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 makes this problem more challenging.

Domain: Finance

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

Content:

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, tax 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)

1. Exploratory data analysis of different factors of the dataset.

2. 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

1. 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

In [121…

```
#import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report,confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model

%matplotlib inline
```

In [3]:

```
# Read the loan data CSV file
df = pd.read_csv("loan_data.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
```

```
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   object
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64
 11  delinq.2yrs        9578 non-null   int64
 12  pub.rec            9578 non-null   int64
 13  not.fully.paid     9578 non-null   int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

In [23]:

```python
df.shape
```

Out[23]:

```
(9578, 14)
```

In [24]:

```python
df.head()
```

Out[24]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	rev
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	2
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	3
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	3
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	

In [20]:

```python
#Checking if df has any null values
df.isnull().sum()
```

Out[20]:

```
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
```

In [124…

```
df.columns
```

Out[124…

```
Index(['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid'],
      dtype='object')
```

In [10]:

```
#check the data frame to understand the std and mean
dfloan.describe()
```

Out[10]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.767197
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.930377
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.000000
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.958333
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.000000
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330

In [47]:

```
df.groupby('not.fully.paid')['not.fully.paid'].count()/len(df)
```

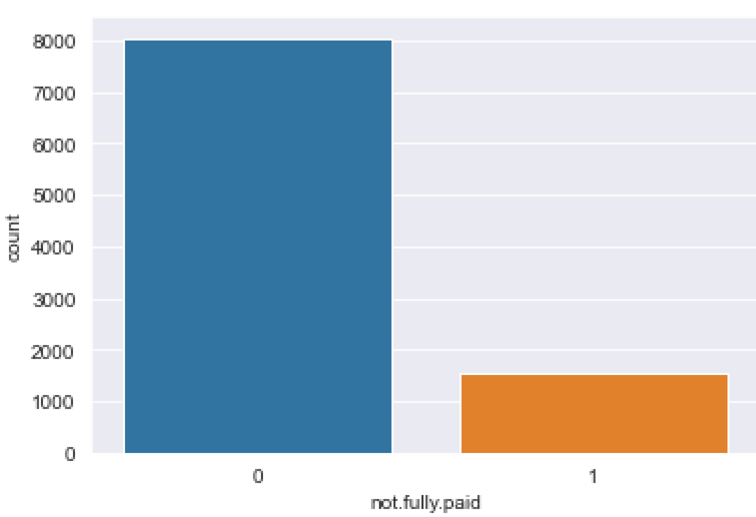
Out[47]:

```
not.fully.paid
0    0.839946
1    0.160054
Name: not.fully.paid, dtype: float64
```

In [48]:

```
sns.countplot(x='not.fully.paid',data=df)
```

Out[48]: <AxesSubplot:xlabel='not.fully.paid', ylabel='count'>



Insight:

The above count plot shows that the dataset is highly imbalanced and includes features that make this problem more challenging. If we do model training with this data, the prediction will be biased since the “not.fully.paid =0 ” has 83.99% filled, and only 16.01% is the “not.fully.paid=1”

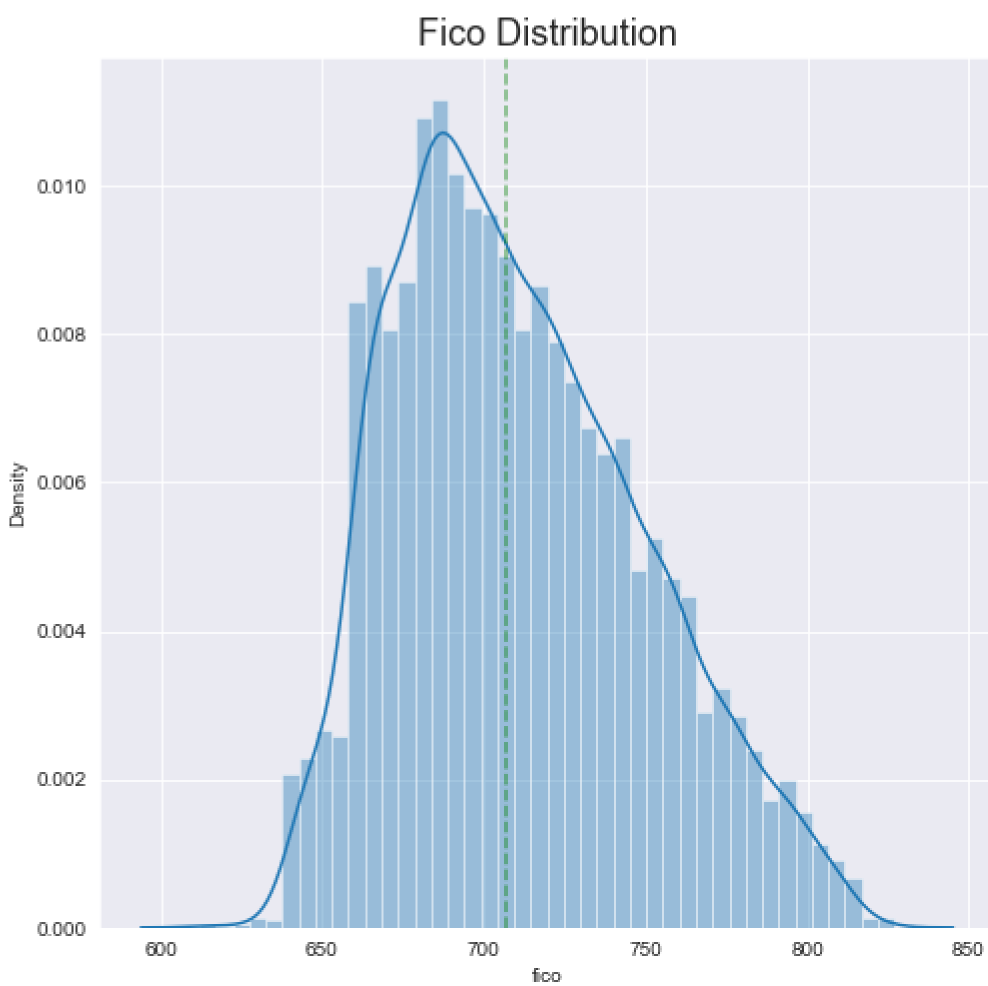
In [73]:

```
# fico distribution plot
fig, ax = plt.subplots(1,1, figsize=(8,8))
ax.set_title('Fico Distribution', fontsize=18)
sns.distplot(df['fico'], ax=ax)
ax.axvline(x=df['fico'].median(), linestyle='--', color='green', alpha=0.5)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[73]:

```
<matplotlib.lines.Line2D at 0x15aa13016d0>
```



Insight: Distribution is left skewed. The median is 707 and mean is 710.846314 which the slightly left of the peak. The negative skewness of the distribution indicates that an Lending Club may expect frequent small gains and a few large losses.

Handling of imbalance dataset

Lets try oversampling to balance this dataset. Oversampling is used when the quantity of data is insufficient. It tries to balance the dataset by increasing the size of rare samples.

In [25]:

```
class_count_0, class_count_1 = df['not.fully.paid'].value_counts()
print(f'Not Paid: {class_count_0}')
print(f'Paid: {class_count_1}')
```

```
Not Paid: 8045
Paid: 1533
```

In [75]:

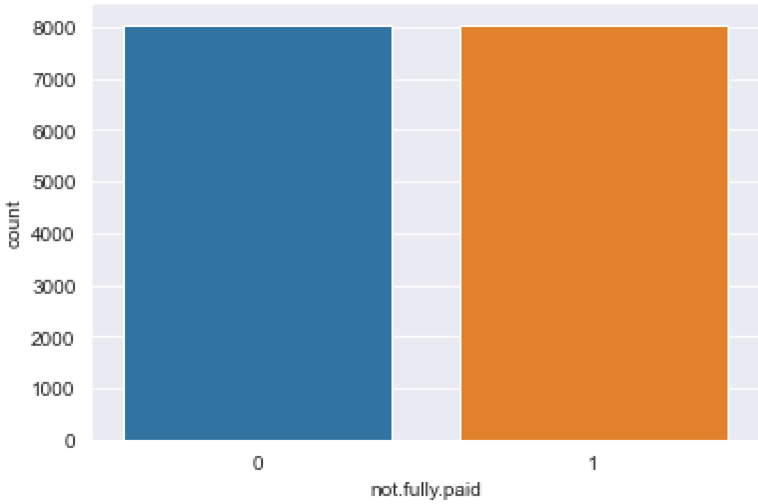
```
df_0 = df[df['not.fully.paid'] == 0]
df_1 = df[df['not.fully.paid'] == 1]
df_1_oversample = df_1.sample(class_count_0, replace=True)
df_test_oversample = pd.concat([df_0, df_1_oversample], axis=0)
df_test_oversample.info()

print('Random over-sampling:')
print(df_test_oversample['not.fully.paid'].value_counts())
sns.countplot(x='not.fully.paid', data=df_test_oversample)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16090 entries, 0 to 145
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit.policy      16090 non-null  int64  
 1   purpose            16090 non-null  object 
 2   int.rate           16090 non-null  float64
 3   installment        16090 non-null  float64
 4   log.annual.inc     16090 non-null  float64
 5   dti                16090 non-null  float64
 6   fico               16090 non-null  int64  
 7   days.with.cr.line  16090 non-null  float64
 8   revol.bal          16090 non-null  int64  
 9   revol.util         16090 non-null  float64
 10  inq.last.6mths     16090 non-null  int64  
 11  delinq.2yrs        16090 non-null  int64  
 12  pub.rec            16090 non-null  int64  
 13  not.fully.paid     16090 non-null  int64  
dtypes: float64(6), int64(7), object(1)
memory usage: 1.8+ MB
Random over-sampling:
0    8045
1    8045
Name: not.fully.paid, dtype: int64
```

Out[75]:

```
<AxesSubplot:xlabel='not.fully.paid', ylabel='count'>
```



Exploratory Data Analysis:

Data visualization with seaborn matplot lib libraries.

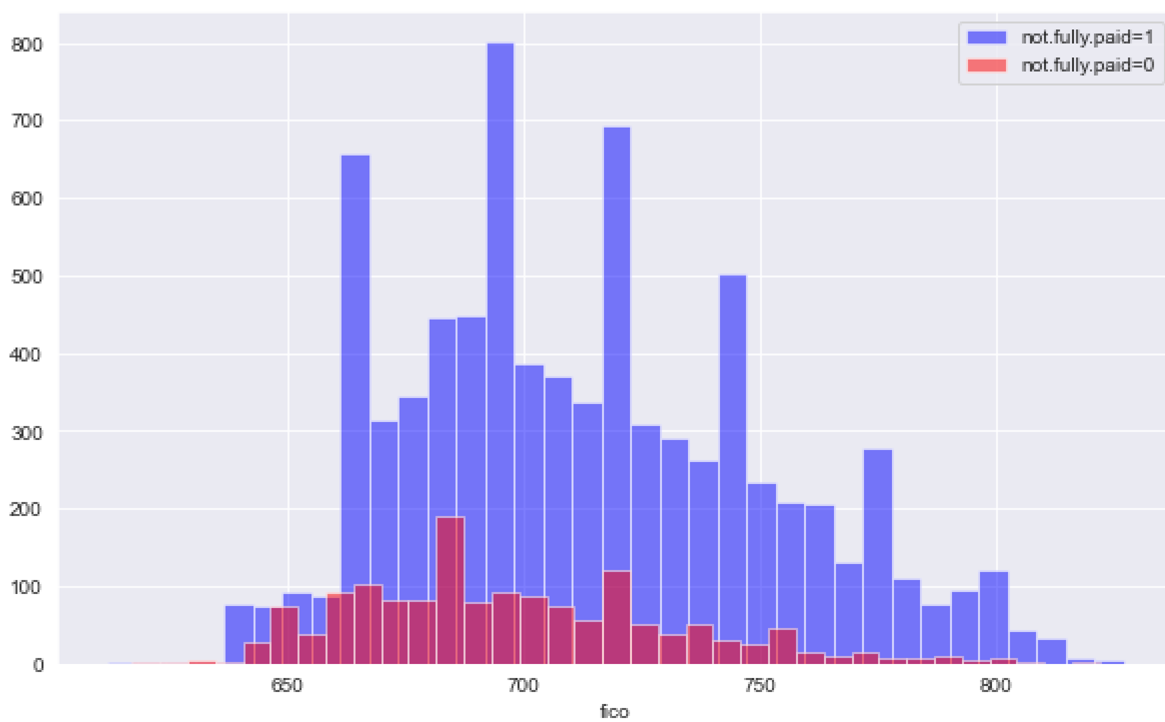
Fico Analysis on Loan paid

In [70]:

```
# Create a histogram of Fico distributions for "not.fully.paid" column.
plt.figure(figsize=(10,6))
df[df['not.fully.paid']==0]['fico'].hist(bins=35,color='blue',alpha=0.5,label='not.full
df[df['not.fully.paid']==1]['fico'].hist(bins=35,color='red', alpha=0.5,label='not.full
plt.xlabel('fico')
plt.legend()
```

Out[70]:

```
<matplotlib.legend.Legend at 0x15aa1036940>
```



Insight: Fico distributions looks similar with not much differneces for those who have fully paid balances vs those didn't paid.

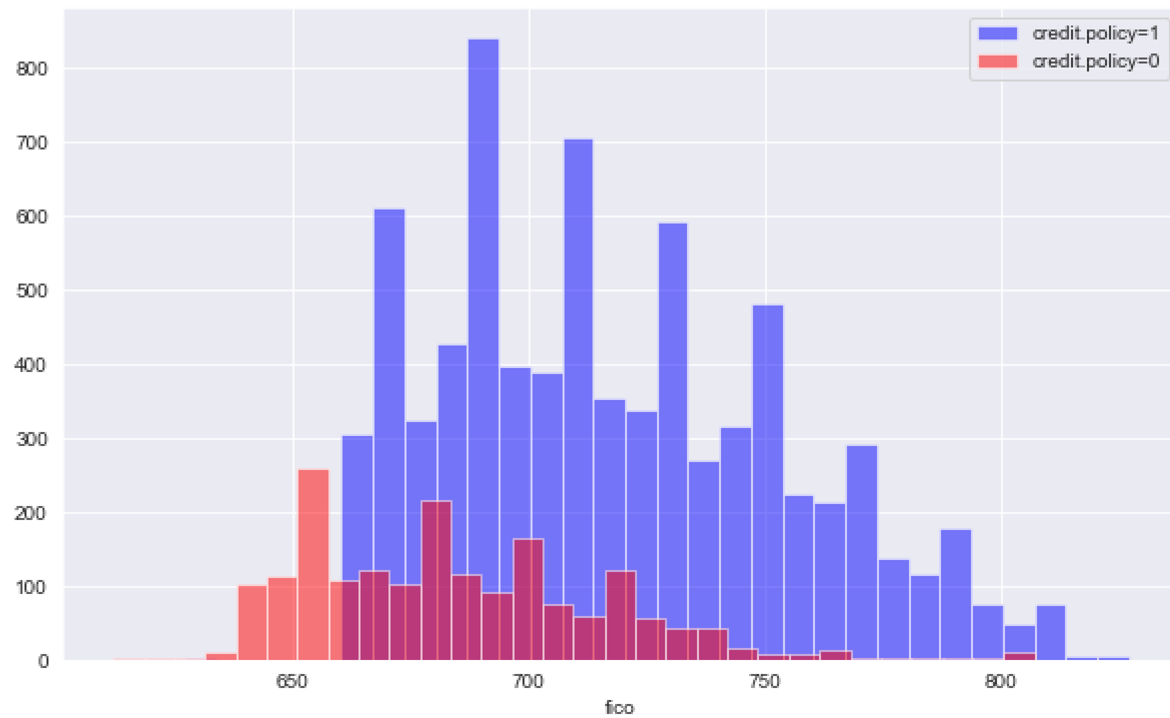
Fico Analysis on credit policy

In [69]:

```
# Create a histogram of Fico distributions for "credit.policy" column.
plt.figure(figsize=(10,6))
df[df['credit.policy']==1]['fico'].hist(bins=30,color='blue', alpha=0.5,label='credit.p
df[df['credit.policy']==0]['fico'].hist(bins=30,color='red', alpha=0.5,label='credit.po
plt.xlabel('fico')
plt.legend()
```

Out[69]:

```
<matplotlib.legend.Legend at 0x15aa0ed4a00>
```



Insight: Applicants with higher Fico scores seems to meet credit policy cretiria.

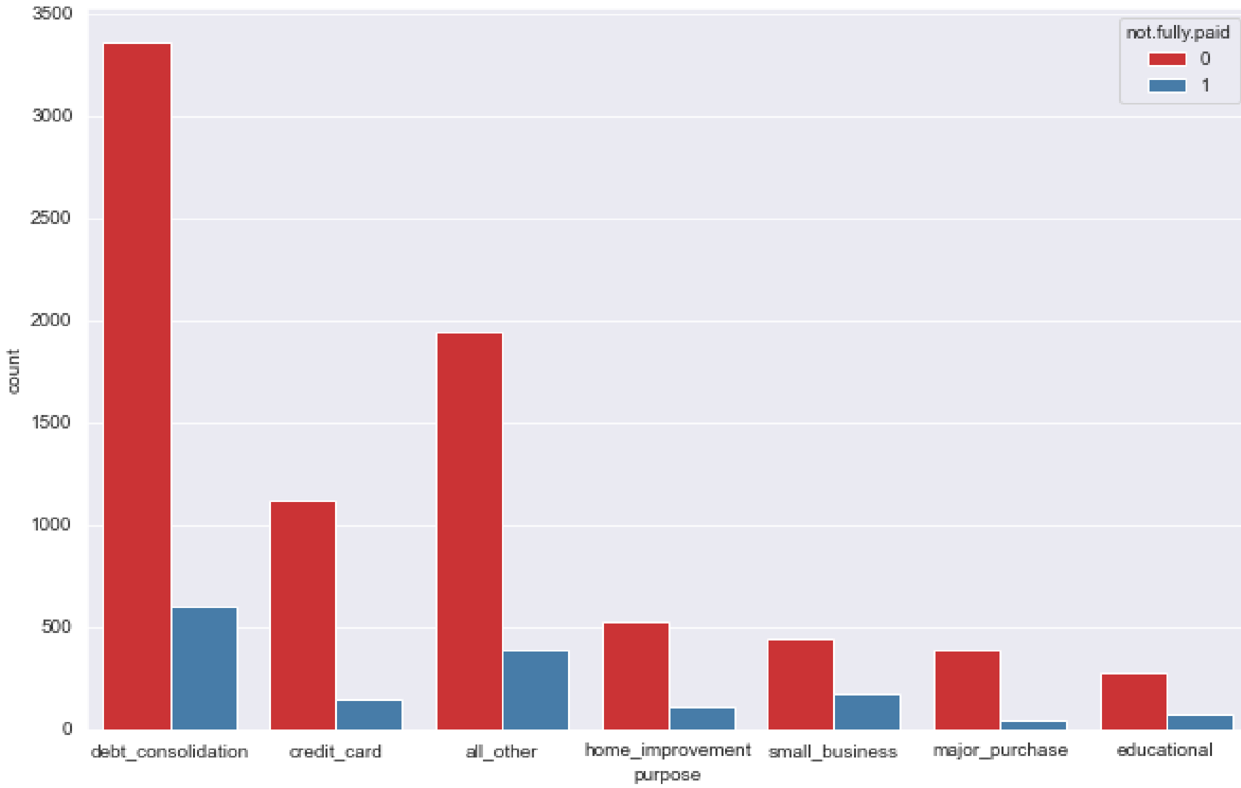
Data set group by loan purpose with Loan paid:

In [54]:

```
# Check the dataset group by loan purpose. Create a countplot with the color hue define
plt.figure(figsize=(11,7))
sns.countplot(x='purpose',hue='not.fully.paid',data=df,palette='Set1')
```

Out[54]:

```
<AxesSubplot:xlabel='purpose', ylabel='count'>
```



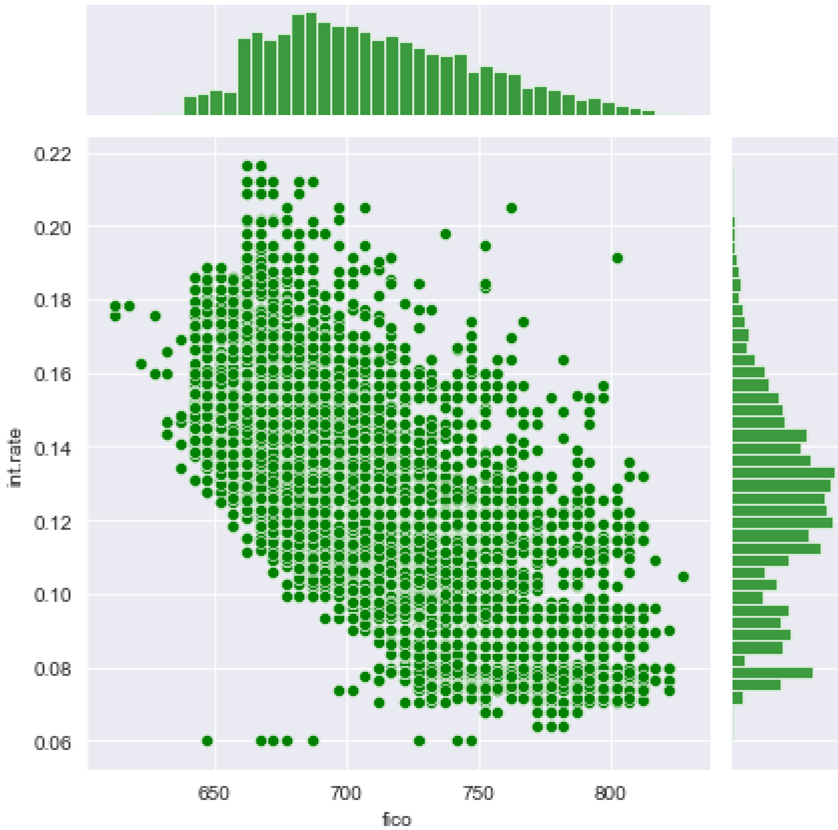
Trend: Fico V/s int.rate

In [66]:

```
#create the joinplot to capture the trend between Fico distributions and int.rate
sns.jointplot(x='fico',y='int.rate',data=df,color='green')
```

Out[66]:

```
<seaborn.axisgrid.JointGrid at 0x15a9fa7b340>
```

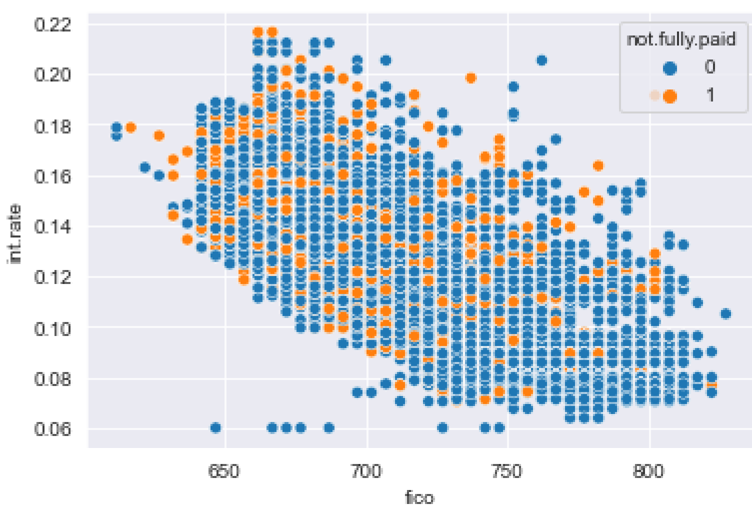


Trend: Fico score V/s interest rate with respect to Loan paid

In [68]:

```
#create scatter plot to visulaize the relationship between fico, int.rate with Loan pai
sns.scatterplot(x='fico', y='int.rate', data=df[['fico', 'int.rate', 'not.fully.paid']]
```

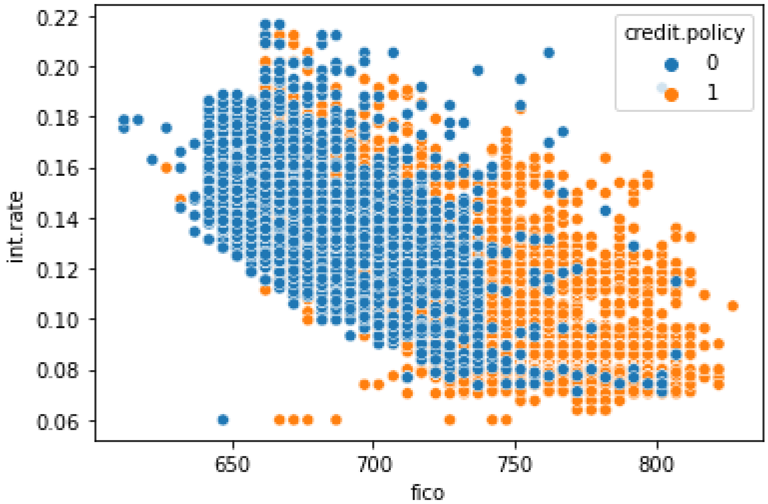
Out[68]: <AxesSubplot:xlabel='fico', ylabel='int.rate'>

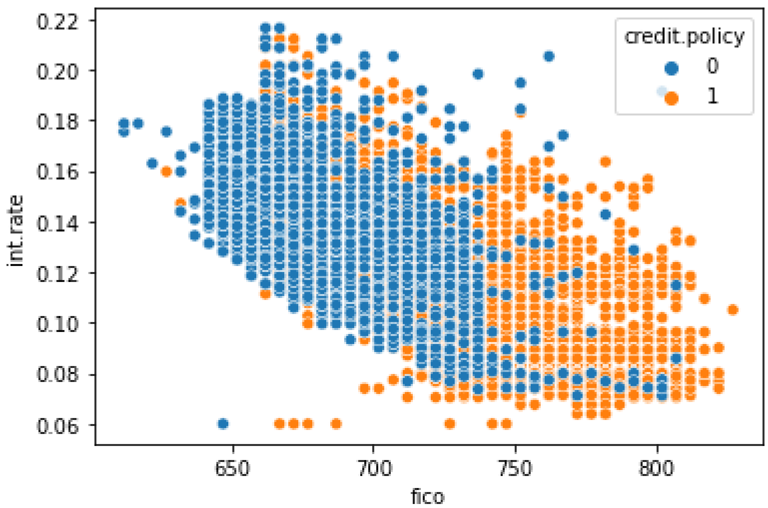


Trend: Fico score V/s interest rate with respect to credit policy

In [167…

```
sns.scatterplot(x='fico', y='int.rate', data=dfxfrm[['fico', 'int.rate', 'credit.policy
plt.show()
sns.scatterplot(x='fico', y='int.rate', data=dfxfrm, hue='credit.policy')
plt.show()
```





Trend: Loan paid V/s Credit policy

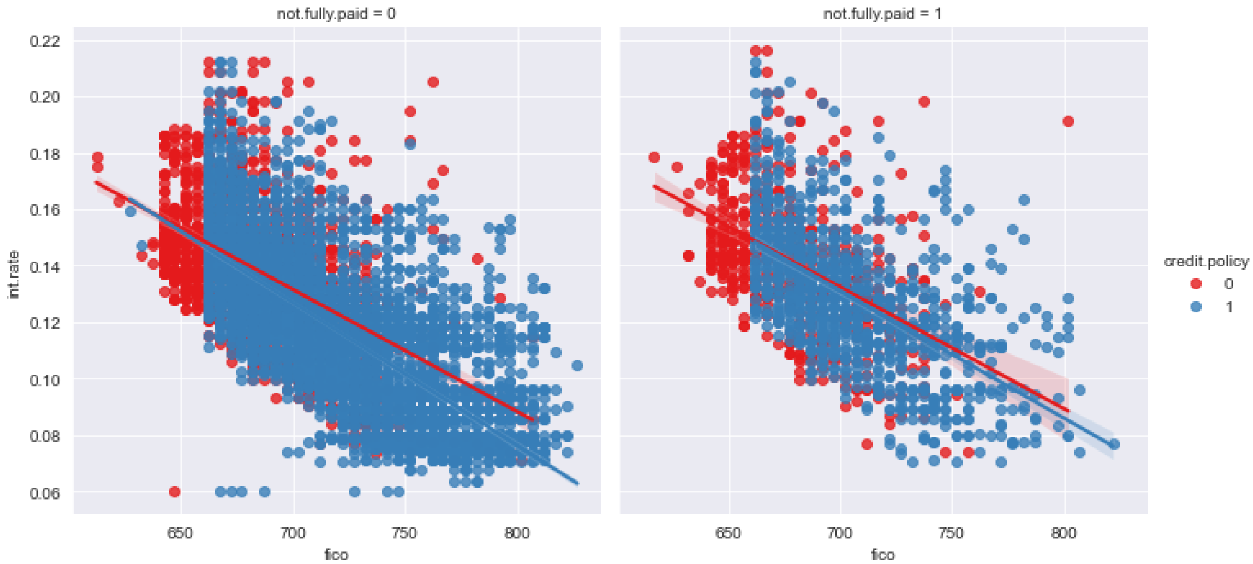
In [67]:

```
#Create seaborn lmplot to compare the trend between not.fully.paid and credit.policy.
plt.figure(figsize=(11,7))
sns.lmplot(x='fico',y='int.rate',data=df,col='not.fully.paid',hue='credit.policy', pale
```

Out[67]:

```
<seaborn.axisgrid.FacetGrid at 0x15a9fa74040>

<Figure size 792x504 with 0 Axes>
```



Insight: The trend of Fico score and interest rate is similar as we expect for those who have fully paid loan vs those didn't paid.

Feature transformation:

The “purpose” data column is categorical. Transforms the categorial value into numerical value.

In [78]:

```
#  Transform categorical values into numerical values (discrete)
dfloan = pd.get_dummies(df, columns=['purpose'], drop_first=True)
dfloan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 19 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   credit.policy               9578 non-null   int64  
 1   int.rate                    9578 non-null   float64
 2   installment                 9578 non-null   float64
 3   log.annual.inc              9578 non-null   float64
 4   dti                         9578 non-null   float64
 5   fico                        9578 non-null   int64  
 6   days.with.cr.line           9578 non-null   float64
 7   revol.bal                   9578 non-null   int64  
 8   revol.util                  9578 non-null   float64
 9   inq.last.6mths              9578 non-null   int64  
 10  delinq.2yrs                 9578 non-null   int64  
 11  pub.rec                     9578 non-null   int64  
 12  not.fully.paid              9578 non-null   int64  
 13  purpose_credit_card         9578 non-null   uint8  
 14  purpose_debt_consolidation  9578 non-null   uint8  
 15  purpose_educational         9578 non-null   uint8  
 16  purpose_home_improvement    9578 non-null   uint8  
 17  purpose_major_purchase      9578 non-null   uint8  
 18  purpose_small_business      9578 non-null   uint8  
dtypes: float64(6), int64(7), uint8(6)
memory usage: 1.0 MB
```

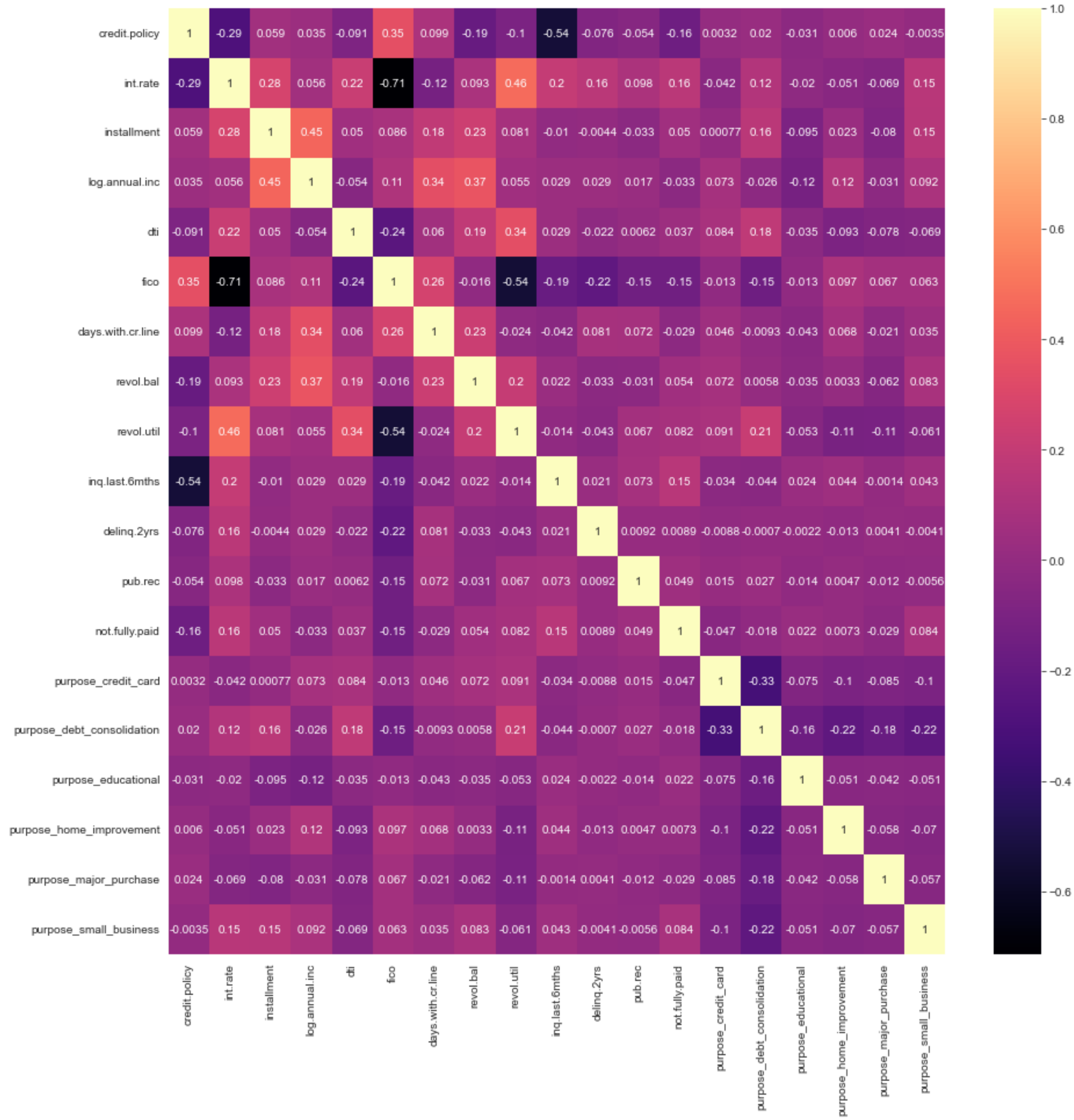
In [77]:
```
dfloan.head()
```

Out[77]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	in
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	

In [94]:
```
plt.figure(figsize=[15,15])
sns.heatmap(data=dfloan.corr(), cmap='magma', annot=True, fmt='.2g')
```

Out[94]:
```
<AxesSubplot:>
```



Build Model:

Train test split

In [109…

```
# extracting feature variables
X =dfloan.drop('not.fully.paid', axis=1)

# target variable
y = dfloan['not.fully.paid']


# train test split of dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state =
```

Normalize the data

In [108…

```
# Normalizaton of data
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train.shape
```

Out[108…

```
(6704, 18)
```

Create model

In [110…

```
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25)

model = Sequential()

#input layer
model.add(Dense(18,  activation='relu'))
model.add(Dropout(0.2))

#hidden layer
model.add(Dense(9, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(5, activation='relu'))
model.add(Dropout(0.2))

#output layer
model.add(Dense(units=1,activation='sigmoid'))

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

In [111…

```
# Fit the model to the training data. adding the validation data for later plotting.
#Added a batch_size of 256.

model.fit(
        X_train,
        y_train,
        epochs=100,
        batch_size=256,
        validation_data=(X_test, y_test), verbose=1,
        callbacks=[early_stop]
)
```

```
Epoch 1/100
27/27 [==============================] - 6s 20ms/step - loss: 915.2096 - accuracy: 0.711
4 - val_loss: 220.2793 - val_accuracy: 0.8459
Epoch 2/100
27/27 [==============================] - 0s 5ms/step - loss: 732.4688 - accuracy: 0.6921
- val_loss: 141.7928 - val_accuracy: 0.8459
Epoch 3/100
27/27 [==============================] - 0s 6ms/step - loss: 453.3801 - accuracy: 0.7148
- val_loss: 87.0846 - val_accuracy: 0.8459
Epoch 4/100
```

```
27/27 [==============================] - 0s 5ms/step - loss: 357.6108 - accuracy: 0.7151
- val_loss: 61.7612 - val_accuracy: 0.8459
Epoch 5/100
27/27 [==============================] - 0s 5ms/step - loss: 197.6937 - accuracy: 0.7504
- val_loss: 32.1508 - val_accuracy: 0.8459
Epoch 6/100
27/27 [==============================] - 0s 5ms/step - loss: 87.8579 - accuracy: 0.7818
- val_loss: 7.8447 - val_accuracy: 0.8459
Epoch 7/100
27/27 [==============================] - 0s 6ms/step - loss: 34.8464 - accuracy: 0.8138
- val_loss: 0.6228 - val_accuracy: 0.8459
Epoch 8/100
27/27 [==============================] - 0s 6ms/step - loss: 10.7414 - accuracy: 0.8209
- val_loss: 0.6112 - val_accuracy: 0.8459
Epoch 9/100
27/27 [==============================] - 0s 6ms/step - loss: 8.0882 - accuracy: 0.8282 -
val_loss: 0.6019 - val_accuracy: 0.8459
Epoch 10/100
27/27 [==============================] - 0s 5ms/step - loss: 8.6921 - accuracy: 0.8297 -
val_loss: 0.5912 - val_accuracy: 0.8459
Epoch 11/100
27/27 [==============================] - 0s 6ms/step - loss: 3.5389 - accuracy: 0.8305 -
val_loss: 0.5817 - val_accuracy: 0.8459
Epoch 12/100
27/27 [==============================] - 0s 5ms/step - loss: 4.3680 - accuracy: 0.8307 -
val_loss: 0.5729 - val_accuracy: 0.8459
Epoch 13/100
27/27 [==============================] - 0s 5ms/step - loss: 3.3548 - accuracy: 0.8319 -
val_loss: 0.5651 - val_accuracy: 0.8459
Epoch 14/100
27/27 [==============================] - 0s 6ms/step - loss: 2.4235 - accuracy: 0.8331 -
val_loss: 0.5578 - val_accuracy: 0.8459
Epoch 15/100
27/27 [==============================] - 0s 6ms/step - loss: 2.3015 - accuracy: 0.8358 -
val_loss: 0.5510 - val_accuracy: 0.8459
Epoch 16/100
27/27 [==============================] - 0s 8ms/step - loss: 2.8636 - accuracy: 0.8328 -
val_loss: 0.5447 - val_accuracy: 0.8459
Epoch 17/100
27/27 [==============================] - 0s 5ms/step - loss: 1.9792 - accuracy: 0.8335 -
val_loss: 0.5387 - val_accuracy: 0.8459
Epoch 18/100
27/27 [==============================] - 0s 5ms/step - loss: 4.2501 - accuracy: 0.8332 -
val_loss: 0.5330 - val_accuracy: 0.8459
Epoch 19/100
27/27 [==============================] - 0s 5ms/step - loss: 2.0361 - accuracy: 0.8329 -
val_loss: 0.5277 - val_accuracy: 0.8459
Epoch 20/100
27/27 [==============================] - 0s 5ms/step - loss: 1.7780 - accuracy: 0.8359 -
val_loss: 0.5229 - val_accuracy: 0.8459
Epoch 21/100
27/27 [==============================] - 0s 5ms/step - loss: 3.1701 - accuracy: 0.8349 -
val_loss: 0.5182 - val_accuracy: 0.8459
Epoch 22/100
27/27 [==============================] - 0s 5ms/step - loss: 1.2838 - accuracy: 0.8346 -
val_loss: 0.5139 - val_accuracy: 0.8459
Epoch 23/100
27/27 [==============================] - 0s 5ms/step - loss: 1.5718 - accuracy: 0.8350 -
val_loss: 0.5098 - val_accuracy: 0.8459
Epoch 24/100
```

```
27/27 [==============================] - 0s 5ms/step - loss: 1.1209 - accuracy: 0.8352 -
val_loss: 0.5059 - val_accuracy: 0.8459
Epoch 25/100
27/27 [==============================] - 0s 5ms/step - loss: 1.6352 - accuracy: 0.8350 -
val_loss: 0.5022 - val_accuracy: 0.8459
Epoch 26/100
27/27 [==============================] - 0s 6ms/step - loss: 1.2470 - accuracy: 0.8346 -
val_loss: 0.4987 - val_accuracy: 0.8459
Epoch 27/100
27/27 [==============================] - 0s 7ms/step - loss: 1.6039 - accuracy: 0.8361 -
val_loss: 0.4954 - val_accuracy: 0.8459
Epoch 28/100
27/27 [==============================] - 0s 5ms/step - loss: 0.9732 - accuracy: 0.8350 -
val_loss: 0.4923 - val_accuracy: 0.8459
Epoch 29/100
27/27 [==============================] - 0s 5ms/step - loss: 1.2853 - accuracy: 0.8355 -
val_loss: 0.4892 - val_accuracy: 0.8459
Epoch 30/100
27/27 [==============================] - 0s 5ms/step - loss: 1.6590 - accuracy: 0.8350 -
val_loss: 0.4862 - val_accuracy: 0.8459
Epoch 31/100
27/27 [==============================] - 0s 6ms/step - loss: 1.2279 - accuracy: 0.8356 -
val_loss: 0.4834 - val_accuracy: 0.8459
Epoch 32/100
27/27 [==============================] - 0s 6ms/step - loss: 2.1841 - accuracy: 0.8355 -
val_loss: 0.4809 - val_accuracy: 0.8459
Epoch 33/100
27/27 [==============================] - 0s 6ms/step - loss: 1.0326 - accuracy: 0.8358 -
val_loss: 0.4783 - val_accuracy: 0.8459
Epoch 34/100
27/27 [==============================] - 0s 5ms/step - loss: 0.7252 - accuracy: 0.8361 -
val_loss: 0.4761 - val_accuracy: 0.8459
Epoch 35/100
27/27 [==============================] - 0s 6ms/step - loss: 0.8668 - accuracy: 0.8365 -
val_loss: 0.4738 - val_accuracy: 0.8459
Epoch 36/100
27/27 [==============================] - 0s 7ms/step - loss: 1.3664 - accuracy: 0.8355 -
val_loss: 0.4716 - val_accuracy: 0.8459
Epoch 37/100
27/27 [==============================] - 0s 5ms/step - loss: 0.8342 - accuracy: 0.8358 -
val_loss: 0.4695 - val_accuracy: 0.8459
Epoch 38/100
27/27 [==============================] - 0s 5ms/step - loss: 1.0104 - accuracy: 0.8356 -
val_loss: 0.4676 - val_accuracy: 0.8459
Epoch 39/100
27/27 [==============================] - 0s 5ms/step - loss: 1.4032 - accuracy: 0.8356 -
val_loss: 0.4658 - val_accuracy: 0.8459
Epoch 40/100
27/27 [==============================] - 0s 5ms/step - loss: 0.8466 - accuracy: 0.8367 -
val_loss: 0.4640 - val_accuracy: 0.8459
Epoch 41/100
27/27 [==============================] - 0s 5ms/step - loss: 0.9408 - accuracy: 0.8359 -
val_loss: 0.4623 - val_accuracy: 0.8459
Epoch 42/100
27/27 [==============================] - 0s 6ms/step - loss: 0.9661 - accuracy: 0.8367 -
val_loss: 0.4608 - val_accuracy: 0.8459
Epoch 43/100
27/27 [==============================] - 0s 6ms/step - loss: 0.6793 - accuracy: 0.8365 -
val_loss: 0.4593 - val_accuracy: 0.8459
Epoch 44/100
```

```
27/27 [==============================] - 0s 6ms/step - loss: 0.7497 - accuracy: 0.8368 -
val_loss: 0.4578 - val_accuracy: 0.8459
Epoch 45/100
27/27 [==============================] - 0s 6ms/step - loss: 0.7005 - accuracy: 0.8365 -
val_loss: 0.4565 - val_accuracy: 0.8459
Epoch 46/100
27/27 [==============================] - 0s 6ms/step - loss: 0.7985 - accuracy: 0.8365 -
val_loss: 0.4552 - val_accuracy: 0.8459
Epoch 47/100
27/27 [==============================] - 0s 6ms/step - loss: 0.6428 - accuracy: 0.8368 -
val_loss: 0.4541 - val_accuracy: 0.8459
Epoch 48/100
27/27 [==============================] - 0s 5ms/step - loss: 0.8660 - accuracy: 0.8367 -
val_loss: 0.4529 - val_accuracy: 0.8459
Epoch 49/100
27/27 [==============================] - 0s 5ms/step - loss: 0.8313 - accuracy: 0.8361 -
val_loss: 0.4517 - val_accuracy: 0.8459
Epoch 50/100
27/27 [==============================] - 0s 6ms/step - loss: 1.2550 - accuracy: 0.8362 -
val_loss: 0.4506 - val_accuracy: 0.8459
Epoch 51/100
27/27 [==============================] - 0s 5ms/step - loss: 1.8454 - accuracy: 0.8373 -
val_loss: 0.4496 - val_accuracy: 0.8459
Epoch 52/100
27/27 [==============================] - 0s 5ms/step - loss: 0.7485 - accuracy: 0.8362 -
val_loss: 0.4487 - val_accuracy: 0.8459
Epoch 53/100
27/27 [==============================] - 0s 5ms/step - loss: 0.7850 - accuracy: 0.8362 -
val_loss: 0.4477 - val_accuracy: 0.8459
Epoch 54/100
27/27 [==============================] - 0s 6ms/step - loss: 2.6169 - accuracy: 0.8361 -
val_loss: 0.4468 - val_accuracy: 0.8459
Epoch 55/100
27/27 [==============================] - 0s 6ms/step - loss: 0.6142 - accuracy: 0.8370 -
val_loss: 0.4460 - val_accuracy: 0.8459
Epoch 56/100
27/27 [==============================] - 0s 6ms/step - loss: 1.0404 - accuracy: 0.8364 -
val_loss: 0.4453 - val_accuracy: 0.8459
Epoch 57/100
27/27 [==============================] - 0s 6ms/step - loss: 0.8086 - accuracy: 0.8367 -
val_loss: 0.4445 - val_accuracy: 0.8459
Epoch 58/100
27/27 [==============================] - 0s 8ms/step - loss: 0.7501 - accuracy: 0.8371 -
val_loss: 0.4438 - val_accuracy: 0.8459
Epoch 59/100
27/27 [==============================] - 0s 5ms/step - loss: 0.9212 - accuracy: 0.8368 -
val_loss: 0.4431 - val_accuracy: 0.8459
Epoch 60/100
27/27 [==============================] - 0s 6ms/step - loss: 0.9901 - accuracy: 0.8368 -
val_loss: 0.4424 - val_accuracy: 0.8459
Epoch 61/100
27/27 [==============================] - 0s 6ms/step - loss: 0.5704 - accuracy: 0.8364 -
val_loss: 0.4418 - val_accuracy: 0.8459
Epoch 62/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4870 - accuracy: 0.8364 -
val_loss: 0.4412 - val_accuracy: 0.8459
Epoch 63/100
27/27 [==============================] - 0s 7ms/step - loss: 0.6144 - accuracy: 0.8364 -
val_loss: 0.4406 - val_accuracy: 0.8459
Epoch 64/100
```

```
27/27 [==============================] - 0s 5ms/step - loss: 0.7156 - accuracy: 0.8367 -
val_loss: 0.4400 - val_accuracy: 0.8459
Epoch 65/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4666 - accuracy: 0.8370 -
val_loss: 0.4396 - val_accuracy: 0.8459
Epoch 66/100
27/27 [==============================] - 0s 5ms/step - loss: 0.7773 - accuracy: 0.8367 -
val_loss: 0.4391 - val_accuracy: 0.8459
Epoch 67/100
27/27 [==============================] - 0s 5ms/step - loss: 1.3454 - accuracy: 0.8362 -
val_loss: 0.4386 - val_accuracy: 0.8459
Epoch 68/100
27/27 [==============================] - 0s 6ms/step - loss: 0.5372 - accuracy: 0.8373 -
val_loss: 0.4382 - val_accuracy: 0.8459
Epoch 69/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4852 - accuracy: 0.8371 -
val_loss: 0.4377 - val_accuracy: 0.8459
Epoch 70/100
27/27 [==============================] - 0s 5ms/step - loss: 0.5019 - accuracy: 0.8376 -
val_loss: 0.4373 - val_accuracy: 0.8459
Epoch 71/100
27/27 [==============================] - 0s 6ms/step - loss: 0.7467 - accuracy: 0.8367 -
val_loss: 0.4370 - val_accuracy: 0.8459
Epoch 72/100
27/27 [==============================] - 0s 6ms/step - loss: 0.5221 - accuracy: 0.8365 -
val_loss: 0.4366 - val_accuracy: 0.8459
Epoch 73/100
27/27 [==============================] - 0s 6ms/step - loss: 0.6585 - accuracy: 0.8368 -
val_loss: 0.4363 - val_accuracy: 0.8459
Epoch 74/100
27/27 [==============================] - 0s 6ms/step - loss: 0.8146 - accuracy: 0.8370 -
val_loss: 0.4360 - val_accuracy: 0.8459
Epoch 75/100
27/27 [==============================] - 0s 5ms/step - loss: 0.5766 - accuracy: 0.8368 -
val_loss: 0.4356 - val_accuracy: 0.8459
Epoch 76/100
27/27 [==============================] - 0s 6ms/step - loss: 1.0840 - accuracy: 0.8367 -
val_loss: 0.4354 - val_accuracy: 0.8459
Epoch 77/100
27/27 [==============================] - 0s 6ms/step - loss: 0.5708 - accuracy: 0.8367 -
val_loss: 0.4351 - val_accuracy: 0.8459
Epoch 78/100
27/27 [==============================] - 0s 5ms/step - loss: 0.5169 - accuracy: 0.8370 -
val_loss: 0.4348 - val_accuracy: 0.8459
Epoch 79/100
27/27 [==============================] - 0s 6ms/step - loss: 1.9403 - accuracy: 0.8367 -
val_loss: 0.4346 - val_accuracy: 0.8459
Epoch 80/100
27/27 [==============================] - 0s 7ms/step - loss: 0.5684 - accuracy: 0.8374 -
val_loss: 0.4343 - val_accuracy: 0.8459
Epoch 81/100
27/27 [==============================] - 0s 7ms/step - loss: 1.2093 - accuracy: 0.8361 -
val_loss: 0.4341 - val_accuracy: 0.8459
Epoch 82/100
27/27 [==============================] - 0s 6ms/step - loss: 0.5543 - accuracy: 0.8367 -
val_loss: 0.4339 - val_accuracy: 0.8459
Epoch 83/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4828 - accuracy: 0.8373 -
val_loss: 0.4336 - val_accuracy: 0.8459
Epoch 84/100
```

```
27/27 [==============================] - 0s 6ms/step - loss: 0.5049 - accuracy: 0.8370 -
val_loss: 0.4334 - val_accuracy: 0.8459
Epoch 85/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4795 - accuracy: 0.8374 -
val_loss: 0.4333 - val_accuracy: 0.8459
Epoch 86/100
27/27 [==============================] - 0s 6ms/step - loss: 0.8141 - accuracy: 0.8371 -
val_loss: 0.4331 - val_accuracy: 0.8459
Epoch 87/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4886 - accuracy: 0.8370 -
val_loss: 0.4329 - val_accuracy: 0.8459
Epoch 88/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4762 - accuracy: 0.8371 -
val_loss: 0.4328 - val_accuracy: 0.8459
Epoch 89/100
27/27 [==============================] - 0s 6ms/step - loss: 0.8243 - accuracy: 0.8365 -
val_loss: 0.4327 - val_accuracy: 0.8459
Epoch 90/100
27/27 [==============================] - 0s 6ms/step - loss: 0.5130 - accuracy: 0.8371 -
val_loss: 0.4325 - val_accuracy: 0.8459
Epoch 91/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4664 - accuracy: 0.8373 -
val_loss: 0.4324 - val_accuracy: 0.8459
Epoch 92/100
27/27 [==============================] - 0s 6ms/step - loss: 0.6960 - accuracy: 0.8367 -
val_loss: 0.4322 - val_accuracy: 0.8459
Epoch 93/100
27/27 [==============================] - 0s 6ms/step - loss: 0.4926 - accuracy: 0.8373 -
val_loss: 0.4321 - val_accuracy: 0.8459
Epoch 94/100
27/27 [==============================] - 0s 6ms/step - loss: 0.7017 - accuracy: 0.8359 -
val_loss: 0.4320 - val_accuracy: 0.8459
Epoch 95/100
27/27 [==============================] - 0s 6ms/step - loss: 0.6071 - accuracy: 0.8370 -
val_loss: 0.4319 - val_accuracy: 0.8459
Epoch 96/100
27/27 [==============================] - 0s 6ms/step - loss: 0.7616 - accuracy: 0.8367 -
val_loss: 0.4318 - val_accuracy: 0.8459
Epoch 97/100
27/27 [==============================] - 0s 5ms/step - loss: 0.6693 - accuracy: 0.8371 -
val_loss: 0.4317 - val_accuracy: 0.8459
Epoch 98/100
27/27 [==============================] - 0s 7ms/step - loss: 0.5502 - accuracy: 0.8373 -
val_loss: 0.4316 - val_accuracy: 0.8459
Epoch 99/100
27/27 [==============================] - 0s 8ms/step - loss: 0.5104 - accuracy: 0.8376 -
val_loss: 0.4316 - val_accuracy: 0.8459
Epoch 100/100
27/27 [==============================] - 0s 7ms/step - loss: 0.4902 - accuracy: 0.8371 -
val_loss: 0.4315 - val_accuracy: 0.8459
```

Out[111…

```
<keras.callbacks.History at 0x15ab1d3b5b0>
```

In [112…

```python
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
dense_4 (Dense)             (None, 18)                342

dropout_3 (Dropout)         (None, 18)                0

dense_5 (Dense)             (None, 9)                 171

dropout_4 (Dropout)         (None, 9)                 0

dense_6 (Dense)             (None, 5)                 50

dropout_5 (Dropout)         (None, 5)                 0

dense_7 (Dense)             (None, 1)                 6

=================================================================
Total params: 569
Trainable params: 569
Non-trainable params: 0
_________________________________________________________________
```

Evaluating Model Performance

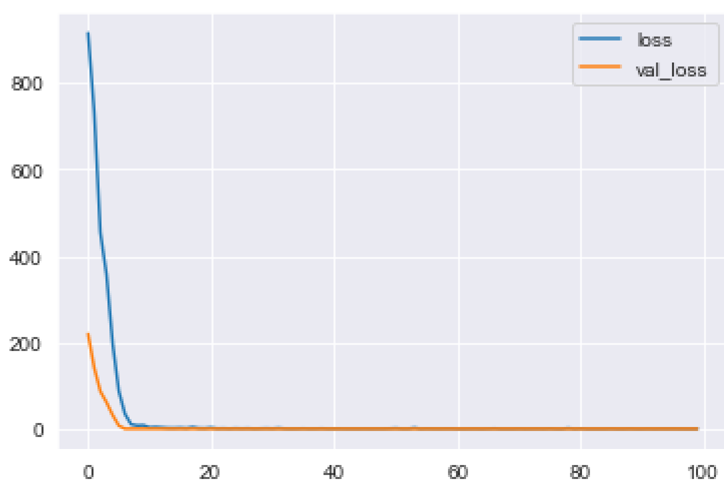
Plot out the validation loss versus the training loss.

In [115…

```
losses = pd.DataFrame(model.history.history)
losses[['loss','val_loss']].plot()
```

Out[115…

```
<AxesSubplot:>
```



Create predictions classification report and confusion metrix for X_test

In [119…

```
predictions = np.argmax(model.predict(X_test), axis=1)
```

In [122…

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2431
           1       0.00      0.00      0.00       443
```

```
    accuracy                           0.85      2874
   macro avg       0.42      0.50      0.46      2874
weighted avg       0.72      0.85      0.78      2874

C:\Users\Admin\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: Unde
finedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: Unde
finedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: Unde
finedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [123…

```
confusion_matrix(y_test,predictions)
```

Out[123…

```
array([[2431,    0],
       [ 443,    0]], dtype=int64)
```