DATA CLEANNING EXERCISE

```
In [118]: import pandas as pd
```

```
In [119]: import numpy as np
```

```
In [120]: run_times = pd.read_excel('Run Times.xlsx')
```

```
In [123]: run_times
```

Out[123]:

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee
0	Alexis	9.2343	3.5	“school”	2023-04-15 12:00:00	2023-06-01	False	$0.00
1	Alexis	10.3842	3.5	School	2023-04-22 12:30:00	2023-06-01	True	$0.00
2	Alexis	8.1209	3 min	“the gym”	2023-05-10 15:00:00	2023-06-01	False	$2.50
3	David	7.2123	2.2	“school”	2023-05-01 15:15:00	2023-06-15	False	$0.00
4	David	6.8342	2	“gym”	2023-05-10 16:30:00	2023-06-15	False	$2.50

```
In [124]: run_times.dtypes
```

```
Out[124]: Name                    object
          Run Time               float64
          Warm Up Time            object
          Location                object
          Run Date        datetime64[ns]
          Race Date       datetime64[ns]
          Rain                      bool
          Fee                     object
          dtype: object
```

In [125]:
```
run_times.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Name          5 non-null      object
 1   Run Time      5 non-null      float64
 2   Warm Up Time  5 non-null      object
 3   Location      5 non-null      object
 4   Run Date      5 non-null      datetime64[ns]
 5   Race Date     5 non-null      datetime64[ns]
 6   Rain          5 non-null      bool
 7   Fee           5 non-null      object
dtypes: bool(1), datetime64[ns](2), float64(1), object(4)
memory usage: 413.0+ bytes
```

In [126]:
```
run_times.head()
```

Out[126]:

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee
0	Alexis	9.2343	3.5	“school”	2023-04-15 12:00:00	2023-06-01	False	$0.00
1	Alexis	10.3842	3.5	School	2023-04-22 12:30:00	2023-06-01	True	$0.00
2	Alexis	8.1209	3 min	“the gym”	2023-05-10 15:00:00	2023-06-01	False	$2.50
3	David	7.2123	2.2	“school”	2023-05-01 15:15:00	2023-06-15	False	$0.00
4	David	6.8342	2	“gym”	2023-05-10 16:30:00	2023-06-15	False	$2.50

In [127]:
```
# To convert Fee feild to floating (changing object type to string first to av
oide errors)
run_times.Fee = run_times.Fee.astype(str)
```

In [128]:
```
run_times.Fee = pd.to_numeric(run_times.Fee.str.replace('$', '', regex=True))
```

In [129]:
```
run_times.dtypes
```

Out[129]:
```
Name                    object
Run Time               float64
Warm Up Time            object
Location                object
Run Date        datetime64[ns]
Race Date       datetime64[ns]
Rain                      bool
Fee                    float64
dtype: object
```

In [130]:
```
# To change the Warm up time from object data type to numeric
run_times['Warm Up Time'] = run_times['Warm Up Time'].astype(str)
run_times['Warm Up Time'] = pd.to_numeric(run_times['Warm Up Time'].str.replac
e('min', '', regex=True))
```

In [131]:
```
run_times.dtypes
```

Out[131]:
```
Name                     object
Run Time                float64
Warm Up Time            float64
Location                 object
Run Date         datetime64[ns]
Race Date        datetime64[ns]
Rain                       bool
Fee                     float64
dtype: object
```

In [132]:
```
run_times.head(2)
```

Out[132]:

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee
0	Alexis	9.2343	3.5	“school”	2023-04-15 12:00:00	2023-06-01	False	0.0
1	Alexis	10.3842	3.5	School	2023-04-22 12:30:00	2023-06-01	True	0.0

In [133]:
```
# To change the Rain data type to integer
run_times.Rain = run_times.Rain .astype('int')
```

In [134]:
```
run_times.head(2)
```

Out[134]:

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee
0	Alexis	9.2343	3.5	“school”	2023-04-15 12:00:00	2023-06-01	0	0.0
1	Alexis	10.3842	3.5	School	2023-04-22 12:30:00	2023-06-01	1	0.0

Missing Data

In [135]:
```
pd.read_excel('../Projects/Maven_Data/Data/Student Grades.xlsx')
```

Out[135]:

	Student	Class	Year	Grade
0	Emma	Freshman Seminar	Freshman	86.0
1	Olivia	Freshman Seminar	Freshman	86.0
2	Noah	Freshman Seminar	Freshman	86.0
3	Sophia	Freshman Seminar	Freshman	87.0
4	Liam	Freshman Seminar	Freshman	90.0
...	...	...	...	...
81	NaN	NaN	NaN	NaN
82	Bennett	NaN	NaN	NaN
83	NaN	EDA	Junior	84.0
84	Gavin	EDA	Senior	NaN
85	Calvin	NaN	NaN	100.0

86 rows × 4 columns

In [136]:
```
df = pd.read_excel('../Projects/Maven_Data/Data/Student Grades.xlsx')
```

In [137]:
```
# To check if there any null values
df.isna().sum()
```

Out[137]:
```
Student     2
Class       3
Year        6
Grade       4
dtype: int64
```

In [138]:
```
# To see in detail
df.isna().any(axis=1)
```

Out[138]:
```
0      False
1      False
2      False
3      False
4      False
       ...  
81      True
82      True
83      True
84      True
85      True
Length: 86, dtype: bool
```

In [139]:
```
#change the formula to
# To see in detail
df[df.isna().any(axis=1)]
```

Out[139]:

	Student	Class	Year	Grade
7	Jacob	Freshman Seminar	NaN	88.0
8	William	Freshman Seminar	NaN	89.0
9	Ethan	Freshman Seminar	NaN	86.0
62	Landon	Exploratory Data Analysis	Junior	NaN
81	NaN	NaN	NaN	NaN
82	Bennett	NaN	NaN	NaN
83	NaN	EDA	Junior	84.0
84	Gavin	EDA	Senior	NaN
85	Calvin	NaN	NaN	100.0

In [140]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 86 entries, 0 to 85
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Student  84 non-null     object 
 1   Class    83 non-null     object 
 2   Year     80 non-null     object 
 3   Grade    82 non-null     float64
dtypes: float64(1), object(3)
memory usage: 2.8+ KB
```

In [141]:
```
df.count()
```

Out[141]:
```
Student    84
Class      83
Year       80
Grade      82
dtype: int64
```

In [142]:
```
import numpy as np # ways to recognize missing(null) values
```

In [143]:
```
np.NaN
```

Out[143]:
```
nan
```

In [144]:
```
None
```

In [145]:
```
# To check the unique values
df.Year.value_counts()
```

Out[145]:
```
Freshman     35
Sophomore    24
Junior       20
Senior        1
Name: Year, dtype: int64
```

In [146]:
```
# to check the if there any null values within the column
df.Year.value_counts(dropna=False)
```

Out[146]:
```
Freshman     35
Sophomore    24
Junior       20
NaN           6
Senior        1
Name: Year, dtype: int64
```

In [147]:
```
# This formula removes if there any NaN values available in the column#it has droped everything because columns has at least
#one missing value so I'm not going to save any changes to the dataframe here
df[df.isna().any(axis=1)].dropna()
```

Out[147]:

Student	Class	Year	Grade

In [148]:
```
# I'm going to drop the null values in student column and the class column
df[df.isna().any(axis=1)].dropna(subset=['Student', 'Class'])
```

Out[148]:

	Student	Class	Year	Grade
7	Jacob	Freshman Seminar	NaN	88.0
8	William	Freshman Seminar	NaN	89.0
9	Ethan	Freshman Seminar	NaN	86.0
62	Landon	Exploratory Data Analysis	Junior	NaN
84	Gavin	EDA	Senior	NaN

```
In [149]: # To drop those Nan values from the entire data frame
          df.dropna(subset=['Student', 'Class'])
```

Out[149]:

	Student	Class	Year	Grade
0	Emma	Freshman Seminar	Freshman	86.0
1	Olivia	Freshman Seminar	Freshman	86.0
2	Noah	Freshman Seminar	Freshman	86.0
3	Sophia	Freshman Seminar	Freshman	87.0
4	Liam	Freshman Seminar	Freshman	90.0
...	...	...	...	...
77	Aaron	EDA	Junior	85.0
78	Charles	EDA	Junior	93.0
79	Connor	EDA	Junior	91.0
80	Riley	EDA	Junior	87.0
84	Gavin	EDA	Senior	NaN

82 rows × 4 columns

```
In [150]: # This is what I got in the original dataset
          df.shape

Out[150]: (86, 4)
```

```
In [151]: # To Add the values without the nulls I use inpalce=True
          df.dropna(subset=['Student', 'Class'], inplace=True)
```

```
In [152]: df.shape # Now It's been applied

Out[152]: (82, 4)
```

Imputing Missing Data

```
In [153]: # To chaeck the missing Grades
          df[df.Grade.isna()]
```

Out[153]:

	Student	Class	Year	Grade
62	Landon	Exploratory Data Analysis	Junior	NaN
84	Gavin	EDA	Senior	NaN

In [154]:
```
df[df.Year.isna()]
```

Out[154]:

	Student	Class	Year	Grade
7	Jacob	Freshman Seminar	NaN	88.0
8	William	Freshman Seminar	NaN	89.0
9	Ethan	Freshman Seminar	NaN	86.0

In [155]:
```
# To Fill the  NaN Values using mean
df.Grade.mean()
```

Out[155]:
```
85.55
```

In [156]:
```
# Filling the NaN values
df.Grade.fillna(df.Grade.mean())
```

Out[156]:
```
0     86.00
1     86.00
2     86.00
3     87.00
4     90.00
      ...
77    85.00
78    93.00
79    91.00
80    87.00
84    85.55
Name: Grade, Length: 82, dtype: float64
```

In [157]:
```
# To apply changes
df.Grade.fillna(df.Grade.mean(), inplace=True)
```

In [158]:
```
df.Grade
```

Out[158]:
```
0     86.00
1     86.00
2     86.00
3     87.00
4     90.00
      ...
77    85.00
78    93.00
79    91.00
80    87.00
84    85.55
Name: Grade, Length: 82, dtype: float64
```

