Big Data Lab – Week 7: Data Exploring and Pre-Processing

In this lab you will learn and practice how to explore data (or exploratory data analysis) and conduct a few data pre-processing in Python.

For this lab, the python libraries you need include:

- Numpy - the fundamental package for scientific computing with Python (https://numpy.org/)
- Pandas – offers data structures and operations for manipulating numerical tables and time series (https://pandas.pydata.org/pandas-docs/stable/)
- Matplotlib - a Python 2D plotting library (https://matplotlib.org/)
- Seaborn – a Python visualization library based on Matplotlib (https://seaborn.pydata.org/)

All these libraries are included in Anaconda 3 and Google Colab.

You can use Jupyter Notebook (which is the one utilized in task 1), Google Colab, Spyder IDE, Python shell, or any other tools you prefer to finish the lab exercises.

Task 1: Exploring and preparing the car dataset

One of the most important skills that every Data Scientist should master is the ability to explore data properly, which includes:

- Quickly describe a dataset with number of rows/columns, missing data, data types, etc.
- Clean corrupted data; handle missing data, invalid data types, incorrect values.
- Visualize data distributions using, e.g., bar charts, histograms, box plots, etc.
- Calculate and visualize correlations between variables using, e.g., heat map, scatter plot, etc.

In this task, you will explore a *car* dataset.

1. Download the data file “***cardata.csv***” from GCUlearn. New a Python script and import the required libraries for data exploring.

   If you are using the VM, do not forget to run the following command in a terminal window to active Python 3 before you start a python tool:

   ```
   source activate py37
   ```

If you are using Google Colab, save "*cardata.csv*" in your google drive. Do not forget to mount your google drive and change working directory. You can refer to "*Mount Google drive and change working directory in Colab.pdf*" on GCUlearn.

```
# Importing required libraries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt #visualisation
import seaborn as sns #visualisation

sns.set(color_codes=True)

# The following line is needed to set backend of matplotlib to inline
# to view visuals in Jupyter Notebook
# Comment it out if you are using Spyder or Google Colab
%matplotlib inline
```

2. Load the data into the pandas dataframe and check the number of rows and columns in the dataframe using the attribute *'.shape'*

```
df = pd.read_csv("cardata.csv")
print(df.shape)
```

Run this script, you will see output like below:

```
(11914, 12)
```

So there are 11914 rows, 12 columns in the *cardata* dataset.

3. Display a few top rows in the dataframe using *'.head'*. You can also display a few last rows using *'.tail'*.

```
# To display the top 5 rows
df.head(5)
```

Run this script, you will see the first 5 rows in the dataframe:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven Mode	Number of Doors	highway MPG	city mpg	Popularity	Price
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	26	19	3916	46135
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	28	19	3916	40650
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	28	20	3916	36350
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	28	18	3916	29450
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	28	18	3916	34500

4. Check the types of data. Sometimes we may have to convert variables saved as string or object in the dataframe to numerical data type, so that we can, for example, plot the data via a graph.

```
# Checking the data type
print(df.dtypes)
```

The execute result below shows the data type of each variable.

```
Make                  object
Model                 object
Year                   int64
Engine HP            float64
Engine Cylinders     float64
Transmission Type     object
Driven Mode           object
Number of Doors      float64
highway MPG            int64
city mpg               int64
Popularity             int64
Price                  int64
dtype: object
```

5. Check if there are any duplicate rows. If yes, drop them.

For a huge data set, as in this case contains more than 10,000 rows, it is quite common there are some duplicated data. So you need check this and remove the duplicate rows when necessary.

```
# Rows containing duplicate data
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

You will see there are some duplications in the *cardata*:

```
number of duplicate rows:  (886, 12)
```

Remove this 886 rows of duplicate data and then count the number of rows:

```
# Dropping the duplicates
df = df.drop_duplicates()

# Counting the number of rows after removing duplicates.
df.count()
```

You will see, only 11028 rows left in the dataset.

