Lab2 – Data Preparation

In this lab, you will learn how to perform data preparation for the with Pandas, Matlibplot and Numpy library. By the end of this lab, you will learn ways to handle missing data, data normalization and binning. The dataset named as auto.csv is used in this lab.

Software: Jupyter Notebook

Procedure

1. In the desktop directory upload Lab 2 file (Lab2 - Data Preparation – Participant Copy.ipynb) to the Jupyter Notebook.

2. Click the Lab2- Data Preparation – Participant Copy.ipynb file to start the lab.

What is the purpose of data preparation?

Data preparation is the process of converting data from the initial format to a format that may be better for analysis.

Import data

Upload the auto.csv into Jupyter Notebook as the data set will be used for this lab.

In [ ]:

```
#Import pandas, numpy and maplotlib library

import pandas as pd

import matplotlib as plt

import numpy as np

#set the filename as "auto.csv", create header list and
# read the file as df

filename = "auto.csv"

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"]

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

Exercise 2.1

Write the Python code to print the first 10 rows of the data set using `df.head()` method to obtain the output as shown below:

In [ ]:
```
# Write the code and press shift + enter
```

Out [ ]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepowe
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	11
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	11
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	15
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	10
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	11
5	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	11
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5	11
7	1	?	audi	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5	11
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.40	8.3	14
9	0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3.40	7.0	16

10 rows × 26 columns

Missing values

As we can see, many question marks (?) appeared in the data frame; those are missing values which may hinder our data analysis.

For the `'normalized-loses'`, how many missing data discovered in the first 10 rows of the data frame? _____

Identify missing values (*?*) and convert to *NaN* (Not a Number)

To replace *?* with *NaN* in auto.csv, we use `.replace(A, B, inplace = True)` to replace A by B.

In [ ]:
```
# replace "?" to NaN

df.replace("?", np.nan, inplace = True)

df.head(10)
```

Out [ ]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepowe
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	11
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	11
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	15
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	10
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	11
5	2	NaN	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	11
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5	11
7	1	NaN	audi	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5	11
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.40	8.3	14
9	0	NaN	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3.40	7.0	16

10 rows × 26 columns

Evaluate Missing Value

There are two methods to detect missing value:

`.isnull()`

`.notnull()`

The output is a Boolean value indicating whether the value that is passed into the argument is a missing value.

In [ ]:

```
missing_data = df.isnull()

missing_data.head(10)
```

Out[ ]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	p
0	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
1	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
2	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
5	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
6	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
7	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
8	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F
9	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	F

10 rows × 26 columns

To count the missing values in each column, we use a `for` loop in Python to quickly figure out the number of missing values in each column. In the body of the `for` loop the method `.value_counts()` counts the number of `'True'` values.

In [ ]:

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

Out[ ]:

```
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
```

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

1. 'normalized-losses' 41 missing values
2. 'num-of-doors' 2 missing values
3. 'bore' 4 missing values
4. 'stroke' 4 missing values
5. 'horsepower' 2 missing values
6. 'peak-rpm' 2 missing values
7. 'price' 4 missing values

Commonly used methods for handling missing values:
1. Replace them with means
2. Replace them with modes
3. Replace them with other function
4. Discard them (by row/column)
5. …….

Replace them with means

- 'normalized-losses' - 41 missing values
- 'stroke' - 4 missing values
- 'bore' - 4 missing values
- 'horsepower' - 2 missing values
- 'peak-rpm' - 2 missing values

In [ ]:
```
#code to find mean of normalized-losses and replace the
# missing values (NaN) in it with mean

avg_norm_loss=df['normalized-
losses'].astype("float").mean(axis=0)

print("Average of normalized losses:", avg_norm_loss)

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

#code to find mean of bore and replace the missing values
#(NaN) in it with mean

avg_bore=df['bore'].astype('float').mean(axis=0)

print("Average of bore:", avg_bore)

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

```
Average of normalized losses: 122.0
Average of bore: 3.3297512437810957
```

Exercise 2.2

Write the Python code to find the mean of `'stroke'`, `'horsepower'` and `'peak-rpm'` and replace them with means.

In [ ]:
```
# Write your code below and press shift + enter to execute
```

```
Average of stroke: 3.2554228855721337
Average horsepower: 104.25615763546799
Average peak rpm: 5125.369458128079
```

Replace them with modes

- num-of-doors: 2 missing data, replace them with "four".

We use `.value_counts()` to find which values are present in a particular column. We can see that `'four doors'` is the most common type by using the `.idxmax()` to calculate it.

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

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

#replace the missing 'num-of-doors' values by the mode

df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

```
four    116
two      89
Name: num-of-doors, dtype: int64
four
```

Discard them

price: 4 missing values (4 rows), simply delete them.

In [ ]:
```
# Drop whole row with NaN in "price" column

df.dropna(subset=["price"], axis=0, inplace=True)

# reset index, because we droped two rows

df.reset_index(drop=True, inplace=True)

df.head(201)
```

Out[ ]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsep
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
196	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	
197	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	
198	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	
199	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	
200	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	

201 rows × 26 columns

Finally, we obtain the dataset with no missing values.

Data Normalization

Normalization is the process of transforming values of several variables into a similar range. Typical normalizations include scaling the variable to the range of 0 to 1.

Example min-max normalization = (original value - minimum value)/(maximum value - minimum value)

Min-max normalization for normalizing the `'length'` and `'width'` to the ranges of 0 to 1.

In [ ]:

```
length_min=df['length'].min()

length_max=df['length'].max()

df['length'] = (df['length']-length_min)/(length_max - length_min)

width_min=df['width'].min()

width_max=df['width'].max()

df['width'] = (df['width']-width_min)/(width_max-width_min)
```

Exercise 2.3

Write the Python code to normalize the column *'height'* and display the normalized values of `'length'`, `'width'` and `'height'`.

In [ ]:	`# Write your code below and press shift + enter to execute`

	length	width	height
0	0.413433	0.324786	0.083333
1	0.413433	0.324786	0.083333
2	0.449254	0.444444	0.383333
3	0.529851	0.504274	0.541667
4	0.529851	0.521368	0.541667
...	...	...	...
196	0.711940	0.735043	0.641667
197	0.711940	0.726496	0.641667
198	0.711940	0.735043	0.641667
199	0.711940	0.735043	0.641667
200	0.711940	0.735043	0.641667

201 rows × 3 columns

Binning

It is the process of converting real valued variable into categorical variable (bins).

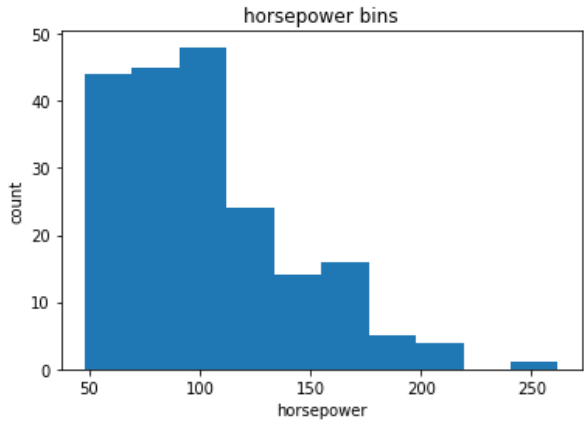
Example 'horsepower' ranges from 48 to 288, and with 57 unique values. If we want to only divide them into 3 categories:

- Low
- Medium
- High

We will use the Pandas method 'cut' to segment the 'horsepower' into 3 bins and plot the histogram to see the distribution of horsepower.

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

%matplotlib inline

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')
```



Use numpy's *linspace(start_value, end_value, numbers_generated)* to generate 3 bins of equal size.

Find the start and end value of horsepower using *min()* and *max()* (*start_value=min(df["horsepower"])*, *end_value=max(df["horsepower"])*. Since we are building 3 bins of equal size, there should be 4 dividers, so numbers_generated=4.

We build a bin array, with a minimum value to a maximum value, with 4 dividers.

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

bins
```

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

Set group names and apply `cut()` to determine what bin each value of `"df['horsepower']"` belongs to.

In [ ]:	

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

binsgroup_names = ['Low', 'Medium', 'High']

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

df[['horsepower','horsepower-binned']].head(201)
```

	horsepower	horsepower-binned
0	111	Low
1	111	Low
2	154	Medium
3	102	Low
4	115	Low
...	...	...
196	114	Low
197	160	Medium
198	134	Medium
199	106	Low
200	114	Low

201 rows × 2 columns

Exercise 2.4

Write the code by using `value_counts()` to see the number of vehicles in each bin (Low, Medium and High).

In [ ]:	`#Write the code and press shift + enter`

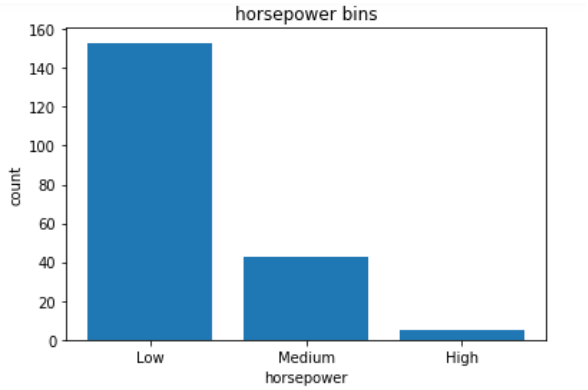
Record the output.

Out [ ]:	

Plot histogram to see the distribution of `'horsepower-binned'` (Low, Medium and High).

In [ ]:

```
%matplotlib inline

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")
```



Save Dataset

Pandas enables us to save the dataset to by using the following methods:

Data Formate	Read	Save
csv	`pd.read_csv()`	`df.to_csv()`
json	`pd.read_json()`	`df.to_json()`
excel	`pd.read_excel()`	`df.to_excel()`
hdf	`pd.read_hdf()`	`df.to_hdf()`
sql	`pd.read_sql()`	`df.to_sql()`
...	...	...

Exercise 2.5

Write the code to save the prepared csv to 'prepared_df.csv'.

In [ ]:	`#Write the code and press shift + enter`

--- End of Lab2 ---