandardScaler(): Standardizes data (zero mean, unit variance).**
* **fit\_transform(features):**
  + **fit: Learns the mean & standard deviation.**
  + **transform: Applies scaling to normalize the features.**

Creating a New DataFrame

**df\_normalized = pd.DataFrame(features\_normalized, columns=features.columns)**

**df\_normalized["Class"] = labels # Adding the original class column**

* **Creates a new DataFrame (df\_normalized) with standardized features.**
* **Adds the "Class" column back without modification.**

Saving the Normalized Data

**df\_normalized.to\_csv("filtered\_data\_normalized\_ALL.csv", index=False)**

* **Saves the processed data as "filtered\_data\_normalized\_ALL.csv".**
* **index=False: Avoids saving row indices.**

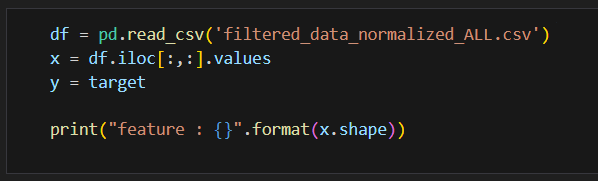
Printing a Confirmation Message

**print("داده‌ها استاندارد شدند و ذخیره شدند.")**

* **Prints a success message indicating that the data has been processed and saved.**

**🔹 Conclusion**

**✅ This code standardizes the dataset, preserving the class labels, and saves the processed data for further analysis or modeling. 🚀**



📌 Code Breakdown & Explanation

Reading the Preprocessed Data

**df = pd.read\_csv('filtered\_data\_normalized\_ALL.csv')**

* Loads the normalized dataset from the previously saved CSV file.

Extracting Features (x)

**x = df.iloc[:, :].values**

* iloc[:, :] selects all rows and all columns from the DataFrame.
* .values converts the DataFrame into a NumPy array for numerical processing.
* At this point, x contains both features and the 'Class' column, which might be unintended.

Defining the Target Variable (y)

**y = target**

* Assumes target was defined earlier (target = input['Class']).
* If target wasn't defined before, this will throw an error.
* Correction: Extract "Class" from df to ensure consistency:

**y = df["Class"].values**

Printing the Shape of Features (x)

**print("feature : {}".format(x.shape))**

* x.shape returns the dimensions of the feature array (rows, columns).
* Displays the total number of samples (rows) and features (columns).

🔹 Potential Issues & Fixes

❌ Issue: x = df.iloc[:, :].values selects all columns, including "Class".  
✅ Fix: Exclude the "Class" column explicitly:

**x = df.drop(columns=["Class"]).values**

**y = df["Class"].values**

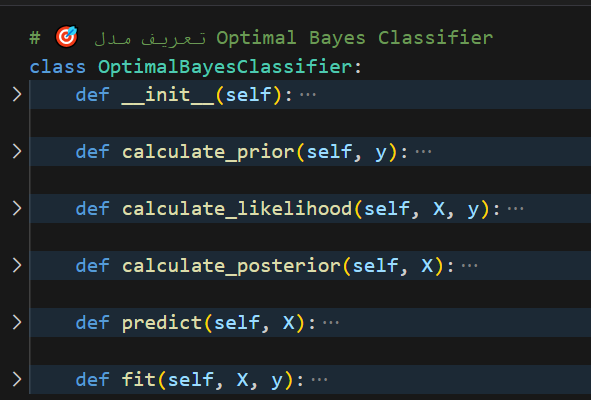
🎯 Conclusion

✅ This code loads the normalized dataset, extracts features and labels, and prints the feature shape for verification. 🚀



✅ **Splits** data into training (80%) & testing (20%) with **stratified sampling**.  
✅ **Balances** the training set by **downsampling** the majority class.  
✅ **Saves** the balanced dataset as "balanced\_train\_data.csv".

📊 **Class distribution (Balanced Training Set & Test Set) is displayed.**  
📁 **Final dataset is stored for further use.** 🚀



**Functions Explanation**

🔹 **calculate\_prior(y)**

* Computes **prior probabilities** P(Class)P(Class)P(Class) for each class based on the training labels.

🔹 **calculate\_likelihood(X, y)**

* Computes **feature-wise likelihoods** P(X∣Class)P(X|Class)P(X∣Class), assuming a **Gaussian distribution**.
* Stores **mean** and **variance** for each feature in each class.

