Introduction

The Credit Card Clients Dataset is a comprehensive financial dataset that provides crucial insights into credit card usage patterns and client behavior. Featuring a diverse array of variables, including demographic information, credit limits, payment history, and bill amounts, this dataset facilitates in-depth analysis of credit card performance and risk assessment. With a focus on client credit behavior and repayment tendencies, it serves as a valuable resource for financial institutions, researchers, and data scientists seeking to enhance their understanding of consumer credit dynamics. The dataset's richness enables the development of predictive models to assess creditworthiness and optimize financial strategies for risk management.

1 import Necessary Library

In [69]:

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

2 import Dataset

In [70]:

```
df = pd.read_csv("/kaggle/input/default-of-credit-card-clients-dataset/UCI_Credit_Card.csv"
```

3 Data Analysis

In [71]:

```
df.head()
```

Out[71]: ID LIMIT_BAL SEX EDUCATION MARRIAGE AGE PAY_0 PAY_2 PAY_3 PAY_4 ... BILL_AMT4

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT1
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0

5 rows × 25 columns

In [72]:
```
df.tail()
```

Out[72]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BIL
29995	29996	220000.0	1	3	1	39	0	0	0	0	...	
29996	29997	150000.0	1	3	2	43	-1	-1	-1	-1	...	
29997	29998	30000.0	1	2	2	37	4	3	2	-1	...	
29998	29999	80000.0	1	3	1	41	1	-1	0	0	...	
29999	30000	50000.0	1	2	1	46	0	0	0	0	...	

5 rows × 25 columns

In [73]:
```
df.columns
```

Out[73]:
```
Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'default.payment.next.month'],
      dtype='object')
```

In [74]:
```
df.shape
```

Out[74]:
```
(30000, 25)
```

In [75]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   ID                          30000 non-null  int64  
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64  
 3   EDUCATION                   30000 non-null  int64  
 4   MARRIAGE                    30000 non-null  int64  
 5   AGE                         30000 non-null  int64  
 6   PAY_0                       30000 non-null  int64  
 7   PAY_2                       30000 non-null  int64  
 8   PAY_3                       30000 non-null  int64  
 9   PAY_4                       30000 non-null  int64  
```

```
 10  PAY_5                       30000 non-null  int64
 11  PAY_6                       30000 non-null  int64
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

In [76]:

```
df.describe()
```

Out[76]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.485500	-0
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904	1
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8

8 rows × 25 columns

In [77]:

```
df.corr()
```

Out[77]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	
ID	1.000000	0.026179	0.018497	0.039177	-0.029079	0.018678	-0
LIMIT_BAL	0.026179	1.000000	0.024755	-0.219161	-0.108139	0.144713	-0
SEX	0.018497	0.024755	1.000000	0.014232	-0.031389	-0.090874	-0
EDUCATION	0.039177	-0.219161	0.014232	1.000000	-0.143464	0.175061	0
MARRIAGE	-0.029079	-0.108139	-0.031389	-0.143464	1.000000	-0.414170	0
AGE	0.018678	0.144713	-0.090874	0.175061	-0.414170	1.000000	-0
PAY_0	-0.030575	-0.271214	-0.057643	0.105364	0.019917	-0.039447	1
PAY_2	-0.011215	-0.296382	-0.070771	0.121566	0.024199	-0.050148	0
PAY_3	-0.018494	-0.286123	-0.066096	0.114025	0.032688	-0.053048	0
PAY_4	-0.002735	-0.267460	-0.060173	0.108793	0.033122	-0.049722	0
PAY_5	-0.022199	-0.249411	-0.055064	0.097520	0.035629	-0.053826	0

PAY_6	-0.020270	-0.235195	-0.044008	0.082316	0.034345	-0.048773	0
BILL_AMT1	0.019389	0.285430	-0.033642	0.023581	-0.023472	0.056239	0
BILL_AMT2	0.017982	0.278314	-0.031183	0.018749	-0.021602	0.054283	0
BILL_AMT3	0.024354	0.283236	-0.024563	0.013002	-0.024909	0.053710	0
BILL_AMT4	0.040351	0.293988	-0.021880	-0.000451	-0.023344	0.051353	0
BILL_AMT5	0.016705	0.295562	-0.017005	-0.007567	-0.025393	0.049345	0
BILL_AMT6	0.016730	0.290389	-0.016733	-0.009099	-0.021207	0.047613	0
PAY_AMT1	0.009742	0.195236	-0.000242	-0.037456	-0.005979	0.026147	-0
PAY_AMT2	0.008406	0.178408	-0.001391	-0.030038	-0.008093	0.021785	-0
PAY_AMT3	0.039151	0.210167	-0.008597	-0.039943	-0.003541	0.029247	-0
PAY_AMT4	0.007793	0.203242	-0.002229	-0.038218	-0.012659	0.021379	-0
PAY_AMT5	0.000652	0.217202	-0.001667	-0.040358	-0.001205	0.022850	-0
PAY_AMT6	0.003000	0.219595	-0.002766	-0.037200	-0.006641	0.019478	-0
default.payment.next.month	-0.013952	-0.153520	-0.039961	0.028006	-0.024339	0.013890	0

25 rows × 25 columns

In [78]:
```
df.ndim
```

Out[78]:
```
2
```

4 Data cleaning and Preprocessing

In [79]:
```
df.isnull().sum()
```

Out[79]:
```
ID                            0
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
PAY_0                         0
PAY_2                         0
PAY_3                         0
PAY_4                         0
PAY_5                         0
PAY_6                         0
BILL_AMT1                     0
BILL_AMT2                     0
BILL_AMT3                     0
BILL_AMT4                     0
BILL_AMT5                     0
BILL_AMT6                     0
PAY_AMT1                      0
PAY_AMT2                      0
PAY_AMT3                      0
PAY_AMT4                      0
PAY_AMT5                      0
PAY_AMT6                      0
```

```
default.payment.next.month    0
dtype: int64
```

In [80]:

```
df.isna().empty
```

Out[80]:

```
False
```

In [81]:

```
df.isna().sum()
```

Out[81]:

```
ID                            0
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
PAY_0                         0
PAY_2                         0
PAY_3                         0
PAY_4                         0
PAY_5                         0
PAY_6                         0
BILL_AMT1                     0
BILL_AMT2                     0
BILL_AMT3                     0
BILL_AMT4                     0
BILL_AMT5                     0
BILL_AMT6                     0
PAY_AMT1                      0
PAY_AMT2                      0
PAY_AMT3                      0
PAY_AMT4                      0
PAY_AMT5                      0
PAY_AMT6                      0
default.payment.next.month    0
dtype: int64
```

In [82]:

```
df['default.payment.next.month'].value_counts()
```

Out[82]:

```
default.payment.next.month
0    23364
1     6636
Name: count, dtype: int64
```

5| Data visualisation 📊 📉

EDA (Exploratory Data Analysis)

