```
import pandas as pd
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Dense,
BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
df = pd.read_csv('creditcard.csv')
```

```
df.head(10) #print first 10 rows
```

```
   Time        V1        V2        V3        V4        V5        V6
V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -
0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
0.592941
5   2.0 -0.425966  0.960523  1.141109 -0.168252  0.420987 -0.029728
0.476201
6   4.0  1.229658  0.141004  0.045371  1.202613  0.191881  0.272708 -
0.005159
7   7.0 -0.644269  1.417964  1.074380 -0.492199  0.948934  0.428118
1.120631
8   7.0 -0.894286  0.286157 -0.113192 -0.271526  2.669599  3.721818
0.370145
9   9.0 -0.338262  1.119593  1.044367 -0.222187  0.499361 -0.246761
0.651583

         V8        V9  ...       V21       V22       V23       V24
V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928
0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846
0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -
0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575
0.647376
```

```
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -
0.206010
5  0.260314 -0.568671  ... -0.208254 -0.559825 -0.026398 -0.371427 -
0.232794
6  0.081213  0.464960  ... -0.167716 -0.270710 -0.154104 -0.780055
0.750137
7 -3.807864  0.615375  ...  1.943465 -1.015455  0.057504 -0.649709 -
0.415267
8  0.851084 -0.392048  ... -0.073425 -0.268092 -0.204233  1.011592
0.373205
9  0.069539 -0.736727  ... -0.246914 -0.633753 -0.120794 -0.385050 -
0.069733

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
5  0.105915  0.253844  0.081080    3.67      0
6 -0.257237  0.034507  0.005168    4.99      0
7 -0.051634 -1.206921 -1.085339   40.80      0
8 -0.384157  0.011747  0.142404   93.20      0
9  0.094199  0.246219  0.083076    3.68      0

[10 rows x 31 columns]
```

```
df.describe()
```

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

                  V5            V6            V7            V8
V9  \
```

```
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -
2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00
1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -
1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -
6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -
5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01
5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01
1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28
Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
284807.000000
mean   5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16
88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01
250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01
0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02
5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02
22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02
77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01
25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
```

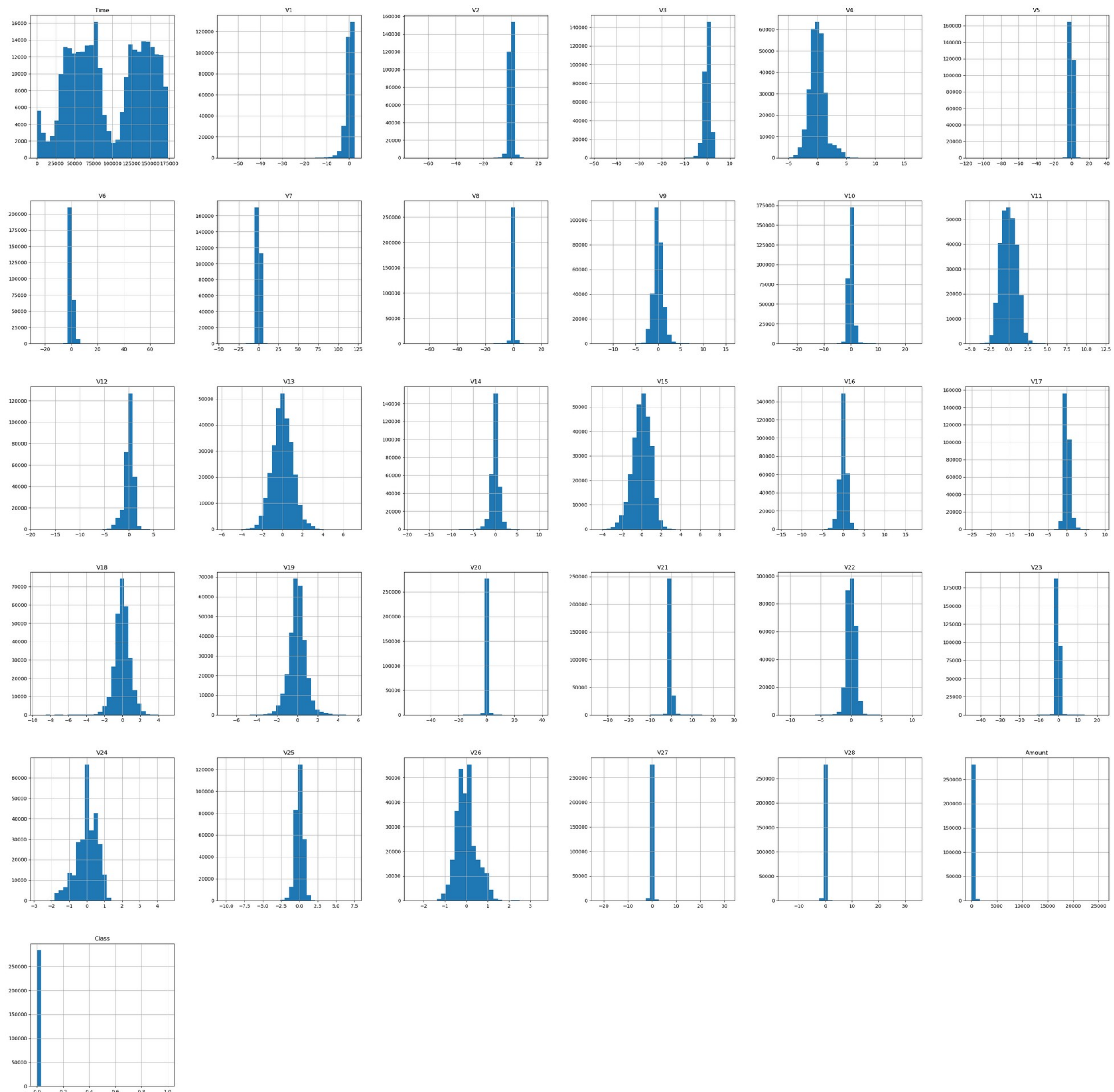
```
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000

