CREDIT CARD FRAUD DETECTION

Credit card fraud impacts consumers, merchants and issuers alike. It's economic cost goes far beyond the cost of illegaly purchased merchandise. We need to spot potential fraud so that consumers can not bill for goods that they haven’t purchased.

The aim is, therefore, to create a classifier that indicates whether a requested transaction is a fraud.

Problem Statement

The objective of Credit Card Fraud Detection is to accurately identify fraudulent transactions from a large pool of credit card transactions by building a predictive model based on past transaction data. The aim is to detect all fraudulent transactions with minimum false alarms.

Observation

. The dataset is highly imbalanced, with only 0.172% of observations being fraudulent.

. The dataset consists of 28 transformed features (V1 to V28) and two untransformed features (Time and Amount).

. There is no missing data in the dataset, and no information about the original features is provided.

Why Treating Class Imbalance as an Issue?

. In general, we want to maximize the recall while capping FPR (False Positive Rate), but we can classify a lot of charges wrong and still maintain a low FPR because we have a large number of true negatives.

. This is conducive to picking a relatively low threshold, which results in a high recall but extremely low precision.

. The Minority class or fraud class requires our main focus, and this imbalance will based its accuracy on the majority class - the Genuine transactions

Bringing our Perspective

. Training a model on a balanced dataset optimizes performance on validation data.

However, our goal is to optimize performance on the imbalanced production dataset, while looking for a balance that works best in production.
. One solution to this problem is: Use all fraudulent transactions but subsample non-fraudulent transactions as needed to hit our target rate

Business Question

Since all features are anonymous, we will focus our analysis on non-anonymized features: Time, Amount

a. How different is the amount of money used in different transaction classes?
b. Do fraudulent transactions occur more often during certain frames?

Tools and Libraries

We will be using the following libraries and frameworks in this credit card fraud detection project.

- Python
- Numpy
- Scikit-learn
- Matplotlib
- Imblearn
- Collections, Itertools
- Seaborn

In [60]:

```
# Let's import our modules

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

%matplotlib inline
sns.set_style("whitegrid")
```

In [61]:

```
# Load the data
df = pd.read_csv("creditcard.csv")
df
```

Out[61]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V2
0	0.00	-1.36	-0.07	2.54	1.38	-0.34	0.46	0.24	0.10	0.36	...	-0.02	0.2
1	0.00	1.19	0.27	0.17	0.45	0.06	-0.08	-0.08	0.09	-0.26	...	-0.23	-0.6
2	1.00	-1.36	-1.34	1.77	0.38	-0.50	1.80	0.79	0.25	-1.51	...	0.25	0.7
3	1.00	-0.97	-0.19	1.79	-0.86	-0.01	1.25	0.24	0.38	-1.39	...	-0.11	0.0
4	2.00	-1.16	0.88	1.55	0.40	-0.41	0.10	0.59	-0.27	0.82	...	-0.01	0.8
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.00	-11.88	10.07	-9.83	-2.07	-5.36	-2.61	-4.92	7.31	1.91	...	0.21	0.1
284803	172787.00	-0.73	-0.06	2.04	-0.74	0.87	1.06	0.02	0.29	0.58	...	0.21	0.9
284804	172788.00	1.92	-0.30	-3.25	-0.56	2.63	3.03	-0.30	0.71	0.43	...	0.23	0.5
284805	172788.00	-0.24	0.53	0.70	0.69	-0.38	0.62	-0.69	0.68	0.39	...	0.27	0.8
284806	172792.00	-0.53	-0.19	0.70	-0.51	-0.01	-0.65	1.58	-0.41	0.49	...	0.26	0.6

284807 rows × 31 columns

OUR APPROACH

Step 1. Perform Exploratory Data Analysis (EDA)

In [62]:

```python
pd.set_option("display.float", "{:.2f}".format)

df.describe()
```

Out[62]:

	Time	V1	V2	V3	V4	V5	V6	
count	284807.00	284807.00	284807.00	284807.00	284807.00	284807.00	284807.00	28480
mean	94813.86	0.00	0.00	-0.00	0.00	0.00	0.00	-(
std	47488.15	1.96	1.65	1.52	1.42	1.38	1.33	'
min	0.00	-56.41	-72.72	-48.33	-5.68	-113.74	-26.16	-4
25%	54201.50	-0.92	-0.60	-0.89	-0.85	-0.69	-0.77	-(
50%	84692.00	0.02	0.07	0.18	-0.02	-0.05	-0.27	(
75%	139320.50	1.32	0.80	1.03	0.74	0.61	0.40	(
max	172792.00	2.45	22.06	9.38	16.88	34.80	73.30	12(