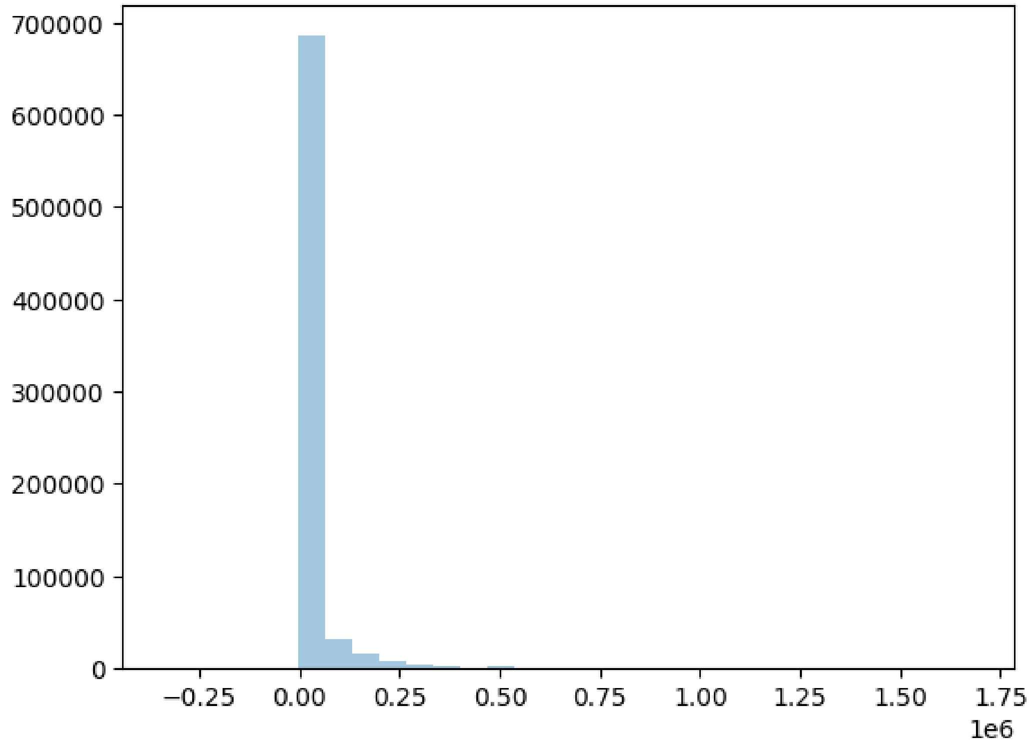
5.1 displot

In [83]:

```
sns.distplot(df, kde = False, bins=30)
plt.show()
```

```
/tmp/ipykernel_42/305216106.py:1: UserWarning:
```

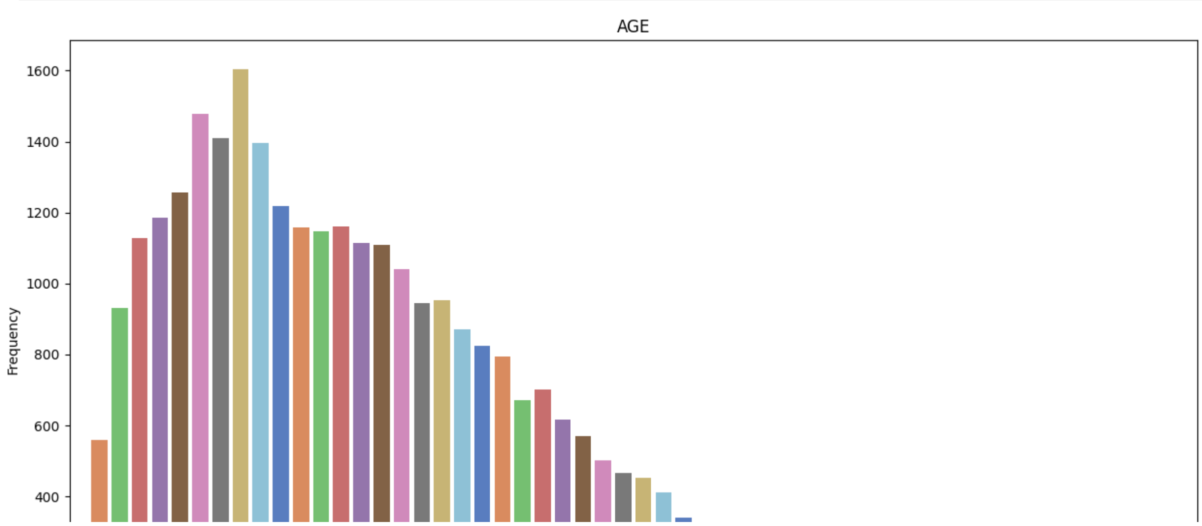
```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df, kde = False, bins=30)
```

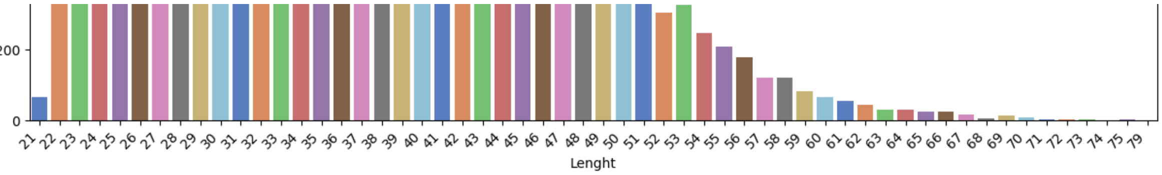


5.2 countplot

In [84]:

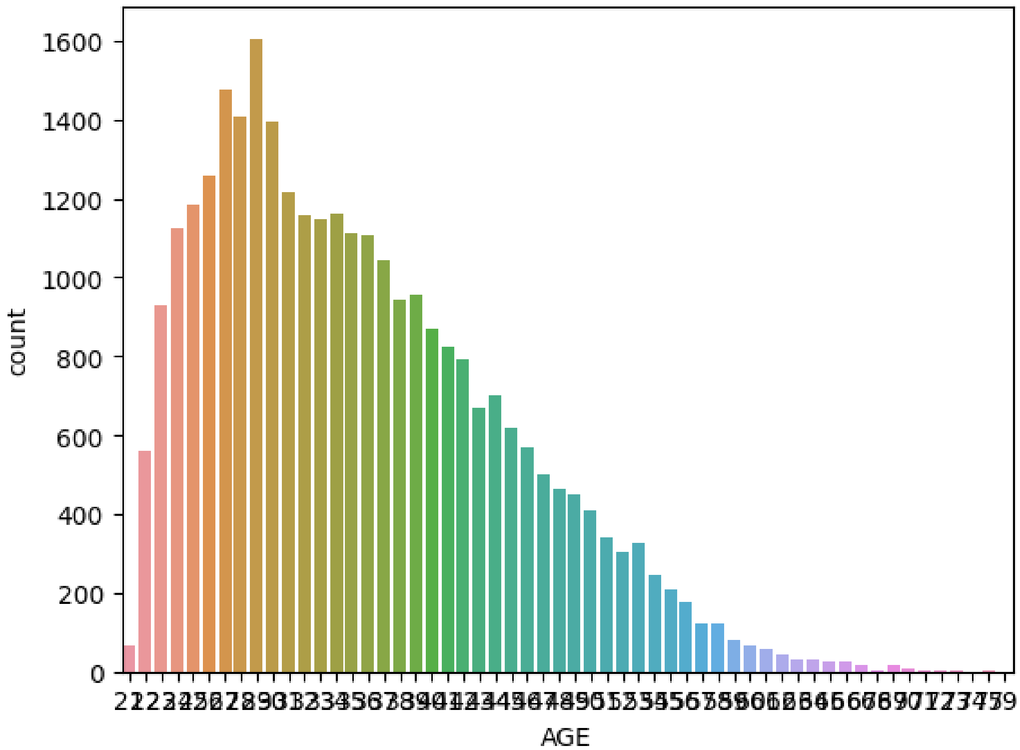
```python
# sepal_length
plt.figure(figsize=(15, 8))
sns.countplot(x='AGE', data=df, palette='muted')
plt.title('AGE')
plt.xlabel('Lenght')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.show()
```



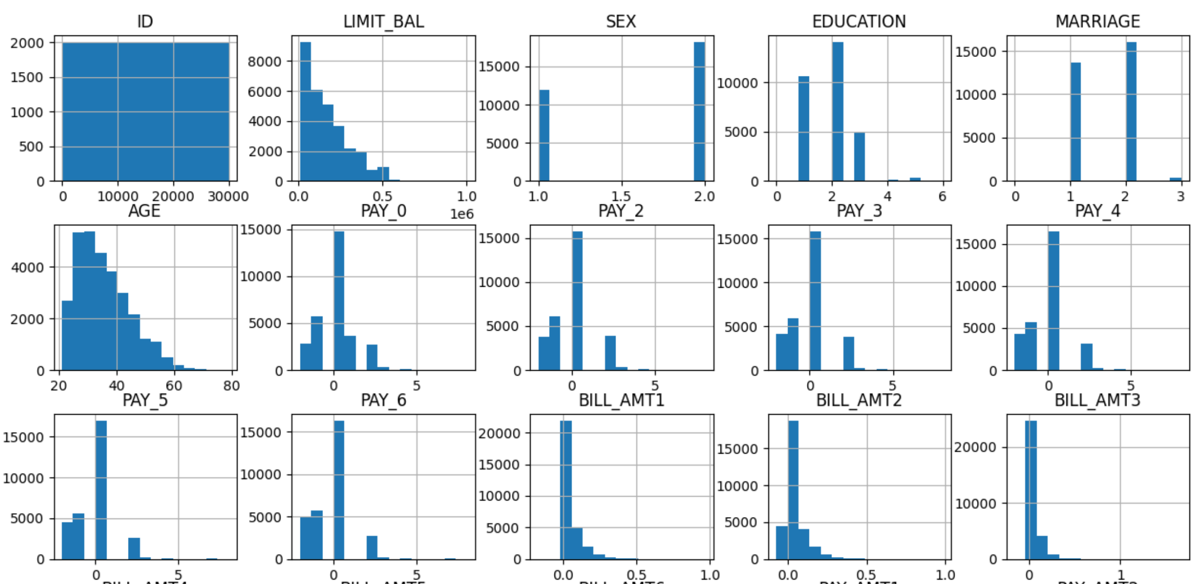


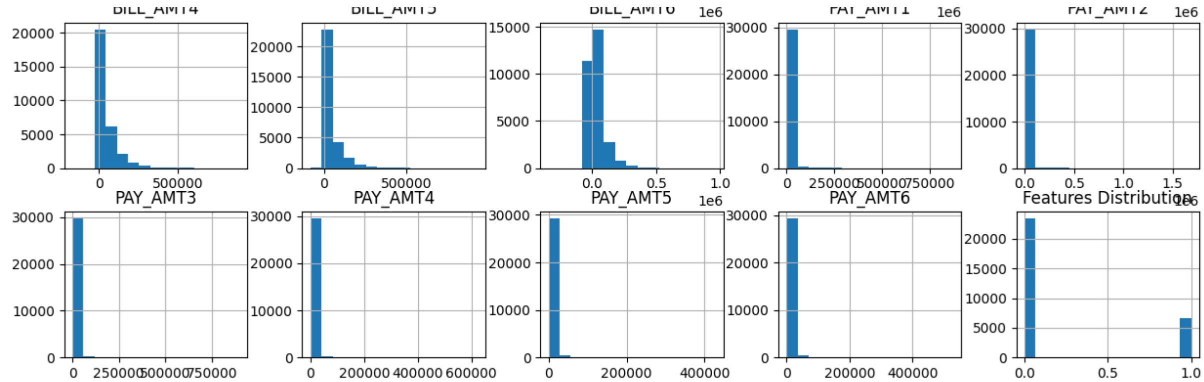
In [85]:
```
sns.countplot(data=df, x='AGE', y=None, hue=None,
              order=None, hue_order=None, orient=None,
              color=None, palette=None, saturation=.75,
              width=.8, dodge=True, ax=None)
```

Out[85]: <Axes: xlabel='AGE', ylabel='count'>



In [86]:
```
df.hist(figsize=(15,12),bins = 15)
plt.title("Features Distribution")
plt.show()
```





6 | Split the Dataset

In [87]:
```
from sklearn.model_selection import train_test_split
```

In [88]:
```
X = df.drop(['default.payment.next.month'], axis = 1)
```

In [89]:
```
y = df['default.payment.next.month']
```

In [90]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42
```

In [91]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[91]:
```
((22500, 24), (7500, 24), (22500,), (7500,))
```

7 | PCA (Principal Component Analysis)

In [92]:
```
from sklearn.decomposition import PCA
```

In [93]:
```
pca = PCA(n_components=2)
```

In [94]:
```
X_pca = pca.fit_transform(X)
```

In [95]:
```
X_pca[0]
```

Out[95]:
```
array([-166511.13375728,  -75548.89621183])
```

In [96]:
```
print("Explained Variance Ratio:")
print(pca.explained_variance_ratio_)
```

```
Explained Variance Ratio:
[0.60943217 0.2948671 ]
```

In [97]:
```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
X_ft = scaler.fit_transform(X)
```

In [98]:

```
X_ft[0]
```

Out[98]:

```
array([0.00000000e+00, 1.01010101e-02, 1.00000000e+00, 3.33333333e-01,
       3.33333333e-01, 5.17241379e-02, 4.00000000e-01, 4.00000000e-01,
       1.00000000e-01, 1.00000000e-01, 0.00000000e+00, 0.00000000e+00,
       1.49981727e-01, 6.91643226e-02, 8.67228923e-02, 1.60137756e-01,
       8.06480880e-02, 2.60978723e-01, 0.00000000e+00, 4.09081976e-04,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00])
```

In [99]:

```
X_ft[1]
```

Out[99]:

```
array([3.33344445e-05, 1.11111111e-01, 1.00000000e+00, 3.33333333e-01,
       6.66666667e-01, 8.62068966e-02, 1.00000000e-01, 4.00000000e-01,
       2.00000000e-01, 2.00000000e-01, 2.00000000e-01, 4.00000000e-01,
       1.48892434e-01, 6.78575089e-02, 8.78171337e-02, 1.63219937e-01,
       8.40739510e-02, 2.63484742e-01, 0.00000000e+00, 5.93732912e-04,
       1.11602161e-03, 1.61030596e-03, 0.00000000e+00, 3.78310691e-03])
```

⚙️ Machine Learning Algorithm

Algorithm 🔄

(1) KNN 🔄

In [100…

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

In [101…

