2023-04-05-file-1

April 5, 2023

[4]:
```
data = [50,50,47,97,49,3,53,42,26,74,82,62,37,15,70,27,36,35,48,52,63,64]
print(data)
```

```
[50, 50, 47, 97, 49, 3, 53, 42, 26, 74, 82, 62, 37, 15, 70, 27, 36, 35, 48, 52,
63, 64]
```

[5]:
```
import numpy as np

grades = np.array(data)
print(grades)
```

```
[50 50 47 97 49  3 53 42 26 74 82 62 37 15 70 27 36 35 48 52 63 64]
```

[6]:
```
print (type(data),'x 2:', data * 2)
print('---')
print (type(grades),'x 2:', grades * 2)
```

```
<class 'list'> x 2: [50, 50, 47, 97, 49, 3, 53, 42, 26, 74, 82, 62, 37, 15, 70,
27, 36, 35, 48, 52, 63, 64, 50, 50, 47, 97, 49, 3, 53, 42, 26, 74, 82, 62, 37,
15, 70, 27, 36, 35, 48, 52, 63, 64]
---
<class 'numpy.ndarray'> x 2: [100 100  94 194  98   6 106  84  52 148 164 124
74  30 140  54  72  70
  96 104 126 128]
```

[7]:
```
grades.shape
```

[7]:
```
(22,)
```

[8]:
```
grades[0]
```

[8]:
```
50
```

[47]:
```
# Define an array of study hours
study_hours = [10.0,11.5,9.0,16.0,9.25,1.0,11.5,9.0,8.5,14.5,15.5,
               13.75,9.0,8.0,15.5,8.0,9.0,6.0,10.0,12.0,12.5,12.0,]

# Create a 2D array (an array of arrays)
```

```
student_data = np.array([study_hours, grades])

# display the array
student_data
```

[47]:
```
array([[10.  , 11.5 ,  9.  , 16.  ,  9.25,  1.  , 11.5 ,  9.  ,  8.5 ,
        14.5 , 15.5 , 13.75,  9.  ,  8.  , 15.5 ,  8.  ,  9.  ,  6.  ,
        10.  , 12.  , 12.5 , 12.  ],
       [50.  , 50.  , 47.  , 97.  , 49.  ,  3.  , 53.  , 42.  , 26.  ,
        74.  , 82.  , 62.  , 37.  , 15.  , 70.  , 27.  , 36.  , 35.  ,
        48.  , 52.  , 63.  , 64.  ]])
```

[10]:
```
student_data.shape
```

[10]:
```
(2, 22)
```

[11]:
```
# Get the mean value of each sub-array
avg_study = student_data[0].mean()
avg_grade = student_data[1].mean()

print('Average study hours: {:.2f}\nAverage grade: {:.2f}'.format(avg_study,␣
 ↪avg_grade))
```

```
Average study hours: 10.52
Average grade: 49.18
```

[12]:
```
import pandas as pd

df_students = pd.DataFrame({'Name': ['Damian', 'Krzyś', 'Sławek', 'Roksana',␣
 ↪'Natan', 'Wiktoria', 'Franek', 'Julia',
                                     'Basia', 'Janek', 'Francesca', 'Rafał',␣
 ↪'Nina', 'Kacper', 'Joanna',
                                    ␣
 ↪'Cezary','Helena','Iza','Anna','Staszek','Daniel','Kacper'],
                              'StudyHours':student_data[0],
                              'Grade':student_data[1]})

df_students
```

[12]:

	Name	StudyHours	Grade
0	Damian	10.00	50.0
1	Krzyś	11.50	50.0
2	Sławek	9.00	47.0
3	Roksana	16.00	97.0
4	Natan	9.25	49.0
5	Wiktoria	1.00	3.0
6	Franek	11.50	53.0

```
7       Julia        9.00   42.0
8       Basia        8.50   26.0
9       Janek       14.50   74.0
10  Francesca       15.50   82.0
11      Rafał       13.75   62.0
12       Nina        9.00   37.0
13     Kacper        8.00   15.0
14     Joanna       15.50   70.0
15     Cezary        8.00   27.0
16     Helena        9.00   36.0
17        Iza        6.00   35.0
18       Anna       10.00   48.0
19    Staszek       12.00   52.0
20     Daniel       12.50   63.0
21     Kacper       12.00   64.0
```

[13]:
```
df_students.loc[5]
```

[13]:
```
Name          Wiktoria
StudyHours         1.0
Grade              3.0
Name: 5, dtype: object
```

[14]:
```
df_students.loc[0:5]
```

[14]:
```
        Name  StudyHours  Grade
0     Damian       10.00   50.0
1      Krzyś       11.50   50.0
2     Sławek        9.00   47.0
3    Roksana       16.00   97.0
4      Natan        9.25   49.0
5  Wiktoria        1.00    3.0
```

[15]:
```
df_students.iloc[0:5]
```

[15]:
```
       Name  StudyHours  Grade
0    Damian       10.00   50.0
1     Krzyś       11.50   50.0
2    Sławek        9.00   47.0
3   Roksana       16.00   97.0
4     Natan        9.25   49.0
```

[16]:
```
df_students.iloc[0,[1,2]]
```

[16]:
```
StudyHours    10.0
Grade         50.0
Name: 0, dtype: object
```

[17]:
```
df_students.loc[0,'Grade']
```

[17]:
```
50.0
```

[ ]:
```

```

[18]:
```
df_students.iloc[0,[1]]
```

[18]:
```
StudyHours    10.0
Name: 0, dtype: object
```

[19]:
```
df_students.loc[df_students['Name']=='Kacper']
```

[19]:

	Name	StudyHours	Grade
13	Kacper	8.0	15.0
21	Kacper	12.0	64.0

[20]:
```
df_students[df_students['Name']=='Kacper']
```

[20]:

	Name	StudyHours	Grade
13	Kacper	8.0	15.0
21	Kacper	12.0	64.0

[21]:
```
df_students.query('Name=="Kacper"')
```

[21]:

	Name	StudyHours	Grade
13	Kacper	8.0	15.0
21	Kacper	12.0	64.0

[22]:
```
df_students.isnull()
```

[22]:

	Name	StudyHours	Grade
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
5	False	False	False
6	False	False	False
7	False	False	False
8	False	False	False
9	False	False	False
10	False	False	False
11	False	False	False
12	False	False	False
13	False	False	False
14	False	False	False

```
15  False       False  False
16  False       False  False
17  False       False  False
18  False       False  False
19  False       False  False
20  False       False  False
21  False       False  False
```

[26]:
```
df_students.isnull().sum()
```

[26]:
```
Name          0
StudyHours    0
Grade         0
dtype: int64
```

[27]:
```
df_students.sum()
```

[27]:
```
Name          DanJoannPedroRosieEthanVickyFredericJimmieRhon…
StudyHours                                              231.5
Grade                                                  1082.0
dtype: object
```

[23]:
```
df_students[df_students.isnull().any(axis=1)]
```

[23]:
```
Empty DataFrame
Columns: [Name, StudyHours, Grade]
Index: []
```

[24]:
```
df_students.StudyHours = df_students.StudyHours.fillna(df_students.StudyHours.
↪mean())
df_students
```

[24]:

	Name	StudyHours	Grade
0	Damian	10.00	50.0
1	Krzyś	11.50	50.0
2	Sławek	9.00	47.0
3	Roksana	16.00	97.0
4	Natan	9.25	49.0
5	Wiktoria	1.00	3.0
6	Franek	11.50	53.0
7	Julia	9.00	42.0
8	Basia	8.50	26.0
9	Janek	14.50	74.0
10	Francesca	15.50	82.0
11	Rafał	13.75	62.0
12	Nina	9.00	37.0
13	Kacper	8.00	15.0

```
14      Joanna       15.50   70.0
15      Cezary        8.00   27.0
16      Helena        9.00   36.0
17         Iza        6.00   35.0
18        Anna       10.00   48.0
19     Staszek       12.00   52.0
20      Daniel       12.50   63.0
21      Kacper       12.00   64.0
```

[25]:
```
df_students = df_students.dropna(axis=0, how='any')
df_students
```

[25]:
```
          Name  StudyHours  Grade
0       Damian       10.00   50.0
1        Krzyś       11.50   50.0
2       Sławek        9.00   47.0
3      Roksana       16.00   97.0
4        Natan        9.25   49.0
5     Wiktoria        1.00    3.0
6       Franek       11.50   53.0
7        Julia        9.00   42.0
8        Basia        8.50   26.0
9        Janek       14.50   74.0
10  Francesca       15.50   82.0
11      Rafał       13.75   62.0
12       Nina        9.00   37.0
13     Kacper        8.00   15.0
14     Joanna       15.50   70.0
15     Cezary        8.00   27.0
16     Helena        9.00   36.0
17        Iza        6.00   35.0
18       Anna       10.00   48.0
19    Staszek       12.00   52.0
20     Daniel       12.50   63.0
21     Kacper       12.00   64.0
```

[26]:
```
# Get the mean study hours using to column name as an index
mean_study = df_students['StudyHours'].mean()

# Get the mean grade using the column name as a property (just to make the␣
 ↪point!)
mean_grade = df_students['Grade'].mean()

# Print the mean study hours and mean grade
print('Average weekly study hours: {:.2f}\nAverage grade: {:.2f}'.
 ↪format(mean_study, mean_grade))
```

```
Average weekly study hours: 10.52
Average grade: 49.18
```

[27]:
```
# Get students who studied for the mean or more hours
df_students[df_students.StudyHours > mean_study]
```

[27]:

	Name	StudyHours	Grade
1	Krzyś	11.50	50.0
3	Roksana	16.00	97.0
6	Franek	11.50	53.0
9	Janek	14.50	74.0
10	Francesca	15.50	82.0
11	Rafał	13.75	62.0
14	Joanna	15.50	70.0
19	Staszek	12.00	52.0
20	Daniel	12.50	63.0
21	Kacper	12.00	64.0

[28]:
```
# What was their mean grade?
df_students[df_students.StudyHours > mean_study].Grade.mean()
```

[28]:
```
66.7
```

[29]:
```
passes  = pd.Series(df_students['Grade'] >= 60)
df_students = pd.concat([df_students, passes.rename("Pass")], axis=1)

df_students
```

[29]:

	Name	StudyHours	Grade	Pass
0	Damian	10.00	50.0	False
1	Krzyś	11.50	50.0	False
2	Sławek	9.00	47.0	False
3	Roksana	16.00	97.0	True
4	Natan	9.25	49.0	False
5	Wiktoria	1.00	3.0	False
6	Franek	11.50	53.0	False
7	Julia	9.00	42.0	False
8	Basia	8.50	26.0	False
9	Janek	14.50	74.0	True
10	Francesca	15.50	82.0	True
11	Rafał	13.75	62.0	True
12	Nina	9.00	37.0	False
13	Kacper	8.00	15.0	False
14	Joanna	15.50	70.0	True
15	Cezary	8.00	27.0	False
16	Helena	9.00	36.0	False
17	Iza	6.00	35.0	False

```
18       Anna        10.00   48.0  False
19    Staszek        12.00   52.0  False
20     Daniel        12.50   63.0   True
21     Kacper        12.00   64.0   True
```

[30]:
```
print(df_students.groupby(df_students.Pass).Name.count())
```

```
Pass
False    15
True      7
Name: Name, dtype: int64
```

[32]:
```
print(df_students.groupby(df_students.Pass)['StudyHours', 'Grade'].mean())
```

```
       StudyHours      Grade
Pass
False    8.783333  38.000000
True    14.250000  73.142857
```

```
/tmp/ipykernel_360/2502225861.py:1: FutureWarning: Indexing with multiple keys
(implicitly converted to a tuple of keys) will be deprecated, use a list
instead.
  print(df_students.groupby(df_students.Pass)['StudyHours', 'Grade'].mean())
```

[33]:
```
# Create a DataFrame with the data sorted by Grade (descending)
df_students = df_students.sort_values('Grade', ascending=False)

# Show the DataFrame
df_students
```

[33]:
```
          Name  StudyHours  Grade   Pass
3     Roksana        16.00   97.0   True
10  Francesca        15.50   82.0   True
9       Janek        14.50   74.0   True
14     Joanna        15.50   70.0   True
21     Kacper        12.00   64.0   True
20     Daniel        12.50   63.0   True
11      Rafał        13.75   62.0   True
6      Franek        11.50   53.0  False
19    Staszek        12.00   52.0  False
1       Krzyś        11.50   50.0  False
0      Damian        10.00   50.0  False
4       Natan         9.25   49.0  False
18       Anna        10.00   48.0  False
2      Sławek         9.00   47.0  False
7       Julia         9.00   42.0  False
12       Nina         9.00   37.0  False
16     Helena         9.00   36.0  False
```

```
17        Iza         6.00   35.0  False
15     Cezary         8.00   27.0  False
8       Basia         8.50   26.0  False
13     Kacper         8.00   15.0  False
5    Wiktoria         1.00    3.0  False
```

[34]:
```
import pandas as pd


df_students
```

[34]:
```
          Name  StudyHours  Grade   Pass
3     Roksana       16.00   97.0   True
10  Francesca       15.50   82.0   True
9       Janek       14.50   74.0   True
14     Joanna       15.50   70.0   True
21     Kacper       12.00   64.0   True
20     Daniel       12.50   63.0   True
11      Rafał       13.75   62.0   True
6      Franek       11.50   53.0  False
19    Staszek       12.00   52.0  False
1       Krzyś       11.50   50.0  False
0      Damian       10.00   50.0  False
4       Natan        9.25   49.0  False
18       Anna       10.00   48.0  False
2      Sławek        9.00   47.0  False
7       Julia        9.00   42.0  False
12       Nina        9.00   37.0  False
16     Helena        9.00   36.0  False
17        Iza        6.00   35.0  False
15     Cezary        8.00   27.0  False
8       Basia        8.50   26.0  False
13     Kacper        8.00   15.0  False
5    Wiktoria        1.00    3.0  False
```

[45]:
```
df_students = pd.DataFrame({'Imie': ['Damian', 'Krzyś', 'Sławek', 'Roksana',␣
↪'Natan', 'Wiktoria', 'Franek', 'Julia',
                                     'Basia', 'Janek', 'Francesca', 'Rafał',␣
↪'Nina', 'Kacper', 'Joanna',
                                    ␣
↪'Cezary','Helena','Iza','Anna','Staszek','Daniel','Kacper', ],
                            'Czas_nauki':student_data[0],
                            'Stopnie':student_data[1]})

df_students
```

[45]:

	Imie	Czas_nauki	Stopnie
0	Damian	10.00	50.0
1	Krzyś	11.50	50.0
2	Sławek	9.00	47.0
3	Roksana	16.00	97.0
4	Natan	9.25	49.0
5	Wiktoria	1.00	3.0
6	Franek	11.50	53.0
7	Julia	9.00	42.0
8	Basia	8.50	26.0
9	Janek	14.50	74.0
10	Francesca	15.50	82.0
11	Rafał	13.75	62.0
12	Nina	9.00	37.0
13	Kacper	8.00	15.0
14	Joanna	15.50	70.0
15	Cezary	8.00	27.0
16	Helena	9.00	36.0
17	Iza	6.00	35.0
18	Anna	10.00	48.0
19	Staszek	12.00	52.0
20	Daniel	12.50	63.0
21	Kacper	12.00	64.0

[54]:
```
# Remove any rows with missing data
df_students = df_students.dropna(axis=0, how='any')
df_students
```

[54]:

	Imie	Czas_nauki	Stopnie
0	Damian	10.00	50.0
1	Krzyś	11.50	50.0
2	Sławek	9.00	47.0
3	Roksana	16.00	97.0
4	Natan	9.25	49.0
5	Wiktoria	1.00	3.0
6	Franek	11.50	53.0
7	Julia	9.00	42.0
8	Basia	8.50	26.0
9	Janek	14.50	74.0
10	Francesca	15.50	82.0
11	Rafał	13.75	62.0
12	Nina	9.00	37.0
13	Kacper	8.00	15.0
14	Joanna	15.50	70.0
15	Cezary	8.00	27.0
16	Helena	9.00	36.0
17	Iza	6.00	35.0

```
18       Anna        10.00     48.0
19    Staszek        12.00     52.0
20     Daniel        12.50     63.0
21     Kacper        12.00     64.0
```

[55]:
```
# Calculate who passed, assuming '60' is the grade needed to pass
passes  = pd.Series(df_students['Stopnie'] >= 60)
df_students
```

[55]:
```
          Imie  Czas_nauki  Stopnie
0       Damian        10.00     50.0
1        Krzyś        11.50     50.0
2       Sławek         9.00     47.0
3      Roksana        16.00     97.0
4        Natan         9.25     49.0
5     Wiktoria         1.00      3.0
6       Franek        11.50     53.0
7        Julia         9.00     42.0
8        Basia         8.50     26.0
9        Janek        14.50     74.0
10   Francesca        15.50     82.0
11       Rafał        13.75     62.0
12        Nina         9.00     37.0
13      Kacper         8.00     15.0
14      Joanna        15.50     70.0
15      Cezary         8.00     27.0
16      Helena         9.00     36.0
17         Iza         6.00     35.0
18        Anna        10.00     48.0
19     Staszek        12.00     52.0
20      Daniel        12.50     63.0
21      Kacper        12.00     64.0
```

