d7joooivs

February 17, 2025

[1]:
```
# Experiment 2  Data Wranglling II : create student Academic Data Base
# 1. Find Missing Values and use any suitable technique to deal with
```

[5]:
```
import pandas as pd
import numpy as np
```

[6]:
```
# create Student Academic Performance Data Set
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":[60,70,80,90,np.
 ↪nan],"English_marks":[70,80,90,np.nan,60],"Science_marks":[80,90,np.
 ↪nan,60,70],"History_marks":[90,np.nan,60,70,80],"Geography_marks":[np.
 ↪nan,60,70,80,90]}
```

[7]:
```
df1=pd.DataFrame(dict1)
```

[8]:
```
# find Missing Values from data set isnull() return True if missing value in␣
 ↪data set.
df1.isnull()
```

[8]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	Geography_marks
0	False	False	False	False	False	False	True
1	False	False	False	False	False	True	False
2	False	False	False	False	True	False	False
3	False	False	False	True	False	False	False
4	False	False	True	False	False	False	False

[9]:
```
# number of missing values from each colum
df1.isnull().sum()
```

[9]:
```
Roll_No            0
Name               0
Maths_marks        1
English_marks      1
Science_marks      1
History_marks      1
Geography_marks    1
dtype: int64
```

[10]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Roll_No          5 non-null      int64
 1   Name             5 non-null      object
 2   Maths_marks      4 non-null      float64
 3   English_marks    4 non-null      float64
 4   Science_marks    4 non-null      float64
 5   History_marks    4 non-null      float64
 6   Geography_marks  4 non-null      float64
dtypes: float64(5), int64(1), object(1)
memory usage: 408.0+ bytes
```

[11]:
```
# Missing values dealing techniques
#1.filling missing value using fillna()
df1.fillna(0,inplace=True)
df1
```

[11]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	Geography_marks
0	1	Amol	60.0	70.0	80.0	90.0	0.0
1	2	Dipak	70.0	80.0	90.0	0.0	60.0
2	3	Shreya	80.0	90.0	0.0	60.0	70.0
3	4	Krisha	90.0	0.0	60.0	70.0	80.0
4	5	Pooja	0.0	60.0	70.0	80.0	90.0

[12]:
```
df1=pd.DataFrame(dict1)
df1
```

[12]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	\
0	1	Amol	60.0	70.0	80.0	90.0	
1	2	Dipak	70.0	80.0	90.0	NaN	
2	3	Shreya	80.0	90.0	NaN	60.0	
3	4	Krisha	90.0	NaN	60.0	70.0	
4	5	Pooja	NaN	60.0	70.0	80.0	

	Geography_marks
0	NaN
1	60.0
2	70.0
3	80.0
4	90.0

[22]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":[60,70,80,93,np.
 ↪nan],"English_marks":[70,80,40,np.nan,50],"Science_marks":[80,91,np.
 ↪nan,65,70],"History_marks":[76,np.nan,68,70,80],"Geography_marks":[np.
 ↪nan,62,70,80,90]}
df1=pd.DataFrame(dict1)
df1
```

[22]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	\
0	1	Amol	60.0	70.0	80.0	76.0	
1	2	Dipak	70.0	80.0	91.0	NaN	
2	3	Shreya	80.0	40.0	NaN	68.0	
3	4	Krisha	93.0	NaN	65.0	70.0	
4	5	Pooja	NaN	50.0	70.0	80.0	

	Geography_marks
0	NaN
1	62.0
2	70.0
3	80.0
4	90.0

[13]:
```
# filling a null values using fillna()
df1["Maths_marks"].fillna(45, inplace = True)
df1["English_marks"].fillna(55, inplace = True)
df1["Science_marks"].fillna(65, inplace = True)
df1["History_marks"].fillna(75, inplace = True)
df1["Geography_marks"].fillna(85, inplace = True)
```

[14]:
```
df1
```

[14]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	\
0	1	Amol	60.0	70.0	80.0	90.0	

```
1        2   Dipak          70.0           80.0           90.0           75.0
2        3  Shreya          80.0           90.0           65.0           60.0
3        4  Krisha          90.0           55.0           60.0           70.0
4        5   Pooja          45.0           60.0           70.0           80.0

   Geography_marks
0             85.0
1             60.0
2             70.0
3             80.0
4             90.0
```

[15]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":[60,70,80,93,np.
 ↪nan],"English_marks":[70,80,40,np.nan,50],"Science_marks":[80,91,np.
 ↪nan,65,70],"History_marks":[76,np.nan,68,70,80],"Geography_marks":[np.
 ↪nan,62,70,80,90]}
df1=pd.DataFrame(dict1)
df1
```

[15]:
```
   Roll_No    Name  Maths_marks  English_marks  Science_marks  History_marks  \
0        1    Amol         60.0           70.0           80.0           76.0
1        2   Dipak         70.0           80.0           91.0            NaN
2        3  Shreya         80.0           40.0            NaN           68.0
3        4  Krisha         93.0            NaN           65.0           70.0
4        5   Pooja          NaN           50.0           70.0           80.0