```
knn_classifier = KNeighborsClassifier(n_neighbors=3)
```

In [102…

```
knn_classifier.fit(X_train, y_train)
```

Out[102…

```
KNeighborsClassifier(n_neighbors=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [103…

```
train_predictions = knn_classifier.predict(X_train)

train_accuracy1 = accuracy_score(y_train, train_predictions)
```

In [104…

```
test_predictions = knn_classifier.predict(X_test)

test_accuracy1 = accuracy_score(y_test, test_predictions)
```

In [105…

```
print(f"Training Accuracy: {train_accuracy1}")
```

```
print(f"Training Accuracy: {train_accuracy1}")
print(f"Testing Accuracy: {test_accuracy1}")
```

```
Training Accuracy: 0.8444
Testing Accuracy: 0.7334666666666667
```

(2) Naive Bayes classifier 🔄

In [106…
```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics
```

In [107…
```
# GaussianNB
```

In [108…
```
G_classifier = GaussianNB()
```

In [109…
```
G_classifier.fit(X_train, y_train)
```

Out[109…
```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [110…
```
train_predictions = G_classifier.predict(X_train)

train_accuracy21 = accuracy_score(y_train, train_predictions)
```

In [111…
```
test_predictions = G_classifier.predict(X_test)

test_accuracy21 = accuracy_score(y_test, test_predictions)
```

In [112…
```
print(f"Training Accuracy: {train_accuracy21}")
print(f"Testing Accuracy: {test_accuracy21}")
```

```
Training Accuracy: 0.3768444444444444
Testing Accuracy: 0.3817333333333333
```

In [113…
```
# BernoulliNB
```

In [114…
```
B_classifier = BernoulliNB()
```

In [115…
```
B_classifier.fit(X_train, y_train)
```

Out[115…
```
BernoulliNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

nbviewer.org.

In [116...
```
train_predictions = B_classifier.predict(X_train)

train_accuracy22 = accuracy_score(y_train, train_predictions)
```

In [117...
```
test_predictions = G_classifier.predict(X_test)

test_accuracy22 = accuracy_score(y_test, test_predictions)
```

In [118...
```
print(f"Training Accuracy: {train_accuracy22}")
print(f"Testing Accuracy: {test_accuracy22}")
```

```
Training Accuracy: 0.7717777777777778
Testing Accuracy: 0.3817333333333333
```

(3) Decision Tree 🔄

In [119...
```
from sklearn.tree import DecisionTreeClassifier
```

In [120...
```
clf = DecisionTreeClassifier()
```

In [121...
```
clf.fit(X_train, y_train)
```

Out[121...
```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [122...
```
train_predictions = clf.predict(X_train)

train_accuracy3 = accuracy_score(y_train, train_predictions)
```

In [123...
```
test_predictions = clf.predict(X_test)

test_accuracy3 = accuracy_score(y_test, test_predictions)
```

In [124...
```
print(f"Training Accuracy: {train_accuracy3}")
print(f"Testing Accuracy: {test_accuracy3}")
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.7298666666666667
```

(4) Random Forest 🔄

In [125...
```
from sklearn.ensemble import RandomForestClassifier
```

In [126...
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [127…
```
rf_classifier.fit(X_train, y_train)
```

Out[127…
```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [128…
```
train_predictions = rf_classifier.predict(X_train)

train_accuracy4 = accuracy_score(y_train, train_predictions)
```

In [129…
```
test_predictions = rf_classifier.predict(X_test)

test_accuracy4 = accuracy_score(y_test, test_predictions)
```

In [130…
```
print(f"Training Accuracy: {train_accuracy4}")
print(f"Testing Accuracy: {test_accuracy4}")
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.8172
```

(5) Boosting Algorithm 🔄

In [131…
```
from sklearn.ensemble import AdaBoostClassifier
```

In [132…
```
base_classifier = DecisionTreeClassifier(max_depth=1)
```

In [133…
```
adaboost_classifier = AdaBoostClassifier(base_classifier, n_estimators=50, random_state=4
```

In [134…
```
adaboost_classifier.fit(X_train, y_train)
```

Out[134…
```
AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
                   random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [135…
```
train_predictions = adaboost_classifier.predict(X_train)

train_accuracy5 = accuracy_score(y_train, train_predictions)
```

In [136…
```
test_predictions = adaboost_classifier.predict(X_test)

test_accuracy5 = accuracy_score(y_test, test_predictions)
```

In [137…

```
print(f"Training Accuracy: {train_accuracy5}")
print(f"Testing Accuracy: {test_accuracy5}")
```

