```
# !pip install catboost
```

```
# import packages
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_validate, GridSearchC
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classific

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
# set the aesthetic style of the plots
sns.set_style()
# filter warning messages
import warnings
warnings.filterwarnings('ignore')
```

Credit risk is associated with the possibility of a client failing to meet contractual loan obligations.

Minimizing the **risk** of **default** is a major concern for financial institutions. Banks and financial institutions firms are increasingly relying on **technology** to predict which clients are more prone to not honoring their debts.

Machine Learning models have been helping these companies to improve the accuracy of their credit risk analysis, to identify potential debtors in advance.

Here: we will build a model to predict the risk of client default for Nubank, a prominent Brazilian Fintech.

```
df_credit = pd.read_csv('http://dl.dropboxusercontent.com/s/xn2a4kzf0zer0xu/acqui
df_credit.head()
```

Show hidden output

Exploratory Data Analysis

What is the shape of the dataset? What are the features?

```
print('Number of rows: ', df_credit.shape[0])
print('Number of columns: ', df_credit.shape[1])

df_credit.info()
```

Show hidden output

`target_default` is a True/False feature and is the target variable we are trying to predict.

Next, let us find the percentage of missing values per feature.

```
print((df_credit.isnull().sum() * 100 / df_credit.shape[0]).sort_values(ascending
```

Show hidden output

`target_default` is our independent variable. 7.24% of our target is missing, so we can drop the missing values for this target.

Next, we drop several columns that are not useful for our analysis, like `target_fraud`

```
df_credit.dropna(subset=['target_default'], inplace=True)
df_credit.drop('target_fraud', axis=1, inplace=True)
```

```
print(df_credit.nunique().sort_values()) # number of unique observations per colu

df_credit.drop(labels=['channel', 'external_data_provider_credit_checks_last_2_ye
                       'shipping_zip_code', 'user_agent', 'profile_tags', 'market
```

Show hidden output

```
df_credit.head()
```

	target_default	score_1	score_2	score_
0	False	1Rk8w4Ucd5yR3KcqZzLdow==	IOVu8au3ISbo6+zmfnYwMg==	350.(
1	False	DGCQep2AE5QRkNCshIAlFQ==	SaamrHMo23l/3TwXOWgVzw==	370.(
2	True	DGCQep2AE5QRkNCshIAlFQ==	Fv28Bz0YRTVAT5kl1bAV6g==	360.(
3	False	1Rk8w4Ucd5yR3KcqZzLdow==	dCm9hFKfdRm7ej3jW+gyxw==	510.(
4	False	8k8UDR4Yx0qasAjkGrUZLw==	+CxEO4w7jv3QPI/BQbyqAA==	500.(

5 rows × 27 columns

Next, let us look as **summary statistics** and see if we are dealing with any outliers.

```
df_credit.describe()
```

Show hidden output

```
df_credit['reported_income'] = df_credit['reported_income'].replace(np.inf, np.na
df_credit.loc[df_credit['external_data_provider_email_seen_before'] == -999, 'ext
```

```
df_credit.describe()
```

Show hidden output

```
df_credit.info()
```

Show hidden output

Plot a histogram for each of the numerical features.

```
df_credit_numerical = df_credit[['score_3', 'risk_rate', 'last_amount_borrowed',
                                 'n_bankruptcies', 'n_defaulted_loans', 'n_accoun

nrows = 3
ncols = 4
fig, ax = plt.subplots(nrows=nrows, ncols=ncols, figsize=(10, 9))

r = 0
c = 0
for i in df_credit_numerical:
  sns.distplot(df_credit_numerical[i], bins=15,kde=False, ax=ax[r][c])
  if c == ncols - 1:
    r += 1
    c = 0
  else:
    c += 1

plt.show()
```

Show hidden output

MISSING VALUES.

We are filling mising values according to the particularities of each feature, as below:

Categorical variables will be filled with the most recurrent value.

