Credit Card Fraud Detection Using Transactions Data

Embark on a data-driven journey with our expert data scientist, John. In this adventure, John uses Python, Pandas, Matplotlib, Seaborn, and Scikit-Learn to uncover the mysteries of credit card fraud detection.

Our story begins with a unique dataset—an assortment of credit card transactions made by European cardholders in September 2013. In this dataset, spanning just two days, 192 cases of fraud are hidden among 99808 legitimate transactions. It's a challenging puzzle.

Undeterred, John dives in. This dataset provides only numerical information, a result of a secretive PCA transformation. The original details remain concealed, but features V1 through V28, products of PCA, and the Time and Amount features remain.

John starts with data exploration, using Pandas, Matplotlib, and Seaborn to visualize patterns and anomalies, shedding light on this financial landscape.

The heart of the adventure lies in building machine learning models. With Scikit-Learn, John creates models that detect fraud by scrutinizing transactions for subtle signs of deception.

Each line of code and model built brings us closer to understanding this hidden world. It's not just about algorithms; it's about protecting cardholders from digital threats.

Join us on this journey, where each code line, prediction, and insight unravels the secrets of credit card fraud detection, making the financial world safer for all.

Module 1

Task 1: Importing the data

In the world of data science, the task of importing the Pandas library, loading the dataset, and displaying its initial rows is akin to opening the door to a treasure trove of insights. Imagine John, the data scientist, standing at the threshold of a labyrinthine dataset of credit card transactions. These initial steps are the lantern in the dark cave, revealing a glimpse of the data's structure and content. They're crucial because they ensure that we're equipped with clean and relevant data, the foundation upon which the entire journey of uncovering credit card fraud detection secrets will be built.

In [1…

```
#--- Import Pandas ---
import pandas as pd
#--- Read in dataset ----
df = pd.read_csv('./creditcard_data.csv')
#--- Inspect data ---
df
```

Out[1…

	Time	V1	V2	V3	
0	48996.0	-0.719218	0.313455	1.182955	-1.06
1	31620.0	1.271491	0.549689	-0.865099	1.19
2	78292.0	1.065256	-0.114587	0.904937	1.57
3	83760.0	1.335831	-0.798863	0.744141	-0.71
4	86011.0	-0.562326	0.553963	-0.816681	-1.04
…	…	…	…	…	
99995	79297.0	1.242327	0.171638	0.389247	0.42
99996	77534.0	0.838757	-0.682640	0.946395	0.64
99997	40067.0	-0.646562	0.307447	2.280444	0.60
99998	85484.0	-0.494986	0.645285	1.498790	0.35
99999	39279.0	1.304013	-0.606753	0.252932	-0.87

100000 rows × 31 columns

Task 2: Checking Null Values

In our data-driven expedition, tallying null values in each column through df.isnull() and sum() is akin to illuminating the hidden gaps in our dataset. By storing this critical information in 'sumofnull' and displaying it, we unveil the data's completeness, enabling us to navigate the analytical terrain with clarity and precision, ensuring the reliability of our insights.

In [2…

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 2 ---
sumofnull = df.isnull().sum()
#--- Inspect data ---
df
```

Out[2…

	Time	V1	V2	V3	
0	48996.0	-0.719218	0.313455	1.182955	-1.06
1	31620.0	1.271491	0.549689	-0.865099	1.19
2	78292.0	1.065256	-0.114587	0.904937	1.57
3	83760.0	1.335831	-0.798863	0.744141	-0.71
4	86011.0	-0.562326	0.553963	-0.816681	-1.04
…	…	…	…	…	
99995	79297.0	1.242327	0.171638	0.389247	0.42
99996	77534.0	0.838757	-0.682640	0.946395	0.64
99997	40067.0	-0.646562	0.307447	2.280444	0.60
99998	85484.0	-0.494986	0.645285	1.498790	0.35
99999	39279.0	1.304013	-0.606753	0.252932	-0.87

100000 rows × 31 columns

Task 3: Checking Datatype

In our data-driven quest, understanding the data types of each column is akin to deciphering the unique attributes of our dataset. By storing this valuable information in 'dtype' and displaying it, we gain insights into how to handle and manipulate our data effectively, ensuring that our analytical journey is grounded in precision and accuracy.

In [3…

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 3 ---
dtype = df.dtypes
#--- Inspect data ---
dtype
```