```
Training Accuracy: 0.8185777777777777
Testing Accuracy: 0.8181333333333334
```

(6). Logistic Regression

In [138…

```
from sklearn import linear_model
```

In [139…

```
lrg = linear_model.LogisticRegression()
```

In [140…

```
lrg.fit(X_train, y_train)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWa
rning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[140…

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [141…

```
train_predictions = lrg.predict(X_train)

train_accuracy7 = accuracy_score(y_train, train_predictions)
```

In [142…

```
test_predictions = lrg.predict(X_test)

test_accuracy7 = accuracy_score(y_test, test_predictions)
```

In [143…

```
print(f"Training Accuracy: {train_accuracy7}")
print(f"Testing Accuracy: {test_accuracy7}")
```

```
Training Accuracy: 0.7772888888888889
Testing Accuracy: 0.7830666666666667
```

(7).Linear Regression

In [144…

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [145…

```
model = LinearRegression()
```

In [146…

```
model.fit(X_train, y_train)
```

```
model.fit(X_train, y_train)
```

Out[146…

```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [147…

```
train_predictions = clf.predict(X_train)

train_accuracy8 = accuracy_score(y_train, train_predictions)
```

In [148…

```
test_predictions = clf.predict(X_test)

test_accuracy8 = accuracy_score(y_test, test_predictions)
```

In [149…

```
print(f"Training Accuracy: {train_accuracy8}")
print(f"Testing Accuracy: {test_accuracy8}")
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.7298666666666667
```

(8).Gradient Boosting Machines (GBM)

In [150…

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [151…

```
model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, rando
```

In [152…

```
model.fit(X_train, y_train)
```

Out[152…

```
GradientBoostingClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [153…

```
train_predictions = model.predict(X_train)

train_accuracy9 = accuracy_score(y_train, train_predictions)
```

In [154…

```
test_predictions = model.predict(X_test)

test_accuracy9 = accuracy_score(y_test, test_predictions)
```

In [155…

```
print(f"Training Accuracy: {train_accuracy9}")
print(f"Testing Accuracy: {test_accuracy9}")
```

```
Training Accuracy: 0.8265333333333333
Testing Accuracy: 0.8216
```

(6).SVM

In [156…
```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

In [157…
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [158…
```
svm_classifier = SVC(kernel='linear', C=1.0)
```

In [159…
```
svm_classifier.fit(X_train, y_train)
```

Out[159…
```
SVC(kernel='linear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [160…
```
train_predictions = svm_classifier.predict(X_train)

train_accuracy6 = accuracy_score(y_train, train_predictions)
```

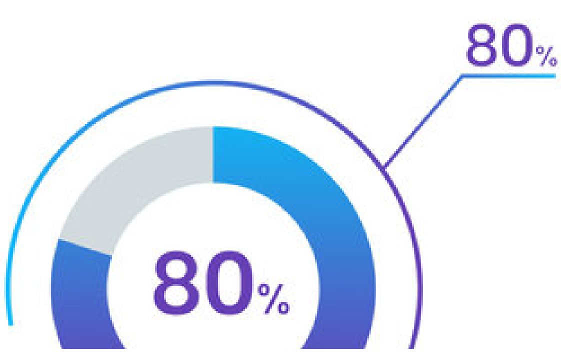
In [161…
```
test_predictions = svm_classifier.predict(X_test)

test_accuracy6 = accuracy_score(y_test, test_predictions)
```

In [162…
```
print(f"Training Accuracy: {train_accuracy6}")
print(f"Testing Accuracy: {test_accuracy6}")
```

```
Training Accuracy: 0.8096888888888889
Testing Accuracy: 0.8089333333333333
```

Decision Tree, Random Forest, SVM, Gradient Boosting Machines (GBM), Algorithm is the best accuracy

accuracy = 0.8



10 | Hierarchical Clustering

In [163…

```
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

In [164…

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [165…

```
linkage_matrix = linkage(X_scaled, method='ward')
```

In [166…

```
plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix, labels=df['default.payment.next.month'].values, orientation='t
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('default.payment.next.month')
plt.ylabel('Distance')
plt.show()
```