DT and Random Forest Project

In this project, you will analyze publicly available loan data from www.LendingClub.com, a platform that connects borrowers (people seeking loans) with investors (people lending money). As an investor, your goal would be to fund loans that have a higher likelihood of being repaid. To assist with this, you will build a predictive model that can classify whether or not a borrower is likely to pay back their loan in full.

You'll focus on Lending Club's data from 2007 to 2010, which captures loan activity before the company went public. This dataset provides valuable insights into borrower behavior and loan outcomes, helping you develop a model that can predict repayment success.

For this analysis, you can use the pre-cleaned CSV file provided, which has already been processed to remove missing (NA) values.

Here are what the columns represent:

- 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).

Importing Libraries

(1 pt)

In [1]:

```

```

Reading the Data & Exploratory Data Analysis & Visualization

- (1 pt) Reading the data

In [2]:

```

```

- (6 pts) Extracting information about data

In [3]:

```

```

```
<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 [4]:

```

```

Out[4]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	day
count	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	
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	:
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	1

In [5]:

Out[5]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000

- (7 pts) Checking if there is Class imbalance

In [6]:

Out[6]:

```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

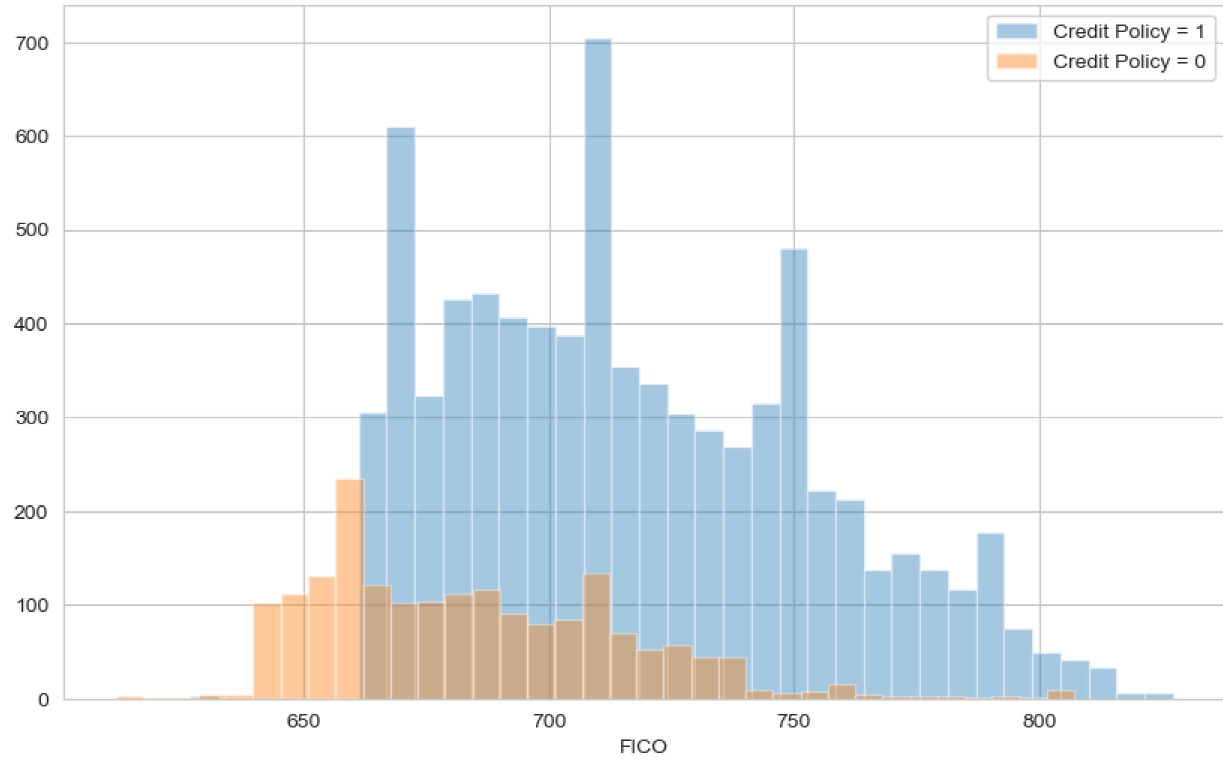
Let's do some data visualization! Don't worry about the colors matching, just worry about getting the main idea of the plot

- (5pts) Create a histogram of two FICO distributions on top of each other, one for each credit.policy outcome.

In [7]:

Out[7]:

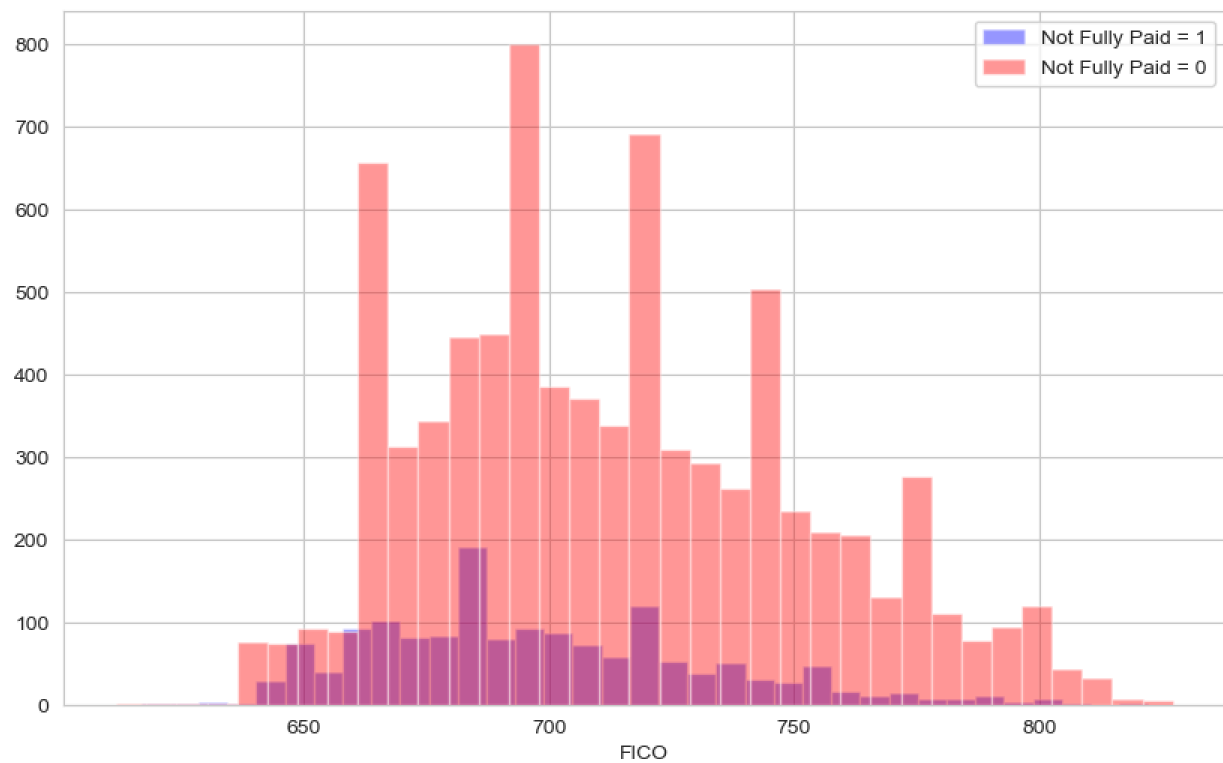
```
Text(0.5, 0, 'FICO')
```



- (5 pts) Create a similar figure, except this time select by the not.fully.paid column.

In [8]:

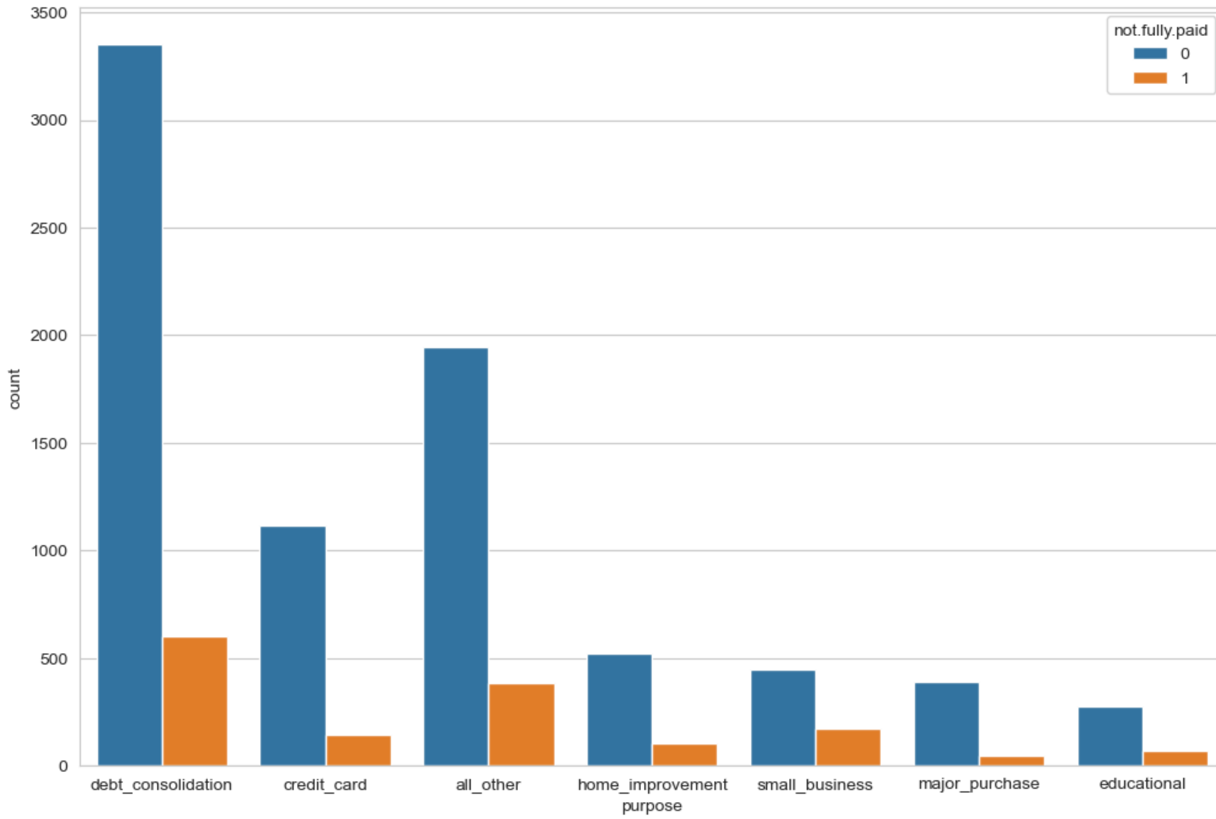
Out[8]: Text(0.5, 0, 'FICO')



- (5pts) Create a plot using seaborn showing the counts of loans by purpose, with the color hue defined by not.fully.paid.

In [9]:

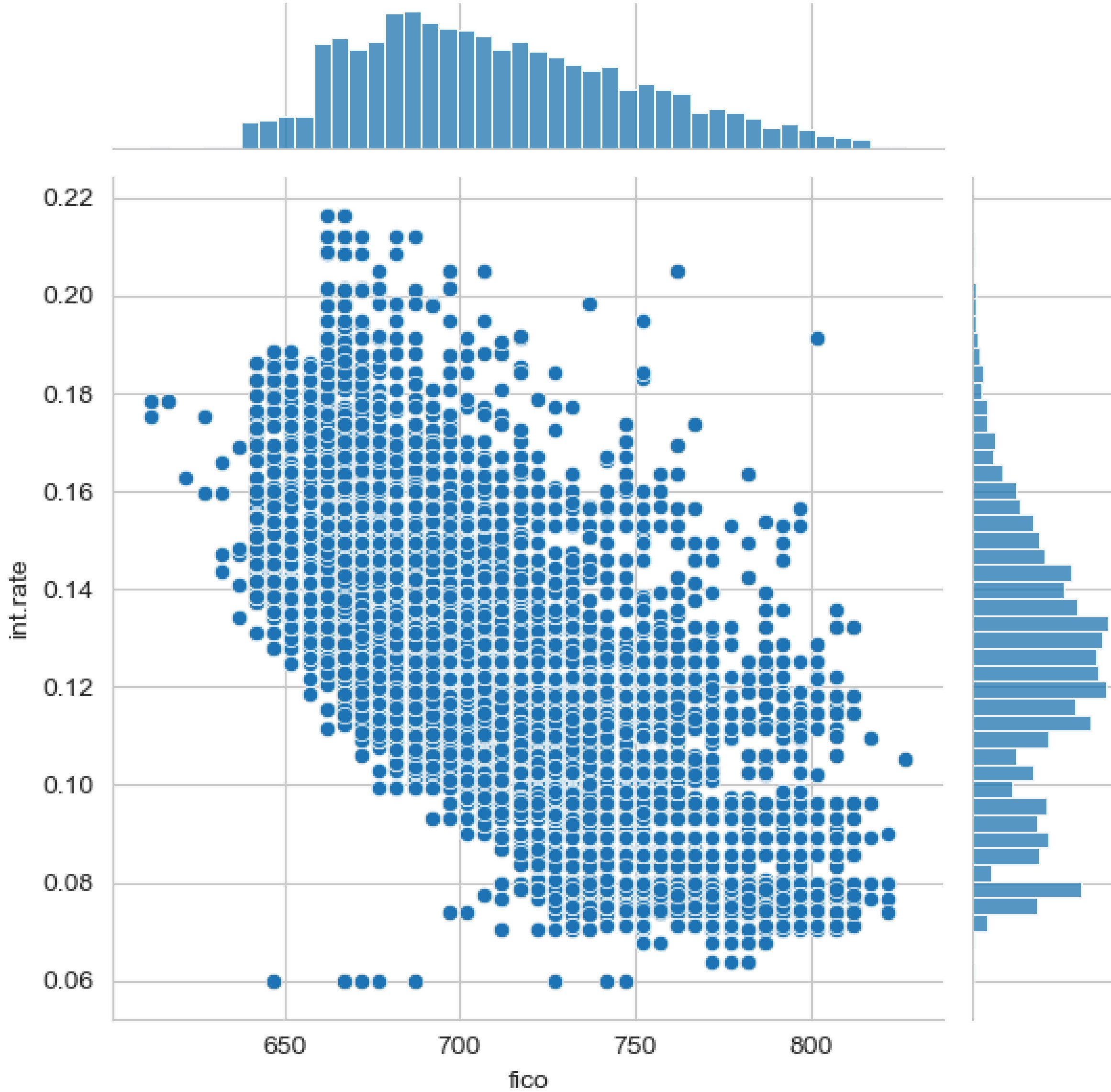
Out[9]: <AxesSubplot:xlabel='purpose', ylabel='count'>



- (5 pts) Let's see the trend between FICO score and interest rate. Recreate the following.

In [10]:

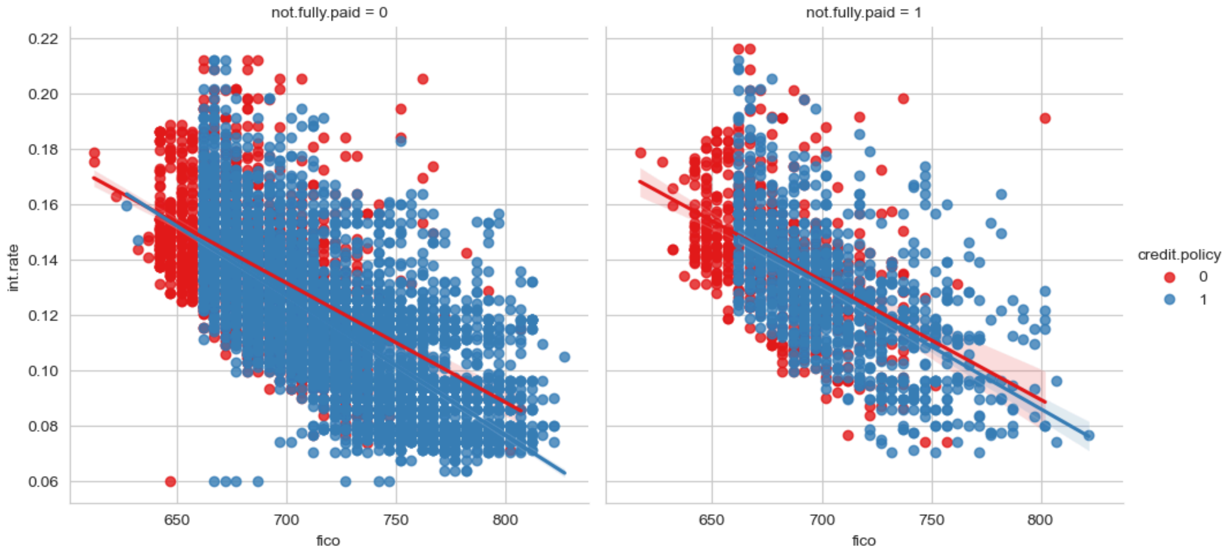
Out[10]: <seaborn.axisgrid.JointGrid at 0x7f994a164ca0>



- (5 pts) Create the following lmplots to see if the trend differed between not.fully.paid and credit.policy. Check the documentation for lmplot() if you can't figure out how to separate it into columns.

In [11]:

Out[11]:
```
<seaborn.axisgrid.FacetGrid at 0x7f994b1e44f0>
```



Setting up the Data

Let's set up our data for your Model!

Categorical Features

Notice that the **purpose** column as categorical.

That means we need to transform them so sklearn will be able to understand them.

A way of dealing with these columns that can be expanded to multiple categorical features if necessary.

- (10 pts) One way: Using OneHotEncoding and ColumnTrasnformer. You can use another method if you would like.

In [12]:

In [13]:

Out[13]:
```
array([1, 'debt_consolidation', 0.1189, 829.1, 11.35040654, 19.48, 737,
       5639.958333, 28854, 52.1, 0, 0, 0], dtype=object)
```

In [14]:

In [15]:

Out[15]:
```
array([0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1, 0.1189, 829.1, 11.35040654, 19.48,
       737, 5639.958333, 28854, 52.1, 0, 0, 0], dtype=object)
```

Train Test Split

Now its time to split our data into a training set and a testing set!

- (5 pts) Split your data into a training set and a testing set.

In [16]:

Training a Decision Tree Model

Let's start by training a single decision tree first!

- (5pts) Create an instance of Decision Tree Classifier and fit it to the training data.

In [17]:

In [18]:

Out[18]:

```
DecisionTreeClassifier()
```

Predictions and Evaluation of Decision Tree

- (10 pts) Create predictions from the test set and create a classification report and a confusion matrix.

In [19]:

In [20]:

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83      1999
           1       0.20      0.22      0.21       396

    accuracy                           0.72      2395
   macro avg       0.52      0.52      0.52      2395
weighted avg       0.74      0.72      0.73      2395


[[1638  361]
 [ 307   89]]
```

Training the Random Forest model

Now its time to train our model!

- (10 pts) Create an instance of the Random Forest Classifier class and fit it to your training data.

In [21]:

In [22]:

Out[22]:
```
RandomForestClassifier(n_estimators=600)
```

Predictions and Evaluation

- (10 pts) Let's predict off the y_test values and evaluate our model with creating a classification report from the results, and confusion matrix

In [23]:
```
```

In [24]:
```
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      1999
           1       0.33      0.01      0.02       396

    accuracy                           0.83      2395
   macro avg       0.58      0.50      0.46      2395
weighted avg       0.75      0.83      0.76      2395


[[1991    8]
 [ 392    4]]
```

- (10 pts) What performed better; the random forest or the decision tree?

In [25]:
```
```

BONUS: (10 pts)

We also noticed a class imbalance problem. Show your solution to that problem as bonus 10 pts.

Great Job!