Fraud Detection on Bank Payments

Explaratory Data Analysis

In [1]:

```
## Data loading, processing and for more
import pandas as pd
import numpy as np
#from imblearn.over_sampling import SMOTE

## Visualization
import seaborn as sns
import matplotlib.pyplot as plt
# set seaborn style because it prettier
sns.set()
```

import the dataset

In [2]:

```
data = pd.read_csv(r"C:\Users\SSD\Downloads\bs140513_032310.csv~\bs140513_03231
```

In [3]:

```
data
```

	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category
0	0	'C1093826151'	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportati
1	0	'C352968107'	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportati
2	0	'C2054744914'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportati
3	0	'C1760612790'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportati
4	0	'C757503768'	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportati
...	...	...	...	...	...	...	...	...
594638	179	'C1753498738'	'3'	'F'	'28007'	'M1823072687'	'28007'	'es_transportati
594639	179	'C650108285'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportati
594640	179	'C123623130'	'2'	'F'	'28007'	'M349281107'	'28007'	'es_fashion'
594641	179	'C1499363341'	'5'	'M'	'28007'	'M1823072687'	'28007'	'es_transportati
594642	179	'C616528518'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportati

594643 rows × 10 columns

```
In [4]: data.head()
```

	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	a
0	0	'C1093826151'	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4
1	0	'C352968107'	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	3
2	0	'C2054744914'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	2
3	0	'C1760612790'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	1
4	0	'C757503768'	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	3

```
In [5]: data.shape
```

```
(594643, 10)
```

Let's look at column types and missing values in data.

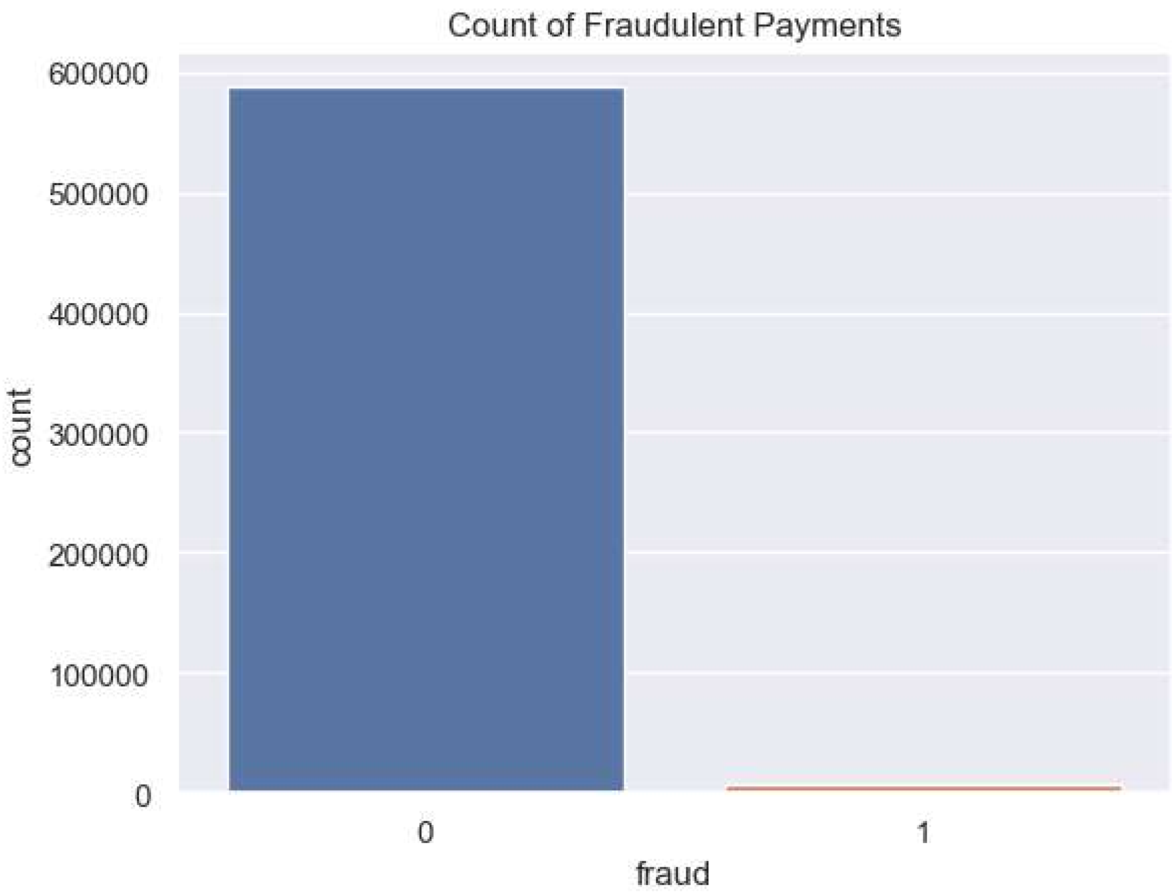
```
In [6]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 594643 entries, 0 to 594642
Data columns (total 10 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   step         594643 non-null  int64  
 1   customer     594643 non-null  object 
 2   age          594643 non-null  object 
 3   gender       594643 non-null  object 
 4   zipcodeOri   594643 non-null  object 
 5   merchant     594643 non-null  object 
 6   zipMerchant  594643 non-null  object 
 7   category     594643 non-null  object 
 8   amount       594643 non-null  float64
 9   fraud        594643 non-null  int64  
dtypes: float64(1), int64(2), object(7)
memory usage: 45.4+ MB
```

In [7]:

```
# Create two dataframes with fraud and non-fraud data
df_fraud = data.loc[data.fraud == 1]
df_non_fraud = data.loc[data.fraud == 0]

sns.countplot(x="fraud",data=data)
plt.title("Count of Fraudulent Payments")
plt.show()
print("Number of normal examples: ",df_non_fraud.fraud.count())
print("Number of fradulent examples: ",df_fraud.fraud.count())
#print(data.fraud.value_counts()) # does the same thing above
```



```
Number of normal examples:  587443
Number of fradulent examples:  7200
```

In [8]:

```
print("Mean feature values per category",data.groupby('category')['amount','frau
```

```
Mean feature values per category                                amount     fraud
category
'es_barsandrestaurants'    43.461014  0.018829
'es_contents'              44.547571  0.000000
'es_fashion'               65.666642  0.017973
'es_food'                  37.070405  0.000000
'es_health'               135.621367  0.105126
'es_home'                 165.670846  0.152064
'es_hotelservices'        205.614249  0.314220
'es_hyper'                 45.970421  0.045917
'es_leisure'              288.911303  0.949900
'es_otherservices'        135.881524  0.250000
'es_sportsandtoys'        215.715280  0.495252
'es_tech'                 120.947937  0.066667
'es_transportation'        26.958187  0.000000
'es_travel'              2250.409190  0.793956
'es_wellnessandbeauty'     65.511221  0.047594


C:\Users\SSD\AppData\Local\Temp\ipykernel_6544\1703450169.py:1: FutureWarning: Indexing with multiple
e of keys) will be deprecated, use a list instead.
  print("Mean feature values per category",data.groupby('category')['amount','fraud'].mean())
```

In [9]:

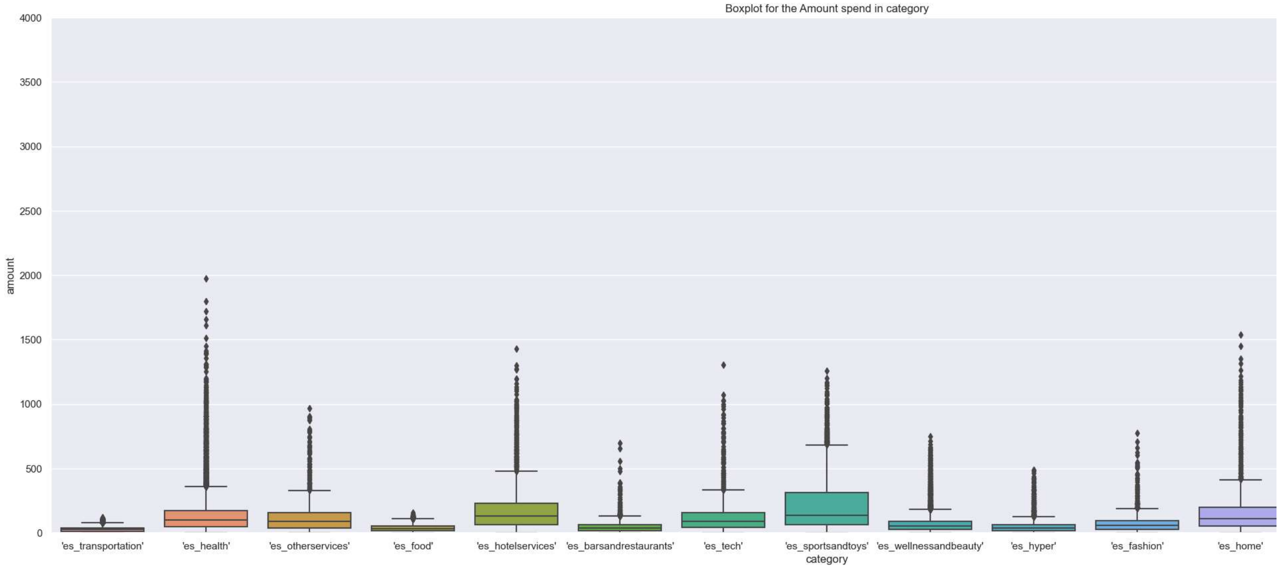
```
# Create two dataframes with fraud and non-fraud data
pd.concat([df_fraud.groupby('category')['amount'].mean(),df_non_fraud.groupby('c
           data.groupby('category')['fraud'].mean()*100],keys=["Fraudulent","Nor
          sort=False).sort_values(by=['Non-Fraudulent'])
```

	Fraudulent	Non-Fraudulent	Percent(%)
category			
'es_transportation'	NaN	26.958187	0.000000
'es_food'	NaN	37.070405	0.000000
'es_hyper'	169.255429	40.037145	4.591669
'es_barsandrestaurants'	164.092667	41.145997	1.882944
'es_contents'	NaN	44.547571	0.000000
'es_wellnessandbeauty'	229.422535	57.320219	4.759380
'es_fashion'	247.008190	62.347674	1.797335
'es_leisure'	300.286878	73.230400	94.989980
'es_otherservices'	316.469605	75.685497	25.000000
'es_sportsandtoys'	345.366811	88.502738	49.525237
'es_tech'	415.274114	99.924638	6.666667
'es_health'	407.031338	103.737228	10.512614
'es_hotelservices'	421.823339	106.548545	31.422018
'es_home'	457.484834	113.338409	15.206445
'es_travel'	2660.802872	669.025533	79.395604

In [10]:

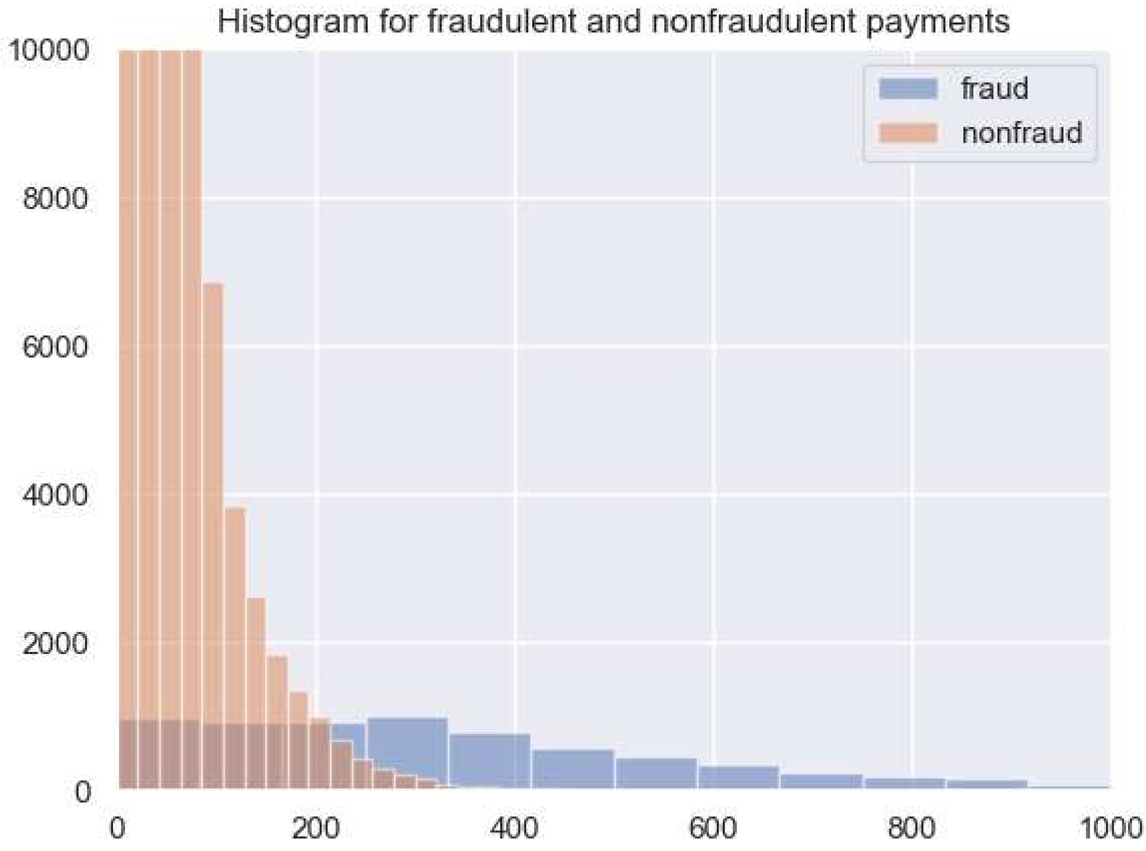
```
# Plot histograms of the amounts in fraud and non-fraud data
plt.figure(figsize=(30,10))
sns.boxplot(x=data.category,y=data.amount)
plt.title("Boxplot for the Amount spend in category")
plt.ylim(0,4000)
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore
with no argument.
```



In [11]:

```
# Plot histograms of the amounts in fraud and non-fraud data
plt.hist(df_fraud.amount, alpha=0.5, label='fraud',bins=100)
plt.hist(df_non_fraud.amount, alpha=0.5, label='nonfraud',bins=100)
plt.title("Histogram for fraudulent and nonfraudulent payments")
plt.ylim(0,10000)
plt.xlim(0,1000)
plt.legend()
plt.show()
```



In [12]:

```
print((data.groupby('age')['fraud'].mean()*100).reset_index().rename(columns={'a
```

```
   Age  Fraud Percent
7  'U'       0.594228
6  '6'       0.974826
5  '5'       1.095112
1  '1'       1.185254
3  '3'       1.192815
2  '2'       1.251401
4  '4'       1.293281
0  '0'       1.957586
```

Data Preprocessing

In [13]:

```python
print("Unique zipCodeOri values: ",data.zipcodeOri.nunique())
print("Unique zipMerchant values: ",data.zipMerchant.nunique())
# dropping zipcodeori and zipMerchant since they have only one unique value
data_reduced = data.drop(['zipcodeOri','zipMerchant'],axis=1)
```

```
Unique zipCodeOri values:  1
Unique zipMerchant values:  1
```

Checking the data after dropping.

In [14]:

```python
data_reduced.columns
```

```
Index(['step', 'customer', 'age', 'gender', 'merchant', 'category', 'amount',
       'fraud'],
      dtype='object')
```

In [15]:

```python
# turning object columns type to categorical for easing the transformation proce
col_categorical = data_reduced.select_dtypes(include= ['object']).columns
for col in col_categorical:
    data_reduced[col] = data_reduced[col].astype('category')
# categorical values ==> numeric values
data_reduced[col_categorical] = data_reduced[col_categorical].apply(lambda x: x.
data_reduced.head(5)
```

	step	customer	age	gender	merchant	category	amount	fraud
0	0	210	4	2	30	12	4.55	0
1	0	2753	2	2	30	12	39.68	0
2	0	2285	4	1	18	12	26.89	0
3	0	1650	3	2	30	12	17.25	0
4	0	3585	5	2	30	12	35.72	0

Let's define independent variable (X) and dependant/target variable y

In [16]:

```
X = data_reduced.drop(['fraud'],axis=1)
y = data['fraud']
print(X.head(),"\n")
print(y.head())
```

```
   step  customer  age  gender  merchant  category  amount
0     0       210    4       2        30        12    4.55
1     0      2753    2       2        30        12   39.68
2     0      2285    4       1        18        12   26.89
3     0      1650    3       2        30        12   17.25
4     0      3585    5       2        30        12   35.72

0    0
1    0
2    0
3    0
4    0
Name: fraud, dtype: int64
```

In [17]:

```
y[y==1].count()
```

```
7200
```

In [18]:

```
y[y==0].count()
```

```
587443
```

Oversampling with SMOTE

In [19]:

```
pip install --upgrade scikit-learn imbalanced-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\ssd\anaconda3\lib\site-packages (1.3.2)
Requirement already satisfied: imbalanced-learn in c:\users\ssd\anaconda3\lib\site-packages (0.11.0)
Requirement already satisfied: numpy<2.0,>=1.17.3 in c:\users\ssd\anaconda3\lib\site-packages (from sc
Requirement already satisfied: scipy>=1.5.0 in c:\users\ssd\anaconda3\lib\site-packages (from scikit-l
Requirement already satisfied: joblib>=1.1.1 in c:\users\ssd\anaconda3\lib\site-packages (from scikit-
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\ssd\anaconda3\lib\site-packages (from
Note: you may need to restart the kernel to use updated packages.
```

In [20]:
```
#pip uninstall imbalanced-learn
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\ssd\anaconda3\lib\site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\ssd\anaconda3\lib\site-packages (from imbalar
Requirement already satisfied: scipy>=1.5.0 in c:\users\ssd\anaconda3\lib\site-packages (from imbalanc
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\ssd\anaconda3\lib\site-packages (from i
Requirement already satisfied: joblib>=1.1.1 in c:\users\ssd\anaconda3\lib\site-packages (from imbalar
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\ssd\anaconda3\lib\site-packages (from 
```

In [21]:
```
from imblearn.over_sampling import SMOTE
```

In [22]:
```
print(SMOTE)
```

```
<class 'imblearn.over_sampling._smote.base.SMOTE'>
```

In [ ]:

In [23]:
```
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
y_res = pd.DataFrame(y_res)
```

In [24]:
```
print(type(y_res))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [25]:
```
print(y_res.head())
```

```
   fraud
0      0
1      0
2      0
3      0
4      0
```

In [26]:

```
print(y_res.value_counts())
```

```
fraud
0        587443
1        587443
dtype: int64
```

In [27]:

```
from sklearn.model_selection import train_test_split
```

In [28]:

```
# I won't do cross validation since we have a lot of instances
X_train, X_test, y_train, y_test = train_test_split(X_res,y_res,test_size=0.3,ra
```

In [29]:

```
X_train.shape
```

```
(822420, 7)
```

In [30]:

```
X_test.shape
```

```
(352466, 7)
```

In [31]:

```
y_train.shape
```

```
(822420, 1)
```

In [32]:

```
y_test.shape
```

```
(352466, 1)
```

In [33]:

```
# The base score should be better than predicting always non-fraduelent
print("Base accuracy score we must beat is: ",
      df_non_fraud.fraud.count()/ np.add(df_non_fraud.fraud.count(),df_fraud.fra
```

```
Base accuracy score we must beat is:  98.7891894800746
```

K-Neighbours Classifier

In [34]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [35]:

```
knn = KNeighborsClassifier(n_neighbors=5,p=1)

knn.fit(X_train,y_train)
#y_pred = knn.predict(X_test)
```

```
C:\Users\SSD\anaconda3\Lib\site-packages\sklearn\neighbors\_classification.py:233: DataConversionWarni
n a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

```
KNeighborsClassifier(p=1)
```

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 [37]:

```
y_pred = knn.predict(X_test)
y_pred
```

```
array([1, 1, 0, ..., 0, 0, 0], dtype=int64)
```

In [38]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[171999   4234]
 [   362 175871]]
```

In [39]:

```
# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
print(ac)
```

```
0.9869604444116595
```

In [40]:

```
# This is to get the Classification Report
from sklearn.metrics import classification_report
cr = classification_report(y_test, y_pred)
cr
```

```
'              precision    recall  f1-score   support\n\n           0       1.00      0.98      0.99
1.00      0.99    176233\n\n    accuracy                           0.99    352466\n   macro avg
weighted avg       0.99      0.99      0.99    352466\n'
```

In [42]:

```
bias = knn.score(X_train,y_train)
bias
```

```
0.9907711388341723
```

In [44]:

```
variance = knn.score(X_train,y_train)
variance
```

```
0.9907711388341723
```

Random Forest Classifier

In [45]:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', ra
classifier.fit(X_train, y_train)
```

```
C:\Users\SSD\anaconda3\Lib\site-packages\sklearn\base.py:1152: DataConversionWarning: A column-vector
ected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

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.

Predicting the Test set results

In [46]:

```
y_pred = classifier.predict(X_test)
y_pred
```

```
array([1, 1, 0, ..., 0, 0, 0], dtype=int64)
```

Making the Confusion Matrix¶

In [47]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[174892   1341]
 [   429 175804]]
```

In [48]:

```
# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
print(ac)
```

```
0.9949782390358219
```

In [49]:

```
bias = classifier.score(X_train,y_train)
bias
```

```
0.9997555993288101
```

In [50]:

```
variance = classifier.score(X_test,y_test)
variance
```

```
0.9949782390358219
```

In [51]:

```
from xgboost import XGBClassifier
classifier = XGBClassifier()
classifier.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

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 [52]:

```
y_pred = classifier.predict(X_test)
y_pred
```

```
array([1, 1, 0, ..., 0, 0, 0])
```

In [53]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[174531   1702]
 [   704 175529]]
```

In [54]:

```
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
print(ac)
```

```
0.9931738096724223
```

In [55]:

```
bias = classifier.score(X_train,y_train)
bias
```

```
0.9941380316626541
```

In [56]:

```
variance = classifier.score(X_test,y_test)
variance
```

```
0.9931738096724223
```

In [ ]: