Page No.
Date

Name:- Rohini Janardan Devkar
Roll no:- 23272
PRN no:- 72030818G
TE-2
DSBDA Lab-

Practical No:-1.
Data Wrangling 1.

* Aim:-

Data Wrangling I :.

Perform the following operations using python on any source dataset (e.g. data.csv)

1. Import all the required python libraries.
2. Locate an open source data from the web (e.g. www.kaggle.com). Provide a clear description of the data and its source (i.e. URL of the website).
3. Load the Dataset into pandas dataframe.
4. Data preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.
5. Data formatting and Data Normalization : summarize the types of variables by checking the data types (i.e. character, numeric, integer, factor, and logical) of the variables in the dataset. If variables are not in the correct data type, apply proper type conversions.
6. Turn categorical variables into quantitive variables in python.

In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import /read/scrape the dataset.

* Prerequisite:-

1. Basic of python programming.

Page No.
Date

2. Concept of data preprocessing, data formatting, data normalization and data cleaning.

* Theory:-

1. Introduction to dataset :-

A dataset is a collection of records, similar to a relational database table. Records are similar to table rows, but the columns can contain not only strings or numbers, but also nested data structures such as lists, maps, and other records.

- Instance :- A single row of data is called an instance. It is an observation from the domain.

- Feature:- A single column of data is called a feature. It is a component of an observation and is called an attribute of a data instance. Some features may be inputs to a model (the predictors) and others may be outputs or the features to be predicted.

- Data type:- Features have a data type. They may be real or integer-valued or may have a categorical or ordinal value. You can have strings, dates, times and more complex types, but typically they are reduced to real or categorical values when working with traditional machine learning methods.

- Data Sets :- A collection of instances is a dataset and when working with machine learning methods we typically need a few datasets for different purposes.

Date

- Training Dataset :- A dataset that we feed into our machine learning algorithm to train our model.

- Testing Dataset :- A dataset that we use to validate the accuracy of our model but is not used to train the model. It may be called the validation dataset.

- Data Represented in a Table:-

Data should be arranged in a two-dimensional space made up of rows and columns. This type of data structure makes it easy to understand the data and pinpoint any problems.

e.g. some raw data stored as a CSV (comma separated values).

1., Avatar, 18-12-2009, 7.8

2., Titanic, 18-11-1997,

3., Avengers Infinity War, 27-04-2018, 8.5

The representation of the same data in a table:-

S.No	Movie	Release Date	Ratings (IMDb)
1.	Avatar	18-12-2009	7.8
2.	Titanic	18-11-1997	Na
3.	Avengers Infinity War	27-04-2018	8.5

- Pandas Data types :-

A data type is essentially an internal construct that a programming language uses to understand how to store and manipulate data.

A possible confusing point about pandas data types is that there is some overlap between pandas, python and numpy. This table summarizes the key points:

Pandas dtype	Python type	NumPy type	Usage
Object	Str or mixed	String_, unicode_, mixed types	Text or mixed numeric and non-numeric values
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64.	Integer numbers
float64	float	float_, float16, float32, float64.	Floating point numbers.
bool	bool	bool_	True/false values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences betⁿ two datetimes
category	NA	NA	Finite list of text values.

2. Python Libraries for :-

a. Pandas :-

Pandas is an open-source python package that provides high-performance, easy-to-use data structures and data analysis tools for the

labeledt data in python programming language.

b. NumPy :-

One of the most fundamental packages in python, NumPy is a general-purpose array-processing package. It provides high-performance multidimensional array objects and tools to work with the arrays. NumPy is an efficient container of generic multi-dimensional data.

c. Matplotlib :-

This is undoubtedly my favourite and a quintessential python library You can create stories with the data visualized with Matplotlib. Another library from the SciPy Stack, Matplotlib plots 2D figures.

d. Seaborn :-

When you read the official documentation on Seaborn, it is defined as the data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. Putting it simply, seaborn is an extension of Matplotlib with advanced features.

e. Scikit Learn :-

Introduced to the world as a Google summer of code project, Scikit Learn is a robust machine learning library for python. It features ML algorithms like SVM's, random forests, k-means clustering, spectral clustering, mean shift, cross-validation and more. Even NumPy, SciPy and related scientific operations are supported by Scikit Learn with Scikit learn being a part of the SciPy Stack.

Data Science And Big Data Analytics Practical 1

==========

NAME: Rohini Janardan Devkar

ROLL NO.: 23272

PRN NO:- 72030818G

Class :- TE2(COMP)

==========

Perform the following operations using Python on any open source dataset (e.g., data.csv)

1. Import all the required Python Libraries.

2. Locate an open source data from the web (e.g. https://www.kaggle.com). Provide a. clear description of the data and its source (i.e., URL of the web site).

3.Load the Dataset into pandas data frame.

4.Data Preprocessing: check for missing values in the data using pandas insull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.

5.Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.

6.Turn categorical variables into quantitative variables in Python.

In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.

==========

Part-1

Data Wrangling

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

Import the required libraries

In [1]:
```
import pandas as pd
```

```
import matplotlib.pylab as plt
import numpy as np
```

Load the dataset using pandas library

In [2]:
```
url_link='https://raw.githubusercontent.com/rohinidevkar/DSBDA/main/autodata.csv'
df = pd.read_csv(url_link)
```

Check the contents of dataset using **df.head()** and **df.tail()** functions

In [3]:
```
df.head(5)
```

Out[3]:

	Unnamed: 0	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base
0	0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6
1	1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6
2	2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5
3	3	2	164	audi	std	four	sedan	fwd	front	99.8
4	4	2	164	audi	std	four	sedan	4wd	front	99.4

5 rows × 30 columns

In [4]:
```
df.tail(5)
```

Out[4]:

	Unnamed: 0	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
196	196	-1	95	volvo	std	four	sedan	rwd	front	109.1	...
197	197	-1	95	volvo	turbo	four	sedan	rwd	front	109.1	...
198	198	-1	95	volvo	std	four	sedan	rwd	front	109.1	...
199	199	-1	95	volvo	turbo	four	sedan	rwd	front	109.1	...
200	200	-1	95	volvo	turbo	four	sedan	rwd	front	109.1	...

5 rows × 30 columns

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
```

```
Data columns (total 30 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         201 non-null    int64
 1   symboling          201 non-null    int64
 2   normalized-losses  201 non-null    int64
 3   make               201 non-null    object
 4   aspiration         201 non-null    object
 5   num-of-doors       201 non-null    object
 6   body-style         201 non-null    object
 7   drive-wheels       201 non-null    object
 8   engine-location    201 non-null    object
 9   wheel-base         201 non-null    float64
 10  length             201 non-null    float64
 11  width              201 non-null    float64
 12  height             201 non-null    float64
 13  curb-weight        201 non-null    int64
 14  engine-type        201 non-null    object
 15  num-of-cylinders   201 non-null    object
 16  engine-size        201 non-null    int64
 17  fuel-system        201 non-null    object
 18  bore               201 non-null    float64
 19  stroke             197 non-null    float64
 20  compression-ratio  201 non-null    float64
 21  horsepower         199 non-null    float64
 22  peak-rpm           199 non-null    float64
 23  city-mpg           201 non-null    int64
 24  highway-mpg        201 non-null    int64
 25  price              201 non-null    float64
 26  city-L/100km       201 non-null    float64
 27  horsepower-binned  199 non-null    object
 28  diesel             201 non-null    int64
 29  gas                201 non-null    int64
dtypes: float64(11), int64(9), object(10)
memory usage: 47.2+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

	Unnamed: 0	symboling	normalized-losses	wheel-base	length	width	height	curb weigh
count	201.000000	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.00000(
mean	100.000000	0.840796	122.00000	98.797015	0.837102	0.915126	53.766667	2555.66666
std	58.167861	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	517.29672
min	0.000000	-2.000000	65.00000	86.600000	0.678039	0.837500	47.800000	1488.00000(
25%	50.000000	0.000000	101.00000	94.500000	0.801538	0.890278	52.000000	2169.00000(
50%	100.000000	1.000000	122.00000	97.000000	0.832292	0.909722	54.100000	2414.00000(
75%	150.000000	2.000000	137.00000	102.400000	0.881788	0.925000	55.500000	2926.00000(
max	200.000000	3.000000	256.00000	120.900000	1.000000	1.000000	59.800000	4066.00000(

Evaluating for Missing Data

The missing values are converted to Python's default. We use Python's built-in functions to identify these missing values. There are two methods to detect missing data:

- isnull()
- .notnull()

The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data. "True" stands for missing value, while "False" stands for not missing value.

Deal with missing data

1. **Drop data**
   - Drop the whole row
   - Drop the whole column
2. **Replace data**
   - Replace it by mean
   - Replace it by frequency / mode
   - Replace it based on other functions

In [7]:
```
df.isnull()
```

Out[7]:

	Unnamed: 0	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
0	False	False	False	False	False	False	False	False	False	False	...
1	False	False	False	False	False	False	False	False	False	False	...
2	False	False	False	False	False	False	False	False	False	False	...
3	False	False	False	False	False	False	False	False	False	False	...
4	False	False	False	False	False	False	False	False	False	False	...
...	...	...	...	...	...	...	...	...	...	...	...
196	False	False	False	False	False	False	False	False	False	False	...
197	False	False	False	False	False	False	False	False	False	False	...
198	False	False	False	False	False	False	False	False	False	False	...
199	False	False	False	False	False	False	False	False	False	False	...
200	False	False	False	False	False	False	False	False	False	False	...

201 rows × 30 columns

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Unnamed: 0           0
```

```
symboling           0
normalized-losses   0
make                0
aspiration          0
num-of-doors        0
body-style          0
drive-wheels        0
engine-location     0
wheel-base          0
length              0
width               0
height              0
curb-weight         0
engine-type         0
num-of-cylinders    0
engine-size         0
fuel-system         0
bore                0
stroke              4
compression-ratio   0
horsepower          2
peak-rpm            2
city-mpg            0
highway-mpg         0
price               0
city-L/100km        0
horsepower-binned   2
diesel              0
gas                 0
dtype: int64
```

In [9]:
```
df.notnull()
```

Out[9]:

	Unnamed: 0	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
0	True	True	True	True	True	True	True	True	True	True	...
1	True	True	True	True	True	True	True	True	True	True	...
2	True	True	True	True	True	True	True	True	True	True	...
3	True	True	True	True	True	True	True	True	True	True	...
4	True	True	True	True	True	True	True	True	True	True	...
...	...	...	...	...	...	...	...	...	...	...	...
196	True	True	True	True	True	True	True	True	True	True	...
197	True	True	True	True	True	True	True	True	True	True	...
198	True	True	True	True	True	True	True	True	True	True	...
199	True	True	True	True	True	True	True	True	True	True	...
200	True	True	True	True	True	True	True	True	True	True	...

201 rows × 30 columns

In [10]:
```
df.notnull().sum()
```

Out[10]:
```
Unnamed: 0           201
symboling            201
normalized-losses    201
make                 201
aspiration           201
num-of-doors         201
body-style           201
drive-wheels         201
engine-location      201
wheel-base           201
length               201
width                201
height               201
curb-weight          201
engine-type          201
num-of-cylinders     201
engine-size          201
fuel-system          201
bore                 201
stroke               197
compression-ratio    201
horsepower           199
peak-rpm             199
city-mpg             201
highway-mpg          201
price                201
city-L/100km         201
horsepower-binned    199
diesel               201
gas                  201
dtype: int64
```

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

stroke : 4 missing data

horsepower: 2 missing data

peak-rpm: 2 missing data

horsepower-binned: 2 missing data

Question 1

In [11]:
```
# 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: 3.2569035532994857
```

Calculate the mean value for the 'horsepower' column:

In [12]:
```
avg_hp = df["horsepower"].astype("float").mean(axis = 0)
print("Average of stroke:", avg_hp)
```

```
Average of stroke: 103.39698492462311
```

Replace "NaN" by mean value:

In [13]:
```
df["horsepower"].replace(np.nan, avg_hp, inplace = True)
```

Calculate the mean value for 'peak-rpm' column:

In [14]:
```
avg_rpm = df["peak-rpm"].astype("float").mean(axis = 0)
print("Average of stroke:", avg_rpm)
```

```
Average of stroke: 5117.587939698493
```

Replace NaN by mean value:

In [15]:
```
df["peak-rpm"].replace(np.nan, avg_hp, inplace = True)
```

In [16]:
```
df['num-of-doors'].value_counts()
```

Out[16]:
```
four    115
two      86
Name: num-of-doors, dtype: int64
```

In [17]:
```
df['num-of-doors'].value_counts().idxmax()
```

Out[17]:
```
'four'
```

In [18]:
```
#replace the missing 'num-of-doors' values by the most frequent
df["num-of-doors"].replace(np.nan, "four", inplace=True)

# simply drop whole row with NaN in "horsepower-binned" column
df.dropna(subset=["horsepower-binned"], axis=0, inplace=True)

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

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
Unnamed: 0           0
symboling            0
normalized-losses    0
make                 0
aspiration           0
num-of-doors         0
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
```

```
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
city-L/100km         0
horsepower-binned    0
diesel               0
gas                  0
dtype: int64
```

Here, we can see that there are no missing values in the data.

Part-2

Data Standardization

Data is usually collected from different agencies with different formats. (Data Standardization is also a term for a particular type of data normalization, where we subtract the mean and divide by the standard deviation)

What is Standardization?

Standardization is the process of transforming data into a common format which allows the researcher to make the meaningful comparison.

Example:

Transform mpg to L/100km:

In our dataset, the fuel consumption columns "city-mpg" and "highway-mpg" are represented by mpg (miles per gallon) unit. Assume we are developing an application in a country that accept the fuel consumption with L/100km standard

In [20]:
```
df['city-L/100km'] = 235/df["city-mpg"]
df.head()
```

Out[20]:

	Unnamed: 0	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base

	Unnamed: 0	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base
0	0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6
1	1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6
2	2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5
3	3	2	164	audi	std	four	sedan	fwd	front	99.8
4	4	2	164	audi	std	four	sedan	4wd	front	99.4

5 rows × 30 columns

In [21]:
```
df['highway-L/100km'] = 235/df["highway-mpg"]
df.head()
```

Out[21]:

	Unnamed: 0	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base
0	0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6
1	1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6
2	2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5
3	3	2	164	audi	std	four	sedan	fwd	front	99.8
4	4	2	164	audi	std	four	sedan	4wd	front	99.4

5 rows × 31 columns

Data Normalization

Normalization is the process of transforming values of several variables into a similar range. Typical normalizations include scaling the variable so the variable average is 0, scaling the variable so the variance is 1, or scaling variable so the variable values range from 0 to 1

Example:

To demonstrate normalization, let's say we want to scale the columns "length", "width" and "height"

Target:would like to Normalize those variables so their value ranges from 0 to 1.

In [22]:
```
df['length'] = df['length']/df['length'].max()
df['width'] = df['width']/df['width'].max()
```

In [23]:
```
df['height'] = df['height']/df['height'].max()
df[["length","width","height"]].head()
```

Out[23]:

	length	width	height
0	0.811148	0.890278	0.816054
1	0.811148	0.890278	0.816054
2	0.822681	0.909722	0.876254
3	0.848630	0.919444	0.908027
4	0.848630	0.922222	0.908027

Indicator variable (or dummy 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?

So we can use categorical variables for regression analysis in the later modules.

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" into indicator variables.

We will use the panda's method 'get_dummies' to assign numerical values to different categories of fuel type.

In [24]:
```
df.columns
```

Out[24]:
```
Index(['Unnamed: 0', 'symboling', 'normalized-losses', 'make', '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', 'horsepower-binned', 'diesel',
       'gas', 'highway-L/100km'],
      dtype='object')
```

In [25]:
```
df['aspiration'].value_counts()
```

Out[25]:
```
std      163
turbo     36
Name: aspiration, dtype: int64
```

In [26]:
```
dummy_variable_1 = pd.get_dummies(df["aspiration"])
```

```
dummy_variable_1.head()
```

Out[26]:

	std	turbo
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

We now have the value 0 to represent "turbo" and 1 to represent "std" in the column "aspiration". We will now insert this column back into our original dataset.

In [27]:

```
df = pd.concat([df, dummy_variable_1], axis=1)
df.drop("aspiration", axis = 1, inplace=True)
```

In [28]:

```
df.head()
```

Out[28]:

	Unnamed: 0	symboling	normalized-losses	make	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length
0	0	3	122	alfa-romero	two	convertible	rwd	front	88.6	0.811148
1	1	3	122	alfa-romero	two	convertible	rwd	front	88.6	0.811148
2	2	1	122	alfa-romero	two	hatchback	rwd	front	94.5	0.822681
3	3	2	164	audi	four	sedan	fwd	front	99.8	0.848630
4	4	2	164	audi	four	sedan	4wd	front	99.4	0.848630

5 rows × 32 columns

The last two columns are now the indicator variable representation of the aspiration variable. It's all 0s and 1s now.

Binning

Binning is a process of transforming continuous numerical variables into discrete categorical 'bins', for grouped analysis.

Example:

In our dataset, "horsepower" is a real valued variable ranging from 48 to 288, it has 57 unique values. What if we only care about the price difference between cars with high horsepower, medium

horsepower, and little horsepower (3 types)? Can we rearrange them into three 'bins' to simplify analysis?

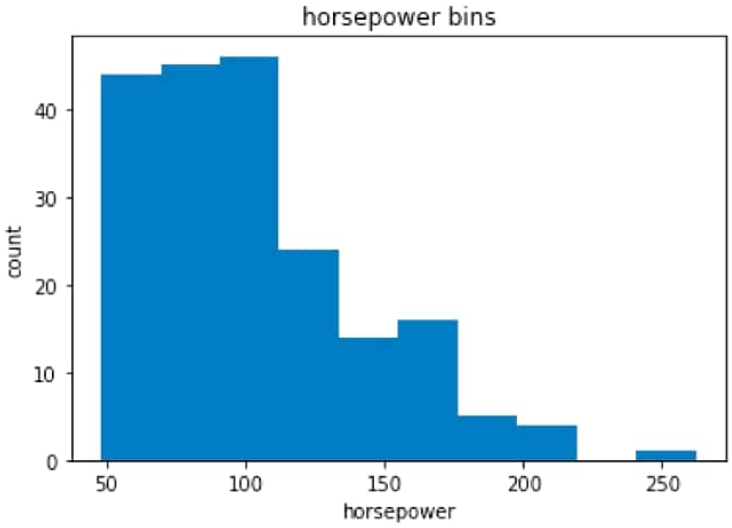
Horsepower

In [29]:
```
df["horsepower"]=df["horsepower"].astype(float, copy=True)
```

In [30]:
```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["horsepower"])

plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

Out[30]:
```
Text(0.5, 1.0, 'horsepower bins')
```



In [31]:
```
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
bins
```

Out[31]:
```
array([ 48.        , 119.33333333, 190.66666667, 262.        ])
```

In [32]:
```
group_names = ['Low', 'Medium', 'High']
```

In [33]:
```
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names, include_lo
df[['horsepower','horsepower-binned']].head(20)
```

Out[33]:

	horsepower	horsepower-binned
0	111.0	Low
1	111.0	Low
2	154.0	Medium

	horsepower	horsepower-binned
3	102.0	Low
4	115.0	Low
5	110.0	Low
6	110.0	Low
7	110.0	Low
8	140.0	Medium
9	101.0	Low
10	101.0	Low
11	121.0	Medium
12	121.0	Medium
13	121.0	Medium
14	182.0	Medium
15	182.0	Medium
16	182.0	Medium
17	48.0	Low
18	70.0	Low
19	70.0	Low

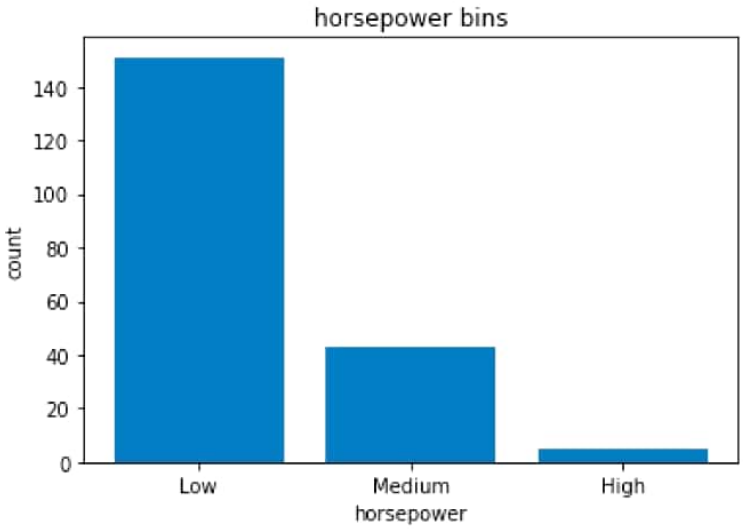
In [34]:
```
df["horsepower-binned"].value_counts()
```

Out[34]:
```
Low       151
Medium     43
High        5
Name: horsepower-binned, dtype: int64
```