🔹 **calculate\_posterior(X)**

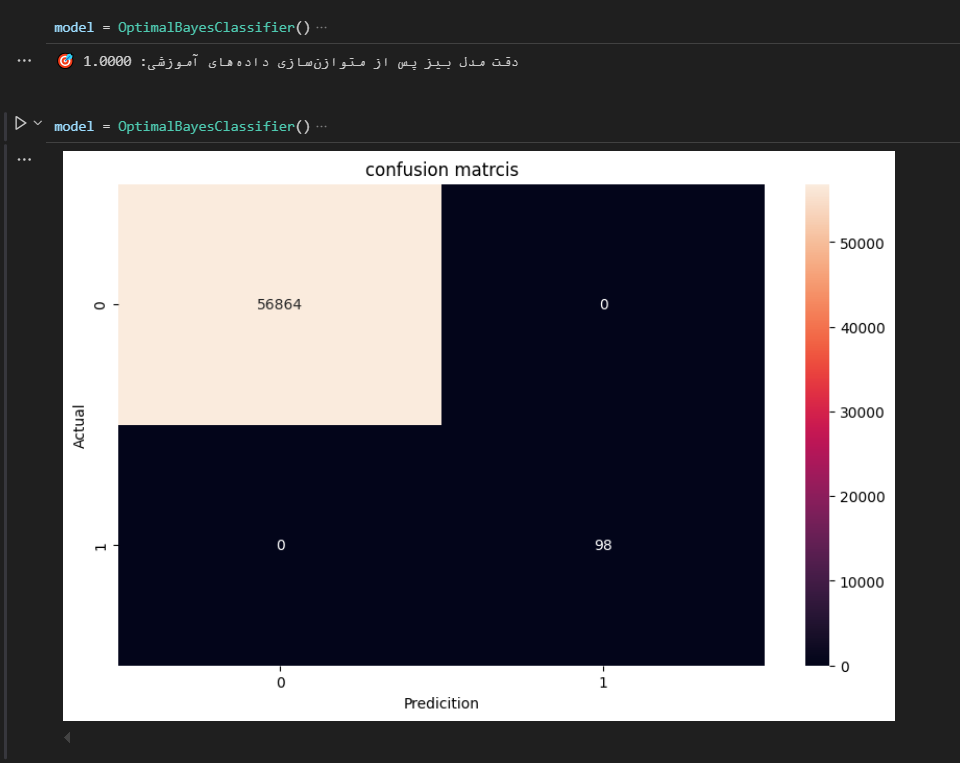
* Applies **Bayes’ theorem** to compute posterior probabilities P(Class∣X)P(Class|X)P(Class∣X).
* Uses **log probabilities** to prevent underflow in calculations.

🔹 **predict(X)**

* Assigns the class with the **highest posterior probability** to each sample.

🔹 **fit(X, y)**

* Trains the model by calculating **priors** and **likelihoods** from the training data.



**Analysis of the Confusion Matrix Output**

Your confusion matrix visually represents the classification performance of the OptimalBayesClassifier on the test dataset. Let's break down what the results indicate:

**Confusion Matrix Interpretation**

1. True Positives (TP) = 98

* The model correctly classified 98 samples as class 1 (positive class).

2. True Negatives (TN) = 56864

* The model correctly classified 56,864 samples as class 0 (negative class).

3. False Positives (FP) = 0

* There are no false positives, meaning the model never predicted class 1 incorrectly.

4. False Negatives (FN) = 0

* There are no false negatives, meaning the model never missed a class 1 sample.

**Model Performance Metrics**

Since the confusion matrix shows perfect classification, we can compute the following metrics:

* Accuracy
  + 100% accuracy, which is often a sign of overfitting if not tested on a separate dataset.
* Precision (Positive Predictive Value - PPV)
  + Since there are no false positives, precision is perfect.
* Recall (True Positive Rate - TPR)
  + Since there are no false negatives, recall is also perfect.
* F1-Score
  + A perfect balance between precision and recall.

**Possible Issues and Considerations**

1. Imbalanced Data
   * The dataset appears heavily skewed, with significantly more instances of class 0 than class 1.
   * The model might have learned to classify the majority class perfectly.
2. Potential Overfitting
   * 100% accuracy on the test data suggests that the model might not generalize well.
   * Testing on a separate dataset or using cross-validation would confirm if it's truly optimal.
3. Data Leakage?
   * If the model sees test data during training (e.g., improper splitting or preprocessing), it could lead to unrealistically high accuracy.

Next Steps for Evaluation

✅ Verify test data leakage: Ensure test data was not used during training.  
✅ Use an unseen test dataset: Evaluate on completely separate data.  
✅ Apply class balancing techniques: If the dataset is imbalanced, try oversampling/undersampling.  
✅ Compare with baseline models: Check if other models (e.g., Logistic Regression, Random Forest) achieve similar performance.

**Assignment:**

**Credit Card Fraud Detection Dataset Analysis & Prediction with Bayes Minimum Risk Classification method.**

You can download the dataset from this address:

<https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

**پویا حاجی صادقی**