```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d mlg-ulb/creditcardfraud
!unzip creditcardfraud.zip
```

```
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
Dataset URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
License(s): DbCL-1.0
Downloading creditcardfraud.zip to /content
 88% 58.0M/66.0M [00:00<00:00, 76.1MB/s]
100% 66.0M/66.0M [00:00<00:00, 72.2MB/s]
Archive:  creditcardfraud.zip
  inflating: creditcard.csv
```

```
import pandas as pd
df = pd.read_csv('creditcard.csv')
df
```

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577

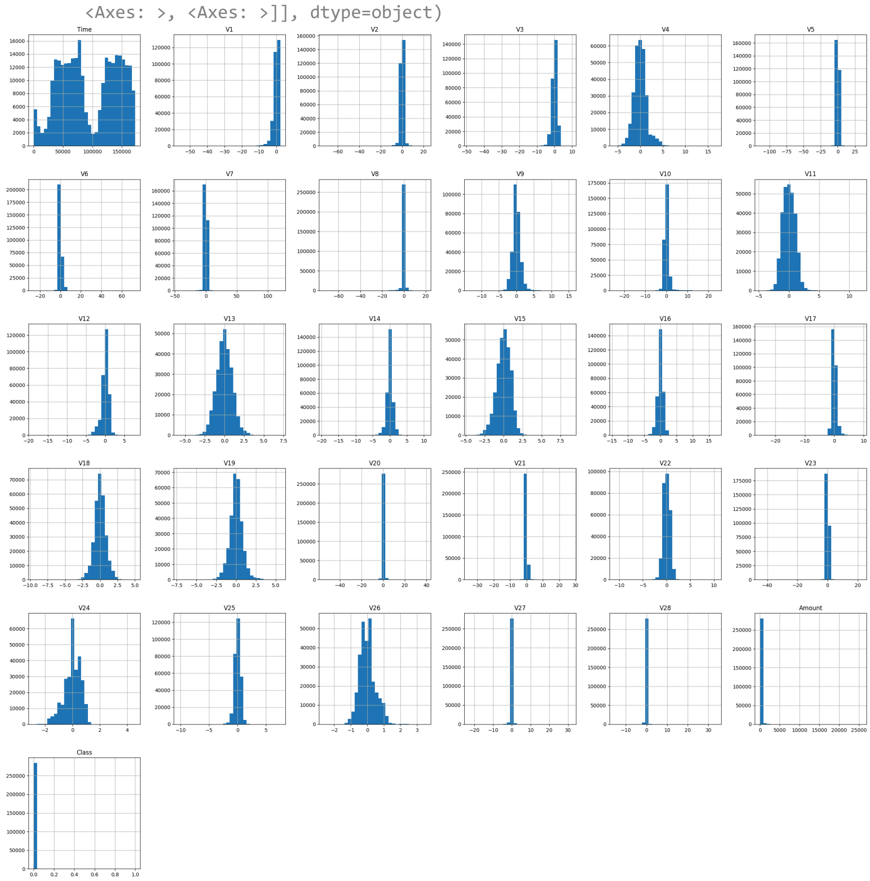
284807 rows × 31 columns

```
df['Class'].value_counts()
```

```
Class
0    284315
1       492
Name: count, dtype: int64
```

```
#Getting a quick overview of all the data through a set of histograms
df.hist(bins=30, figsize=(30, 30))
```

```
array([[<Axes: title={'center': 'Time'}>, <Axes: title={'center': 'V1'}>,
        <Axes: title={'center': 'V2'}>, <Axes: title={'center': 'V3'}>,
        <Axes: title={'center': 'V4'}>, <Axes: title={'center': 'V5'}>],
       [<Axes: title={'center': 'V6'}>, <Axes: title={'center': 'V7'}>,
        <Axes: title={'center': 'V8'}>, <Axes: title={'center': 'V9'}>,
        <Axes: title={'center': 'V10'}>, <Axes: title={'center': 'V11'}>],
       [<Axes: title={'center': 'V12'}>, <Axes: title={'center': 'V13'}>,
        <Axes: title={'center': 'V14'}>, <Axes: title={'center': 'V15'}>,
        <Axes: title={'center': 'V16'}>, <Axes: title={'center': 'V17'}>],
       [<Axes: title={'center': 'V18'}>, <Axes: title={'center': 'V19'}>,
        <Axes: title={'center': 'V20'}>, <Axes: title={'center': 'V21'}>,
        <Axes: title={'center': 'V22'}>, <Axes: title={'center': 'V23'}>],
       [<Axes: title={'center': 'V24'}>, <Axes: title={'center': 'V25'}>,
        <Axes: title={'center': 'V26'}>, <Axes: title={'center': 'V27'}>,
        <Axes: title={'center': 'V28'}>,
        <Axes: title={'center': 'Amount'}>],
       [<Axes: title={'center': 'Class'}>, <Axes: >, <Axes: >, <Axes: >,
        <Axes: >, <Axes: >]], dtype=object)
```



```
df.describe()
```

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8480
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604(
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.3802
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.1374
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915(
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433(
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119(
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.4801

8 rows × 31 columns

```
#Preprocessing data using RobustScaler from scikit-learn
#Placing this processed data into a new dataframe
from sklearn.preprocessing import RobustScaler
new_df = df.copy()
new_df['Amount'] = RobustScaler().fit_transform(new_df['Amount'].to_numpy().reshape(-1, 1))
time = new_df['Time']
new_df['Time'] = (time - time.min()) / (time.max() - time.min())
new_df
```

	Time	V1	V2	V3	V4	V5	V6	
0	0.000000	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239
1	0.000000	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078
2	0.000006	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791
3	0.000006	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237
4	0.000012	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592
...	...	...	...	...	...	...	...	
284802	0.999965	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918
284803	0.999971	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024
284804	0.999977	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296
284805	0.999977	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686
284806	1.000000	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577

284807 rows × 31 columns

```
new_df = new_df.sample(frac=1, random_state=4)
new_df
```

	Time	V1	V2	V3	V4	V5	V6	
135406	0.470161	0.999672	-0.034679	0.446984	1.374760	-0.272838	0.033625	0.0257
137826	0.476567	-0.844413	1.032424	1.090921	-0.671593	-0.006061	-0.621923	0.3226
70830	0.312717	-0.474271	1.027526	1.546229	-0.082036	0.180465	-0.407305	0.6950
194993	0.757292	-1.619583	-0.460686	0.219034	-0.418723	0.933105	-0.477342	0.9028
87575	0.357337	-1.159349	0.816687	1.743063	-0.724069	-0.398590	-0.796834	0.2752
...	...	...	...	...	...	...	...	
107578	0.408046	1.329792	-0.532095	0.410993	-0.557842	-1.134187	-1.030980	-0.3950
94601	0.375874	-4.886561	2.942698	0.260037	-0.229327	-2.422474	-0.161059	-1.0692
115144	0.426889	1.211935	-1.052726	1.179067	-0.510623	-1.841176	-0.316907	-1.2875
129384	0.457573	1.560825	-1.325706	-0.770189	-2.497505	0.693174	3.454604	-1.8344
120705	0.439442	1.248423	-0.845492	1.042291	-0.729424	-1.444453	-0.063721	-1.2346

284807 rows × 31 columns

```
#Seperating the dataframe into three seperate dataframes for training, testing and validation.
train, test, val = new_df[:240000], new_df[240000:262000], new_df[262000:]
train['Class'].value_counts(), test['Class'].value_counts(), val['Class'].value_counts()
```

```
(Class
 0    239576
 1       424
 Name: count, dtype: int64,
 Class
 0    21967
 1       33
 Name: count, dtype: int64,
 Class
 0    22772
 1       35
 Name: count, dtype: int64)
```

```
#Converting the dataframes into NumPy arrays then checking the shapes of these arrays. This will help some of the machine learning libra
train_np, test_np, val_np = train.to_numpy(), test.to_numpy(), val.to_numpy()
train_np.shape, test_np.shape, val_np.shape
```

```
((240000, 31), (22000, 31), (22807, 31))
```

```
#Splitting the features and labels from the training, testing and validation NumPy arrays.
x_train, y_train = train_np[:, :-1], train_np[:, -1]
x_test, y_test = test_np[:, :-1], test_np[:, -1]
x_val, y_val = val_np[:, :-1], val_np[:, -1]
x_train.shape, y_train.shape, x_test.shape, y_test.shape, x_val.shape, y_val.shape
```

```
((240000, 30), (240000,), (22000, 30), (22000,), (22807, 30), (22807,))
```

```
#Conducting the logistics regression model using the training dataframe
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
logistic_model.fit(x_train, y_train)
logistic_model.score(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
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(
0.9991875
```

The accuracy shows a high percentage, this is a misinterpretation by the logistics regression model based on the overwhelming one sided nature of the dataset. Meaning a lot more Fraudulant creditcards were passed than we would like.

```
#Creating a classification report to determine if the model was a good fit.
from sklearn.metrics import classification_report
print(classification_report(y_val, logistic_model.predict(x_val), target_names=['Not Fraud', 'Fraud']))
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22772
       Fraud       0.81      0.49      0.61        35

    accuracy                           1.00     22807
   macro avg       0.90      0.74      0.80     22807
weighted avg       1.00      1.00      1.00     22807
```

Using tensorflow neural network, we will train the data to be able to more likely predict the fraudulant creditcard transactions to obtain a better precision and recall score.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Dense, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint

#Creating a neural network using Tensorflow Keras to help create a good fit model
shallow_nn = Sequential()
shallow_nn.add(InputLayer((x_train.shape[1],)))
#Adds a connected layer with 2 neurons using rectified linear unit activation function
shallow_nn.add(Dense(2, 'relu'))
shallow_nn.add(BatchNormalization())
#Adds a dense layer with 1 neuron and a sigmoid activation function, which will be suitable for a binary classification.
shallow_nn.add(Dense(1, 'sigmoid'))

checkpoint = ModelCheckpoint('shallow_nn', save_best_only=True)
shallow_nn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
shallow_nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 62

 batch_normalization (Batch  (None, 2)                 8
 Normalization)

 dense_1 (Dense)             (None, 1)                 3

=================================================================
Total params: 73 (292.00 Byte)
Trainable params: 69 (276.00 Byte)
Non-trainable params: 4 (16.00 Byte)
_________________________________________________________________
```

```
shallow_nn.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=5, callbacks=checkpoint)
```

```
Epoch 1/5
7500/7500 [==============================] - 20s 3ms/step - loss: 0.0543 - accuracy: 0.9899 - val_loss: 0.0139 - val_accuracy: 0.998
Epoch 2/5
7500/7500 [==============================] - 25s 3ms/step - loss: 0.0034 - accuracy: 0.9993 - val_loss: 0.0138 - val_accuracy: 0.998
Epoch 3/5
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0033 - accuracy: 0.9993 - val_loss: 0.0120 - val_accuracy: 0.998
Epoch 4/5
```

```
7500/7500 [==============================] - 26s 3ms/step - loss: 0.0031 - accuracy: 0.9994 - val_loss: 0.0089 - val_accuracy: 0.998
Epoch 5/5
7500/7500 [==============================] - 25s 3ms/step - loss: 0.0033 - accuracy: 0.9993 - val_loss: 0.0096 - val_accuracy: 0.999
<keras.src.callbacks.History at 0x7f7618267a90>
```

```
#Creating a function that takes a trained neural network model and input data and returns binary predictions based on a threshold of 0.5
def neural_net_predictions(model, x):
  return (model.predict(x).flatten() > 0.5).astype(int)
neural_net_predictions(shallow_nn, x_val)
```

```
713/713 [==============================] - 1s 2ms/step
array([0, 0, 0, ..., 0, 0, 0])
```

```
print(classification_report(y_val, neural_net_predictions(shallow_nn, x_val), target_names=['Not Fraud', 'Fraud']))
```

```
713/713 [==============================] - 2s 2ms/step
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22772
       Fraud       0.66      0.71      0.68        35

    accuracy                           1.00     22807
   macro avg       0.83      0.86      0.84     22807
weighted avg       1.00      1.00      1.00     22807
```

Using various other machine learning libraries to find the best model fit for the dataset

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(max_depth=2, n_jobs=-1)
rf.fit(x_train, y_train)
print(classification_report(y_val, rf.predict(x_val), target_names=['Not Fraud', 'Fraud']))
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22772
       Fraud       0.70      0.40      0.51        35

    accuracy                           1.00     22807
   macro avg       0.85      0.70      0.75     22807
weighted avg       1.00      1.00      1.00     22807
```

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier(n_estimators=50, learning_rate=1.0, max_depth=1, random_state=0)
gbc.fit(x_train, y_train)
print(classification_report(y_val, gbc.predict(x_val), target_names=['Not Fraud', 'Fraud']))
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22772
       Fraud       0.62      0.57      0.60        35

    accuracy                           1.00     22807
   macro avg       0.81      0.79      0.80     22807
weighted avg       1.00      1.00      1.00     22807
```

```
from sklearn.svm import LinearSVC
svc = LinearSVC(class_weight='balanced')
svc.fit(x_train, y_train)
print(classification_report(y_val, svc.predict(x_val), target_names=['Not Fraud', 'Fraud']))
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22772
       Fraud       0.61      0.71      0.66        35

    accuracy                           1.00     22807
   macro avg       0.80      0.86      0.83     22807
weighted avg       1.00      1.00      1.00     22807

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the nu
  warnings.warn(
```

Because of the vast imbalance of the dataset, the accuracy throughout all of the models have been greatly scewed and unreliable. Therefore, the best solution is to have the not frauds equal or be very close to the frauds, meaning there will be a greatly reduced dataset but will hopefully provide better insight on detecting fraudulant credit payments.

```
not_frauds = new_df.query('Class == 0')
frauds = new_df.query('Class == 1')
not_frauds['Class'].value_counts(), frauds['Class'].value_counts()
```

```
(Class
 0    284315
 Name: count, dtype: int64,
 Class
 1    492
 Name: count, dtype: int64)
```

```
balanced_df = pd.concat([frauds, not_frauds.sample(len(frauds), random_state=1)])
balanced_df['Class'].value_counts()
```

```
Class
1    492
0    492
Name: count, dtype: int64
```

```
balanced_df = balanced_df.sample(frac=1, random_state=4)
balanced_df
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
220565	0.823088	0.360727	-0.071976	-0.472297	-1.657803	0.423601	-0.623627	0.489919	-0.027917	-0.762730	...	-0.311176	-0.415651
60596	0.285870	1.266928	0.333935	0.065557	0.997921	-0.007024	-0.654947	0.224890	-0.207205	0.001958	...	-0.001760	0.032588
73784	0.319916	-5.753852	0.577610	-6.312782	5.159401	-1.698320	-2.683286	-7.934389	2.373550	-3.073079	...	1.177852	0.175331
162319	0.665610	-0.223628	0.928293	-0.303305	-0.899439	1.846683	1.484353	0.843002	0.181299	0.003342	...	0.371886	1.674868
172997	0.702104	1.950496	-0.071947	-0.904485	1.945975	0.205477	0.016533	-0.034238	-0.126755	-0.471526	...	0.015326	0.141488
...	...	...	...	...	...	...	...	...	...	...	...	...	...
26945	0.198603	1.228194	-0.440688	-0.355368	-0.311012	1.431263	3.957330	-1.199332	1.071137	0.797731	...	-0.142254	-0.377075
266924	0.940663	2.110242	-0.174260	-1.628030	-0.014880	0.543329	-0.114451	0.029630	-0.078812	0.434357	...	-0.330361	-0.871920
191359	0.747847	1.177824	2.487103	-5.330608	5.324547	1.150243	-1.281843	-1.171994	0.413778	-2.659840	...	0.262325	-0.431790
163149	0.669539	-1.550273	1.088689	-2.393388	1.008733	-1.087562	-1.104602	-2.670503	0.147655	-0.978626	...	0.802316	1.037105
252774	0.902617	-1.201398	4.864535	-8.328823	7.652399	-0.167445	-2.767695	-3.176421	1.623279	-4.367228	...	0.532320	-0.556913

984 rows × 31 columns

```
492*2
```

```
984
```

```
balanced_df_np = balanced_df.to_numpy()

x_train_b, y_train_b = balanced_df_np[:700, :-1], balanced_df_np[:700, -1].astype(int)
x_test_b, y_test_b = balanced_df_np[700:842, :-1], balanced_df_np[700:842, -1].astype(int)
x_val_b, y_val_b = balanced_df_np[842:, :-1], balanced_df_np[842:, -1].astype(int)
x_train_b.shape, y_train_b.shape, x_test_b.shape, y_test_b.shape, x_val_b.shape, y_val_b.shape
```

```
((700, 30), (700,), (142, 30), (142,), (142, 30), (142,))
```

Reconducting all the model fit training and visualisations from the unbalanced dataframe

```
logistic_model_b = LogisticRegression()
logistic_model_b.fit(x_train_b, y_train_b)
logistic_model_b.score(x_train_b, y_train_b)
print(classification_report(y_val_b, logistic_model_b.predict(x_val_b), target_names=['Not Fraud', 'Fraud']))
```

```
              precision    recall  f1-score   support

   Not Fraud       0.91      0.95      0.93        64
       Fraud       0.96      0.92      0.94        78

    accuracy                           0.94       142
   macro avg       0.94      0.94      0.94       142
weighted avg       0.94      0.94      0.94       142
```

```
shallow_nn_b = Sequential()
shallow_nn_b.add(InputLayer((x_train.shape[1],)))
shallow_nn_b.add(Dense(2, 'relu'))
shallow_nn_b.add(BatchNormalization())
```

```
shallow_nn_b.add(Dense(1, 'sigmoid'))

checkpoint = ModelCheckpoint('shallow_nn_b', save_best_only=True)
shallow_nn_b.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
shallow_nn_b.fit(x_train_b, y_train_b, validation_data=(x_val_b, y_val_b), epochs=40, callbacks=checkpoint)
```

```
Epoch 1/40
22/22 [==============================] - 2s 46ms/step - loss: 0.6237 - accuracy: 0.6657 - val_loss: 0.7075 - val_accuracy: 0.5986
Epoch 2/40
22/22 [==============================] - 1s 37ms/step - loss: 0.5086 - accuracy: 0.7843 - val_loss: 0.5711 - val_accuracy: 0.7254
Epoch 3/40
22/22 [==============================] - 1s 38ms/step - loss: 0.4374 - accuracy: 0.8357 - val_loss: 0.4822 - val_accuracy: 0.8451
Epoch 4/40
22/22 [==============================] - 1s 37ms/step - loss: 0.3869 - accuracy: 0.8786 - val_loss: 0.4298 - val_accuracy: 0.8944
Epoch 5/40
22/22 [==============================] - 1s 38ms/step - loss: 0.3706 - accuracy: 0.8700 - val_loss: 0.3940 - val_accuracy: 0.9155
Epoch 6/40
22/22 [==============================] - 1s 36ms/step - loss: 0.3605 - accuracy: 0.8757 - val_loss: 0.3725 - val_accuracy: 0.9014
Epoch 7/40
22/22 [==============================] - 1s 50ms/step - loss: 0.3495 - accuracy: 0.8771 - val_loss: 0.3603 - val_accuracy: 0.9085
Epoch 8/40
22/22 [==============================] - 2s 102ms/step - loss: 0.3385 - accuracy: 0.8829 - val_loss: 0.3498 - val_accuracy: 0.901
Epoch 9/40
22/22 [==============================] - 3s 134ms/step - loss: 0.3274 - accuracy: 0.8929 - val_loss: 0.3418 - val_accuracy: 0.894
Epoch 10/40
22/22 [==============================] - 3s 153ms/step - loss: 0.3223 - accuracy: 0.8929 - val_loss: 0.3357 - val_accuracy: 0.894
Epoch 11/40
22/22 [==============================] - 2s 104ms/step - loss: 0.3153 - accuracy: 0.8943 - val_loss: 0.3312 - val_accuracy: 0.894
Epoch 12/40
22/22 [==============================] - 2s 91ms/step - loss: 0.3110 - accuracy: 0.8943 - val_loss: 0.3263 - val_accuracy: 0.9014
Epoch 13/40
22/22 [==============================] - 2s 84ms/step - loss: 0.3078 - accuracy: 0.8929 - val_loss: 0.3226 - val_accuracy: 0.9014
Epoch 14/40
22/22 [==============================] - 2s 76ms/step - loss: 0.3042 - accuracy: 0.9000 - val_loss: 0.3179 - val_accuracy: 0.9014
Epoch 15/40
22/22 [==============================] - 1s 43ms/step - loss: 0.2949 - accuracy: 0.8943 - val_loss: 0.3131 - val_accuracy: 0.8944
Epoch 16/40
22/22 [==============================] - 1s 44ms/step - loss: 0.2838 - accuracy: 0.9043 - val_loss: 0.3112 - val_accuracy: 0.8873
Epoch 17/40
22/22 [==============================] - 3s 120ms/step - loss: 0.2731 - accuracy: 0.9129 - val_loss: 0.3078 - val_accuracy: 0.887
Epoch 18/40
22/22 [==============================] - 1s 64ms/step - loss: 0.2679 - accuracy: 0.9114 - val_loss: 0.3035 - val_accuracy: 0.8873
Epoch 19/40
22/22 [==============================] - 1s 48ms/step - loss: 0.2687 - accuracy: 0.9029 - val_loss: 0.3001 - val_accuracy: 0.8873
Epoch 20/40
22/22 [==============================] - 1s 40ms/step - loss: 0.2605 - accuracy: 0.9057 - val_loss: 0.2960 - val_accuracy: 0.8873
Epoch 21/40
22/22 [==============================] - 1s 38ms/step - loss: 0.2601 - accuracy: 0.9114 - val_loss: 0.2930 - val_accuracy: 0.8873
Epoch 22/40
22/22 [==============================] - 1s 36ms/step - loss: 0.2503 - accuracy: 0.9143 - val_loss: 0.2905 - val_accuracy: 0.8873
Epoch 23/40
22/22 [==============================] - 1s 36ms/step - loss: 0.2441 - accuracy: 0.9143 - val_loss: 0.2871 - val_accuracy: 0.8873
Epoch 24/40
22/22 [==============================] - 1s 39ms/step - loss: 0.2404 - accuracy: 0.9200 - val_loss: 0.2829 - val_accuracy: 0.8873
Epoch 25/40
22/22 [==============================] - 1s 53ms/step - loss: 0.2360 - accuracy: 0.9129 - val_loss: 0.2802 - val_accuracy: 0.8873
Epoch 26/40
22/22 [==============================] - 1s 42ms/step - loss: 0.2256 - accuracy: 0.9271 - val_loss: 0.2792 - val_accuracy: 0.8873
Epoch 27/40
22/22 [==============================] - 1s 38ms/step - loss: 0.2357 - accuracy: 0.9214 - val_loss: 0.2764 - val_accuracy: 0.8873
Epoch 28/40
22/22 [==============================] - 1s 45ms/step - loss: 0.2309 - accuracy: 0.9214 - val_loss: 0.2729 - val_accuracy: 0.8944
Epoch 29/40
```

```
print(classification_report(y_val_b, neural_net_predictions(shallow_nn_b, x_val_b), target_names=['Not Fraud', 'Fraud']))
```

```
5/5 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

   Not Fraud       0.89      0.98      0.93        64
       Fraud       0.99      0.90      0.94        78

    accuracy                           0.94       142
   macro avg       0.94      0.94      0.94       142
weighted avg       0.94      0.94      0.94       142
```

```
shallow_nn_b1 = Sequential()
shallow_nn_b1.add(InputLayer((x_train.shape[1],)))
#Reducing the number of neurons for the relu function down to 1 to see if this will improve the f1 score
shallow_nn_b1.add(Dense(1, 'relu'))
shallow_nn_b1.add(BatchNormalization())
shallow_nn_b1.add(Dense(1, 'sigmoid'))

checkpoint = ModelCheckpoint('shallow_nn_b1', save_best_only=True)
shallow_nn_b1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
shallow_nn_b1.fit(x_train_b, y_train_b, validation_data=(x_val_b, y_val_b), epochs=40, callbacks=checkpoint)
```

```
Epoch 11/40
22/22 [==============================] - 1s 36ms/step - loss: 0.4882 - accuracy: 0.8757 - val_loss: 0.5125 - val_accuracy: 0.8732
Epoch 12/40
22/22 [==============================] - 1s 39ms/step - loss: 0.4745 - accuracy: 0.8857 - val_loss: 0.5043 - val_accuracy: 0.8803
Epoch 13/40
22/22 [==============================] - 1s 39ms/step - loss: 0.4622 - accuracy: 0.8986 - val_loss: 0.4951 - val_accuracy: 0.8803
Epoch 14/40
22/22 [==============================] - 1s 47ms/step - loss: 0.4506 - accuracy: 0.8914 - val_loss: 0.4837 - val_accuracy: 0.8803
Epoch 15/40
22/22 [==============================] - 1s 56ms/step - loss: 0.4440 - accuracy: 0.9100 - val_loss: 0.4705 - val_accuracy: 0.8873
Epoch 16/40
22/22 [==============================] - 1s 57ms/step - loss: 0.4336 - accuracy: 0.8971 - val_loss: 0.4572 - val_accuracy: 0.8944
Epoch 17/40
22/22 [==============================] - 1s 63ms/step - loss: 0.4210 - accuracy: 0.9014 - val_loss: 0.4463 - val_accuracy: 0.9014
Epoch 18/40
22/22 [==============================] - 1s 43ms/step - loss: 0.4122 - accuracy: 0.9000 - val_loss: 0.4369 - val_accuracy: 0.9014
Epoch 19/40
22/22 [==============================] - 1s 40ms/step - loss: 0.4083 - accuracy: 0.9029 - val_loss: 0.4249 - val_accuracy: 0.9014
Epoch 20/40
22/22 [==============================] - 1s 41ms/step - loss: 0.3952 - accuracy: 0.9000 - val_loss: 0.4200 - val_accuracy: 0.9014
Epoch 21/40
22/22 [==============================] - 1s 38ms/step - loss: 0.3819 - accuracy: 0.9057 - val_loss: 0.4079 - val_accuracy: 0.9155
Epoch 22/40
22/22 [==============================] - 1s 38ms/step - loss: 0.3759 - accuracy: 0.9014 - val_loss: 0.3967 - val_accuracy: 0.9155
Epoch 23/40
22/22 [==============================] - 2s 74ms/step - loss: 0.3672 - accuracy: 0.9086 - val_loss: 0.3930 - val_accuracy: 0.9155
Epoch 24/40
22/22 [==============================] - 1s 40ms/step - loss: 0.3550 - accuracy: 0.9157 - val_loss: 0.3840 - val_accuracy: 0.9155
Epoch 25/40
22/22 [==============================] - 1s 38ms/step - loss: 0.3514 - accuracy: 0.9229 - val_loss: 0.3731 - val_accuracy: 0.9155
Epoch 26/40
22/22 [==============================] - 1s 38ms/step - loss: 0.3429 - accuracy: 0.9157 - val_loss: 0.3668 - val_accuracy: 0.9155
Epoch 27/40
22/22 [==============================] - 1s 37ms/step - loss: 0.3395 - accuracy: 0.9014 - val_loss: 0.3622 - val_accuracy: 0.9155
Epoch 28/40
22/22 [==============================] - 1s 36ms/step - loss: 0.3321 - accuracy: 0.9129 - val_loss: 0.3609 - val_accuracy: 0.9085
Epoch 29/40
22/22 [==============================] - 1s 54ms/step - loss: 0.3239 - accuracy: 0.9129 - val_loss: 0.3532 - val_accuracy: 0.9085
Epoch 30/40
22/22 [==============================] - 1s 59ms/step - loss: 0.3161 - accuracy: 0.9243 - val_loss: 0.3450 - val_accuracy: 0.9155
Epoch 31/40
22/22 [==============================] - 1s 59ms/step - loss: 0.3043 - accuracy: 0.9300 - val_loss: 0.3398 - val_accuracy: 0.9155
Epoch 32/40
22/22 [==============================] - 1s 60ms/step - loss: 0.3057 - accuracy: 0.9129 - val_loss: 0.3310 - val_accuracy: 0.9155
Epoch 33/40
22/22 [==============================] - 1s 37ms/step - loss: 0.3009 - accuracy: 0.9257 - val_loss: 0.3303 - val_accuracy: 0.9155
Epoch 34/40
22/22 [==============================] - 1s 37ms/step - loss: 0.2979 - accuracy: 0.9100 - val_loss: 0.3247 - val_accuracy: 0.9155
Epoch 35/40
22/22 [==============================] - 1s 40ms/step - loss: 0.2908 - accuracy: 0.9243 - val_loss: 0.3213 - val_accuracy: 0.9155
Epoch 36/40
22/22 [==============================] - 1s 40ms/step - loss: 0.2837 - accuracy: 0.9257 - val_loss: 0.3180 - val_accuracy: 0.9155
Epoch 37/40
22/22 [==============================] - 1s 37ms/step - loss: 0.2773 - accuracy: 0.9314 - val_loss: 0.3084 - val_accuracy: 0.9225
Epoch 38/40
22/22 [==============================] - 1s 38ms/step - loss: 0.2749 - accuracy: 0.9300 - val_loss: 0.3071 - val_accuracy: 0.9155
Epoch 39/40
22/22 [==============================] - 1s 38ms/step - loss: 0.2789 - accuracy: 0.9271 - val_loss: 0.3008 - val_accuracy: 0.9155
Epoch 40/40
```

```
print(classification_report(y_val_b, neural_net_predictions(shallow_nn_b1, x_val_b), target_names=['Not Fraud', 'Fraud']))
```

```
5/5 [==============================] - 0s 5ms/step
              precision    recall  f1-score   support

   Not Fraud       0.85      0.99      0.91        72
       Fraud       0.98      0.81      0.89        70

    accuracy                           0.90       142
   macro avg       0.91      0.90      0.90       142
weighted avg       0.91      0.90      0.90       142
```

```
rf_b = RandomForestClassifier(max_depth=2, n_jobs=-1)
rf_b.fit(x_train_b, y_train_b)
print(classification_report(y_val_b, rf_b.predict(x_val_b), target_names=['Not Fraud', 'Fraud']))
```

```
              precision    recall  f1-score   support

   Not Fraud       0.92      1.00      0.96        72
       Fraud       1.00      0.91      0.96        70

    accuracy                           0.96       142
   macro avg       0.96      0.96      0.96       142
weighted avg       0.96      0.96      0.96       142
```