- The process of converting or mapping data from the initial "raw" form into another format, to make it ready for further analysis.
- It is also known as Data Cleaning and Data Wrangling.

1. Identify, Evaluate and Count missing data
2. Deal with missing data
3. Correct the Data Format and Standardize the Data
4. Normalize the Data (centering/scaling)
5. Data Binning
6. Turn Categorical values into Numeric values

Roll No:- 22354 Name:- Malhar Choure

Performance Date: 30/3/2022

Submission date: 13/4/2022

- A -> To **create** cell **above**
- B -> To **create** Cell **below**
- D D -> For **deleting** the cell
- M -> To **markdown** the Cell
- Y -> For **code** the cell
- Z -> To **undo** the deleted cell

You can find the "Automobile Dataset" from the following link: https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data.

```
# Import the libraries pandas and matplotlib
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
```

First, we assign the URL of the dataset to "filename".

Note: This file does not have column headers, which we need to assign.

```
filename = 'https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data'
```

Then, we create a Python list headers containing name of headers.

```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style",
         "drive-wheels","engine-location","wheel-base", "length","width","height","curb-weight","engine-type",
         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepower",
         "peak-rpm","city-mpg","highway-mpg","price"]
```

Use the Pandas method read_csv() to load the data from the web address. Set the parameter "names" equal to the Python list "headers".

```
df = pd.read_csv(filename, names = headers)
```

Use the method head() to display the first five rows of the dataframe.

```
# To see what the data set looks like, we'll use the head() method.
df.head()
```

- Missing values occur when no data value is stored for a variable(feature) in an observation.
- Could be represented as ?, NA, 0 or just a blank cell.

Convert "?" to NaN

In the car dataset, missing data comes with the question mark "?". We replace "?" with NaN (Not a Number), Python's default missing value marker for reasons of computational speed and convenience. Here we use the function: dataframe.replace(A, B, inplace = True) to replace A by B.

```
# replace "?" to NaN

df.replace("?", np.nan, inplace = True)  # Question: explian the 
meaning of "inplace = True"
df.head(5)

   symboling normalized-losses         make fuel-type aspiration num-
of-doors  \
0          3               NaN  alfa-romero       gas        std       
two   
1          3               NaN  alfa-romero       gas        std       
two   
2          1               NaN  alfa-romero       gas        std       
two   
3          2               164         audi       gas        std      
four   
4          2               164         audi       gas        std      
four   

    body-style drive-wheels engine-location  wheel-base  ...  engine-
size  \
0  convertible          rwd           front        88.6  ...          
130   
1  convertible          rwd           front        88.6  ...          
130   
2    hatchback          rwd           front        94.5  ...          
152   
3        sedan          fwd           front        99.8  ...          
109   
4        sedan          4wd           front        99.4  ...          
```

```
136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm 
city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000 
21   
1         mpfi  3.47    2.68               9.0        111      5000 
21   
2         mpfi  2.68    3.47               9.0        154      5000 
19   
3         mpfi  3.19    3.40              10.0        102      5500 
24   
4         mpfi  3.19    3.40               8.0        115      5500 
18   

  highway-mpg  price  
0          27  13495  
1          27  16500  
2          26  16500  
3          30  13950  
4          22  17450  

[5 rows x 26 columns]
```

The missing values (NaN) are converted by default. We use the following functions to identify these missing values. There are two methods to detect missing data:

The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data.

```
missing_data = df.isnull()
missing_data.head(5)
```

```
   symboling  normalized-losses   make  fuel-type  aspiration  num-of-
doors  \
0      False               True  False      False       False      
False   
1      False               True  False      False       False      
False   
2      False               True  False      False       False      
False   
3      False              False  False      False       False      
False   
4      False              False  False      False       False      
False   

   body-style  drive-wheels  engine-location  wheel-base  ...  engine-
size  \
0       False         False            False       False  ...       
False
```

```
1       False         False             False       False  ...
False
2       False         False             False       False  ...
False
3       False         False             False       False  ...
False
4       False         False             False       False  ...
False

   fuel-system   bore  stroke  compression-ratio  horsepower  peak-rpm
\
0        False  False   False              False       False     False

1        False  False   False              False       False     False

2        False  False   False              False       False     False

3        False  False   False              False       False     False

4        False  False   False              False       False     False


   city-mpg  highway-mpg  price
0     False        False  False
1     False        False  False
2     False        False  False
3     False        False  False
4     False        False  False

[5 rows x 26 columns]
```

"True" means the value is a missing value while "False" means the value is not a missing value.

Using a for loop in Python, we can quickly figure out the number of missing values in each column. As mentioned above, "True" represents a missing value and "False" means the value is present in the dataset. In the body of the for loop the method ".value_counts()" counts the number of "True" values.

```
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
symboling
False    205
Name: symboling, dtype: int64

normalized-losses
False    164
```

```
True      41
Name: normalized-losses, dtype: int64

make
False    205
Name: make, dtype: int64

fuel-type
False    205
Name: fuel-type, dtype: int64

aspiration
False    205
Name: aspiration, dtype: int64

num-of-doors
False    203
True       2
Name: num-of-doors, dtype: int64

body-style
False    205
Name: body-style, dtype: int64

drive-wheels
False    205
Name: drive-wheels, dtype: int64

engine-location
False    205
Name: engine-location, dtype: int64

wheel-base
False    205
Name: wheel-base, dtype: int64

length
False    205
Name: length, dtype: int64

width
False    205
Name: width, dtype: int64

height
False    205
Name: height, dtype: int64

curb-weight
```

```
False    205
Name: curb-weight, dtype: int64

engine-type
False    205
Name: engine-type, dtype: int64

num-of-cylinders
False    205
Name: num-of-cylinders, dtype: int64

engine-size
False    205
Name: engine-size, dtype: int64

fuel-system
False    205
Name: fuel-system, dtype: int64

bore
False    201
True       4
Name: bore, dtype: int64

stroke
False    201
True       4
Name: stroke, dtype: int64

compression-ratio
False    205
Name: compression-ratio, dtype: int64

horsepower
False    203
True       2
Name: horsepower, dtype: int64

peak-rpm
False    203
True       2
Name: peak-rpm, dtype: int64

city-mpg
False    205
Name: city-mpg, dtype: int64

highway-mpg
False    205
```

```
Name: highway-mpg, dtype: int64

price
False    201
True       4
Name: price, dtype: int64
```

Based on the summary above, each column has 205 rows of data and seven of the columns containing missing data:

- **Check with the data collection source**
- **Replace the missing values**
  - replace it with an average (of similar data points)
  - replace it by frequency
  - replace it based on other functions
- **Drop the missing values**
  - drop the variable (column)
  - drop the data entry (row)
- **Leave it as missing data**

Use `dataframe.replace(missing_data, new_data)`

```
avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0)
print("Average of normalized-losses:", avg_norm_loss)
```

```
Average of normalized-losses: 122.0
```

```
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
```

```
df.head(5)
```

```
   symboling normalized-losses         make fuel-type aspiration num-
of-doors  \
0          3             122.0  alfa-romero       gas        std
two
1          3             122.0  alfa-romero       gas        std
two
2          1             122.0  alfa-romero       gas        std
two
3          2               164         audi       gas        std
four
4          2               164         audi       gas        std
four

    body-style drive-wheels engine-location  wheel-base  ...  engine-
size  \
0  convertible          rwd           front        88.6  ...
130
1  convertible          rwd           front        88.6  ...
```

```
130
2    hatchback          rwd           front        94.5  ...
152
3        sedan          fwd           front        99.8  ...
109
4        sedan          4wd           front        99.4  ...
136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm
city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000
21
1         mpfi  3.47    2.68               9.0        111      5000
21
2         mpfi  2.68    3.47               9.0        154      5000
19
3         mpfi  3.19    3.40              10.0        102      5500
24
4         mpfi  3.19    3.40               8.0        115      5500
18

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500
3          30  13950
4          22  17450

[5 rows x 26 columns]
```

```
avg_bore=df['bore'].astype('float').mean(axis=0)
print("Average of bore:", avg_bore)
```

```
Average of bore: 3.3297512437810957
```

```
df["bore"].replace(np.nan, avg_bore, inplace=True)
```

```
# Write your code below and press Shift+Enter to execute
#Calculate the mean vaule for "stroke" column
avg_stroke = df["stroke"].astype("float").mean(axis = 0)
print("Average of stroke:", avg_stroke)

# replace NaN by mean value in "stroke" column
df["stroke"].replace(np.nan, avg_stroke, inplace = True)
```

```
Average of stroke: 3.2554228855721337
```

```
avg_horsepower = df['horsepower'].astype('float').mean(axis=0)
print("Average horsepower:", avg_horsepower)
```

```
Average horsepower: 104.25615763546799
```

```
df['horsepower'].replace(np.nan, avg_horsepower, inplace=True)
```

```
avg_peakrpm=df['peak-rpm'].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peakrpm)
```

```
Average peak rpm: 5125.369458128079
```

```
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
```

To see which values are present in a particular column, we can use the ".value_counts()" method:

```
df['num-of-doors'].value_counts()
```

```
four    114
two      89
Name: num-of-doors, dtype: int64
```

We can see that four doors are the most common type. We can also use the ".idxmax()" method to calculate the most common type automatically:

```
df['num-of-doors'].value_counts().idxmax()
```

```
'four'
```