In [35]:
```
%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")
```

Out[35]:
```
Text(0.5, 1.0, 'horsepower bins')
```



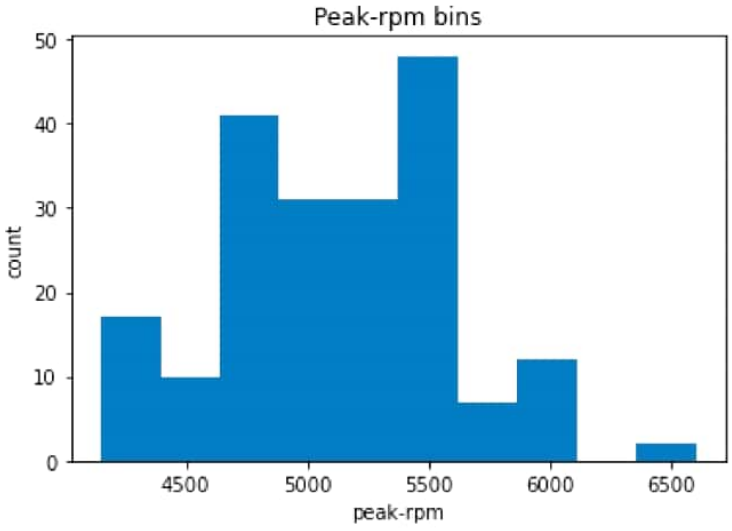
Peak-rpm

In [36]:
```
df["peak-rpm"]=df["peak-rpm"].astype(float, copy=True)
```

In [37]:
```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["peak-rpm"])

plt.pyplot.xlabel("peak-rpm")
plt.pyplot.ylabel("count")
plt.pyplot.title("Peak-rpm bins")
```

Out[37]:
```
Text(0.5, 1.0, 'Peak-rpm bins')
```



In [38]:
```
bins = np.linspace(min(df["peak-rpm"]), max(df["peak-rpm"]), 4)
bins
```

Out[38]:
```
array([4150.        , 4966.66666667, 5783.33333333, 6600.        ])
```

In [39]:
```
group_names1 = ['Low', 'Medium', 'High']
```

In [40]:
```
df['peakrpm-binned'] = pd.cut(df['peak-rpm'], bins, labels=group_names, include_lowest=
df[['peak-rpm','peakrpm-binned']].head(20)
```

Out[40]:

	peak-rpm	peakrpm-binned
0	5000.0	Medium
1	5000.0	Medium
2	5000.0	Medium
3	5500.0	Medium
4	5500.0	Medium
5	5500.0	Medium
6	5500.0	Medium
7	5500.0	Medium
8	5500.0	Medium
9	5800.0	High
10	5800.0	High
11	4250.0	Low
12	4250.0	Low
13	4250.0	Low
14	5400.0	Medium
15	5400.0	Medium
16	5400.0	Medium
17	5100.0	Medium
18	5400.0	Medium
19	5400.0	Medium

In [41]:
```
df["peakrpm-binned"].value_counts()
```

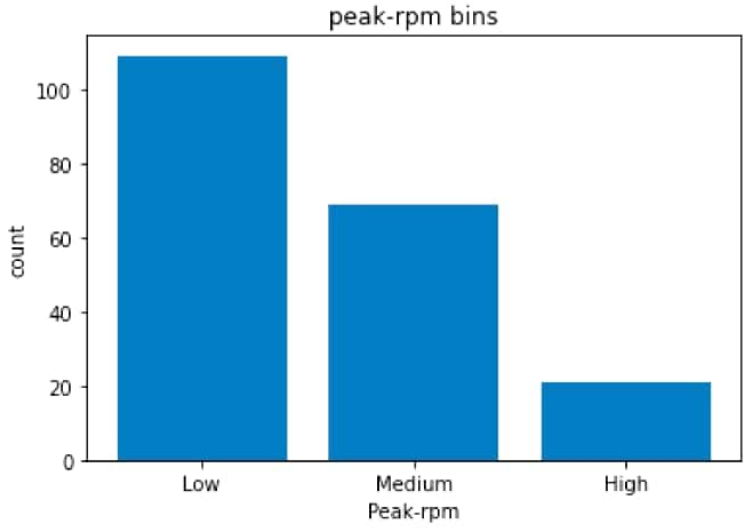
Out[41]:
```
Medium    109
Low        69
High       21
Name: peakrpm-binned, dtype: int64
```

In [42]:
```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
pyplot.bar(group_names, df["peakrpm-binned"].value_counts())

# set x/y labels and plot title
plt.pyplot.xlabel("Peak-rpm")
```

```
plt.pyplot.ylabel("count")
plt.pyplot.title("peak-rpm bins")
```

Out[42]:
```
Text(0.5, 1.0, 'peak-rpm bins')
```



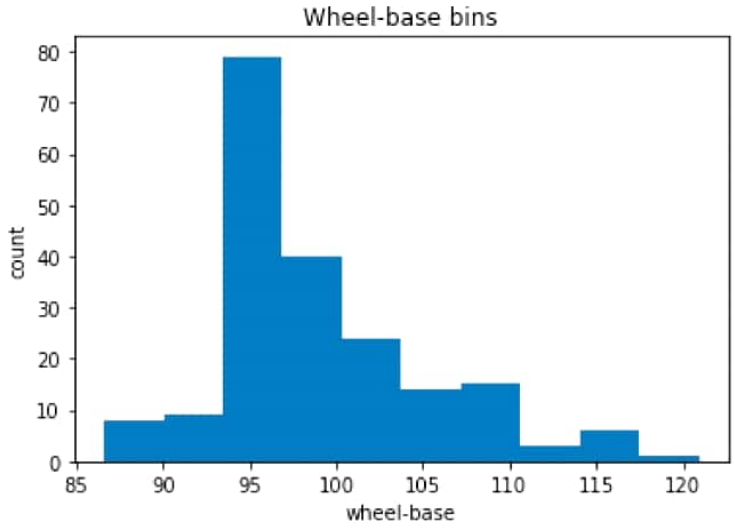
Wheel-base

In [43]:
```
df["wheel-base"]=df["wheel-base"].astype(float, copy=True)
```

In [44]:
```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["wheel-base"])

plt.pyplot.xlabel("wheel-base")
plt.pyplot.ylabel("count")
plt.pyplot.title("Wheel-base bins")
```

Out[44]:
```
Text(0.5, 1.0, 'Wheel-base bins')
```



In [45]:

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

Out[45]:
```
array([ 86.6       ,  98.03333333, 109.46666667, 120.9       ])
```

In [46]:
```
group_names = ['Low', 'Medium', 'High']
```

In [47]:
```
df['wheelbase-binned'] = pd.cut(df['wheel-base'], bins, labels=group_names, include_low
df[['wheel-base','wheelbase-binned']].head(20)
```

Out[47]:

	wheel-base	wheelbase-binned
0	88.6	Low
1	88.6	Low
2	94.5	Low
3	99.8	Medium
4	99.4	Medium
5	99.8	Medium
6	105.8	Medium
7	105.8	Medium
8	105.8	Medium
9	101.2	Medium
10	101.2	Medium
11	101.2	Medium
12	101.2	Medium
13	103.5	Medium
14	103.5	Medium
15	103.5	Medium
16	110.0	High
17	88.4	Low
18	94.5	Low
19	94.5	Low

In [48]:
```
df["wheelbase-binned"].value_counts()
```

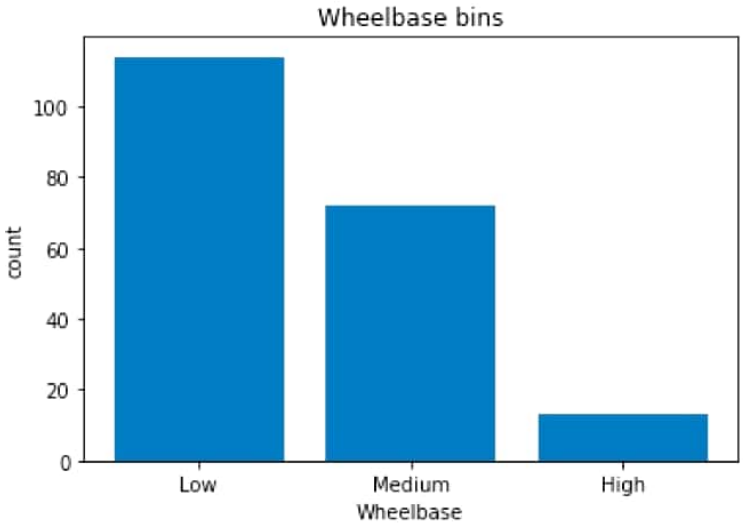
Out[48]:
```
Low       114
Medium     72
High       13
Name: wheelbase-binned, dtype: int64
```

In [49]:
```
%matplotlib inline
```

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

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

Out[49]: Text(0.5, 1.0, 'Wheelbase bins')



Page No.
Date

* Conclusion:-

In this way we have explored the functions of python library for data preprocessing, data wrangling techniques and how to handle missing values on iris dataset.