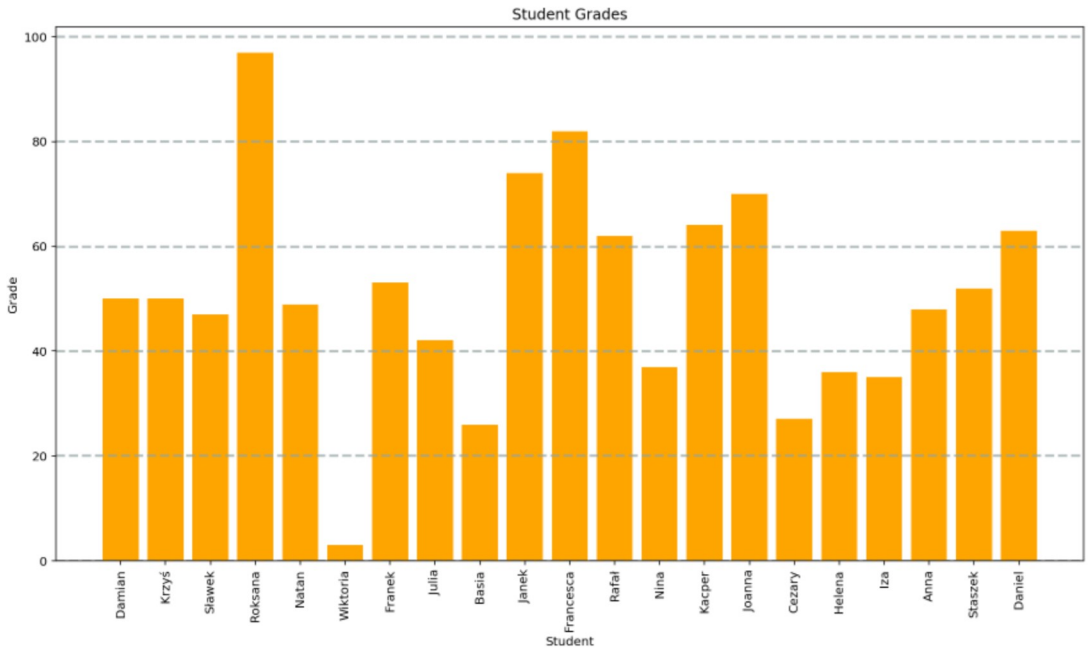
[56]:
```
# Save who passed to the Pandas dataframe
df_students = pd.concat([df_students, passes.rename("Pass")], axis=1)
df_students
```

[56]:
```
          Imie  Czas_nauki  Stopnie   Pass
0       Damian        10.00     50.0  False
1        Krzyś        11.50     50.0  False
2       Sławek         9.00     47.0  False
3      Roksana        16.00     97.0   True
4        Natan         9.25     49.0  False
5     Wiktoria         1.00      3.0  False
6       Franek        11.50     53.0  False
7        Julia         9.00     42.0  False
8        Basia         8.50     26.0  False
```

```
9       Janek       14.50     74.0   True
10  Francesca       15.50     82.0   True
11      Rafał       13.75     62.0   True
12       Nina        9.00     37.0  False
13     Kacper        8.00     15.0  False
14     Joanna       15.50     70.0   True
15     Cezary        8.00     27.0  False
16     Helena        9.00     36.0  False
17        Iza        6.00     35.0  False
18       Anna       10.00     48.0  False
19    Staszek       12.00     52.0  False
20     Daniel       12.50     63.0   True
21     Kacper       12.00     64.0   True
```

[79]:
```
# Create a Figure
fig = plt.figure(figsize=(15,8))

# Create a bar plot of name vs grade
plt.bar(x=df_students.Imie, height=df_students.Stopnie, color='orange')

# Customize the chart
plt.title('Student Grades')
plt.xlabel('Student')
plt.ylabel('Grade')
plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.7)
plt.xticks(rotation=90)

# Show the figure
plt.show()
```

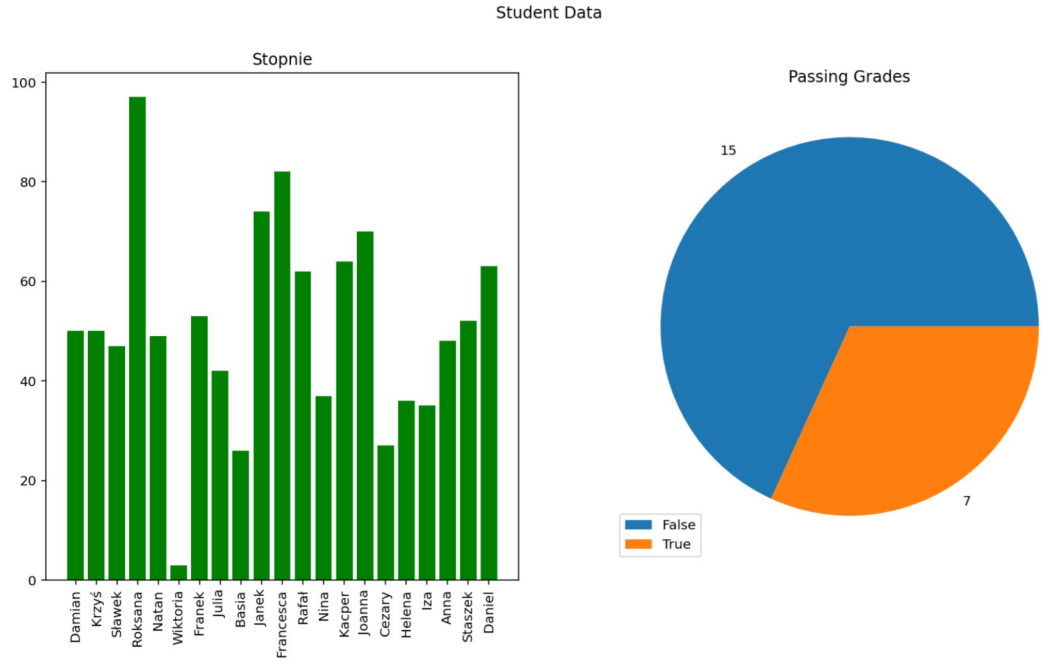
[79]:



[103]:
```
# Create a figure for 2 subplots (1 row, 2 columns)
fig, ax = plt.subplots(1, 2, figsize = (14,7))

# Create a bar plot of name vs grade on the first axis
ax[0].bar(x=df_students.Imie, height=df_students.Stopnie, color='green')
ax[0].set_title('Stopnie')
ax[0].set_xticklabels(df_students.Imie, rotation=90)

# Create a pie chart of pass counts on the second axis
pass_counts = df_students['Pass'].value_counts()
ax[1].pie(pass_counts, labels=pass_counts)
ax[1].set_title('Passing Grades')
ax[1].legend(pass_counts.keys().tolist())

# Add a title to the Figure
fig.suptitle('Student Data')

# Show the figure
fig.show()
```

```
/tmp/ipykernel_360/1205890887.py:7: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax[0].set_xticklabels(df_students.Imie, rotation=90)
```

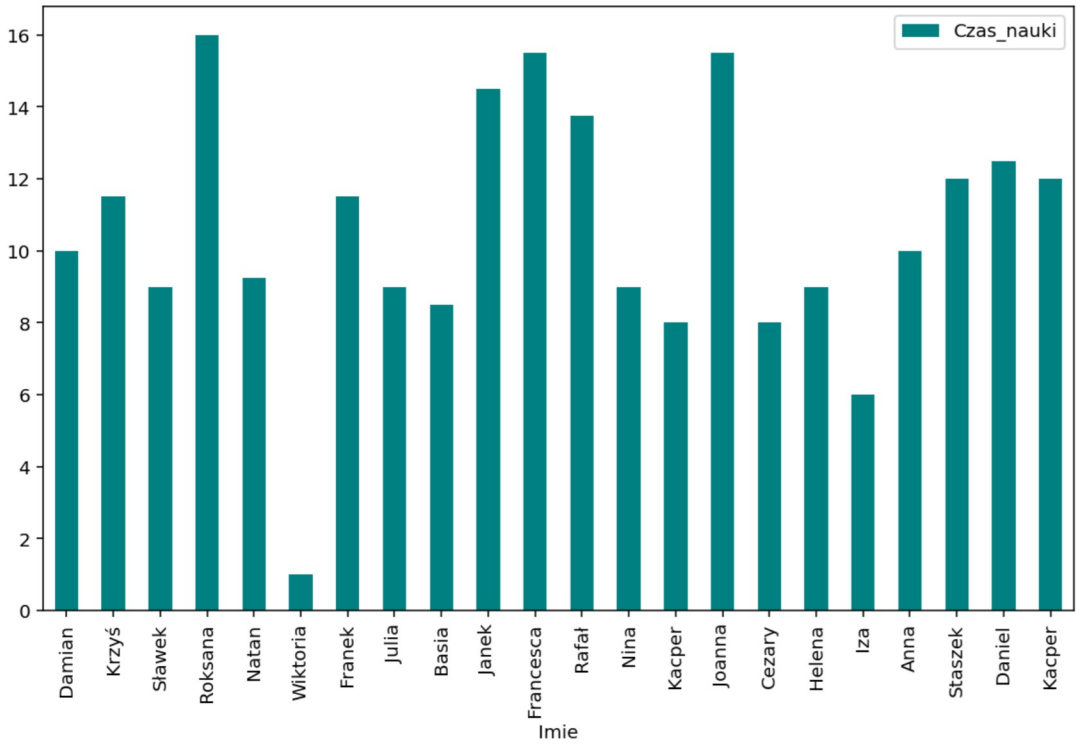
[103]:



[106]:
```
df_students.plot.bar(x='Imie', y='Czas_nauki', color='teal', figsize=(10,6))
```

[106]: <AxesSubplot: xlabel='Imie'>

[106]:

