Rudra_Ronit_HW1_Practicum_Q2

September 21, 2016

1 Question

Load the auto-mpg sample dataset into Python using a Pandas dataframe. The horsepower feature has a few missing values with a ? - replace these with a NaN from NumPy, and calculate summary statistics for each numerical column. How do the summary statistics vary when excluding the NaNs, vs. imputing them with the mean (Hint: Use an Imputer from Scikit) - can we do better than just using the overall sample mean?

2 Solution

First, we import the necessary python modules for loading up the dataset, performing operations and visualization.

```
In [1]: %config IPCompleter.greedy=True

In [2]: %matplotlib inline

        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        plt.style.use('ggplot')
        from sklearn.preprocessing import Imputer
```

Now we import the dataset and store it in a variable. We use the function read_table() in pandas for this. The file extension is .tab which is a tab separated file.

```
In [3]: auto = pd.read_table('auto-mpg.tab',sep='\t')
```

After the file is loaded, let us look what the data is actually like.

```
In [4]: print("The dimensions of the data is:",auto.shape)
        print("This is what the column headers are:")
        print(auto.columns)
        print("First few instances of the dataset")
        auto.head(6)

The dimensions of the data is: (400, 9)
This is what the column headers are:
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'car_name'],
      dtype='object')
First few instances of the dataset
```

```
Out[4]:      mpg cylinders displacement horsepower weight acceleration model_year  \
        0      c         d            c          c      c            c          d
        1  class       NaN          NaN        NaN    NaN          NaN        NaN
        2     18         8          307        130   3504           12         70
        3     15         8          350        165   3693         11.5         70
        4     18         8          318        150   3436           11         70
        5     16         8          304        150   3433           12         70

          origin                   car_name
        0      d                          d
        1    NaN                        NaN
        2      1  chevrolet chevelle malibu
        3      1          buick skylark 320
        4      1         plymouth satellite
        5      1              amc rebel sst
```

The first two rows give us an idea of what type of data is contained in the data frame. Here, c means continuous and d is discrete. The 1st column, mpg is the class variable and others are the attributes. Hence, for our purposes, we are interested in Row index 2 to 400. Thus we have 398 instances.

Let us remove the first two rows as they are no longer required and would only mess up the indexing. We then reindex the entire dataset to start from 0 and end at 397.

```
In [5]: auto = auto[2:]
        auto.index = range(auto.shape[0])
        auto.head(4)
```

```
Out[5]:   mpg cylinders displacement horsepower weight acceleration model_year origin  \
        0  18         8          307        130   3504           12         70      1
        1  15         8          350        165   3693         11.5         70      1
        2  18         8          318        150   3436           11         70      1
        3  16         8          304        150   3433           12         70      1

                          car_name
        0  chevrolet chevelle malibu
        1          buick skylark 320
        2         plymouth satellite
        3              amc rebel sst
```

According to the question, the horsepower attribute has missing values. Let us find them by searching unique values in the column.

```
In [6]: pd.unique(auto['horsepower'])
```

```
Out[6]: array(['130', '165', '150', '140', '198', '220', '215', '225', '190',
               '170', '160', '95', '97', '85', '88', '46', '87', '90', '113',
               '200', '210', '193', '?', '100', '105', '175', '153', '180', '110',
               '72', '86', '70', '76', '65', '69', '60', '80', '54', '208', '155',
               '112', '92', '145', '137', '158', '167', '94', '107', '230', '49',
               '75', '91', '122', '67', '83', '78', '52', '61', '93', '148', '129',
               '96', '71', '98', '115', '53', '81', '79', '120', '152', '102',
               '108', '68', '58', '149', '89', '63', '48', '66', '139', '103',
               '125', '133', '138', '135', '142', '77', '62', '132', '84', '64',
               '74', '116', '82'], dtype=object)
```

We have some rows with a '?'. Let us find how many and which ones.

```
In [7]: print("Number of rows with a '?' under horsepower:",np.sum(auto['horsepower']=='?'))
        print("Displaying the rows with the missing values")
        auto.iloc[np.where(auto['horsepower']=='?')]
```

```
Number of rows with a '?' under horsepower: 6
Displaying the rows with the missing values
```

```
Out[7]:       mpg cylinders displacement horsepower weight acceleration model_year  \
        32     25         4           98          ?   2046           19         71
        126    21         6          200          ?   2875           17         74
        330  40.9         4           85          ?   1835         17.3         80
        336  23.6         4          140          ?   2905         14.3         80
        354  34.5         4          100          ?   2320         15.8         81
        374    23         4          151          ?   3035         20.5         82

            origin             car_name
        32       1            ford pinto
        126      1         ford maverick
        330      2  renault lecar deluxe
        336      1    ford mustang cobra
        354      2          renault 18i
        374      1        amc concord dl
```