[8 rows x 31 columns]
```

```
df['Class'].value_counts() #count the occurence of 2 classes
```

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

```
df.hist(bins=30, figsize=(40, 40))
```

```
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)
```




```
#DATA PRE PROCESSING
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())

#Shulfing the rows
new_df = new_df.sample(frac=1, random_state=1)
new_df.head(10)
```

```
             Time        V1        V2        V3        V4        V5
V6  \
169876  0.693938 -0.611712 -0.769705 -0.149759 -0.224877  2.028577 -
2.019887
127467  0.453377 -0.814682  1.319219  1.329415  0.027273 -0.284871 -
0.653985
137900  0.476770 -0.318193  1.118618  0.969864 -0.127052  0.569563 -
0.532484
21513   0.183556 -1.328271  1.018378  1.775426 -1.574193 -0.117696 -
0.457733
134700  0.468326  1.276712  0.617120 -0.578014  0.879173  0.061706 -
1.472002
196117  0.760244  0.077197  0.482928 -2.234233 -1.309124  2.386570
3.392581
24533   0.192567 -0.958584  1.109086  1.558159  0.878707  1.914559
1.564757
13629   0.139810 -0.992899  1.430204  1.071256  1.363127  0.116315
0.217868
246673  0.887055 -1.143693 -0.250983  1.013022 -0.671080  1.363438
0.312673
91842   0.368356  0.555043 -0.099484 -0.102234 -0.624145  1.484364
4.154536

              V7        V8        V9  ...       V21       V22
V23  \
169876  0.292491 -0.523020  0.358468  ... -0.075208  0.045536
0.380739
127467  0.321552  0.435975 -0.704298  ... -0.128619 -0.368565
0.090660
137900  0.706252 -0.064966 -0.463271  ... -0.305402 -0.774704 -
0.123884
21513   0.681867 -0.031641  0.383872  ... -0.220815 -0.419013 -
0.239197
134700  0.373692 -0.287204 -0.084482  ... -0.160161 -0.430404 -
0.076738
196117 -0.156385  1.353569 -0.047112  ...  0.060865 -0.060530
0.379575
24533   1.300978  0.074098 -1.376977  ...  0.155271  0.657607 -
0.296222
13629   0.208391  0.319128  1.483134  ... -0.258903 -0.104189 -
0.100144
246673  0.786158 -0.089323 -0.272429  ... -0.094134 -0.137349 -
0.047086
91842  -1.242699  0.286054  0.694670  ...  0.649935 -0.578303
0.057823

             V24       V25       V26       V27       V28    Amount
Class
169876  0.023440 -2.220686 -0.201146  0.066501  0.221180 -0.282401
0
```