- **Categorical** variables will be filled with the most recurrent value.
- **Numerical** variables will be filled with their median values.
- In the specific cases of `last_amount_borrowed`, `last_borrowed_in_months` and `n_issues` we'll fill the missing values with zero, as it is reasonable to believe that not every client would have values assigned to these variables.

```
df_credit_num = df_credit.select_dtypes(exclude='object').columns
df_credit_cat = df_credit.select_dtypes(include='object').columns

# fill missing values for "last_amount_borrowed", "last_borrowed_in_months" and "
df_credit['last_amount_borrowed'].fillna(value=0, inplace=True)
df_credit['last_borrowed_in_months'].fillna(value=0, inplace=True)
df_credit['n_issues'].fillna(value=0, inplace=True)

# fill missing values for numerical variables
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
imputer = imputer.fit(df_credit.loc[:, df_credit_num])
df_credit.loc[:, df_credit_num] = imputer.transform(df_credit.loc[:, df_credit_nu

# fill missing values for categorical variables
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imputer = imputer.fit(df_credit.loc[:, df_credit_cat])
df_credit.loc[:, df_credit_cat] = imputer.transform(df_credit.loc[:, df_credit_ca
```

We'll now preprocess the data, converting the categorical features into numerical values. `LabelEncoder` will be used for the binary variables while `get_dummies` will be used for the other categorical variables.

```
bin_var = df_credit.nunique()[df_credit.nunique() == 2].keys().tolist()
num_var = [col for col in df_credit.select_dtypes(['int', 'float']).columns.tolis
cat_var = [col for col in df_credit.select_dtypes(['object']).columns.tolist() if

df_credit_encoded = df_credit.copy()

# label encoding for the binary variables
le = LabelEncoder()
for col in bin_var:
  df_credit_encoded[col] = le.fit_transform(df_credit_encoded[col])

# encoding with get_dummies for the categorical variables
df_credit_encoded = pd.get_dummies(df_credit_encoded, columns=cat_var)

df_credit_encoded.head()
```

Show hidden output

Train-test Split for Machine Learning

```
X = df_credit_encoded.drop('target_default', axis=1)
y = df_credit_encoded['target_default']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s


# standardize and resample the training set
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)

rus = RandomUnderSampler()
X_train_rus, y_train_rus = rus.fit_resample(X_train, y_train)
```

Random Forest Classifier

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_rus, y_train_rus)
predictions = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.7695440389682984
```

```
# classification report
print(classification_report(y_test, predictions))

# confusion matrix
fig, ax = plt.subplots()
sns.heatmap(confusion_matrix(y_test, predictions, normalize='true'), annot=True,
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted Value')
ax.set_ylabel('Real Value')
```

Show hidden output

Decision Tree Classifier

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train_rus, y_train_rus)
y_pred = dtc.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# confusion matrix
fig, ax = plt.subplots()
sns.heatmap(confusion_matrix(y_test, y_pred, normalize='true'), annot=True, ax=ax)
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted Value')
ax.set_ylabel('Real Value')
```

Show hidden output

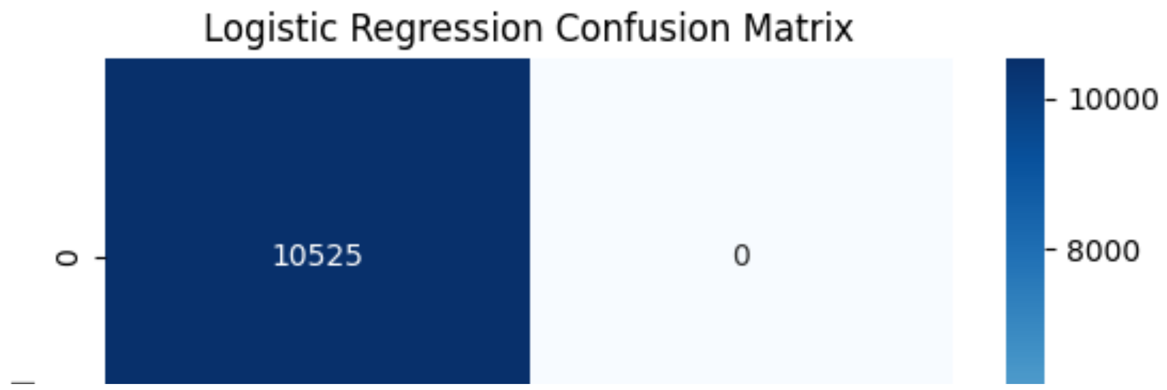
```
# ig = plt.figure(figsize=(20,20))
# tree.plot_tree(dtc, label='root', fontsize=9, class_names=['Low', 'High'], fillec
```

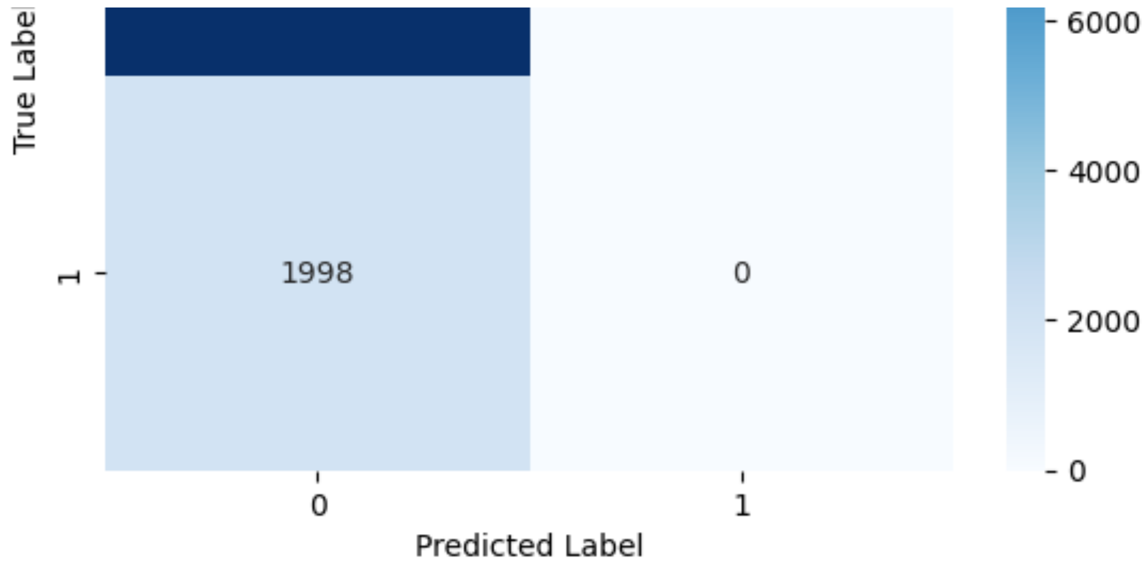
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report


# Assuming X_train_rus, y_train_rus, X_test, y_test are already defined

# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train_rus, y_train_rus)
y_pred_logreg = logreg.predict(X_test)

# KNN
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust n_neighbors
knn.fit(X_train_rus, y_train_rus)
y_pred_knn = knn.predict(X_test)
```

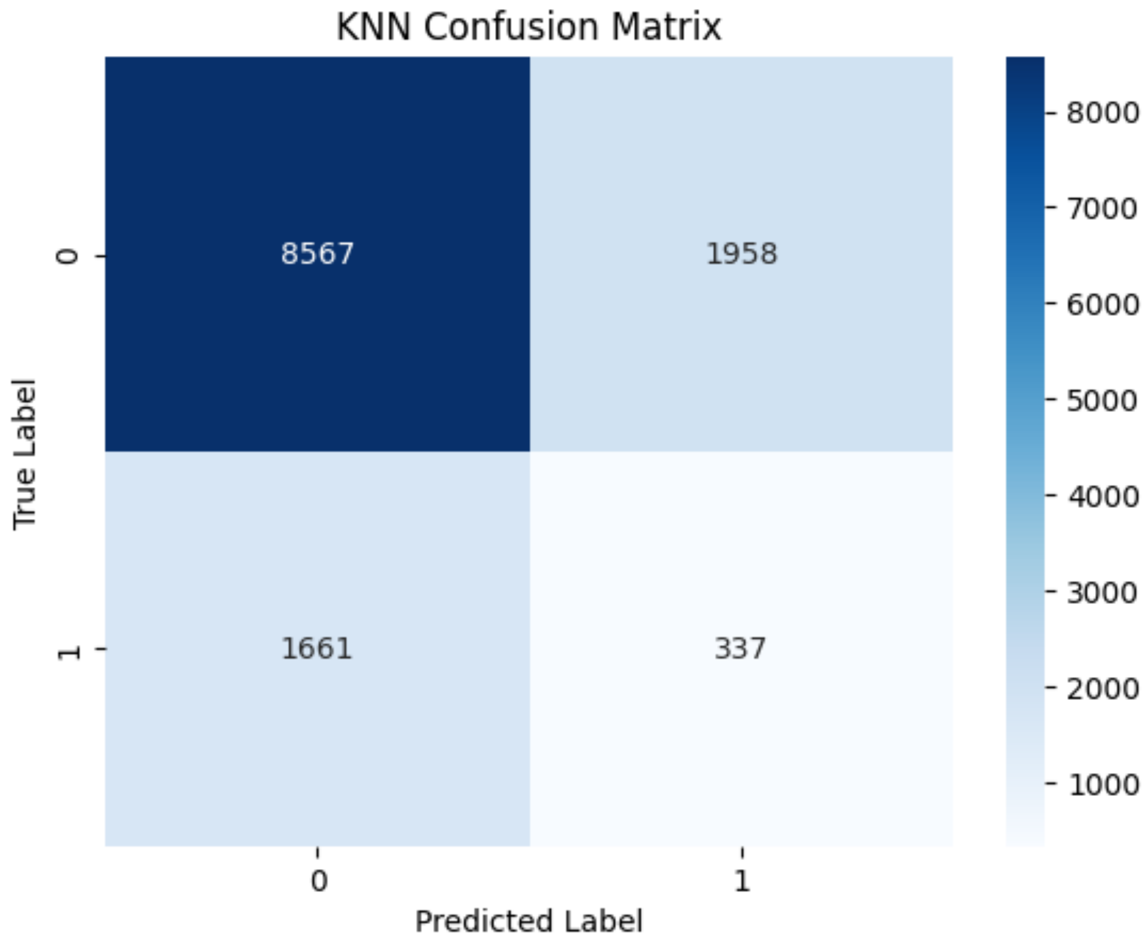
```
# Logistic Regression Confusion Matrix
cm_logreg = confusion_matrix(y_test, y_pred_logreg)
print("Logistic Regression Confusion Matrix:")
print(cm_logreg)

# KNN Confusion Matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)
print("\nKNN Confusion Matrix:")
print(cm_knn)
```

```
Logistic Regression Confusion Matrix:
[[10525     0]
 [ 1998     0]]

KNN Confusion Matrix:
[[8567 1958]
 [1661  337]]
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Logistic Regression
sns.heatmap(cm_logreg, annot=True, fmt="d", cmap="Blues")
plt.title("Logistic Regression Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# KNN
sns.heatmap(cm_knn, annot=True, fmt="d", cmap="Blues")
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```




True Label
1
1998
0
0
1
Predicted Label
6000
4000
2000
0


KNN Confusion Matrix
True Label
0
1
8567
1958
1661
337
0
1
Predicted Label
8000
7000
6000
5000
4000
3000
2000
1000