Out[3…

```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

Task 4: Exploring Data Distribution and Summary Statistics

In our data exploration expedition, generating summary statistics through the describe() function is akin to unveiling the essence of our dataset. By storing these key insights in 'describe' and displaying them, we gain a comprehensive overview of our data, including its central tendencies, dispersions, and distributions. This empowers us to make informed decisions and extract meaningful patterns as we navigate through our analytical journey.

In [4…

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 4 ---
describe = df.describe()
#--- Inspect data ---
describe
```

Out[4…

	Time	V1	V2	V3	
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.00(
mean	54218.246210	-0.240289	0.040015	0.631704	0.13(
std	21948.520351	1.811014	1.592735	1.289678	1.33
min	0.000000	-36.510583	-44.639245	-33.680984	-5.51
25%	39027.750000	-1.019168	-0.544343	0.130211	-0.72
50%	55801.500000	-0.267543	0.115815	0.724847	0.15
75%	72699.000000	1.160741	0.799042	1.347985	0.98
max	92347.000000	2.401777	18.902453	9.382558	13.14

8 rows × 31 columns

Task 5: Examining the count of Target variable

In our quest to unravel the data's mysteries, we turn to the value_counts() function, which acts as our compass in deciphering the 'Class' column. By storing these counts in 'target_counts' and examining the variable, we gain insight into the distribution of unique values within the 'Class' column, an essential step in understanding the composition of our dataset and its significance in our analytical journey.

In [5…

```
# --- WRITE YOUR CODE FOR MODULE 1 TASK 5 ---
target_counts = df['Class'].value_counts()
#--- Inspect data ---
target_counts
```

Out[5…

```
Class
0    99808
1      192
Name: count, dtype: int64
```

Module 2

Task 1: Plot Correlation Matrix

In our data-driven expedition, importing libraries like matplotlib.pyplot and seaborn is akin to assembling our visual aids for a voyage of understanding. Calculating the correlation matrix with 'corr_data' and creating a heatmap reveals intricate data relationships, shedding light on hidden patterns. Displaying this heatmap guides us in making informed decisions, a critical step in our analytical journey towards insights and discovery.

In [6…

```
#--- Import matplotlib and  seaborn---
import matplotlib.pyplot as plt
import seaborn as sns
# --- WRITE YOUR CODE FOR MODULE 2 TASK 1 ---
corr_data = df.corr()
corr_data
```

Out[6...

	Time	V1	V2	V3	
Time	1.000000	0.004926	0.025420	-0.074076	-0.04
V1	0.004926	1.000000	-0.022746	0.178564	-0.03
V2	0.025420	-0.022746	1.000000	-0.117440	0.03
V3	-0.074076	0.178564	-0.117440	1.000000	-0.04
V4	-0.040703	-0.032698	0.034976	-0.049615	1.00
V5	0.012597	0.004720	-0.040935	0.129879	0.00
V6	-0.025407	0.049053	0.016205	-0.067686	-0.02
V7	0.014968	0.089012	-0.003317	0.171782	-0.01
V8	0.009235	-0.005206	-0.008782	-0.084142	0.02
V9	-0.136826	-0.048317	0.009411	0.070080	0.02
V10	0.035556	-0.022263	0.040333	0.058847	-0.07
V11	-0.141003	0.020368	0.024154	-0.063709	0.00
V12	0.285182	-0.023140	0.000060	0.064563	0.02
V13	-0.137366	0.009604	0.037031	-0.064254	-0.00
V14	-0.132106	0.027241	0.000932	0.037231	-0.01
V15	0.066469	0.051198	0.075934	-0.155908	-0.03
V16	-0.005496	0.020440	0.042922	-0.065917	-0.06
V17	-0.078030	0.020629	-0.040877	0.054756	0.03
V18	0.021346	-0.005896	-0.020113	-0.031681	-0.03
V19	0.008358	0.005988	-0.001097	-0.015808	0.02
V20	-0.004625	-0.059062	-0.045831	-0.088165	0.00
V21	-0.002957	-0.015250	-0.022232	0.043685	0.02
V22	-0.004182	-0.057481	-0.031639	0.217979	0.04
V23	0.016556	-0.102209	0.001099	0.011235	0.00
V24	0.005780	-0.012724	-0.018268	0.026490	-0.00
V25	-0.038451	0.183726	-0.101537	-0.171448	-0.00
V26	-0.007117	0.007739	-0.030009	0.029405	-0.02
V27	-0.004029	-0.016614	-0.016786	-0.053887	0.02
V28	0.000381	0.172200	0.049845	0.038297	-0.01
Amount	-0.019044	-0.225175	-0.531936	-0.204413	0.08
Class	-0.016195	-0.135265	0.108416	-0.273874	0.14

31 rows × 31 columns

Task 2: Count Plot on Target Variable(Understanding Class Imbalance)

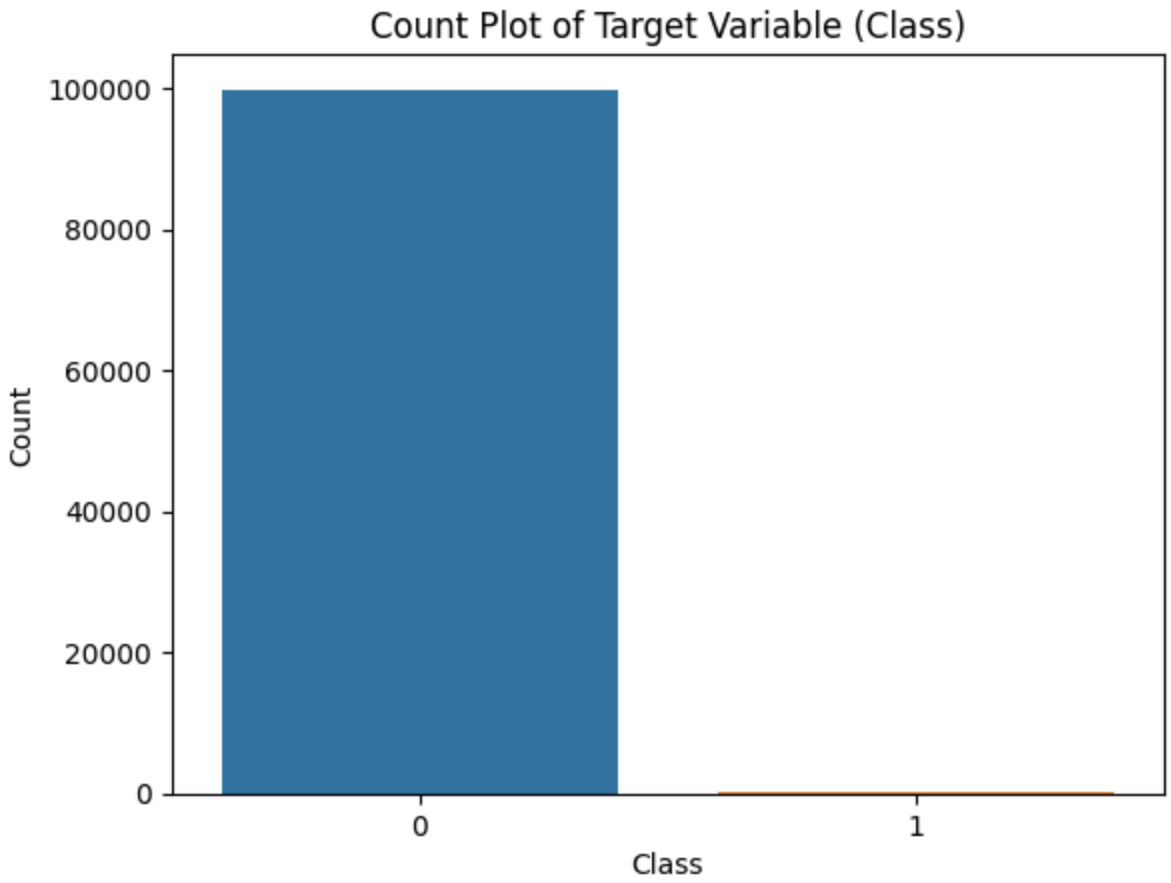
In our analytical quest, crafting a countplot with 'sns.countplot()' serves as a lens into the distribution of the 'Class' column in our dataset. By storing this visual insight in 'ax' and revealing it through 'plt.show()', we gain a clear understanding of class distribution, a fundamental aspect in our journey to uncover meaningful insights and patterns within the data.