```
127467  0.401147 -0.261034  0.080621  0.162427  0.059456 -0.279746
0
137900 -0.495687 -0.018148  0.121679  0.249050  0.092516 -0.294977
0
21513   0.009967  0.232829  0.814177  0.098797 -0.004273 -0.084119
0
134700  0.258708  0.552170  0.370701 -0.034255  0.041709 -0.296793
0
196117  0.598853 -0.878618  0.254859 -0.088550 -0.023446  0.669322
0
24533  -1.053362  0.006475 -0.058981 -0.528679 -0.374450 -0.215888
0
13629  -0.369103 -0.068048 -0.266731  0.080402 -0.034571 -0.293440
0
246673  0.058485  0.825118  0.316019 -0.377194 -0.246404  0.868441
0
91842   1.026542  0.440906  0.303285  0.146932  0.172708 -0.066513
0

[10 rows x 31 columns]
```

```
#Data splitting into train, test, validation datasets
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    239589
 1       411
 Name: count, dtype: int64,
 Class
 0    21955
 1       45
 Name: count, dtype: int64,
 Class
 0    22771
 1       36
 Name: count, dtype: int64)
```

```
#convert dataframes into array
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))
```

```
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,))
```

```
#LOGISTIC REGRESSION
# Scale the input features
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)

logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(x_train_scaled, y_train)

train_accuracy = logistic_model.score(x_train_scaled, y_train)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.9992375
```

```
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     22771
       Fraud       0.37      0.61      0.46        36

    accuracy                           1.00     22807
   macro avg       0.69      0.80      0.73     22807
weighted avg       1.00      1.00      1.00     22807
```

```
#RANDOM FOREST CLASSIFIER
# Create a RandomForestClassifier instance with specified parameters
rf = RandomForestClassifier(max_depth=2, n_jobs=-1)

# Train the RandomForestClassifier on the training data
rf.fit(x_train, y_train)

# Make predictions on the validation set and print classification
report
predictions = rf.predict(x_val)
report = classification_report(y_val, predictions, target_names=['Not
Fraud', 'Fraud'])
print(report)
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22771
       Fraud       0.77      0.47      0.59        36

    accuracy                           1.00     22807
```

```
   macro avg       0.89      0.74      0.79     22807
weighted avg       1.00      1.00      1.00     22807
```

```
#GRADIENT BOOSTING CLASSIFIER
# Define the parameters
params = {
    'n_estimators': 50,
    'learning_rate': 1.0,
    'max_depth': 1,
    'random_state': 0
}
# Create a GradientBoostingClassifier instance with specified 
parameters
gbc = GradientBoostingClassifier(**params)

# Train the GradientBoostingClassifier on the training data
gbc.fit(x_train, y_train)

# Make predictions on the validation set and print classification 
report
predictions = gbc.predict(x_val)
report = classification_report(y_val, predictions, target_names=['Not 
Fraud', 'Fraud'])
print(report)
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22771
       Fraud       0.67      0.67      0.67        36

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

```
#LINEAR SVC
# Create a LinearSVC instance with balanced class weights
svc = LinearSVC(class_weight='balanced',max_iter=50)

# Train the LinearSVC on the training data
svc.fit(x_train, y_train)

# Make predictions on the validation set and print classification 
report
predictions = svc.predict(x_val)
report = classification_report(y_val, predictions, target_names=['Not 
Fraud', 'Fraud'])
print(report)
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22771
       Fraud       0.78      0.69      0.74        36

    accuracy                           1.00     22807
   macro avg       0.89      0.85      0.87     22807
weighted avg       1.00      1.00      1.00     22807


C:\Users\meena\anaconda3\Lib\site-packages\sklearn\svm\_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.
  warnings.warn(
```

```
#SHALLOW NEURAL NETWORK
# Define the shallow neural network model
shallow_nn = Sequential([
    InputLayer(input_shape=(x_train.shape[1],)),
    Dense(2, activation='relu'),
    BatchNormalization(),
    Dense(1, activation='sigmoid')
])

# Define the checkpoint to save the best model
checkpoint = ModelCheckpoint('shallow_nn.keras', save_best_only=True)

# Compile the model
shallow_nn.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Display model summary
shallow_nn.summary()

# Train the model
shallow_nn.fit(x_train, y_train, validation_data=(x_val, y_val),
epochs=5, callbacks=[checkpoint])

# Define a function for making predictions
def neural_net_predictions(model, x):
    return (model.predict(x).flatten() > 0.5).astype(int)

# Make predictions and print classification report
predictions = neural_net_predictions(shallow_nn, x_val)
report = classification_report(y_val, predictions, target_names=['Not
Fraud', 'Fraud'])
print(report)
```

```
C:\Users\meena\anaconda3\Lib\site-packages\keras\src\layers\core\
input_layer.py:25: UserWarning: Argument `input_shape` is deprecated.
```

```
Use `shape` instead.
  warnings.warn(

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ dense (Dense)                         │ (None, 2)                    │
62 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ batch_normalization                   │ (None, 2)                    │
8 │
│ (BatchNormalization)                  │                              │
│
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_1 (Dense)                       │ (None, 1)                    │
3 │
└───────────────────────────────────────┴──────────────────────────────┴
─────────────────┘

 Total params: 73 (292.00 B)

 Trainable params: 69 (276.00 B)

 Non-trainable params: 4 (16.00 B)

Epoch 1/5
7500/7500 ━━━━━━━━━━━━━━━━━━━━ 10s 1ms/step - accuracy: 0.9458 - loss:
0.1726 - val_accuracy: 0.9990 - val_loss: 0.0152
Epoch 2/5
7500/7500 ━━━━━━━━━━━━━━━━━━━━ 8s 1ms/step - accuracy: 0.9993 - loss:
0.0039 - val_accuracy: 0.9991 - val_loss: 0.0142
Epoch 3/5
7500/7500 ━━━━━━━━━━━━━━━━━━━━ 8s 1ms/step - accuracy: 0.9993 - loss:
0.0036 - val_accuracy: 0.9992 - val_loss: 0.0100
Epoch 4/5
7500/7500 ━━━━━━━━━━━━━━━━━━━━ 7s 958us/step - accuracy: 0.9994 -
loss: 0.0038 - val_accuracy: 0.9990 - val_loss: 0.0122
Epoch 5/5
7500/7500 ━━━━━━━━━━━━━━━━━━━━ 7s 943us/step - accuracy: 0.9993 -
loss: 0.0034 - val_accuracy: 0.9990 - val_loss: 0.0132
713/713 ━━━━━━━━━━━━━━━━━━━━ 1s 662us/step
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22771
```

```
       Fraud       0.65      0.78      0.71        36

    accuracy                           1.00     22807
   macro avg       0.83      0.89      0.85     22807
weighted avg       1.00      1.00      1.00     22807
```

```
#BALANCING DATASET
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=1)
balanced_df.head()
```

```
            Time        V1        V2        V3        V4        V5
V6  \
18372   0.170309 -1.762593  0.256143  1.683125 -1.279233 -1.902762
1.004210
96341   0.380388  1.227614 -0.668974 -0.271785 -0.589440 -0.604795 -
0.350285
248296  0.890522 -0.613696  3.698772 -5.534941  5.620486  1.649263 -
2.335145
264328  0.933932 -0.011624  0.640413  0.868046 -0.505279  0.261938
0.223098
208904  0.794730 -0.679341  1.217389 -0.316778 -1.086725  0.855349 -
0.980760

              V7        V8        V9  ...       V21       V22
V23  \
18372  -1.009748 -2.432546  0.458860  ...  2.493579  0.320829 -
0.535481
96341  -0.486365 -0.010809 -0.794944  ... -0.026055 -0.295255 -
0.180459
248296 -0.907188  0.706362 -3.747646  ...  0.319261 -0.471379 -
```

```
0.075890
264328  0.239049  0.150877  0.225142  ...  0.069401  0.268024
0.261459
208904  0.970589  0.133116 -0.357671  ... -0.083048 -0.137032 -
0.238920

             V24       V25       V26       V27       V28    Amount
Class
18372   0.499401 -0.915196 -0.423434  0.107049  0.175922  2.906449
0
96341  -0.436539  0.494649 -0.283738 -0.001128  0.035075  1.062111
1
248296 -0.667909 -0.642848  0.070600  0.488410  0.292345 -0.307413
1
264328  0.683742 -1.567901 -0.816674  0.185781  0.283021 -0.272619
0
208904 -0.617244  0.039020 -0.081848  0.234633  0.128382 -0.307273
0

[5 rows x 31 columns]
```

```
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,))
```

```
pd.Series(y_train_b).value_counts(),
pd.Series(y_test_b).value_counts(), pd.Series(y_val_b).value_counts()
```

```
(1    353
 0    347
 Name: count, dtype: int64,
 0    73
 1    69
 Name: count, dtype: int64,
 0    72
 1    70
 Name: count, dtype: int64)
```

```
#LOGISTIC REGRESSION WITH BALANCED DATA
# Create a LogisticRegression instance
logistic_model_b = LogisticRegression()
```

```
# Train the logistic regression model on the training data
logistic_model_b.fit(x_train_b, y_train_b)

# Make predictions on the validation set and print classification
report
predictions = logistic_model_b.predict(x_val_b)
report = classification_report(y_val_b, predictions,
target_names=['Not Fraud', 'Fraud'])
print(report)
```

```
              precision    recall  f1-score   support

   Not Fraud       0.96      0.93      0.94        72
       Fraud       0.93      0.96      0.94        70

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

```
#RANDOM FOREST CLASSIFIER WITH BALANCED DATA
# Create a RandomForestClassifier instance with specified parameters
rf_b = RandomForestClassifier(max_depth=2, n_jobs=-1)

# Train the RandomForestClassifier on the training data
rf_b.fit(x_train_b, y_train_b)

# Make predictions on the validation set and print classification
report
predictions = rf_b.predict(x_val_b)
report = classification_report(y_val_b, predictions,
target_names=['Not Fraud', 'Fraud'])
print(report)
```

```
              precision    recall  f1-score   support

   Not Fraud       0.93      0.97      0.95        72
       Fraud       0.97      0.93      0.95        70

    accuracy                           0.95       142
   macro avg       0.95      0.95      0.95       142
weighted avg       0.95      0.95      0.95       142
```

```
#GRADIENT BOOSTING CLASSIFIER
# Create a GradientBoostingClassifier instance with specified
parameters
gbc_b = GradientBoostingClassifier(n_estimators=50, learning_rate=1.0,
max_depth=2, random_state=0)

# Train the GradientBoostingClassifier on the training data
```

```
gbc_b.fit(x_train_b, y_train_b)

# Make predictions on the validation set and print classification
report
predictions = gbc_b.predict(x_val_b)
report = classification_report(y_val_b, predictions,
target_names=['Not Fraud', 'Fraud'])
print(report)
```

```
              precision    recall  f1-score   support

   Not Fraud       0.94      0.92      0.93        72
       Fraud       0.92      0.94      0.93        70

    accuracy                           0.93       142
   macro avg       0.93      0.93      0.93       142
weighted avg       0.93      0.93      0.93       142
```

```
# Create a LinearSVC instance with balanced class weights
svc_b = LinearSVC(class_weight='balanced')

# Train the LinearSVC on the training data
svc_b.fit(x_train_b, y_train_b)

# Make predictions on the validation set and print classification
report
predictions = svc_b.predict(x_val_b)
report = classification_report(y_val_b, predictions,
target_names=['Not Fraud', 'Fraud'])
print(report)
```

```
              precision    recall  f1-score   support

   Not Fraud       0.96      0.93      0.94        72
       Fraud       0.93      0.96      0.94        70

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

```
C:\Users\meena\anaconda3\Lib\site-packages\sklearn\svm\_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.
  warnings.warn(
```

```
#SHALLOW NEURAL NETWORK

# Define the shallow neural network model
shallow_nn_b = Sequential([
```

```
    InputLayer(input_shape=(x_train_b.shape[1],)),
    Dense(2, activation='relu'),
    BatchNormalization(),
    Dense(1, activation='sigmoid')
])

# Define the checkpoint to save the best model
checkpoint = ModelCheckpoint('shallow_nn_b.keras',
save_best_only=True)

# Compile the model
shallow_nn_b.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
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
```

```
C:\Users\meena\anaconda3\Lib\site-packages\keras\src\layers\core\
input_layer.py:25: UserWarning: Argument `input_shape` is deprecated.
Use `shape` instead.
  warnings.warn(
```

```
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.6818 - loss:
0.6696 - val_accuracy: 0.8028 - val_loss: 0.6096
Epoch 2/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7053 - loss:
0.6089 - val_accuracy: 0.7746 - val_loss: 0.5888
Epoch 3/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7337 - loss:
0.5689 - val_accuracy: 0.7817 - val_loss: 0.5724
Epoch 4/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7548 - loss:
0.5538 - val_accuracy: 0.7958 - val_loss: 0.5592
Epoch 5/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7618 - loss:
0.5383 - val_accuracy: 0.7958 - val_loss: 0.5446
Epoch 6/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8082 - loss:
0.5110 - val_accuracy: 0.8099 - val_loss: 0.5290
Epoch 7/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7909 - loss:
0.5186 - val_accuracy: 0.8310 - val_loss: 0.5117
Epoch 8/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8237 - loss:
0.4868 - val_accuracy: 0.8310 - val_loss: 0.4945
Epoch 9/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8133 - loss:
```

```
0.4875 - val_accuracy: 0.8592 - val_loss: 0.4761
Epoch 10/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8307 - loss:
0.4541 - val_accuracy: 0.8592 - val_loss: 0.4559
Epoch 11/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8687 - loss:
0.4305 - val_accuracy: 0.8662 - val_loss: 0.4341
Epoch 12/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8503 - loss:
0.4324 - val_accuracy: 0.8662 - val_loss: 0.4120
Epoch 13/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8911 - loss:
0.3971 - val_accuracy: 0.8803 - val_loss: 0.3894
Epoch 14/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9044 - loss:
0.3812 - val_accuracy: 0.8944 - val_loss: 0.3693
Epoch 15/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8889 - loss:
0.3570 - val_accuracy: 0.9014 - val_loss: 0.3504
Epoch 16/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9112 - loss:
0.3343 - val_accuracy: 0.9085 - val_loss: 0.3314
Epoch 17/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8927 - loss:
0.3433 - val_accuracy: 0.9155 - val_loss: 0.3134
Epoch 18/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9080 - loss:
0.3044 - val_accuracy: 0.9225 - val_loss: 0.2989
Epoch 19/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9310 - loss:
0.2867 - val_accuracy: 0.9225 - val_loss: 0.2838
Epoch 20/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9260 - loss:
0.2737 - val_accuracy: 0.9225 - val_loss: 0.2706
Epoch 21/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9166 - loss:
0.2727 - val_accuracy: 0.9225 - val_loss: 0.2618
Epoch 22/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9259 - loss:
0.2636 - val_accuracy: 0.9296 - val_loss: 0.2535
Epoch 23/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9181 - loss:
0.2583 - val_accuracy: 0.9296 - val_loss: 0.2459
Epoch 24/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9308 - loss:
0.2453 - val_accuracy: 0.9296 - val_loss: 0.2383
Epoch 25/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9359 - loss:
0.2378 - val_accuracy: 0.9296 - val_loss: 0.2316
```

```
Epoch 26/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9237 - loss:
0.2503 - val_accuracy: 0.9296 - val_loss: 0.2269
Epoch 27/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9140 - loss:
0.2576 - val_accuracy: 0.9296 - val_loss: 0.2218
Epoch 28/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9298 - loss:
0.2197 - val_accuracy: 0.9296 - val_loss: 0.2151
Epoch 29/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9388 - loss:
0.2149 - val_accuracy: 0.9296 - val_loss: 0.2126
Epoch 30/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9306 - loss:
0.2164 - val_accuracy: 0.9296 - val_loss: 0.2090
Epoch 31/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9402 - loss:
0.2168 - val_accuracy: 0.9296 - val_loss: 0.2058
Epoch 32/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9453 - loss:
0.2002 - val_accuracy: 0.9366 - val_loss: 0.2007
Epoch 33/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9330 - loss:
0.2015 - val_accuracy: 0.9507 - val_loss: 0.2000
Epoch 34/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9496 - loss:
0.1637 - val_accuracy: 0.9437 - val_loss: 0.1985
Epoch 35/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9455 - loss:
0.1830 - val_accuracy: 0.9437 - val_loss: 0.1955
Epoch 36/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9425 - loss:
0.1761 - val_accuracy: 0.9507 - val_loss: 0.1955
Epoch 37/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9295 - loss:
0.2030 - val_accuracy: 0.9507 - val_loss: 0.1944
Epoch 38/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9306 - loss:
0.1952 - val_accuracy: 0.9507 - val_loss: 0.1941
Epoch 39/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9312 - loss:
0.2102 - val_accuracy: 0.9507 - val_loss: 0.1970
Epoch 40/40
22/22 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9401 - loss:
0.1836 - val_accuracy: 0.9437 - val_loss: 0.1966

<keras.src.callbacks.history.History at 0x23ff7138850>
```

```python
# Define a function for making predictions
def neural_net_predictions(model, x):
```

```
    return (model.predict(x).flatten() > 0.5).astype(int)

# Make predictions using the shallow neural network model on the
validation set
predictions = neural_net_predictions(shallow_nn_b, x_val_b)

# Print classification report
report = classification_report(y_val_b, predictions,
target_names=['Not Fraud', 'Fraud'])
print(report)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
              precision    recall  f1-score   support

   Not Fraud       0.96      0.93      0.94        72
       Fraud       0.93      0.96      0.94        70

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