8 rows × 31 columns

```
Let's Check for any missing values in the dataset
```

In [63]:

```python
df.isnull().values.any()
```

Out[63]: False

```
The only non-transformed variables to work with are:
Time
Amount
Class (1: fraud, 0: not_fraud)
```

In [64]:

```python
LABELS = ["Genuine", "Fraud"]

count_classes = pd.value_counts(df['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency");
```

In [65]:

```python
# Let's check the number of occurrences of each class label.
not_fraud = len(df[df.Class == 0])
fraud = len(df[df.Class == 1])
fraud_percent = (fraud / (fraud + not_fraud)) * 100

print("Number of Genuine transactions: ", not_fraud)
print("Number of Fraud transactions: ", fraud)
print("Percentage of Fraud transactions: {:.2f}".format(fraud_percent
```

```
Number of Genuine transactions:  284315
Number of Fraud transactions:  492
Percentage of Fraud transactions: 0.17
```

Notice how imbalanced is our original dataset! Most of the transactions are not-fraud. If we use this Dataset as the base for our predictive models and analysis, we might get a lot of errors, and our algorithms will probably overfit since they will “assume” that most transactions are not a fraud. But we don’t want our model to assume, we want our model to detect patterns that give signs of fraud!

In [66]:

```python
# Let's detect the number of fraud and valid transactions in the enti

fraud = df[df['Class']==1]
genuine = df[df['Class']==0]

print(f"Shape of Fraudulent transactions: {fraud.shape}")
print(f"Shape of Non-Fraudulent transactions: {genuine.shape}")
```

```
Shape of Fraudulent transactions: (492, 31)
Shape of Non-Fraudulent transactions: (284315, 31)
```

How different is the amount of money used in different transaction classes?

In [67]:

```python
# Let's analyze the feature amount

pd.concat([fraud.Amount.describe(), genuine.Amount.describe()], axis=
```

Out[67]:

	Amount	Amount
count	492.00	284315.00
mean	122.21	88.29
std	256.68	250.11
min	0.00	0.00
25%	1.00	5.65
50%	9.25	22.00
75%	105.89	77.05
max	2125.87	25691.16

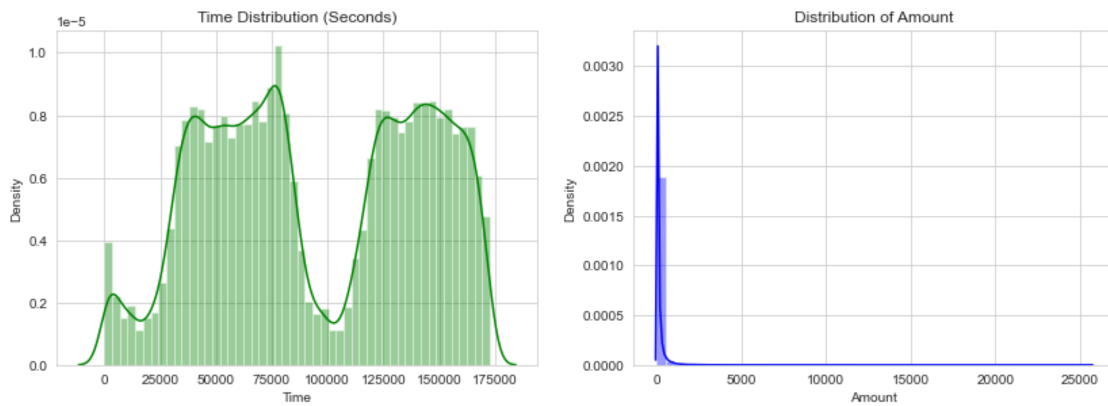
Do fraudulent transactions occur more often during a certain time frame?

It doesn’t seem like the time of the transaction really matters here, as per the above observation. Now let us take a sample of the dataset for our modeling and prediction.

In [68]:

```python
# Let's plot the time feature
plt.figure(figsize=(14,10))

plt.subplot(2, 2, 1)
plt.title('Time Distribution (Seconds)')
sns.distplot(df['Time'], color='green');

# Let's plot the amount feature
plt.subplot(2, 2, 2)
plt.title('Distribution of Amount')
sns.distplot(df['Amount'],color='blue');
```

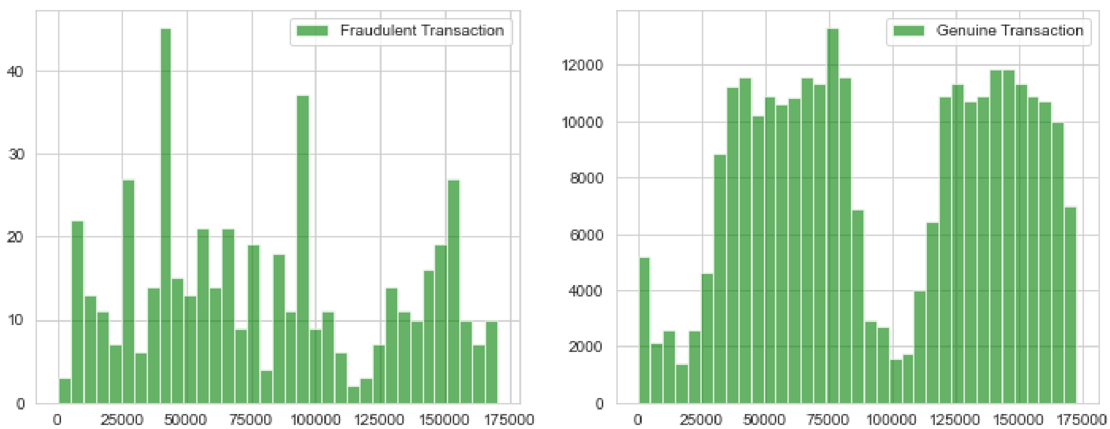
```
C:\Users\zyaobai\Anaconda3\envs\learn-env\lib\site-packages\seaborn\dist
ributions.py:2551: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use eit
her `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\zyaobai\Anaconda3\envs\learn-env\lib\site-packages\seaborn\dist
ributions.py:2551: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use eit
her `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [69]:

```python
# Let's visualize our data
# df[df.Class == 0].Time.hist(bins=35, color='blue', alpha=0.6)
plt.figure(figsize=(12, 10))

plt.subplot(2, 2, 1)
df[df.Class == 1].Time.hist(
  bins=35, color='green', alpha=0.6, label="Fraudulent Transaction"
)
plt.legend()

plt.subplot(2, 2, 2)
df[df.Class == 0].Time.hist(
  bins=35, color='green', alpha=0.6, label="Genuine Transaction"
)
plt.legend()
```

Out[69]: <matplotlib.legend.Legend at 0x192c29c9f10>



Correlation Matrices

Do we have features that influence heavily whether a specific transaction is a fraud. If so, it is important that we use the correct DataFrame (subsample) in order for us to see which features have a high positive or negative correlation with regard to fraudulent transactions.

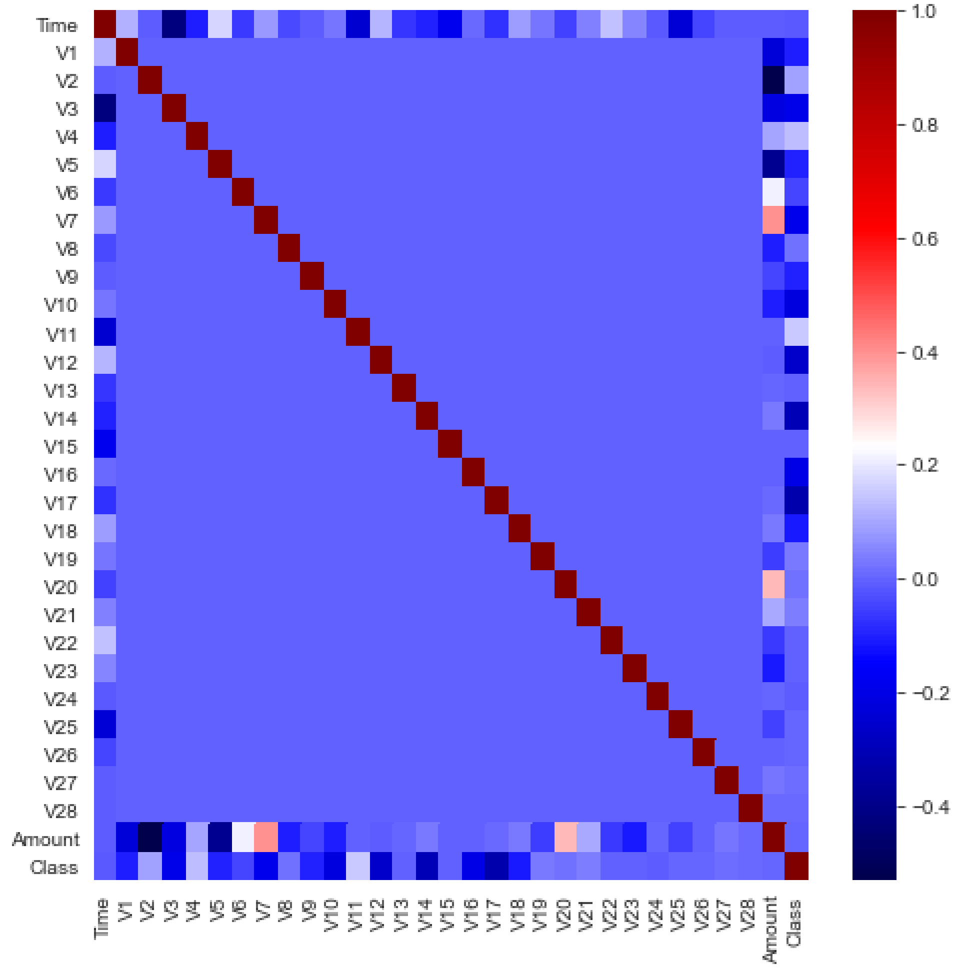
Observations

. Negative Correlations: V17, V14, V12, and V10 are negatively correlated. Notice how the lower these values are, the more likely the end result will be a fraudulent transaction.

. Positive Correlations: V2, V4, V11, and V19 are positively correlated. Notice how the higher these values are, the more likely the end result will be a fraudulent transaction.

In [70]:

