Lending Club Loan Data Analysis Course-end Project 2 Description

Create a model that predicts whether or not a loan will be default using the historical data.

Problem Statement:

For companies like Lending Club correctly predicting whether or not a loan will be a default is very important. In this project, using the historical data from 2007 to 2015, you have to build a deep learning model to predict the chance of default for future loans. As you will see later this dataset is highly imbalanced and includes a lot of features that make this problem more challenging.

Domain: Finance

Analysis to be done: Perform data preprocessing and build a deep learning prediction model. Dataset columns and definition:

credit.policy: 1 if the customer meets the credit underwriting criteria of LendingClub.com, and 0 otherwise.

purpose: The purpose of the loan (takes values "credit_card", "debt_consolidation", "educational", "major_purchase", "small_business", and "all_other").

int.rate: The interest rate of the loan, as a proportion (a rate of 11% would be stored as 0.11). Borrowers judged by LendingClub.com to be more risky are assigned higher interest rates.

installment: The monthly installments owed by the borrower if the loan is funded.

log.annual.inc: The natural log of the self-reported annual income of the borrower.

dti: The debt-to-income ratio of the borrower (amount of debt divided by annual income).

fico: The FICO credit score of the borrower.

days.with.cr.line: The number of days the borrower has had a credit line.

revol.bal: The borrower's revolving balance (amount unpaid at the end of the credit card billing cycle).

revol.util: The borrower's revolving line utilization rate (the amount of the credit line used relative to total credit available).

inq.last.6mths: The borrower's number of inquiries by creditors in the last 6 months.

delinq.2yrs: The number of times the borrower had been 30+ days past due on a payment in the past 2 years.

pub.rec: The borrower's number of derogatory public records (bankruptcy filings, tx liens, or judgments).

Steps to perform:

Perform exploratory data analysis and feature engineering and then apply feature engineering. Follow up with a deep learning model to predict whether or not the loan will be default using the historical data.

Tasks:

1. `Feature Transformation`

Transform categorical values into numerical values (discrete)

2. `Exploratory data analysis of different factors of the dataset.`

3. `Additional Feature Engineering`

You will check the correlation between features and will drop those features which have a strong correlation

This will help reduce the number of features and will leave you with the most relevant features

4. `Modeling`

After applying EDA and feature engineering, you are now ready to build the predictive models

In this part, you will create a deep learning model using Keras with Tensorflow backend

To download the data sets click here

Content:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [20]:
```
import csv
# csv file name
df = pd.read_csv(r'D:\OneDrive\Knowledge Center\AI - ML\Masters in Artifical Eng
```

Basic Data Check

In [21]:
```
# Check the first few rows of the DataFrame
print(df.head())
```

```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407   
1              1         credit_card    0.1071       228.22       11.082143   
2              1  debt_consolidation    0.1357       366.86       10.373491   
3              1  debt_consolidation    0.1008       162.34       11.350407   
4              1         credit_card    0.1426       102.92       11.299732   

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0   
1  14.29   707        2760.000000      33623        76.7               0   
2  11.63   682        4710.000000       3511        25.6               1   
3   8.10   712        2699.958333      33667        73.2               1   
4  14.97   667        4066.000000       4740        39.5               0   

   delinq.2yrs  pub.rec  not.fully.paid  
0            0        0               0  
1            0        0               0  
2            0        0               0  
3            0        0               0  
4            1        0               0  
```

In [41]:
```
df.head()
```

Out[41]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revc
0	1	0.1189	829.10	11.350407	19.48	737.0	5639.958333	288
1	1	0.1071	228.22	11.082143	14.29	707.0	2760.000000	336
2	1	0.1357	366.86	10.373491	11.63	682.0	4710.000000	35
3	1	0.1008	162.34	11.350407	8.10	712.0	2699.958333	336
4	1	0.1426	102.92	11.299732	14.97	667.0	4066.000000	47

5 rows × 24 columns

In [22]:
```
# Check the summary statistics
print(df.describe())
```

```
       credit.policy     int.rate  installment  log.annual.inc          dti  \
count    9578.000000  9578.000000  9578.000000     9578.000000  9578.000000   
mean        0.804970     0.122640   319.089413       10.932117    12.606679   
std         0.396245     0.026847   207.071301        0.614813     6.883970   
min         0.000000     0.060000    15.670000        7.547502     0.000000   
25%         1.000000     0.103900   163.770000       10.558414     7.212500   
50%         1.000000     0.122100   268.950000       10.928884    12.665000   
75%         1.000000     0.140700   432.762500       11.291293    17.950000   
max         1.000000     0.216400   940.140000       14.528354    29.960000   

              fico  days.with.cr.line     revol.bal   revol.util  \
count  9578.000000        9578.000000  9.578000e+03  9578.000000   
mean    710.846314        4560.767197  1.691396e+04    46.799236   
std      37.970537        2496.930377  3.375619e+04    29.014417   
min     612.000000         178.958333  0.000000e+00     0.000000   
25%     682.000000        2820.000000  3.187000e+03    22.600000   
50%     707.000000        4139.958333  8.596000e+03    46.300000   
75%     737.000000        5730.000000  1.824950e+04    70.900000   
max     827.000000       17639.958330  1.207359e+06   119.000000   

       inq.last.6mths  delinq.2yrs      pub.rec  not.fully.paid  
count     9578.000000  9578.000000  9578.000000     9578.000000  
mean         1.577469     0.163708     0.062122        0.160054  
std          2.200245     0.546215     0.262126        0.366676  
min          0.000000     0.000000     0.000000        0.000000  
25%          0.000000     0.000000     0.000000        0.000000  
50%          1.000000     0.000000     0.000000        0.000000  
75%          2.000000     0.000000     0.000000        0.000000  
max         33.000000    13.000000     5.000000        1.000000  
```

In [23]:

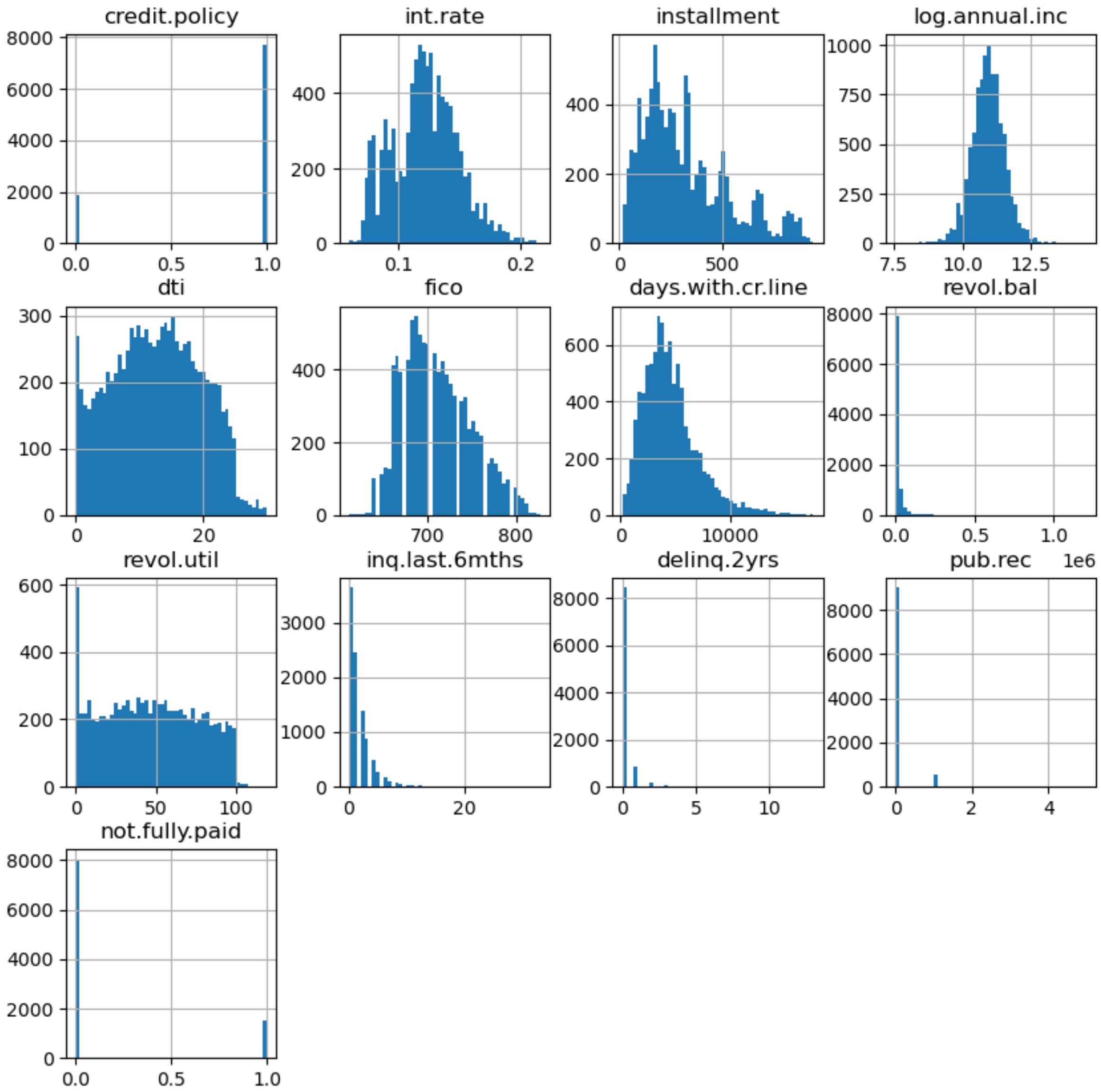
```python
# Check for missing values
print(df.isnull().sum())
```

```
credit.policy        0
purpose              0
int.rate             0
installment          0
log.annual.inc       0
dti                  0
fico                 0
days.with.cr.line    0
revol.bal            0
revol.util           0
inq.last.6mths       0
delinq.2yrs          0
pub.rec              0
not.fully.paid       0
dtype: int64
```

No data is missing, there there is no need to fill in with mean or modal data

In [24]:

```python
# Plot histograms for each variable
df.hist(figsize=(10, 10), bins=50)
plt.show()
```



In [25]:
```
#Checking for outliers
from scipy.stats import zscore

def detect_outliers(data):
    outliers = []
    threshold = 3
    mean = np.mean(data)
    std = np.std(data)

    for i in data:
        z_score = (i - mean) / std
        if np.abs(z_score) > threshold:
            outliers.append(i)
    return outliers
```

In [26]:
```
def remove_outliers(data):
    threshold = 3
    mean = np.mean(data)
    std = np.std(data)

    for i in data:
        z_score = (i - mean) / std
        if np.abs(z_score) > threshold:
```

```
            data = data[data != i]
    return data
```

function calculates the Z-score for each value in the data, and if the Z-score is greater than the specified threshold. The Z-score method of outlier detection uses a threshold, typically of 3 or -3, which corresponds to data points that are 3 standard deviations away from the mean. This is based on the empirical rule or the 68-95-99.7 rule, which states that nearly all data lies within 3 standard deviations of the mean in a normal distribution.Now printing them

In [27]:
```
for column in df.columns:
    if df[column].dtype in ['int64', 'float64']:
        outliers = detect_outliers(df[column])
        print(f'Outliers in {column}: {outliers}')
        df[column] = remove_outliers(df[column])
```

```
Outliers in credit.policy: []
Outliers in int.rate: [0.2086, 0.2086, 0.2121, 0.2121, 0.2086, 0.2121, 0.2052, 0.
2121, 0.209, 0.2052, 0.2086, 0.2086, 0.2121, 0.2052, 0.2121, 0.2121, 0.2086, 0.20
52, 0.209, 0.2164, 0.2164]
Outliers in installment: []
Outliers in log.annual.inc: [8.987196821, 13.08154138, 13.33100224, 8.517193191,
13.16542287, 12.8346813, 8.9751293, 13.25759333, 14.52835448, 9.047821442, 13.122
36338, 8.342839804, 12.87390202, 8.998136761, 8.612503371, 8.987196821, 8.4763711
97, 13.45883561, 13.30468493, 13.01700286, 8.494538501, 8.853665428, 8.987196821,
8.29404964, 8.987196821, 13.45883561, 13.5670492, 12.79385931, 13.08154138, 13.48
700649, 8.699514748, 14.12446477, 13.71015004, 8.476371197, 13.71015004, 8.699514
748, 14.18015367, 12.82395734, 13.47019937, 13.54370183, 13.00357984, 13.5670492,
13.99783211, 8.881836305, 13.36295384, 13.30468493, 12.85839783, 13.142166, 12.94
800999, 13.30468493, 8.9226583, 8.699514748, 8.779557456, 8.29404964, 8.29404964,
8.853665428, 13.12236338, 8.892886141, 7.547501683, 8.862483576, 7.60090246, 13.3
3100224, 8.517193191, 8.517193191, 9.071078305, 8.699514748, 8.987196821, 8.88183
6305, 8.9226583, 8.411832676, 8.160518247, 12.83201108, 8.699514748, 8.188689124,
13.45883561, 8.101677747, 8.9226583, 9.035986985, 13.01700286, 13.01700286, 8.672
999643, 13.30468493, 8.987196821, 12.94800999, 13.01700286, 12.86099861]
Outliers in dti: []
Outliers in fico: [827]
Outliers in days.with.cr.line: [14008.95833, 13349.95833, 16213.0, 12960.04167, 1
6259.04167, 12930.04167, 12780.0, 13109.0, 12554.04167, 13319.04167, 12668.04167,
12330.0, 14100.0, 13259.95833, 13349.95833, 12150.0, 12450.0, 12407.0, 13260.0, 1
4159.95833, 13620.0, 14580.0, 13770.0, 12433.0, 12153.04167, 14167.0, 13470.0, 15
150.04167, 12540.04167, 13380.04167, 14760.0, 15420.95833, 13080.0, 12391.0, 1368
1.0, 12330.0, 12390.0, 13770.0, 13410.0, 12539.95833, 13830.0, 12209.95833, 1508
9.95833, 13140.0, 14133.0, 14191.0, 12060.0, 13020.0, 12810.0, 13334.95833, 1226
6.95833, 13410.0, 12629.95833, 15419.95833, 13140.0, 15119.95833, 15360.0, 12120.
0, 13049.95833, 14009.95833, 14039.95833, 13500.0, 13111.0, 13379.95833, 12480.0,
14310.0, 13170.0, 13109.95833, 15299.95833, 12930.0, 12689.95833, 14400.0, 14879.
95833, 13950.0, 14580.0, 13110.0, 15692.0, 12690.0, 16652.0, 13020.04167, 14130.
0, 12690.0, 15360.04167, 16350.0, 12990.04167, 12990.04167, 13023.04167, 14580.0,
12450.0, 12600.04167, 12482.0, 12570.04167, 12150.0, 14640.0, 14580.0, 15990.0, 1
4923.0, 14761.04167, 12791.0, 13590.04167, 12061.0, 13200.0, 15030.0, 14100.0416
7, 13620.04167, 13170.0, 12692.04167, 13410.04167, 14671.0, 12360.0, 17616.0, 139
80.0, 12184.04167, 12752.04167, 12570.04167, 12450.0, 12150.04167, 12240.04167, 1
3979.04167, 13530.0, 13979.0, 13499.0, 17639.95833, 12990.0, 13740.0, 16260.0, 12
218.95833, 12450.0, 13170.0, 14550.0, 13920.0, 14009.95833, 14529.0, 15089.95833,
12599.95833, 13590.0, 12209.95833, 12150.0, 15271.0, 13170.04167, 12344.0, 12930.
04167]
Outliers in revol.bal: [128000, 148829, 141287, 119420, 120563, 120338, 126302, 1
41165, 127192, 132103, 129705, 149527, 144723, 145711, 121159, 144165, 124784, 12
9224, 121563, 126088, 139636, 120322, 143499, 150971, 152416, 150786, 168496, 150
786, 222702, 130799, 216959, 121719, 401941, 128850, 245886, 407794, 275925, 2759
25, 270800, 127093, 212629, 198023, 210887, 133963, 165108, 242194, 188854, 14854
0, 204954, 247970, 120830, 200583, 226567, 220710, 167006, 161129, 214559, 20534
7, 119451, 205347, 150334, 205016, 161337, 190486, 276911, 149621, 152189, 25500
1, 134025, 238903, 138492, 211463, 146579, 228727, 164974, 224090, 141532, 21287
8, 125642, 182274, 199390, 255805, 122464, 351453, 186072, 164861, 952013, 12640
2, 311616, 290291, 125743, 167381, 229400, 139544, 394107, 508961, 276570, 12135
2, 273853, 374487, 181638, 163522, 156752, 122620, 164324, 210913, 190153, 12071
8, 121874, 191303, 229908, 388892, 144217, 162716, 290341, 602519, 140420, 12418
5, 146743, 152002, 269726, 134842, 215175, 182893, 149822, 256757, 198058, 19005
0, 203525, 163772, 165867, 158000, 217827, 203460, 172842, 119702, 168334, 14196
7, 156798, 184938, 205956, 230420, 201583, 155946, 235868, 136960, 248480, 21925
0, 226936, 232755, 237551, 259128, 1207359, 211931, 206877, 197716, 338935, 38548
9, 215372]
Outliers in revol.util: []
Outliers in inq.last.6mths: [33, 9, 9, 18, 14, 9, 15, 13, 12, 10, 13, 9, 19, 10,
```

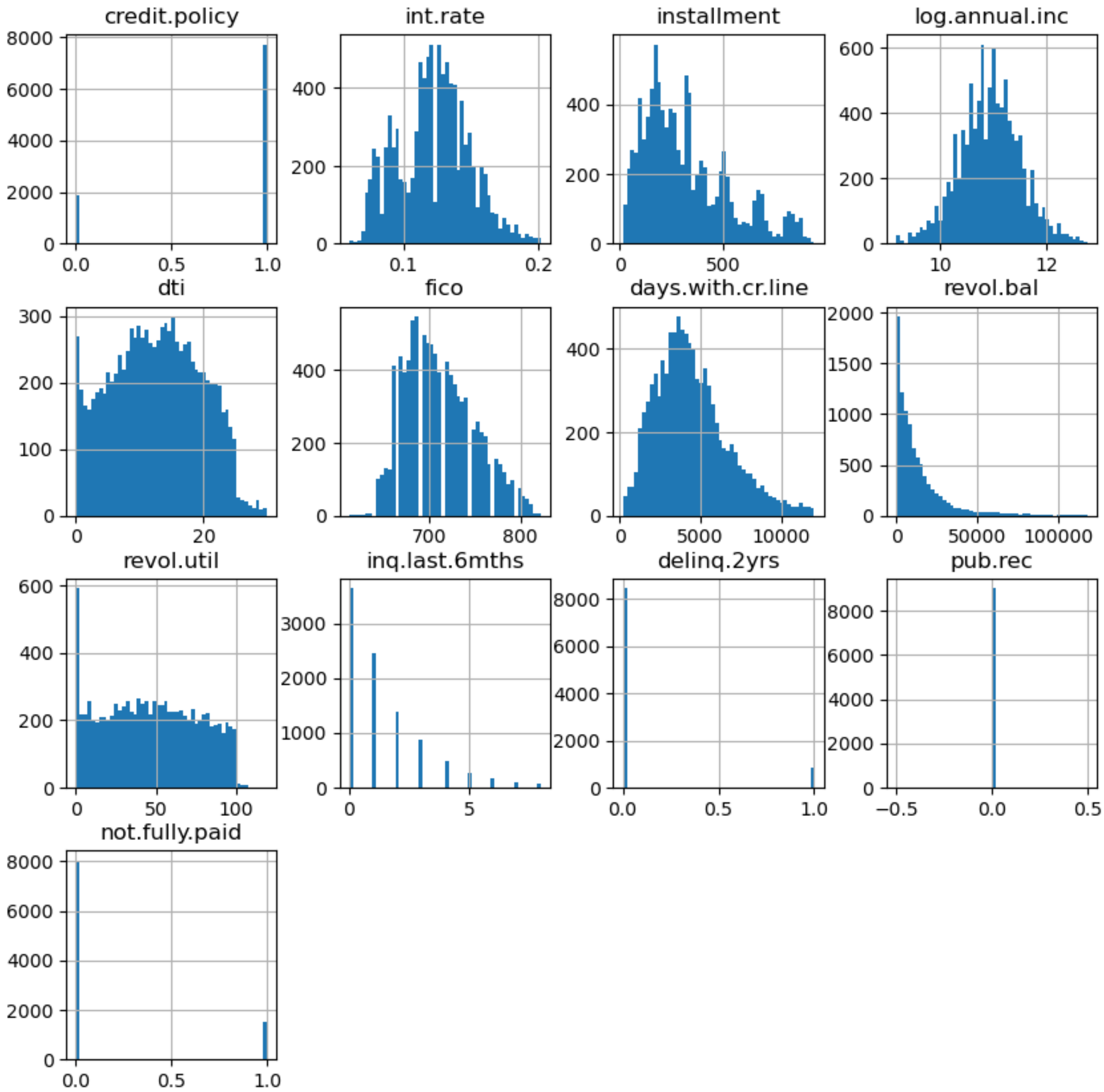
```
12, 9, 9, 12, 10, 10, 18, 12, 12, 10, 19, 11, 16, 15, 10, 12, 15, 11, 13, 20, 18,
9, 27, 25, 15, 28, 11, 31, 11, 14, 9, 24, 16, 15, 16, 14, 11, 18, 13, 9, 14, 13,
12, 10, 15, 11, 10, 9, 12, 12, 12, 13, 11, 11, 17, 12, 15, 24, 14, 9, 12, 32, 9,
15, 10, 9, 11, 9, 10, 10, 9, 17, 11, 12, 9, 14, 11, 11, 12, 15, 11, 10, 11, 12, 1
1, 9, 9, 9, 9, 9, 9, 9, 10, 9, 9, 9, 10, 9, 10, 10, 10, 9, 10, 9, 10, 9, 9, 9, 9,
9, 10, 10, 9, 9, 9, 9, 10, 9, 9, 9, 9, 9, 9, 9, 10, 9, 9]
Outliers in delinq.2yrs: [2, 2, 4, 2, 2, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 3, 2, 2, 4, 3, 5, 3, 2, 3, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 2, 2, 3, 2, 3, 2, 2,
3, 2, 3, 2, 2, 2, 2, 3, 2, 2, 2, 3, 2, 2, 2, 2, 6, 2, 2, 2, 3, 3, 2, 2, 2, 3, 3,
2, 3, 2, 3, 2, 2, 2, 4, 3, 2, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 3,
2, 2, 2, 2, 2, 2, 2, 3, 4, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 3, 3,
2, 3, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 3, 2, 4, 3, 2, 2, 3, 3, 2, 3, 5, 2, 2, 2, 3,
4, 3, 2, 2, 5, 2, 2, 3, 2, 4, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 4, 2,
4, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 4, 4, 13, 2, 2, 2, 2, 3, 5, 2, 2, 2, 2, 4, 2,
2, 3, 6, 2, 3, 3, 3, 2, 2, 2, 2, 2, 2, 7, 2, 2, 2, 2, 2, 2, 2, 8, 2, 4, 2, 2, 2,
4, 3, 3, 3, 2, 2, 2, 2, 3, 3, 2, 5, 11, 3, 2, 2, 2, 2, 2, 4, 3, 2, 3, 2, 2, 3, 2,
4, 2, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 3, 2, 3, 2, 2, 4, 3, 2, 3, 2]
Outliers in pub.rec: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 2, 2, 1, 1, 1, 3, 2, 1, 5, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Outliers in not.fully.paid: []
```

In [28]:

```python
# Check the summary statistics
print(df.describe())
```

```
       credit.policy     int.rate  installment  log.annual.inc          dti  \
count    9578.000000  9557.000000  9578.000000     9492.000000  9578.000000
mean        0.804970     0.122448   319.089413       10.932222    12.606679
std         0.396245     0.026562   207.071301        0.575705     6.883970
min         0.000000     0.060000    15.670000        9.169518     0.000000
25%         1.000000     0.103900   163.770000       10.571317     7.212500
50%         1.000000     0.122100   268.950000       10.929887    12.665000
75%         1.000000     0.139900   432.762500       11.289782    17.950000
max         1.000000     0.201700   940.140000       12.765700    29.960000

              fico  days.with.cr.line      revol.bal   revol.util  \
count  9577.000000        9436.000000    9419.000000  9578.000000
mean    710.834186        4424.895833   13726.008706    46.799236
std      37.953962        2249.876766   16775.258870    29.014417
min     612.000000         178.958333       0.000000     0.000000
25%     682.000000        2790.000000    3109.000000    22.600000
50%     707.000000        4109.020834    8372.000000    46.300000
75%     737.000000        5640.041667   17524.000000    70.900000
max     822.000000       12033.000000  117814.000000   119.000000

       inq.last.6mths  delinq.2yrs  pub.rec  not.fully.paid
count     9437.000000  9290.000000   9019.0     9578.000000
mean         1.417612     0.089559      0.0        0.160054
std          1.685511     0.285564      0.0        0.366676
min          0.000000     0.000000      0.0        0.000000
25%          0.000000     0.000000      0.0        0.000000
50%          1.000000     0.000000      0.0        0.000000
75%          2.000000     0.000000      0.0        0.000000
max          8.000000     1.000000      0.0        1.000000
```

In [29]:
```
# Plot histograms for each variable
df.hist(figsize=(10, 10), bins=50)
plt.show()
```



In [30]:
```
# Select columns containing categorical data
categorical_columns = df.select_dtypes(include=['object']).columns

print("Categorical columns in the DataFrame:")
for column in categorical_columns:
    print(column)

print("\nUnique values in each categorical column:")
for column in categorical_columns:
    print(f"{column}: {df[column].unique()}")
```

```
Categorical columns in the DataFrame:
purpose

Unique values in each categorical column:
purpose: ['debt_consolidation' 'credit_card' 'all_other' 'home_improvement'
 'small_business' 'major_purchase' 'educational']
```

We'll convert the categorical variables into dummy variables: Using one-hot encoding

In [31]:
```
df = pd.get_dummies(df, drop_first=True)
```

Feature engineering. feature engineering is an iterative and experimental process guided by trying out different ideas and checking if they improve your model's performance.

1. Interaction Features: You can create new features that are interactions of existing features. 'income_to_debt' which is the ratio of 'log.annual.inc' to 'dti'.

df['income_to_debt'] = df['log.annual.inc'] / df['dti'] 2. Polynomial Features: useful if the relationship between the feature and the target is non-linear. 3. Binning: You can also bin numerical variables to convert them into categorical variables. This is useful for variables like 'fico' where different ranges could have different default probabilities. 4. Feature Scaling: This includes algorithms that use a weighted sum of the input, like linear regression, and algorithms that use distance measures, like k-nearest neighbors. Using MinMaxScalar

In [34]:
```
#Interaction Features - income_to_debt_ratio
df['income_to_debt'] = df['log.annual.inc'] / df['dti']
#Polynomial Features: - fico
df['fico_squared'] = df['fico'] ** 2
#Binning
df['fico_range'] = pd.cut(df['fico'], bins=[0, 650, 700, 750, 800, 850], labels=
#Scalar
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['scaled_fico'] = scaler.fit_transform(df[['fico']])
```

In [37]:
```
#Have introduced fico_range which is categorical. Need one-hot encoding for that
df = pd.get_dummies(df, drop_first=True)
```

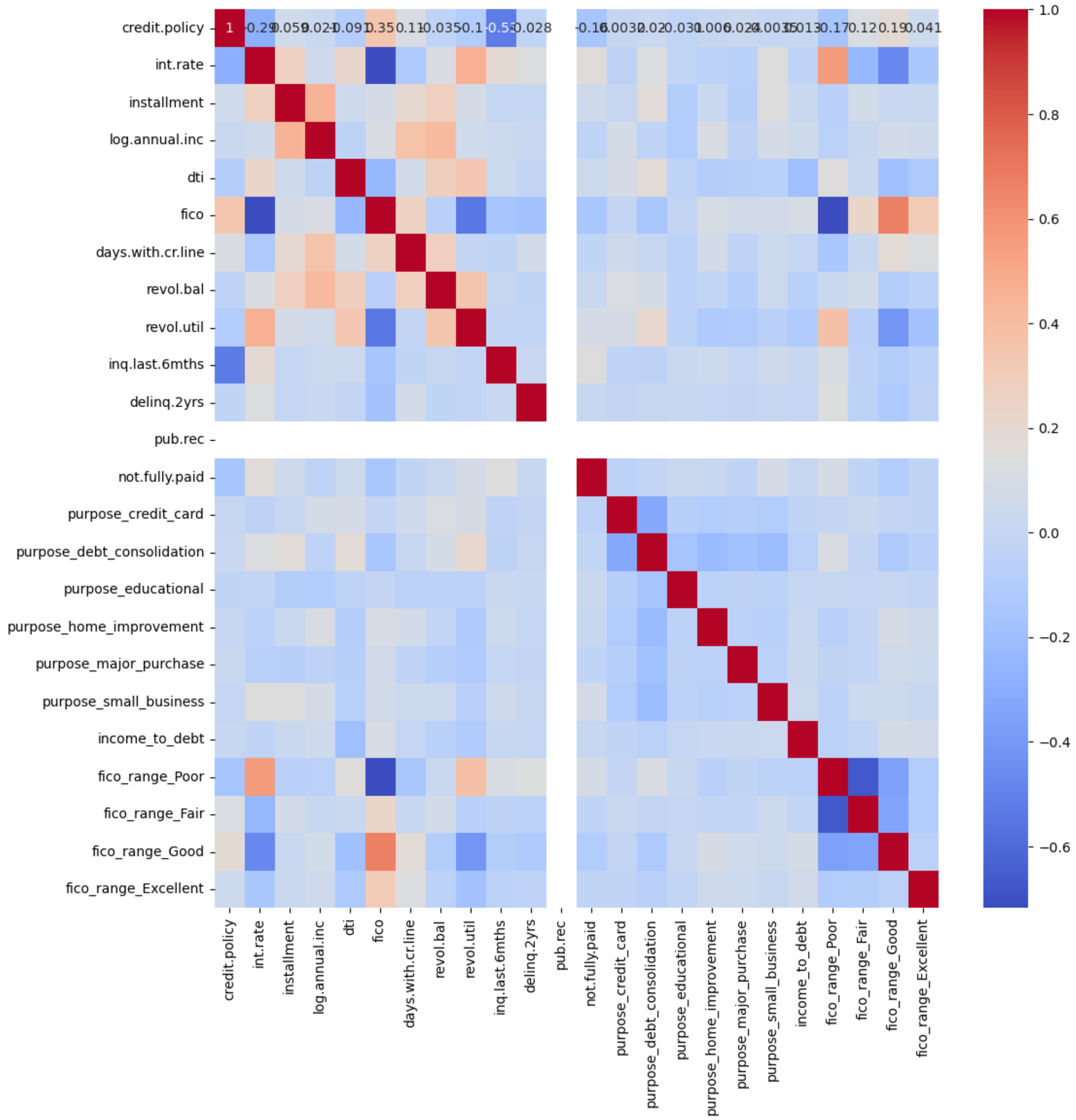
In [38]:
```
# Check the updated DataFrame
print(df.head())
```

```
   credit.policy  int.rate  installment  log.annual.inc    dti   fico  \
0              1    0.1189       829.10       11.350407  19.48  737.0
1              1    0.1071       228.22       11.082143  14.29  707.0
2              1    0.1357       366.86       10.373491  11.63  682.0
3              1    0.1008       162.34       11.350407   8.10  712.0
4              1    0.1426       102.92       11.299732  14.97  667.0

   days.with.cr.line  revol.bal  revol.util  inq.last.6mths  ...  \
0        5639.958333    28854.0        52.1             0.0  ...
1        2760.000000    33623.0        76.7             0.0  ...
2        4710.000000     3511.0        25.6             1.0  ...
3        2699.958333    33667.0        73.2             1.0  ...
4        4066.000000     4740.0        39.5             0.0  ...

   purpose_home_improvement  purpose_major_purchase  purpose_small_business  \
0                     False                   False                   False
1                     False                   False                   False
2                     False                   False                   False
3                     False                   False                   False
4                     False                   False                   False

   income_to_debt  fico_squared  scaled_fico  fico_range_Poor  \
0        0.582670      543169.0     0.689445            False
1        0.775517      499849.0    -0.101027            False
2        0.891960      465124.0    -0.759754             True
3        1.401285      506944.0     0.030718            False
4        0.754825      444889.0    -1.154991             True

   fico_range_Fair  fico_range_Good  fico_range_Excellent
0             True            False                 False
1             True            False                 False
2            False            False                 False
3             True            False                 False
4            False            False                 False

[5 rows x 26 columns]
```

In [45]:

```python
import seaborn as sns
# Additional Feature Engineering
# Correlation Matrix
corr_matrix = df.corr()

# Plotting the correlation matrix
plt.figure(figsize=(12, 12))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```

```
C:\anaconda3\Lib\site-packages\seaborn\matrix.py:260: FutureWarning: Format strin
gs passed to MaskedConstant are ignored, but in future may error or produce diffe
rent behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```



In [52]:
```
#Show higly correlated columns and remove them
columns = np.full((corr_matrix.shape[0],), True, dtype=bool)
for i in range(corr_matrix.shape[0]):
    for j in range(i+1, corr_matrix.shape[0]):
        print(corr_matrix.iloc[i,j])
        if corr_matrix.iloc[i,j] >= 0.9:
            if columns[j]:
                columns[j] = False
selected_columns = df.columns[columns]
df = df[selected_columns]
```

```
-0.29148462629776906
0.05876961631321294
0.02063948805862432
-0.09090056913279637
0.3483360957969314
0.10977458786199744
-0.03547714461092405
-0.10409494555108892
-0.52662328019139
-0.028298880951978054
nan
-0.1581191503140471
0.0032161339031254333
0.020193497516531767
-0.03134560437416619
0.00603586312466811
0.024280687188155115
-0.0035112537939735325
0.012883662127726155
-0.1666714947965311
0.11863810116249128
0.18539145041273264
0.04132450173931782
0.27163491047519356
0.05067666847620666
0.22318344148596397
-0.7177785360338564
-0.12447392044117564
0.11254492052694882
0.46753890910357926
0.1923960400176897
0.12851311677171215
nan
0.15827464753464426
-0.04081495761283574
0.12501166786558532
-0.018458282164694313
-0.05226932539561881
-0.06821729788196083
0.14595054374814567
-0.03145427620721436
0.5595751560779518
-0.23490668011704963
-0.4704961305217298
-0.1399250967649121
0.450905488771987
0.05020184110319647
0.08597240557969194
0.19664193603770116
0.27669163656484574
0.08135621749582012
-0.0003834495382613887
-0.0012320484658879022
nan
0.04995516207059644
0.0007738090709420617
0.1616580013104614
-0.09451004931538608
0.023024122449635278
-0.07983622886170423
```

```
0.1456540419788518
0.02342354147769268
-0.0648201223126771
0.07050360163582786
0.017197726119239223
0.023504856998974335
-0.04455281279541422
0.10971139807278624
0.36882315419937245
0.4137277965316285
0.057510168418178305
0.03188356741200561
0.013882209134881682
nan
-0.03355455628804915
0.07692956656716249
-0.03351807625033631
-0.10356262703784876
0.1140741198208717
-0.040740091382781736
0.08571890535506885
0.048881550330030304
-0.05874960581926279
0.012420974734598956
0.0683476968203624
0.05367712370181254
-0.24088409543781536
0.06898319751251
0.28243106967691994
0.33710917923817196
0.028763189837195523
-0.006971668489504266
nan
0.03736152382778221
0.08447616917617881
0.17914851809661236
-0.035325332054787405
-0.09278763943818578
-0.07771858283152618
-0.0692450732383655
-0.19836484935207374
0.14905541367074657
0.021081510732516615
-0.19301367757700247
-0.12647781705471406
0.25800448655473335
-0.06306602523945731
-0.5411489376080869
-0.1603913043494145
-0.17725207941055363
nan
-0.1496014645915545
-0.01239414089870829
-0.15394497797301424
-0.012956689886774219
0.09638830023307289
0.06723199405561829
0.06340744296211245
0.09766528965859293
-0.7156060582338638
```

```
0.23283075576885845
0.6672487014917651
0.3136360239452224
0.27854808944686116
-0.00731477399588132
-0.028267855571136338
0.07051643805175514
nan
-0.030697334241953846
0.05023944629623834
0.004196060173449497
-0.05477375620690278
0.0682838051369036
-0.03466310334291514
0.030690073583202325
-0.0036313348353803317
-0.14967139543632987
0.012035333937447433
0.18106754174296064
0.129482427398205
0.34933794983892874
0.003991269771214988
-0.02941853747154703
nan
0.023722658514239624
0.11654951222654487
0.07831336932887342
-0.050844805125235044
-0.013371078387681605
-0.08474244570098279
0.03283853168810074
-0.0704445498468904
0.020839617744617497
0.07313559423549146
-0.09754479400078324
-0.050499584491933536
-0.01817965705401736
-0.015819602721874545
nan
0.08208776761019922
0.09132072320088004
0.2118692277029554
-0.053127895393180385
-0.11444875757115762
-0.10807928211374968
-0.06096238825165564
-0.11059194013346865
0.3777286592569536
-0.06385648214868653
-0.4127843315024517
-0.17502115961784703
0.017315840608064986
nan
0.14604330281713063
-0.035918595400253724
-0.04531591125366048
0.024158553673322734
0.04362494841659997
-0.0055994847118961966
0.04754265942304494
```

```
0.008489134640709472
0.1036280716830586
-0.044033251437581904
-0.09398430031487223
-0.051872612840038135
nan
0.012636138632704973
-0.013046178632133672
0.0007694454015450834
0.012127196248153522
-0.006052607563084513
-0.010187218567051804
0.0018010486817167444
-0.0019171474185131756
0.13990130753552849
-0.0526001456601173
-0.11744074612467127
-0.03935441583686259
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
-0.04713600429131976
-0.017542540026761925
0.02160910503274935
0.00727188674272054
-0.02857966205519426
0.08445986378384562
0.0026213185497850795
0.09321480335285527
-0.032256720333767565
-0.10123875967086815
-0.030531591038181295
-0.326850135700532
-0.07507598644998942
-0.10327863890913194
-0.08517581565824141
-0.1023971792753939
-0.028710196795240023
-0.011723602128244744
0.024868011013565804
-0.011756104764095209
-0.02783576393452269
-0.1616980723626747
-0.22244072462466916
-0.18345100550934088
-0.22054224376040688
-0.05509400976649876
0.11251193905392373
-0.010992665043138438
-0.12403532795729366
-0.06184123584633742
```

```
-0.05109362060399515
-0.042137859830889364
-0.05065754820238676
-0.005079563747742851
0.0069672529428168725
-0.005815213884509939
0.0005956340776261271
-0.014575707510232374
-0.05796714789458013
-0.06968729784591507
0.01567812035585814
-0.06329176224884896
-0.01657107009800824
0.0885317929279189
0.05037289217353349
-0.05747241150483179
0.02316445922085912
-0.02541844990676977
-0.019764146221388598
0.05801058375241628
0.04288348172081158
0.02918782153877646
-0.0498407556801637
0.028554333818134745
0.03533290074937197
0.012889290227609873
-0.05407778466008145
-0.015628533919175798
0.07717523815780361
0.06396202324675586
-0.6684982990372754
-0.3677175191350446
-0.10439785106849261
-0.3442691678064869
-0.0977406825562104
-0.05376372888292391
```

In [ ]:
```
#As we see we do not have any highly correlated columns.
```

In [40]:
```
df.head()
```

Out[40]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revc
0	1	0.1189	829.10	11.350407	19.48	737.0	5639.958333	28
1	1	0.1071	228.22	11.082143	14.29	707.0	2760.000000	33
2	1	0.1357	366.86	10.373491	11.63	682.0	4710.000000	3
3	1	0.1008	162.34	11.350407	8.10	712.0	2699.958333	33
4	1	0.1426	102.92	11.299732	14.97	667.0	4066.000000	4

5 rows × 24 columns

In [62]:
```
# Check for infinity
if np.any(np.isinf(df)):
    print("DataFrame contains infinity. REmoving them")
```

```
    df.replace([np.inf, -np.inf], np.nan, inplace=True)

# Check for NaN
if df.isnull().values.any():
    print("DataFrame contains NaN values. . REmoving them")
    df.dropna(inplace=True)  # drop NaN values
```

In [63]:
```
#EDA is done. building the predictive models
#Now, let's split the data into a training set and a test set:
X = df.drop('credit.policy', axis=1)
y = df['credit.policy']
```

In [64]:
```
from sklearn.preprocessing import MinMaxScaler
# Create a MinMaxScaler object
scaler = MinMaxScaler()
# Fit the scaler to the features and transform
# Fit the scaler to the features and transform
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

In [69]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
```

In [70]:
```
# Check for infinity
if np.any(np.isinf(X_train)) or np.any(np.isinf(X_test)):
    print("DataFrame contains infinity")
    X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
    X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

# Check for NaN
if X_train.isnull().values.any() or X_test.isnull().values.any():
    print("DataFrame contains NaN values")
    X_train.dropna(inplace=True)  # drop NaN values
    X_test.dropna(inplace=True)  # drop NaN values
```

In [71]:
```
#build and train a deep learning model:
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']

history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=
```

```
Epoch 1/10
206/206 [==============================] - 2s 5ms/step - loss: 0.4561 - accuracy:
0.7952 - val_loss: 0.3334 - val_accuracy: 0.8544
Epoch 2/10
206/206 [==============================] - 1s 4ms/step - loss: 0.2857 - accuracy:
0.8859 - val_loss: 0.2361 - val_accuracy: 0.9238
Epoch 3/10
206/206 [==============================] - 1s 4ms/step - loss: 0.2417 - accuracy:
0.9145 - val_loss: 0.2186 - val_accuracy: 0.9250
Epoch 4/10
206/206 [==============================] - 1s 4ms/step - loss: 0.2321 - accuracy:
0.9185 - val_loss: 0.2148 - val_accuracy: 0.9250
Epoch 5/10
206/206 [==============================] - 1s 3ms/step - loss: 0.2256 - accuracy:
0.9183 - val_loss: 0.2087 - val_accuracy: 0.9281
Epoch 6/10
206/206 [==============================] - 0s 2ms/step - loss: 0.2195 - accuracy:
0.9202 - val_loss: 0.2047 - val_accuracy: 0.9299
Epoch 7/10
206/206 [==============================] - 1s 3ms/step - loss: 0.2127 - accuracy:
0.9215 - val_loss: 0.2030 - val_accuracy: 0.9257
Epoch 8/10
206/206 [==============================] - 0s 2ms/step - loss: 0.2050 - accuracy:
0.9220 - val_loss: 0.1982 - val_accuracy: 0.9317
Epoch 9/10
206/206 [==============================] - 1s 3ms/step - loss: 0.1987 - accuracy:
0.9241 - val_loss: 0.1967 - val_accuracy: 0.9305
Epoch 10/10
206/206 [==============================] - 1s 4ms/step - loss: 0.1913 - accuracy:
0.9270 - val_loss: 0.1900 - val_accuracy: 0.9348
```

In [74]:

```python
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Loss: {loss}, Accuracy: {accuracy}")
```

```
52/52 [==============================] - 0s 2ms/step - loss: 0.1900 - accuracy:
0.9348
Loss: 0.190029576420784, Accuracy: 0.9347958564758301
```

In [78]:

```python
# Get the predicted values
y_pred = model.predict(X_test)

# Since the model outputs probabilities, convert probabilities to class labels
y_pred = [1 if prob >= 0.5 else 0 for prob in y_pred]

# Create a DataFrame for comparison
comparison = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Calculate the difference
comparison['Difference'] = comparison['Actual'] - comparison['Predicted']

# Print the DataFrame
print(comparison)
```

```
52/52 [==============================] - 0s 1ms/step
      Actual  Predicted  Difference
1419       1          1           0
3904       1          1           0
9408       0          0           0
9364       0          0           0
1025       1          1           0
...      ...        ...         ...
9501       0          0           0
8618       0          0           0
1392       1          1           0
1617       1          0           1
6495       1          1           0

[1641 rows x 3 columns]
```

In [81]:

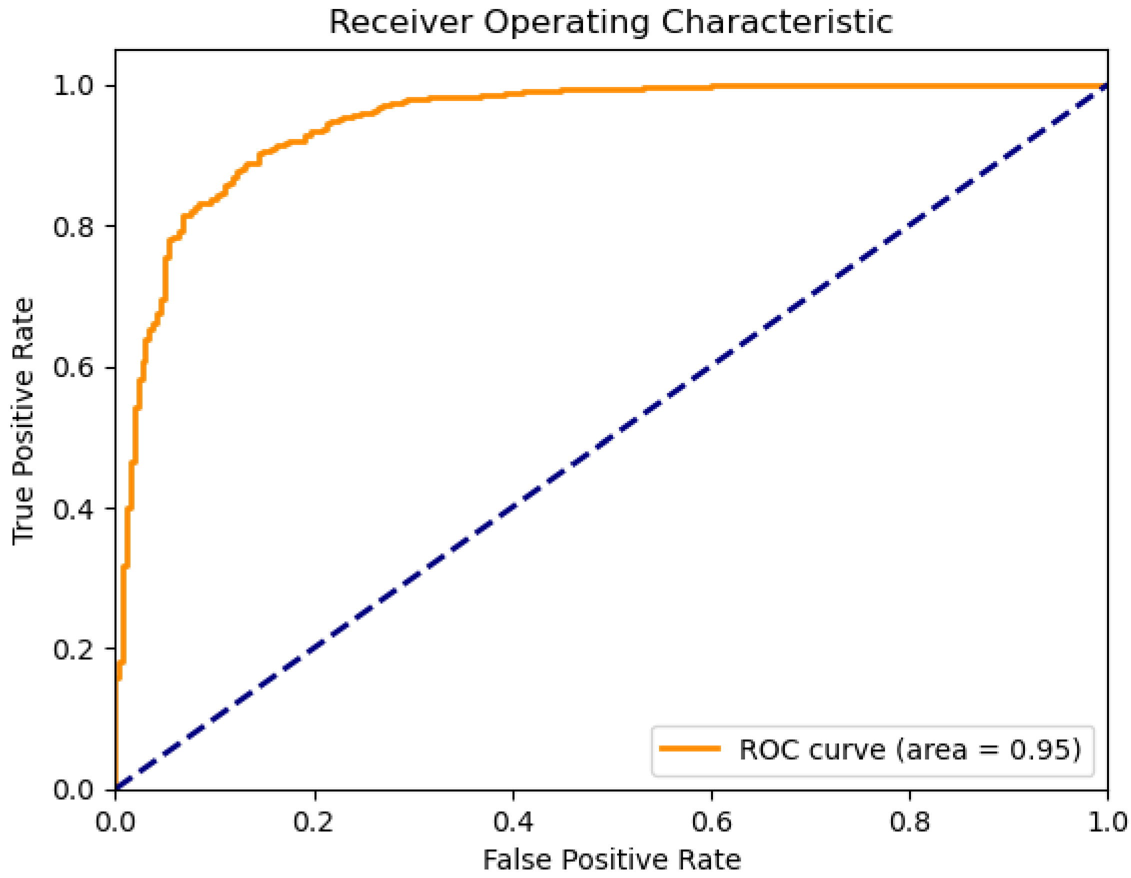
```
from sklearn.metrics import roc_curve, auc, recall_score
import matplotlib.pyplot as plt

# Get the predicted probabilities
y_pred_proba = model.predict(X_test)
```

```
52/52 [==============================] - 0s 1ms/step
```

In [83]:

```
# Calculate sensitivity/recall
y_pred = [1 if prob >= 0.5 else 0 for prob in y_pred]
sensitivity = recall_score(y_test, y_pred)
print(f'Sensitivity: {sensitivity}')

# Calculate ROC curve (fpr: false positive rate, tpr: true positive rate)
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

# Calculate AUC (Area Under Curve)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

```
Sensitivity: 0.978955007256894
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.95)
0.0
0.2
0.4
0.6
0.8
1.0

```
In [ ]:
```