The replacement procedure is very similar to what we have seen previously:

```
#replace the missing 'num-of-doors' values by the most frequent
df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

```
df.head(10)
```

```
   symboling normalized-losses         make fuel-type aspiration num-
of-doors  \
0          3             122.0  alfa-romero       gas        std
two
1          3             122.0  alfa-romero       gas        std
two
2          1             122.0  alfa-romero       gas        std
two
3          2               164         audi       gas        std
four
4          2               164         audi       gas        std
four
5          2             122.0         audi       gas        std
two
6          1               158         audi       gas        std
four
7          1             122.0         audi       gas        std
four
8          1               158         audi       gas      turbo
four
9          0             122.0         audi       gas      turbo
```

```
two

    body-style drive-wheels engine-location  wheel-base  ...  engine-
size  \
0  convertible          rwd           front        88.6  ...
130
1  convertible          rwd           front        88.6  ...
130
2    hatchback          rwd           front        94.5  ...
152
3        sedan          fwd           front        99.8  ...
109
4        sedan          4wd           front        99.4  ...
136
5        sedan          fwd           front        99.8  ...
136
6        sedan          fwd           front       105.8  ...
136
7        wagon          fwd           front       105.8  ...
136
8        sedan          fwd           front       105.8  ...
131
9    hatchback          4wd           front        99.5  ...
131

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm
city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000
21
1         mpfi  3.47    2.68               9.0        111      5000
21
2         mpfi  2.68    3.47               9.0        154      5000
19
3         mpfi  3.19    3.40              10.0        102      5500
24
4         mpfi  3.19    3.40               8.0        115      5500
18
5         mpfi  3.19    3.40               8.5        110      5500
19
6         mpfi  3.19    3.40               8.5        110      5500
19
7         mpfi  3.19    3.40               8.5        110      5500
19
8         mpfi  3.13    3.40               8.3        140      5500
17
9         mpfi  3.13    3.40               7.0        160      5500
16

  highway-mpg  price
0          27  13495
```

```
1           27  16500
2           26  16500
3           30  13950
4           22  17450
5           25  15250
6           25  17710
7           25  18920
8           20  23875
9           22    NaN

[10 rows x 26 columns]
```

- Use `dataframe.dropna()`
  - `axis= 0` to drop the entire row
  - `axis= 1` to drop the entire column
- Whole columns should be dropped only if most entries in the column are empty. In our dataset, none of the columns are empty enough to drop entirely.
- Drop the whole row: "price": 4 missing data, simply delete the whole row Reason: price is what we want to predict in later experiment. Any data entry without price data cannot be used for prediction; therefore any row now without price data is not useful to us

```
# simply drop whole row with NaN in "price" column
df.dropna(subset=["price"], axis=0, inplace=True) # equivalent to: df
= df.dropna(subset= ['price'], axis= 0)

# reset index, because we droped two rows
df.reset_index(drop=True, inplace=True)

print(df)

     symboling normalized-losses         make fuel-type aspiration  \
0            3             122.0  alfa-romero       gas        std
1            3             122.0  alfa-romero       gas        std
2            1             122.0  alfa-romero       gas        std
3            2               164         audi       gas        std
4            2               164         audi       gas        std
..         ...               ...          ...       ...        ...
196         -1                95        volvo       gas        std
197         -1                95        volvo       gas      turbo
198         -1                95        volvo       gas        std
199         -1                95        volvo    diesel      turbo
200         -1                95        volvo       gas      turbo

    num-of-doors   body-style drive-wheels engine-location  wheel-base
...  \
0            two  convertible          rwd           front        88.6
...
```

```
1            two  convertible          rwd           front        88.6
...
2            two    hatchback          rwd           front        94.5
...
3           four        sedan          fwd           front        99.8
...
4           four        sedan          4wd           front        99.4
...
..           ...          ...          ...             ...         ...
...
196         four        sedan          rwd           front       109.1
...
197         four        sedan          rwd           front       109.1
...
198         four        sedan          rwd           front       109.1
...
199         four        sedan          rwd           front       109.1
...
200         four        sedan          rwd           front       109.1
...

     engine-size  fuel-system  bore  stroke compression-ratio
horsepower  \
0            130         mpfi  3.47    2.68               9.0
111
1            130         mpfi  3.47    2.68               9.0
111
2            152         mpfi  2.68    3.47               9.0
154
3            109         mpfi  3.19    3.40              10.0
102
4            136         mpfi  3.19    3.40               8.0
115
..           ...          ...   ...     ...               ...         .
..
196          141         mpfi  3.78    3.15               9.5
114
197          141         mpfi  3.78    3.15               8.7
160
198          173         mpfi  3.58    2.87               8.8
134
199          145          idi  3.01    3.40              23.0
106
200          141         mpfi  3.78    3.15               9.5
114

     peak-rpm city-mpg highway-mpg  price
0        5000       21          27  13495
1        5000       21          27  16500
2        5000       19          26  16500
```

```
3         5500       24          30  13950
4         5500       18          22  17450
..         ...      ...         ...    ...
196       5400       23          28  16845
197       5300       19          25  19045
198       5500       18          23  21485
199       4800       26          27  22470
200       5400       19          25  22625

[201 rows x 26 columns]
```

Good! Now, we have a dataset with no missing values.

In this section, we will look at the problem of data with different formats, units and conventions and the pandas methods that help us deal with these issues.

- Data are generally collected from different places and stored in different formats.
- Data formatting and standardization: Bringing (transforming) data into a common standard of expression allow users to make meaningful comparision.
- As a part of data cleaning, formatting ensures the data is consistent and easily understandable.

Steps for Data formating and standardization

- Correcting the incorrect data types (Data Formatting)
- Applying calculation to an entire column (Data Standardization)

In Pandas, we use:

```
df.dtypes
```

```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
```

```
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

```
df
```

```
     symboling normalized-losses         make fuel-type aspiration  \
0            3             122.0  alfa-romero       gas        std
1            3             122.0  alfa-romero       gas        std
2            1             122.0  alfa-romero       gas        std
3            2               164         audi       gas        std
4            2               164         audi       gas        std
..         ...               ...          ...       ...        ...
196         -1                95        volvo       gas        std
197         -1                95        volvo       gas      turbo
198         -1                95        volvo       gas        std
199         -1                95        volvo    diesel      turbo
200         -1                95        volvo       gas      turbo

    num-of-doors   body-style drive-wheels engine-location  wheel-base
...  \
0            two  convertible          rwd           front        88.6
...
1            two  convertible          rwd           front        88.6
...
2            two    hatchback          rwd           front        94.5
...
3           four        sedan          fwd           front        99.8
...
4           four        sedan          4wd           front        99.4
...
..           ...          ...          ...             ...         ...
...
196         four        sedan          rwd           front       109.1
...
197         four        sedan          rwd           front       109.1
...
198         four        sedan          rwd           front       109.1
...
199         four        sedan          rwd           front       109.1
...
200         four        sedan          rwd           front       109.1
...

     engine-size  fuel-system  bore  stroke compression-ratio
horsepower  \
```

```
0             130         mpfi  3.47    2.68               9.0
111
1             130         mpfi  3.47    2.68               9.0
111
2             152         mpfi  2.68    3.47               9.0
154
3             109         mpfi  3.19    3.40              10.0
102
4             136         mpfi  3.19    3.40               8.0
115
..            ...          ...   ...     ...               ...          .
..
196           141         mpfi  3.78    3.15               9.5
114
197           141         mpfi  3.78    3.15               8.7
160
198           173         mpfi  3.58    2.87               8.8
134
199           145          idi  3.01    3.40              23.0
106
200           141         mpfi  3.78    3.15               9.5
114

     peak-rpm city-mpg highway-mpg  price
0        5000       21          27  13495
1        5000       21          27  16500
2        5000       19          26  16500
3        5500       24          30  13950
4        5500       18          22  17450
..        ...      ...         ...    ...
196      5400       23          28  16845
197      5300       19          25  19045
198      5500       18          23  21485
199      4800       26          27  22470
200      5400       19          25  22625

[201 rows x 26 columns]
```

```
df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
df[["price"]] = df[["price"]].astype("float")
df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
```

```
df.dtypes
```

```
symboling              int64
normalized-losses      int32
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
```

```
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower            object
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
dtype: object
```

Wonderful!

Now we have finally obtained the cleaned dataset with no missing values with all data in its proper format.

Example

The formula for unit conversion is: L/100km = 235 / mpg We can do many mathematical operations directly in Pandas.

```
df.head()
```

```
   symboling  normalized-losses         make fuel-type aspiration  \
0          3                122  alfa-romero       gas        std
1          3                122  alfa-romero       gas        std
2          1                122  alfa-romero       gas        std
3          2                164         audi       gas        std
4          2                164         audi       gas        std

  num-of-doors   body-style drive-wheels engine-location  wheel-
base  ...  \
0          two  convertible          rwd           front
88.6  ...
1          two  convertible          rwd           front
88.6  ...
2          two    hatchback          rwd           front
94.5  ...
3         four        sedan          fwd           front
99.8  ...
```

```
4         four        sedan          4wd           front
99.4  ...

   engine-size  fuel-system  bore  stroke compression-ratio horsepower
\
0          130         mpfi  3.47    2.68               9.0        111

1          130         mpfi  3.47    2.68               9.0        111

2          152         mpfi  2.68    3.47               9.0        154

3          109         mpfi  3.19    3.40              10.0        102

4          136         mpfi  3.19    3.40               8.0        115


   peak-rpm city-mpg  highway-mpg    price
0    5000.0       21           27  13495.0
1    5000.0       21           27  16500.0
2    5000.0       19           26  16500.0
3    5500.0       24           30  13950.0
4    5500.0       18           22  17450.0

[5 rows x 26 columns]
```

```
# Convert mpg to L/100km by mathematical operation (235 divided by
mpg)
df['city-L/100km'] = 235/df["city-mpg"] # This will create a new
column "city-L/100km"

# check your transformed data
df.head()
```

```
   symboling  normalized-losses         make fuel-type aspiration  \
0          3                122  alfa-romero       gas        std
1          3                122  alfa-romero       gas        std
2          1                122  alfa-romero       gas        std
3          2                164         audi       gas        std
4          2                164         audi       gas        std

  num-of-doors   body-style drive-wheels engine-location  wheel-
base  ...  \
0          two  convertible          rwd           front
88.6  ...
1          two  convertible          rwd           front
88.6  ...
2          two    hatchback          rwd           front
94.5  ...
3         four        sedan          fwd           front
99.8  ...
```

```
4          four        sedan          4wd           front
99.4  ...

   fuel-system  bore  stroke  compression-ratio horsepower peak-rpm
city-mpg  \
0          mpfi  3.47    2.68                9.0        111   5000.0
21
1          mpfi  3.47    2.68                9.0        111   5000.0
21
2          mpfi  2.68    3.47                9.0        154   5000.0
19
3          mpfi  3.19    3.40               10.0        102   5500.0
24
4          mpfi  3.19    3.40                8.0        115   5500.0
18

  highway-mpg    price  city-L/100km
0          27  13495.0     11.190476
1          27  16500.0     11.190476
2          26  16500.0     12.368421
3          30  13950.0      9.791667
4          22  17450.0     13.055556

[5 rows x 27 columns]
```

```python
# Write your code below and press Shift+Enter to execute
df["highway-L/100km"]=235/df["highway-mpg"]
df.head(10)
```

```
   symboling  normalized-losses         make fuel-type aspiration  \
0          3                122  alfa-romero       gas        std
1          3                122  alfa-romero       gas        std
2          1                122  alfa-romero       gas        std
3          2                164         audi       gas        std
4          2                164         audi       gas        std
5          2                122         audi       gas        std
6          1                158         audi       gas        std
7          1                122         audi       gas        std
8          1                158         audi       gas      turbo
9          2                192          bmw       gas        std

  num-of-doors   body-style drive-wheels engine-location  wheel-
base  ...  \
0          two  convertible          rwd           front
88.6  ...
1          two  convertible          rwd           front
88.6  ...
2          two    hatchback          rwd           front
94.5  ...
3         four        sedan          fwd           front
```

```
99.8  ...
4        four       sedan         4wd          front
99.4  ...
5         two       sedan         fwd          front
99.8  ...
6        four       sedan         fwd          front
105.8  ...
7        four       wagon         fwd          front
105.8  ...
8        four       sedan         fwd          front
105.8  ...
9         two       sedan         rwd          front
101.2  ...

   bore  stroke  compression-ratio  horsepower peak-rpm city-mpg
highway-mpg  \
0  3.47    2.68                9.0         111   5000.0       21
27
1  3.47    2.68                9.0         111   5000.0       21
27
2  2.68    3.47                9.0         154   5000.0       19
26
3  3.19    3.40               10.0         102   5500.0       24
30
4  3.19    3.40                8.0         115   5500.0       18
22
5  3.19    3.40                8.5         110   5500.0       19
25
6  3.19    3.40                8.5         110   5500.0       19
25
7  3.19    3.40                8.5         110   5500.0       19
25
8  3.13    3.40                8.3         140   5500.0       17
20
9  3.50    2.80                8.8         101   5800.0       23
29

     price  city-L/100km  highway-L/100km
0  13495.0     11.190476         8.703704
1  16500.0     11.190476         8.703704
2  16500.0     12.368421         9.038462
3  13950.0      9.791667         7.833333
4  17450.0     13.055556        10.681818
5  15250.0     12.368421         9.400000
6  17710.0     12.368421         9.400000
7  18920.0     12.368421         9.400000
8  23875.0     13.823529        11.750000
9  16430.0     10.217391         8.103448

[10 rows x 28 columns]
```

Example

Few Methods of normalizing data

1. **Simple feature scaling:** $x_{new} = \frac{x_{old}}{x_{max}}$

2. **Min-Max:** $x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$

3. **Z-score:** $x_{new} = \frac{x_{old} - \mu}{\sigma}$ where μ is the mean and σ is the standard deviation of the feature.

```
df['length']
```

```
0      168.8
1      168.8
2      171.2
3      176.6
4      176.6
       ...
196    188.8
197    188.8
198    188.8
199    188.8
200    188.8
Name: length, Length: 201, dtype: float64
```

5.1 Simple feature scaling

```
# replace (original value) by (original value)/(maximum value)
df['length'] = df['length']/df['length'].max()
df['width'] = df['width']/df['width'].max()
```

```
# Write your code below and press Shift+Enter to execute
df['height']=df['height']/df['height'].max()
print(df['height'].max())
print(df['height'])
```

```
1.0
0      0.816054
1      0.816054
2      0.876254
3      0.908027
4      0.908027
         ...
196    0.928094
197    0.928094
198    0.928094
199    0.928094
```

```
200    0.928094
Name: height, Length: 201, dtype: float64
```

Here we can see we've normalized "length", "width" and "height" in the range of [0,1].

- **Binning:** Grouping of **values** into **bins** for grouped analysis.
  - Example: we can bin "age" into [0, 5], [6, 10], [11, 15] and so on.
- Converts **numeric** into **categorical** variables.
- Group a **set of numerical values** into a **set of bins**.

Example:

Convert data to correct format:

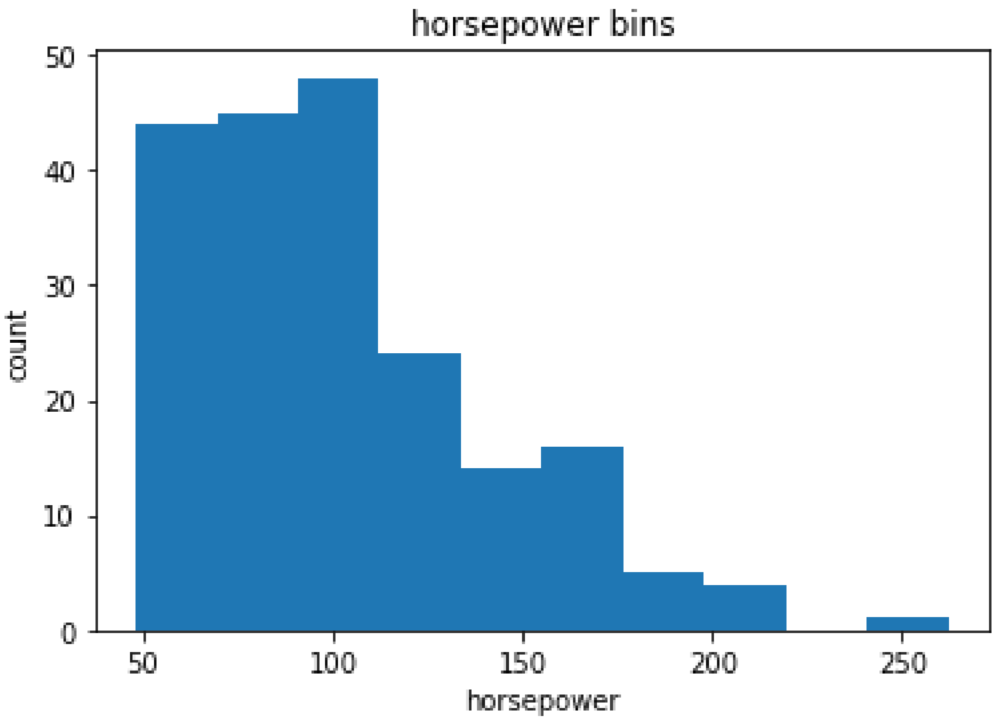
```
df["horsepower"]=df["horsepower"].astype(int, copy=True)
```

Let's plot the histogram of horsepower to see what the distribution of horsepower looks like.

```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["horsepower"])

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```



We build a bin array with a minimum value to a maximum value by using the bandwidth calculated above. The values will determine when one bin ends and another begins.

```
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
bins
```

```
array([ 48.        , 119.33333333, 190.66666667, 262.        ])
```

We set group names:

```
group_names = ['Low', 'Medium', 'High']
```

We apply the function "cut" to determine what each value of `df['horsepower']` belongs to.

```
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, 
labels=group_names, include_lowest=True )
df[['horsepower','horsepower-binned']].head(20)
```

```
    horsepower horsepower-binned
0          111               Low
1          111               Low
2          154            Medium
3          102               Low
4          115               Low
5          110               Low
6          110               Low
7          110               Low
8          140            Medium
```

```
9          101             Low
10         101             Low
11         121          Medium
12         121          Medium
13         121          Medium
14         182          Medium
15         182          Medium
16         182          Medium
17          48             Low
18          70             Low
19          70             Low
```

Let's see the number of vehicles in each bin:

```
df["horsepower-binned"].value_counts()
```

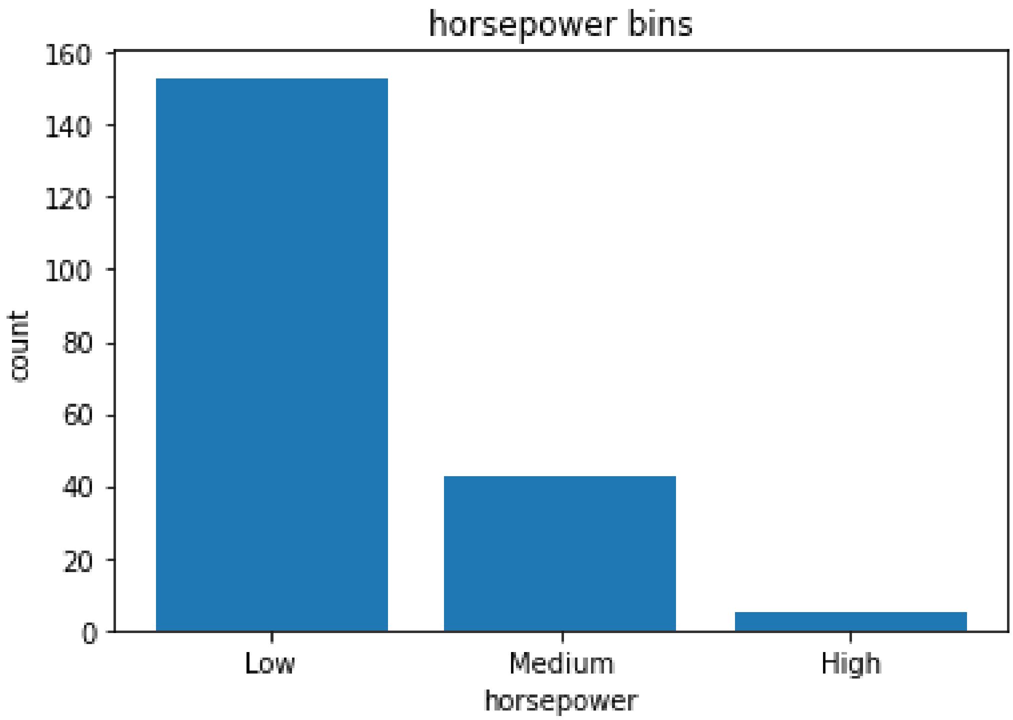
```
Low       153
Medium     43
High        5
Name: horsepower-binned, dtype: int64
```

Let's plot the distribution of each bin:

```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
pyplot.bar(group_names, df["horsepower-binned"].value_counts())

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```



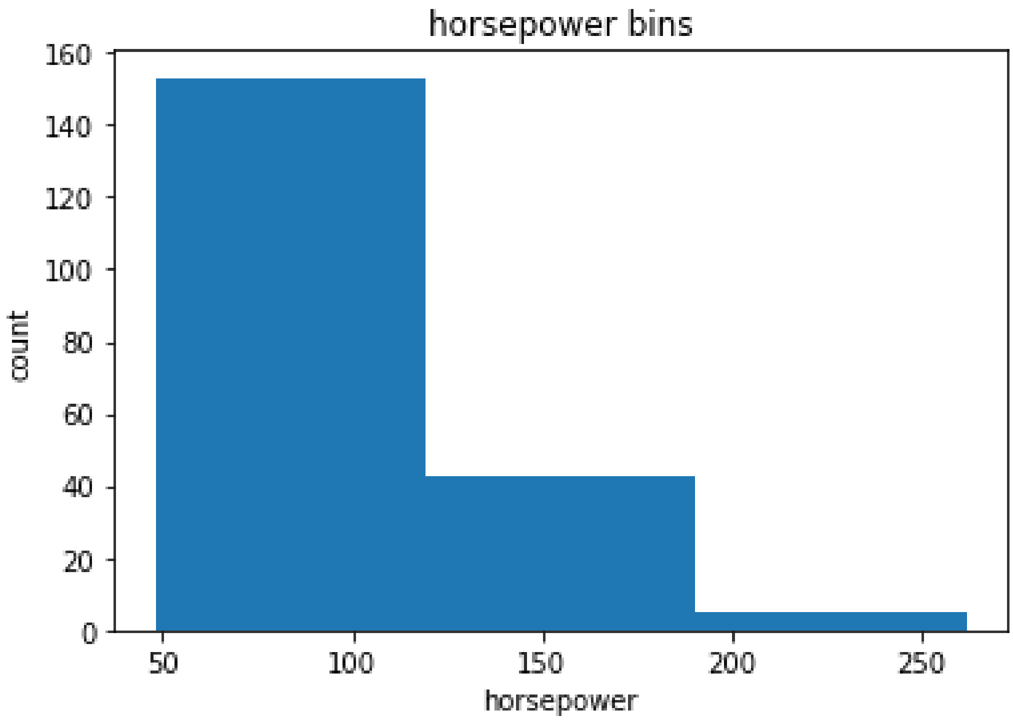
Normally, a histogram is used to visualize the distribution of bins we created above.

```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot


# draw historgram of attribute "horsepower" with bins = 3
plt.pyplot.hist(df["horsepower"], bins = 3)

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```



The plot above shows the binning result for the attribute "horsepower".

What is an indicator variable? An indicator variable (or dummy variable) is a numerical variable used to label categories. They are called 'dummies' because the numbers themselves don't have inherent meaning.

Why we use indicator variables?

Example We see the column "fuel-type" has two unique values: "gas" or "diesel". Regression doesn't understand words, only numbers. To use this attribute in regression analysis, we convert "fuel-type" to indicator variables.

```
df.columns
```

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type',
'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-
location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight',
'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore',
'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price', 'city-L/100km', 'highway-L/100km',
       'horsepower-binned'],
      dtype='object')
```

Get the indicator variables and assign it to data frame "dummy_variable_1":

```
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

```
   diesel  gas
0       0    1
1       0    1
2       0    1
3       0    1
4       0    1
```

Change the column names for clarity:

```
dummy_variable_1.rename(columns={'gas':'fuel-type-gas',
'diesel':'fuel-type-diesel'}, inplace=True)
dummy_variable_1.head()
```

```
   fuel-type-diesel  fuel-type-gas
0                 0              1
1                 0              1
2                 0              1
3                 0              1
4                 0              1
```

In the dataframe, column 'fuel-type' has values for 'gas' and 'diesel' as 0s and 1s now.

```
# merge data frame "df" and "dummy_variable_1"
df = pd.concat([df, dummy_variable_1], axis=1)

# drop original column "fuel-type" from "df"
df.drop("fuel-type", axis = 1, inplace=True)

df.head()
```

```
   symboling  normalized-losses         make aspiration num-of-
doors  \
0          3                122  alfa-romero        std          two

1          3                122  alfa-romero        std          two

2          1                122  alfa-romero        std          two

3          2                164         audi        std         four

4          2                164         audi        std         four


    body-style drive-wheels engine-location  wheel-base    length  ...
\
0  convertible          rwd           front        88.6  0.811148  ...

1  convertible          rwd           front        88.6  0.811148  ...
```

```
2    hatchback          rwd           front        94.5  0.822681  ...

3        sedan          fwd           front        99.8  0.848630  ...

4        sedan          4wd           front        99.4  0.848630  ...


   horsepower  peak-rpm  city-mpg highway-mpg    price  
city-L/100km  \
0         111    5000.0        21          27  13495.0     11.190476

1         111    5000.0        21          27  16500.0     11.190476

2         154    5000.0        19          26  16500.0     12.368421

3         102    5500.0        24          30  13950.0      9.791667

4         115    5500.0        18          22  17450.0     13.055556


  highway-L/100km  horsepower-binned  fuel-type-diesel  fuel-type-gas

0        8.703704                Low                 0              1

1        8.703704                Low                 0              1

2        9.038462             Medium                 0              1

3        7.833333                Low                 0              1

4       10.681818                Low                 0              1


[5 rows x 30 columns]
```

The last two columns are now the indicator variable representation of the fuel-type variable. They're all 0s and 1s now.

```
# Write your code below and press Shift+Enter to execute
print(df["aspiration"])
```

```
0        std
1        std
2        std
3        std
4        std
       ...
196      std
```

```
197    turbo
198      std
199    turbo
200    turbo
Name: aspiration, Length: 201, dtype: object
```

Question #5:

Merge the new dataframe to the original dataframe, then drop the column 'aspiration'.

```
# Write your code below and press Shift+Enter to execute
dummy_variable_2= pd.get_dummies(df['aspiration'])
dummy_variable_2.head()
dummy_variable_2.rename(columns={'std':'aspiration_std',
'turbo':'aspiration_turbo'}, inplace=True)
dummy_variable_2.head()
df=pd.concat([df,dummy_variable_2],axis=1)
df.drop("aspiration",axis=1,inplace=True)
```

Save the new csv:

```
df.to_csv('clean_df.csv')
```