```python
# Let's create heatmap to find any high correlations

plt.figure(figsize=(8,8))
sns.heatmap(data=df.corr(), cmap="seismic", annot=False)
plt.show();

# Save the plot as PNG file
plt.savefig('corr_heatmap.png')
```




```
<Figure size 432x288 with 0 Axes>
```

```
Observation

The highest correlations come from:
- Time & V3 (-0.42)
- Amount & V2 (-0.53)
- Amount & V4 (0.4)
```

Data Pre-Processing

In [71]:

```python
# Let's scale Time and Amount as the other columns.

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X = df.drop('Class', axis=1)
y = df.Class

X_train_v, X_test, y_train_v, y_test = train_test_split(X, y,
                                                        test_size=0.3, ra
X_train, X_validate, y_train, y_validate = train_test_split(X_train_v
                                                            test_size

X_train = scaler.fit_transform(X_train)
X_validate = scaler.transform(X_validate)
X_test = scaler.transform(X_test)

w_p = y_train.value_counts()[0] / len(y_train)
w_n = y_train.value_counts()[1] / len(y_train)

print(f"Fraudulent transaction weight: {w_n}")
print(f"Genuine transaction weight: {w_p}")
```

```
Fraudulent transaction weight: 0.0017994745785028623
Genuine transaction weight: 0.9982005254214972
```

In [72]:

```python
print(f"TRAINING: X_train: {X_train.shape}, y_train: {y_train.shape}\
print(f"VALIDATION: X_validate: {X_validate.shape}, y_validate: {y_va
print(f"TESTING: X_test: {X_test.shape}, y_test: {y_test.shape}")
```

```
TRAINING: X_train: (159491, 30), y_train: (159491,)
_______________________________________________________
VALIDATION: X_validate: (39873, 30), y_validate: (39873,)
__________________________________________________
TESTING: X_test: (85443, 30), y_test: (85443,)
```

In [74]:

```python
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay, f1
from sklearn.datasets import make_classification

def print_score(label, prediction, train=True):
    if train:
        clf_report = pd.DataFrame(classification_report(label, predic
        print("Train Result:\n=======================================
        print(f"Accuracy Score: {accuracy_score(label, prediction) * 1
        print("_______________________________________________")
        print(f"Classification Report:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, predi

    elif train==False:
        clf_report = pd.DataFrame(classification_report(label, predic
        print("Test Result:\n========================================
        print(f"Accuracy Score: {accuracy_score(label, prediction) * 1
        print("_______________________________________________")
        print(f"Classification Report:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(label, predicti
```

Step 2. Our Model Building

Artificial Neural Networks (ANNs)

In [48]:

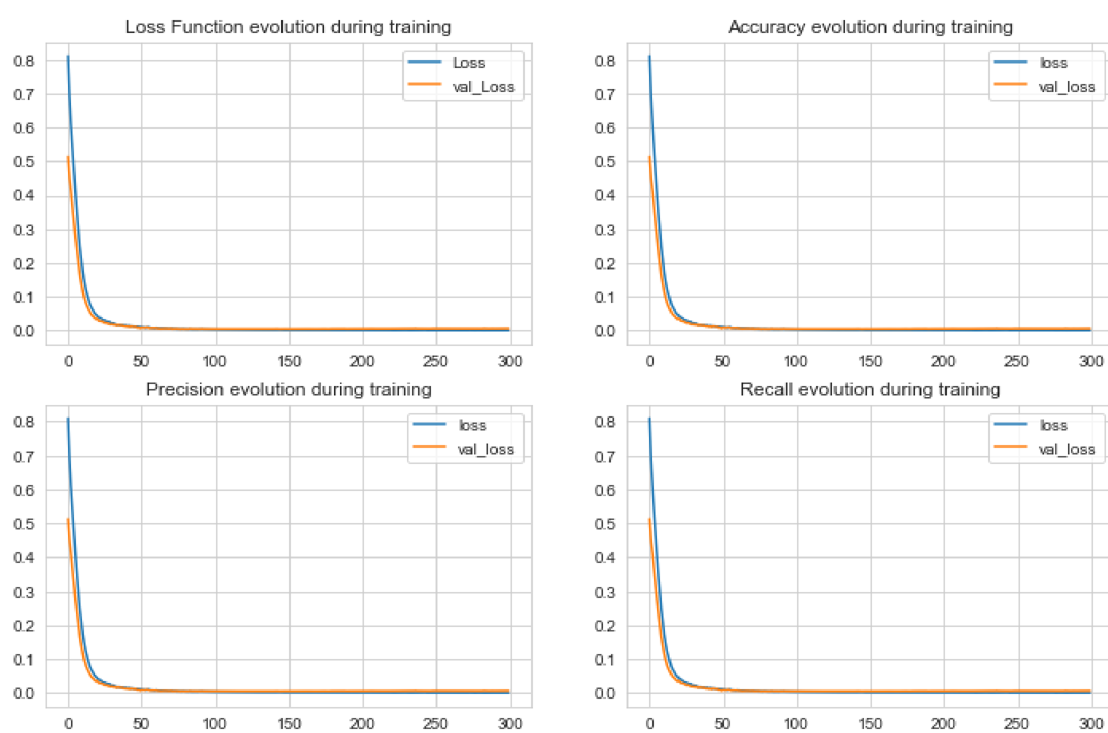
```
from sklearn.metrics import accuracy_score, precision_score, confusio
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Dense(256, activation='relu', input_shape=(X_train.s
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(1, activation='sigmoid'),
])

model = keras.Sequential([
    keras.layers.Dense(256, activation='relu', input_shape=(X_train.s
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(1, activation='sigmoid'),
])

model.compile(optimizer=keras.optimizers.Adam(1e-4), loss='binary_cro

callbacks = [keras.callbacks.ModelCheckpoint('fraud_model_at_epoch_{e
class_weight = {0:w_p, 1:w_n}

r = model.fit(
    X_train, y_train,
    validation_data=(X_validate, y_validate),
    batch_size=2048,
    epochs=300,
#     class_weight=class_weight,
    callbacks=callbacks,
)
score = model.evaluate(X_test, y_test)
print(score)
```

```
Epoch 1/300
78/78 [==============================] - 4s 45ms/step - loss: 0.8102 -
val_loss: 0.5137
Epoch 2/300
78/78 [==============================] - 3s 42ms/step - loss: 0.6947 -
val_loss: 0.4514
Epoch 3/300
78/78 [==============================] - 3s 41ms/step - loss: 0.6124 -
val_loss: 0.4133
Epoch 4/300
78/78 [==============================] - 4s 45ms/step - loss: 0.5437 -
val_loss: 0.3671
Epoch 5/300
78/78 [==============================] - 3s 38ms/step - loss: 0.4780 -
val_loss: 0.3180
Epoch 6/300
78/78 [==============================] - 3s 37ms/step - loss: 0.4153 -
val_loss: 0.2802
Epoch 7/300
```

In [87]:

```python
# Let's visualize it

plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='val_Loss')
plt.title('Loss Function evolution during training')
plt.legend()

plt.subplot(4, 2, 2)
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.title('Accuracy evolution during training')
plt.legend()

plt.subplot(4, 2, 3)
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.title('Precision evolution during training')
plt.legend()

plt.subplot(4, 2, 4)
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.title('Recall evolution during training')
plt.legend()
```

Out[87]: <matplotlib.legend.Legend at 0x192b446d850>



In [88]:

```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print_score(y_train, y_train_pred.round(), train=True)
print_score(y_test, y_test_pred.round(), train=False)

scores_dict = {
    'ANNs': {
        'Train': f1_score(y_train, y_train_pred.round()),
        'Test': f1_score(y_test, y_test_pred.round()),
    },
}
```

```
Train Result:
================================================
Accuracy Score: 99.99%
_______________________________________________
Classification Report:
                   0       1  accuracy   macro avg  weighted avg
precision       1.00    1.00      1.00        1.00          1.00
recall          1.00    0.95      1.00        0.98          1.00
f1-score        1.00    0.97      1.00        0.99          1.00
support    159204.00  287.00      1.00   159491.00     159491.00
_______________________________________________
Confusion Matrix:
 [[159203      1]
 [    14    273]]

Test Result:
================================================
Accuracy Score: 99.95%
_______________________________________________
Classification Report:
                  0       1  accuracy  macro avg  weighted avg
precision      1.00    0.89      1.00       0.95          1.00
recall         1.00    0.80      1.00       0.90          1.00
f1-score       1.00    0.84      1.00       0.92          1.00
support    85307.00  136.00      1.00   85443.00      85443.00
_______________________________________________
Confusion Matrix:
 [[85294    13]
 [   27   109]]
```

XGBoost

In [89]:

```python
from xgboost import XGBClassifier

xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train, eval_metric='aucpr')

y_train_pred = xgb_clf.predict(X_train)
y_test_pred = xgb_clf.predict(X_test)

print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

scores_dict['XGBoost'] = {
        'Train': f1_score(y_train,y_train_pred),
        'Test': f1_score(y_test, y_test_pred),
}
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:
                   0      1  accuracy  macro avg  weighted avg
precision       1.00   1.00      1.00       1.00          1.00
recall          1.00   1.00      1.00       1.00          1.00
f1-score        1.00   1.00      1.00       1.00          1.00
support    159204.00 287.00      1.00  159491.00     159491.00
_______________________________________________
Confusion Matrix:
 [[159204      0]
 [     0    287]]

Test Result:
================================================
Accuracy Score: 99.96%
_______________________________________________
Classification Report:
                  0      1  accuracy  macro avg  weighted avg
precision      1.00   0.95      1.00       0.97          1.00
recall         1.00   0.82      1.00       0.91          1.00
f1-score       1.00   0.88      1.00       0.94          1.00
support    85307.00 136.00      1.00   85443.00      85443.00
_______________________________________________
Confusion Matrix:
 [[85301     6]
 [   25   111]]
```

RANDOM FOREST

In [90]:

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=100, oob_score=False)
rf_clf.fit(X_train, y_train)

y_train_pred = rf_clf.predict(X_train)
y_test_pred = rf_clf.predict(X_test)

print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

scores_dict['Random Forest'] = {
        'Train': f1_score(y_train,y_train_pred),
        'Test': f1_score(y_test, y_test_pred),
}
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:
                   0      1  accuracy  macro avg  weighted avg
precision       1.00   1.00      1.00       1.00          1.00
recall          1.00   1.00      1.00       1.00          1.00
f1-score        1.00   1.00      1.00       1.00          1.00
support    159204.00 287.00      1.00  159491.00     159491.00
_______________________________________________
Confusion Matrix:
 [[159204      0]
 [     0    287]]

Test Result:
================================================
Accuracy Score: 99.96%
_______________________________________________
Classification Report:
                  0      1  accuracy  macro avg  weighted avg
precision      1.00   0.93      1.00       0.97          1.00
recall         1.00   0.80      1.00       0.90          1.00
f1-score       1.00   0.86      1.00       0.93          1.00
support    85307.00 136.00      1.00   85443.00      85443.00
_______________________________________________
Confusion Matrix:
 [[85299     8]
 [   27   109]]
```

DECISION TREES

In [91]:

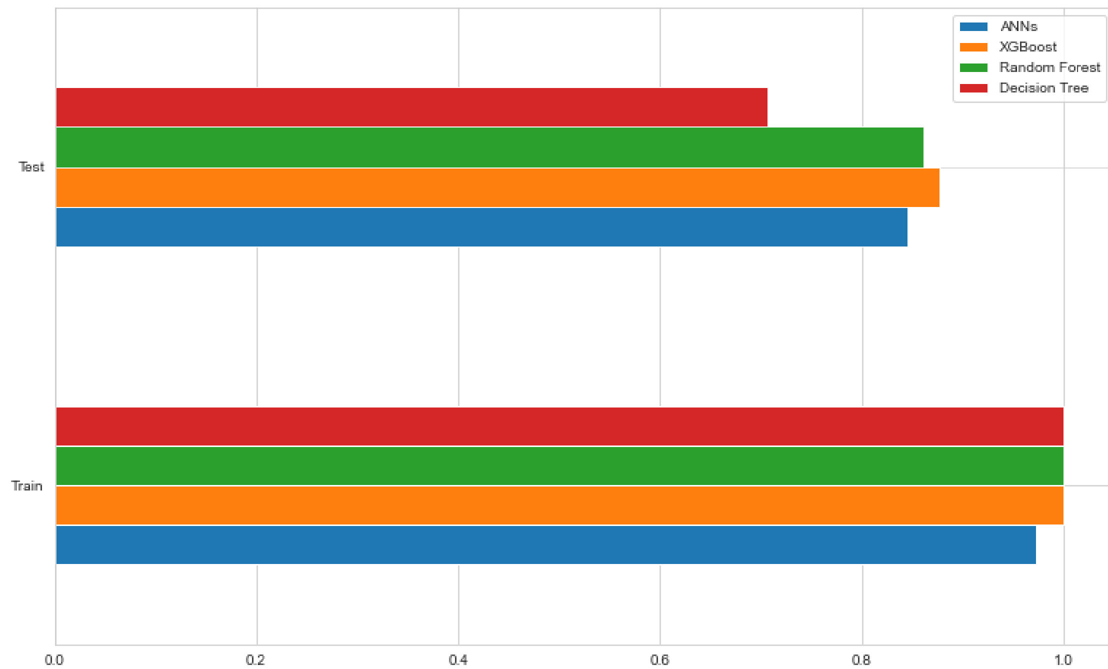
```python
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)

y_train_pred = dt_clf.predict(X_train)
y_test_pred = dt_clf.predict(X_test)

print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

scores_dict['Decision Tree'] = {
        'Train': f1_score(y_train,y_train_pred),
        'Test': f1_score(y_test, y_test_pred),
}
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:
                   0      1  accuracy  macro avg  weighted avg
precision       1.00   1.00      1.00       1.00          1.00
recall          1.00   1.00      1.00       1.00          1.00
f1-score        1.00   1.00      1.00       1.00          1.00
support    159204.00 287.00      1.00  159491.00     159491.00
_______________________________________________
Confusion Matrix:
 [[159204      0]
 [     0    287]]

Test Result:
================================================
Accuracy Score: 99.90%
_______________________________________________
Classification Report:
                  0      1  accuracy  macro avg  weighted avg
precision      1.00   0.67      1.00       0.84          1.00
recall         1.00   0.74      1.00       0.87          1.00
f1-score       1.00   0.71      1.00       0.85          1.00
support    85307.00 136.00      1.00   85443.00      85443.00
_______________________________________________
Confusion Matrix:
 [[85258    49]
 [   35   101]]
```

Step 3. MODEL COMPARISON

In [94]:

```python
scores_df = pd.DataFrame(scores_dict)

scores_df.plot(kind='barh', figsize=(13, 8))

# Save the plot as PNG file
plt.savefig('model_comparison.png')
```



CONCLUSION

```
We developed our credit card fraud detection model using machine learning algorithms. We used a variety of them, including ANNs and Tree-based models. In sum, our studied focused on supervised processes. At the end of the training, out of 85443 validation transaction, XGBoost performs better than other models:

Correctly identifying 111 of them as fraudulent
Missing 9 fraudulent transactions
At the cost of incorrectly flagging 25 legitimate transactions
```