Lending Club Loan Data Analysis - Neural Network

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
pd.set_option("display.max_columns",None)
```

In [4]:
```
ls
```

```
 Volume in drive C is OS
 Volume Serial Number is D890-AB6F

 Directory of C:\Users\Nirav Mahnot\Downloads

27-04-2023  13:05    <DIR>          .
27-04-2023  11:54    <DIR>          ..
27-04-2023  13:05    <DIR>          .ipynb_checkpoints
23-04-2023  17:33             7,531 .Rhistory
31-10-2022  12:16    <DIR>          16.20.00000.228_x64
24-04-2023  23:21            79,958 Assignment 2.xlsx
30-03-2023  19:41           790,331 BAB268-XLS-ENG (1).xlsx
13-03-2023  12:24           861,179 BAB268-XLS-ENG.xlsx
29-03-2023  12:39            73,012 baseball.csv
27-04-2023  13:05           918,679 Beauty_Lending_Club_loan_data_analysis_v
2.ipynb
12-04-2023  11:50         5,711,360 BigBasket IMB575-XLS-ENG.xls
11-04-2023  08:30           140,470 Case Discussions for Class Participation
Assignment 11-04-2023.pdf
11-04-2023  09:48             8,760 CD- Education OPSM326_NiravMahnot.docx
18-04-2023  12:29               280 class_table.csv
27-04-2023  12:59           936,329 Copy_of_Loan_Default.ipynb
16-04-2023  12:18            47,512 DeepLearning.ipynb
18-04-2023  12:29               370 department_table.csv
11-04-2023  12:58           130,891 EDM Case and Problem 11-04-2023 (1).pdf
11-04-2023  13:34           130,891 EDM Case and Problem 11-04-2023 (2).pdf
14-04-2023  02:07           130,891 EDM Case and Problem 11-04-2023 (3).pdf
18-04-2023  09:13           130,891 EDM Case and Problem 11-04-2023 (4).pdf
11-04-2023  12:58           130,891 EDM Case and Problem 11-04-2023.pdf
18-04-2023  12:29               278 employee_table.csv
24-04-2023  23:21           150,414 Enterprise Data Model Creation Problem-1
31-03-2023.xlsx
18-04-2023  12:29            14,837 exam_table.csv
11-04-2023  19:39            14,854 Example 3 (1).xlsx
24-04-2023  23:20           170,346 Excel Business Enterprise Data Management
Problem 17-03-2023.xlsx
26-04-2023  15:30       532,100,536 final_loan_data_analysis.csv
30-01-2023  12:14           878,670 IMB447-XLS-ENG.xlsx
22-04-2023  19:04        11,127,957 Introduction to Probability Models ( PDFD
rive ).pdf
10-02-2023  18:24            57,092 iris.ipynb
26-04-2023  15:26     1,675,133,810 loan_data_analysis.csv
23-04-2023  15:22           151,976 Loan_Default.ipynb
16-04-2023  10:17            12,089 logisticdrawback.csv
23-04-2023  12:19            37,509 MATH203-Introduction to Probability and S
tatistics.docx
23-04-2023  12:19           180,513 MATH333-Numerical Methods I.docx
29-11-2022  15:42       217,400,627 MIMU why WHYYYYYYYY - Made with Clipcham
p.mp4
23-04-2023  16:03           117,577 MONSTER GGCORR
16-04-2023  12:21            35,519 Nirav_Mahnot_quiz2.ipynb
06-12-2022  12:41    <DIR>          NiravM_SimplexProj
10-02-2023  19:50             3,416 Niravmahnot.ipynb
21-04-2023  21:21         2,336,770 NiravMahnot_200438 - Nirav Mahnot.pdf
10-02-2023  19:56             3,568 niravmahnot_200438.html
11-04-2023  21:49         2,470,218 OR_7.pptx
20-04-2023  20:58           171,034 question 1.pdf
```

```
12-04-2023  01:56            13,713 Research methods (1).xlsx
12-04-2023  00:19            12,130 Research methods.xlsx
18-02-2023  18:27    <DIR>          Resident Evil 2 [FitGirl Repack]
18-04-2023  08:53           225,037 Screenshot_20230418-085040.png
22-04-2023  19:04         2,865,004 Simulation, Fifth Edition ( PDFDrive ).pd
f
18-04-2023  12:29               531 student_table.csv
18-04-2023  12:29               338 subject_table.csv
03-03-2023  14:41       515,434,800 TableauPublicDesktop-64bit-2022-4-1.exe
11-04-2023  12:57            11,921 Teacher Performance.twbx
11-04-2023  09:56            40,712 Teacher_performance.xlsx
17-04-2023  17:53    <DIR>          Telegram Desktop
10-02-2023  19:50             3,416 test (2).ipynb
21-04-2023  21:14        12,357,698 Test 1 Applied Probability 2022-23.docx
23-04-2023  11:23         1,013,826 Test 1 Solutions (1).pdf
21-04-2023  21:21         1,013,826 Test 1 Solutions.pdf
16-04-2023  12:16           293,107 Test 2 - Deep Learning.docx
23-03-2023  17:17           475,792 Test 2 Paper.pdf
19-04-2023  14:01            43,813 Test 3 Applied Probability and Simulatio
n.docx
20-04-2023  20:51           107,369 Test 3_NiravMahnot.pdf
10-02-2023  19:24             2,627 test.ipynb
12-04-2023  00:30            14,918 Transportation - North west.docx
12-04-2023  01:57            14,963 transportation model.xlsx
11-04-2023  21:49            18,971 Transportation Model_using_simplex_via_ex
cel_solver (2).xlsx
16-04-2023  10:17            38,914 Untitled document (4).pdf
22-11-2022  10:36    <DIR>          Wallpaper
22-01-2023  14:14    <DIR>          wedding pics
25-01-2023  11:55       230,202,576 xlstat.exe
              63 File(s)  3,217,005,838 bytes
               9 Dir(s)  170,328,338,432 bytes free
```

In [5]:
```
df = pd.read_csv("C:/Users/Nirav Mahnot/Downloads/final_loan_data_analysis.csv
```

In [6]:
```
# Checking nummber of missing values in each feature
for feature in df.columns:
    print("Feature: ",feature,"-",df[feature].isnull().sum())
    print("====================================")
```

```
====================================
Feature:  hardship_last_payment_amount - 1042463
====================================
Feature:  disbursement_method - 12
====================================
Feature:  debt_settlement_flag - 12
====================================
Feature:  debt_settlement_flag_date - 1029866
====================================
Feature:  settlement_status - 1029866
====================================
Feature:  settlement_date - 1029866
====================================
Feature:  settlement_amount - 1029866
====================================
Feature:  settlement_percentage - 1029866
====================================
Feature:  settlement_term - 1029866
====================================
```

In [7]:
```
# Percent of missing values in each features
for feature in df.columns:
    print("Feature: ",feature,"-",np.round(df[feature].isnull().mean()*100,3))
    print("====================================")
```

```
Feature:  loan_amnt - 0.001
====================================
Feature:  funded_amnt - 0.001
====================================
Feature:  funded_amnt_inv - 0.001
====================================
Feature:  term - 0.001
====================================
Feature:  int_rate - 0.001
====================================
Feature:  installment - 0.001
====================================
Feature:  grade - 0.001
====================================
Feature:  sub_grade - 0.001
====================================
Feature:  emp_title - 7.009
====================================
Feature:  emp_length - 6.672
```

In [8]:
```
# Count of number of features having NaN values >=25%
count=0
for feature in df.columns:
    if df[feature].isnull().mean() >= 0.25 :
        count = count + 1

print("The number of features having missing value greater than 25% is: ",cour
```

```
The number of features having missing value greater than 25% is:  56
```

In [9]:
```
# Dropping the 58 columns having more than 25% missing values

for feature in df.columns:
    if df[feature].isnull().mean() >= 0.25 :
        df.drop(columns=feature,inplace=True)

df.head()
```

Out[9]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	
0	3600.0	3600.0	3600.0	36 months	13.99	123.03	C	C4	
1	24700.0	24700.0	24700.0	36 months	11.99	820.28	C	C1	
2	20000.0	20000.0	20000.0	60 months	10.78	432.66	B	B4	t
3	35000.0	35000.0	35000.0	60 months	14.85	829.90	C	C5	I
4	10400.0	10400.0	10400.0	60 months	22.45	289.91	F	F1	

In [10]:
```
df.shape
```

Out[10]:
```
(1048575, 91)
```

Categorical Features

In [11]:
```
categorical_features = [feature for feature in df.columns if df[feature].dtype
print("Number of categorical features are: ",len(categorical_features))
categorical_features
```

```
Number of categorical features are:  22
```

Out[11]:
```
['term',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'verification_status',
 'issue_d',
 'loan_status',
 'pymnt_plan',
 'purpose',
 'title',
 'zip_code',
 'addr_state',
 'earliest_cr_line',
 'initial_list_status',
 'last_pymnt_d',
 'last_credit_pull_d',
 'application_type',
 'hardship_flag',
 'disbursement_method',
 'debt_settlement_flag']
```

In [12]:
```
# Checking the number of unique values in each categorical column

for feature in categorical_features:
    print(feature)
    print(len(df[feature].unique()),"-->",df[feature].unique())
    print("===================================")
```

```
term
3 --> [' 36 months' ' 60 months' nan]
===================================
grade
8 --> ['C' 'B' 'F' 'A' 'E' 'D' 'G' nan]
===================================
sub_grade
36 --> ['C4' 'C1' 'B4' 'C5' 'F1' 'C3' 'B2' 'B1' 'A2' 'B5' 'C2' 'E2' 'A4'
'E3'
 'A1' 'D4' 'F3' 'D1' 'B3' 'E4' 'D3' 'D2' 'D5' 'A5' 'F2' 'E1' 'F5' 'E5'
 'A3' 'G2' 'G1' 'G3' 'G4' 'F4' 'G5' nan]
===================================
emp_title
237271 --> ['leadman' 'Engineer' 'truck driver' ... 'Office Culture Manage
r '
 'SR CLAIMS ADJUSTER' 'Whose manager']
===================================
emp_length
12 --> ['10+ years' '3 years' '4 years' '6 years' '1 year' '7 years' '8 ye
   '
```

In [13]:
```
# Checking which categorical feature has more than 100 categories
# Keeping only those categorical columns for which the unique categories are <

for col in categorical_features:
    if (len(df[col].unique())) > 100 :
        print(col)
        print(len(df[col].unique()))
        print(df[col].unique())
        print("==================================")
```

```
 'Jan-56' 'Jun-62' 'May-65' 'Jun-66' 'Jun-55' 'May-64' 'Jan-70' 'Feb-67'
 'Sep-67' 'Aug-66' 'Feb-64' 'Dec-67' 'Apr-71' 'May-68' 'Jul-67' 'Jul-66'
 'Dec-69' 'Jan-66' 'Sep-68' 'Jun-63' 'Oct-65' 'Oct-63' 'Nov-63' 'Apr-63'
 'Feb-66' 'Sep-53' 'Apr-55' 'Aug-59' 'May-63' 'Jun-58' 'Nov-56' 'Jul-59'
 'Jan-55' 'Mar-66' 'Mar-58' 'Jan-58' 'Dec-59' 'Apr-62' 'Apr-60' 'Nov-57'
 'Jun-64' 'May-60' 'Nov-62' 'Oct-67' 'Aug-60' 'Jul-62' 'May-55' 'Mar-60'
 'Nov-64' 'Aug-51' 'Aug-65' 'Sep-59' 'Nov-59' 'Apr-64' 'Mar-64' 'Jun-57'
 'Jan-53' 'Mar-65' 'Nov-58' 'Sep-56' 'May-57' 'Mar-61' 'May-62' 'Aug-61'
 'Dec-56' 'Jul-64' 'Dec-60' 'Aug-63' 'Jan-52' 'Mar-55' 'Jul-58' 'May-59'
 'Sep-60' 'Oct-61' 'Oct-60' 'Jul-51' 'Aug-50' 'Aug-55' 'Jun-59' 'Apr-58'
 'Nov-61' 'Jan-54' 'Jan-51' 'Jan-44' nan 'Nov-13' 'Aug-14' 'Jan-15'
 'Jun-13' 'Feb-14' 'Apr-13' 'Mar-14' 'Jul-13' 'Sep-14' 'Feb-15' 'Dec-14'
 'Dec-12' 'Dec-13' 'Jul-14' 'Aug-13' 'May-13' 'Apr-14' 'Sep-13' 'Feb-13'
 'Jan-13' 'Oct-14' 'May-14' 'Nov-14' 'Oct-13' 'Mar-13' 'Jan-14' 'Jun-14'
 'Jul-61' 'Jun-56' 'Jun-61' 'Feb-58' 'Jun-60' 'Apr-61' 'Nov-52' 'Aug-58'
 'Aug-53' 'Feb-63' 'Apr-34' 'Jul-60' 'Feb-62' 'Mar-33' 'Mar-54' 'Jun-52'
 'Mar-59' 'Apr-15' 'Jul-15' 'May-15' 'Aug-15' 'Mar-15' 'Jun-15' 'Dec-47'
 'Apr-57' 'Apr-59' 'Sep-51' 'Feb-34' 'Aug-57' 'May-58']
==================================
```

In [15]:
```
# Dropping all columns having unique values >100 --> emp_title, title, zip_coa

not_required_columns = ["emp_title","title","zip_code"]
df.drop(columns = not_required_columns,axis=1,inplace=True)
print("Shape after removing unnecessary columns: ",df.shape)
```

```
Shape after removing unnecessary columns:  (1048575, 88)
```

In [17]:
```
# Also removing them from categorical_features list

categorical_features.remove("emp_title")
categorical_features.remove("title")
categorical_features.remove("zip_code")
categorical_features
```

Out[17]:
```
['term',
 'grade',
 'sub_grade',
 'emp_length',
 'home_ownership',
 'verification_status',
 'issue_d',
 'loan_status',
 'pymnt_plan',
 'purpose',
 'addr_state',
 'earliest_cr_line',
 'initial_list_status',
 'last_pymnt_d',
 'last_credit_pull_d',
 'application_type',
 'hardship_flag',
 'disbursement_method',
 'debt_settlement_flag']
```

In [18]:
```
# Lets check if there are columns having only 1 unique value

for col in df.columns:
    if (len(df[col].unique())) == 1 :
        print(col)
        print(len(df[col].unique()))
        print(df[col].unique())
        print("===================================")
```

In [19]:
```
# Handling the features containing dates
date_features = ['issue_d','earliest_cr_line','last_pymnt_d','last_credit_pull
```

In [20]:
```
# Creating new features out of date features, containing month and year separa

df[['issue_d_month','issue_d_year']] = df['issue_d'].str.split("-",expand=True
df[['earliest_cr_line_month','earliest_cr_line_year']] = df['earliest_cr_line'
df[['last_pymnt_d_month','last_pymnt_d_year']] = df['last_pymnt_d'].str.split(
df[['last_credit_pull_d_month','last_credit_pull_d_year']] = df['last_credit_p
```

In [21]:
```
# Dropping the date columns
date_features = ['issue_d','earliest_cr_line','last_pymnt_d','last_credit_pull
df.drop(date_features,axis=1,inplace=True)
df.head()
```

Out[21]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	e
0	3600.0	3600.0	3600.0	36 months	13.99	123.03	C	C4	
1	24700.0	24700.0	24700.0	36 months	11.99	820.28	C	C1	
2	20000.0	20000.0	20000.0	60 months	10.78	432.66	B	B4	
3	35000.0	35000.0	35000.0	60 months	14.85	829.90	C	C5	
4	10400.0	10400.0	10400.0	60 months	22.45	289.91	F	F1	

In [22]:
```
# Count Plots
categorical_features
```

Out[22]:
```
['term',
 'grade',
 'sub_grade',
 'emp_length',
 'home_ownership',
 'verification_status',
 'issue_d',
 'loan_status',
 'pymnt_plan',
 'purpose',
 'addr_state',
 'earliest_cr_line',
 'initial_list_status',
 'last_pymnt_d',
 'last_credit_pull_d',
 'application_type',
 'hardship_flag',
 'disbursement_method',
 'debt_settlement_flag']
```

Numerical Features

In [23]:
```
numerical_features = [feature for feature in df.columns if df[feature].dtypes!
print("Number of numerical features are: ",len(numerical_features))
numerical_features
```

```
Number of numerical features are:  69
```

Out[23]:
```
['loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
 'int_rate',
 'installment',
 'annual_inc',
 'dti',
 'delinq_2yrs',
 'fico_range_low',
 'fico_range_high',
 'inq_last_6mths',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'out_prncp',
 'out_prncp_inv',
 'total_pymnt',
 'total_pymnt_inv',
 'total_rec_prncp',
 'total_rec_int',
 'total_rec_late_fee',
 'recoveries',
 'collection_recovery_fee',
 'last_pymnt_amnt',
 'last_fico_range_high',
 'last_fico_range_low',
 'collections_12_mths_ex_med',
 'policy_code',
 'acc_now_delinq',
 'tot_coll_amt',
 'tot_cur_bal',
 'total_rev_hi_lim',
 'acc_open_past_24mths',
 'avg_cur_bal',
 'bc_open_to_buy',
 'bc_util',
 'chargeoff_within_12_mths',
 'delinq_amnt',
 'mo_sin_old_il_acct',
 'mo_sin_old_rev_tl_op',
 'mo_sin_rcnt_rev_tl_op',
 'mo_sin_rcnt_tl',
 'mort_acc',
 'mths_since_recent_bc',
 'mths_since_recent_inq',
 'num_accts_ever_120_pd',
 'num_actv_bc_tl',
 'num_actv_rev_tl',
 'num_bc_sats',
 'num_bc_tl',
 'num_il_tl',
 'num_op_rev_tl',
 'num_rev_accts',
 'num_rev_tl_bal_gt_0',
 'num_sats',
```

```
'num_tl_120dpd_2m',
'num_tl_30dpd',
'num_tl_90g_dpd_24m',
'num_tl_op_past_12m',
'pct_tl_nvr_dlq',
'percent_bc_gt_75',
'pub_rec_bankruptcies',
'tax_liens',
'tot_hi_cred_lim',
'total_bal_ex_mort',
'total_bc_limit',
'total_il_high_credit_limit']
```

Creating new data having only 2 categories from previous data "Fully Charged" & "Charged Off"

In [24]:
```
# Keeping rows with loan_status category either "Fully Paid" or "Charged Off"

loan = df[(df["loan_status"] == "Fully Paid") |(df["loan_status"] == "Charged
loan.head(10)
```

Out[24]:

_plan	purpose	addr_state	dti	delinq_2yrs	fico_range_low	fico_range_high	inq_last_
n	debt_consolidation	PA	5.91	0.0	675.0	679.0	
n	small_business	SD	16.06	1.0	715.0	719.0	
n	home_improvement	IL	10.78	0.0	695.0	699.0	
n	major_purchase	PA	25.37	1.0	695.0	699.0	
n	debt_consolidation	GA	10.20	0.0	690.0	694.0	
n	debt_consolidation	MN	14.67	0.0	680.0	684.0	
n	major_purchase	SC	17.61	1.0	705.0	709.0	
n	credit_card	PA	13.07	0.0	685.0	689.0	
n	credit_card	RI	34.80	0.0	700.0	704.0	
n	other	NC	34.95	0.0	700.0	704.0	

In [25]:
```
# Reseting the index of loan data
loan.reset_index(drop=True,inplace=True)
loan
```

Out[25]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_gra
0	3600.0	3600.0	3600.0	36 months	13.99	123.03	C	
1	24700.0	24700.0	24700.0	36 months	11.99	820.28	C	
2	20000.0	20000.0	20000.0	60 months	10.78	432.66	B	
3	10400.0	10400.0	10400.0	60 months	22.45	289.91	F	
4	11950.0	11950.0	11950.0	36 months	13.44	405.18	C	
...	...	...	...	...	...	...	...	
611777	14400.0	14400.0	14400.0	36 months	16.29	508.33	D	
611778	10000.0	10000.0	10000.0	36 months	11.47	329.62	B	
611779	16000.0	16000.0	16000.0	36 months	10.75	521.93	B	
611780	6000.0	6000.0	6000.0	36 months	7.89	187.72	A	
611781	1000.0	1000.0	1000.0	36 months	6.49	30.65	A	

611782 rows × 92 columns

In [26]:
```
loan.shape
```

Out[26]: (611782, 92)

In [27]:
```
# Replacing "Fully Paid" with 1 and "Charged Off" as 0

mapping_dict = {"loan_status":{"Fully Paid":1,"Charged Off":0}}
loan = loan.replace(mapping_dict)
```

In [28]:
```
loan.sample(5)
```

Out[28]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_gra
482064	7500.0	7500.0	7500.0	36 months	9.75	241.13	B	
248205	15000.0	15000.0	15000.0	36 months	9.17	478.19	B	
76393	7200.0	7200.0	7200.0	36 months	8.18	226.23	B	
566226	10000.0	10000.0	10000.0	36 months	10.75	326.21	B	
406618	1500.0	1500.0	1500.0	36 months	20.00	55.75	D	

The data is imbalanced, we have to consider f1_score for evaluation.

Visualization of numerical data

In [29]:
```
len(numerical_features)
```

Out[29]: 69

Handling Null values

In [30]:
```
# Categorical Features
cat_f = [feature for feature in loan.columns if loan[feature].dtypes=='object'

for feature in cat_f:
    loan[feature].fillna(value=loan[feature].mode(),inplace=True)
```

In [31]:
```
loan.head()
```

Out[31]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	e
0	3600.0	3600.0	3600.0	36 months	13.99	123.03	C	C4	
1	24700.0	24700.0	24700.0	36 months	11.99	820.28	C	C1	
2	20000.0	20000.0	20000.0	60 months	10.78	432.66	B	B4	
3	10400.0	10400.0	10400.0	60 months	22.45	289.91	F	F1	
4	11950.0	11950.0	11950.0	36 months	13.44	405.18	C	C3	

In [32]:
```
len(cat_f)
```

Out[32]:
```
22
```

In [33]:
```
# Numerical Features
num_f = [feature for feature in loan.columns if loan[feature].dtypes != 'objec

for feature in num_f:
    loan[feature].fillna(value=loan[feature].median(), inplace=True)
```

In [34]:
```
loan.isna().sum()
```

Out[34]:
```
loan_amnt                     0
funded_amnt                   0
funded_amnt_inv               0
term                          0
int_rate                      0
                           ...
earliest_cr_line_year         0
last_pymnt_d_month         1096
last_pymnt_d_year          1096
last_credit_pull_d_month     12
last_credit_pull_d_year      12
Length: 92, dtype: int64
```

In [35]:
```
# Dropping last 4 columns since, there null values are not replaced
drop = ['last_pymnt_d_month','last_pymnt_d_year','last_credit_pull_d_month','l
loan.drop(columns=drop,axis=1,inplace=True)
loan.shape
```

Out[35]:
```
(611782, 88)
```

In [36]:
```
loan.isna().sum()
```

Out[36]:
```
loan_amnt                 0
funded_amnt               0
funded_amnt_inv           0
term                      0
int_rate                  0
                        ..
debt_settlement_flag      0
issue_d_month             0
issue_d_year              0
earliest_cr_line_month    0
earliest_cr_line_year     0
Length: 88, dtype: int64
```

Label Encoding - Categorical features

In [37]:
```
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
cat_f.remove('last_pymnt_d_month')
cat_f.remove('last_pymnt_d_year')
cat_f.remove('last_credit_pull_d_month')
cat_f.remove('last_credit_pull_d_year')
cat_f
```

Out[53]:
```
['term',
 'grade',
 'sub_grade',
 'emp_length',
 'home_ownership',
 'verification_status',
 'pymnt_plan',
 'purpose',
 'addr_state',
 'initial_list_status',
 'application_type',
 'hardship_flag',
 'disbursement_method',
 'debt_settlement_flag',
 'issue_d_month',
 'issue_d_year',
 'earliest_cr_line_month',
 'earliest_cr_line_year']
```

In [39]:
```
feature
```

Out[39]:
```
'last_pymnt_d_month'
```

In [38]:
```
label_encoder = LabelEncoder()
for feature in cat_f:
    loan[feature] = label_encoder.fit_transform(loan[feature])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, ke
y, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.In
dexEngine.get_loc()

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.In
dexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

KeyError: 'last_pymnt_d_month'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18640\4225221017.py in <module>
      1 label_encoder = LabelEncoder()
      2 for feature in cat_f:
----> 3     loan[feature] = label_encoder.fit_transform(loan[feature])

~\anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3503             if self.columns.nlevels > 1:
   3504                 return self._getitem_multilevel(key)
-> 3505             indexer = self.columns.get_loc(key)
   3506             if is_integer(indexer):
   3507                 indexer = [indexer]

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, ke
y, method, tolerance)
   3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
-> 3631                 raise KeyError(key) from err
   3632             except TypeError:
   3633                 # If we have a listlike key, _check_indexing_error wi
ll raise

KeyError: 'last_pymnt_d_month'
```

In [ ]:
```
loan.head()
```

Out[55]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	em
0	3600.0	3600.0	3600.0	0	13.99	123.03	2	13	
1	24700.0	24700.0	24700.0	0	11.99	820.28	2	10	
2	20000.0	20000.0	20000.0	1	10.78	432.66	1	8	
3	10400.0	10400.0	10400.0	1	22.45	289.91	5	25	
4	11950.0	11950.0	11950.0	0	13.44	405.18	2	12	

In [40]:
```
# Changing position of target variable
loan['loan_sts'] = loan['loan_status']
loan.drop(columns='loan_status',inplace=True)
loan.head()
```

Out[40]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	em
0	3600.0	3600.0	3600.0	0	13.99	123.03	2	13	
1	24700.0	24700.0	24700.0	0	11.99	820.28	2	10	
2	20000.0	20000.0	20000.0	1	10.78	432.66	1	8	
3	10400.0	10400.0	10400.0	1	22.45	289.91	5	25	
4	11950.0	11950.0	11950.0	0	13.44	405.18	2	12	

Scaling the features

In [41]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [42]:
```
loan_copy = loan.copy()
y = loan_copy.iloc[:,-1]
loan_copy.drop(columns='loan_sts',axis=1,inplace=True)
x=loan_copy
print(x.shape,y.shape)
```

```
(611782, 87) (611782,)
```

In [43]:
```
y
```

Out[43]:
```
0         1
1         1
2         1
3         1
4         1
         ..
611777    0
611778    1
611779    1
611780    1
611781    1
Name: loan_sts, Length: 611782, dtype: int64
```

In [44]:
```
x
```

Out[44]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade
0	3600.0	3600.0	3600.0	0	13.99	123.03	2	13
1	24700.0	24700.0	24700.0	0	11.99	820.28	2	10
2	20000.0	20000.0	20000.0	1	10.78	432.66	1	8
3	10400.0	10400.0	10400.0	1	22.45	289.91	5	25
4	11950.0	11950.0	11950.0	0	13.44	405.18	2	12
...	...	...	...	...	...	...	...	..
611777	14400.0	14400.0	14400.0	0	16.29	508.33	3	15
611778	10000.0	10000.0	10000.0	0	11.47	329.62	1	9
611779	16000.0	16000.0	16000.0	0	10.75	521.93	1	8
611780	6000.0	6000.0	6000.0	0	7.89	187.72	0	4
611781	1000.0	1000.0	1000.0	0	6.49	30.65	0	1

611782 rows × 87 columns

In [45]:
```
scaler = MinMaxScaler()
scaler.fit(x)
norm_df = pd.DataFrame(scaler.transform(x),index=x.index, columns=x.columns)
norm_df
```

Out[45]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_g
0	0.066667	0.066667	0.072020	0.0	0.338006	0.063911	0.333333	0.38
1	0.607692	0.607692	0.609943	0.0	0.260125	0.472658	0.333333	0.29
2	0.487179	0.487179	0.490121	1.0	0.213006	0.245424	0.166667	0.23
3	0.241026	0.241026	0.245379	1.0	0.667445	0.161740	0.833333	0.73
4	0.280769	0.280769	0.284895	0.0	0.316589	0.229315	0.333333	0.35
...	...	...	...	...	...	...	...	
611777	0.343590	0.343590	0.347355	0.0	0.427570	0.289784	0.500000	0.44
611778	0.230769	0.230769	0.235182	0.0	0.239875	0.185020	0.166667	0.26
611779	0.384615	0.384615	0.388145	0.0	0.211838	0.297757	0.166667	0.23
611780	0.128205	0.128205	0.133206	0.0	0.100467	0.101834	0.000000	0.11
611781	0.000000	0.000000	0.005736	0.0	0.045950	0.009755	0.000000	0.02

611782 rows × 87 columns

Random Forest for feature selection

In [46]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [47]:
```
new_feature = np.random.normal(size=(611782 ,1))
```

In [48]:
```
new_feature
```

Out[48]:
```
array([[ 1.36684076],
       [ 0.77450416],
       [ 0.28179727],
       ...,
       [ 0.6032161 ],
       [-1.36022203],
       [-0.11688042]])
```

In [49]:
```
# scaling the new feature
scaler.fit(new_feature)
new_feature = pd.DataFrame(scaler.transform(new_feature))
new_feature.rename(columns={0:'new_feature'},inplace=True)
new_feature
```

Out[49]:

	new_feature
0	0.655426
1	0.590476
2	0.536450
3	0.437970
4	0.336826
...	...
611777	0.615890
611778	0.385124
611779	0.571694
611780	0.356403
611781	0.492735

611782 rows × 1 columns

In [50]:
```
new_feature.shape
```

Out[50]: (611782, 1)

In [51]:
```
norm_df.shape
```

Out[51]: (611782, 87)

In [52]:
```
# Adding this new column to the norm_df
x = pd.concat([norm_df,new_feature],axis=1,ignore_index=False)
x
```

Out[52]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_g
0	0.066667	0.066667	0.072020	0.0	0.338006	0.063911	0.333333	0.38
1	0.607692	0.607692	0.609943	0.0	0.260125	0.472658	0.333333	0.29
2	0.487179	0.487179	0.490121	1.0	0.213006	0.245424	0.166667	0.23
3	0.241026	0.241026	0.245379	1.0	0.667445	0.161740	0.833333	0.73
4	0.280769	0.280769	0.284895	0.0	0.316589	0.229315	0.333333	0.35
...	...	...	...	...	...	...	...	
611777	0.343590	0.343590	0.347355	0.0	0.427570	0.289784	0.500000	0.44
611778	0.230769	0.230769	0.235182	0.0	0.239875	0.185020	0.166667	0.26
611779	0.384615	0.384615	0.388145	0.0	0.211838	0.297757	0.166667	0.23
611780	0.128205	0.128205	0.133206	0.0	0.100467	0.101834	0.000000	0.11
611781	0.000000	0.000000	0.005736	0.0	0.045950	0.009755	0.000000	0.02

611782 rows × 88 columns

Splitting into training and testing

In [53]:
```
from sklearn.model_selection import train_test_split
```

In [54]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_stat
print(x_train.shape,y_train.shape)
```

```
(428247, 88) (428247,)
```