   Geography_marks
0              NaN
1             62.0
2             70.0
3             80.0
4             90.0
```

[33]:
```
# filling a null values using fillna() as mean of colum
df1["Maths_marks"].fillna(int(df1["Maths_marks"].mean()), inplace=True)
```

[34]:
```
df1
```

[34]:
```
   Roll_No    Name  Maths_marks  English_marks  Science_marks  History_marks  \
0        1    Amol         60.0           70.0           80.0           76.0
1        2   Dipak         70.0           80.0           91.0            NaN
2        3  Shreya         80.0           40.0            NaN           68.0
3        4  Krisha         93.0            NaN           65.0           70.0
4        5   Pooja         75.0           50.0           70.0           80.0

   Geography_marks
```

```
0              NaN
1             62.0
2             70.0
3             80.0
4             90.0
```

[27]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":[60,70,80,93,np.
 ↪nan],"English_marks":[70,80,40,np.nan,50],"Science_marks":[80,91,np.
 ↪nan,65,70],"History_marks":[76,np.nan,68,70,80],"Geography_marks":[np.
 ↪nan,62,70,80,90]}
df1=pd.DataFrame(dict1)
df1
```

[27]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	\
0	1	Amol	60.0	70.0	80.0	76.0	
1	2	Dipak	70.0	80.0	91.0	NaN	
2	3	Shreya	80.0	40.0	NaN	68.0	
3	4	Krisha	93.0	NaN	65.0	70.0	
4	5	Pooja	NaN	50.0	70.0	80.0	

	Geography_marks
0	NaN
1	62.0
2	70.0
3	80.0
4	90.0

[17]:
```
# to interpolate the missing values
df1.interpolate(method ='linear', limit_direction ='forward')
#c=(a+b)/2
```

[17]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	\
0	1	Amol	60.0	70.0	80.0	76.0	
1	2	Dipak	70.0	80.0	91.0	72.0	
2	3	Shreya	80.0	40.0	78.0	68.0	
3	4	Krisha	93.0	45.0	65.0	70.0	
4	5	Pooja	93.0	50.0	70.0	80.0	

	Geography_marks
0	NaN
1	62.0
2	70.0
3	80.0
4	90.0

[18]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":
 ↪[60,70,80,93,75],"English_marks":[70,80,40,np.nan,50],"Science_marks":
 ↪[80,91,np.nan,65,70],"History_marks":[76,np.nan,68,70,80],"Geography_marks":
 ↪[np.nan,62,70,80,90]}
df1=pd.DataFrame(dict1)
df1
```

[18]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	Geography_marks
0	1	Amol	60	70.0	80.0	76.0	NaN
1	2	Dipak	70	80.0	91.0	NaN	62.0
2	3	Shreya	80	40.0	NaN	68.0	70.0
3	4	Krisha	93	NaN	65.0	70.0	80.0
4	5	Pooja	75	50.0	70.0	80.0	90.0

[26]:
```
# using dropna() function
df1.dropna()
```

[26]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	Geography_marks
4	5	Pooja	75	50.0	70.0	80.0	90.0

[19]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":[60,70,80,93,np.
 ↪nan],"English_marks":[70,80,40,np.nan,50],"Science_marks":[80,91,np.
 ↪nan,65,70],"History_marks":[76,np.nan,68,70,80],"Geography_marks":[np.
 ↪nan,62,70,80,90]}
df1=pd.DataFrame(dict1)
df1
```

[19]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	Geography_marks
0	1	Amol	60.0	70.0	80.0	76.0	NaN
1	2	Dipak	70.0	80.0	91.0	NaN	62.0
2	3	Shreya	80.0	40.0	NaN	68.0	70.0
3	4	Krisha	93.0	NaN	65.0	70.0	80.0
4	5	Pooja	NaN	50.0	70.0	80.0	90.0

```
0              NaN
1             62.0
2             70.0
3             80.0
4             90.0
```

[28]:
```
# using dropna() function
df1.dropna(how = 'all',inplace=True)
```

[20]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":[60,70,80,93,np.
 ↪nan],"English_marks":[70,80,40,np.nan,50],"Science_marks":[80,91,np.
 ↪nan,65,70],"History_marks":[76,np.nan,68,70,80],"Geography_marks":[np.
 ↪nan,62,70,80,90]}
df1=pd.DataFrame(dict1)
df1
```

[20]:
```
   Roll_No    Name  Maths_marks  English_marks  Science_marks  History_marks  \
0        1    Amol         60.0           70.0           80.0           76.0
1        2   Dipak         70.0           80.0           91.0            NaN
2        3  Shreya         80.0           40.0            NaN           68.0
3        4  Krisha         93.0            NaN           65.0           70.0
4        5   Pooja          NaN           50.0           70.0           80.0

   Geography_marks
0              NaN
1             62.0
2             70.0
3             80.0
4             90.0
```

[21]:
```
# using dropna() function
df1.dropna(axis = 1)
```

[21]:
```
   Roll_No    Name
0        1    Amol
1        2   Dipak
2        3  Shreya
3        4  Krisha
4        5   Pooja
```

[31]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":[60,70,80,93,np.
 ↪nan],"English_marks":[70,80,40,np.nan,50],"Science_marks":[80,91,np.
 ↪nan,65,70],"History_marks":[76,np.nan,68,70,80],"Geography_marks":
 ↪[60,62,70,80,90]}
df1=pd.DataFrame(dict1)
```

```
df1
```

[31]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	Geography_marks
0	1	Amol	60.0	70.0	80.0	76.0	60
1	2	Dipak	70.0	80.0	91.0	NaN	62
2	3	Shreya	80.0	40.0	NaN	68.0	70
3	4	Krisha	93.0	NaN	65.0	70.0	80
4	5	Pooja	NaN	50.0	70.0	80.0	90

[22]:
```
# making new data frame with dropped NA values
df2 = df1.dropna(axis = 0, how ='any')
```

[33]:
```
df2
```

[33]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	Geography_marks
0	1	Amol	60.0	70.0	80.0	76.0	60

[23]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
 ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":[60,70,80,93,np.
 ↪nan],"English_marks":[70,80,40,np.nan,50],"Science_marks":[80,91,np.
 ↪nan,65,70],"History_marks":[76,np.nan,68,70,80],"Geography_marks":[np.
 ↪nan,62,70,80,90]}
df1=pd.DataFrame(dict1)
df1
```

[23]:

	Roll_No	Name	Maths_marks	English_marks	Science_marks	History_marks	Geography_marks
0	1	Amol	60.0	70.0	80.0	76.0	NaN
1	2	Dipak	70.0	80.0	91.0	NaN	62.0
2	3	Shreya	80.0	40.0	NaN	68.0	70.0
3	4	Krisha	93.0	NaN	65.0	70.0	80.0
4	5	Pooja	NaN	50.0	70.0	80.0	90.0

```
4             90.0
```

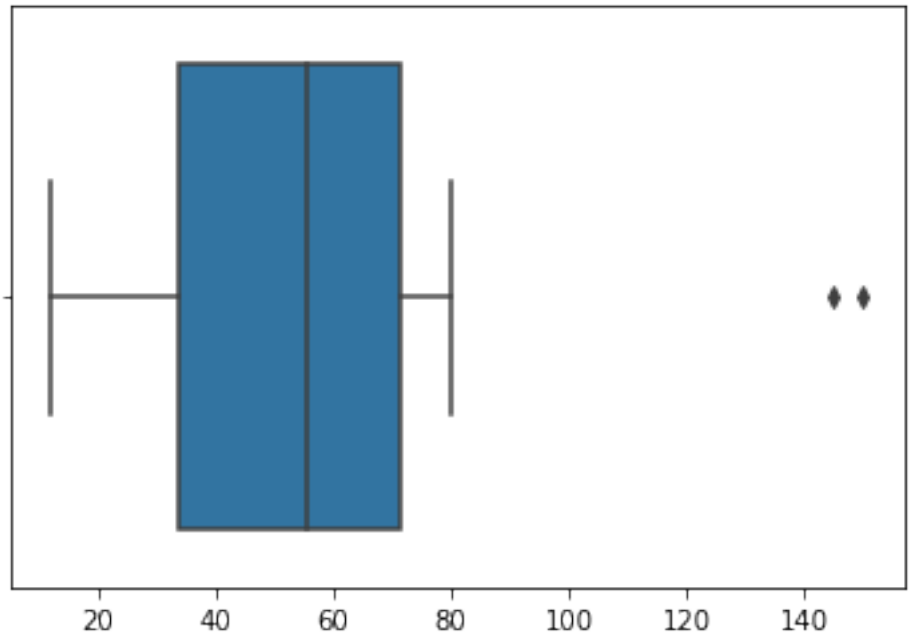
[25]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
dataset = [12,35,40,60,70,80,25,41,20,30,145,55,68,75,56,150]
sorted(dataset)
quantile1, quantile3  = np.percentile(dataset, [25,75])
print(quantile1, quantile3)
iqr_value = (quantile3 - quantile1)
print(iqr_value)
lower_bound_value = quantile1 - (1.5*iqr_value)
upper_bound_value = quantile3 + (1.5*iqr_value)
print(lower_bound_value, upper_bound_value)
#find outliers
outlier =[]
for x in dataset:
    if ((x> upper_bound_value) or (x<lower_bound_value)):
         outlier.append(x)
print(' outlier in the dataset is', outlier)
```

```
33.75 71.25
37.5
-22.5 127.5
 outlier in the dataset is [145, 150]
```

[32]:
```
import seaborn as sns
sns.boxplot(dataset)
```

```
C:\Users\Pccoe\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[32]: <AxesSubplot:>



[30]:
```
dict1={"Roll_No":[1,2,3,4,5],"Name":
  ↪["Amol","Dipak","Shreya","Krisha","Pooja"],"Maths_marks":
  ↪[60,70,80,90,50],"seminar":[45,100,45,50,120],"History_marks":
  ↪[90,60,60,70,80],}
```

[86]:
```
df1=pd.DataFrame(dict1)
```

[31]:
```
result = df1.transform(func = ['sqrt', 'exp'])
result
```

[31]:

	Roll_No		Maths_marks		English_marks		\
	sqrt	exp	sqrt	exp	sqrt	exp	
0	1.000000	2.718282	7.745967	1.142007e+26	8.366600	2.515439e+30	
1	1.414214	7.389056	8.366600	2.515439e+30	8.944272	5.540622e+34	
2	1.732051	20.085537	8.944272	5.540622e+34	6.324555	2.353853e+17	
3	2.000000	54.598150	9.643651	2.451246e+40	NaN	NaN	
4	2.236068	148.413159	NaN	NaN	7.071068	5.184706e+21	

	Science_marks		History_marks		Geography_marks	\
	sqrt	exp	sqrt	exp	sqrt	
0	8.944272	5.540622e+34	8.717798	1.014800e+33	NaN	
1	9.539392	3.317400e+39	NaN	NaN	7.874008	
2	NaN	NaN	8.246211	3.404276e+29	8.366600	
3	8.062258	1.694889e+28	8.366600	2.515439e+30	8.944272	
4	8.366600	2.515439e+30	8.944272	5.540622e+34	9.486833	

```
            exp
0           NaN
1  8.438357e+26
2  2.515439e+30
3  5.540622e+34
4  1.220403e+39
```

[88]:
```
# Min-Max Normalization
df1 = data.drop('species', axis=1)
df_norm = (df-df.min())/(df.max()-df.min())
df_norm = pd.concat((df_norm, data.species), 1)

print("Scaled Dataset Using Pandas")
df_norm.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15780/94325439.py in <module>
      1 # Min-Max Normalization
----> 2 df1 = data.drop('species', axis=1)
      3 df_norm = (df-df.min())/(df.max()-df.min())
      4 df_norm = pd.concat((df_norm, data.species), 1)
      5

NameError: name 'data' is not defined
```

[89]:
```
df1.head()
```

[89]:

	Roll_No	Name	Maths_marks	seminar	History_marks
0	1	Amol	60	45	90
1	2	Dipak	70	100	60
2	3	Shreya	80	45	60
3	4	Krisha	90	50	70
4	5	Pooja	50	120	80

[ ]:

[ ]:

1 OUTLIERS

[ ]:

[31]:
```
# Importing
import sklearn
from sklearn.datasets import load_boston
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
bos_hou = load_boston()

# Create the dataframe
column_name = bos_hou.feature_names
df_boston = pd.DataFrame(bos_hou.data)
df_boston.columns = column_name
df_boston.head()
```

```
Matplotlib is building the font cache; this may take a moment.
```

[31]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	

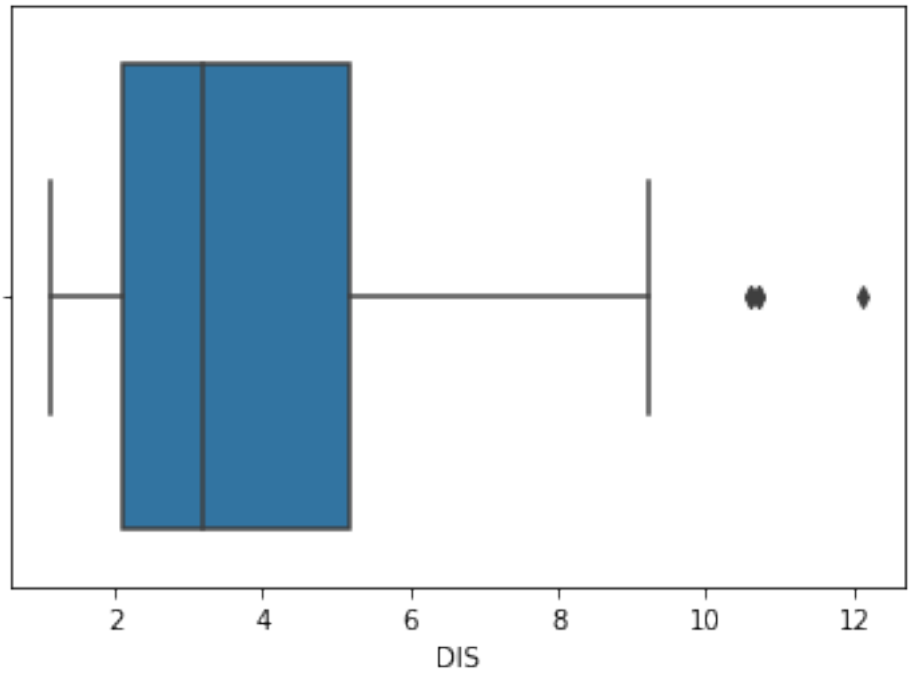
	PTRATIO	B	LSTAT
0	15.3	396.90	4.98
1	17.8	396.90	9.14
2	17.8	392.83	4.03
3	18.7	394.63	2.94
4	18.7	396.90	5.33

2 1. Visualization

[32]:
```
# Box Plot
import seaborn as sns
sns.boxplot(df_boston['DIS'])
```

```
C:\Users\Pccoe\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[32]:
```
<AxesSubplot:xlabel='DIS'>
```



[5]:
```
# Position of the Outlier
print(np.where(df_boston['DIS']>10))
```

```
(array([351, 352, 353, 354, 355], dtype=int64),)
```