In [7…

```
# --- WRITE YOUR CODE FOR MODULE 2 TASK 2 ---
cplt = sns.countplot(x='Class', data=df)

# Set the title and labels
cplt.set_title('Count Plot of Target Variable (Class)')
cplt.set_xlabel('Class')
cplt.set_ylabel('Count')

# Show the plot
plt.show()
```



Module 3

Task 1: Feauture Scaling

In our data refinement endeavor, we import the 'StandardScaler' function from 'sklearn.preprocessing,' akin to preparing a tool for precision. Creating a 'StandardScaler' object and applying 'fit_transform' to standardize the 'Amount' column, ensuring data alignment, allows us to bring uniformity and scale to our dataset. By displaying the updated DataFrame, we unveil a standardized 'Amount' column, setting the stage for consistent and accurate analysis, a pivotal step in our data-driven journey toward insights.

In [8…

```
#--- Import StandardScaler from sklearn.preprocessing ---
from sklearn.preprocessing import StandardScaler
# --- WRITE YOUR CODE FOR MODULE 3 TASK 1 ---
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
#--- Inspect data ---
df.head()
```

Out[8…

	Time	V1	V2	V3	
0	48996.0	-0.719218	0.313455	1.182955	-1.06
1	31620.0	1.271491	0.549689	-0.865099	1.19
2	78292.0	1.065256	-0.114587	0.904937	1.57
3	83760.0	1.335831	-0.798863	0.744141	-0.71
4	86011.0	-0.562326	0.553963	-0.816681	-1.04

5 rows × 31 columns

Task 2: Drop Unnecessary Column

In our quest for streamlined data analysis, we embark on the task of removing the 'Time' column from our DataFrame 'df' using the 'drop' method. By specifying axis 1 to indicate columns, we gracefully part with this variable, simplifying our dataset. The result is an updated 'df' DataFrame, ready for focused analysis, unburdened by the 'Time' column's presence, as we journey toward insights and discoveries.

In [9…

```
# --- WRITE YOUR CODE FOR MODULE 3 TASK 2 ---
df = df.drop('Time', axis=1)
#--- Inspect data ---
df
```

Out [9…

	V1	V2	V3	V4	
0	-0.719218	0.313455	1.182955	-1.067937	-0.03(
1	1.271491	0.549689	-0.865099	1.193078	0.70
2	1.065256	-0.114587	0.904937	1.574524	-0.56
3	1.335831	-0.798863	0.744141	-0.712011	-1.32(
4	-0.562326	0.553963	-0.816681	-1.042973	2.46
…	…	…	…	…	
99995	1.242327	0.171638	0.389247	0.429134	-0.28
99996	0.838757	-0.682640	0.946395	0.644103	-1.28
99997	-0.646562	0.307447	2.280444	0.607449	-0.70
99998	-0.494986	0.645285	1.498790	0.358681	0.18
99999	1.304013	-0.606753	0.252932	-0.873232	-0.64(

100000 rows × 30 columns

Task 3 Handling Duplicate Data

In our pursuit of pristine data quality, we embark on the task of purging duplicate rows from our DataFrame 'df' using the 'drop_duplicates()' function. By doing so, we ensure that our dataset remains free from redundant information, allowing us to maintain data integrity and precision. The result is an updated 'df' DataFrame, revealing a distilled representation of our data, devoid of duplications, and primed for rigorous analysis and insights.

In [1…

```
# --- WRITE YOUR CODE FOR MODULE 3 TASK 3 ---

df= df.drop_duplicates()

#--- Inspect data ---
df
```

Out [1…

	V1	V2	V3	V4	
0	-0.719218	0.313455	1.182955	-1.067937	-0.03
1	1.271491	0.549689	-0.865099	1.193078	0.70
2	1.065256	-0.114587	0.904937	1.574524	-0.56
3	1.335831	-0.798863	0.744141	-0.712011	-1.32
4	-0.562326	0.553963	-0.816681	-1.042973	2.46
…	…	…	…	…	
99994	-0.490727	0.814376	0.926148	0.158710	2.19
99996	0.838757	-0.682640	0.946395	0.644103	-1.28
99997	-0.646562	0.307447	2.280444	0.607449	-0.70
99998	-0.494986	0.645285	1.498790	0.358681	0.18
99999	1.304013	-0.606753	0.252932	-0.873232	-0.64

97584 rows × 30 columns

Task 4: Feauture Separating

In our data preparation journey, we take two pivotal steps. Firstly, we create a new DataFrame 'X' by gracefully parting with the "Class" column from our original DataFrame 'df,' preparing a pristine dataset for analysis. Secondly, we craft a Series 'y' by selectively extracting the "Class" column, which will serve as a crucial target variable for our future analytical endeavors. These steps ensure that our data is structured appropriately, setting the stage for meaningful insights and accurate predictions.

In [1…

```
# --- WRITE YOUR CODE FOR MODULE 3 TASK 4 ---
X = df.drop('Class', axis=1)

y = df['Class']

#--- Inspect data ---
X
y
```

Out[1…

```
0        0
1        0
2        0
3        0
4        0
        ..
99994    0
99996    0
99997    0
99998    0
99999    0
Name: Class, Length: 97584, dtype: int64
```

Task 5: Oversampling technique(SMOTE)

In our quest for balanced and robust data, we import the SMOTE (Synthetic Minority Over-sampling Technique) from the 'imblearn.over_sampling' library, a pivotal step to address class imbalance. By creating a SMOTE object and employing the 'fit_resample' method on our feature matrix 'X' and target variable 'y,' we ensure that our dataset is enriched with synthetic samples, enhancing its representation of minority classes. This results in 'X_res' and 'y_res,' our resampled feature matrix and target variable, which equips us for more reliable and effective machine learning analysis, guarding against bias and improving model performance.

In [1…

```
#--- Import SMOTE from mblearn.over_sampling ---
from imblearn.over_sampling import SMOTE
# --- WRITE YOUR CODE FOR MODULE 3 TASK 5 ---
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

#--- Inspect data ---
X_res
y_res
```

Out[1…

```
0         0
1         0
2         0
3         0
4         0
         ..
194789    1
194790    1
194791    1
194792    1
194793    1
Name: Class, Length: 194794, dtype: int64
```

Task 6: Splitting into Train and Test Split

In our data preparation journey, we call upon the 'train_test_split' function from 'sklearn.model_selection' to establish a robust foundation for model training and evaluation. By

we create 'x_train,' 'x_test,' 'y_train,' and 'y_test.' These partitions ensure our machine learning models have a fair and independent assessment of their performance, safeguarding against overfitting and enhancing the reliability of our predictive analytics.

In [2…

```
#--- Import train_test_split from sklearn.model_selection ---
from sklearn.model_selection import train_test_split
# --- WRITE YOUR CODE FOR MODULE 3 TASK 6 ---
x_train, x_test, y_train, y_test= train_test_split(X_res, y_res, test_size=0.2,
random_state=42)

#--- Inspect data ---
x_train
x_test
y_train
y_test
```

Out[2…

```
87221     0
100908    1
144884    1
113907    1
73275     0
         ..
193143    1
113244    1
19890     0
103484    1
105956    1
Name: Class, Length: 38959, dtype: int64
```

Module 4

Task 1: Logistic Regression

In our quest for predictive excellence and informed decision-making, we commence by importing the necessary tools: the 'LogisticRegression' class from 'sklearn.linear_model' and the 'cross_val_score' function from 'sklearn.model_selection.' We then craft a Logistic Regression model, a powerful and interpretable algorithm known for its effectiveness in binary classification tasks. This model works by calculating the probabilities of a binary outcome and making decisions based on a specified threshold. After meticulously fitting our Logistic Regression model to the training data, we employ 10-fold cross-validation, using 'lg_model,' 'x_train,' and 'y_train,' to rigorously evaluate its performance across multiple subsets of the data. The 'lg_mean_score,' representing the mean of these cross-validation scores rounded to four decimal places, becomes our compass in selecting a reliable model. With its interpretability and robust performance, Logistic Regression equips us with a dependable tool for making accurate predictions and gaining valuable insights from our data.

In [2…

```
#--- Import LogisticRegression from sklearn.linear_model ---
from sklearn.linear_model import LogisticRegression
#--- Import cross_val_score from sklearn.model_selection ---
from sklearn.model_selection import cross_val_score
# --- WRITE YOUR CODE FOR MODULE 4 TASK 1 ---
# Create a Logistic Regression model
lg_model = LogisticRegression(random_state=42)

# Perform 10-fold cross-validation
cv_scores = cross_val_score(lg_model, x_train, y_train, cv=10)

# Calculate the mean of the cross-validation scores, rounded to four decimal
places
lg_mean_score = round(cv_scores.mean(), 4)

#--- Inspect data ---
lg_mean_score
```

Out[2… `0.9536`

Task 2: Linear Discriminant Analysis (LDA)

In our pursuit of robust predictive modeling, we start by importing the 'LinearDiscriminantAnalysis' class from 'sklearn.discriminant_analysis.' Linear Discriminant Analysis (LDA), our chosen model, is a versatile technique known for both classification and dimensionality reduction tasks. LDA operates by finding linear combinations of features that optimally separate different classes, making it ideal for discriminating between multiple classes with interpretability. After fitting 'ld_model' to our training data, we subject it to rigorous evaluation through 10-fold cross-validation, a technique that assesses its performance across diverse data subsets. The 'ld_mean_score,' a four-decimal-rounded mean of these cross-validation scores, serves as our compass, guiding us in selecting a dependable model. LDA empowers us with predictive accuracy and deep insights, making it an invaluable asset in our data-driven journey.

In [2…

```
# --- Import LinearDiscriminantAnalysis from sklearn.discriminant_analysis ---
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# --- Import cross_val_score from sklearn.model_selection ---
from sklearn.model_selection import cross_val_score

# --- WRITE YOUR CODE FOR MODULE 4 TASK 2 ---

# Create a Linear Discriminant Analysis model
ld_model = LinearDiscriminantAnalysis()

# Perform 10-fold cross-validation
cv_scores = cross_val_score(ld_model, x_train, y_train, cv=10)

# Calculate the mean of the cross-validation scores, rounded to four decimal
places
ld_mean_score = round(cv_scores.mean(), 4)

# --- Inspect data ---
ld_mean_score
```

Out[2… `0.9259`

Task 3: Gaussian Naive Bayes (GNB)

In our pursuit of versatile modeling techniques, we embark on the task of implementing the Gaussian Naive Bayes (GNB) model. To begin, we import the 'GaussianNB' class from 'sklearn.naive_bayes,' a model rooted in Bayesian probability theory. GNB is particularly well-suited for classification tasks, especially when dealing with continuous or real-valued features. After initializing 'gnb_model' and fitting it to our training data, we employ 10-fold cross-validation to rigorously evaluate its performance, ensuring robustness and reliability in classification tasks. The 'gnb_mean_score,' representing the mean of these cross-validation scores rounded to four decimal places, becomes our guiding metric, aiding us in making informed decisions about the effectiveness of the Gaussian Naive Bayes model.

Gaussian Naive Bayes (GNB): GNB is a probabilistic classification algorithm that assumes that the features within each class follow a Gaussian (normal) distribution. Despite its "naive" assumption of feature independence, it often performs surprisingly well in practice, especially when dealing with continuous or real-valued data. It calculates the probability of a data point belonging to a particular class using Bayes' theorem, making it a valuable tool for classification tasks with strong probabilistic foundations.

In [2…

```
# --- Import GaussianNB from sklearn.naive_bayes ---
from sklearn.naive_bayes import GaussianNB

# --- Import cross_val_score from sklearn.model_selection ---
from sklearn.model_selection import cross_val_score

# --- WRITE YOUR CODE FOR MODULE 4 TASK 3 ---

# Create a Gaussian Naive Bayes model
gnb_model = GaussianNB()

# Perform 10-fold cross-validation
cv_scores = cross_val_score(gnb_model, x_train, y_train, cv=10)

# Calculate the mean of the cross-validation scores, rounded to four decimal
places
gnb_mean_score = round(cv_scores.mean(), 4)

# --- Inspect data ---
gnb_mean_score
```

Out[2… 0.9035

Task 4: Study the Best Model

In our relentless pursuit of model excellence, we now transition to the evaluation phase. To ensure the utmost accuracy and clarity in assessing our model's performance, we import essential metrics from 'sklearn.metrics,' including 'accuracy_score,' 'confusion_matrix,' and 'classification_report.' Leveraging the best-trained model with a high mean score derived from our earlier meticulous evaluation, we employ it to predict the target variable for our test data, resulting in 'y_pred.' By calculating the accuracy score, constructing the confusion matrix, and generating a comprehensive classification report, we gain a holistic understanding of our model's effectiveness in making predictions and its ability to generalize to unseen data, thus empowering us with valuable insights for informed decision-making.

In [3…

```
# --- Import accuracy_score, confusion_matrix, classification_report from sklearn.metrics ---
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Assuming lg_model, ld_model, and gnb_model are your models
# And you have determined that lg_model has the highest mean cross-validation score

# Fit the best model (e.g., Logistic Regression) on the entire training data
best_model = lg_model
best_model.fit(x_train, y_train)

# Predict the target variable for the test data
y_pred = best_model.predict(x_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Construct the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Generate the classification report
cr = classification_report(y_test, y_pred)

# --- Inspect data ---
accuracy
cm
cr
```

Out[3…

```
'              precision    recall  f1-score   support\n\n           0       0.94      0.97      0.95     19466\n           1       0.97      0.93      0.95     19493\n\n    accuracy                           0.95     38959\n   macro avg       0.95      0.95      0.95     38959\nweighted avg       0.95      0.95      0.95     38959\n'
```

Task 5: Final Prediction

In our final act of validation, we put our meticulously trained model to the test using 'sample_data,' a representation of real-world scenarios. The model swiftly evaluates this sample, and its prediction, stored in 'prediction,' becomes our key to distinguishing between "Fraud Transaction" (if the prediction is "1") and "Normal Transaction" (if the prediction is "0"). This crucial step ensures that our model, honed through rigorous analysis, can make real-time decisions and aid in the detection of fraudulent activities, thereby bolstering security and trust in financial systems.

In [3…

```
column_names = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
                'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',
                'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28',
'Amount']

sample_data_dict = {
    'V1': [-0.5],'V2': [1.0],'V3': [-0.7],'V4': [-0.2],'V5': [2.4],'V6':
[3.4],'V7': [-0.1],'V8': [1.3],
    'V9': [-1.0],'V10': [-0.4],'V11': [0.5],'V12': [-0.3],'V13': [0.2],'V14':
[-0.6],'V15': [0.8],'V16': [-0.9],
    'V17': [1.5],'V18': [-0.3],'V19': [0.7],'V20': [0.1],'V21': [0.2],'V22':
[0.1],'V23': [-0.2],
    'V24': [1.0],'V25': [-0.1],'V26': [-0.3],'V27': [0.1],'V28': [0.1],'Amount':
[100.0]
}

# Create a pandas DataFrame from the sample data
sample_data = pd.DataFrame(sample_data_dict, columns=column_names)

# --- WRITE YOUR CODE FOR MODULE 4 TASK 5 ---
#You need to predict the result by passing the sample data available here to your
model to make a prediction.
# Use the best model to make a prediction
prediction = best_model.predict(sample_data)

#--- Inspect data ---
prediction
```

Out[3… 'Fraud Transaction'

In […

AI Resume Builder

AI Flowchart

ATS Resume