In [55]:
```
rf_model = RandomForestClassifier(n_estimators=100,random_state=0)
rf_model.fit(x_train,y_train)
```

Out[55]:
```
RandomForestClassifier(random_state=0)
```

In [56]:
```
# Storing all feature importances in a variable
importances = rf_model.feature_importances_             # Its an array

# Sort the feature importance in descending order and get the indexes
sorted_indices = np.argsort(importances)[::-1]          #Returns the indices

# Extracting feature names
feat_labels = x.columns[:]

# Printing feature name and its importance
for f in range(x_train.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30,
                            feat_labels[sorted_indices[f]],
                            importances[sorted_indices[f]]))
```

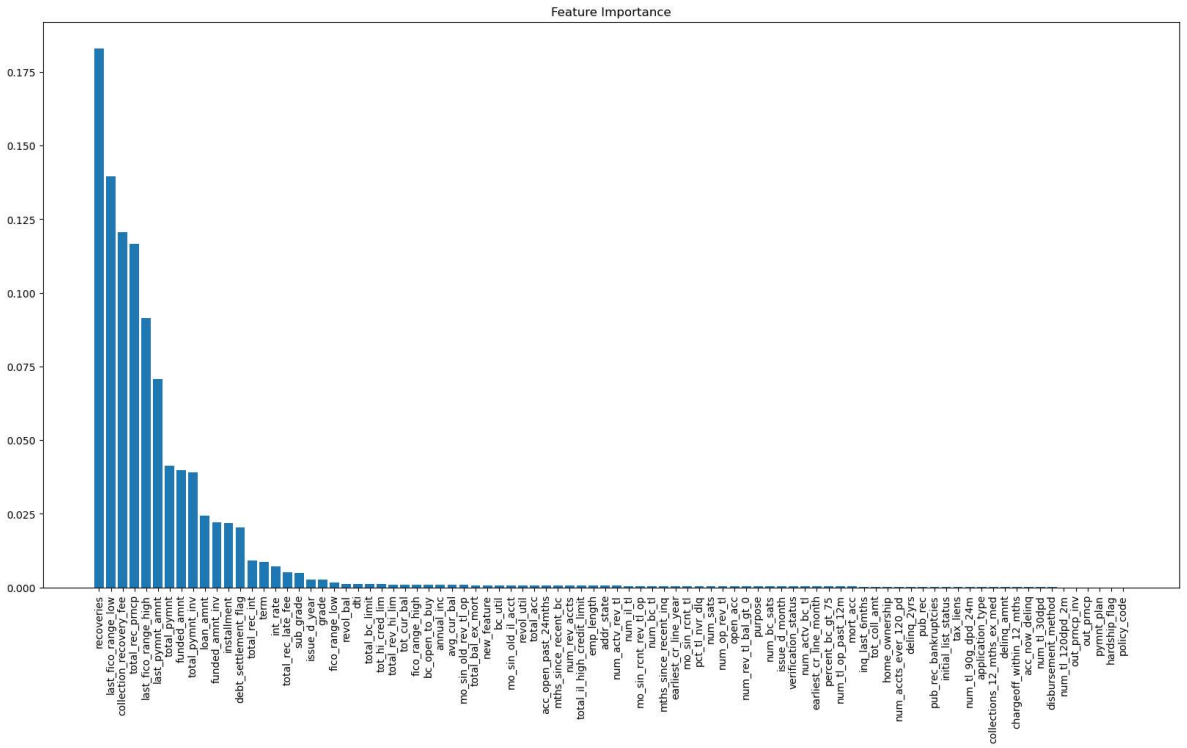
```
 1) recoveries                       0.182868
 2) last_fico_range_low              0.139454
 3) collection_recovery_fee          0.120455
 4) total_rec_prncp                  0.116557
 5) last_fico_range_high             0.091323
 6) last_pymnt_amnt                  0.070708
 7) total_pymnt                      0.041217
 8) funded_amnt                      0.039846
 9) total_pymnt_inv                  0.039093
10) loan_amnt                        0.024276
11) funded_amnt_inv                  0.021983
12) installment                      0.021814
13) debt_settlement_flag             0.020425
14) total_rec_int                    0.009144
15) term                             0.008642
16) int_rate                         0.007135
17) total_rec_late_fee               0.005059
18) sub_grade                        0.004809
19) issue_d_year                     0.002690
20) grade                            0.002533
21) fico_range_low                   0.001629
22) revol_bal                        0.001197
23) dti                              0.001089
24) total_bc_limit                   0.001039
25) tot_hi_cred_lim                  0.001020
26) total_rev_hi_lim                 0.000980
27) tot_cur_bal                      0.000949
28) fico_range_high                  0.000945
29) bc_open_to_buy                   0.000902
30) annual_inc                       0.000846
31) avg_cur_bal                      0.000826
32) mo_sin_old_rev_tl_op             0.000776
33) total_bal_ex_mort                0.000727
34) new_feature                      0.000687
35) bc_util                          0.000674
36) mo_sin_old_il_acct               0.000660
37) revol_util                       0.000651
38) total_acc                        0.000580
39) acc_open_past_24mths             0.000579
40) mths_since_recent_bc             0.000555
41) num_rev_accts                    0.000547
42) total_il_high_credit_limit       0.000547
43) emp_length                       0.000541
44) addr_state                       0.000517
45) num_actv_rev_tl                  0.000513
46) num_il_tl                        0.000501
47) mo_sin_rcnt_rev_tl_op            0.000500
48) num_bc_tl                        0.000481
49) mths_since_recent_inq            0.000475
50) earliest_cr_line_year            0.000470
51) mo_sin_rcnt_tl                   0.000463
52) pct_tl_nvr_dlq                   0.000441
53) num_sats                         0.000439
54) num_op_rev_tl                    0.000423
55) open_acc                         0.000423
56) num_rev_tl_bal_gt_0              0.000407
57) purpose                          0.000399
```

```
58) num_bc_sats                 0.000384
59) issue_d_month               0.000383
60) verification_status         0.000380
61) num_actv_bc_tl              0.000364
62) earliest_cr_line_month      0.000362
63) percent_bc_gt_75            0.000353
64) num_tl_op_past_12m          0.000333
65) mort_acc                    0.000289
66) inq_last_6mths              0.000247
67) tot_coll_amt                0.000236
68) home_ownership              0.000190
69) num_accts_ever_120_pd       0.000173
70) delinq_2yrs                 0.000167
71) pub_rec                     0.000140
72) pub_rec_bankruptcies        0.000125
73) initial_list_status         0.000115
74) tax_liens                   0.000087
75) num_tl_90g_dpd_24m          0.000068
76) application_type            0.000051
77) collections_12_mths_ex_med  0.000046
78) delinq_amnt                 0.000024
79) chargeoff_within_12_mths    0.000022
80) acc_now_delinq              0.000011
81) num_tl_30dpd                0.000011
82) disbursement_method         0.000010
83) num_tl_120dpd_2m            0.000005
84) out_prncp_inv               0.000000
85) out_prncp                   0.000000
86) pymnt_plan                  0.000000
87) hardship_flag               0.000000
88) policy_code                 0.000000
```

There are 33 such features whose imporance is more than the random feature we created, hence we will consider only those features in our model.

In [57]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
plt.title('Feature Importance')
plt.bar(range(x_train.shape[1]), importances[sorted_indices], align='center')
plt.xticks(range(x_train.shape[1]), x_train.columns[sorted_indices],rotation=9
#plt.tight_layout()
plt.show()
```



In [58]:
```
rf_model.feature_names_in_
```

Out[58]:
```
array(['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate',
       'installment', 'grade', 'sub_grade', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status',
       'pymnt_plan', 'purpose', 'addr_state', 'dti', 'delinq_2yrs',
       'fico_range_low', 'fico_range_high', 'inq_last_6mths', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', 'out_prncp', 'out_prncp_inv', 'total_pymnt',
       'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int',
       'total_rec_late_fee', 'recoveries', 'collection_recovery_fee',
       'last_pymnt_amnt', 'last_fico_range_high', 'last_fico_range_low',
       'collections_12_mths_ex_med', 'policy_code', 'application_type',
       'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal',
       'total_rev_hi_lim', 'acc_open_past_24mths', 'avg_cur_bal',
       'bc_open_to_buy', 'bc_util', 'chargeoff_within_12_mths',
       'delinq_amnt', 'mo_sin_old_il_acct', 'mo_sin_old_rev_tl_op',
       'mo_sin_rcnt_rev_tl_op', 'mo_sin_rcnt_tl', 'mort_acc',
       'mths_since_recent_bc', 'mths_since_recent_inq',
       'num_accts_ever_120_pd', 'num_actv_bc_tl', 'num_actv_rev_tl',
       'num_bc_sats', 'num_bc_tl', 'num_il_tl', 'num_op_rev_tl',
       'num_rev_accts', 'num_rev_tl_bal_gt_0', 'num_sats',
       'num_tl_120dpd_2m', 'num_tl_30dpd', 'num_tl_90g_dpd_24m',
       'num_tl_op_past_12m', 'pct_tl_nvr_dlq', 'percent_bc_gt_75',
       'pub_rec_bankruptcies', 'tax_liens', 'tot_hi_cred_lim',
       'total_bal_ex_mort', 'total_bc_limit',
       'total_il_high_credit_limit', 'hardship_flag',
       'disbursement_method', 'debt_settlement_flag', 'issue_d_month',
       'issue_d_year', 'earliest_cr_line_month', 'earliest_cr_line_year',
       'new_feature'], dtype=object)
```

In [59]:
```
names = []
values = []
for f in range(x_train.shape[1]):
    names.append(feat_labels[sorted_indices[f]])
    values.append(importances[sorted_indices[f]])
```

In [60]:
```
importance = pd.DataFrame({ "Features":names,
                            "Importance":values  })

importance
```

Out[60]:

	Features	Importance
0	recoveries	0.182868
1	last_fico_range_low	0.139454
2	collection_recovery_fee	0.120455
3	total_rec_prncp	0.116557
4	last_fico_range_high	0.091323
...	...	...
83	out_prncp_inv	0.000000
84	out_prncp	0.000000
85	pymnt_plan	0.000000
86	hardship_flag	0.000000
87	policy_code	0.000000

88 rows × 2 columns

Dropping less important features

In [61]:
```
# Setting the threshold
thr = importance[importance["Features"]=="new_feature"]["Importance"]
thr = float(thr)
thr
```

Out[61]: 0.0006865794219289554

In [62]:
```
features_to_drop = list(importance.loc[importance["Importance"]< thr,"Features
features_to_drop
```

Out[62]:
```
['bc_util',
 'mo_sin_old_il_acct',
 'revol_util',
 'total_acc',
 'acc_open_past_24mths',
 'mths_since_recent_bc',
 'num_rev_accts',
 'total_il_high_credit_limit',
 'emp_length',
 'addr_state',
 'num_actv_rev_tl',
 'num_il_tl',
 'mo_sin_rcnt_rev_tl_op',
 'num_bc_tl',
 'mths_since_recent_inq',
 'earliest_cr_line_year',
 'mo_sin_rcnt_tl',
 'pct_tl_nvr_dlq',
 'num_sats',
 'num_op_rev_tl',
 'open_acc',
 'num_rev_tl_bal_gt_0',
 'purpose',
 'num_bc_sats',
 'issue_d_month',
 'verification_status',
 'num_actv_bc_tl',
 'earliest_cr_line_month',
 'percent_bc_gt_75',
 'num_tl_op_past_12m',
 'mort_acc',
 'inq_last_6mths',
 'tot_coll_amt',
 'home_ownership',
 'num_accts_ever_120_pd',
 'delinq_2yrs',
 'pub_rec',
 'pub_rec_bankruptcies',
 'initial_list_status',
 'tax_liens',
 'num_tl_90g_dpd_24m',
 'application_type',
 'collections_12_mths_ex_med',
 'delinq_amnt',
 'chargeoff_within_12_mths',
 'acc_now_delinq',
 'num_tl_30dpd',
 'disbursement_method',
 'num_tl_120dpd_2m',
 'out_prncp_inv',
 'out_prncp',
 'pymnt_plan',
 'hardship_flag',
 'policy_code']
```

In [63]:
```
for f in features_to_drop:
    norm_df.drop(columns=f,axis=1,inplace=True)

norm_df.head()
```

Out[63]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade
0	0.066667	0.066667	0.072020	0.0	0.338006	0.063911	0.333333	0.382353
1	0.607692	0.607692	0.609943	0.0	0.260125	0.472658	0.333333	0.294118
2	0.487179	0.487179	0.490121	1.0	0.213006	0.245424	0.166667	0.235294
3	0.241026	0.241026	0.245379	1.0	0.667445	0.161740	0.833333	0.735294
4	0.280769	0.280769	0.284895	0.0	0.316589	0.229315	0.333333	0.352941

In [64]:
```
# Final normal dataframe with less number of features
norm_df.shape
```

Out[64]: (611782, 33)

In [65]:
```
y.shape
```

Out[65]: (611782,)

In [66]:
```
final_df = pd.concat([norm_df,y],axis=1)
final_df
```

Out[66]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_g
0	0.066667	0.066667	0.072020	0.0	0.338006	0.063911	0.333333	0.38
1	0.607692	0.607692	0.609943	0.0	0.260125	0.472658	0.333333	0.29
2	0.487179	0.487179	0.490121	1.0	0.213006	0.245424	0.166667	0.23
3	0.241026	0.241026	0.245379	1.0	0.667445	0.161740	0.833333	0.73
4	0.280769	0.280769	0.284895	0.0	0.316589	0.229315	0.333333	0.35
...	...	...	...	...	...	...	...	
611777	0.343590	0.343590	0.347355	0.0	0.427570	0.289784	0.500000	0.44
611778	0.230769	0.230769	0.235182	0.0	0.239875	0.185020	0.166667	0.26
611779	0.384615	0.384615	0.388145	0.0	0.211838	0.297757	0.166667	0.23
611780	0.128205	0.128205	0.133206	0.0	0.100467	0.101834	0.000000	0.11
611781	0.000000	0.000000	0.005736	0.0	0.045950	0.009755	0.000000	0.02

611782 rows × 34 columns

In [67]:
```
# Spliting into training and testing
x_train,x_test,y_train,y_test = train_test_split(final_df.iloc[:,:-1],final_df
```

In [68]:
```
print(x_train.shape,y_train.shape)
```

```
(428247, 33) (428247,)
```

In [69]:
```
print(x_test.shape,y_test.shape)
```

```
(183535, 33) (183535,)
```

Creating the Neural Network

In [70]:
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.layers.core import Dropout

# fix random seed for reproducibility
seed=7
np.random.seed(seed)
```

In [74]:
```
# Input layer will contain 38 neurons since there are 38 features

model = tf.keras.models.Sequential()

# Layer 1
model.add(Dense(512,input_dim=33,activation="relu",kernel_initializer="uniform

# Layer 2
model.add(Dense(256,activation="relu",kernel_initializer="uniform"))

# Layer 3
model.add(Dense(128,activation="relu",kernel_initializer="uniform"))

# Layer 4
model.add(Dense(64,activation="relu",kernel_initializer="uniform"))

# Layers 5
model.add(Dense(32,activation="relu",kernel_initializer="uniform"))

# Output Layer
model.add(Dense(1,activation="sigmoid"))
```

In [75]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 512)               17408

 dense_7 (Dense)             (None, 256)               131328

 dense_8 (Dense)             (None, 128)               32896

 dense_9 (Dense)             (None, 64)                8256

 dense_10 (Dense)            (None, 32)                2080

 dense_11 (Dense)            (None, 1)                 33

=================================================================
Total params: 192,001
Trainable params: 192,001
Non-trainable params: 0
_________________________________________________________________
```

In [76]:
```
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy

# Fit the model
history = model.fit(x_train, y_train, validation_data=(x_test,y_test), epochs=
                    callbacks=[EarlyStopping(monitor='val_accuracy', mode='max'
```

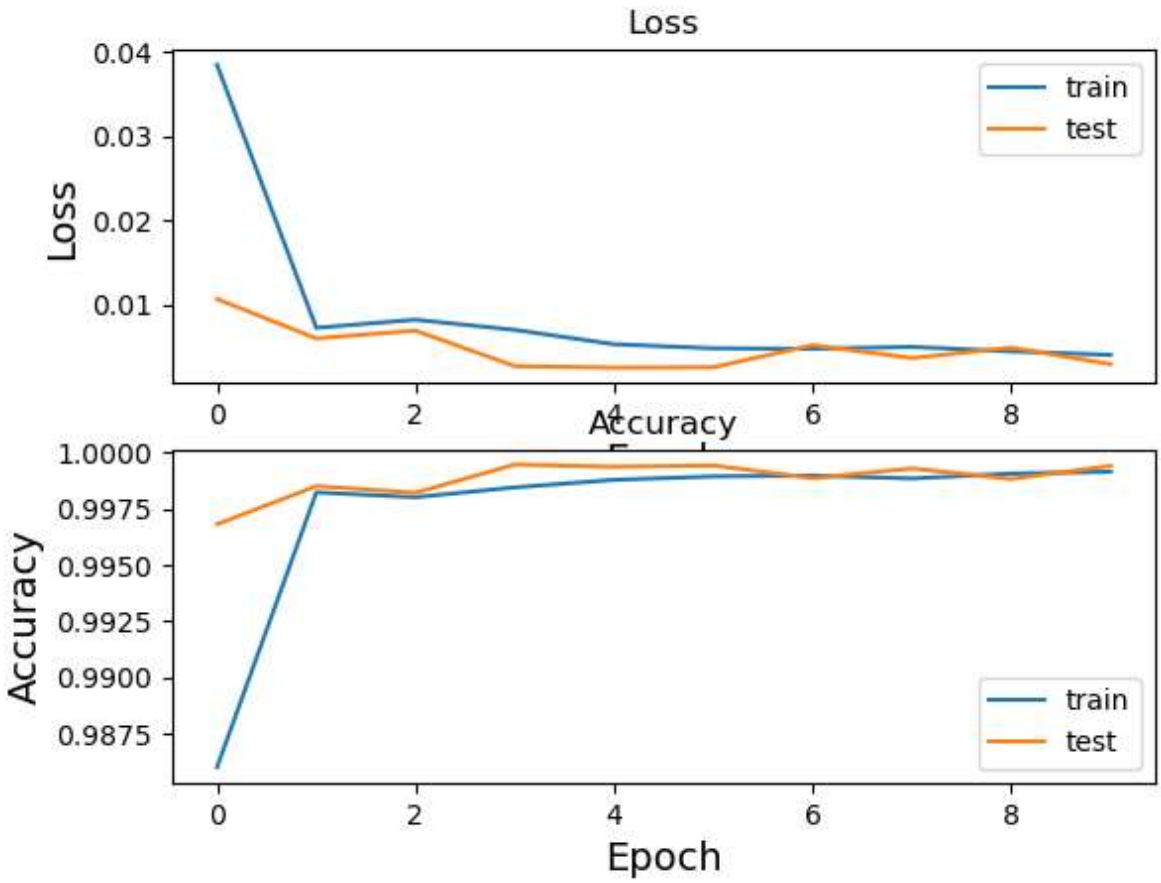
```
Epoch 1/10
837/837 [==============================] - 8s 8ms/step - loss: 0.0385 - accur
acy: 0.9860 - val_loss: 0.0107 - val_accuracy: 0.9968
Epoch 2/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0072 - accur
acy: 0.9982 - val_loss: 0.0060 - val_accuracy: 0.9985
Epoch 3/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0082 - accur
acy: 0.9980 - val_loss: 0.0069 - val_accuracy: 0.9982
Epoch 4/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0070 - accur
acy: 0.9985 - val_loss: 0.0027 - val_accuracy: 0.9995
Epoch 5/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0053 - accur
acy: 0.9988 - val_loss: 0.0025 - val_accuracy: 0.9994
Epoch 6/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0048 - accur
acy: 0.9989 - val_loss: 0.0025 - val_accuracy: 0.9994
Epoch 7/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0047 - accur
acy: 0.9990 - val_loss: 0.0052 - val_accuracy: 0.9989
Epoch 8/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0050 - accur
acy: 0.9989 - val_loss: 0.0036 - val_accuracy: 0.9993
Epoch 9/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0044 - accur
acy: 0.9991 - val_loss: 0.0049 - val_accuracy: 0.9988
Epoch 10/10
837/837 [==============================] - 7s 8ms/step - loss: 0.0040 - accur
acy: 0.9992 - val_loss: 0.0029 - val_accuracy: 0.9994
```

In [77]:
```
# evaluate the model
_, train_acc = model.evaluate(x_train, y_train, verbose=0)
_, test_acc = model.evaluate(x_train, y_train, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))