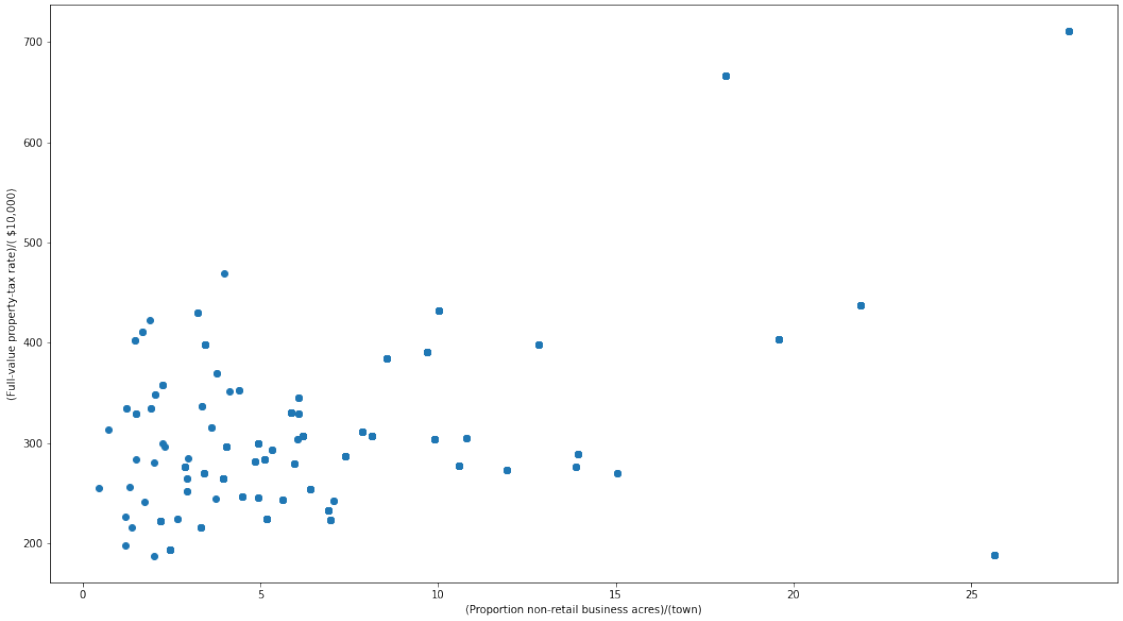
[6]:
```
df_boston.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
```

```
dtypes: float64(13)
memory usage: 51.5 KB
```

3 Using ScatterPlot

[7]:
```
# Scatter plot
fig, ax = plt.subplots(figsize = (18,10))
ax.scatter(df_boston['INDUS'], df_boston['TAX'])

# x-axis label
ax.set_xlabel('(Proportion non-retail business acres)/(town)')

# y-axis label
ax.set_ylabel('(Full-value property-tax rate)/( $10,000)')
plt.show()
```



[8]:
```
# Position of the Outlier
print(np.where((df_boston['INDUS']>20) & (df_boston['TAX']>600)))
```

```
(array([488, 489, 490, 491, 492], dtype=int64),)
```

4 2. Z-score

[9]:
```
# Z score
from scipy import stats
import numpy as np

z = np.abs(stats.zscore(df_boston['DIS']))
print(z)
```

```
0      0.140214
1      0.557160
2      0.557160
3      1.077737
4      1.077737
         …
501    0.625796
502    0.716639
503    0.773684
504    0.668437
505    0.613246
Name: DIS, Length: 506, dtype: float64
```

[10]:
```
threshold = 3

# Position of the outlier
print(np.where(z > 3))
```

```
(array([351, 352, 353, 354, 355], dtype=int64),)
```

5 3. IQR (Inter Quartile Range)

[11]:
```
# IQR
Q1 = np.percentile(df_boston['DIS'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(df_boston['DIS'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1
```

[12]:
```
# Above Upper bound
upper = df_boston['DIS'] >= (Q3+1.5*IQR)

print("Upper bound:",upper)
print(np.where(upper))

# Below Lower bound
lower = df_boston['DIS'] <= (Q1-1.5*IQR)
```

```
print("Lower bound:", lower)
print(np.where(lower))
```

```
Upper bound: 0      False
1      False
2      False
3      False
4      False
       …
501    False
502    False
503    False
504    False
505    False
Name: DIS, Length: 506, dtype: bool
(array([351, 352, 353, 354, 355], dtype=int64),)
Lower bound: 0      False
1      False
2      False
3      False
4      False
       …
501    False
502    False
503    False
504    False
505    False
Name: DIS, Length: 506, dtype: bool
(array([], dtype=int64),)
```

6 Removing the outliers

[52]:
```
# Importing
import sklearn
from sklearn.datasets import load_boston
import pandas as pd

# Load the dataset
bos_hou = load_boston()

# Create the dataframe
column_name = bos_hou.feature_names
df_boston = pd.DataFrame(bos_hou.data)
df_boston.columns = column_name
df_boston.head()
```

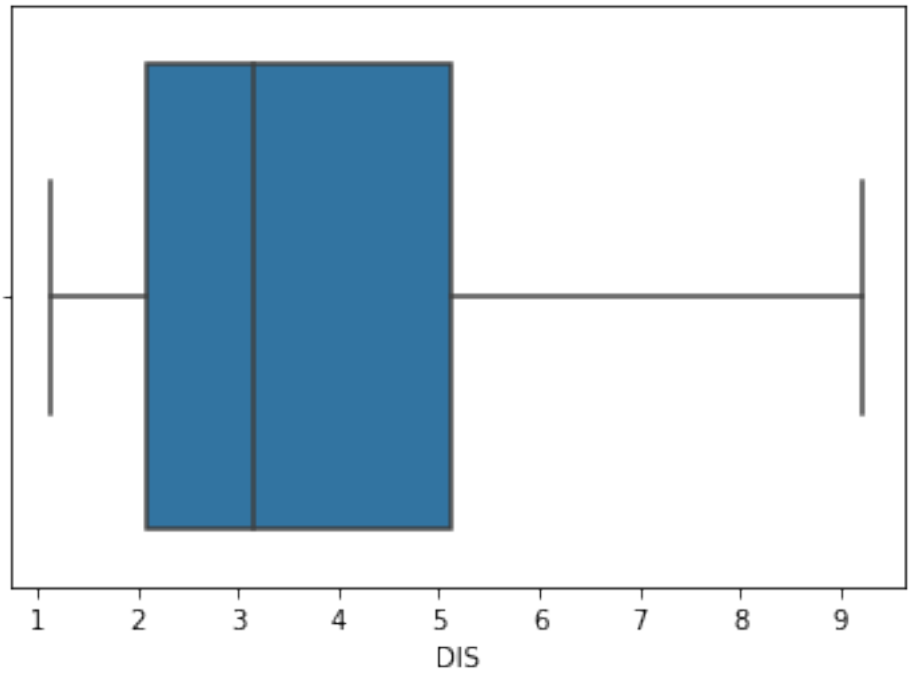
```
''' Detection '''
# IQR
Q1 = np.percentile(df_boston['DIS'], 25,
                                 interpolation = 'midpoint')

Q3 = np.percentile(df_boston['DIS'], 75,
                                 interpolation = 'midpoint')
IQR = Q3 - Q1

print("Old Shape: ", df_boston.shape)

# Upper bound
upper = np.where(df_boston['DIS'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(df_boston['DIS'] <= (Q1-1.5*IQR))

''' Removing the Outliers '''
df_boston.drop(upper[0], inplace = True)
df_boston.drop(lower[0], inplace = True)

print("New Shape: ", df_boston.shape)
```

```
Old Shape:  (506, 13)
New Shape:  (501, 13)
```

[53]:
```
import seaborn as sns
sns.boxplot(df_boston['DIS'])
```

```
C:\Users\Pccoe\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[53]: <AxesSubplot:xlabel='DIS'>



7 data transformations

[21]:
```
# importing pandas as pd
import pandas as pd

# Creating the DataFrame
df = pd.DataFrame({"A":[12, 4, 5, None, 1],
                                "B":[7, 2, 54, 3, None],
                                "C":[20, 16, 11, 3, 8],
                                "D":[14, 3, None, 2, 6]})

# Create the index
index_ = ['Row_1', 'Row_2', 'Row_3', 'Row_4', 'Row_5']

# Set the index
df.index = index_

# Print the DataFrame
print(df)
```

```
          A     B   C     D
Row_1  12.0   7.0  20  14.0
Row_2   4.0   2.0  16   3.0
Row_3   5.0  54.0  11   NaN
```

```
Row_4   NaN   3.0   3   2.0
Row_5   1.0   NaN   8   6.0
```

[23]:
```
# pass a list of functions
result = df.transform(func = ['sqrt', 'exp'])
result
# Print the result
#print(result)
```

[23]:

	A		B		C	\
	sqrt	exp	sqrt	exp	sqrt	
Row_1	3.464102	162754.791419	2.645751	1.096633e+03	4.472136	
Row_2	2.000000	54.598150	1.414214	7.389056e+00	4.000000	
Row_3	2.236068	148.413159	7.348469	2.830753e+23	3.316625	
Row_4	NaN	NaN	1.732051	2.008554e+01	1.732051	
Row_5	1.000000	2.718282	NaN	NaN	2.828427	

	D	
	exp	sqrt	exp
Row_1	4.851652e+08	3.741657	1.202604e+06
Row_2	8.886111e+06	1.732051	2.008554e+01
Row_3	5.987414e+04	NaN	NaN
Row_4	2.008554e+01	1.414214	7.389056e+00
Row_5	2.980958e+03	2.449490	4.034288e+02

8 SCALING

9 Method 1: Using Pandas and Numpy

[33]:
```
import seaborn as sns
import pandas as pd
import numpy as np

data = sns.load_dataset('iris')
print('Original Dataset')
data.head()
```

```
Original Dataset
```

[33]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[35]:
```
# Min-Max Normalization
df = data.drop('species', axis=1)
df_norm = (df-df.min())/(df.max()-df.min())
df_norm = pd.concat((df_norm, data.species), 1)

print("Scaled Dataset Using Pandas")
df_norm.head()
```

```
Scaled Dataset Using Pandas

C:\Users\Pccoe\AppData\Local\Temp/ipykernel_11980/1663693476.py:4:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  df_norm = pd.concat((df_norm, data.species), 1)
```

[35]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	0.222222	0.625000	0.067797	0.041667	setosa
1	0.166667	0.416667	0.067797	0.041667	setosa
2	0.111111	0.500000	0.050847	0.041667	setosa
3	0.083333	0.458333	0.084746	0.041667	setosa
4	0.194444	0.666667	0.067797	0.041667	setosa

[36]:
```
# skewness along the index axis
df_norm.skew(axis = 0, skipna = True)
```

```
C:\Users\Pccoe\AppData\Local\Temp/ipykernel_11980/2697892242.py:2:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise
TypeError.  Select only valid columns before calling the reduction.
  df_norm.skew(axis = 0, skipna = True)
```

[36]:
```
sepal_length    0.314911
sepal_width     0.318966
petal_length   -0.274884
petal_width    -0.102967
dtype: float64
```

[37]:
```
# find skewness in each row
df.skew(axis = 1, skipna = True)
```

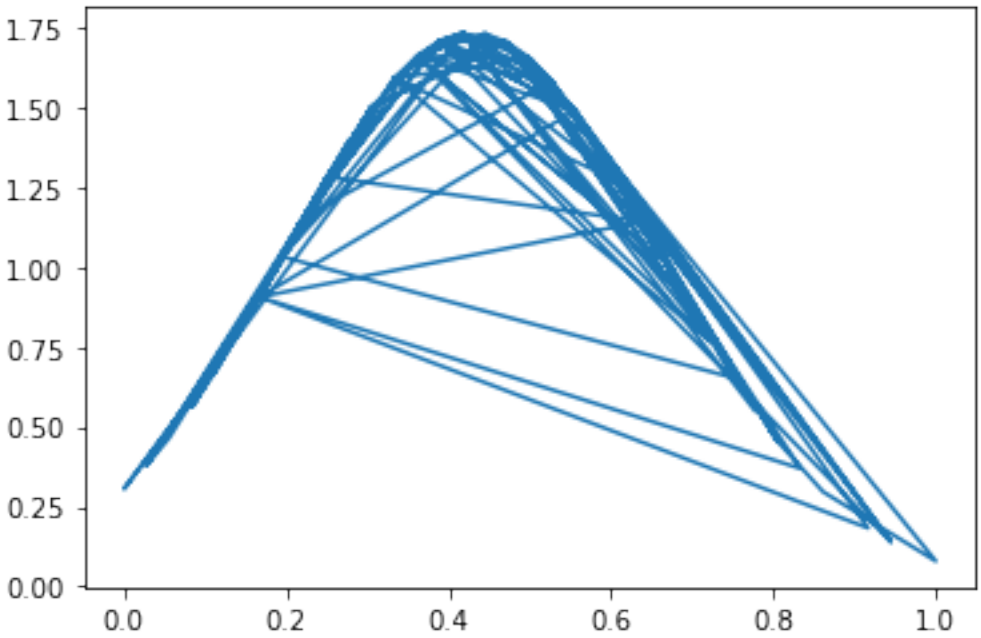
[37]:
```
0      0.189328
1      0.404389
2      0.208782
3      0.120130
4      0.090753
         …
145    0.351514
146    0.256478
```

```
147    0.134795
148   -0.146244
149   -0.185387
Length: 150, dtype: float64
```

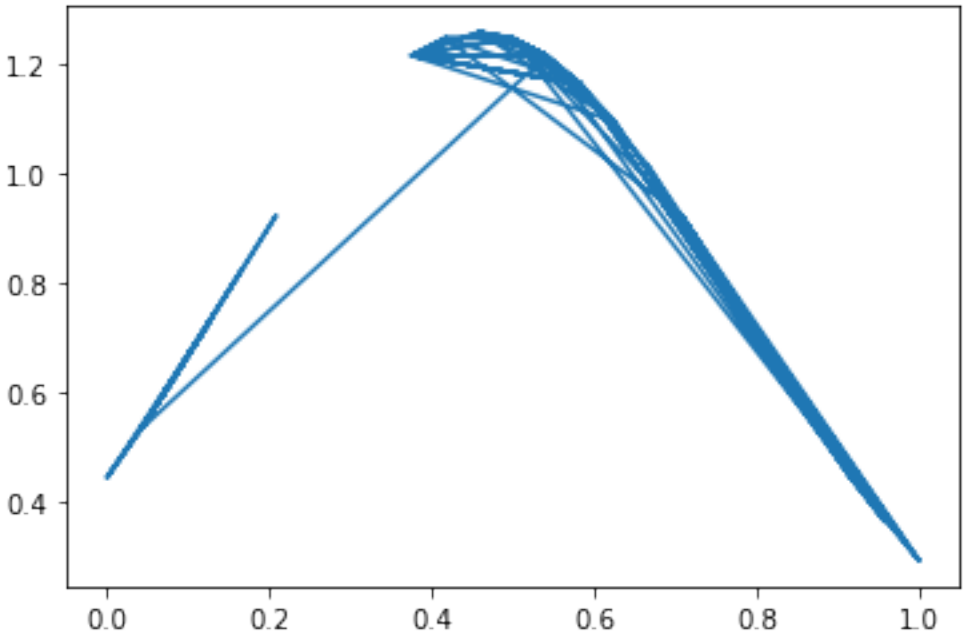
For a unimodal distribution, negative skew commonly indicates that the tail is on the left side of the distribution, and positive skew indicates that the tail is on the right. In cases where one tail is long but the other tail is fat, skewness does not obey a simple rule. For example, a zero value means that the tails on both sides of the mean balance out overall; this is the case for a symmetric distribution, but can also be true for an asymmetric distribution where one tail is long and thin, and the other is short but fat.

10 Distribution

[34]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import statistics



# Calculating mean and standard deviation
mean = statistics.mean(df_norm["sepal_length"])
sd = statistics.stdev(df_norm["sepal_length"])

plt.plot(df_norm["sepal_length"], norm.pdf(df_norm["sepal_length"], mean, sd))
plt.show()
```



[35]:
```
# Calculating mean and standard deviation
mean = statistics.mean(df_norm["petal_width"])
sd = statistics.stdev(df_norm["petal_width"])

plt.plot(df_norm["petal_width"], norm.pdf(df_norm["petal_width"], mean, sd))
plt.show()
```



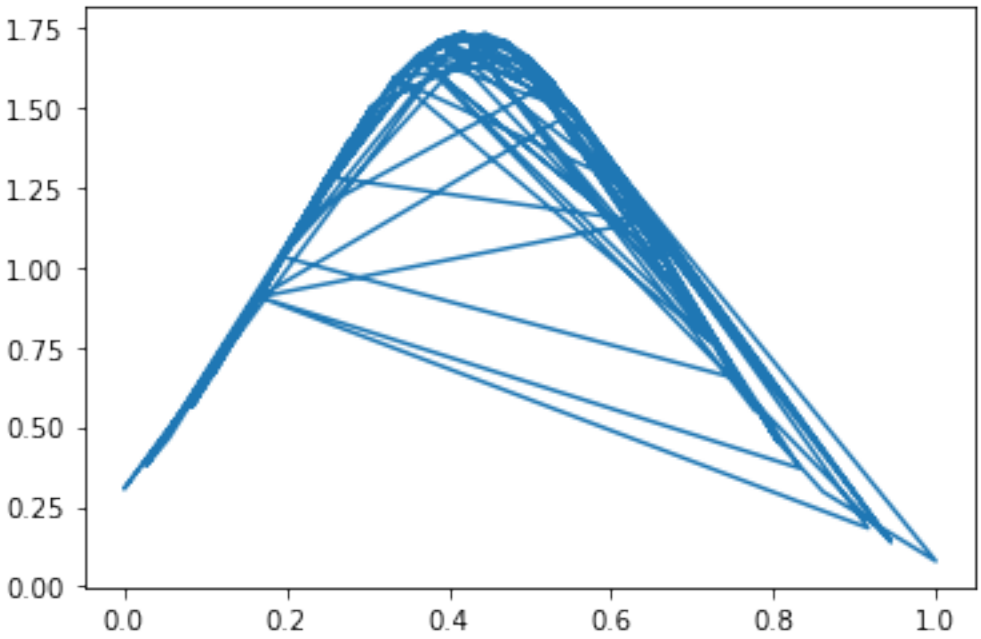
[36]:
```
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

data = sns.load_dataset('iris')
print('Original Dataset')
data.head()

scaler = MinMaxScaler()

df_scaled = scaler.fit_transform(df.to_numpy())
df_scaled = pd.DataFrame(df_scaled, columns=[
'sepal_length', 'sepal_width', 'petal_length', 'petal_width'])

print("Scaled Dataset Using MinMaxScaler")
df_scaled.head()
```

```
Original Dataset
Scaled Dataset Using MinMaxScaler
```

[36]:

	sepal_length	sepal_width	petal_length	petal_width
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667

[38]:
```
# Calculating mean and standard deviation
mean = statistics.mean(df_scaled["sepal_length"])
sd = statistics.stdev(df_scaled["sepal_length"])

plt.plot(df_scaled["sepal_length"], norm.pdf(df_scaled["sepal_length"], mean,␣
 ↪sd))
plt.show()
```



11 Standardization

[13]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
import seaborn as sns

data = sns.load_dataset('iris')
```

```
print('Original Dataset')
data.head()
```

```
Original Dataset
```

[13]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[15]:
```
data.drop(["species"],inplace=True,axis=1)
```

[16]:
```
data
```

[16]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	…	…	…	…
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

```
[150 rows x 4 columns]
```

[17]:
```
std_scaler = StandardScaler()

df_scaled = std_scaler.fit_transform(data.to_numpy())
df_scaled = pd.DataFrame(df_scaled, columns=[
'sepal_length','sepal_width','petal_length','petal_width'])

print("Scaled Dataset Using StandardScaler")
df_scaled.head()
```

```
Scaled Dataset Using StandardScaler
```

[17]:

	sepal_length	sepal_width	petal_length	petal_width
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444

```
4     -1.021849     1.249201     -1.340227     -1.315444
```

Standardization doesn't have any fixed minimum or maximum value. Here, the values of all the columns are scaled in such a way that they all have a mean equal to 0 and standard deviation equal to 1. This scaling technique works well with outliers. Thus, this technique is preferred if outliers are present in the dataset.

[ ]: