Bank Loan Approval Prediction using Artificial Neural Network

In this project, we will build and train a deep neural network model to predict the likelyhood ofa liability customer buying personal loans based on customer features.

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

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy
import matplotlib.pyplot as plt

bank_df = pd.read_csv("UniversalBank.csv")

bank_df.head()
```

```
   ID  Age  Experience  Income  ZIP Code  Family  CCAvg  Education  
Mortgage  \
0   1   25           1      49     91107       4    1.6          1  
0
1   2   45          19      34     90089       3    1.5          1  
0
2   3   39          15      11     94720       1    1.0          1  
0
3   4   35           9     100     94112       1    2.7          2  
0
4   5   35           8      45     91330       4    1.0          2  
0

   Personal Loan  Securities Account  CD Account  Online  CreditCard
0              0                   1           0       0           0
1              0                   1           0       0           0
2              0                   0           0       0           0
3              0                   0           0       0           0
4              0                   0           0       0           1
```

```
bank_df.shape
```

```
(5000, 14)
```

Exploratory Data Analysis

```
bank_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):

-----  ------------------------------------------  ---------
 0   ID                    5000 non-null   int64
 1   Age                   5000 non-null   int64
 2   Experience            5000 non-null   int64
 3   Income                5000 non-null   int64
 4   ZIP Code              5000 non-null   int64
 5   Family                5000 non-null   int64
 6   CCAvg                 5000 non-null   float64
 7   Education             5000 non-null   int64
 8   Mortgage              5000 non-null   int64
 9   Personal Loan         5000 non-null   int64
 10  Securities Account    5000 non-null   int64
 11  CD Account            5000 non-null   int64
 12  Online                5000 non-null   int64
 13  CreditCard            5000 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
```

```
bank df.describe().transpose()
```

```
                  count         mean          std     min
25%  \
ID               5000.0  2500.500000  1443.520003     1.0
1250.75
Age              5000.0    45.338400    11.463166    23.0
35.00
```

```
Experience              5000.0      20.104600    11.467954    -3.0
10.00
Income                  5000.0      73.774200    46.033729     8.0
39.00
ZIP Code                5000.0   93152.503000  2121.852197  9307.0
91911.00
Family                  5000.0       2.396400     1.147663     1.0
1.00
CCAvg                   5000.0       1.937938     1.747659     0.0
0.70
Education               5000.0       1.881000     0.839869     1.0
1.00
Mortgage                5000.0      56.498800   101.713802     0.0
0.00
Personal Loan           5000.0       0.096000     0.294621     0.0
0.00
Securities Account      5000.0       0.104400     0.305809     0.0
0.00
CD Account              5000.0       0.060400     0.238250     0.0
0.00
Online                  5000.0       0.596800     0.490589     0.0
0.00
CreditCard              5000.0       0.294000     0.455637     0.0
0.00

                            50%        75%      max
ID                       2500.5    3750.25   5000.0
Age                        45.0      55.00     67.0
Experience                 20.0      30.00     43.0
Income                     64.0      98.00    224.0
ZIP Code                93437.0   94608.00  96651.0
Family                      2.0       3.00      4.0
CCAvg                       1.5       2.50     10.0
Education                   2.0       3.00      3.0
Mortgage                    0.0     101.00    635.0
Personal Loan               0.0       0.00      1.0
Securities Account          0.0       0.00      1.0
CD Account                  0.0       0.00      1.0
Online                      1.0       1.00      1.0
CreditCard                  0.0       1.00      1.0
```

```
bank_df.isnull().sum()
```

```
[illegible]
CCAvg                       0
```

```
Education               0
Mortgage                0
Personal Loan           0
Securities Account      0
CD Account              0
Online                  0
CreditCard              0
dtype: int64
```

```
avg_age = bank_df["Age"].mean()
print ("The average age of this dataset is {:.1f}.".format(avg_age))
```

```
The average age of this dataset is 45.3.
```

```
percent_cc = sum(bank_df["CreditCard"] == 1)/len(bank_df)
print ("The percentage of customers that own the bank's credit card is
{:.2%}.".format(percent_cc))
```

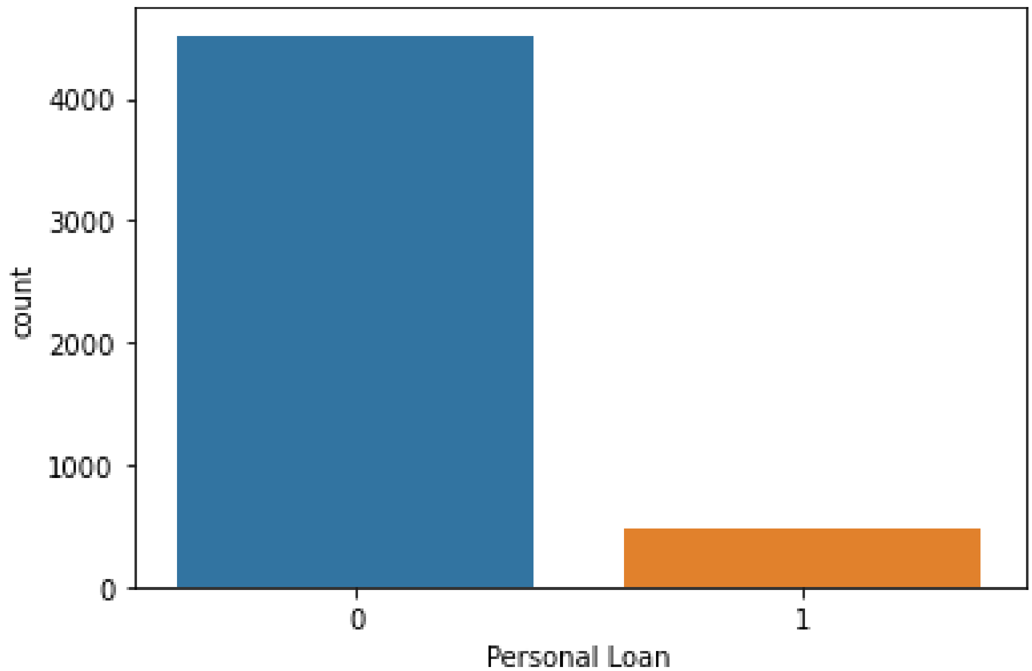
```
The percentage of customers that own the bank's credit card is 29.40%.
```

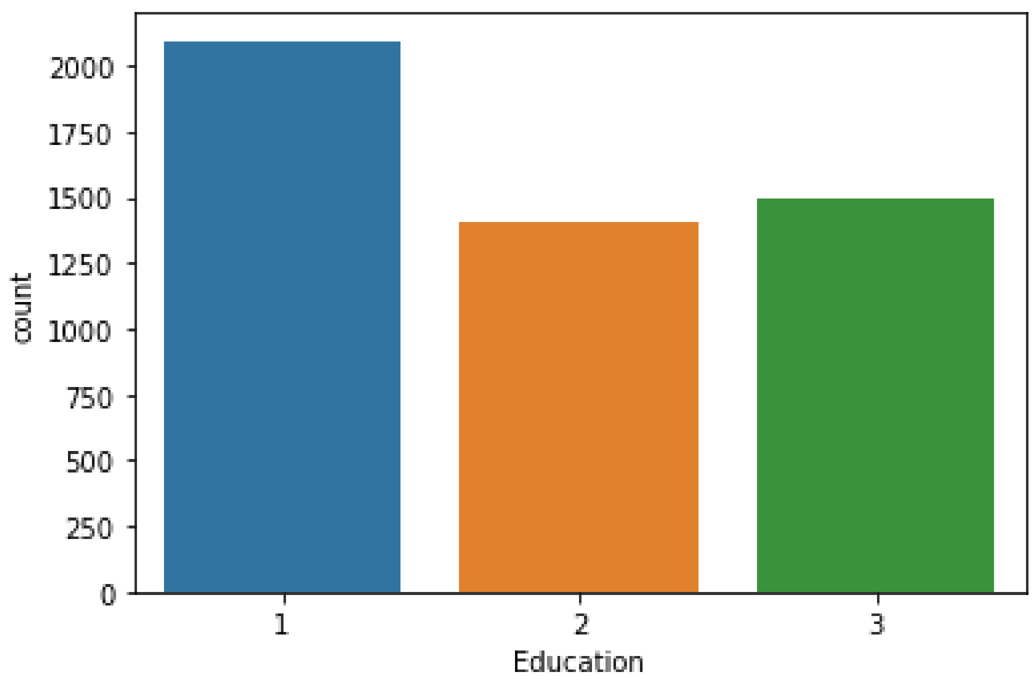
```
percent_loan = sum(bank_df["Personal Loan"] == 1)/len(bank_df)
print ("The percentage of customers that took out a personal loan is
{:.2%}.".format(percent_loan))
```

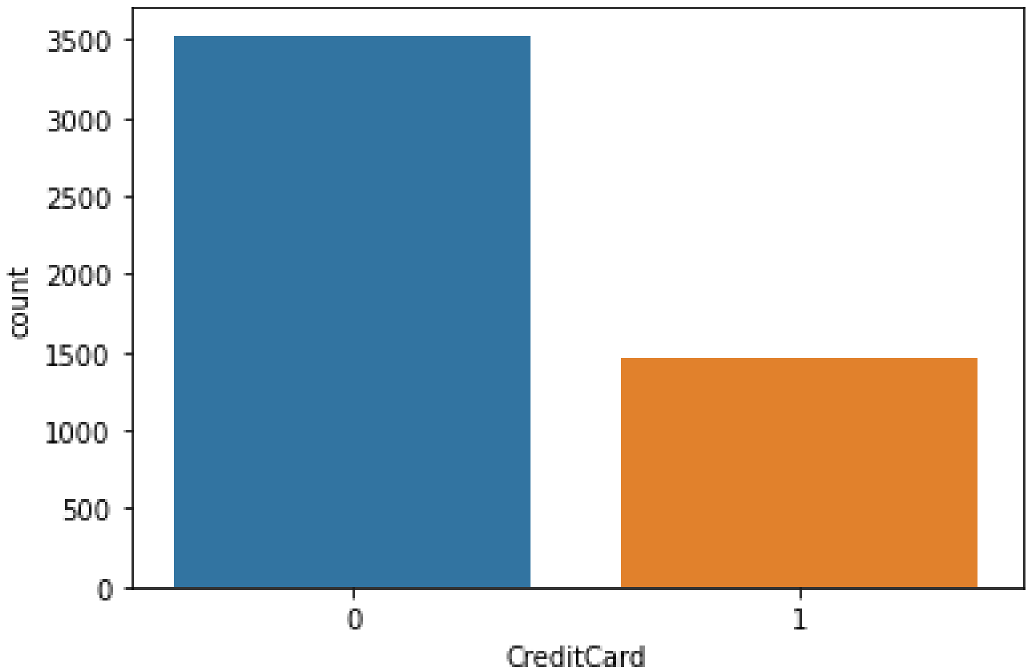
```
The percentage of customers that took out a personal loan is 9.60%.
```

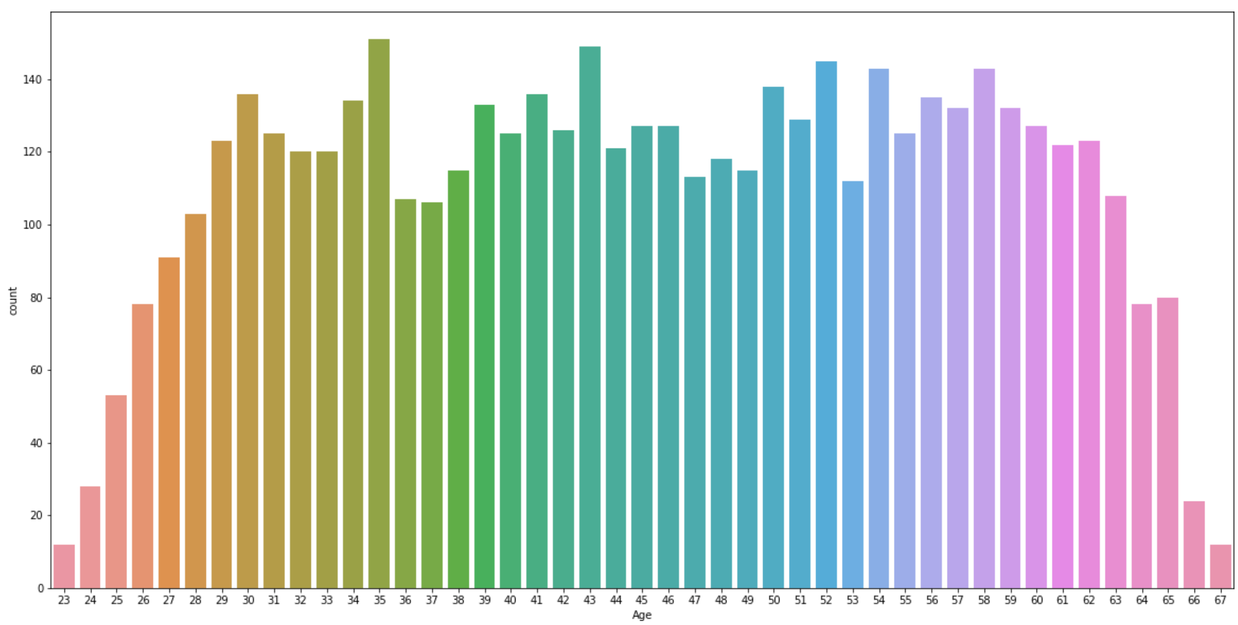
Data Visualization

```
sns.countplot(x=bank_df["Personal Loan"])
plt.show()

sns.countplot(x=bank_df["Education"])
plt.show()

sns.countplot(x=bank_df["CreditCard"])
plt.show()

plt.figure(figsize=(20,10))
sns.countplot(x=bank_df["Age"])
plt.savefig('age.png', facecolor='w', bbox_inches='tight')
plt.show()
```

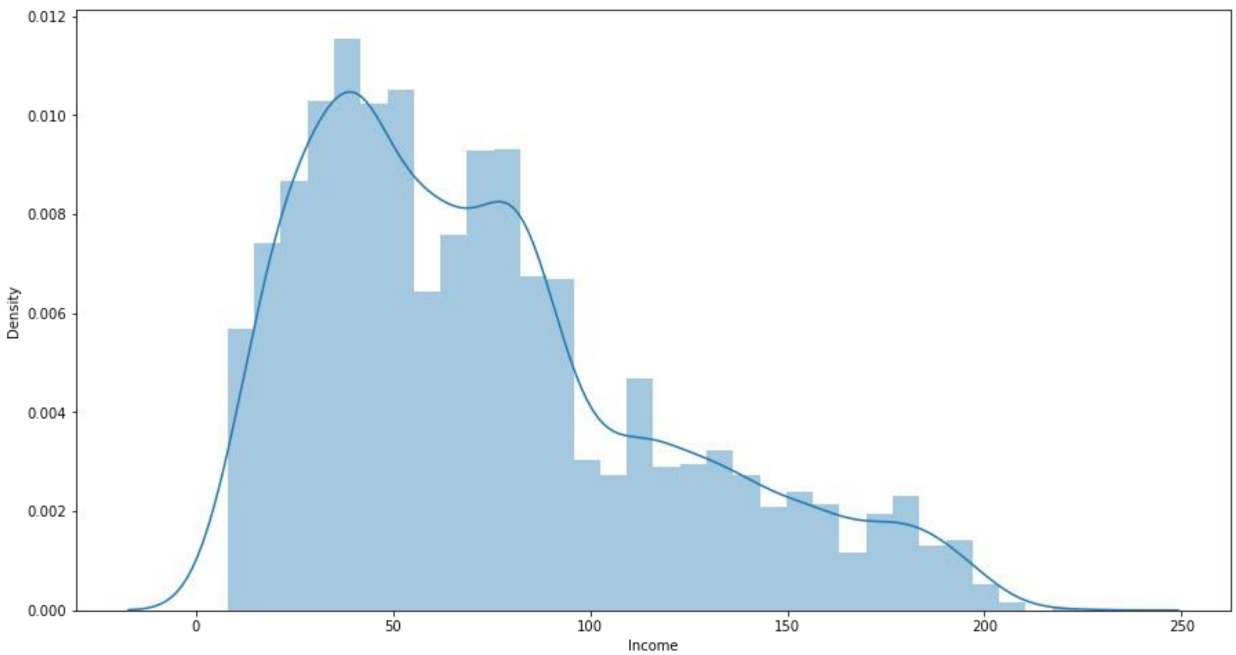

4000
3000
2000
1000
0
count
0
1
Personal Loan


2000
1750
1500
1250
1000
750
500
250
0
count
1
2
3
Education






```
# lets look at the distribution of the income
plt.figure(figsize=(15,8))
sns.distplot(bank_df["Income"])
plt.savefig('income.png', facecolor='w', bbox_inches='tight')
plt.show()
```




```
personal_loans = bank_df[bank_df['Personal Loan'] == 1].copy()
no_personal_loans = bank_df[bank_df['Personal Loan'] == 0].copy()
```

```
personal_loans.describe().T
```

```
                    count          mean          std      min
25%  \
ID                  480.0   2390.650000  1394.393674     10.0
1166.50
Age                 480.0     45.066667    11.590964     26.0
35.00
Experience          480.0     19.843750    11.582443      0.0
9.00
Income              480.0    144.745833    31.584429     60.0
122.00
ZIP Code            480.0  93153.202083  1759.223753  90016.0
91908.75
Family              480.0      2.612500     1.115393      1.0
2.00
CCAvg               480.0      3.905354     2.097681      0.0
2.60
Education           480.0      2.233333     0.753373      1.0
2.00
Mortgage            480.0    100.845833   160.847862      0.0
0.00
Personal Loan       480.0      1.000000     0.000000      1.0
1.00
Securities Account  480.0      0.125000     0.331064      0.0
```

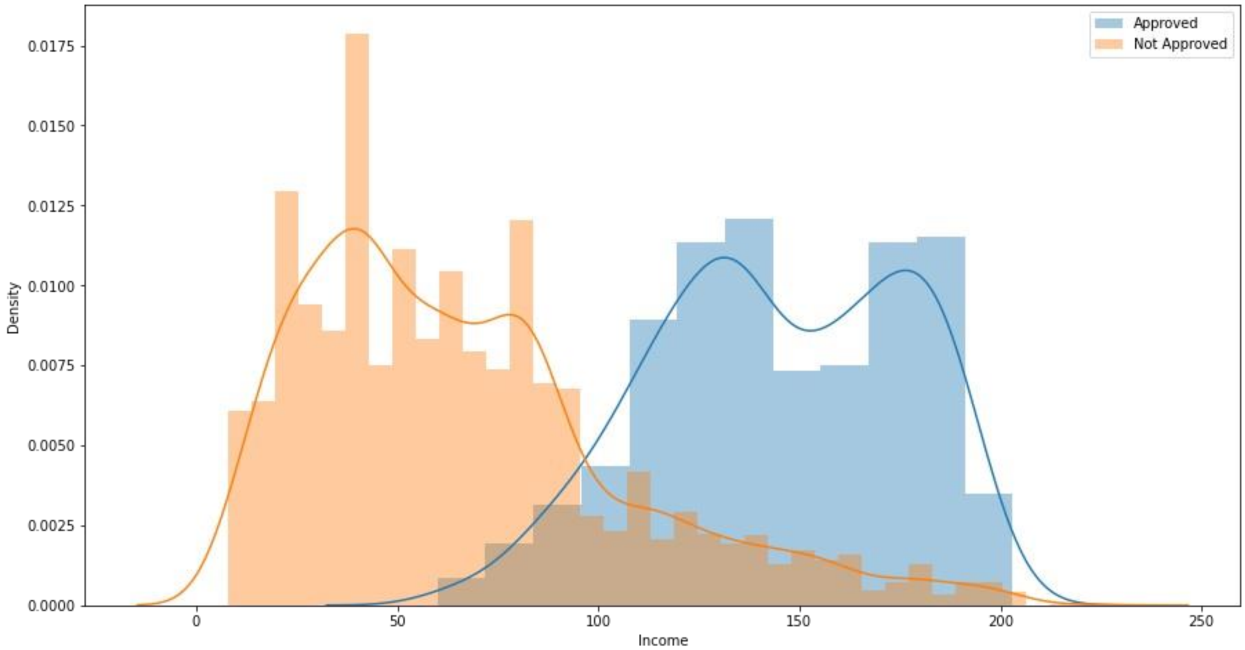
```
0.00
CD Account            480.0      0.291667     0.455004       0.0
0.00
Online                480.0      0.606250     0.489090       0.0
0.00
CreditCard            480.0      0.297917     0.457820       0.0
0.00

                          50%          75%      max
ID                     2342.0    3566.0000   4981.0
Age                      45.0      55.0000     65.0
Experience               20.0      30.0000     41.0
Income                  142.5     172.0000    203.0
ZIP Code              93407.0   94705.5000  96008.0
Family                    3.0       4.0000      4.0
CCAvg                     3.8       5.3475     10.0
Education                 2.0       3.0000      3.0
Mortgage                  0.0     192.5000    617.0
Personal Loan             1.0       1.0000      1.0
Securities Account        0.0       0.0000      1.0
CD Account                0.0       1.0000      1.0
Online                    1.0       1.0000      1.0
CreditCard                0.0       1.0000      1.0
```

```
no_personal_loans.describe().T
```

```
                       count          mean          std      min
25%  \
ID                    4520.0   2512.165487  1448.299331      1.0
1259.75
Age                   4520.0     45.367257    11.450427     23.0
35.00
Experience            4520.0     20.132301    11.456672     -3.0
10.00
Income                4520.0     66.237389    40.578534      8.0
35.00
ZIP Code              4520.0  93152.428761  2156.949654   9307.0
91911.00
Family                4520.0      2.373451     1.148771      1.0
1.00
CCAvg                 4520.0      1.729009     1.567647      0.0
0.60
Education             4520.0      1.843584     0.839975      1.0
1.00
Mortgage              4520.0     51.789381    92.038931      0.0
0.00
Personal Loan         4520.0      0.000000     0.000000      0.0
0.00
Securities Account    4520.0      0.102212     0.302961      0.0
0.00
```

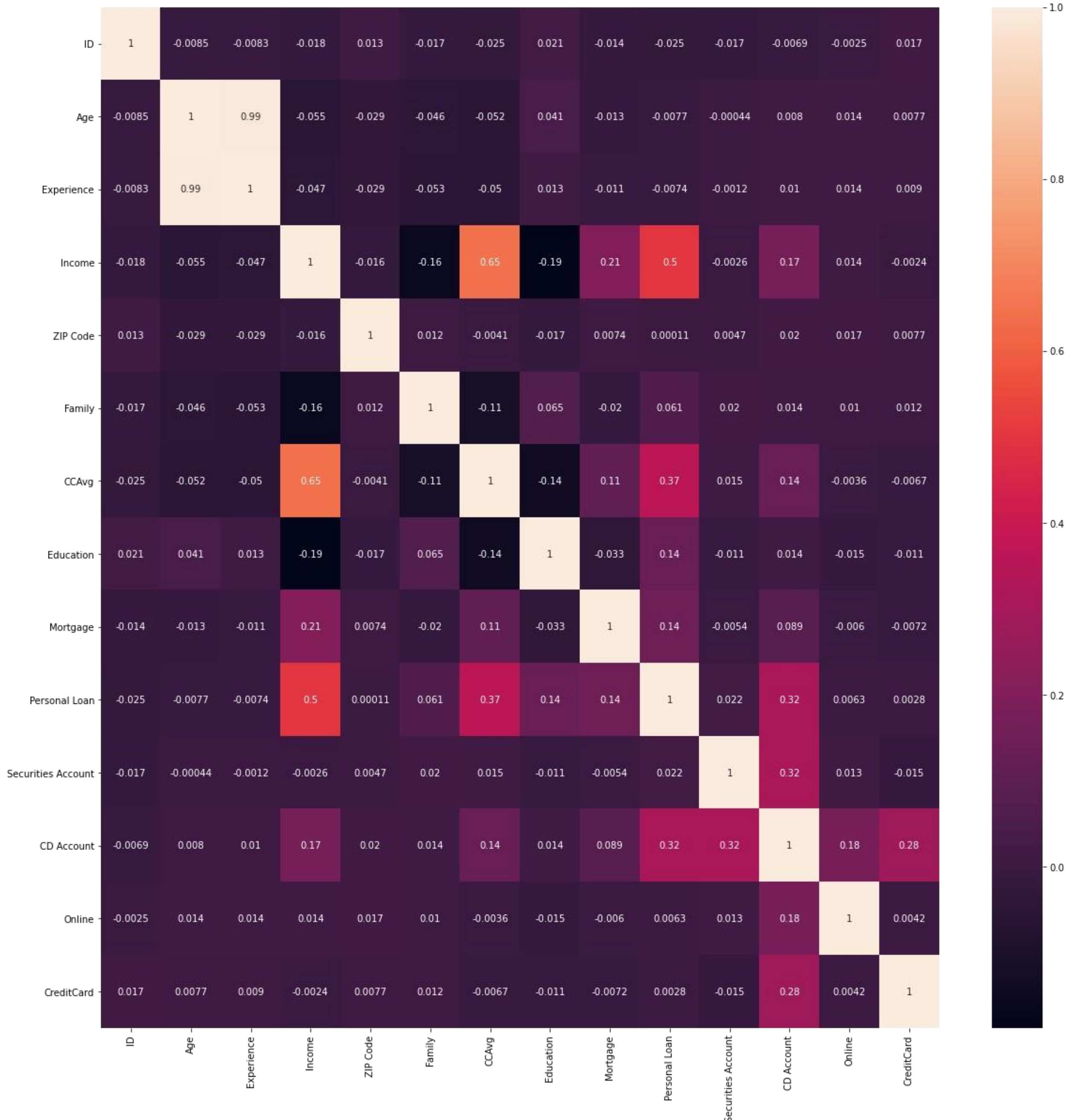
```
CD Account              4520.0      0.035841     0.185913     0.0
0.00
Online                  4520.0      0.595796     0.490792     0.0
0.00
CreditCard              4520.0      0.293584     0.455454     0.0
0.00

                            50%        75%       max
ID                       2518.5    3768.25    5000.0
Age                        45.0      55.00      67.0
Experience                 20.0      30.00      43.0
Income                     59.0      84.00     224.0
ZIP Code                93437.0   94608.00   96651.0
Family                      2.0       3.00       4.0
CCAvg                       1.4       2.30       8.8
Education                   2.0       3.00       3.0
Mortgage                    0.0      98.00     635.0
Personal Loan               0.0       0.00       0.0
Securities Account          0.0       0.00       1.0
CD Account                  0.0       0.00       1.0
Online                      1.0       1.00       1.0
CreditCard                  0.0       1.00       1.0
```

```
plt.figure(figsize=(15,8))
sns.distplot(personal_loans["Income"], label='Approved')
sns.distplot(no_personal_loans["Income"], label='Not Approved')
plt.legend()
plt.savefig('approved_not_approved.png', facecolor='w',
bbox_inches='tight')
plt.show()
```

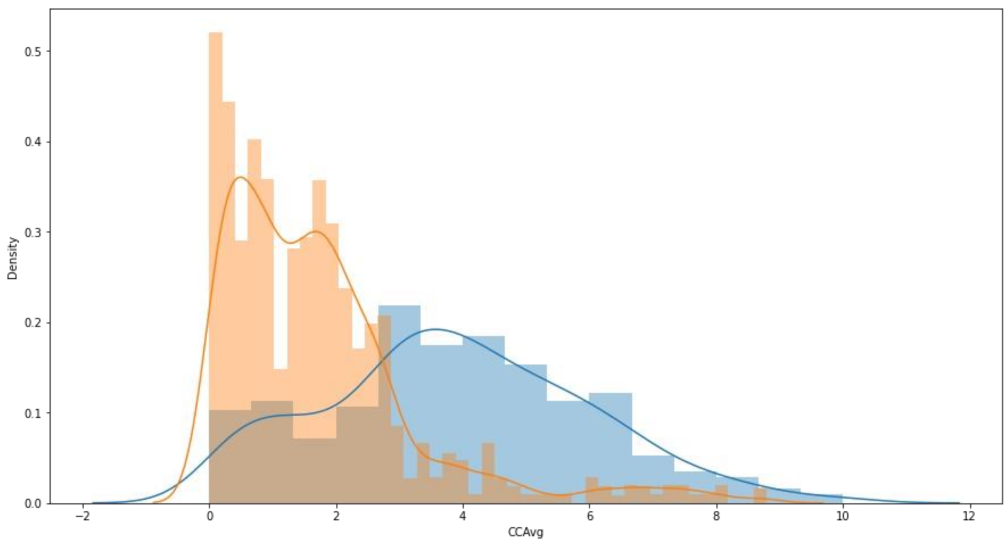



```
cm = bank_df.corr()
plt.figure(figsize=(20,20))
sns.heatmap(cm, annot=True)
plt.savefig('heatmap.png', facecolor='w', bbox_inches='tight')
plt.show()
```

```
plt.figure(figsize=(15,8))
sns.distplot(bank_df["CCAvg"])
plt.show()
```



```
plt.figure(figsize=(15,8))
sns.distplot(personal_loans["CCAvg"])
sns.distplot(no_personal_loans["CCAvg"])
plt.show()
```



Data Preparation

```
from tensorflow.keras.utils import to_categorical

X = bank_df.drop(columns=["Personal Loan"])
y = bank_df["Personal Loan"]

y = to_categorical(y)

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

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((4500, 13), (500, 13), (4500, 2), (500, 2))
```

Building a multi-layer neural network model

```
# sequential model
ann_model = keras.Sequential()

# adding dense layer
ann_model.add(Dense(250, input_dim=13, kernel_initializer='normal',
```

```
activation='relu'))
ann_model.add(Dropout(0.3))
ann_model.add(Dense(500, activation='relu'))
ann_model.add(Dropout(0.3))
ann_model.add(Dense(500, activation='relu'))
ann_model.add(Dropout(0.3))
ann_model.add(Dense(500, activation='relu'))
ann_model.add(Dropout(0.4))
ann_model.add(Dense(250, activation='linear'))
ann_model.add(Dropout(0.4))

# adding dense layer with softmax activation/output layer
ann_model.add(Dense(2, activation='softmax'))
ann_model.summary()
```

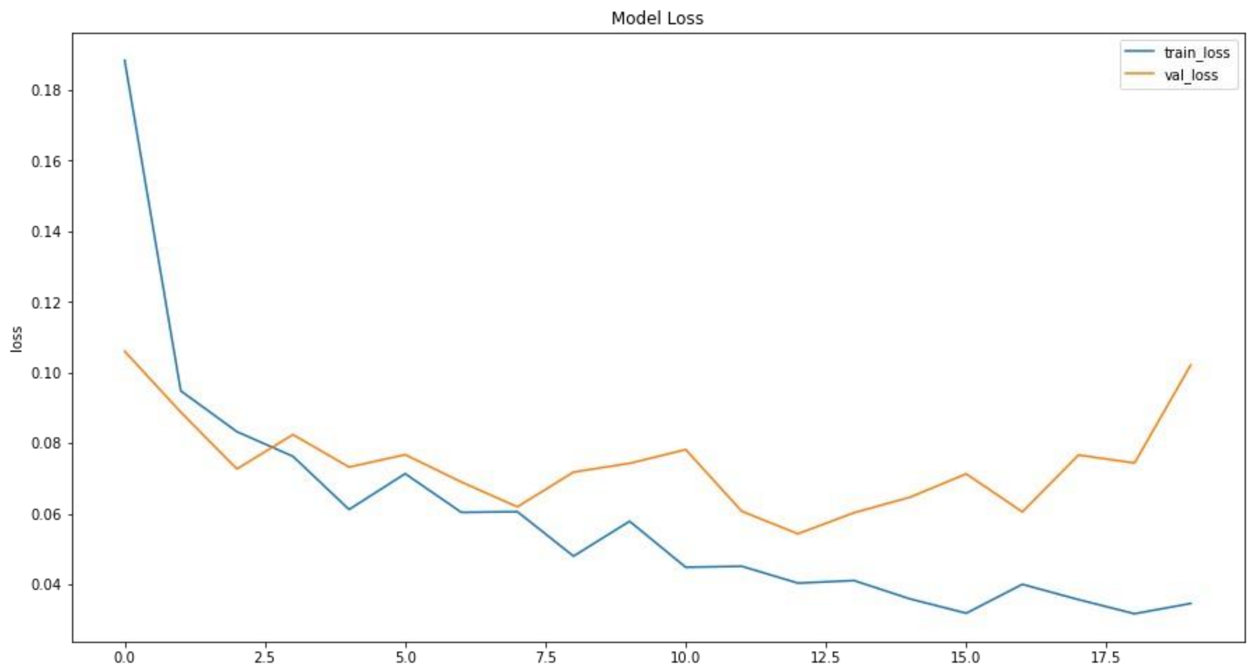
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 250)               3500
_________________________________________________________________
dropout_5 (Dropout)          (None, 250)               0
_________________________________________________________________
dense_7 (Dense)              (None, 500)               125500
_________________________________________________________________
dropout_6 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_8 (Dense)              (None, 500)               250500
_________________________________________________________________
dropout_7 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_9 (Dense)              (None, 500)               250500
_________________________________________________________________
dropout_8 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_10 (Dense)             (None, 250)               125250
_________________________________________________________________
dropout_9 (Dropout)          (None, 250)               0
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 502
=================================================================
Total params: 755,752
Trainable params: 755,752
Non-trainable params: 0
_________________________________________________________________
```

Compilation and training of deep learning model

```
from keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

ann_model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=[f1_m]) # metrics=['accuracy']

history = ann_model.fit(X_train, y_train, epochs=20,
validation_split=0.2, verbose=1)
```

```
Epoch 1/20
113/113 [==============================] - 3s 14ms/step - loss: 0.2799
- f1_m: 0.9006 - val_loss: 0.1060 - val_f1_m: 0.9537
Epoch 2/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0927
- f1_m: 0.9710 - val_loss: 0.0887 - val_f1_m: 0.9655
Epoch 3/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0805
- f1_m: 0.9745 - val_loss: 0.0727 - val_f1_m: 0.9688
Epoch 4/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0689
- f1_m: 0.9781 - val_loss: 0.0824 - val_f1_m: 0.9666
Epoch 5/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0637
- f1_m: 0.9758 - val_loss: 0.0732 - val_f1_m: 0.9677
Epoch 6/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0649
- f1_m: 0.9754 - val_loss: 0.0767 - val_f1_m: 0.9709
Epoch 7/20
113/113 [==============================] - 1s 10ms/step - loss: 0.0674
- f1_m: 0.9785 - val_loss: 0.0690 - val_f1_m: 0.9741
Epoch 8/20
```

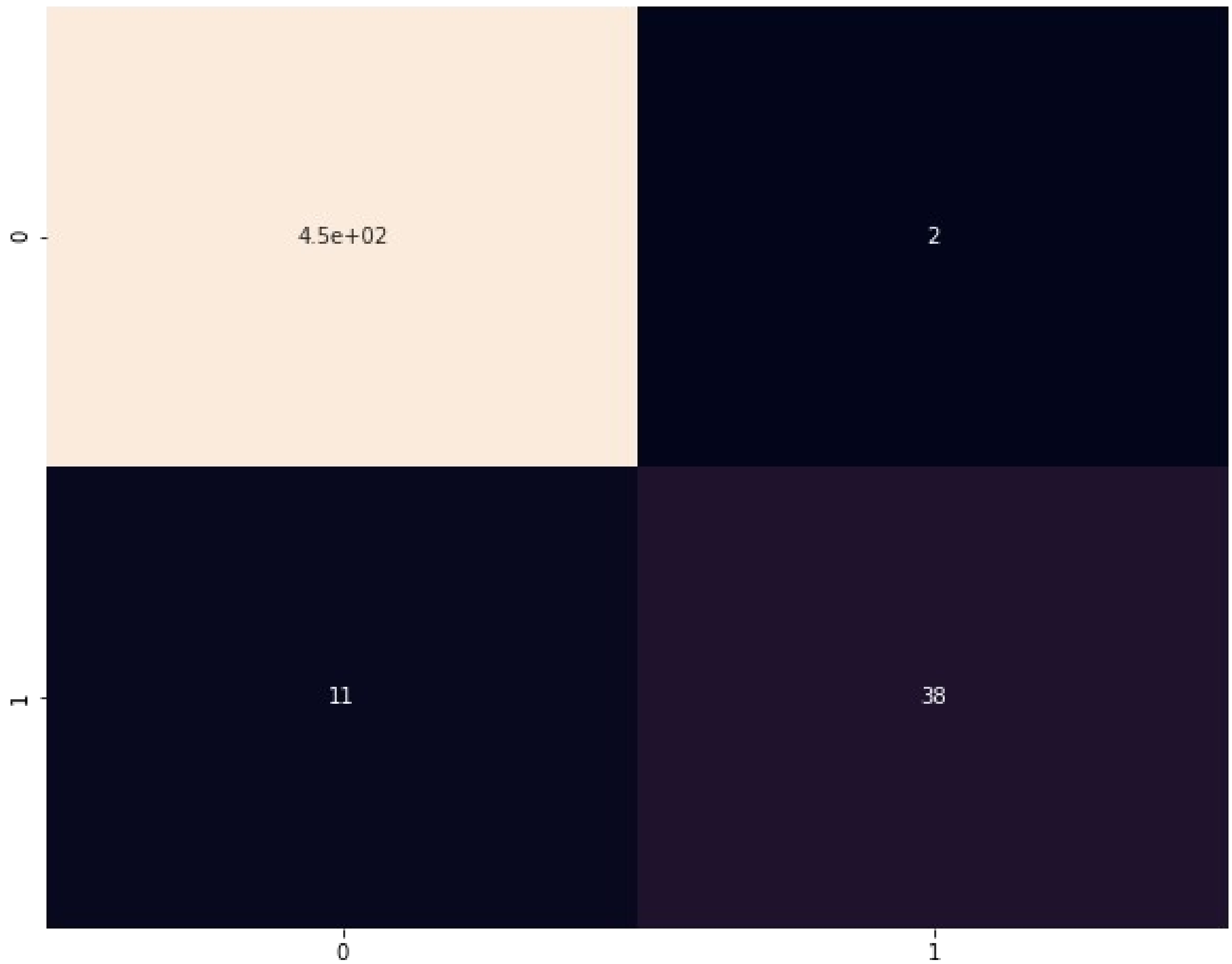
```
113/113 [==============================] - 1s 12ms/step - loss: 0.0535
- f1_m: 0.9840 - val_loss: 0.0619 - val_f1_m: 0.9784
Epoch 9/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0426
- f1_m: 0.9823 - val_loss: 0.0718 - val_f1_m: 0.9741
Epoch 10/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0615
- f1_m: 0.9804 - val_loss: 0.0742 - val_f1_m: 0.9698
Epoch 11/20
113/113 [==============================] - 1s 12ms/step - loss: 0.0487
- f1_m: 0.9811 - val_loss: 0.0781 - val_f1_m: 0.9752
Epoch 12/20
113/113 [==============================] - 1s 12ms/step - loss: 0.0436
- f1_m: 0.9818 - val_loss: 0.0606 - val_f1_m: 0.9795
Epoch 13/20
113/113 [==============================] - 1s 12ms/step - loss: 0.0374
- f1_m: 0.9905 - val_loss: 0.0543 - val_f1_m: 0.9828
Epoch 14/20
113/113 [==============================] - 1s 12ms/step - loss: 0.0395
- f1_m: 0.9845 - val_loss: 0.0602 - val_f1_m: 0.9784
Epoch 15/20
113/113 [==============================] - 1s 12ms/step - loss: 0.0392
- f1_m: 0.9835 - val_loss: 0.0647 - val_f1_m: 0.9752
Epoch 16/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0365
- f1_m: 0.9863 - val_loss: 0.0713 - val_f1_m: 0.9795
Epoch 17/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0328
- f1_m: 0.9895 - val_loss: 0.0605 - val_f1_m: 0.9752
Epoch 18/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0386
- f1_m: 0.9832 - val_loss: 0.0766 - val_f1_m: 0.9784
Epoch 19/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0340
- f1_m: 0.9901 - val_loss: 0.0743 - val_f1_m: 0.9774
Epoch 20/20
113/113 [==============================] - 1s 11ms/step - loss: 0.0383
- f1_m: 0.9851 - val_loss: 0.1021 - val_f1_m: 0.9731

# Plot the model performance across epochs
plt.figure(figsize=(15,8))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model  Loss')
plt.ylabel('loss')
plt.legend(['train_loss','val_loss'], loc = 'upper right')
plt.savefig('modelloss.png', facecolor='w', bbox_inches='tight')
plt.show()
```



Evaluating model performance

```
predictions = ann_model.predict(X_test)
predict = []

for i in predictions:
    predict.append(np.argmax(i))

from sklearn import metrics
y_test = np.argmax(y_test, axis=1)

f1_test = metrics.f1_score(y_test, predict)
prec = metrics.precision_score(y_test, predict)
rec = metrics.recall_score(y_test, predict)
acc = metrics.accuracy_score(y_test, predict)

print ("F1 Score: {:.4f}.".format(f1_test))
print ("Precision: {:.4f}.".format(prec))
print ("Recall: {:.4f}.".format(rec))
print ("Accuracy: {:.4f}.".format(acc)) # note this is not a good
measure of performance for this project as dataset is unbalanced.

F1 Score: 0.8539.
Precision: 0.9500.
Recall: 0.7755.
Accuracy: 0.9740.

conf_mat = metrics.confusion_matrix(y_test, predict)
plt.figure(figsize=(10,8))
sns.heatmap(conf_mat, annot=True, cbar=False)
```

```
plt.savefig('conf_matrix.png', facecolor='w', bbox_inches='tight')
plt.show()
```




```
print(metrics.classification_report(y_test, predict))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       451
           1       0.95      0.78      0.85        49

    accuracy                           0.97       500
   macro avg       0.96      0.89      0.92       500
weighted avg       0.97      0.97      0.97       500
```