caxyrfsok

May 7, 2023

1 exploratory data analysis Algerian Forest Fire Dataset

[2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pylab as plt
%matplotlib inline
```

[3]:
```
data = pd.read_csv(r"../Algerian_forest_fires_dataset_UPDATE.csv",header=1)
data
```

[3]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire
..	..	…	…	…	..	..	…	…	…	…	…	…	…	…
241	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	fire
242	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fire
243	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire
244	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire
245	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire

```
[246 rows x 14 columns]
```

[ ]:
```
data[data.isna().any(axis=1)]
data.iloc[121:125,:]
data.drop([122,123],inplace=True)
data.reset_index(inplace=True)
data.drop(['index',"day","month","year"],axis=1,inplace=True)
data["region"] = None
data.iloc[:122,-1] = "Bejaia"
data.iloc[122:,-1] = "Abbes"
data
```

2 Data cleaning operations

[ ]:
```
data.info()
```

Getting unique values from y data column:
Getting unique values from a column involves identifying and selecting only the distinct or unique values in that column.

[6]:
```
data["Classes  "].unique()
```

[6]:
```
array(['not fire   ', 'fire   ', 'fire', 'fire ', 'not fire', 'not fire ',
       'not fire    ', 'not fire    '], dtype=object)
```

Apply `str.strip()` to clean the data:
As we can see y data has some blank spaces so we need to remove then before use.
I have used the `.strip()` method in Python to remove the leading and trailing spaces from the data in a column.

[7]:
```
data["Classes  "] = data["Classes  "].str.strip()
```

[ ]:
```
data["Classes  "].unique()
```

Convert data type of all data column:
* In below code I am selecting all data which are intiger and making the column data type as float64

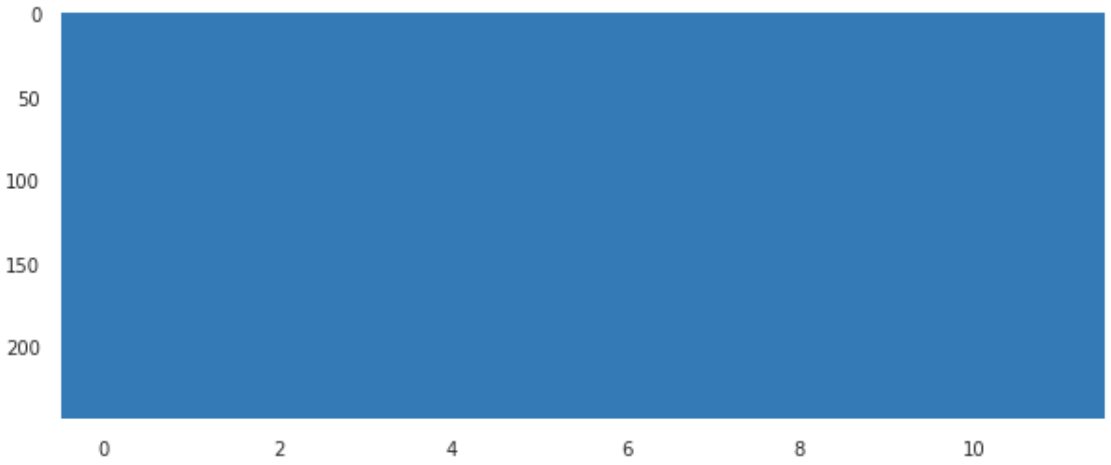
[ ]:
```
columns = data.columns[:-2]
for i in columns:
    data[i] = data[i].astype("float64")
data.info()
```

3 EDA for data set

[10]:
```
from pandas_profiling import ProfileReport
profile = ProfileReport(data, explorative=True)

#Saving results to a HTML file
profile.to_file("pandas_profiling.html")
```

```
Summarize dataset:  96%|     | 120/125 [00:17<00:00,  8.56it/s, Missing
diagram matrix]         /home/sanjiv/anaconda3/lib/python3.9/site-
packages/pandas_profiling/model/missing.py:89: UserWarning: There was an attempt
to generate the Matrix missing values diagrams, but this failed.
To hide this warning, disable the calculation
(using `df.profile_report(missing_diagrams={"Matrix": False}`)
If this is problematic for your use case, please report this as an issue:
https://github.com/ydataai/pandas-profiling/issues
(include the error message: 'keyword grid_b is not recognized; valid keywords
are ['size', 'width', 'color', 'tickdir', 'pad', 'labelsize', 'labelcolor',
'zorder', 'gridOn', 'tick1On', 'tick2On', 'label1On', 'label2On', 'length',
'direction', 'left', 'bottom', 'right', 'top', 'labelleft', 'labelbottom',
'labelright', 'labeltop', 'labelrotation', 'grid_agg_filter', 'grid_alpha',
'grid_animated', 'grid_antialiased', 'grid_clip_box', 'grid_clip_on',
'grid_clip_path', 'grid_color', 'grid_dash_capstyle', 'grid_dash_joinstyle',
'grid_dashes', 'grid_data', 'grid_drawstyle', 'grid_figure', 'grid_fillstyle',
'grid_gapcolor', 'grid_gid', 'grid_in_layout', 'grid_label', 'grid_linestyle',
'grid_linewidth', 'grid_marker', 'grid_markeredgecolor', 'grid_markeredgewidth',
'grid_markerfacecolor', 'grid_markerfacecoloralt', 'grid_markersize',
'grid_markevery', 'grid_mouseover', 'grid_path_effects', 'grid_picker',
'grid_pickradius', 'grid_rasterized', 'grid_sketch_params', 'grid_snap',
'grid_solid_capstyle', 'grid_solid_joinstyle', 'grid_transform', 'grid_url',
'grid_visible', 'grid_xdata', 'grid_ydata', 'grid_zorder', 'grid_aa', 'grid_c',
'grid_ds', 'grid_ls', 'grid_lw', 'grid_mec', 'grid_mew', 'grid_mfc',
'grid_mfcalt', 'grid_ms']')
  warnings.warn(
Summarize dataset: 100%|     | 125/125 [00:17<00:00,  6.96it/s, Completed]
Generate report structure: 100%|     | 1/1 [00:04<00:00,  4.08s/it]
Render HTML: 100%|     | 1/1 [00:01<00:00,  1.52s/it]
Export report to file: 100%|     | 1/1 [00:00<00:00, 27.59it/s]
```



[11]:
```
import sweetviz as sv

#EDA using Autoviz
sweet_report = sv.analyze(data)

#Saving results to HTML file
sweet_report.show_html('sweet_report.html')
```

```
Done! Use 'show' commands to display/save.   |      | [100%]   00:00 ->
(00:00 left)

Report sweet_report.html was generated! NOTEBOOK/COLAB USERS: the web browser
MAY not pop up, regardless, the report IS saved in your notebook/colab files.
```

[11]:
```
data.columns
```

[11]:
```
Index(['Temperature', ' RH', ' Ws', 'Rain ', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI',
       'FWI', 'Classes  ', 'region'],
      dtype='object')
```

[12]:
```
data.describe()
```

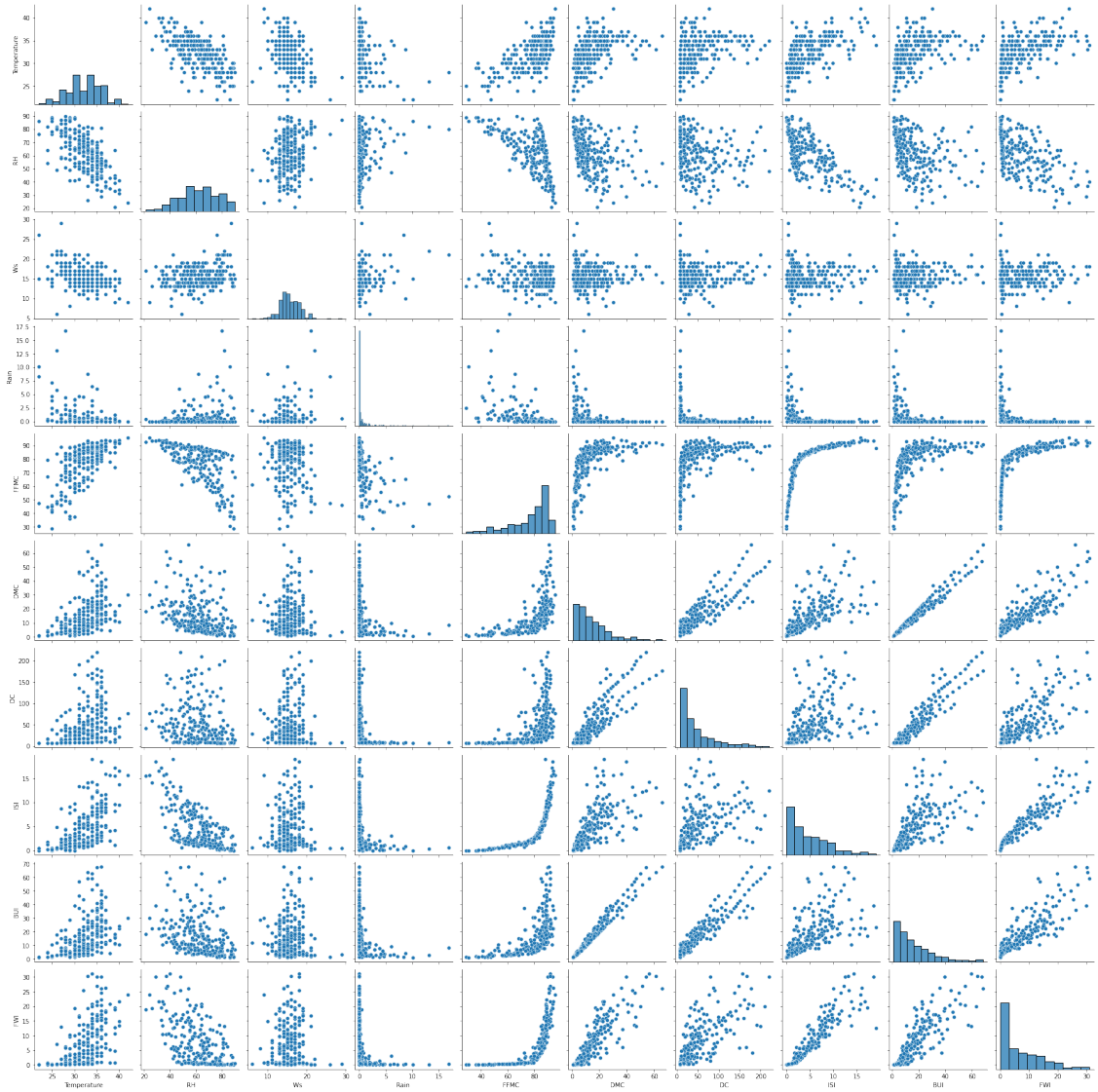
[12]:

	Temperature	RH	Ws	Rain	FFMC	\
count	244.000000	244.000000	244.000000	244.000000	244.000000	
mean	32.172131	61.938525	15.504098	0.760656	77.887705	
std	3.633843	14.884200	2.810178	1.999406	14.337571	
min	22.000000	21.000000	6.000000	0.000000	28.600000	
25%	30.000000	52.000000	14.000000	0.000000	72.075000	
50%	32.000000	63.000000	15.000000	0.000000	83.500000	
75%	35.000000	73.250000	17.000000	0.500000	88.300000	
max	42.000000	90.000000	29.000000	16.800000	96.000000	

	DMC	DC	ISI	BUI	FWI
count	244.000000	244.000000	244.000000	244.000000	244.000000
mean	14.673361	49.288115	4.759836	16.673361	7.049180
std	12.368039	47.619662	4.154628	14.201648	7.428366
min	0.700000	6.900000	0.000000	1.100000	0.000000
25%	5.800000	13.275000	1.400000	6.000000	0.700000
50%	11.300000	33.100000	3.500000	12.450000	4.450000
75%	20.750000	68.150000	7.300000	22.525000	11.375000
max	65.900000	220.400000	19.000000	68.000000	31.100000

[13]:
```
sns.pairplot(data)
```

[13]:
```
<seaborn.axisgrid.PairGrid at 0x7f2d9e9e9d00>
```



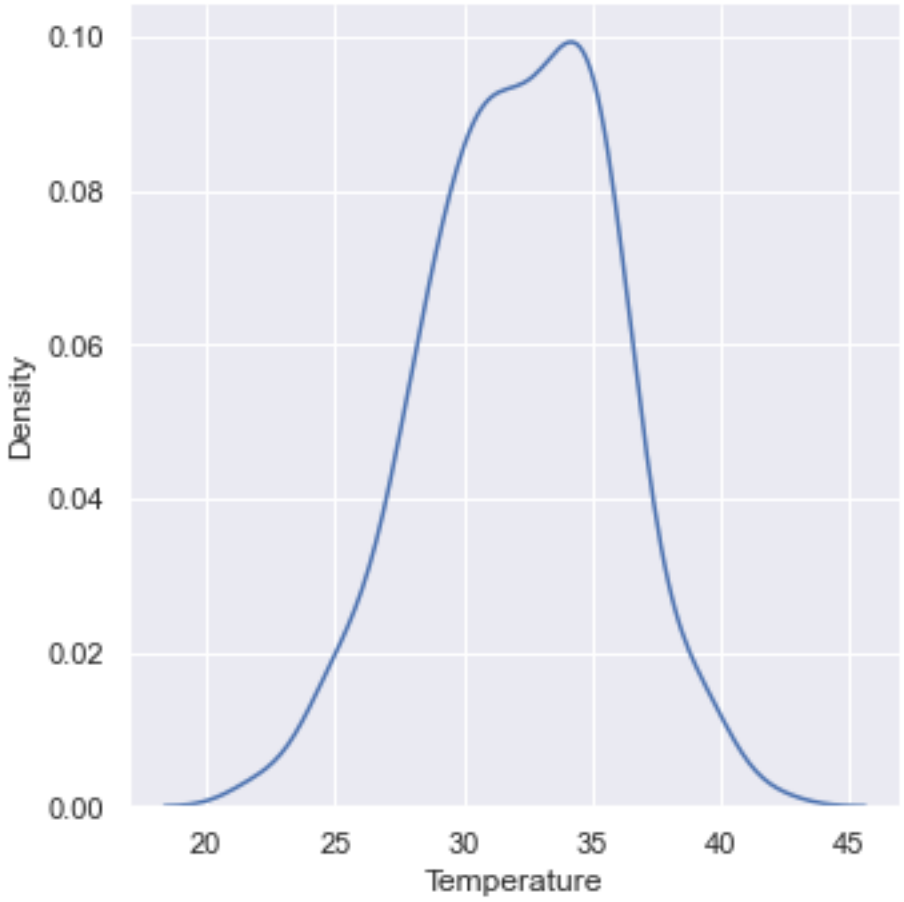
From above `pairplot` we can see `Temperature`, `RH` and `Ws` are seems like normally distributed so we need to do `Normality tests` for same

3.1 univariate analysis

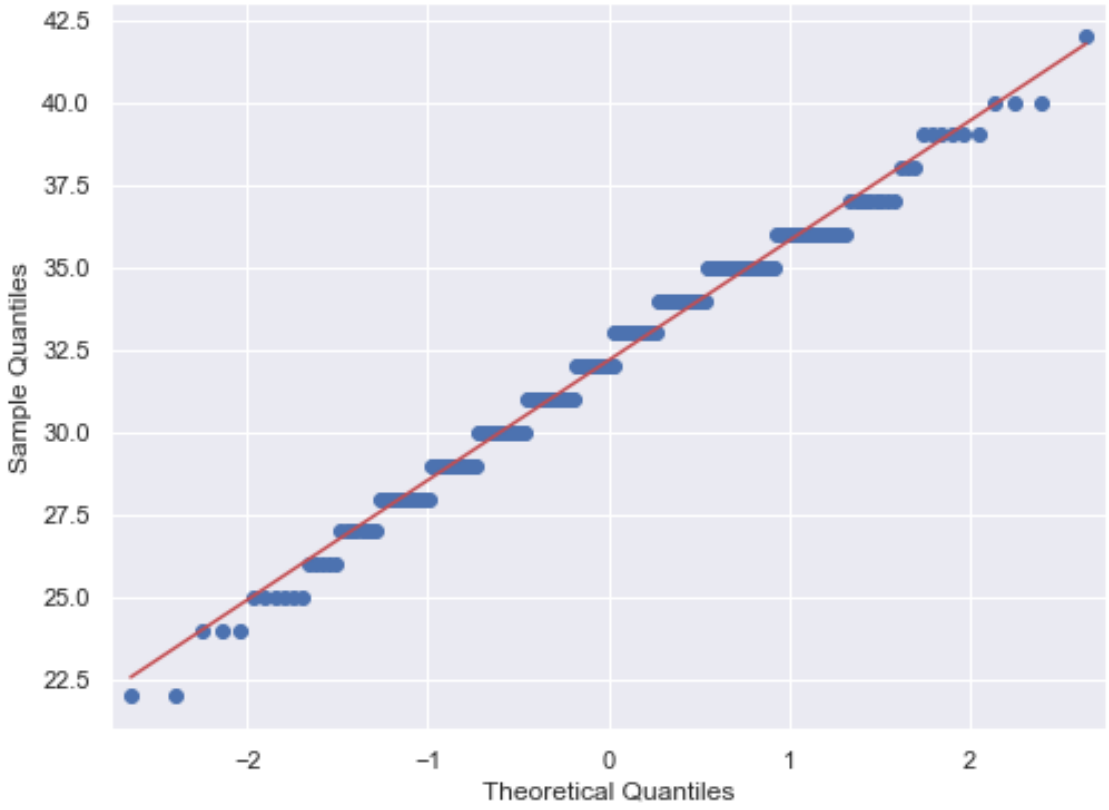
- Univariate analysis is a statistical analysis technique that focuses on analyzing a single variable at a time. It is a type of data analysis that involves examining the distribution, central tendency, and dispersion of a single variable without considering any other variable. Univariate analysis is useful in summarizing and understanding the characteristics of a single variable, such as its range, mean, median, mode, and standard deviation.
- univariate analysis techniques include descriptive statistics such as frequency distribution, histograms, bar charts, and box plots. These techniques can help to identify outliers, missing values, and other patterns in the data.

[14]:
```
# analysis for Temperature
sns.set(rc={'figure.figsize':(8,6)})
sns.displot(data=data["Temperature"],kind="kde")
```

[14]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2d9a8bc5e0>
```



[15]:
```
from numpy.random import seed
from statsmodels.graphics.gofplots import qqplot
from matplotlib import pyplot
# q-q plot
qqplot(data["Temperature"], line='s')
pyplot.show()
```



- The `normaltest` is a statistical test used to determine whether a given sample of data follows a normal distribution or not. The test is based on the null hypothesis that the sample is normally distributed, and the alternative hypothesis that it is not.

[16]:
```
from scipy.stats import normaltest
normaltest(data["Temperature"])
```

[16]:
```
NormaltestResult(statistic=1.7805389205299786, pvalue=0.41054511225166823)
```

- We can see p-value for `Temperature` is `0.41054511225166823` which is greater then 0.05 to we can accept the null hypothesis
- i.e `Temperature` is normally distributed

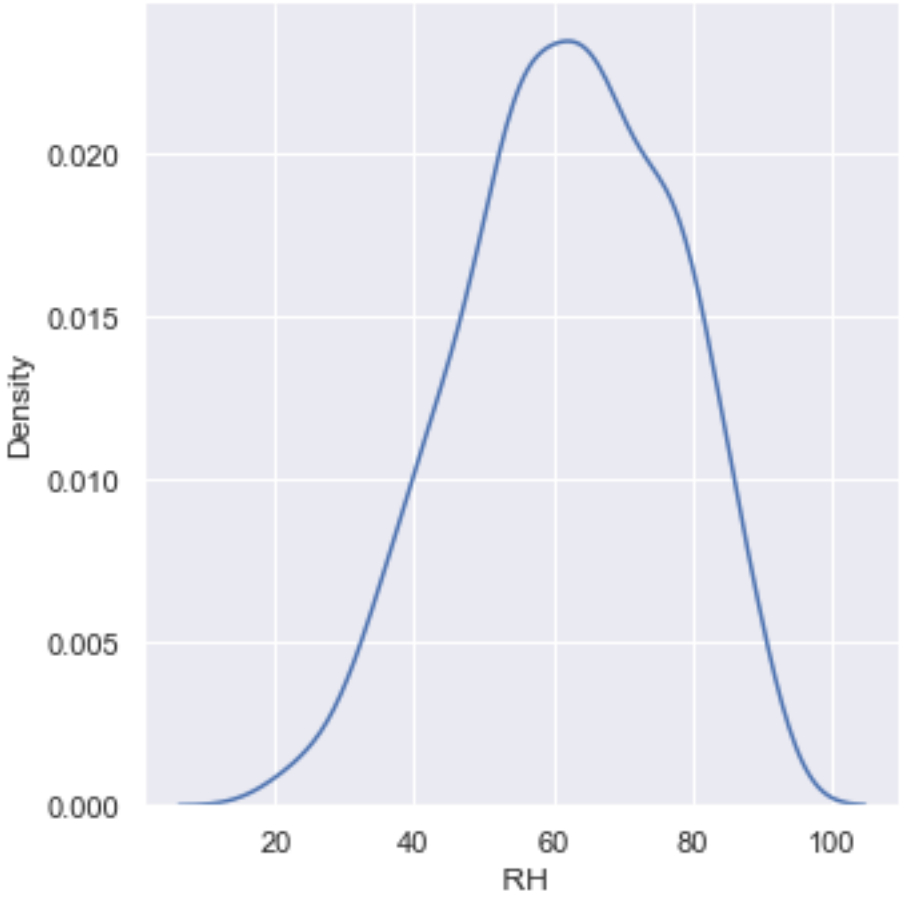
[17]:
```
from scipy.stats import kurtosis
from scipy.stats import skew

print(skew(data["Temperature"],bias=True))
print(kurtosis(data["Temperature"],bias=True))
```

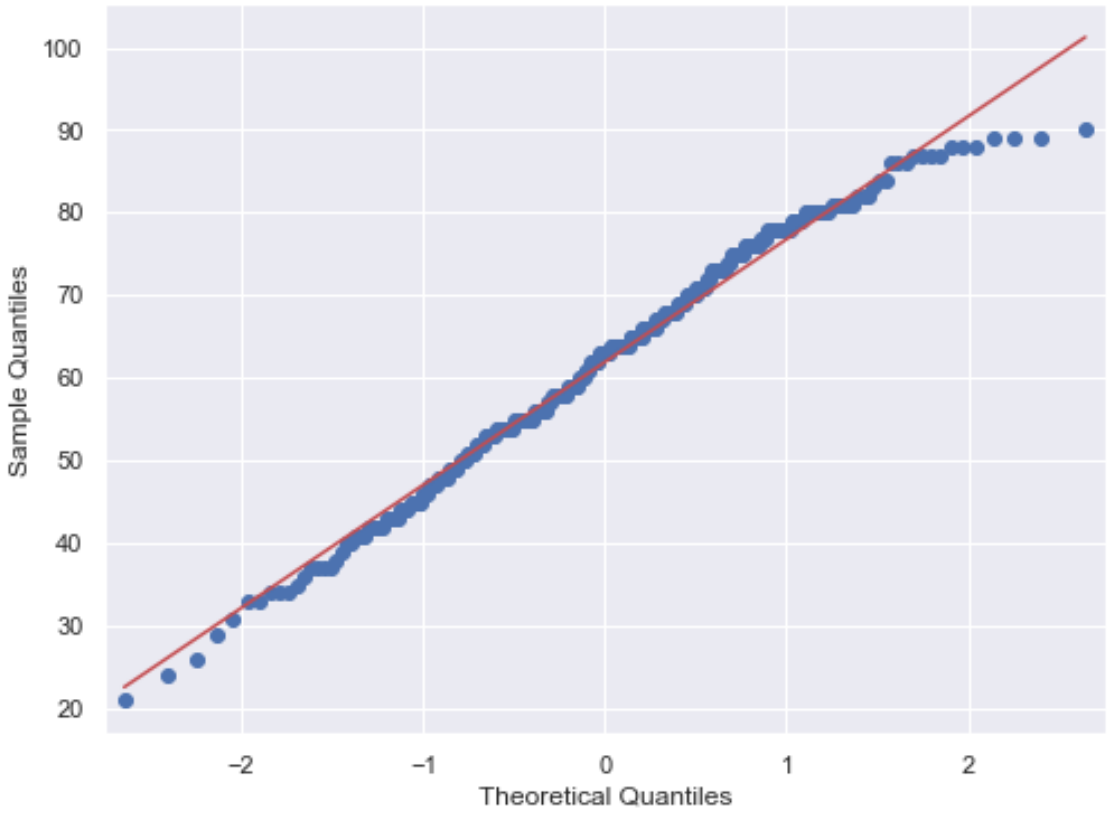
```
-0.1950999958767491
-0.17565616412106388
```

[18]:
```
# analysis for Temperature
sns.set(rc={'figure.figsize':(8,6)})
sns.displot(data=data[" RH"],kind="kde")
```

[18]: <seaborn.axisgrid.FacetGrid at 0x7f2d9825f970>



[19]:
```
qqplot(data[" RH"], line='s')
pyplot.show()
```



[20]:
```
from scipy.stats import normaltest
normaltest(data[" RH"])
```

[20]:
```
NormaltestResult(statistic=7.260230503875478, pvalue=0.0265131285433602)
```

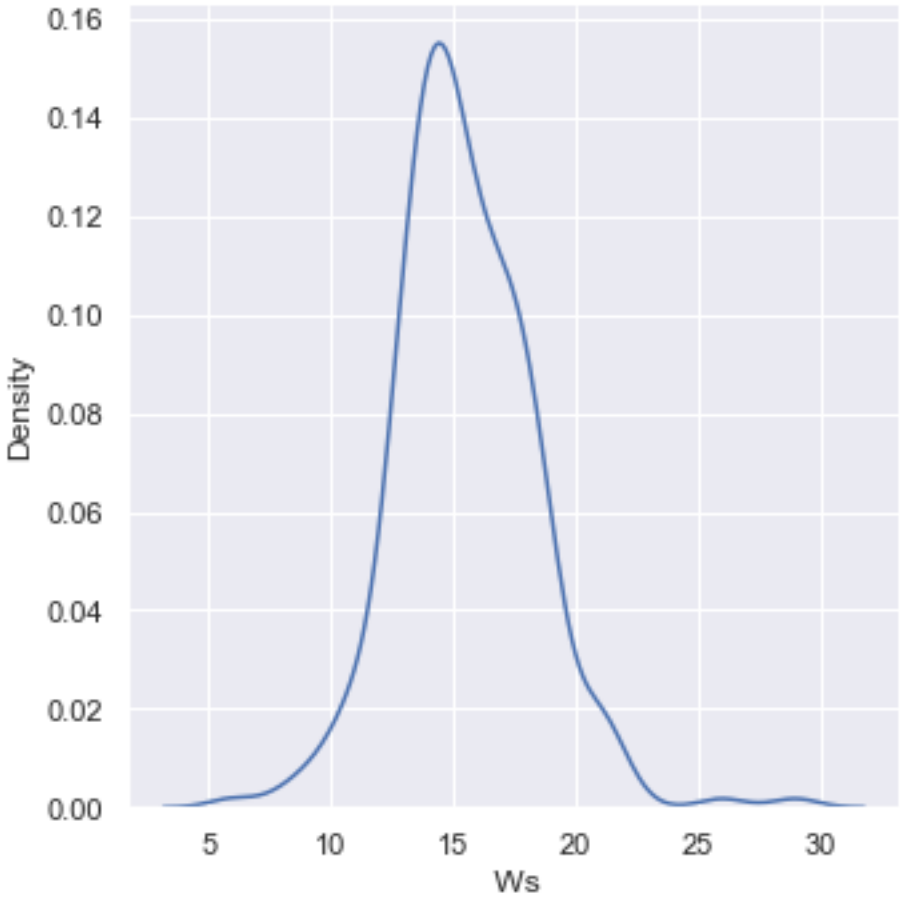
[21]:
```
print(skew(data[" RH"],bias=True))
print(kurtosis(data[" RH"],bias=True))
```

```
-0.2364989921040004
-0.5440124652305531
```

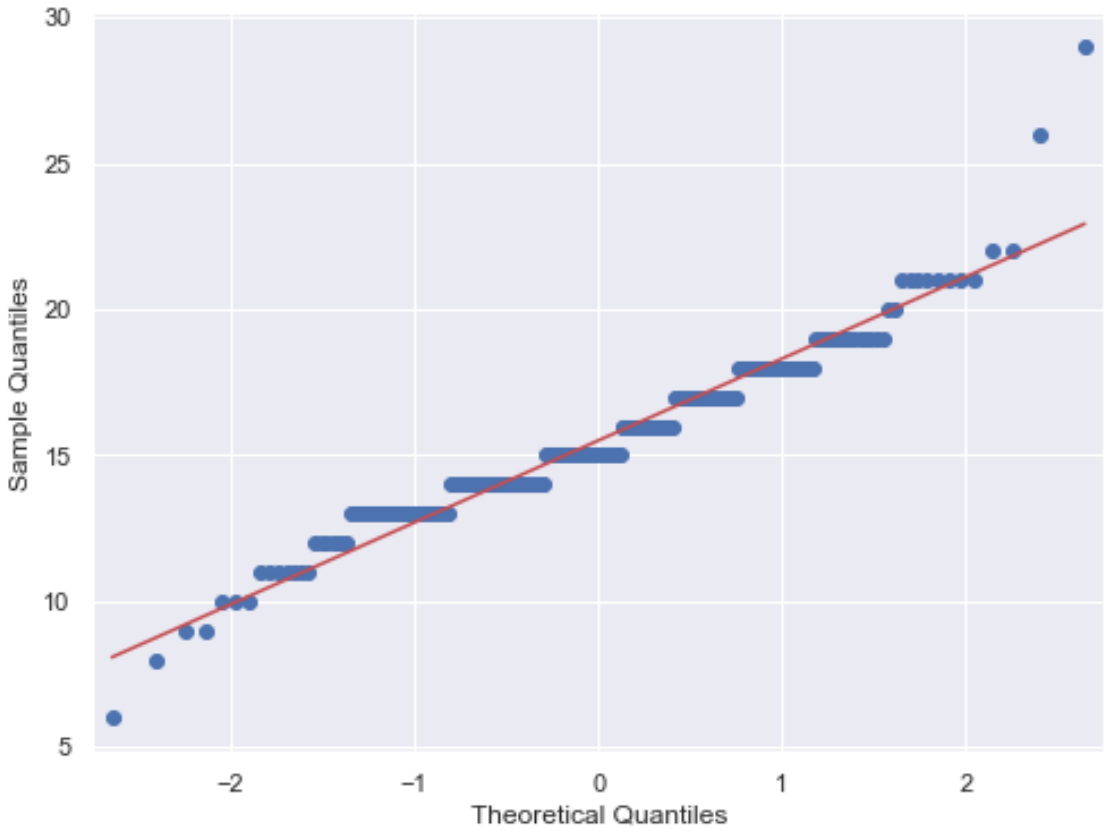
- We can see p-value for `RH` is `0.0265131285433602` which is less then 0.05 to we can reject the null hypothesis
- i.e `RH` is not normally distributed

[22]:
```
# analysis for Ws
sns.set(rc={'figure.figsize':(8,6)})
sns.displot(data=data[" Ws"],kind="kde")
```

[22]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2d9b3a6430>
```



[23]:
```
qqplot(data[" Ws"], line='s')
pyplot.show()
```



[24]:
```
from scipy.stats import normaltest
normaltest(data[" Ws"])
```

[24]:
```
NormaltestResult(statistic=30.110834461628137, pvalue=2.894112266037264e-07)
```

[25]:
```
print(skew(data[" Ws"],bias=True))
print(kurtosis(data[" Ws"],bias=True))
```

```
0.5425196754701939
2.5246482239889394
```

- We can see p-value for `Ws` is `2.894112266037264e-07` which is less then 0.05 to we can reject the null hypothesis
- i.e `Ws` is not normally distributed

3.2 Biveriot analysis

[26]:
```
data.describe()
```

[26]:

	Temperature	RH	Ws	Rain	FFMC	\
count	244.000000	244.000000	244.000000	244.000000	244.000000	
mean	32.172131	61.938525	15.504098	0.760656	77.887705	
std	3.633843	14.884200	2.810178	1.999406	14.337571	
min	22.000000	21.000000	6.000000	0.000000	28.600000	
25%	30.000000	52.000000	14.000000	0.000000	72.075000	
50%	32.000000	63.000000	15.000000	0.000000	83.500000	
75%	35.000000	73.250000	17.000000	0.500000	88.300000	
max	42.000000	90.000000	29.000000	16.800000	96.000000	

	DMC	DC	ISI	BUI	FWI
count	244.000000	244.000000	244.000000	244.000000	244.000000
mean	14.673361	49.288115	4.759836	16.673361	7.049180
std	12.368039	47.619662	4.154628	14.201648	7.428366
min	0.700000	6.900000	0.000000	1.100000	0.000000
25%	5.800000	13.275000	1.400000	6.000000	0.700000
50%	11.300000	33.100000	3.500000	12.450000	4.450000
75%	20.750000	68.150000	7.300000	22.525000	11.375000
max	65.900000	220.400000	19.000000	68.000000	31.100000

[27]:
```
data.corr()
```

[27]:

	Temperature	RH	Ws	Rain	FFMC	DMC	\
Temperature	1.000000	-0.654443	-0.278132	-0.326786	0.677491	0.483105	
RH	-0.654443	1.000000	0.236084	0.222968	-0.645658	-0.405133	
Ws	-0.278132	0.236084	1.000000	0.170169	-0.163255	-0.001246	
Rain	-0.326786	0.222968	0.170169	1.000000	-0.544045	-0.288548	
FFMC	0.677491	-0.645658	-0.163255	-0.544045	1.000000	0.602391	
DMC	0.483105	-0.405133	-0.001246	-0.288548	0.602391	1.000000	
DC	0.370498	-0.220330	0.076245	-0.296804	0.503910	0.875358	
ISI	0.605971	-0.688268	0.012245	-0.347862	0.740751	0.678355	
BUI	0.456415	-0.349685	0.030303	-0.299409	0.590251	0.982206	
FWI	0.566839	-0.580457	0.033957	-0.324755	0.691430	0.875191	

	DC	ISI	BUI	FWI
Temperature	0.370498	0.605971	0.456415	0.566839
RH	-0.220330	-0.688268	-0.349685	-0.580457
Ws	0.076245	0.012245	0.030303	0.033957
Rain	-0.296804	-0.347862	-0.299409	-0.324755
FFMC	0.503910	0.740751	0.590251	0.691430
DMC	0.875358	0.678355	0.982206	0.875191
DC	1.000000	0.503919	0.941672	0.737041
ISI	0.503919	1.000000	0.641351	0.922422
BUI	0.941672	0.641351	1.000000	0.856912
FWI	0.737041	0.922422	0.856912	1.000000

[28]:
```
sns.heatmap(data.corr(), cmap="YlGnBu", annot=True)
plt.show()
```