

## Financial Probabilities:

**Worked with: Steven Y, Anthony C, Ryan C, Skyler M, Ji J**

For this activity, you will be responsible for modeling the risk of loans based on a series of attributes. To do this, you will need to clean a raw dataset, examine your data, and compute the risks associated with different features in your data. The following problems will walk you through this process.

1. Download the data in the Data/Financial Data folder on Canvas. Open the file and take a peek at the data dictionary. What do you think this data is used for?

*I believe this data is for showing a spread of people taking out loans and how they specifically pay them back and the demographics that surround that.*

2. Create a function called `loadAndCleanData` that takes as an argument a filename and returns a Pandas dataframe. That dataframe should contain the data from the CSV file cleaned such that any cells missing data, containing a NaN value or the string "NA" are filled with 0s (this is a technique called zero-filling that we will talk about shortly!)

```
def loadAndCleanData(FName):
```

```
    data = pd.read_csv(FName)
```

```
    NData = data.fillna("0")
```

```
    print(NData)
```

```
    return NData
```

3. Add a line to your Python file that uses the function to load in the creditData.csv file from Canvas when the Python script is run.

```
from utils import *
```

```
df = loadAndCleanData("creditData.csv")
```

4. Now that you've got your data loading, you can generate probability density functions for each feature. These PDFs will tell you the probability of a given feature occurring based on our data. You can use Kernel Density Estimation (KDE) to do this.

Write a function called `computePDF` that takes as arguments a target feature and a dataset and generates a KDE plot for each feature in your data (*hint: check out the `plot.kde` function [here \(Links to an external site.\)](#)*). You will need to import `matplotlib.pyplot` as `plt` and use `plt.show()` to make the graphs appear. Call that feature on each column of your dataset when you run your script.

```
def computePDF(columnName, DS):  
    NPlot = DS[columnName].plot.kde()  
    plt.show(NPlot)
```

5. Given the skews that you see in your data, you might want to step back and take a look at what's actually in your data. You can look at the distribution of values in the columns. This will help you understand what data you have. To do this, write a function called `viewDistribution` that takes in the name of a column and a dataframe and shows a histogram of values in that column (*hint: check out the `hist` function [here \(Links to an external site.\)](#)*). Comment out your `computePDF` function call and instead use `viewDistribution` to look at the distribution of each column in your dataset when the Python script is run. This should come after you call the `loadAndCleanData` function. Notice anything strange about some of these histograms?

```
def viewDistribution(CName, DS):  
    NPlot = dataSet.hist(column = CName)  
    plt.show(NPlot)
```

6. When your data distributions are radically skewed, you can use a log scale to help reveal data that is otherwise too sparse to see. Write a new version of the `viewDistribution` function called `viewLogDistribution` to show the log distribution of each column. Add this function call after your `viewDistribution` call to view the regular and log distributions of each feature.

```
def viewLogDistribution(CName, DS):
    NPlot = DS.hist(column = CName, log = True)
    plt.show(NPlot)
```

7. Use the two distributions to identify three bins per column that divide your data into roughly equal numbers. What are those bins? Note you do not need bins for "SeriousDlqin2yrs" as that is the feature you are modeling (it is your *dependent variable*).

```
def binsCount(CName, data):
    bins = pd.qcut(data[CName], q=3, duplicate = 'drop')
    print(bins[0])

CNames = list(cleanedData.columns.values)

for i in columnNames:
    newBins = bins(i, cleanedData)
```

8. Write a function called `computeDefaultRisk` that takes four arguments---a column name, a bin (as an array `[start,end]`), a target feature, and a dataframe---and returns the probability that someone will be at least 90 days delinquent on their account (in other words, "SeriousDlqin2yrs" = 1). Keep in mind that this probability is *conditional*, that means you'll want to use the equation for conditional probabilities to compute it. In plain English, you should compute the probability that a loan will become seriously delinquent given your target feature falls into the bin range. For example, if I'm looking at ages between 0 and 40, I want to compute the probability that a loan will go into serious delinquency given the applicant is between 0 and 40.

```
def computeDefaultRisk(column, TF, bin, DF):
    total = ''
    total2 = ''

    for i, datapoint in dataframe.iterrows():
```

```

        if datapoint[TF]>=bin[0] and datapoint[TF]<bin[1]:
            count+= 1
        if dataPoints[column]==1:
            count2 += 1

lengthData = len(data)
prob = total/lengthData
prob2 = total/lengthData
print(prob2 / prob)
return prob2 / prob

```


9. Print out the risk of default for each of the feature bins in your dataset. Note it's helpful to label these with the feature and bins such that you can better understand your output.

*ComputeDefaultRisk*

---

## Rating Schemes:

Problems 1-9 allow you to build a table of conditional probabilities for each feature in your dataset. These probabilities represent the risk that someone falling into each of these bins will go into default. In the next problems, you will use these probabilities to rate new loans. These ratings are used to assemble CDOs and other investments. As we saw in our reading, these ratings are based on risk and getting the risks correctly assessed is a key part of creating economically viable investment opportunities and of keeping financial companies afloat.

Download [newLoans.csv](#)  and put it in the same directory as your other data. Look at the data in Excel or some other program you use to view CSVs. You'll notice that the first column (SeriousDlqin2yrs) is empty. You're going to use your probabilities you've computed to estimate the risk of delinquency and put it in this column.

10. In your main file, use your `loadAndCleanData` function to load in newLoans.csv.

```
loadAndCleanData("newLoans.csv")
```

11. Use your conditional probabilities to predict the probability of default for each row in your CSV file. To do this, write a function called `predictDefaultRisk` that takes a row from your dataset as a parameter and returns the risk of default based on that data and the probabilities you computed from `creditData.csv` (*hint*: you might want to have `predictDefaultRisk` take a second parameter representing the risk of default for various data features computed from `creditData.csv`). You will want to compute the risk of default using a weighted sum with the following weights:

Age	2.5%
Number of Dependents	2.5%
Monthly Income	10%
Debt Ratio	10%
Revolving Utilization	10%
Number of Credit Lines & Loans	10%
Number of Real Estate Loans or Lines	10%
Number of Times 30-59 Days Late	15%
Number of Times 60-89 Days Late	15%
Number of Times 90 Days Late	15%

```
weights =  
{'age':0.025,'NumberOfDependents':0.025,'MonthlyIncome':0.1,'DebtRatio':0.1,'RevolvingUtilizationOfUnsecuredLines':0.1,'NumberOfOpenCreditLinesAndLoans':0.1,'NumberRealEstateLoansOrLines':0.1,'NumberOfTime30-59DaysPastDueNotWorse':0.15,'NumberOfTime60-89DaysPastDueNotWorse':0.15,'NumberOfTimes90DaysLate':0.15}
```

12. Store the result of this function in the `SeriousDlqin2yrs` column.

```
for row in range(len(newLoans.index)):  
    val = predictDefaultRisk(newLoans.iloc[[row]],risks,weights)  
    newLoans['SeriousDlqin2yrs'][row] = val
```

13. Plot the distribution of risks using your `computePDF` function. What do you notice about this distribution?

```
ComputePDF('SeriousDlqin2yrs',df)
```

**Update 2.20: Problems 14-18 will be extra credit problems worth 1 point each.**

14. Loan rating algorithms vary drastically and are largely trade secrets (meaning we don't have access to the specific scores they use). Instead, we'll create a few rating systems of our own. Add two columns to your dataframe containing your newLoans.csv data. One should be called AbsoluteRating and the second DistributedRating.

15. Create a function called `rateLoanAbsolute` that rates loan risks as A, B, C, D, or F using a typical grading scale (e.g., 0-10% risk of default = A, 11.1%-20% risk B, etc.). The function should take in a dataframe, compute these ratings, and store the ratings in the AbsoluteRating column.

15. Print out how many ratings fall in each category and the highest risk in each category. What do you notice about the distribution of ratings?

16. Create a function called `rateLoanDistribution` that groups loans into five equally sized categories based on their loan risks and assigns each group a rating (e.g., the lowest risk 20,000 loans are As, the next 20,000 are Bs, etc). The function should take in a dataframe, compute these ratings, and store the ratings in the DistributedRating column.

17. Print out how many ratings fall in each category and the highest risk in each category. How does this compare to the distribution of loans from your absolute ratings?

18. Take some time to explore these new ratings and think about their differences. When might you use the absolute rating scheme? When might you use the distributional rating scheme? You can add your response as a comment in your Python file labeled "# 18. ".