```
Make                 11028
Model                11028
Year                 11028
Engine HP            10959
Engine Cylinders     10998
Transmission Type    11028
Driven Mode          11028
Number of Doors      11022
highway MPG          11028
city mpg             11028
Popularity           11028
Price                11028
dtype: int64
```

6. Check if there are any missing values. If yes, dealing with them in a proper way.

```
# count the number of null values in each column
print(df.isnull().sum())
```

Run the script and you will get:

```
Make                 0
Model                0
Year                 0
Engine HP           69
Engine Cylinders    30
Transmission Type    0
Driven Mode          0
Number of Doors      6
highway MPG          0
city mpg             0
Popularity           0
Price                0
dtype: int64
```

It is shown there are about 100 missing values existed in three columns: '*Engine HP*' , '*Engine Cylinders*' and '*Number of Doors*'. The simplest strategy for handling missing data is to remove samples that contain a missing value. In '*cardata',* the percentage of samples with missing values is very low (less than 1%). So it is reasonable to use *dropna()* to remove all rows with missing data, as follows:

```
# Dropping the missing values.
df = df.dropna()
print(df.count())
print(df.shape)
```

Now, we have removed all the rows which contains the Null The output or N/A values, and there are 10929 rows left in the dataframe.

```
Make                 10929
Model                10929
Year                 10929
Engine HP            10929
Engine Cylinders     10929
Transmission Type    10929
Driven Mode          10929
Number of Doors      10929
highway MPG          10929
city mpg             10929
Popularity           10929
Price                10929
dtype: int64
(10929, 12)
```

7. Print summary statistics on attributes.

```python
# Printing summary statistics on attributes
print(df.describe())
```

For numeric data, the result's index includes *count*, *mean*, *std*, *min*, *max* as well as *lower, 50* and *upper percentiles*.

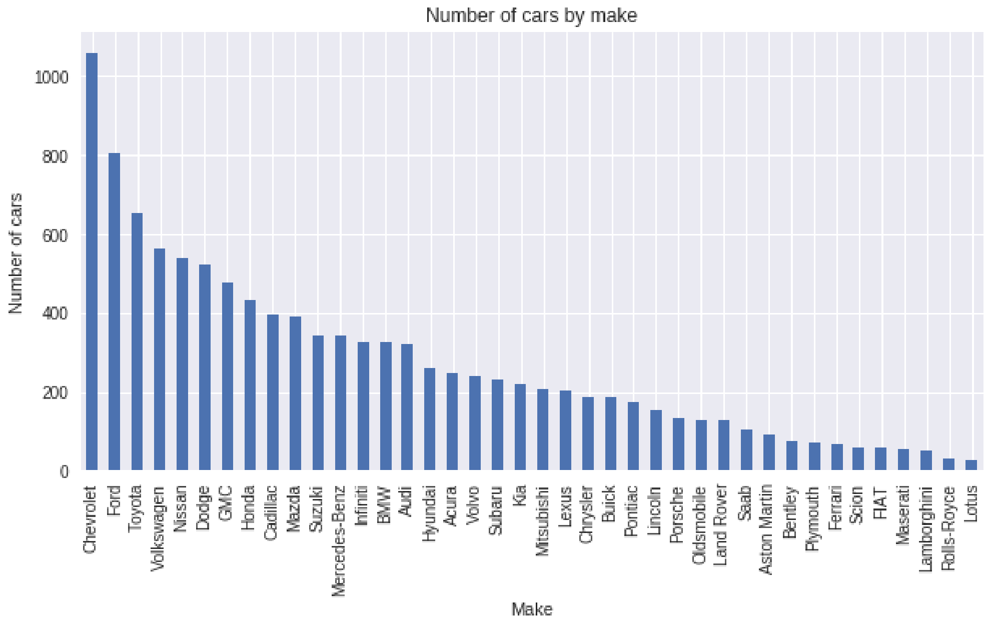
```
               Year     Engine HP  Engine Cylinders  Number of Doors  \
count  10929.000000  10929.000000      10929.000000     10929.000000
mean    2010.768780    253.367188          5.679477         3.449172
std        7.144636    109.969181          1.765286         0.875798
min     1990.000000     55.000000          0.000000         2.000000
25%     2007.000000    172.000000          4.000000         2.000000
50%     2015.000000    240.000000          6.000000         4.000000
75%     2016.000000    303.000000          6.000000         4.000000
max     2017.000000   1001.000000         16.000000         4.000000

        highway MPG      city mpg    Popularity         Price
count  10929.000000  10929.000000  10929.000000  1.092900e+04
mean      26.336719     19.346875   1557.566932  4.213557e+04
std        7.489187      6.625464   1448.307334  6.205717e+04
min       12.000000      7.000000      2.000000  2.000000e+03
25%       22.000000     16.000000    549.000000  2.169000e+04
50%       25.000000     18.000000   1385.000000  3.062000e+04
75%       30.000000     22.000000   2009.000000  4.310000e+04
max      354.000000    137.000000   5657.000000  2.065902e+06
```

8. Plot a bar chart for make variable

In the dataset used in this task, there are different types of car manufacturing companies, and it is often important to know who has the most number of cars. To do this bar chart is one of the trivial solutions which lets us know the total number of car manufactured by a different company.

```python
# Plotting a bar chart for make variable
df.Make.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title("Number of cars by make")
plt.ylabel('Number of cars')
plt.xlabel('Make');
```

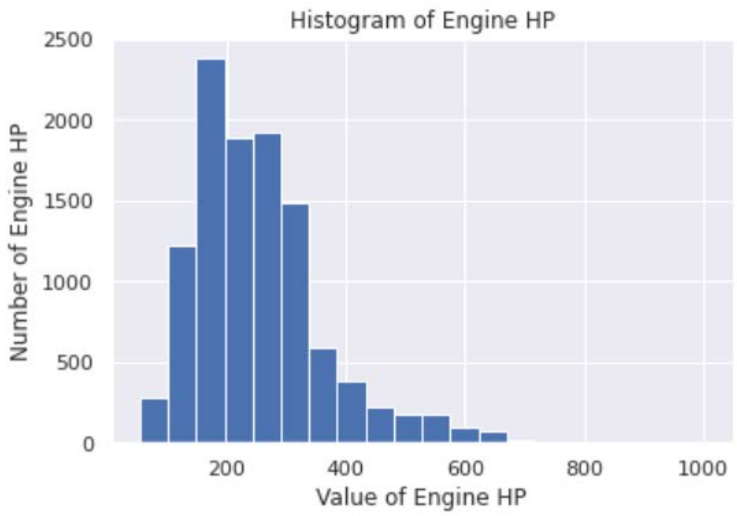
You should see a bar chart like:



9. Histogram refers to the frequency of occurrence of variables in an interval. Using the following script to plot the histogram of “Engine HP”

```
# plot the histogram of Engine HP
plt.hist(df['Engine HP'], bins=20)
plt.title("Histogram of Engine HP")
plt.ylabel('Number of Engine HP')
plt.xlabel('Value of Engine HP');
```

You should see a histogram like below. You can change the number of bins and see how it effect the histogram.

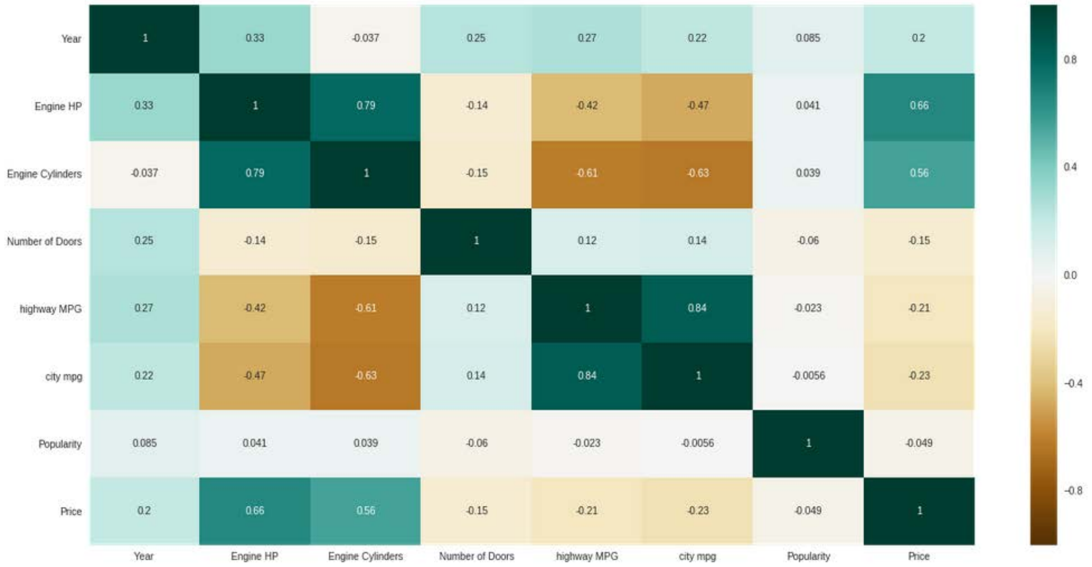


10. Calculate and visualize correlations using Seaborn heat map

Heat Maps plot rectangular data as a color-encoded matrix. It is one of the best way to find the relationship between the features. In the below heat map we know that the price feature depends mainly on the “*Engine HP*” and “*Engine Cylinders*”, as these two variables have highest absolute correlation coefficients (0.66 and 0.56 separately) with “*Price*”. Also, variables “*city mpg*” and “*highway MPG*” have the highest correlation coefficients (0.84).

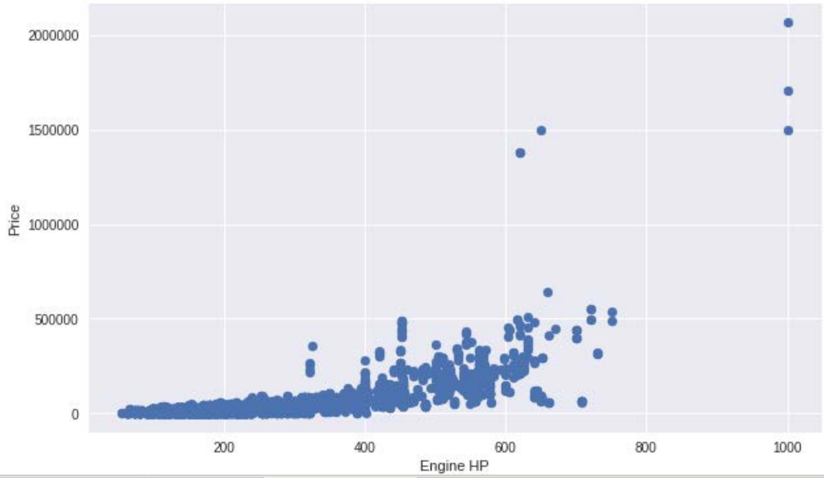
```
# Finding the relations between the variables.
plt.figure(figsize=(20,10))
corl= df.corr()
sns.heatmap(corl,cmap="BrBG",annot=True)
print(corl)
```

	Year	Engine HP	Engine Cylinders	Number of Doors	highway MPG	city mpg	Popularity	Price
Year	1.000000	0.327957	-0.037097	0.245177	0.272054	0.224110	0.085127	0.203604
Engine HP	0.327957	1.000000	0.788036	-0.136082	-0.421254	-0.473772	0.040507	0.660307
Engine Cylinders	-0.037097	0.788036	1.000000	-0.153036	-0.612161	-0.632701	0.039295	0.555170
Number of Doors	0.245177	-0.136082	-0.153036	1.000000	0.120556	0.140437	-0.060403	-0.148377
highway MPG	0.272054	-0.421254	-0.612161	0.120556	1.000000	0.841519	-0.022937	-0.209936
city mpg	0.224110	-0.473772	-0.632701	0.140437	0.841519	1.000000	-0.005554	-0.234567
Popularity	0.085127	0.040507	0.039295	-0.060403	-0.022937	-0.005554	1.000000	-0.048684
Price	0.203604	0.660307	0.555170	-0.148377	-0.209936	-0.234567	-0.048684	1.000000



11. Scatterplot is generally used to find the correlation between two variables. Here the scatter plots are plotted between “*Engine HP”* (Horsepower) and “*Price”* and we can see the plot below. With the plot given below, we can easily draw a trend line. These features provide a good scattering of points.

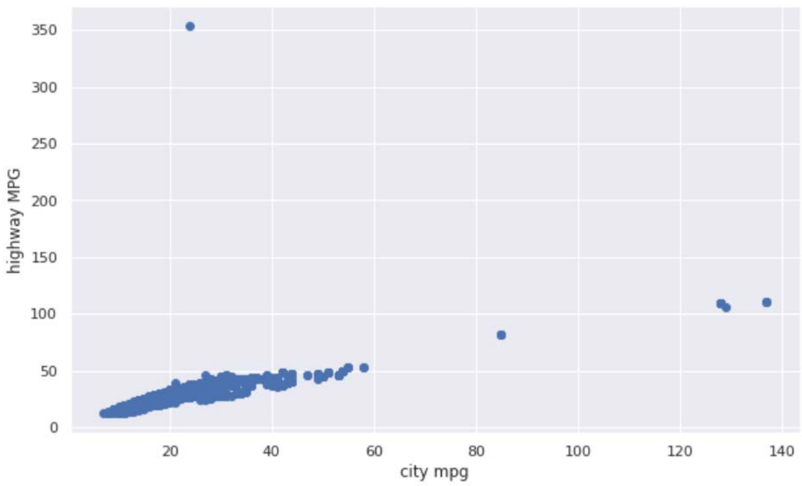
```
# Plotting a scatter plot
fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(df['Engine HP'], df['Price'])
ax.set_xlabel('Engine HP')
ax.set_ylabel('Price')
plt.show()
```



You can also using the following script to get the scatter plots between “city mpg” and “highway MPG”:

```
# Plotting a scatter plot
fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(df['city mpg'], df['highway MPG'])
ax.set_xlabel('city mpg')
ax.set_ylabel('highway MPG')
plt.show()
```

And you will see a scatter plot like below:



Task 2: Exploring and preparing the diabetes dataset

In this task, you should refer to what you did in task 1 to explore and prepare a diabetes dataset.

First, download the provided data file '*diabetes.csv*' and '*diabetes.names*' from GCUlearn.

You can open '***diabetes.names'*** (within an editor e.g. Notepad++), and go through it for general information about the data.

The variable names in the .csv data file are as follows:

- 0. Number of times pregnant.
- 1. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
- 2. Diastolic blood pressure (mm Hg).
- 3. Triceps skinfold thickness (mm).
- 4. 2-Hour serum insulin (mu U/ml).
- 5. Body mass index (weight in kg/(height in m)^2).
- 6. Diabetes pedigree function.
- 7. Age (years).
- 8. Class variable (0 or 1).

You should create a Python script file to finish exercises in this task.

1. Load the '*diabetes.csv*' dataset into Python as a Pandas DataFrame and check the number of rows and columns in the dataframe.

   (Hint: use the '*header* = *None*' parameter in the **read_csv** function, because column names are not saved in the data file)

2. Check the type of data. It should look like:

```
0      int64
1      int64
2      int64
3      int64
4      int64
5    float64
6    float64
7      int64
8      int64
dtype: object
```

3. Display the first 10 rows in the dataframe. You should get the output as shown below:

```
   0    1   2   3    4     5      6   7  8
0  6  148  72  35    0  33.6  0.627  50  1
1  1   85  66  29    0  26.6  0.351  31  0
2  8  183  64   0    0  23.3  0.672  32  1
3  1   89  66  23   94  28.1  0.167  21  0
4  0  137  40  35  168  43.1  2.288  33  1
5  5  116  74   0    0  25.6  0.201  30  0
6  3   78  50  32   88  31.0  0.248  26  1
7 10  115   0   0    0  35.3  0.134  29  0
8  2  197  70  45  543  30.5  0.158  53  1
9  8  125  96   0    0   0.0  0.232  54  1
```

4. Check if there are any duplicate rows

5. Print summary statistics on attributes

6. Dealing with missing value

   If you use the ***isnull*** function, you will notice there is no null values in this dataset. However, as shown in the printed summary statistics on attributes, there are columns with minimum value of zero. On some columns, a value of zero does not make sense and indicates an invalid or missing value. Specifically, it is invalid to have a zero minimum value in the following columns:

   1: Plasma glucose concentration
   2: Diastolic blood pressure
   3: Triceps skinfold thickness
   4: 2-Hour serum insulin
   5: Body mass index

   So we should treat zero values in those columns as invalid/missing values. You can get the number of missing values on each of these columns using:

```python
# count of the number of missing values on each of these columns
print((dataset[[1,2,3,4,5]] == 0).sum())
```

```
1      5
2     35
3    227
4    374
5     11
dtype: int64
```

   As you can see from the output, columns 1,2 and 5 have just a few zero values, whereas columns 3 and 4 have a lot more, nearly half of the rows.

   In Python, specifically Pandas, NumPy and Scikit-Learn, missing values are marked as *NaN*. *NaN* values are ignored from operations like sum, count, etc. Using the following scripts to mark the missing value in the dataset as *NaN*. You can print the first few rows again after the change.

```
# mark zero values as missing (with the value of NaN)
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, np.NaN)
# check the number of NaN values in each column
print(dataset.isnull().sum())
# print the first 10 rows of data
print(dataset.head(10))
```

The simplest strategy for handling missing data is to remove records that contain a missing value. You can use *dropna()* to remove all rows with missing data as in task 1. However, normally, this method is use only when the percentage of samples with missing values is low (i.e., less than 5%). If the missing value percentage is high, imputation could be a better option.

There are many options you could consider to impute a missing value. For example, impute with **mean** column values. Pandas provides the *fillna()* function for replacing missing values with a specific value. In the script below, we use *fillna()* to replace missing values with the mean value for each column:

```
# fill missing values with mean column values
dataset.fillna(dataset.mean(), inplace=True)
# check if there is still any NaN values in the dataset
print(dataset.isnull().sum())
# check the imputated the first 10 rows of data
print(dataset.head(10))
```

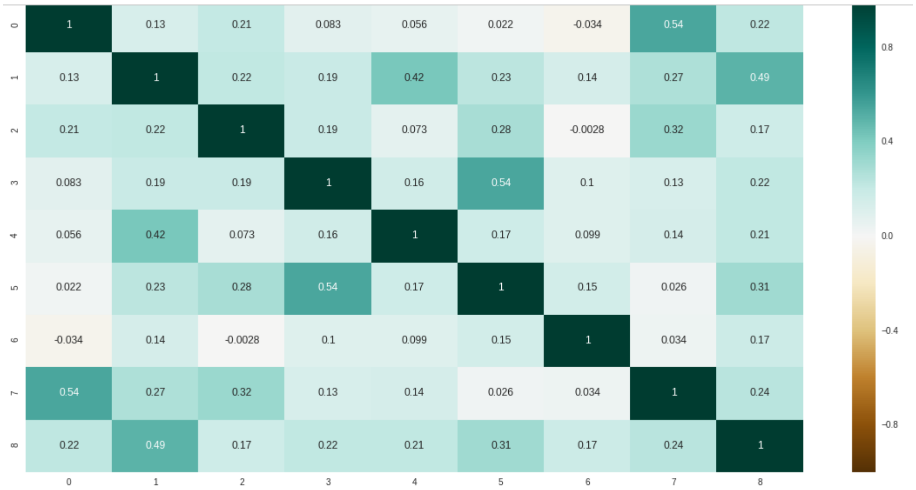
7. Plot the histograms

   Using the following script to plot the histogram of the second variable (Plasma glucose concentration, with index number of 1):

```
# plot the histogram of Plasma glucose concentration
plt.figure(0)
plt.hist(dataset[1], bins=20)
plt.title("Histogram of Plasma Glucose concentration")
plt.ylabel('Number of Plasma Glucose concentration')
plt.xlabel('Value of Plasma Glucose concentration');
```

   Write scripts to plot the histogram of 'Body mass index'.

8. Find the correlations between the variables and visualize them as a Heat map. You should get a heat map look like:



9. Scatter plot between Triceps skinfold thickness and Body mass index, and you should get a figure looks like: