Intermediate Pandas and Visualizations

Investigating data

Assignment Contents

EXPECTED TIME 1.5 HRS

Overview

This assignment steps through an abreviated process of data exploration. This includes reading in data, looking for outliars / null values, and calculating statistics on the data and subsets of the data. In conclusion a demonstartion of plotting functionality w ill be offered. Plotting w ill not be tested. How ever, it is suggested you spend time w orking w ith the plotting demonstrations.

Activities in this Assignment

- Filtering by value
- Use `pandas` .groupby() method
- Use `.apply()`
- Calculate statistics

`Pandas` is built for data manipulation, thus the primary focus of this assignment is performing those manipulations and filterings.

Imports

```
### When starting any datascience work, these are farily standard imports

import pandas as pd ### standard to import pandas as "pd"
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

Reading in Data

`pandas` has a number of useful functions for reading data in from a file. For more esoteric file types, you should check out the `pandas` documentation, how ever, much of the time the tw o functions `.read_csv()` and `.read_table()` w ill fulfill your needs.

The file "part_acc.csv" is 1000 values from the **Lending Club** dataset (of funded loans), the full versions of w hich is available on kaggle.

```
acc_path = "../resource/asnlib/publicdata/part_acc.csv"


acc_df = pd.read_csv(acc_path, index_col = 0)

# Look at first row:
acc_df.head(1)
```

Tw o excellent functions for investigating new data are the `.describe()` and `.info()` methods. Both are called below. Study the output from both to answ er the follow ing few questions.

```
acc_df.info()
```

```
acc_df.describe(include="all").T
```

Question 1

```
### GRADED
### How many records (rows) are in `acc_df`?

### Assign int answer to ans1
### YOUR ANSWER BELOW

ans1 = 1000
```

Question 2

```
### GRADED
### How many null values (total) are in `acc_df`

### Assign int answer to ans1
### YOUR ANSWER BELOW
```

```
ans1 = 0
```

Question 3

```
### GRADED

### What are the mean and minimum amounts for 'loan_amnt'?

### Assign numeric answers to `mean_loan_amnt` and `min_loan_amnt` below
### YOUR ANSWER BELOW

mean_loan_amnt = 14216.2

min_loan_amnt = 1000
```

As demonstrated in lecture this w eek, this is a good place to start to think about w hether or not the values in the data make sense.

This data has been curated to reduce cleaning time, so there w ill not be the full "messiness" of the data-processing in this assignment.

Subsetting by Category

To start our analysis, w e w ill look at splitting our data by various categorical data.

All categorical variables described below

```
acc_df.describe(include = "object").T
```

Three of the potentially interesting categorical variables are "term", "home_ow nership", and "grade".

Below examples are performed w ith "term."

```
# Find unique values in term
print(".unique(); Notice the leading spaces;\n", acc_df['term'].unique(), sep = '')
print("\n.value_counts()\n", acc_df['term'].value_counts(), sep = "")
### Notice the leading spaces.
```

The most frequent "term" in this dataframe occurs 723 times.

We w ill be taking a look at the loans that are of the 36 month term. Below w e should the conditional that w e w ill use to subset the full DataFrame. Notice how it consists only of `True`s and `False`s (This is used to subset data in Lecture 9-4)

```
acc_df['term'] == ' 36 months'
```

Now below, using the logic demonstrated above, w e w ill subset the DataFrame, calling both `.info()`, and `.describe()` on the subsetted data for 'loan_amnt' -- Notice how the `.info()` call uses chained indexing, and the call to `.describe()` uses `.loc[]`

```
acc_df[acc_df['term'] == ' 36 months'][['loan_amnt']].info()
```

```
acc_df.loc[acc_df['term'] == ' 36 months', 'loan_amnt'].describe().T
```

Question 4

```
### GRADED
### The mean `loan_amnt` for just the 36 month loans
### 'a') Less
### `b`) More
### than the mean `loan_amnt` for all of the loans?

### Assign character associated with your choice as string to ans1
### YOUR ANSWER BELOW

ans1 = 'a'
```

Question 5

```
### GRADED

### How frequent is the most frequent "grade" in `acc_df`?

### Assign int to ans1
### YOUR ANSWER BELOW

ans1 = 292 # Use acc_df['grade'].value_counts()
```

Question 6

```
### GRADED
### What is the mean `loan_amnt` for loans of 'grade' == 'A'
### Assign number to ans1
### YOUR ANSWER BELOW

ans1 = acc_df[acc_df['grade'] == 'A']['loan_amnt'].mean()
```

Group By

Much of the lectures and much of this lesson focus on using `Panda`'s `.groupby()` function, and creating visualizations of the subsequent results.

First, the basic use and output of `.groupby` is examined.

The `.grouby()`s in lecture, due to the data being examined, all used `.size()` to aggregate the `.groupby()`. With this data, w e can also try other aggregations.

Example of using `.size()` show n below. Notice that after the end of the `.groupby()`, `['loan_amnt']` is specified. That means that all of the aggregations (`.size()` in this case) w ill be performed only w ith that feature. In your ow n w ork you should experiment w ith changing w hich features are specified both w ithin the `.groupby()` (Note: multiple features must be contained w ithin a list) and w hich features are specified follow ing the `.groupby()`.

```
acc_df.groupby(['term'])['loan_amnt'].size() ### Instead of `.size()`, could also use `.count()
```

The above should look familiar -- almost the exact same output as the call to `.value_counts()`.

How ever, instead of `.size()`, w hich simply counts the instances of a certain class, `.mean()` - and other `pandas` aggregators - can be used to aggregate a numeric variable:

```
acc_df.groupby('term')['loan_amnt'].mean()
```

Above, w e can see the mean loan amounts for loans w ith the term of 36 months and 60 months.

Question 7

```
### GRADED

### What is the mean loan amount for loans with a grade of "C"

### Assign number to ans1
### YOUR ANSWER BELOW

ans1 = acc_df.groupby('grade')['loan_amnt'].mean()['C']
```

The `.groupby()` function is also capable of grouping by mutliple variables. First, below, w e look at the mean of `'loan_amnt'` w hen categories w ithin `'term'` and `'home_ownership'` are grouped together

```
acc_df.groupby(['term', 'home_ownership'])['loan_amnt'].mean()
```

Question 8

```
### GRADED

### What is the mean loan amount for loans with a term of 36 months and a grade of "D"?

### Assign number to ans1
### YOUR ANSWER BELOW

ans1 = acc_df.groupby(['term','grade'])['loan_amnt'].mean()[(' 36 months', 'D')]
```

Question 9

```
### GRADED

### HOW MANY (count) loans in this dataset are of grade "A" where home_ownership is "OWN"

### Assign number to ans1
### YOUR ANSWER BELOW

ans1 = acc_df.groupby(['grade','home_ownership'])['loan_amnt'].count()[("A", "OWN")]
```

Custom Groupby

Before moving on to visualization, w e w ill lok at creating custom `.groupby()`'s. These are referenced in Lecture 9-10.

In the example below, interest rates (`'int_rate'`) w ill be divided into "high", "med", and "low " groups. You w ill be asked to do something similar w ith `'loan_amnt'`.

First, look at the distribution of interest rates to determine w hat "high", "med" and "low " should be.

```
acc_df['int_rate'].hist(bins = 30)
```

From the above, it looks as if a "low " interest rate is below 10%, and a "high" interest rate is above 15%, w ith "med(ium)" interest rates falling inbetw een.

The function w e need to pass to `.groupby()` should take one arguments; an index from the DataFrame.

The demonstration given in Lecture 9-10, converts a function of three arguments (DataFrame, index, column) to a function of a single argument (index) by using an anonymous `lambda` function.

HOWEVER, in lecture, `iloc[]` is used in the function instead of `loc[]`. `iloc[]` only works because the index **happens** to be the range from 0 to n-1. It is recommended you use `.loc[]` as shown below.

For the questions, you may use either the strategy from Lecture, or the strategy below

```
# Building function with one argument: index
def int_rate_groups(index):
    # Note how the column I want to group by ('int_rate') is always in the `.loc` brackets
    # Define what to return if interest rate less than 10
    if acc_df.loc[index,'int_rate'] <10:
        return "low"
    # Define what to return if interest rate greater than 15
    elif acc_df.loc[index,'int_rate'] >15:
        return "high"
    # Define what to return if interest rate neither greater than 15 nor less than 10
    else:
        return "med"

# Use function in a groupby
acc_df.groupby(int_rate_groups)['loan_amnt'].mean()
```

The above shows the mean loan amount for the three groups we defined

Question 10

```
### GRADED

### What is the mean interest rate for loans where the `loan_amnt` is:
### Greater than or equal to 15,000, AND less than or equal to 17,000

### There are many ways to answer this question. However, using a custom `groupby` function will be
### one of the easiest.

### Assign numeric answer to ans1
### YOUR ANSWER BELOW

def loan_amnt_groups(index):
    # Note how the column I want to group by ('int_rate') is always in the `.loc` brackets
    # Define what to return if interest rate less than 10
    if acc_df.loc[index,'loan_amnt'] >17000:
        return "hi"
    elif acc_df.loc[index, 'loan_amnt'] <15000:
        return "low"
    else:
        return "med"

ans1 = acc_df.groupby(loan_amnt_groups)['int_rate'].mean()['med']
```

Apply

Apply is a pandas functions that may be used to modify the values of a `Series` programatically. (Demonstrated in Lecture 9-3).

`.apply()` takes a function as an argument, returning the value of `<func>(value)`, for each value in the Series.

See the example below of modifying the `grade` column from `acc_df` using `.apply()`.

```
def grade_change(g):
    if g == "A":
        return 1
    elif g == "B":
        return 2
    elif g == "C":
        return 3
    elif g == "D":
        return 4
    else:
        return 0

print("Original Grades: \n",acc_df['grade'].head(),sep = "")
print("\n\nGrades after .apply():  \n", acc_df['grade'].apply(grade_change).head(), sep = "")
```

Question 11

Create a series that is full of the integers 36 and 60 corresponding to whether the loan has a term of 36 or 60 months. The series should have a length of 1,000.

For example, if instead of asking for a converstion of the terms to integers, the question asked for converstion of grades to integers -- as above -- the correct code would be: `acc_df['grad'].apply(grade_change)`

```
### GRADED
### See instructions above

### Assign series of length 1,000 to ans1
### YOUR ANSWER BELOW

def term_convert(t):
    if t == ' 36 months':
        return 36
    else:
        return 60

ans1 = acc_df['term'].apply(term_convert)
```

Visualizations

Unfortunately, visualizations cannot be graded on Vocareum. Thus this section is designed to complement the visualizations demonstrated in Lectures.

The main visualizations show n in lecture involve calling methods on `pandas` DataFrames. These can be very simple and pow erful tools for Exploratory Data Analysis.

```
# Create histogram
acc_df['loan_amnt'].hist()
```

```
# Create histogram another way
acc_df['loan_amnt'].plot(kind = 'hist')
```

```
# Increase "bins" in histogram
acc_df['loan_amnt'].hist(bins = 50)
```

```
# Not all plot types have their own method like histograms
# Also note that in order to create a bar-plot processing is needed
acc_df.groupby('grade')['loan_amnt'].count().plot(kind = "bar")
```

Some plot types require extra arguments -- see w ith "scatter" below

```
acc_df.plot(kind = 'scatter', x = 'loan_amnt', y = 'int_rate')
```

Note that the default plot is just a line plot of each variable -- This is the most frequent error people make w hen creating visualizations -- simply specifying plot and expecting a scatter plot instead of a line plot.

```
acc_df[['loan_amnt','int_rate']].plot()
```

As mentioned in lecture, all of the plotting done w ith `pandas` is actually done w ith the package `matplotlib`.

`matplotlib` w as used explicitly in lecture 9-11 to create the subplots. The conventional import for `matplotlib` is:

```
`import matplotlib.pyplot as plt`
```

The `pyplot` library exposes the majority of plotting functionality that you w ould use. Many of the `pandas` plotting can be called in a similar manner:

```
plt.hist(acc_df['loan_amnt'])
```

```
plt.scatter(acc_df['loan_amnt'], acc_df['int_rate'])
```

Tw o of the most important abilities of `matplotlib` are to change figure size, and create multiple subplots. Figure size is modified in Lecture 9-9 through lecture 9-12. Subplots are made in lecture 9-11.
First Figure size:

```
# figsize may be passed as an additional argument to `pandas` plotting methods
# figsize takes a tuple as input
acc_df['loan_amnt'].hist(figsize = (10,10))
```

```
acc_df['int_rate'].hist(bins = 20, figsize = (2,2))
```

Subplots are significantly more complicated to implement. The below code is commented to give a sense of w hat each line is accomplishing

```
### This is the data that will be plotted
acc_df.groupby(['home_ownership','term'])['loan_amnt'].mean().unstack()
```

```
row_num = 1 # Number of rows of plots to create
col_num = 2 # Number of columns of plots to create

data = acc_df.groupby(['home_ownership','term'])['loan_amnt'].mean().unstack()

### The Below code creates the figures and the axes for the plots.
### The figure can be ignored. The axes, are the areas in which the plots will be drawn
### The 'sharey' option specifies that the y-axis should be the same for all plots
fig, axes = plt.subplots(row_num, col_num, figsize = (10,5), sharey = True)

# Notice how `.flatten()` is called on the axes. By default, the axes come back as a 2-d array.
# `.flatten()` changes that to a 1-d array.
# The "data" will return the column names: 36 / 60 months
for ax, col in zip(axes.flatten(), data):
    series = data[col] # This pulls out the series from the data
    series.plot(kind = 'bar', ax= ax) # notice how the axis is specified in the arguments
    ax.set_title(col) # In order to set the title, a direct reference to the axis must be made
```