import matplotlib.pyplot as plt
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.xlabel('Epoch',size=14)
plt.ylabel('Loss',size=14)
plt.legend()

# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.xlabel('Epoch',size=14)
plt.ylabel('Accuracy',size=14)
plt.legend()
plt.show()
```

```
Train: 0.999, Test: 0.999
```



In [78]:
```
# predict probabilities for test set
yhat_probs = model.predict(x_test, verbose=0)
# predict crisp classes for test set
yhat_classes = np.argmax(model.predict(x_test),axis=1)
```

```
5736/5736 [==============================] - 8s 1ms/step
```

In [79]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [80]:
```
# Setting up the threshhold (the distribution of actual data is 80-20)
threshold = np.quantile(yhat_probs,0.2)
```

In [81]:
```
# Classification Report of test data
print(classification_report(y_test,(yhat_probs)>threshold))
```

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.97     38731
           1       0.99      1.00      0.99    144804

    accuracy                           0.99    183535
   macro avg       0.99      0.97      0.98    183535
weighted avg       0.99      0.99      0.99    183535
```

In [82]:
```
yhat_probs
```

Out[82]:
```
array([[9.995576e-01],
       [0.000000e+00],
       [9.999684e-01],
       ...,
       [1.000000e+00],
       [3.797182e-37],
       [0.000000e+00]], dtype=float32)
```

In [83]:
```
yhat_probs.mean()
```

Out[83]:
```
0.78885794
```

In [84]:
```
yhat_classes
```

Out[84]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [85]:
```
yhat_classes.mean()
```

Out[85]:
```
0.0
```

In [86]:
```
from sklearn.metrics import f1_score,accuracy_score,recall_score,precision_scc

# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, (yhat_probs)>thr)
print('Accuracy: %f' % accuracy)

# precision tp / (tp + fp)
precision = precision_score(y_test, (yhat_probs)>thr)
print('Precision: %f' % precision)

# recall: tp / (tp + fn)
recall = recall_score(y_test, (yhat_probs)>thr)
print('Recall: %f' % recall)

# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, (yhat_probs)>thr)
print('F1 score: %f' % f1)

final_results = pd.DataFrame( {"parameters": ["Accuracy", "Precision", "Recall
                               "values": [accuracy,precision,recall,f1]})
final_results
```

```
Accuracy: 0.998256
Precision: 0.997802
Recall: 0.999993
F1 score: 0.998896
```

Out[86]:

	parameters	values
0	Accuracy	0.998256
1	Precision	0.997802
2	Recall	0.999993
3	F1 score	0.998896

ROC-AUC

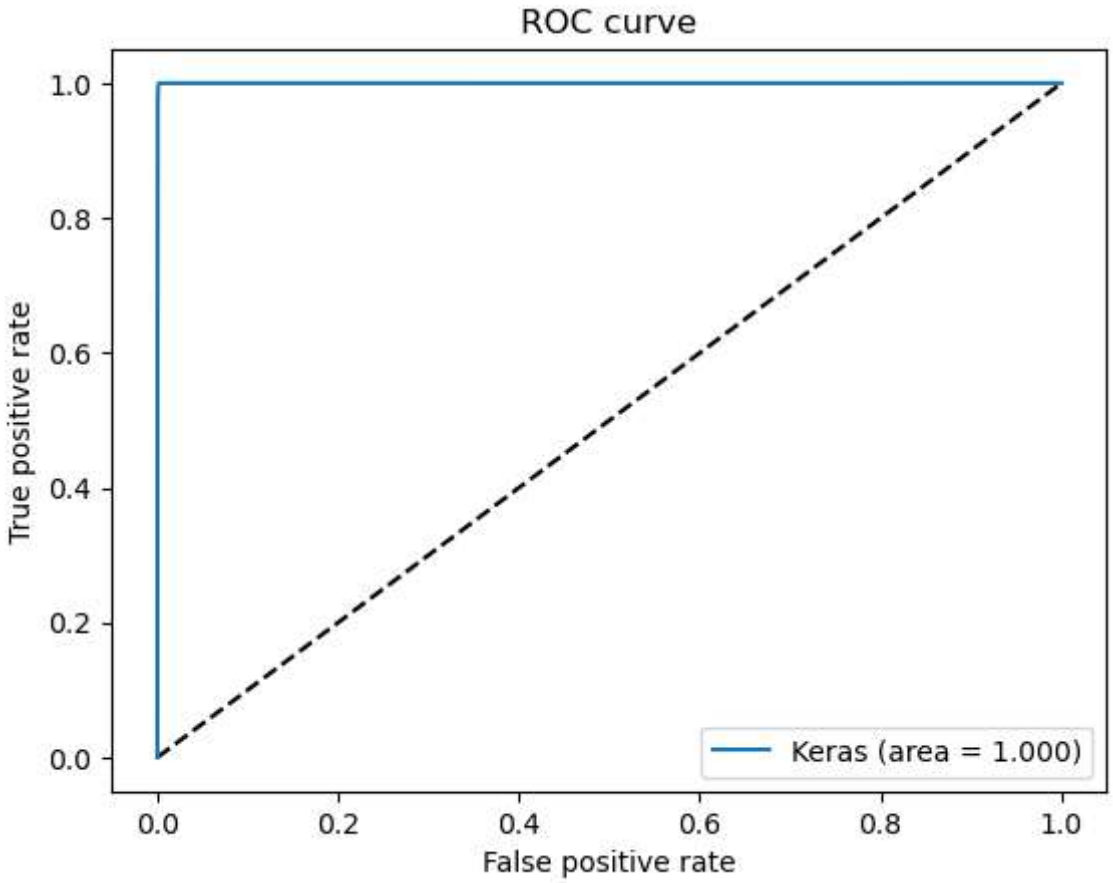
In [87]:
```
from sklearn.metrics import roc_curve
y_pred_keras = model.predict(x_test).ravel()
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred_keras)
```

```
5736/5736 [==============================] - 8s 1ms/step
```

In [88]:
```
# AUC Score
from sklearn.metrics import auc
auc_keras = auc(fpr_keras, tpr_keras)
auc_keras
```

Out[88]:
```
0.9998281586798263
```

In [89]:
```
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Keras (area = {:.3f})'.format(auc_keras)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```



In [ ]:
```
```