```
In [159]: # To chaeck if there any missing values in the data
          df[df.isna().any(axis=1)]
```

Out[159]:

	Student	Class	Year	Grade
7	Jacob	Freshman Seminar	NaN	88.0
8	William	Freshman Seminar	NaN	89.0
9	Ethan	Freshman Seminar	NaN	86.0

```
In [160]: # I can see in year feild, I've nulls so need to further look at that class a
          bit
          df[df.Class == 'Freshman Seminar']
```

Out[160]:

	Student	Class	Year	Grade
0	Emma	Freshman Seminar	Freshman	86.0
1	Olivia	Freshman Seminar	Freshman	86.0
2	Noah	Freshman Seminar	Freshman	86.0
3	Sophia	Freshman Seminar	Freshman	87.0
4	Liam	Freshman Seminar	Freshman	90.0
5	Mason	Freshman Seminar	Freshman	90.0
6	Isabella	Freshman Seminar	Freshman	90.0
7	Jacob	Freshman Seminar	NaN	88.0
8	William	Freshman Seminar	NaN	89.0
9	Ethan	Freshman Seminar	NaN	86.0
10	Ava	Freshman Seminar	Freshman	88.0
11	Michael	Freshman Seminar	Freshman	88.0

```
In [161]: # To update it I use
          df.loc[7, 'Year'] = 'Freshman'
```

In [162]:
```
df[df.Class == 'Freshman Seminar']
```

Out[162]:
```
    Student             Class      Year  Grade
0     Emma   Freshman Seminar  Freshman   86.0
1    Olivia  Freshman Seminar  Freshman   86.0
2     Noah   Freshman Seminar  Freshman   86.0
3    Sophia  Freshman Seminar  Freshman   87.0
4     Liam   Freshman Seminar  Freshman   90.0
5    Mason   Freshman Seminar  Freshman   90.0
6  Isabella  Freshman Seminar  Freshman   90.0
7    Jacob   Freshman Seminar  Freshman   88.0
8   William  Freshman Seminar       NaN   89.0
9    Ethan   Freshman Seminar       NaN   86.0
10     Ava   Freshman Seminar  Freshman   88.0
11 Michael   Freshman Seminar  Freshman   88.0
```

In [163]:
```
# Instead of doint it one by one I use
import numpy as np
df.Year = np.where(df.Year.isna(), 'Freshman', df.Year)
```

In [164]:
```
df[df.Class == 'Freshman Seminar']
```

Out[164]:
```
    Student             Class      Year  Grade
0     Emma   Freshman Seminar  Freshman   86.0
1    Olivia  Freshman Seminar  Freshman   86.0
2     Noah   Freshman Seminar  Freshman   86.0
3    Sophia  Freshman Seminar  Freshman   87.0
4     Liam   Freshman Seminar  Freshman   90.0
5    Mason   Freshman Seminar  Freshman   90.0
6  Isabella  Freshman Seminar  Freshman   90.0
7    Jacob   Freshman Seminar  Freshman   88.0
8   William  Freshman Seminar  Freshman   89.0
9    Ethan   Freshman Seminar  Freshman   86.0
10     Ava   Freshman Seminar  Freshman   88.0
11 Michael   Freshman Seminar  Freshman   88.0
```

Inconsistant Text & Typos

```
In [165]: df.head()
```

Out[165]:

	Student	Class	Year	Grade
0	Emma	Freshman Seminar	Freshman	86.0
1	Olivia	Freshman Seminar	Freshman	86.0
2	Noah	Freshman Seminar	Freshman	86.0
3	Sophia	Freshman Seminar	Freshman	87.0
4	Liam	Freshman Seminar	Freshman	90.0

```
In [166]: df.Class.value_counts(dropna=False)
```

```
Out[166]: Intro to Python                25
          Intro to SQL                   20
          Freshman Seminar               12
          Exploratory Data Analysis      12
          EDA                            12
          Python                          1
          Name: Class, dtype: int64
```

In [167]:
```python
# I can see through the above details EDA and Exploratory Data Analysis, and a
lso Intro to python and python
#Their likely the same clasess
df[df.Class.isin(['Exploratory Data Analysis', 'EDA'])]
```

Out[167]:

	Student	Class	Year	Grade
58	Evelyn	Exploratory Data Analysis	Sophomore	89.00
59	Jack	Exploratory Data Analysis	Sophomore	84.00
60	Ella	Exploratory Data Analysis	Sophomore	200.00
61	Chloe	Exploratory Data Analysis	Sophomore	87.00
62	Landon	Exploratory Data Analysis	Junior	85.55
63	Christian	Exploratory Data Analysis	Junior	77.00
64	Jordan	Exploratory Data Analysis	Junior	83.00
65	Jonathan	Exploratory Data Analysis	Junior	82.00
66	Levi	Exploratory Data Analysis	Junior	91.00
67	Victoria	Exploratory Data Analysis	Junior	90.00
68	Aubrey	Exploratory Data Analysis	Junior	83.00
69	Jaxon	Exploratory Data Analysis	Junior	64.00
70	Julian	EDA	Junior	95.00
71	Grace	EDA	Junior	77.00
72	Isaiah	EDA	Junior	88.00
73	Cameron	EDA	Junior	72.00
74	Eli	EDA	Junior	92.00
75	Angel	EDA	Junior	79.00
76	Zoey	EDA	Junior	91.00
77	Aaron	EDA	Junior	85.00
78	Charles	EDA	Junior	93.00
79	Connor	EDA	Junior	91.00
80	Riley	EDA	Junior	87.00
84	Gavin	EDA	Senior	85.55

```
In [168]: df[df.Class.isin(['Intro to Python', 'Python'])]
```

Out[168]:

	Student	Class	Year	Grade
12	Alexander	Intro to Python	Freshman	85.0
13	Logan	Intro to Python	Freshman	85.0
14	James	Intro to Python	Freshman	82.0
15	Daniel	Intro to Python	Freshman	85.0
16	Elijah	Intro to Python	Freshman	85.0
17	Benjamin	Intro to Python	Freshman	81.0
18	Mia	Intro to Python	Freshman	80.0
19	Mia	Python	Freshman	80.0
20	Jayden	Intro to Python	Freshman	82.0
21	Aiden	Intro to Python	Freshman	86.0
22	Matthew	Intro to Python	Freshman	87.0
23	Emily	Intro to Python	Freshman	78.0
24	Jackson	Intro to Python	Freshman	88.0
25	Lucas	Intro to Python	Freshman	77.0
26	David	Intro to Python	Freshman	74.0
27	Joseph	Intro to Python	Freshman	93.0
28	Abigail	Intro to Python	Freshman	89.0
29	Avery	Intro to Python	Freshman	79.0
30	Anthony	Intro to Python	Freshman	84.0
31	Dylan	Intro to Python	Freshman	84.0
32	Andrew	Intro to Python	Freshman	94.0
33	Carter	Intro to Python	Freshman	95.0
34	Samuel	Intro to Python	Freshman	83.0
35	Gabriel	Intro to Python	Freshman	82.0
36	Joshua	Intro to Python	Freshman	71.0
37	John	Intro to Python	Freshman	50.0

```
In [169]: # I can see that thosee are the same. I'm goinng to join
          np.where(df.Class == 'EDA', 'Exploratory Data Analysis', df.Class)
```

```
Out[169]: array(['Freshman Seminar', 'Freshman Seminar', 'Freshman Seminar',
                 'Freshman Seminar', 'Freshman Seminar', 'Freshman Seminar',
                 'Freshman Seminar', 'Freshman Seminar', 'Freshman Seminar',
                 'Freshman Seminar', 'Freshman Seminar', 'Freshman Seminar',
                 'Intro to Python', 'Intro to Python', 'Intro to Python',
                 'Intro to Python', 'Intro to Python', 'Intro to Python',
                 'Intro to Python', 'Python', 'Intro to Python', 'Intro to Python',
                 'Intro to Python', 'Intro to Python', 'Intro to Python',
                 'Intro to Python', 'Intro to Python', 'Intro to Python',
                 'Intro to Python', 'Intro to Python', 'Intro to Python',
                 'Intro to Python', 'Intro to Python', 'Intro to Python',
                 'Intro to Python', 'Intro to Python', 'Intro to Python',
                 'Intro to Python', 'Intro to SQL', 'Intro to SQL', 'Intro to SQL',
                 'Intro to SQL', 'Intro to SQL', 'Intro to SQL', 'Intro to SQL',
                 'Intro to SQL', 'Intro to SQL', 'Intro to SQL', 'Intro to SQL',
                 'Intro to SQL', 'Intro to SQL', 'Intro to SQL', 'Intro to SQL',
                 'Intro to SQL', 'Intro to SQL', 'Intro to SQL', 'Intro to SQL',
                 'Intro to SQL', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis',
                 'Exploratory Data Analysis', 'Exploratory Data Analysis'], dtype=object)
```

```
In [170]: df.Class = np.where(df.Class == 'EDA', 'Exploratory Data Analysis', df.Class)
```

```
In [171]: df.Class.value_counts()
```

```
Out[171]: Intro to Python              25
          Exploratory Data Analysis    24
          Intro to SQL                 20
          Freshman Seminar             12
          Python                        1
          Name: Class, dtype: int64
```

```
In [172]: df.Class = np.where(df.Class == 'Python', 'Intro to Python', df.Class)
```

```
In [173]: df.Class.value_counts()
```

```
Out[173]: Intro to Python              26
          Exploratory Data Analysis    24
          Intro to SQL                 20
          Freshman Seminar             12
          Name: Class, dtype: int64
```

```
In [174]: df.describe()
```

Out[174]:

	Grade
count	82.000000
mean	85.550000
std	15.443965
min	45.000000
25%	81.000000
50%	85.275000
75%	89.750000
max	200.000000

```
In [175]: # I can see that the max grade is really high neet to further investigate
          df[df.Grade > 100]
```

Out[175]:

	Student	Class	Year	Grade
60	Ella	Exploratory Data Analysis	Sophomore	200.0

```
In [176]: # requirement is those who have > 100 turn into 100 so
          df.Grade = np.where(df.Grade > 100, 100, df.Grade)
```

```
In [177]: df.Grade.value_counts()
```

```
Out[177]: 88.00     6
          85.00     6
          86.00     5
          90.00     5
          84.00     5
          82.00     4
          80.00     4
          87.00     4
          79.00     3
          96.00     3
          83.00     3
          93.00     3
          91.00     3
          77.00     3
          81.00     3
          89.00     3
          95.00     2
          71.00     2
          85.55     2
          76.00     2
          92.00     2
          45.00     1
          64.00     1
          100.00    1
          94.00     1
          98.00     1
          50.00     1
          74.00     1
          78.00     1
          72.00     1
          Name: Grade, dtype: int64
```

```
In [178]: df.describe()
```

Out[178]:

	Grade
count	82.000000
mean	84.330488
std	8.824663
min	45.000000
25%	81.000000
50%	85.275000
75%	89.750000
max	100.000000

Mapping Values

```
In [179]: df1 = pd.read_excel('../Projects/Maven_Data/Data/Student Grades.xlsx')
```

```
In [180]: df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 86 entries, 0 to 85
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Student  84 non-null     object 
 1   Class    83 non-null     object 
 2   Year     80 non-null     object 
 3   Grade    82 non-null     float64
dtypes: float64(1), object(3)
memory usage: 2.8+ KB
```

```
In [181]: df1.head()
```

Out[181]:

	Student	Class	Year	Grade
0	Emma	Freshman Seminar	Freshman	86.0
1	Olivia	Freshman Seminar	Freshman	86.0
2	Noah	Freshman Seminar	Freshman	86.0
3	Sophia	Freshman Seminar	Freshman	87.0
4	Liam	Freshman Seminar	Freshman	90.0

```
In [182]: df1.Class.value_counts()
```

```
Out[182]: Intro to Python                25
          Intro to SQL                   20
          EDA                            13
          Freshman Seminar               12
          Exploratory Data Analysis      12
          Python                          1
          Name: Class, dtype: int64
```

```
In [183]: # Mapping EDA to Exploratory Data Analysis and python to Intro to Python
          Class_mappings = {'Intro to Python': 'Intro to Python',
                            'Intro to SQL': 'Intro to SQL',
                            'EDA' : 'Exploratory Data Analysis',
                            'Exploratory Data Analysis' : 'Exploratory Data Analysis',
                            'python' : 'Intro to Python',
                            'Freshman Seminar' : 'Frshman Seminar'}
```

```
In [184]: df1.Class = df.Class.map(Class_mappings)
```

In [185]:
```
df1.Class.value_counts()
```

Out[185]:
```
Intro to Python             26
Exploratory Data Analysis   24
Intro to SQL                20
Frshman Seminar             12
Name: Class, dtype: int64
```

Handlling Inconsistancies in Data

In [186]:
```
# To check the Location in detail
run_times.Location
```

Out[186]:
```
0     “school”
1       School
2    “the gym”
3     “school”
4        “gym”
Name: Location, dtype: object
```

In [187]:
```
run_times.Location = run_times.Location.str.lower().str.replace('the', '').st
r.strip('“”')
```

In [188]:
```
run_times
```

Out[188]:

	Name	Run Time	Warm Up Time	Location	Run Date	Race Date	Rain	Fee
0	Alexis	9.2343	3.5	school	2023-04-15 12:00:00	2023-06-01	0	0.0
1	Alexis	10.3842	3.5	school	2023-04-22 12:30:00	2023-06-01	1	0.0
2	Alexis	8.1209	3.0	gym	2023-05-10 15:00:00	2023-06-01	0	2.5
3	David	7.2123	2.2	school	2023-05-01 15:15:00	2023-06-15	0	0.0
4	David	6.8342	2.0	gym	2023-05-10 16:30:00	2023-06-15	0	2.5

Handlling Duplicate Data

In [189]:
```
df[df.duplicated(keep=False)] # This will gives the duplicated rows
```

Out[189]:

	Student	Class	Year	Grade
18	Mia	Intro to Python	Freshman	80.0
19	Mia	Intro to Python	Freshman	80.0
42	Isaac	Intro to SQL	Sophomore	96.0
43	Isaac	Intro to SQL	Sophomore	96.0
44	Isaac	Intro to SQL	Sophomore	96.0

In [190]:
```
# To remove all the dulpicates
df.drop_duplicates(inplace=True)
```

In [191]:
```
df[df.duplicated(keep=False)]
```

Out[191]:

Student	Class	Year	Grade

In [192]:
```
# To check the records in between 40-45
df.iloc[40:45, :]
```

Out[192]:

	Student	Class	Year	Grade
41	Charlotte	Intro to SQL	Sophomore	92.0
42	Isaac	Intro to SQL	Sophomore	96.0
45	Harper	Intro to SQL	Sophomore	93.0
46	Ryan	Intro to SQL	Sophomore	76.0
47	Sofia	Intro to SQL	Sophomore	79.0

In [193]:
```
df.reset_index(drop=True)
```

Out[193]:

	Student	Class	Year	Grade
0	Emma	Freshman Seminar	Freshman	86.00
1	Olivia	Freshman Seminar	Freshman	86.00
2	Noah	Freshman Seminar	Freshman	86.00
3	Sophia	Freshman Seminar	Freshman	87.00
4	Liam	Freshman Seminar	Freshman	90.00
...	...	...	...	...
74	Aaron	Exploratory Data Analysis	Junior	85.00
75	Charles	Exploratory Data Analysis	Junior	93.00
76	Connor	Exploratory Data Analysis	Junior	91.00
77	Riley	Exploratory Data Analysis	Junior	87.00
78	Gavin	Exploratory Data Analysis	Senior	85.55

79 rows × 4 columns

In [194]:
```
df.reset_index(drop=True, inplace=True)
```

In [195]:
```
df.iloc[40:45, :]
# now I have the correct order
```

Out[195]:

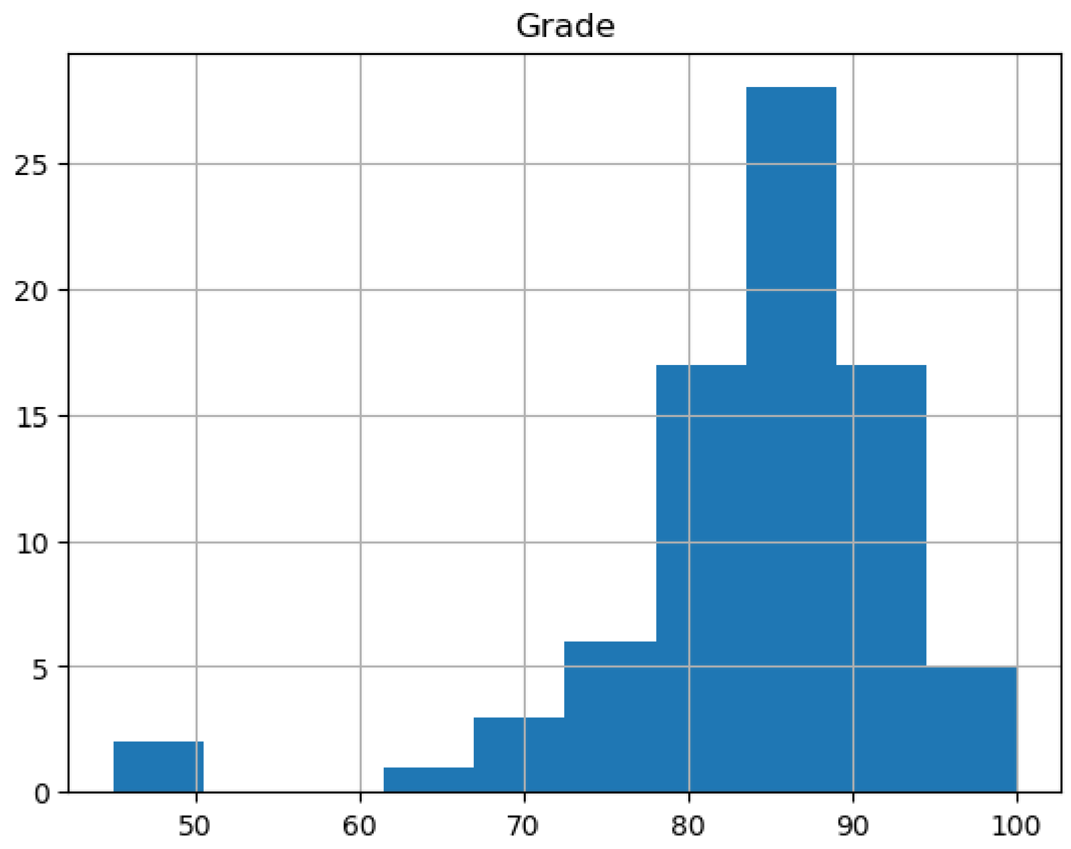
	Student	Class	Year	Grade
40	Charlotte	Intro to SQL	Sophomore	92.0
41	Isaac	Intro to SQL	Sophomore	96.0
42	Harper	Intro to SQL	Sophomore	93.0
43	Ryan	Intro to SQL	Sophomore	76.0
44	Sofia	Intro to SQL	Sophomore	79.0

Outlier Detection

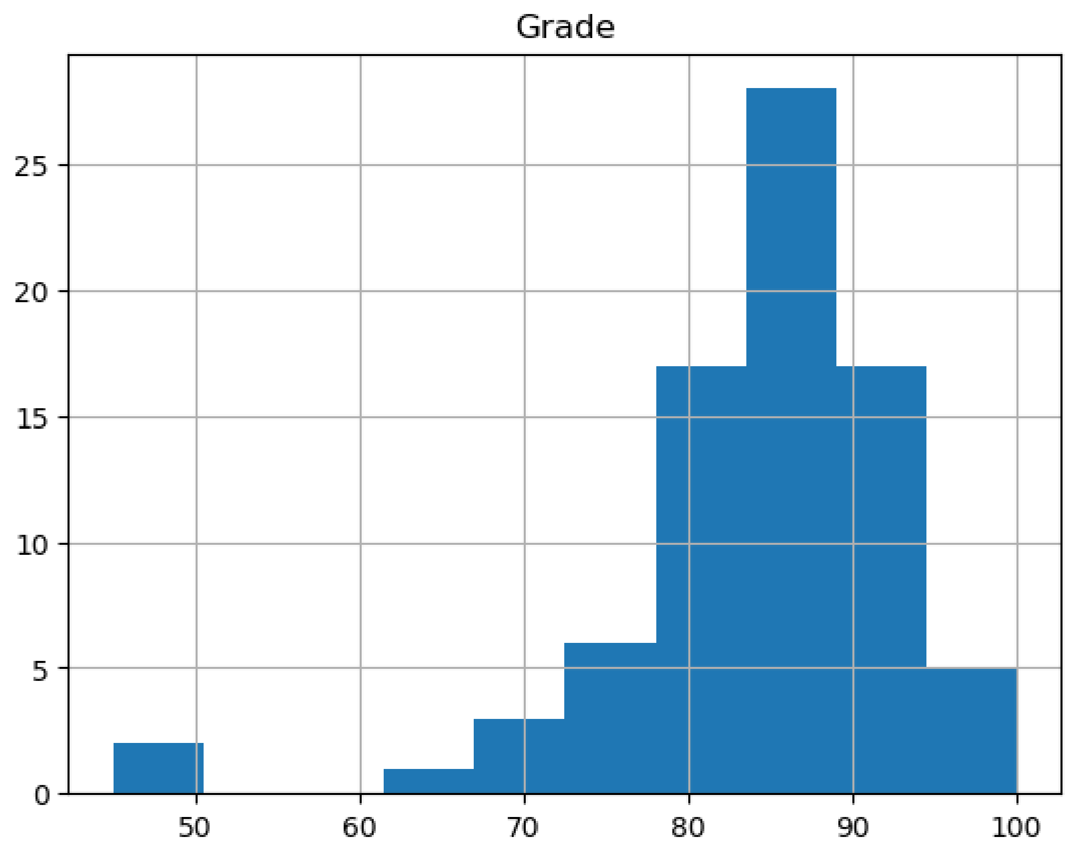
In [196]:
```
import seaborn as sns
```

```
In [197]: df.hist()
```

```
Out[197]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002B3CFD312C8
          >]],
                dtype=object)
```



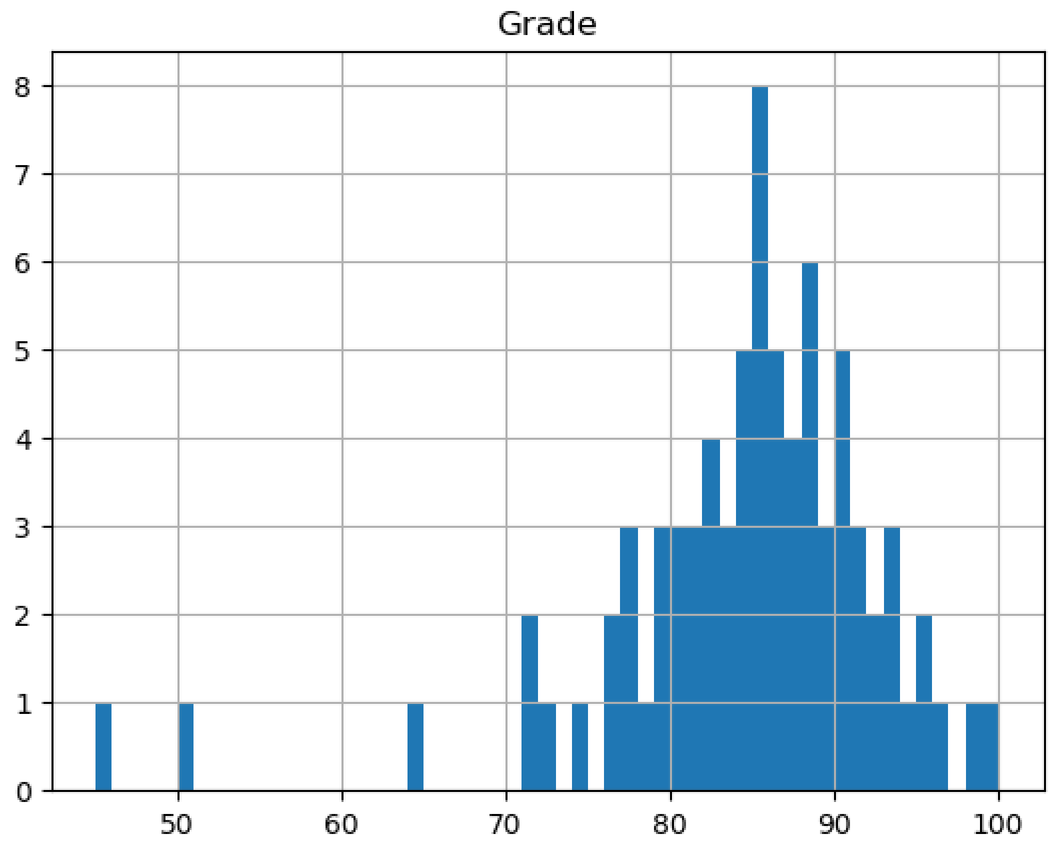
In [198]:
```
# remove the wordingd below the command type ;
df.hist();
```



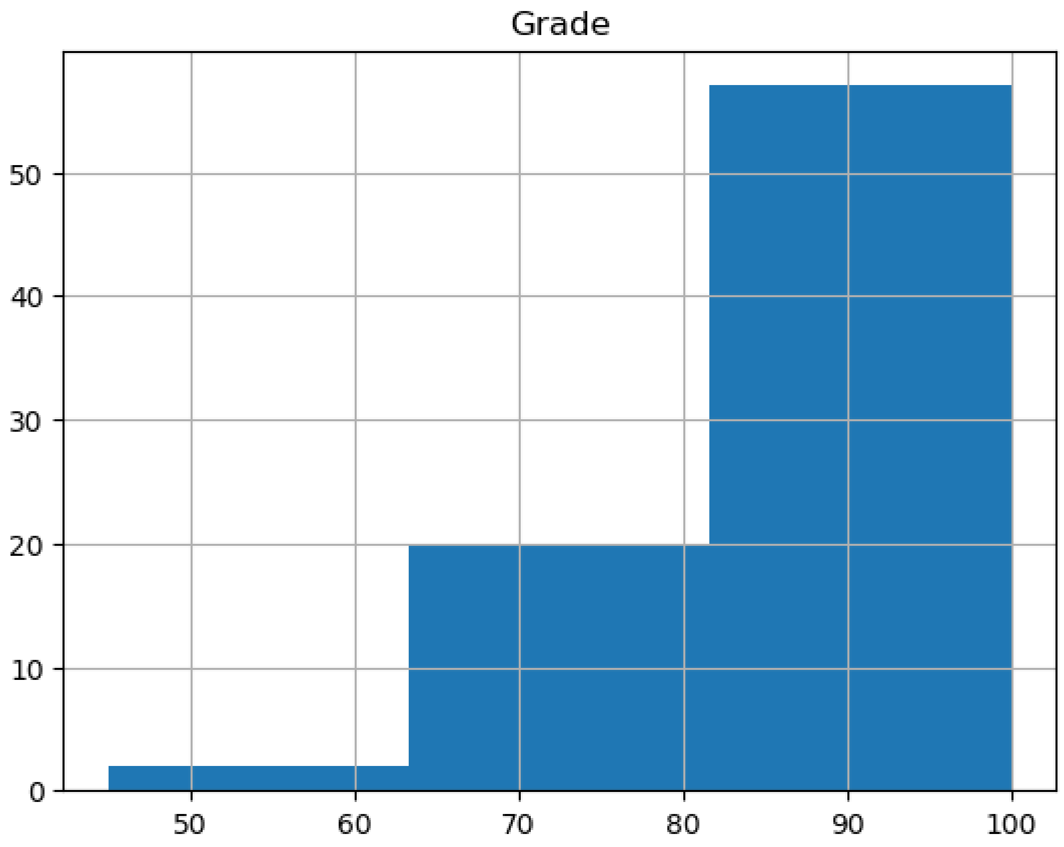
In [199]:
```
# to make this histogram more fine tune or less we use to select the bins
# i wanna see 1 bar for each grade
# to deside the bins  i'll take the differenece between  max range and the min
df.Grade.max() - df.Grade.min()
```

Out[199]:
```
55.0
```

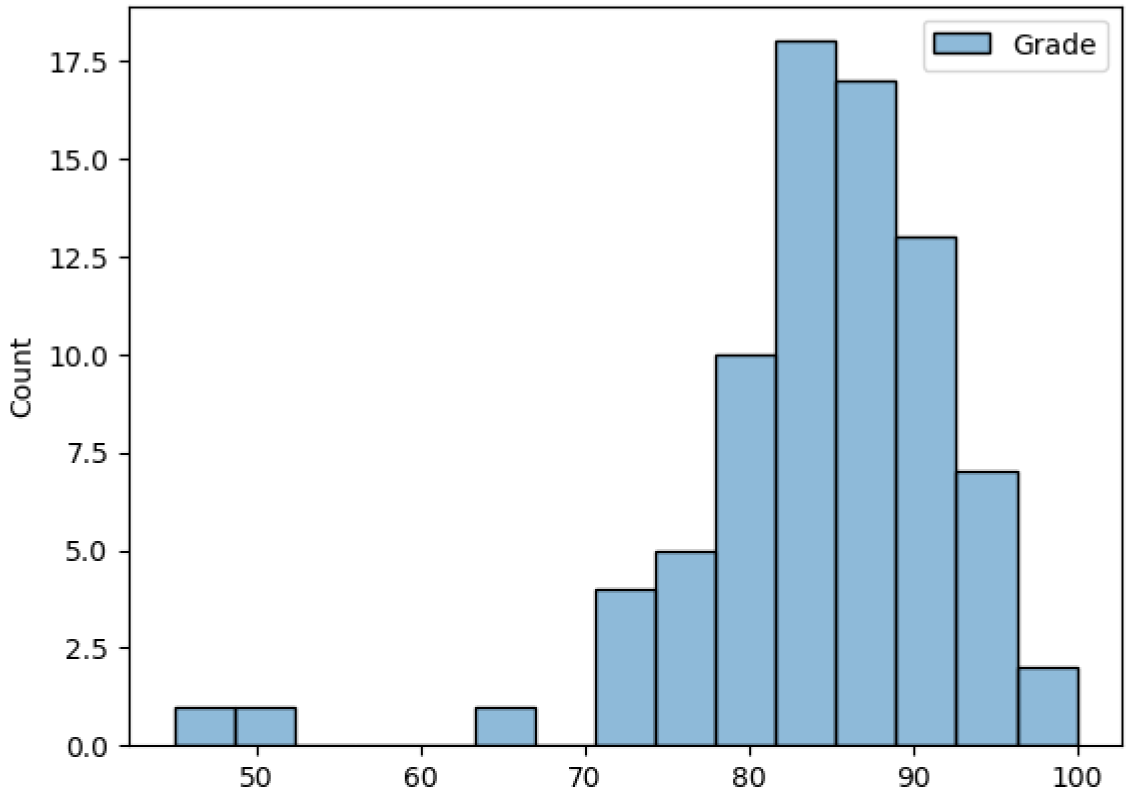
```
In [200]: df.hist(bins=55);
```



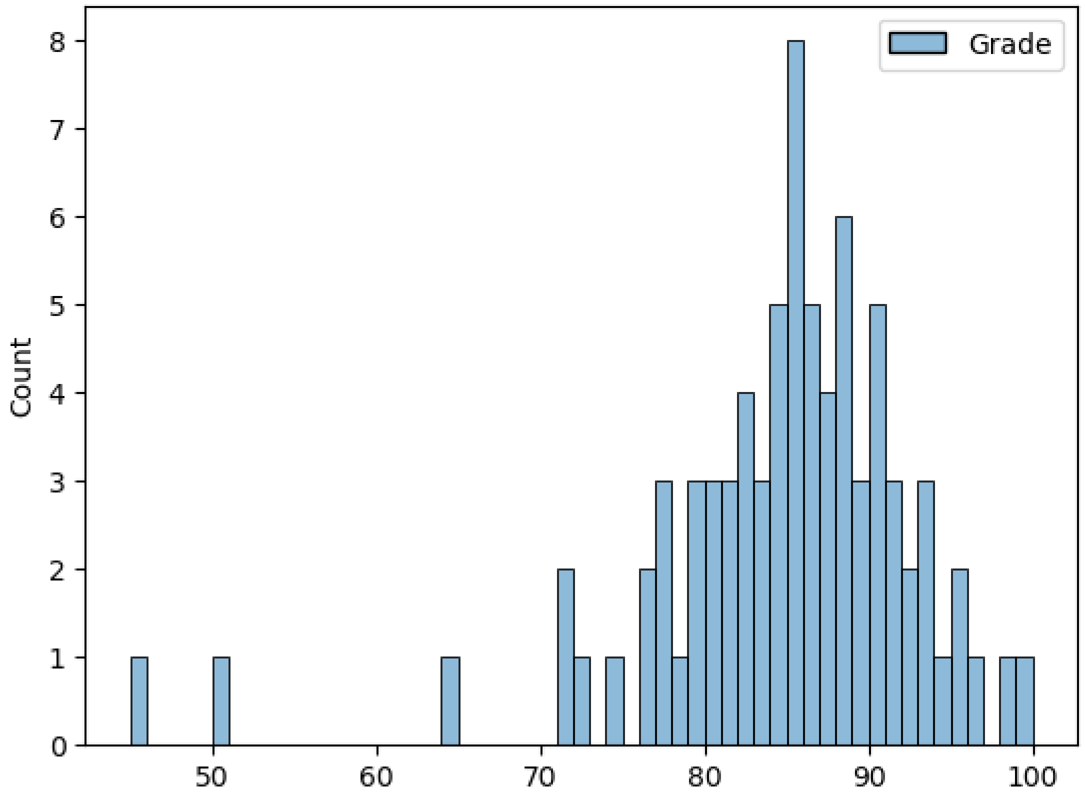
In [201]:
```
# to get the rough view of histogram
df.hist(bins=3);
```



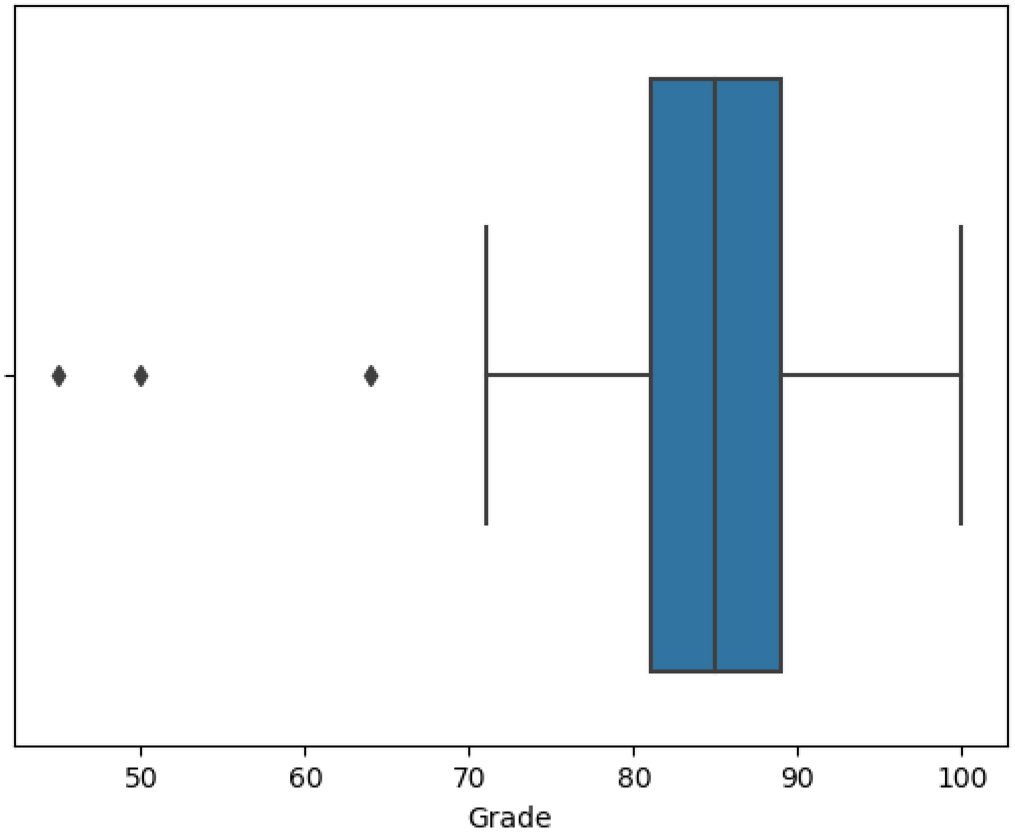
In [202]:
```
sns.histplot(df);
```



```
In [203]: sns.histplot(df, binwidth=1);
```



```
In [204]: sns.boxplot(x=df.Grade);
```



Above 3 dots are my outliers

```
In [205]: import numpy as np
```

```
In [206]: Q25, Q50, Q75 = np.percentile(df.Grade, (25, 50, 75))
```

```
In [207]: #To get the min and max lines
          iqr = Q75 - Q25
```

```
In [208]: min_grade = Q25 - 1.5*iqr
          max_grade = Q75 + 1.5*iqr
```

```
In [209]: min_grade, Q25, Q50, Q75, max_grade

Out[209]: (69.0, 81.0, 85.0, 89.0, 101.0)
```

```
In [210]: # To check the outliers of the dataset
          df[df.Grade < 69]
```

Out[210]:

	Student	Class	Year	Grade
36	John	Intro to Python	Freshman	50.0
53	Wyatt	Intro to SQL	Sophomore	45.0
66	Jaxon	Exploratory Data Analysis	Junior	64.0

```
In [211]: mean = np.mean(df.Grade)
```

```
In [212]: sd = np.std(df.Grade)
```

```
In [213]: mean, sd

Out[213]: (84.08987341772152, 8.723725033779411)
```

My grades are + or - 8 to 84 : I'll check the grades from 3 standard deviation away from the mean

```
In [214]: # Return me the grades when i'm going through the grades within the Grade colu
          mn if so treturn those if grades < mean -3 *sd
          # or if the grade is > mean + 3sd
          [grade for grade in df.Grade if (grade < mean - 3*sd) or (grade > mean + 3*s
          d)]

Out[214]: [50.0, 45.0]
```

In [215]:
```
# It gives me that outliers are 50 and 45 but if I change the outlier to 2std
[grade for grade in df.Grade if (grade < mean - 2*sd) or (grade > mean +2*sd)]
```
Out[215]:
```
[50.0, 45.0, 64.0]
```

In [216]:
```
# Sorting Data
df.Grade.sort_values()
```
Out[216]:
```
53     45.0
36     50.0
66     64.0
35     71.0
39     71.0
       ...
67     95.0
32     95.0
41     96.0
49     98.0
57    100.0
Name: Grade, Length: 79, dtype: float64
```

In [217]:
```
# Sorting data in Decending Order
df.Grade.sort_values(ascending=False)
```
Out[217]:
```
57    100.0
49     98.0
41     96.0
32     95.0
67     95.0
       ...
35     71.0
39     71.0
66     64.0
36     50.0
53     45.0
Name: Grade, Length: 79, dtype: float64
```

In [218]:
```
df.head()
```
Out[218]:

	Student	Class	Year	Grade
0	Emma	Freshman Seminar	Freshman	86.0
1	Olivia	Freshman Seminar	Freshman	86.0
2	Noah	Freshman Seminar	Freshman	86.0
3	Sophia	Freshman Seminar	Freshman	87.0
4	Liam	Freshman Seminar	Freshman	90.0

In [219]:
```
df.shape
```
Out[219]:
```
(79, 4)
```

We have 79 Rows and 4 Columns

In [220]:
```python
# We were to remove the grades < 60
df[df.Grade < 60]
```

Out[220]:

	Student	Class	Year	Grade
36	John	Intro to Python	Freshman	50.0
53	Wyatt	Intro to SQL	Sophomore	45.0

In [221]:
```python
# impiting Data
min_grade = df[df.Grade >=60]. Grade.min()
min_grade
```

Out[221]:
```
64.0
```

In [222]:
```python
df.Grade = np.where(df.Grade < 60, min_grade, df.Grade)
```

In [223]:
```python
df[df.Grade == 64]
```

Out[223]:

	Student	Class	Year	Grade
36	John	Intro to Python	Freshman	64.0
53	Wyatt	Intro to SQL	Sophomore	64.0
66	Jaxon	Exploratory Data Analysis	Junior	64.0

In [224]:
```python
# I'm changing Johns marks to 74
df.loc[36, 'Grade'] = 74
```

In [225]:
```python
df[df.Student == 'John']
```

Out[225]:

	Student	Class	Year	Grade
36	John	Intro to Python	Freshman	74.0

Missing Values

In [226]:
```python
df[df.isna(). any(axis=1)]
```

Out[226]:

Student	Class	Year	Grade

- I do not have any missing values

Inconsistant text anf Typos

In [227]:
```
df.Class.value_counts()
```

Out[227]:
```
Intro to Python              25
Exploratory Data Analysis    24
Intro to SQL                 18
Freshman Seminar             12
Name: Class, dtype: int64
```

- I do not have any Inconsistant text or Typos

Duplicate Data

In [228]:
```
df[df.duplicated()]
```

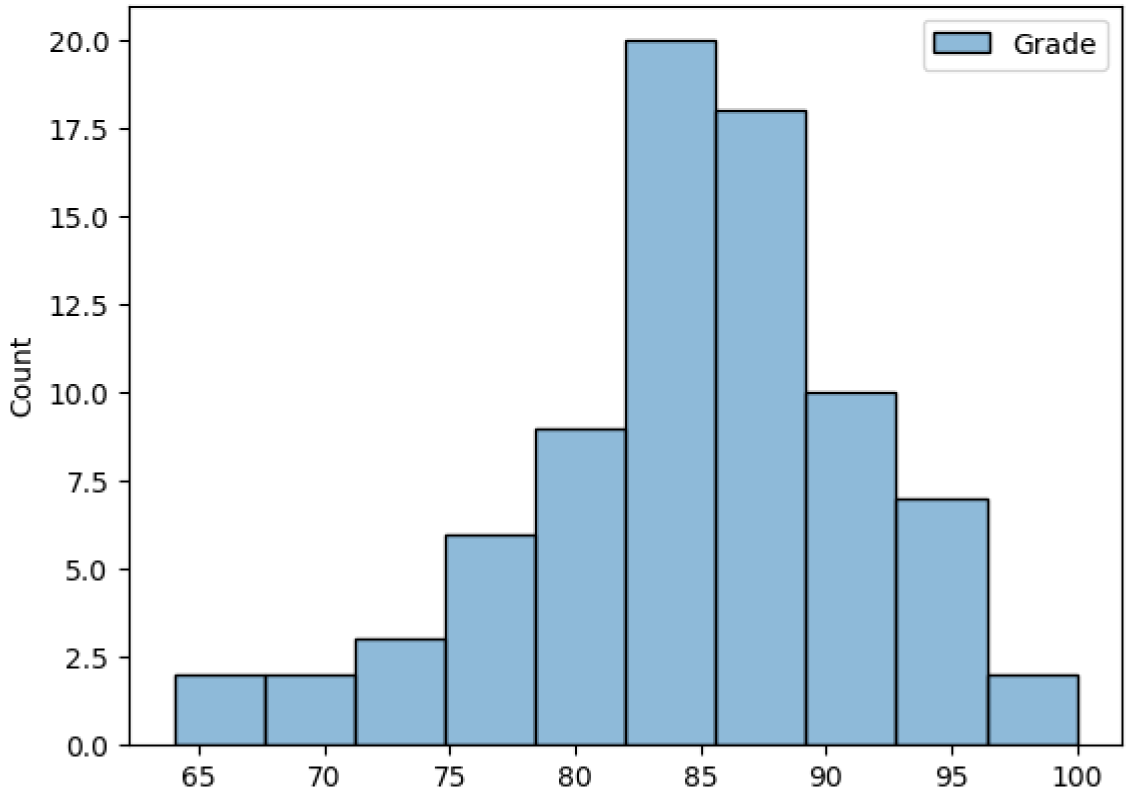
Out[228]:

Student	Class	Year	Grade

- I do nt have any Duplicates Values

Outliers

```
In [229]: sns.histplot(df);
```



- I do not have any outliers in the data

Creating New Columns

```
In [231]: import pandas as pd
```

```
In [232]: groceries = pd.read_excel('Groceries.xlsx')
In [233]: groceries.head()
```

Out[233]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Sh
0	P100010	Produce: Fruit	Apple	1.50	349	2023-06-12 15:35:00	202
1	P100011	Produce: Fruit	Banana	0.40	500	2023-06-12 18:30:00	202
2	P100012	Produce: Fruit	Grapes	4.00	200	2023-06-12 17:22:00	202
3	P100013	Produce: Fruit	Grapefruit	0.99	50	2023-06-12 16:29:00	202
4	P100014	Produce: Fruit	Organic Strawberries	3.99	148	2023-06-12 18:10:00	202

```
In [334]: groceries.isna().sum()
```

```
Out[334]: Product_ID                0
          Category                  0
          Item                      0
          Price_Dollars             0
          Inventory                 0
          Last_Updated              0
          Next_Scheduled_Shipment   0
          New Column                0
          Total Inventory           0
          Precent Inventory         0
          Low Inventory             0
          Last_Updated_Time         0
          Shipment_Date_DOW         0
          Next_Scheduled_Date       0
          New_Shipment_Date         0
          Product_ID_Num            0
          Sub_Category              0
          Organic                   0
          dtype: int64
```

```
In [336]: groceries.shape
```

```
Out[336]: (25, 18)
```

In [337]:
```
groceries.dtypes
```

Out[337]:
```
Product_ID                          object
Category                            object
Item                                object
Price_Dollars                      float64
Inventory                            int64
Last_Updated                datetime64[ns]
Next_Scheduled_Shipment     datetime64[ns]
New Column                         float64
Total Inventory                      int64
Precent Inventory                  float64
Low Inventory                       object
Last_Updated_Time                   object
Shipment_Date_DOW                   object
Next_Scheduled_Date         datetime64[ns]
New_Shipment_Date           datetime64[ns]
Product_ID_Num                       int32
Sub_Category                        object
Organic                               bool
dtype: object
```

In [234]:
```
round(groceries.Price_Dollars *1.05, 2)
```

Out[234]:
```
0      1.58
1      0.42
2      4.20
3      1.04
4      4.19
5      6.29
6      1.87
7      2.10
8      2.09
9      1.04
10     9.44
11    11.01
12     8.39
13     3.68
14     3.45
15     4.71
16     1.05
17     1.58
18     2.62
19     5.66
20     3.14
21     7.34
22     5.21
23    11.54
24     8.36
Name: Price_Dollars, dtype: float64
```

In [235]:
```
# I need to save it for a new column
groceries['New Column'] = round(groceries.Price_Dollars *1.05, 2)
```

In [238]:
```
groceries.head(2)
```
Out[238]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-

In [241]:
```
# To find out the percentage of Inventory
groceries['Total Inventory'] = groceries.Inventory.sum()
```
In [242]:
```
groceries.head(3)
```
Out[242]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-
2	P100012	Produce: Fruit	Grapes	4.0	200	2023-06-12 17:22:00	2023-06-

In [243]:
```
# To find out the percentage of Inventory
groceries['Precent Inventory'] = round(groceries['Inventory'] / groceries['Tot
al Inventory'] *100, 2)
```
In [244]:
```
groceries.head(3)
```
Out[244]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-
2	P100012	Produce: Fruit	Grapes	4.0	200	2023-06-12 17:22:00	2023-06-

In [245]:
```
#Let's again look at our inventory column and let's say if there's
#any item that has an inventory of under 50, then I want to
#flag that as low inventory.So what I can do is first importing numpy
#as NP and then I'm going to do NP dot where and in the cases
#where the inventory is less than 50, then set it equal to low inventory.
import numpy as np
groceries['Low Inventory'] = np.where(groceries.Inventory < 50, 'Low Inventor
y', '')
```

```
In [246]: groceries.head(20)
```

Out[246]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_S
0	P100010	Produce: Fruit	Apple	1.50	349	2023-06-12 15:35:00	20
1	P100011	Produce: Fruit	Banana	0.40	500	2023-06-12 18:30:00	20
2	P100012	Produce: Fruit	Grapes	4.00	200	2023-06-12 17:22:00	20
3	P100013	Produce: Fruit	Grapefruit	0.99	50	2023-06-12 16:29:00	20
4	P100014	Produce: Fruit	Organic Strawberries	3.99	148	2023-06-12 18:10:00	20
5	P100015	Produce: Fruit	Watermelon	5.99	99	2023-06-12 19:15:00	20
6	P100016	Produce: Vegetable	Cabbage	1.78	78	2023-06-12 19:25:00	20
7	P100017	Produce: Vegetable	Carrots	2.00	200	2023-06-12 18:05:00	20
8	P100018	Produce: Vegetable	Celery	1.99	50	2023-06-12 16:42:00	20
9	P100019	Produce: Vegetable	Cucumber	0.99	230	2023-06-12 17:47:00	20
10	P100020	Produce: Meat	Beef	8.99	145	2023-06-13 07:00:00	20
11	P100021	Produce: Meat	Chicken (Organic)	10.49	284	2023-06-13 07:20:00	20
12	P100022	Produce: Meat	Turkey	7.99	188	2023-06-13 07:32:00	20
13	P100023	Produce: Dairy	Butter	3.50	400	2023-06-13 08:35:00	20
14	P100024	Produce: Dairy	Eggs	3.29	234	2023-06-13 08:54:00	20
15	P100025	Produce: Dairy	Milk (Soy)	4.49	32	2023-06-13 08:37:00	20
16	P100026	Produce: Dairy	Yogurt	1.00	432	2023-06-13 08:41:00	20
17	P100027	Pantry: Snacks	Apple Sauce - organic	1.50	27	2023-06-10 12:02:00	20
18	P100028	Pantry: Snacks	Chips	2.50	365	2023-06-10 12:12:00	20
19	P100029	Pantry: Snacks	Cookies (Oatmeal)	5.39	340	2023-06-10 12:24:00	20

- Extracting Dates.

In [248]:
```
groceries['Last_Updated_Time'] = groceries.Last_Updated.dt.time
```

In [249]:
```
groceries.head(3)
```

Out[249]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-
2	P100012	Produce: Fruit	Grapes	4.0	200	2023-06-12 17:22:00	2023-06-

In [251]:
```
# Extracting day of the week from the schedule shipment feild
groceries['Shipment_Date_DOW'] = groceries.Next_Scheduled_Shipment.dt.dayofwee
k
```

In [254]:
```
groceries.head(3)
```

Out[254]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-
2	P100012	Produce: Fruit	Grapes	4.0	200	2023-06-12 17:22:00	2023-06-

In [255]:
```
# to get the day of the week, I use mapping
DOW_mappings = {0 : 'Monday',
                1 : 'Tuesday',
                2 : 'Wednesday',
                3 : 'Thursday',
                4 : 'Friday',
                5 : 'Saturday',
                6 : 'Sunday'}
```

```
In [259]: groceries['Shipment_Date_DOW'].map(DOW_mappings)
```

```
Out[259]: 0       Thursday
          1       Thursday
          2       Thursday
          3       Thursday
          4       Thursday
          5       Thursday
          6       Thursday
          7       Thursday
          8       Thursday
          9       Thursday
          10      Saturday
          11      Saturday
          12      Saturday
          13      Saturday
          14      Saturday
          15      Saturday
          16      Saturday
          17      Saturday
          18      Saturday
          19      Saturday
          20      Saturday
          21     Wednesday
          22     Wednesday
          23     Wednesday
          24     Wednesday
          Name: Shipment_Date_DOW, dtype: object
```

```
In [262]: groceries['Shipment_Date_DOW'] = groceries['Shipment_Date_DOW'].map(DOW_mappin
          gs)
```

```
In [263]: groceries.head(2)
```

Out[263]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-

In [266]:
```
# I want to add 1 day to all those next scheduled shipment
#goceries.Next_Sceduled_Shipment +1 # I can not add 1 straight away becase its
# data type is datetime
groceries.Next_Scheduled_Shipment + pd.to_timedelta(1,'D')
```

Out[266]:
```
0     2023-06-16
1     2023-06-16
2     2023-06-16
3     2023-06-16
4     2023-06-16
5     2023-06-16
6     2023-06-16
7     2023-06-16
8     2023-06-16
9     2023-06-16
10    2023-06-18
11    2023-06-18
12    2023-06-18
13    2023-06-18
14    2023-06-18
15    2023-06-18
16    2023-06-18
17    2023-06-25
18    2023-06-25
19    2023-06-25
20    2023-06-25
21    2023-06-29
22    2023-06-29
23    2023-06-29
24    2023-06-29
Name: Next_Scheduled_Shipment, dtype: datetime64[ns]
```

In [268]:
```
groceries['Next_Scheduled_Date'] = groceries.Next_Scheduled_Shipment + pd.to_t
imedelta(1, 'D')
```

In [269]:
```
groceries.head(3)
```

Out[269]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-
2	P100012	Produce: Fruit	Grapes	4.0	200	2023-06-12 17:22:00	2023-06-

In [273]:
```
# Lets say i only wanted to add a additional date only for the fruit items
groceries['New_Shipment_Date'] = np.where(groceries.Category == 'Produce: Frui
t',
                                          groceries.Next_Scheduled_Shipment + p
d.to_timedelta(1, 'D'),
                                          groceries.Next_Scheduled_Date)
```

```
In [274]: groceries.head(25)
```

Out[274]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_S
0	P100010	Produce: Fruit	Apple	1.50	349	2023-06-12 15:35:00	20
1	P100011	Produce: Fruit	Banana	0.40	500	2023-06-12 18:30:00	20
2	P100012	Produce: Fruit	Grapes	4.00	200	2023-06-12 17:22:00	20
3	P100013	Produce: Fruit	Grapefruit	0.99	50	2023-06-12 16:29:00	20
4	P100014	Produce: Fruit	Organic Strawberries	3.99	148	2023-06-12 18:10:00	20
5	P100015	Produce: Fruit	Watermelon	5.99	99	2023-06-12 19:15:00	20
6	P100016	Produce: Vegetable	Cabbage	1.78	78	2023-06-12 19:25:00	20
7	P100017	Produce: Vegetable	Carrots	2.00	200	2023-06-12 18:05:00	20
8	P100018	Produce: Vegetable	Celery	1.99	50	2023-06-12 16:42:00	20
9	P100019	Produce: Vegetable	Cucumber	0.99	230	2023-06-12 17:47:00	20
10	P100020	Produce: Meat	Beef	8.99	145	2023-06-13 07:00:00	20
11	P100021	Produce: Meat	Chicken (Organic)	10.49	284	2023-06-13 07:20:00	20
12	P100022	Produce: Meat	Turkey	7.99	188	2023-06-13 07:32:00	20
13	P100023	Produce: Dairy	Butter	3.50	400	2023-06-13 08:35:00	20
14	P100024	Produce: Dairy	Eggs	3.29	234	2023-06-13 08:54:00	20
15	P100025	Produce: Dairy	Milk (Soy)	4.49	32	2023-06-13 08:37:00	20
16	P100026	Produce: Dairy	Yogurt	1.00	432	2023-06-13 08:41:00	20
17	P100027	Pantry: Snacks	Apple Sauce - organic	1.50	27	2023-06-10 12:02:00	20
18	P100028	Pantry: Snacks	Chips	2.50	365	2023-06-10 12:12:00	20
19	P100029	Pantry: Snacks	Cookies (Oatmeal)	5.39	340	2023-06-10 12:24:00	20
20	P100030	Pantry: Snacks	Raisins	2.99	5	2023-06-10 12:38:00	20
21	P100031	Frozen: Frozen Snacks	Chicken Nuggets	6.99	85	2023-05-28 22:02:00	20

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_S
22	P100032	Frozen: Frozen Snacks	Spinach Dip	4.96	76	2023-05-28 22:05:00	20
23	P100033	Frozen: Frozen Fruit	Frozen Blueberries	10.99	162	2023-05-28 22:14:00	20
24	P100034	Frozen: Frozen Fruit	Frozen Pineapple	7.96	178	2023-05-28 22:11:00	20

Using Text Data to Create New Columns

In [276]:
```
groceries.dtypes
```

Out[276]:
```
Product_ID                        object
Category                          object
Item                              object
Price_Dollars                    float64
Inventory                          int64
Last_Updated              datetime64[ns]
Next_Scheduled_Shipment   datetime64[ns]
New Column                       float64
Total Inventory                    int64
Precent Inventory                float64
Low Inventory                     object
Last_Updated_Time                 object
Shipment_Date_DOW                 object
Next_Scheduled_Date       datetime64[ns]
New_Shipment_Date         datetime64[ns]
dtype: object
```

In [277]:
```
groceries.head(3)
```

Out[277]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-
2	P100012	Produce: Fruit	Grapes	4.0	200	2023-06-12 17:22:00	2023-06-

- Removing Charactors.

In [286]:
```
# inorder to turn product ID into numeric column we remove the letter
# infront of the ID
groceries['Product_ID_Num'] = groceries.Product_ID.str[1:]
```

In [287]:
```
groceries.dtypes
```

Out[287]:
```
Product_ID                          object
Category                            object
Item                                object
Price_Dollars                      float64
Inventory                            int64
Last_Updated                datetime64[ns]
Next_Scheduled_Shipment     datetime64[ns]
New Column                         float64
Total Inventory                      int64
Precent Inventory                  float64
Low Inventory                       object
Last_Updated_Time                   object
Shipment_Date_DOW                   object
Next_Scheduled_Date         datetime64[ns]
New_Shipment_Date           datetime64[ns]
Product_ID_Num                      object
dtype: object
```

In [285]:
```
groceries.head(2)
```

Out[285]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce: Fruit	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce: Fruit	Banana	0.4	500	2023-06-12 18:30:00	2023-06-

In [299]:
```
# To change the data type
groceries['Product_ID_Num']. astype('int')
```

Out[299]:
```
0     100010
1     100011
2     100012
3     100013
4     100014
5     100015
6     100016
7     100017
8     100018
9     100019
10    100020
11    100021
12    100022
13    100023
14    100024
15    100025
16    100026
17    100027
18    100028
19    100029
20    100030
21    100031
22    100032
23    100033
24    100034
Name: Product_ID_Num, dtype: int32
```

In [300]:
```
# To save the changes
groceries['Product_ID_Num'] = groceries['Product_ID_Num']. astype('int')
```

In [301]:
```
groceries.dtypes
```

Out[301]:
```
Product_ID                          object
Category                            object
Item                                object
Price_Dollars                      float64
Inventory                            int64
Last_Updated                datetime64[ns]
Next_Scheduled_Shipment     datetime64[ns]
New Column                         float64
Total Inventory                      int64
Precent Inventory                  float64
Low Inventory                       object
Last_Updated_Time                   object
Shipment_Date_DOW                   object
Next_Scheduled_Date         datetime64[ns]
New_Shipment_Date           datetime64[ns]
Product_ID_Num                       int32
dtype: object
```

- Split into columns

```
In [303]: # going to split the Category column into 2 columns
          groceries.Category.value_counts()
```

```
Out[303]: Produce: Fruit           6
          Produce: Vegetable       4
          Produce: Dairy           4
          Pantry: Snacks           4
          Produce: Meat            3
          Frozen: Frozen Snacks    2
          Frozen: Frozen Fruit     2
          Name: Category, dtype: int64
```

```
In [304]: # I need product in one column and items are in anotheter column
          groceries.Category.str.split(':').to_list()
```

```
Out[304]: [['Produce', ' Fruit'],
           ['Produce', ' Fruit'],
           ['Produce', ' Fruit'],
           ['Produce', ' Fruit'],
           ['Produce', ' Fruit'],
           ['Produce', ' Fruit'],
           ['Produce', ' Vegetable'],
           ['Produce', ' Vegetable'],
           ['Produce', ' Vegetable'],
           ['Produce', ' Vegetable'],
           ['Produce', ' Meat'],
           ['Produce', ' Meat'],
           ['Produce', ' Meat'],
           ['Produce', ' Dairy'],
           ['Produce', ' Dairy'],
           ['Produce', ' Dairy'],
           ['Produce', ' Dairy'],
           ['Pantry', ' Snacks'],
           ['Pantry', ' Snacks'],
           ['Pantry', ' Snacks'],
           ['Pantry', ' Snacks'],
           ['Frozen', ' Frozen Snacks'],
           ['Frozen', ' Frozen Snacks'],
           ['Frozen', ' Frozen Fruit'],
           ['Frozen', ' Frozen Fruit']]
```

In [306]:
```
pd.DataFrame(groceries.Category.str.split(':').to_list())
```

Out[306]:

	0	1
0	Produce	Fruit
1	Produce	Fruit
2	Produce	Fruit
3	Produce	Fruit
4	Produce	Fruit
5	Produce	Fruit
6	Produce	Vegetable
7	Produce	Vegetable
8	Produce	Vegetable
9	Produce	Vegetable
10	Produce	Meat
11	Produce	Meat
12	Produce	Meat
13	Produce	Dairy
14	Produce	Dairy
15	Produce	Dairy
16	Produce	Dairy
17	Pantry	Snacks
18	Pantry	Snacks
19	Pantry	Snacks
20	Pantry	Snacks
21	Frozen	Frozen Snacks
22	Frozen	Frozen Snacks
23	Frozen	Frozen Fruit
24	Frozen	Frozen Fruit

In [307]:
```
# To name the above feilds
groceries[['Category', 'Sub_Category']] = pd.DataFrame(groceries.Category.str.
split(':').to_list())
```

In [323]:
```
groceries.head(3)
```
Out[323]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_Shipme
0	P100010	Produce	Apple	1.5	349	2023-06-12 15:35:00	2023-06-
1	P100011	Produce	Banana	0.4	500	2023-06-12 18:30:00	2023-06-
2	P100012	Produce	Grapes	4.0	200	2023-06-12 17:22:00	2023-06-

In [310]:
```
# To check the items contain organic product
groceries.Item.str.contains('Organic')
```
Out[310]:
```
0     False
1     False
2     False
3     False
4      True
5     False
6     False
7     False
8     False
9     False
10    False
11     True
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
Name: Item, dtype: bool
```
In [311]:
```
# to save it into a new colum
groceries['Organic'] = groceries.Item.str.contains('Organic')
```

```
In [326]: groceries
```

Out[326]:

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_S
0	P100010	Produce	Apple	1.50	349	2023-06-12 15:35:00	20
1	P100011	Produce	Banana	0.40	500	2023-06-12 18:30:00	20
2	P100012	Produce	Grapes	4.00	200	2023-06-12 17:22:00	20
3	P100013	Produce	Grapefruit	0.99	50	2023-06-12 16:29:00	20
4	P100014	Produce	Organic Strawberries	3.99	148	2023-06-12 18:10:00	20
5	P100015	Produce	Watermelon	5.99	99	2023-06-12 19:15:00	20
6	P100016	Produce	Cabbage	1.78	78	2023-06-12 19:25:00	20
7	P100017	Produce	Carrots	2.00	200	2023-06-12 18:05:00	20
8	P100018	Produce	Celery	1.99	50	2023-06-12 16:42:00	20
9	P100019	Produce	Cucumber	0.99	230	2023-06-12 17:47:00	20
10	P100020	Produce	Beef	8.99	145	2023-06-13 07:00:00	20
11	P100021	Produce	Chicken (Organic)	10.49	284	2023-06-13 07:20:00	20
12	P100022	Produce	Turkey	7.99	188	2023-06-13 07:32:00	20
13	P100023	Produce	Butter	3.50	400	2023-06-13 08:35:00	20
14	P100024	Produce	Eggs	3.29	234	2023-06-13 08:54:00	20
15	P100025	Produce	Milk (Soy)	4.49	32	2023-06-13 08:37:00	20
16	P100026	Produce	Yogurt	1.00	432	2023-06-13 08:41:00	20
17	P100027	Pantry	Apple Sauce - organic	1.50	27	2023-06-10 12:02:00	20
18	P100028	Pantry	Chips	2.50	365	2023-06-10 12:12:00	20
19	P100029	Pantry	Cookies (Oatmeal)	5.39	340	2023-06-10 12:24:00	20
20	P100030	Pantry	Raisins	2.99	5	2023-06-10 12:38:00	20
21	P100031	Frozen	Chicken Nuggets	6.99	85	2023-05-28 22:02:00	20

	Product_ID	Category	Item	Price_Dollars	Inventory	Last_Updated	Next_Scheduled_S
22	P100032	Frozen	Spinach Dip	4.96	76	2023-05-28 22:05:00	20:
23	P100033	Frozen	Frozen Blueberries	10.99	162	2023-05-28 22:14:00	20:
24	P100034	Frozen	Frozen Pineapple	7.96	178	2023-05-28 22:11:00	20:

```
In [329]: # To reorganize the column
          groceries[['Product_ID', 'Product_ID_Num', 'Category', 'Sub_Category',
                    'Item', 'Organic', 'Price_Dollars', 'Inventory', 'Precent Inventor
          y',
                    'Low Inventory', 'Last_Updated', 'Last_Updated_Time',
                    'New_Shipment_Date', 'Shipment_Date_DOW']]
```

Out[329]:

	Product_ID	Product_ID_Num	Category	Sub_Category	Item	Organic	Price_Dollars	I
0	P100010	100010	Produce	Fruit	Apple	False	1.50	
1	P100011	100011	Produce	Fruit	Banana	False	0.40	
2	P100012	100012	Produce	Fruit	Grapes	False	4.00	
3	P100013	100013	Produce	Fruit	Grapefruit	False	0.99	
4	P100014	100014	Produce	Fruit	Organic Strawberries	True	3.99	
5	P100015	100015	Produce	Fruit	Watermelon	False	5.99	
6	P100016	100016	Produce	Vegetable	Cabbage	False	1.78	
7	P100017	100017	Produce	Vegetable	Carrots	False	2.00	
8	P100018	100018	Produce	Vegetable	Celery	False	1.99	
9	P100019	100019	Produce	Vegetable	Cucumber	False	0.99	
10	P100020	100020	Produce	Meat	Beef	False	8.99	
11	P100021	100021	Produce	Meat	Chicken (Organic)	True	10.49	
12	P100022	100022	Produce	Meat	Turkey	False	7.99	
13	P100023	100023	Produce	Dairy	Butter	False	3.50	
14	P100024	100024	Produce	Dairy	Eggs	False	3.29	
15	P100025	100025	Produce	Dairy	Milk (Soy)	False	4.49	
16	P100026	100026	Produce	Dairy	Yogurt	False	1.00	
17	P100027	100027	Pantry	Snacks	Apple Sauce - organic	False	1.50	
18	P100028	100028	Pantry	Snacks	Chips	False	2.50	
19	P100029	100029	Pantry	Snacks	Cookies (Oatmeal)	False	5.39	
20	P100030	100030	Pantry	Snacks	Raisins	False	2.99	
21	P100031	100031	Frozen	Frozen Snacks	Chicken Nuggets	False	6.99	

	Product_ID	Product_ID_Num	Category	Sub_Category	Item	Organic	Price_Dollars	I
22	P100032	100032	Frozen	Frozen Snacks	Spinach Dip	False	4.96	
23	P100033	100033	Frozen	Frozen Fruit	Frozen Blueberries	False	10.99	
24	P100034	100034	Frozen	Frozen Fruit	Frozen Pineapple	False	7.96	

In [330]:
```
# To save the data
groceries_with_new_column = groceries[['Product_ID', 'Product_ID_Num', 'Catego
ry', 'Sub_Category',
          'Item', 'Organic', 'Price_Dollars', 'Inventory', 'Precent Inventor
y',
          'Low Inventory', 'Last_Updated', 'Last_Updated_Time',
          'New_Shipment_Date', 'Shipment_Date_DOW']]
```

In [331]:
```
groceries_with_new_column
```

Out[331]:

	Product_ID	Product_ID_Num	Category	Sub_Category	Item	Organic	Price_Dollars	I
0	P100010	100010	Produce	Fruit	Apple	False	1.50	
1	P100011	100011	Produce	Fruit	Banana	False	0.40	
2	P100012	100012	Produce	Fruit	Grapes	False	4.00	
3	P100013	100013	Produce	Fruit	Grapefruit	False	0.99	
4	P100014	100014	Produce	Fruit	Organic Strawberries	True	3.99	
5	P100015	100015	Produce	Fruit	Watermelon	False	5.99	
6	P100016	100016	Produce	Vegetable	Cabbage	False	1.78	
7	P100017	100017	Produce	Vegetable	Carrots	False	2.00	
8	P100018	100018	Produce	Vegetable	Celery	False	1.99	
9	P100019	100019	Produce	Vegetable	Cucumber	False	0.99	
10	P100020	100020	Produce	Meat	Beef	False	8.99	
11	P100021	100021	Produce	Meat	Chicken (Organic)	True	10.49	
12	P100022	100022	Produce	Meat	Turkey	False	7.99	
13	P100023	100023	Produce	Dairy	Butter	False	3.50	
14	P100024	100024	Produce	Dairy	Eggs	False	3.29	
15	P100025	100025	Produce	Dairy	Milk (Soy)	False	4.49	
16	P100026	100026	Produce	Dairy	Yogurt	False	1.00	
17	P100027	100027	Pantry	Snacks	Apple Sauce - organic	False	1.50	
18	P100028	100028	Pantry	Snacks	Chips	False	2.50	
19	P100029	100029	Pantry	Snacks	Cookies (Oatmeal)	False	5.39	
20	P100030	100030	Pantry	Snacks	Raisins	False	2.99	
21	P100031	100031	Frozen	Frozen Snacks	Chicken Nuggets	False	6.99	

	Product_ID	Product_ID_Num	Category	Sub_Category	Item	Organic	Price_Dollars	I
22	P100032	100032	Frozen	Frozen Snacks	Spinach Dip	False	4.96	
23	P100033	100033	Frozen	Frozen Fruit	Frozen Blueberries	False	10.99	
24	P100034	100034	Frozen	Frozen Fruit	Frozen Pineapple	False	7.96	

```
In [332]: groceries_with_new_column.shape
Out[332]: (25, 14)
```

```
In [325]: groceries.shape
Out[325]: (25, 18)
```

- Creating a new notebook with a new column.

```
In [318]: # To open up with a new notebook to work with Exploratory data analysis
          groceries_with_new_column.to_pickle('groceries_with_new_column.pkl')
          # the note bookcan be found in the same working folder and when you wanted to
          use it open up a new python workbook then use the
          #following code
          #import pandas as pd
          #pd.read_pickle('groceries_with_new_columns.pkl')
```

```
In [333]: # If I need the clean dats to export into csv
          groceries_with_new_column.to_csv('groceries_with_new_column.csv')
```