Before we start computing summary statistics, we need to convert the columns to the proper datatype as the default is object. Therfore, converting the continuous variables to float, namely displacement; horsepower; weight and acceleration

```
In [8]: auto[['displacement','horsepower','weight',
              'acceleration']]=auto[['displacement','horsepower','weight',
                                     'acceleration']].apply(pd.to_numeric,
                                                            errors='coerce')
        auto.dtypes
```

```
Out[8]: mpg             object
        cylinders       object
        displacement   float64
        horsepower     float64
        weight           int64
        acceleration   float64
        model_year      object
        origin          object
        car_name        object
        dtype: object
```

Now we have the required columns as numeric data. Note that non-numeric values were coerced to NaN. Let us see the horsepower column.

```
In [9]: print("The '?' were replaced with NaN")
        auto.iloc[np.where(np.isnan(auto['horsepower']))]
```

```
The '?' were replaced with NaN
```

```
Out[9]:       mpg cylinders  displacement  horsepower  weight  acceleration  \
        32     25         4          98.0         NaN    2046          19.0
        126    21         6         200.0         NaN    2875          17.0
        330  40.9         4          85.0         NaN    1835          17.3
```

```
336  23.6         4         140.0         NaN    2905          14.3
354  34.5         4         100.0         NaN    2320          15.8
374    23         4         151.0         NaN    3035          20.5

    model_year origin             car_name
32          71      1            ford pinto
126         74      1         ford maverick
330         80      2  renault lecar deluxe
336         80      1    ford mustang cobra
354         81      2           renault 18i
374         82      1        amc concord dl
```

Let us find out the mean values of these variables. The function pd.mean can be used to do this. Note, it automatically ignores the NaN values. If we explicitly tell it to include NaN values then the mean would be NaN. This is shown below.

```
In [17]: print("Means without NaN",'\n',auto.mean(),'\n')
         print("Means with NaN",'\n',auto.mean(skipna=False),'\n')
         print("Summary Statistics",'\n')
         auto.describe()
```

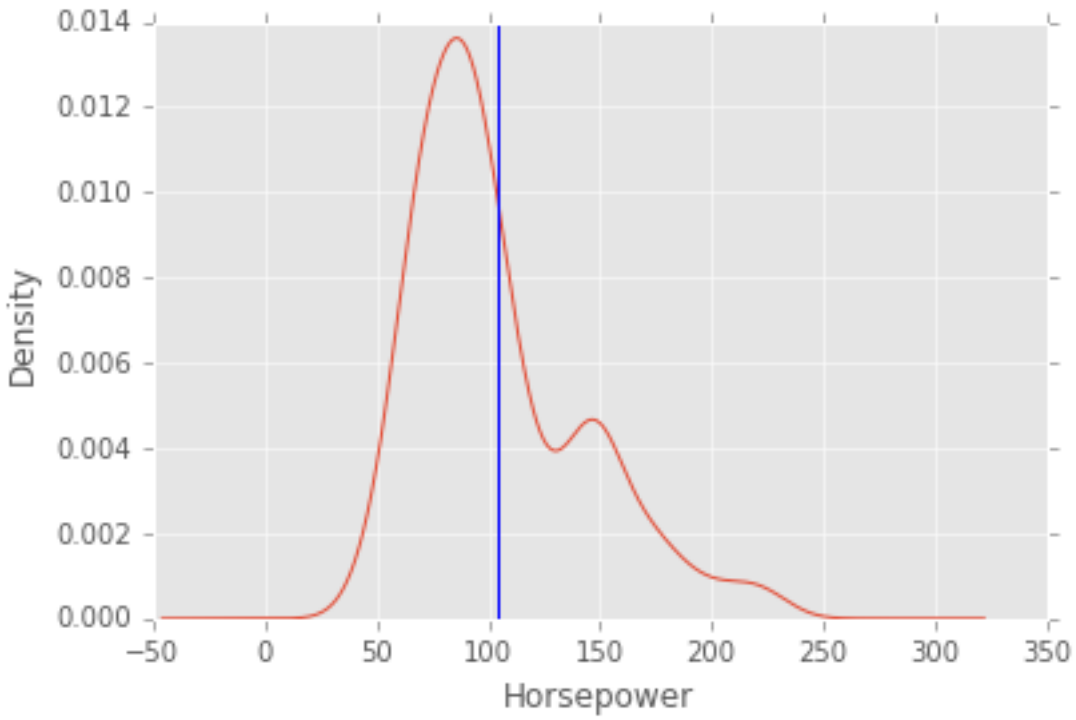
```
Means without NaN
 displacement     193.425879
horsepower       104.469388
weight          2970.424623
acceleration      15.568090
dtype: float64

Means with NaN
 displacement     193.425879
horsepower       104.469388
weight          2970.424623
acceleration      15.568090
dtype: float64

Summary Statistics
```

Out[17]:

	displacement	horsepower	weight	acceleration
count	398.000000	398.000000	398.000000	398.000000
mean	193.425879	104.469388	2970.424623	15.568090
std	104.269838	38.199187	846.841774	2.757689
min	68.000000	46.000000	1613.000000	8.000000
25%	104.250000	76.000000	2223.750000	13.825000
50%	148.500000	95.000000	2803.500000	15.500000
75%	262.000000	125.000000	3608.000000	17.175000
max	455.000000	230.000000	5140.000000	24.800000

Since other variables do not contain NaN values, we should focus on the horsepower variable and see how imputing missing values changes the distribution. Let us look at the original distribution.

```
In [11]: auto['horsepower'].plot.kde()
         plt.xlabel('Horsepower')
         plt.axvline(auto['horsepower'].mean())
         plt.show()
```



The blue line is the mean value of the horsepower. We shall impute the missing values from the mean and see how the distribution varies. Since we are experimenting with the imputation, let us store the horsepower column into another variable to avoid messing up the dataset and having to load it again.

```
In [12]: X = auto['horsepower'].reshape(-1,1)
```

```
In [13]: imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
         imp.fit(X)
         X = pd.DataFrame(imp.transform(X),columns=['horsepower'])
         print("Mean is:",'\n',X.mean()[0],'\n')
         print("Number of missing values:",np.sum(np.isnan(X['horsepower'])))
```

```
Mean is:
 104.469387755

Number of missing values: 0
```

The mean remains the same but all missing values have been imputed. Let us look at some other imputations, such as median and mode. For this we declare two other imputer objects, one for median and the other for mode. After fitting, we shall tabulate the difference between the three.

```
In [14]: Y = auto['horsepower'].reshape(-1,1)
         Z = auto['horsepower'].reshape(-1,1)
         imp_median = Imputer(missing_values='NaN', strategy='median', axis=0)
         imp_mode = Imputer(missing_values='NaN', strategy='most_frequent', axis=0)
         imp_median.fit(Y)
         imp_mode.fit(Z)
         Y = pd.DataFrame(imp_median.transform(Y),columns=['horsepower'])
         Z = pd.DataFrame(imp_mode.transform(Z),columns=['horsepower'])
```

```
stats = pd.DataFrame([[auto['horsepower'].mean(),auto['horsepower'].median(),
                       auto['horsepower'].mode().iloc[0]],
                      [X['horsepower'].mean(),X['horsepower'].median(),
                       X['horsepower'].mode().iloc[0]],
                      [Y['horsepower'].mean(),Y['horsepower'].median(),
                       Y['horsepower'].mode().iloc[0]],
                      [Z['horsepower'].mean(),Z['horsepower'].median(),
                       Z['horsepower'].mode().iloc[0]]],
                      columns=['Mean','Median',
                               'Mode'],index=["Original","Mean_impute",
                                              "Median_impute","Mode_impute"])
stats
```

Out[14]:

	Mean	Median	Mode
Original	104.469388	93.5	150.0
Mean_impute	104.469388	95.0	150.0
Median_impute	104.304020	93.5	150.0
Mode_impute	105.155779	95.0	150.0

The above table shows a quick summary of the horsepower column after different imputations. For the mean impute, the mean remains the same for imputed column and column excluding NaNs. This is because the missing values are replaced by the mean thus keeping the mean unchanged. For the meadian impute, we have a decimal value of 93.5 as there are an even number of observations and the median was the average of the middle two elements. Furthermore, since the imputed values were less than the original mean, the new mean was lower. For the Mode Impute, the mean went higher as the missing values were replaced by 150.

Below is the summary statistics for the four numerical columns after mean imputation of horsepower

In [15]:
```
auto['horsepower']=X
auto.describe()
```

Out[15]:

	displacement	horsepower	weight	acceleration
count	398.000000	398.000000	398.000000	398.000000
mean	193.425879	104.469388	2970.424623	15.568090
std	104.269838	38.199187	846.841774	2.757689
min	68.000000	46.000000	1613.000000	8.000000
25%	104.250000	76.000000	2223.750000	13.825000
50%	148.500000	95.000000	2803.500000	15.500000
75%	262.000000	125.000000	3608.000000	17.175000
max	455.000000	230.000000	5140.000000	24.800000

Other impute methods, apart from mean; median and mode, could include:

- weighted averages
- randomized selection
- downright removal of instances with missing values (provided it does not change bias the dataset)
- hot decking (impute missing data of an observation from another similar observation)
- cold decking (impute values from a previously recorded dataset)
- model based imputation