Credit Card Fraud Detection

Credit card fraud is when someone uses another person's credit card or account information to make unauthorized purchases or access funds through cash advances. Credit card fraud doesn't just happen online; it happens in brick-and-mortar stores, too.

In [47]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [48]:
```
data = pd.read_csv("/creditcardfraud/creditcard.csv")
data.head()
```

Out[48]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25
0	0.00	-1.36	-0.07	2.54	1.38	-0.34	0.46	0.24	0.10	0.36	...	-0.02	0.28	-0.11	0.07	0.13
1	0.00	1.19	0.27	0.17	0.45	0.06	-0.08	-0.08	0.09	-0.26	...	-0.23	-0.64	0.10	-0.34	0.17
2	1.00	-1.36	-1.34	1.77	0.38	-0.50	1.80	0.79	0.25	-1.51	...	0.25	0.77	0.91	-0.69	-0.33
3	1.00	-0.97	-0.19	1.79	-0.86	-0.01	1.25	0.24	0.38	-1.39	...	-0.11	0.01	-0.19	-1.18	0.65
4	2.00	-1.16	0.88	1.55	0.40	-0.41	0.10	0.59	-0.27	0.82	...	-0.01	0.80	-0.14	0.14	-0.21

5 rows × 31 columns

Exploratory Data Analysis

In [49]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [50]:
```
pd.set_option("display.float", "{:.2f}".format)
data.describe()
```

Out[50]:

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.00	284807.00	284807.00	284807.00	284807.00	284807.00	284807.00	284807.00	284807
mean	94813.86	0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	0
std	47488.15	1.96	1.65	1.52	1.42	1.38	1.33	1.24	1
min	0.00	-56.41	-72.72	-48.33	-5.68	-113.74	-26.16	-43.56	-73
25%	54201.50	-0.92	-0.60	-0.89	-0.85	-0.69	-0.77	-0.55	-0
50%	84692.00	0.02	0.07	0.18	-0.02	-0.05	-0.27	0.04	0
75%	139320.50	1.32	0.80	1.03	0.74	0.61	0.40	0.57	0
max	172792.00	2.45	22.06	9.38	16.88	34.80	73.30	120.59	20

8 rows × 31 columns

In [51]:
```
data.isnull().sum().sum()
```

Out[51]:
```
0
```

In [52]:
```
data.columns
```

Out[52]:
```
Index(['Time', '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',
       'Class'],
      dtype='object')
```

The only non-transformed variables to work with are:

- `Time`
- `Amount`
- `Class` (1: fraud, 0: not_fraud)

In [53]:
```
LABELS = ["Normal", "Fraud"]

count_classes = pd.value_counts(data['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 [54]:
```
data.Class.value_counts()
```

Out[54]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

In [55]:
```
fraud = data[data['Class']==1]
normal = data[data['Class']==0]

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

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

In [56]:
```
pd.concat([fraud.Amount.describe(), normal.Amount.describe()], axis=1)
```

Out[56]:

	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

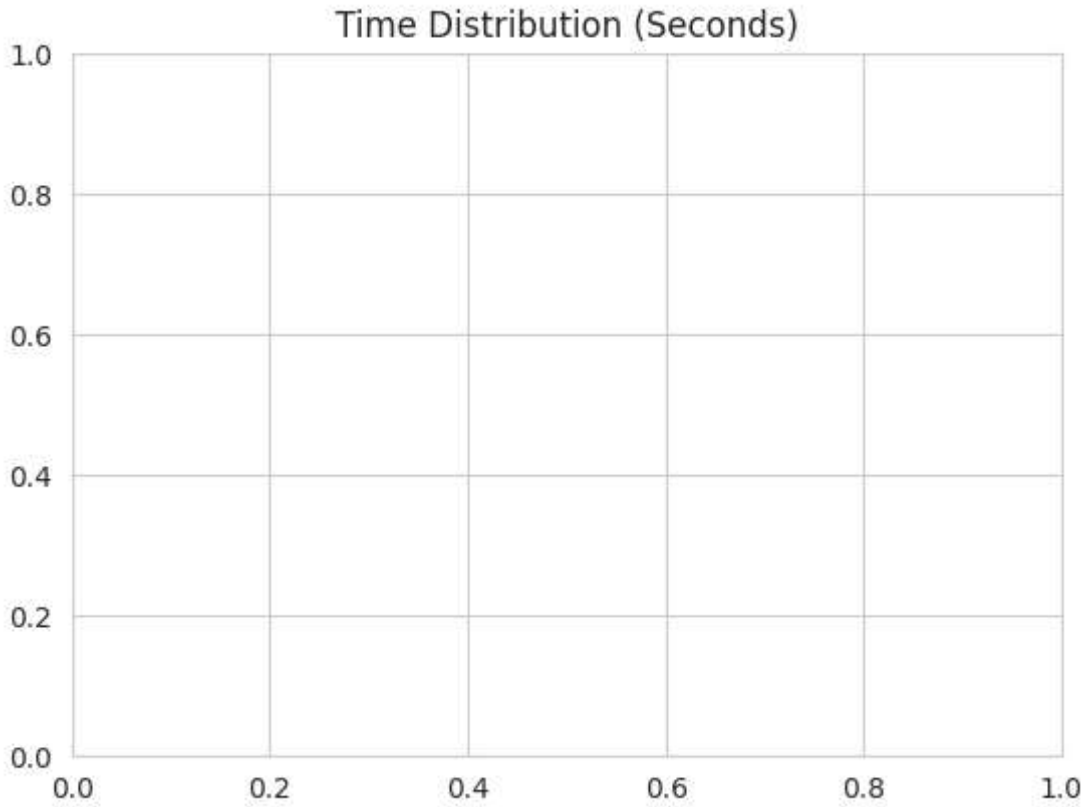
In [57]:
```
pd.concat([fraud.Time.describe(), normal.Time.describe()], axis=1)
```

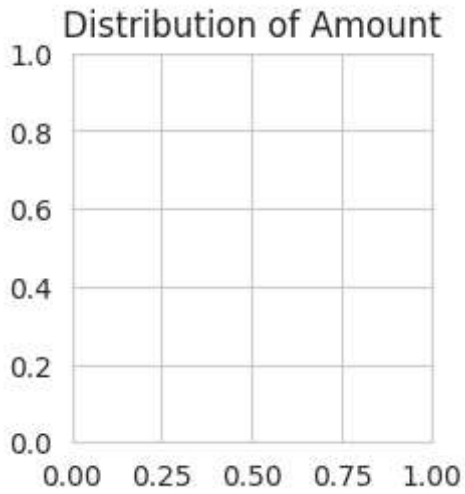
Out[57]:

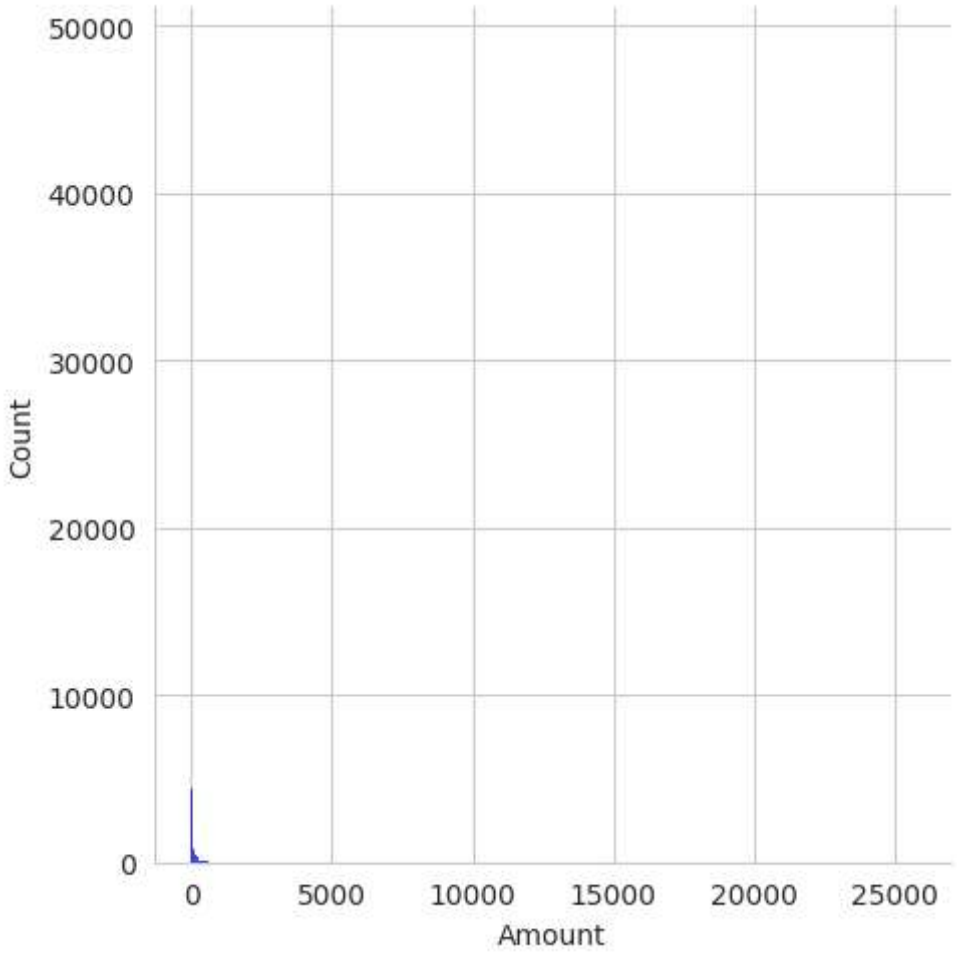
	Time	Time
count	492.00	284315.00
mean	80746.81	94838.20
std	47835.37	47484.02
min	406.00	0.00
25%	41241.50	54230.00
50%	75568.50	84711.00
75%	128483.00	139333.00
max	170348.00	172792.00

In [58]:
```
# plot the time feature
plt.figure(figsize=(14,10))

plt.subplot(2, 2, 1)
plt.title('Time Distribution (Seconds)')

sns.displot(data['Time'], color='blue');

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

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The fig
ure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
/tmp/ipykernel_28/3161143837.py:10: MatplotlibDeprecationWarning: Auto-removal of ove
rlapping axes is deprecated since 3.6 and will be removed two minor releases later; e
xplicitly call ax.remove() as needed.
  plt.subplot(2, 2, 2)
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The fig
ure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

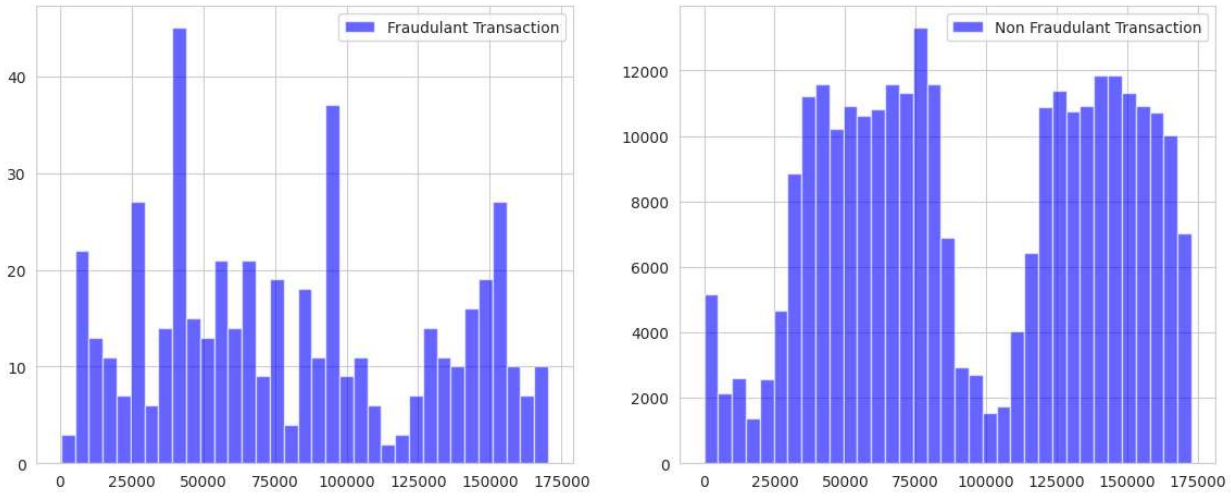

Time Distribution (Seconds)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Distribution of Amount
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00

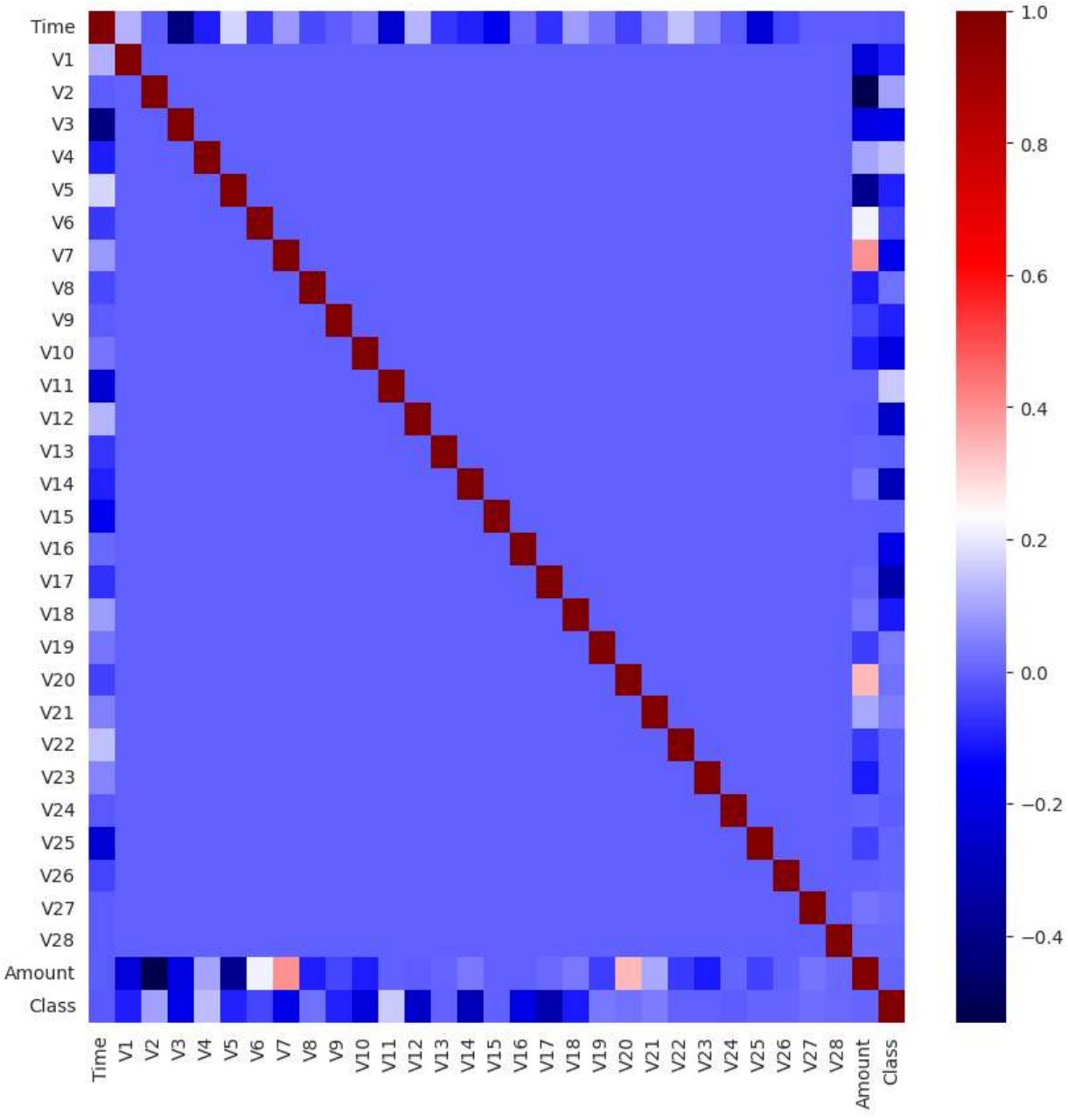


In [59]:
```
# data[data.Class == 0].Time.hist(bins=35, color='blue', alpha=0.6)
plt.figure(figsize=(14, 12))

plt.subplot(2, 2, 1)
data[data.Class == 1].Time.hist(bins=35, color='blue', alpha=0.6, label="Fraudulant Tr
plt.legend()

plt.subplot(2, 2, 2)
data[data.Class == 0].Time.hist(bins=35, color='blue', alpha=0.6, label="Non Fraudular
plt.legend()
```

Out[59]:
```
<matplotlib.legend.Legend at 0x7c2f4330d540>
```



In [60]:
```
# heatmap to find any high correlations

plt.figure(figsize=(10,10))
sns.heatmap(data=data.corr(), cmap="seismic")
plt.show();
```



Highest correlations come from:

- Time & V3 (-0.42)
- Amount & V2 (-0.53)
- Amount & V4 (0.4)

Data Pre-processing

`Time` and `Amount` should be scaled as the other columns.

In [61]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
scalar = StandardScaler()

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

X_train_v, X_test, y_train_v, y_test = train_test_split(X, y,
                                                        test_size=0.3, random_state=42)
X_train, X_validate, y_train, y_validate = train_test_split(X_train_v, y_train_v,
                                                            test_size=0.2, random_stat

X_train = scalar.fit_transform(X_train)
X_validate = scalar.transform(X_validate)
X_test = scalar.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"Fraudulant transaction weight: {w_n}")
print(f"Non-Fraudulant transaction weight: {w_p}")
```

```
Fraudulant transaction weight: 0.0017994745785028623
Non-Fraudulant transaction weight: 0.9982005254214972
```

In [62]:
```
print(f"TRAINING: X_train: {X_train.shape}, y_train: {y_train.shape}\n{'_'*55}")
print(f"VALIDATION: X_validate: {X_validate.shape}, y_validate: {y_validate.shape}\n{'
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 [63]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f

def print_score(label, prediction, train=True):
    if train:
        clf_report = pd.DataFrame(classification_report(label, prediction, output_dict
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(label, prediction) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Classification Report:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, prediction)}\n")

    elif train==False:
        clf_report = pd.DataFrame(classification_report(label, prediction, output_dict
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(label, prediction) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Classification Report:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(label, prediction)}\n")
```

Model Building

Artificial Neural Network (ANNs)

In [64]:
```
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Dense(256, activation='relu', input_shape=(X_train.shape[-1],)),
    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.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 256)               7936

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 256)               65792

 batch_normalization_4 (Batc  (None, 256)              1024
 hNormalization)

 dropout_4 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 256)               65792

 batch_normalization_5 (Batc  (None, 256)              1024
 hNormalization)

 dropout_5 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 1)                 257

=================================================================
Total params: 142,849
Trainable params: 141,313
Non-trainable params: 1,536
_________________________________________________________________
```

In [ ]:
```
METRICS = [
#     keras.metrics.Accuracy(name='accuracy'),
    keras.metrics.FalseNegatives(name='fn'),
    keras.metrics.FalsePositives(name='fp'),
    keras.metrics.TrueNegatives(name='tn'),
    keras.metrics.TruePositives(name='tp'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall')
]
```

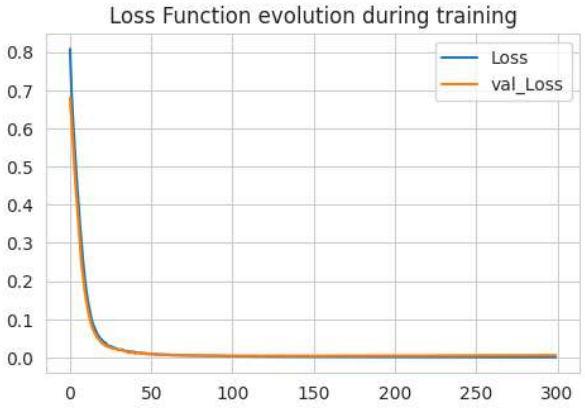
```
model.compile(optimizer=keras.optimizers.Adam(1e-4), loss='binary_crossentropy', metri

callbacks = [keras.callbacks.ModelCheckpoint('fraud_model_at_epoch_{epoch}.h5')]
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,
)
```

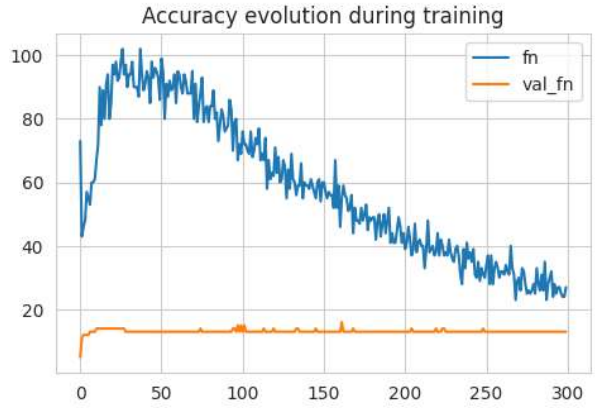
In [66]:
```
score = model.evaluate(X_test, y_test)
print(score)
```

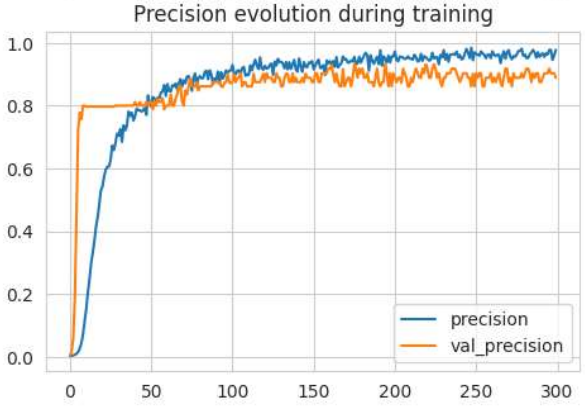
```
2671/2671 [==============================] - 8s 3ms/step - loss: 0.0035 - fn: 26.0000
- fp: 17.0000 - tn: 85290.0000 - tp: 110.0000 - precision: 0.8661 - recall: 0.8088
[0.0035336429718881845, 26.0, 17.0, 85290.0, 110.0, 0.8661417365074158, 0.80882352590
56091]
```

In [67]:
```
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['fn'], label='fn')
plt.plot(r.history['val_fn'], label='val_fn')
plt.title('Accuracy evolution during training')
plt.legend()

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

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

Out[67]:
```
<matplotlib.legend.Legend at 0x7c2f3d47d8d0>
```







In [68]:
```
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()),
    },
}
```

```
4985/4985 [==============================] - 8s 2ms/step
2671/2671 [==============================] - 4s 2ms/step
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   0.96      1.00       0.98          1.00
f1-score         1.00   0.98      1.00       0.99          1.00
support     159204.00 287.00      1.00  159491.00     159491.00

------------------------------------------------
Confusion Matrix:
 [[159203      1]
 [    11    276]]

Test Result:
================================================
Accuracy Score: 99.95%

------------------------------------------------
Classification Report:
                    0      1  accuracy  macro avg  weighted avg
precision        1.00   0.87      1.00       0.93          1.00
recall           1.00   0.81      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:
 [[85290    17]
 [   26   110]]
```

XGBoost

In [69]:

```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),
}
```

```
/opt/conda/lib/python3.10/site-packages/xgboost/sklearn.py:835: UserWarning: `eval_me
tric` in `fit` method is deprecated for better compatibility with scikit-learn, use `
eval_metric` in constructor or`set_params` instead.
  warnings.warn(
```

```
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 [70]:

```
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   0.90      1.00       0.95          1.00
recall           1.00   0.82      1.00       0.91          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:
 [[85295    12]
 [   25   111]]
```

CatBoost

In [ ]:
```
from catboost import CatBoostClassifier

cb_clf = CatBoostClassifier()
cb_clf.fit(X_train, y_train)
```

In [72]:
```
y_train_pred = cb_clf.predict(X_train)
y_test_pred = cb_clf.predict(X_test)

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

scores_dict['CatBoost'] = {
        '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]
 [     1    286]]

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   0.82      1.00       0.91          1.00
f1-score         1.00   0.87      1.00       0.94          1.00
support     85307.00 136.00      1.00   85443.00      85443.00

------------------------------------------------
Confusion Matrix:
 [[85299     8]
 [   25   111]]
```

LightGBM

In [73]:

```python
from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier()
lgbm_clf.fit(X_train, y_train)

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

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

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

```
Train Result:
================================================
Accuracy Score: 99.58%

------------------------------------------------
Classification Report:
                    0      1  accuracy  macro avg  weighted avg
precision        1.00   0.23      1.00       0.62          1.00
recall           1.00   0.59      1.00       0.79          1.00
f1-score         1.00   0.33      1.00       0.67          1.00
support     159204.00 287.00      1.00  159491.00     159491.00

------------------------------------------------
Confusion Matrix:
 [[158652    552]
 [   119    168]]

Test Result:
================================================
Accuracy Score: 99.50%

------------------------------------------------
Classification Report:
                    0      1  accuracy  macro avg  weighted avg
precision        1.00   0.16      0.99       0.58          1.00
recall           1.00   0.53      0.99       0.76          0.99
f1-score         1.00   0.25      0.99       0.62          1.00
support     85307.00 136.00      0.99   85443.00      85443.00

------------------------------------------------
Confusion Matrix:
 [[84942   365]
 [   64    72]]
```

Model Comparaison

In [74]:
```
scores_df = pd.DataFrame(scores_dict)

scores_df.plot(kind='barh', figsize=(15, 8))
```

Out[74]:
```
<Axes: >
```