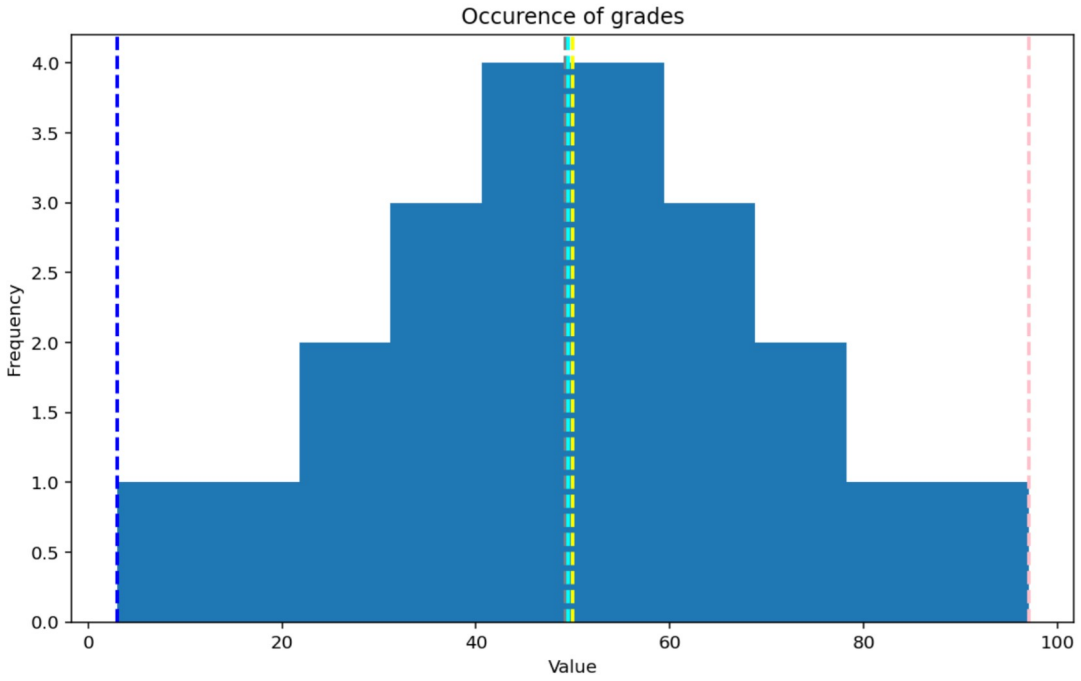


[111]:
```
# Get the variable to examine
var = df_students['Stopnie']

# Get statistics
min_val = var.min()
max_val = var.max()
mean_val = var.mean()
med_val = var.median()
mod_val = var.mode()[0]

print('Minimum:{:.2f}\nMean:{:.2f}\nMedian:{:.2f}\nMode:{:.2f}\nMaximum:{:.
 ↪2f}\n'.format(min_val,
                                                                          ␣
 ↪        mean_val,
                                                                          ␣
 ↪        med_val,
                                                                          ␣
 ↪        mod_val,
                                                                          ␣
 ↪        max_val))

# Create a Figure
fig = plt.figure(figsize=(10,6))

# Plot a histogram
plt.hist(var)

# Add lines for the statistics
plt.axvline(x=min_val, color = 'blue', linestyle='dashed', linewidth = 2)
plt.axvline(x=mean_val, color = 'gray', linestyle='dashed', linewidth = 2)
plt.axvline(x=med_val, color = 'cyan', linestyle='dashed', linewidth = 2)
plt.axvline(x=mod_val, color = 'yellow', linestyle='dashed', linewidth = 2)
plt.axvline(x=max_val, color = 'pink', linestyle='dashed', linewidth = 2)

# Add titles and labels
plt.title('Occurence of grades')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Show the figure
fig.show()
```

```
Minimum:3.00
Mean:49.18
Median:49.50
```

```
Mode:50.00
Maximum:97.00
```

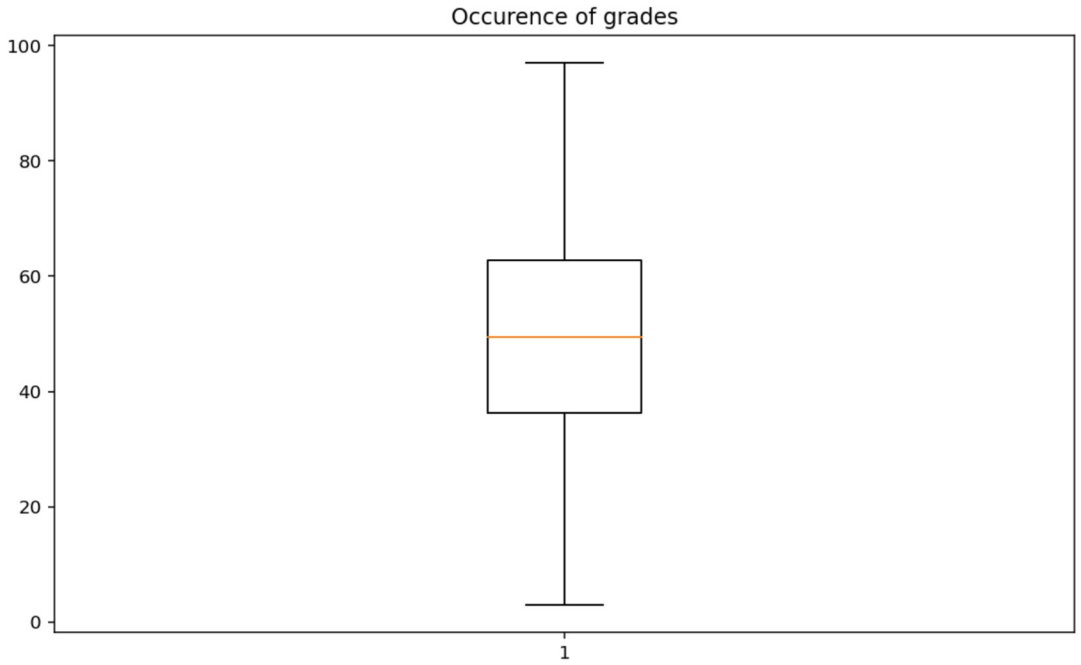
[111]:



[113]:

```
# Get the variable to examine
var = df_students['Stopnie']

# Create a Figure
fig = plt.figure(figsize=(10,6))

# Plot a histogram
plt.boxplot(var)

# Add titles and labels
plt.title('Occurence of grades')

# Show the figure
fig.show()
```

[113]:

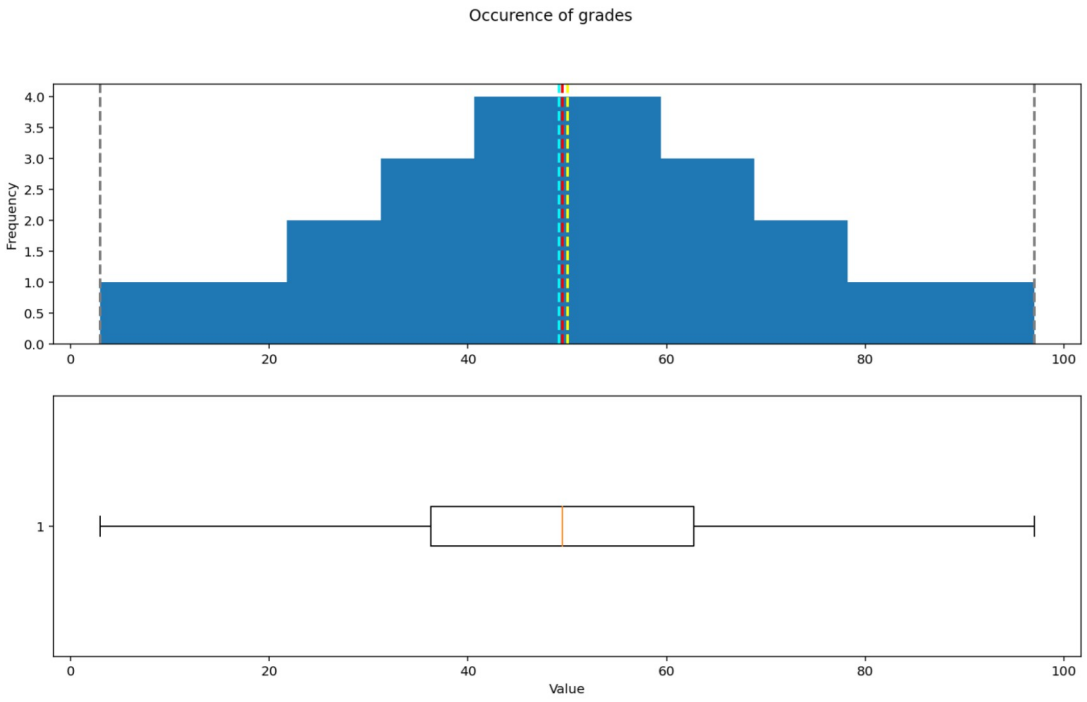


[117]:
```
# Create a function that we can re-use
def examine_occurence(var_data):
    from matplotlib import pyplot as plt

    # Get statistics
    min_val = var_data.min()
    max_val = var_data.max()
    mean_val = var_data.mean()
    med_val = var_data.median()
    mod_val = var_data.mode()[0]

    print('Minimum:{:.2f}\nMean:{:.2f}\nMedian:{:.2f}\nMode:{:.2f}\nMaximum:{:.
↪2f}\n'.format(min_val,
                                                                              ␣
↪               mean_val,
                                                                              ␣
↪               med_val,
                                                                              ␣
↪               mod_val,
                                                                              ␣
↪               max_val))

    # Create a figure for 2 subplots (2 rows, 1 column)
    fig, ax = plt.subplots(2, 1, figsize = (14,8))
```

```
    # Plot the histogram
    ax[0].hist(var_data)
    ax[0].set_ylabel('Frequency')

    # Add lines for the mean, median, and mode
    ax[0].axvline(x=min_val, color = 'gray', linestyle='dashed', linewidth = 2)
    ax[0].axvline(x=mean_val, color = 'cyan', linestyle='dashed', linewidth = 2)
    ax[0].axvline(x=med_val, color = 'red', linestyle='dashed', linewidth = 2)
    ax[0].axvline(x=mod_val, color = 'yellow', linestyle='dashed', linewidth =␣
↪2)
    ax[0].axvline(x=max_val, color = 'gray', linestyle='dashed', linewidth = 2)

    # Plot the boxplot
    ax[1].boxplot(var_data, vert=False)
    ax[1].set_xlabel('Value')

    # Add a title to the Figure
    fig.suptitle('Occurence of grades')

    # Show the figure
    fig.show()

# Get the variable to examine
col = df_students['Stopnie']

# Call the function - normal use of function!!!!!!
examine_occurence(col)
```

```
Minimum:3.00
Mean:49.18
Median:49.50
Mode:50.00
Maximum:97.00
```

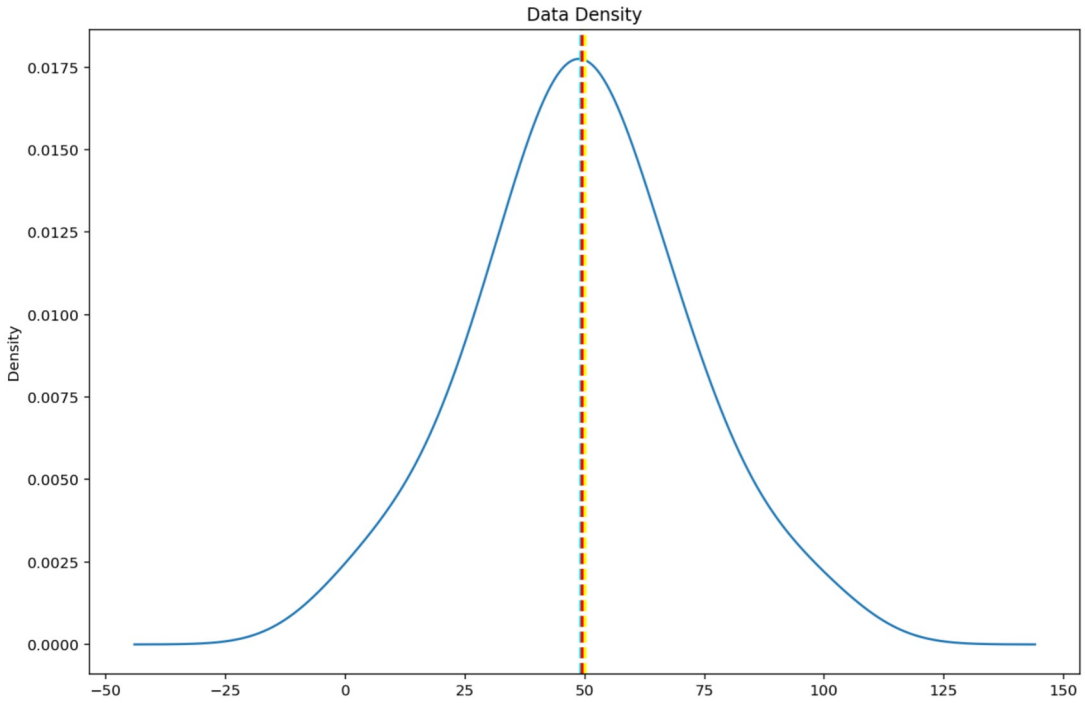
[117]:



[120]:
```
def examine_density(var_data):
    from matplotlib import pyplot as plt

    fig = plt.figure(figsize=(12,8))

    # Plot density
    var_data.plot.density()

    # Add titles and labels
    plt.title('Data Density')

    # Show the mean, median, and mode
    plt.axvline(x=var_data.mean(), color = 'cyan', linestyle='dashed',␣
 ↪linewidth = 2)
    plt.axvline(x=var_data.median(), color = 'red', linestyle='dashed',␣
 ↪linewidth = 2)
    plt.axvline(x=var_data.mode()[0], color = 'yellow', linestyle='dashed',␣
 ↪linewidth = 2)

    # Show the figure
    plt.show()

# Get the density of Grade
col = df_students['Stopnie']
```

```
#Use of function
examine_density(col)
```

[120]:

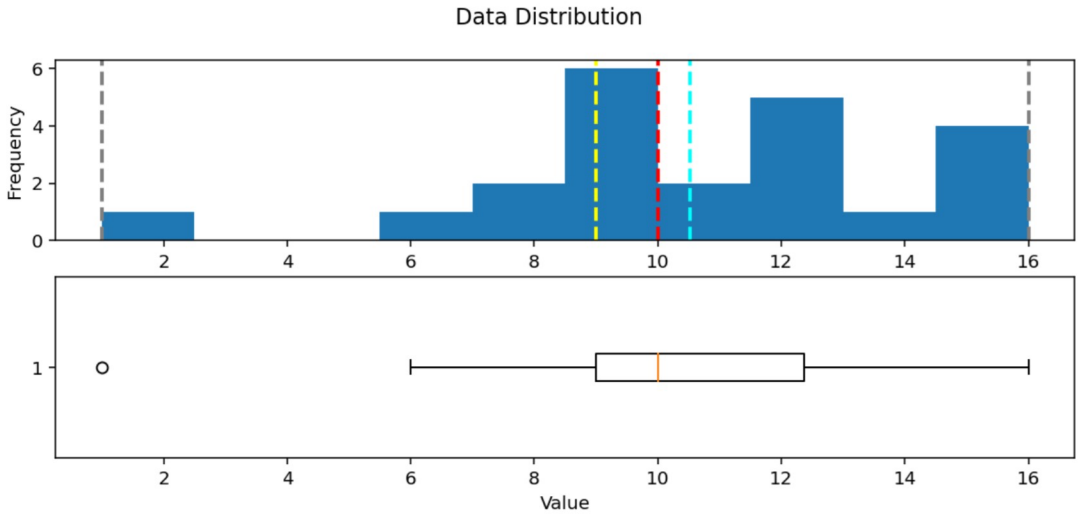


[127]:

```
# Get the variable to examine
col = df_students['Czas_nauki']

# Call the function
show_distribution(col)
```

```
Minimum:1.00
Mean:10.52
Median:10.00
Mode:9.00
Maximum:16.00
```

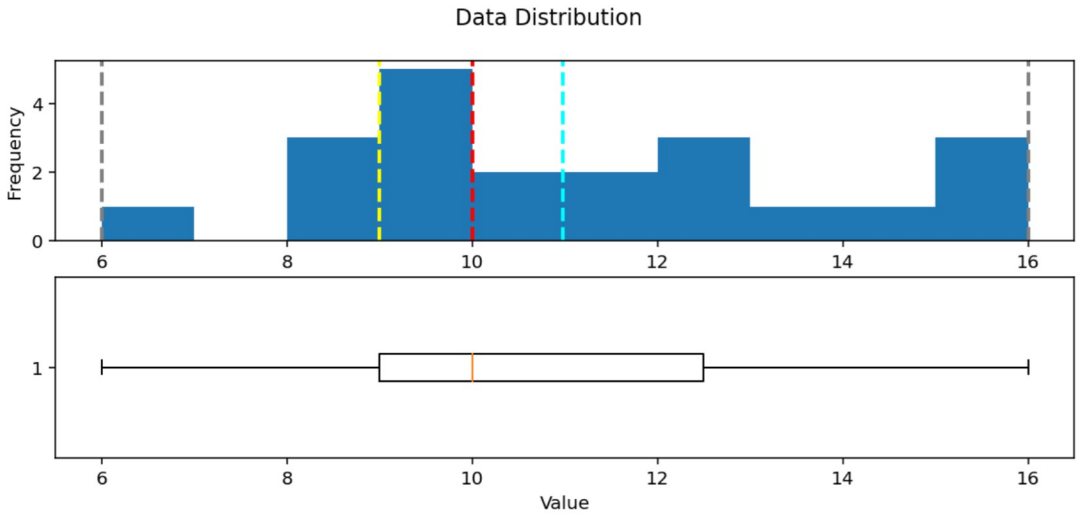
[127]:



[128]:
```
# Get the variable to examine
# We will only get students who have studied more than one hour
col = df_students[df_students.Czas_nauki>1]['Czas_nauki']

# Call the function
show_distribution(col)
```

```
Minimum:6.00
Mean:10.98
Median:10.00
Mode:9.00
Maximum:16.00
```

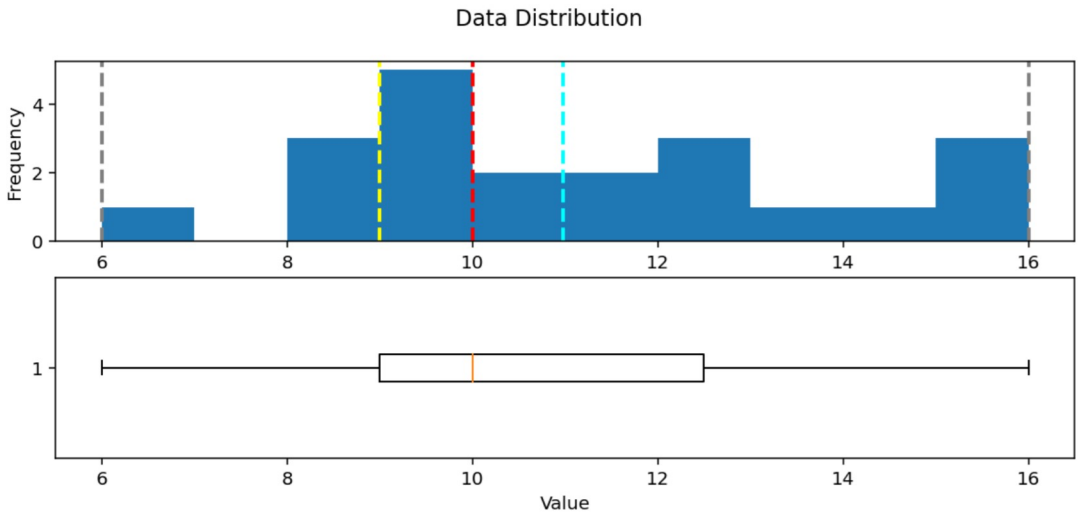
[128]:



[129]:
```
# calculate the 0.01th percentile
q01 = df_students.Czas_nauki.quantile(0.01)
# Get the variable to examine
col = df_students[df_students.Czas_nauki>q01]['Czas_nauki']
# Call the function
show_distribution(col)
```

```
Minimum:6.00
Mean:10.98
Median:10.00
Mode:9.00
Maximum:16.00
```

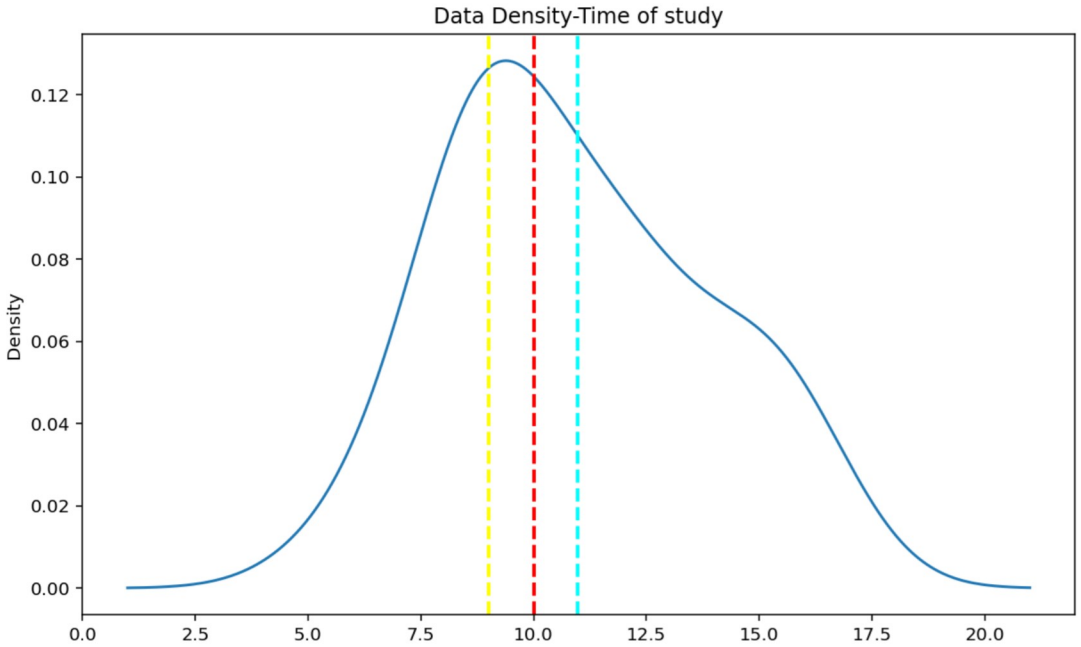
[129]:



[131]:
```
def show_density(var_data):
    fig = plt.figure(figsize=(10,6))

    # Plot density
    var_data.plot.density()

    # Add titles and labels
    plt.title('Data Density-Time of study')

    # Show the mean, median, and mode
    plt.axvline(x=var_data.mean(), color = 'cyan', linestyle='dashed',␣
 ↪linewidth = 2)
```

```
    plt.axvline(x=var_data.median(), color = 'red', linestyle='dashed',␣
 ↪linewidth = 2)
    plt.axvline(x=var_data.mode()[0], color = 'yellow', linestyle='dashed',␣
 ↪linewidth = 2)

    # Show the figure
    plt.show()

# Get the density of StudyHours
show_density(col)
```

[131]:



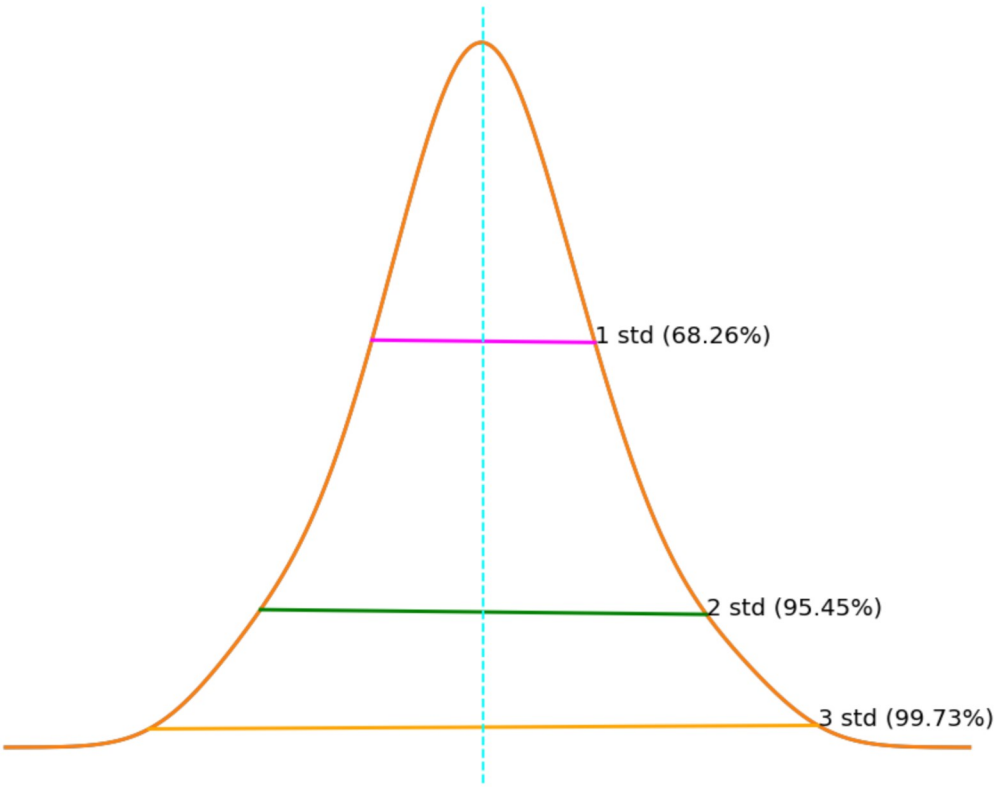
[133]:
```
for column_name in ['Stopnie','Czas_nauki']:
    col = df_students[col_name]
    rng = col.max() - col.min()
    var = col.var()
    std = col.std()
    print('\n{}:\n - Range: {:.2f}\n - Variance: {:.2f}\n - Std.Dev: {:.2f}'.
 ↪format(column_name, rng, var, std))
```

```
Stopnie:
 - Range: 15.00
 - Variance: 12.16
 - Std.Dev: 3.49
```

```
Czas_nauki:
 - Range: 15.00
 - Variance: 12.16
 - Std.Dev: 3.49
```

[139]:
```
import scipy.stats as stats

# Get the Grade column
col = df_students['Stopnie']

# get the density
density = stats.gaussian_kde(col)
fig, ax = plt.subplots(figsize=(8, 6))
col.plot.density(ax=ax)

# Plot the density
col.plot.density()

# Get the mean and standard deviation
s = col.std()
m = col.mean()

# Annotate 1 stdev
x1 = [m-s, m+s]
y1 = density(x1)
plt.plot(x1,y1, color='magenta')
plt.annotate('1 std (68.26%)', (x1[1],y1[1]))

# Annotate 2 stdevs
x2 = [m-(s*2), m+(s*2)]
y2 = density(x2)
plt.plot(x2,y2, color='green')
plt.annotate('2 std (95.45%)', (x2[1],y2[1]))

# Annotate 3 stdevs
x3 = [m-(s*3), m+(s*3)]
y3 = density(x3)
plt.plot(x3,y3, color='orange')
plt.annotate('3 std (99.73%)', (x3[1],y3[1]))

# Show the location of the mean
plt.axvline(col.mean(), color='cyan', linestyle='dashed', linewidth=1)

plt.axis('off')

plt.show()
```

[139]:



[140]:
```
                                                                    #COMPARING␣
↪DATA
```

[142]:
```
df_sample = df_students[df_students['Czas_nauki']>1]
df_sample
```

[142]:

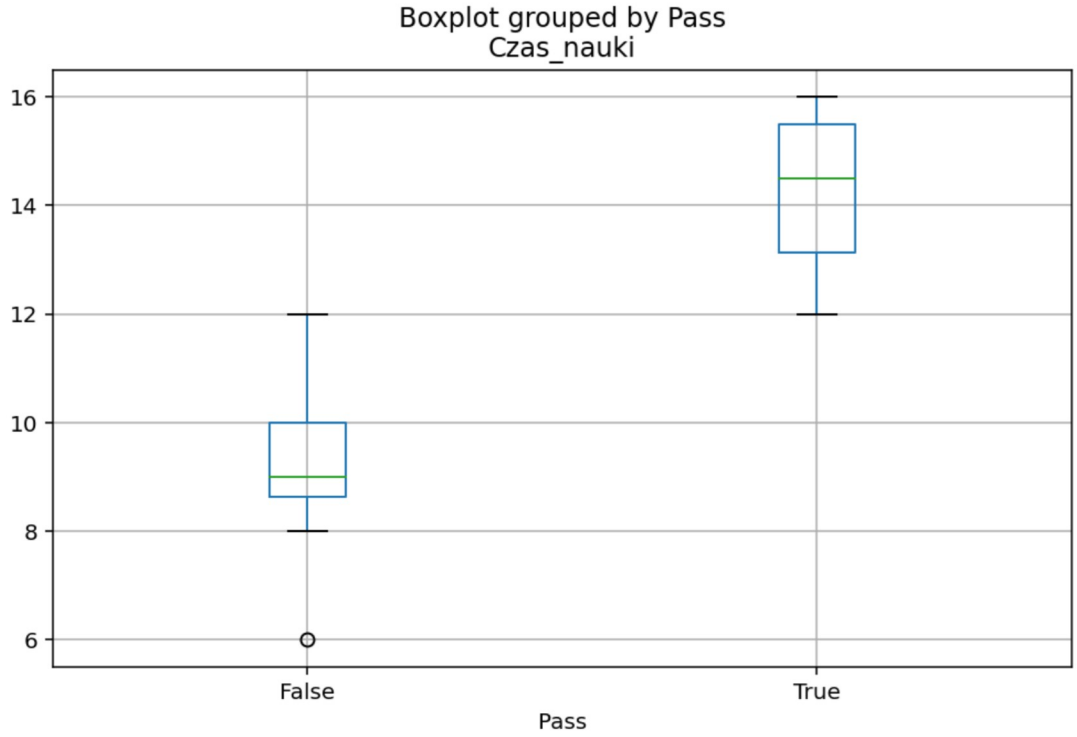
	Imie	Czas_nauki	Stopnie	Pass
0	Damian	10.00	50.0	False
1	Krzyś	11.50	50.0	False
2	Sławek	9.00	47.0	False
3	Roksana	16.00	97.0	True
4	Natan	9.25	49.0	False
6	Franek	11.50	53.0	False
7	Julia	9.00	42.0	False
8	Basia	8.50	26.0	False
9	Janek	14.50	74.0	True
10	Francesca	15.50	82.0	True
11	Rafał	13.75	62.0	True
12	Nina	9.00	37.0	False
13	Kacper	8.00	15.0	False
14	Joanna	15.50	70.0	True

```
15     Cezary        8.00     27.0  False
16     Helena        9.00     36.0  False
17        Iza        6.00     35.0  False
18       Anna       10.00     48.0  False
19    Staszek       12.00     52.0  False
20     Daniel       12.50     63.0   True
21     Kacper       12.00     64.0   True
```

[144]:
```
df_sample.boxplot(column='Czas_nauki', by='Pass', figsize=(8,5))
```

[144]:
```
<AxesSubplot: title={'center': 'Czas_nauki'}, xlabel='Pass'>
```

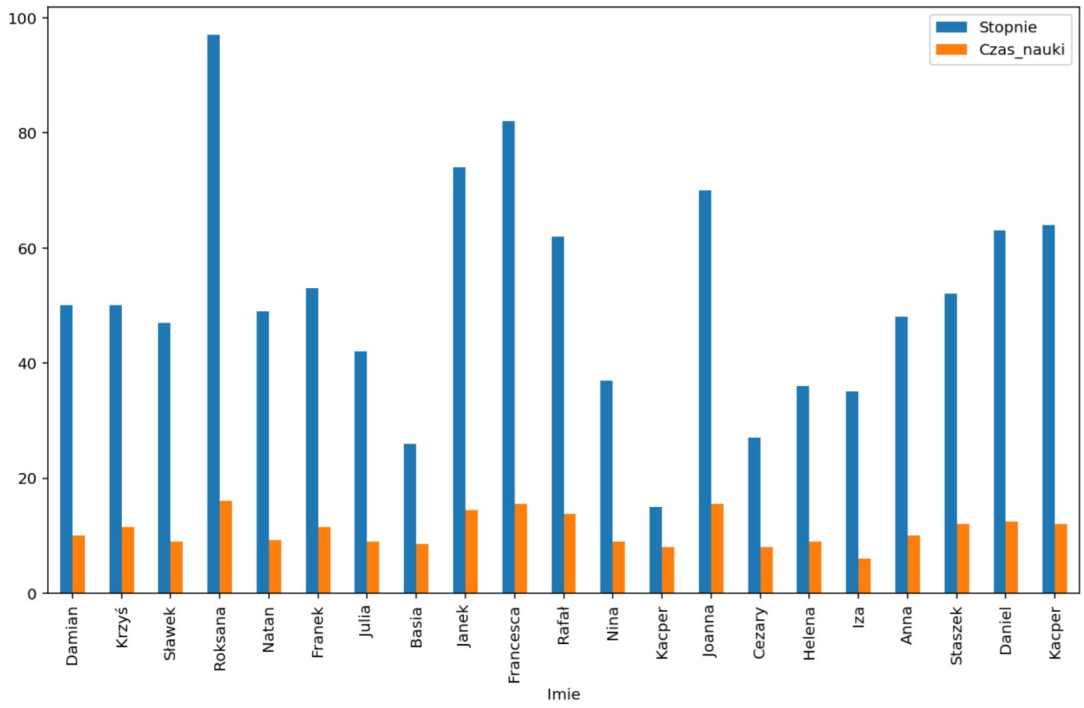
[144]:



[147]:
```
# Create a bar plot of name vs grade and study hours
df_sample.plot(x='Imie', y=['Stopnie','Czas_nauki'], kind='bar', figsize=(12,7))
```

[147]:
```
<AxesSubplot: xlabel='Imie'>
```

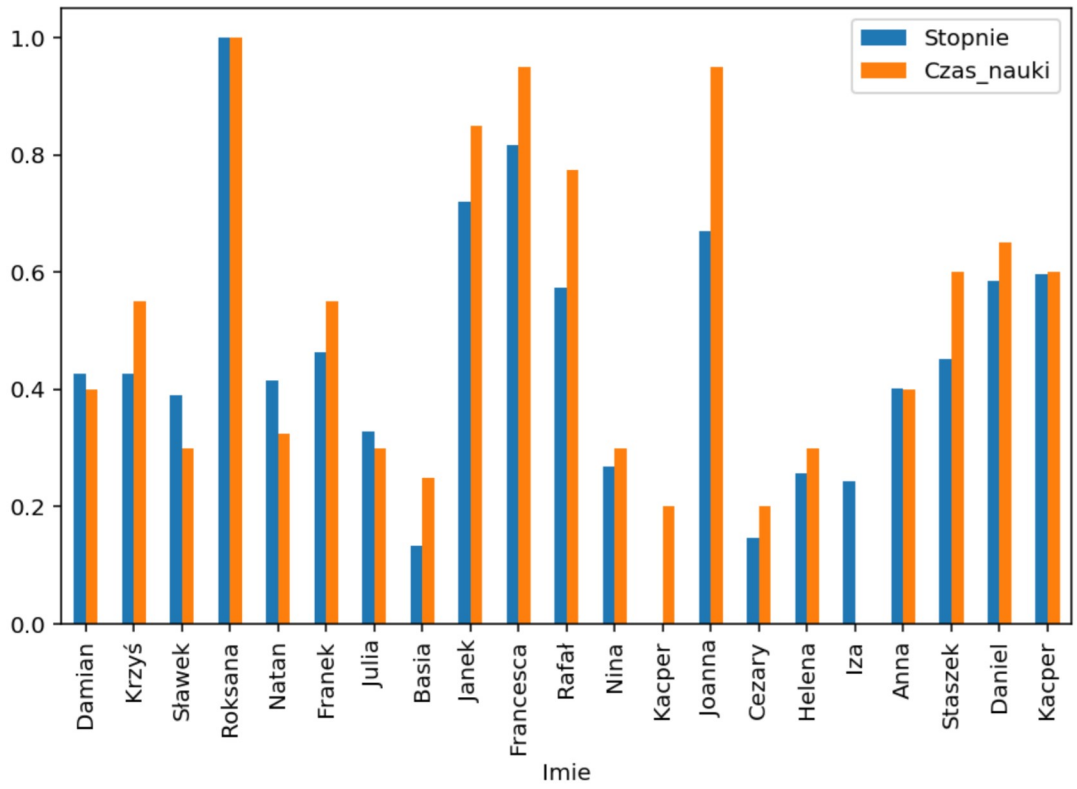
[147]:



[148]:
```
from sklearn.preprocessing import MinMaxScaler

# Get a scaler object
scaler = MinMaxScaler()

# Create a new dataframe for the scaled values
df_normalized = df_sample[['Imie', 'Stopnie', 'Czas_nauki']].copy()

# Normalize the numeric columns
df_normalized[['Stopnie','Czas_nauki']] = scaler.
 ↪fit_transform(df_normalized[['Stopnie','Czas_nauki']])

# Plot the normalized values
df_normalized.plot(x='Imie', y=['Stopnie','Czas_nauki'], kind='bar',␣
 ↪figsize=(8,5))
```

[148]: <AxesSubplot: xlabel='Imie'>

[148]:



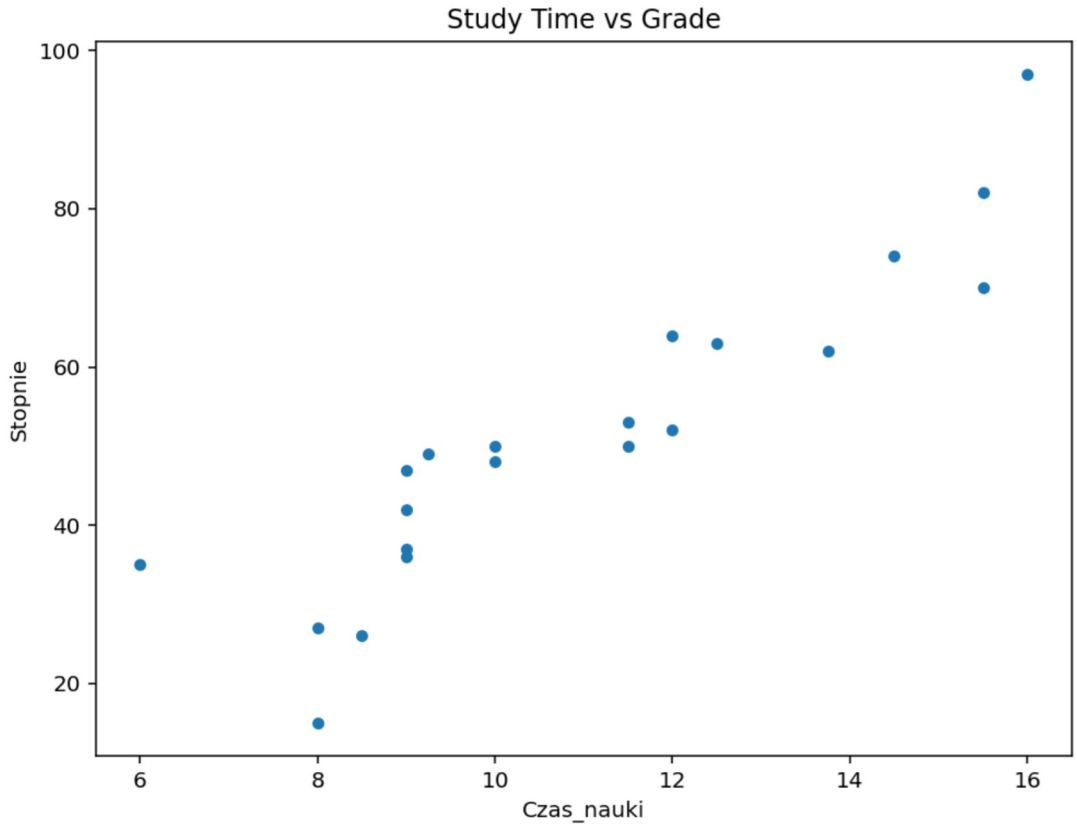
[149]:
```
df_normalized.Stopnie.corr(df_normalized.Czas_nauki)
```

[149]:
```
0.9117666413789677
```

[154]:
```
# Create a scatter plot
df_sample.plot.scatter(title='Study Time vs Grade', x='Czas_nauki',␣
↪y='Stopnie',figsize=(8, 6))
```

[154]:
```
<AxesSubplot: title={'center': 'Study Time vs Grade'}, xlabel='Czas_nauki',
ylabel='Stopnie'>
```

[154]:

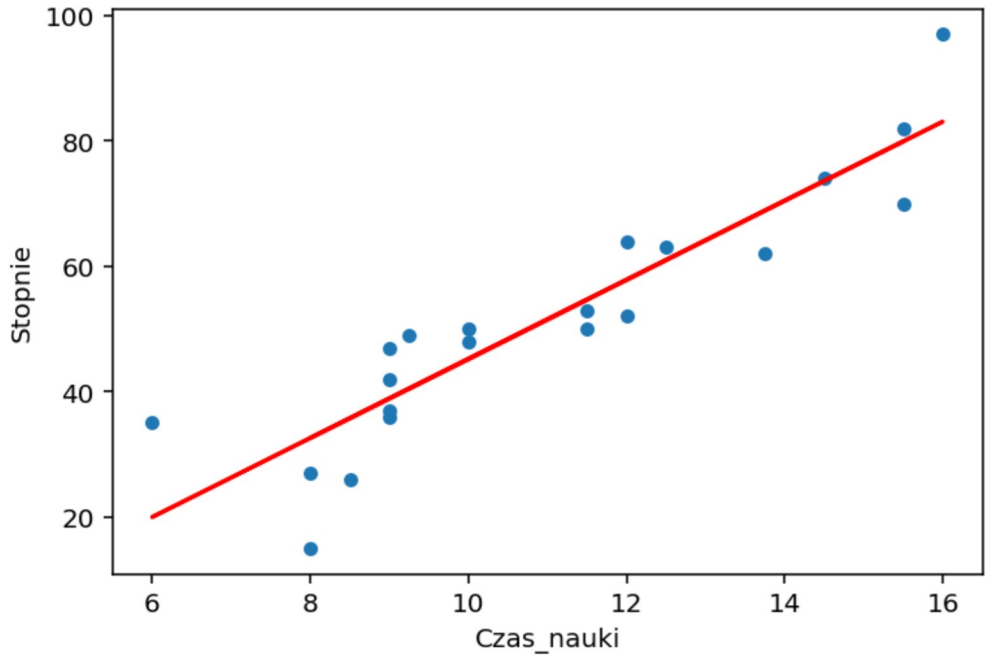


[156]:
```
from scipy import stats

#
df_regression = df_sample[['Stopnie', 'Czas_nauki']].copy()

# Get the regression slope and intercept
m, b, r, p, se = stats.linregress(df_regression['Czas_nauki'],␣
 ↪df_regression['Stopnie'])
print('slope: {:.4f}\ny-intercept: {:.4f}'.format(m,b))
print('so...\n f(x) = {:.4f}x + {:.4f}'.format(m,b))

# Use the function (mx + b) to calculate f(x) for each x (StudyHours) value
df_regression['fx'] = (m * df_regression['Czas_nauki']) + b

# Calculate the error between f(x) and the actual y (Grade) value
df_regression['error'] = df_regression['fx'] - df_regression['Stopnie']

# Create a scatter plot of Grade vs StudyHours
df_regression.plot.scatter(x='Czas_nauki', y='Stopnie')
```

```
# Plot the regression line
plt.plot(df_regression['Czas_nauki'],df_regression['fx'], color='red')

# Display the plot
plt.show()
```

```
slope: 6.3134
y-intercept: -17.9164
so…
 f(x) = 6.3134x + -17.9164
```

[156]:



[157]:
```
# Show the original x,y values, the f(x) value, and the error
df_regression[['Czas_nauki', 'Stopnie', 'fx', 'error']]
```

[157]:

	Czas_nauki	Stopnie	fx	error
0	10.00	50.0	45.217846	-4.782154
1	11.50	50.0	54.687985	4.687985
2	9.00	47.0	38.904421	-8.095579
3	16.00	97.0	83.098400	-13.901600
4	9.25	49.0	40.482777	-8.517223
6	11.50	53.0	54.687985	1.687985
7	9.00	42.0	38.904421	-3.095579
8	8.50	26.0	35.747708	9.747708
9	14.50	74.0	73.628262	-0.371738
10	15.50	82.0	79.941687	-2.058313

```
11       13.75     62.0  68.893193   6.893193
12        9.00     37.0  38.904421   1.904421
13        8.00     15.0  32.590995  17.590995
14       15.50     70.0  79.941687   9.941687
15        8.00     27.0  32.590995   5.590995
16        9.00     36.0  38.904421   2.904421
17        6.00     35.0  19.964144 -15.035856
18       10.00     48.0  45.217846  -2.782154
19       12.00     52.0  57.844698   5.844698
20       12.50     63.0  61.001410  -1.998590
21       12.00     64.0  57.844698  -6.155302
```

[159]:
```
#PREDICTIONS
```

[164]:
```
# Define a function based on our regression coefficients
def f(x):
    m = 6.3134
    b = -17.9164
    return m*x + b

study_time = 14

# Get f(x) for study time
prediction = f(study_time)

# Grade can't be less than 0 or more than 100
expected_grade = max(0,min(100,prediction))

#Print the estimated grade
print ('Studying for {} hours per week may result in a grade of {:.0f}'.
 ↪format(study_time, expected_grade))
print ('Nauka przez {} tygodniowo powinna dac uczniowi ocenę {:.0f}'.
 ↪format(study_time, expected_grade))
```

```
Studying for 14 hours per week may result in a grade of 70
Nauka przez 14 tygodniowo powinna dac uczniowi ocenę 70
```

[ ]: