Relationship between Hours of Study and Grades

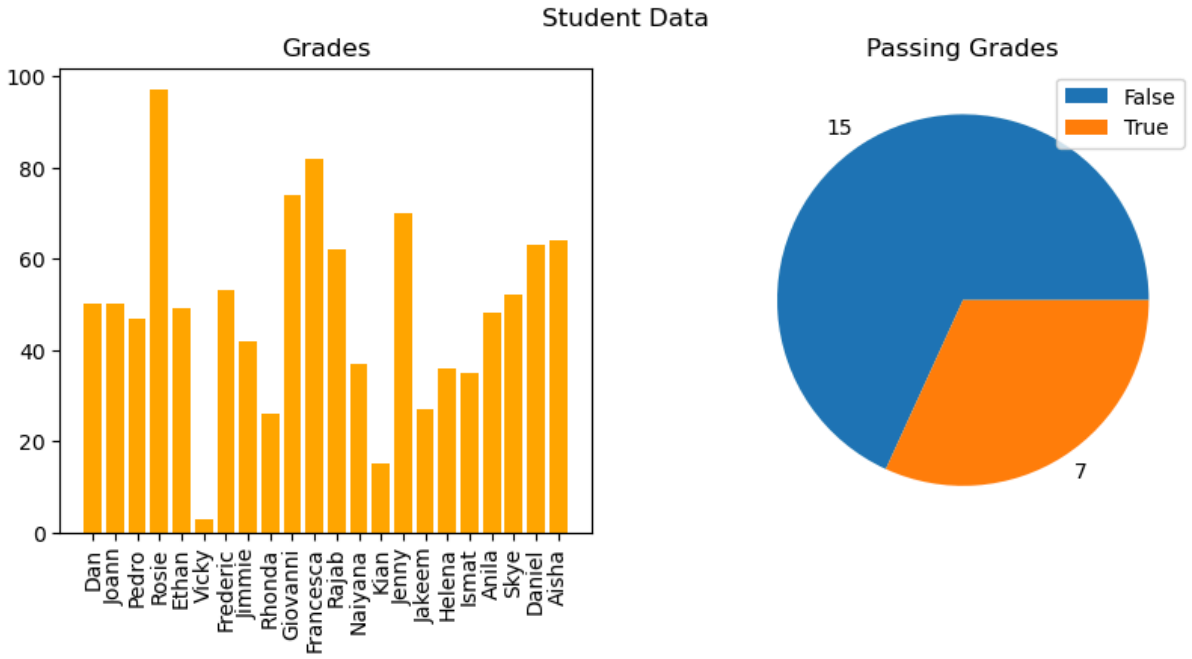
In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import wget
```

In [3]:
```
# Load data from a text file
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machi
df_students = pd.read_csv('grades.csv',delimiter=',',header='infer')

# Remove any rows with missing data
df_students = df_students.dropna(axis=0, how='any')

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

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


# Print the result out into this notebook
#print(df_students)
```

```
'wget' is not recognized as an internal or external command,
operable program or batch file.
```

In [5]:
```
fig, ax = plt.subplots(1, 2, figsize = (10,4))

# Create a bar plot of name vs grade on the first axis
ax[0].bar(x=df_students.Name, height=df_students.Grade, color='orange')
ax[0].set_title('Grades')
ax[0].set_xticklabels(df_students.Name, 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()
```

```
C:\Users\maria\AppData\Local\Temp\ipykernel_13560\3128466064.py:6: UserWarning: set_
ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks
() or using a FixedLocator.
  ax[0].set_xticklabels(df_students.Name, rotation=90)
C:\Users\maria\AppData\Local\Temp\ipykernel_13560\3128466064.py:18: UserWarning: Fig
ureCanvasAgg is non-interactive, and thus cannot be shown
  fig.show()
```

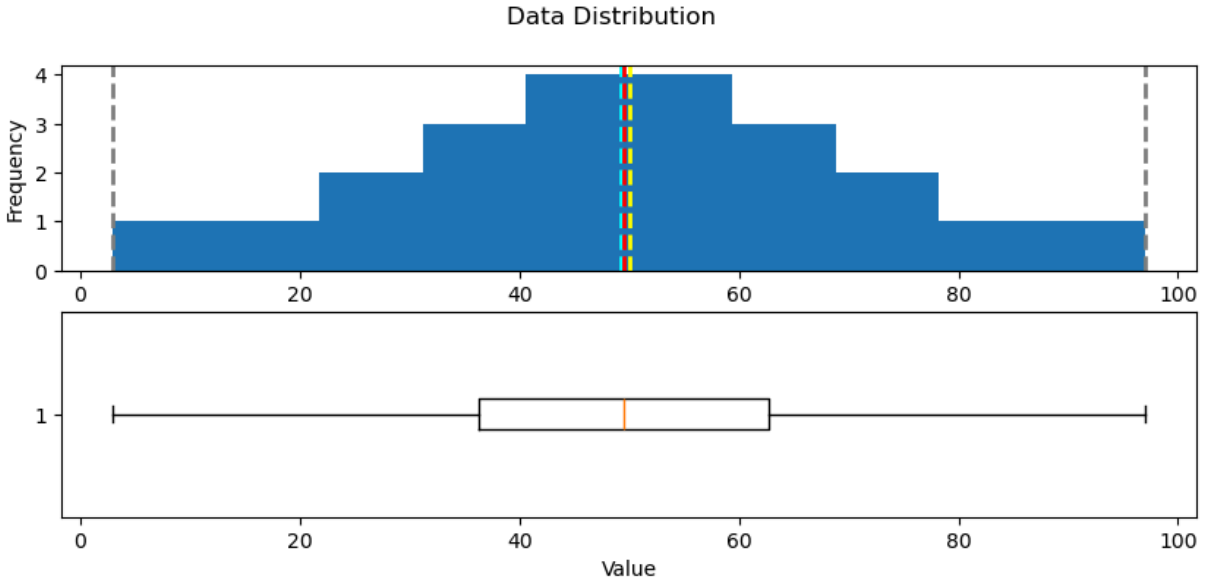


In [90]:
```python
# Create a function that we can re-use
def show_distribution(var_data):
    '''
    This function will make a distribution (graph) and display it
    '''

    # 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}\

    # Create a figure for 2 subplots (2 rows, 1 column)
    fig, ax = plt.subplots(2, 1, figsize = (10,4))

    # 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('Data Distribution')

    # Show the figure
    fig.show()


show_distribution(df_students['Grade'])
```

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

```
C:\Users\maria\AppData\Local\Temp\ipykernel_13788\2138730488.py:42: UserWarning: Fig
ureCanvasAgg is non-interactive, and thus cannot be shown
  fig.show()
```

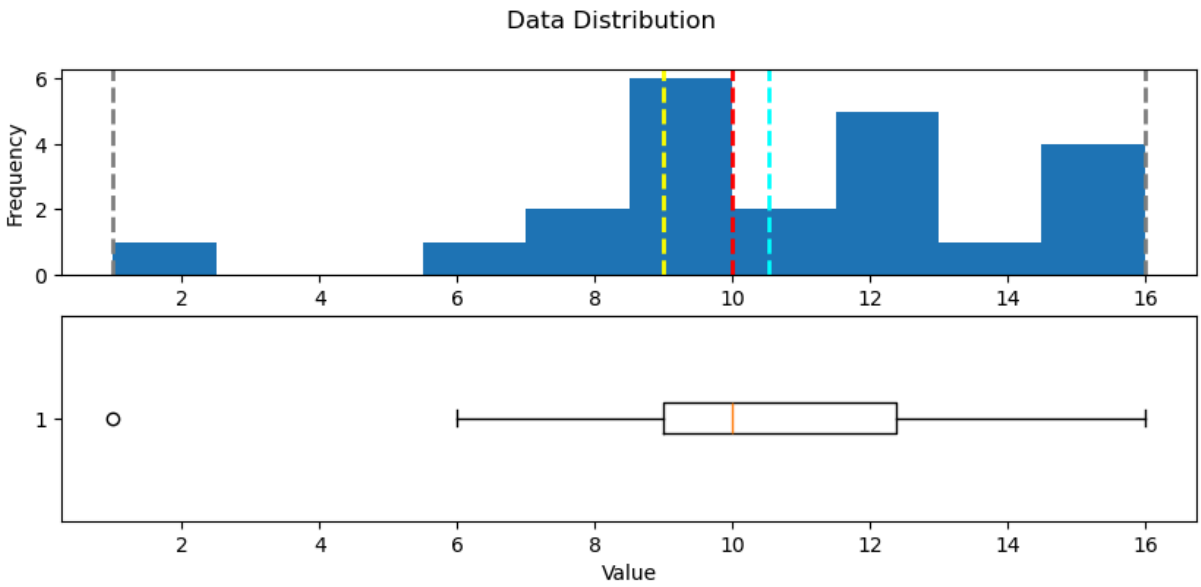


Let's look at the distribution of study hours

In [7]:
```
# Get the variable to examine
col = df_students['StudyHours']
# Call the function
show_distribution(col)
```

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

```
C:\Users\maria\AppData\Local\Temp\ipykernel_13788\547977180.py:60: UserWarning: Figu
reCanvasAgg is non-interactive, and thus cannot be shown
  fig.show()
```

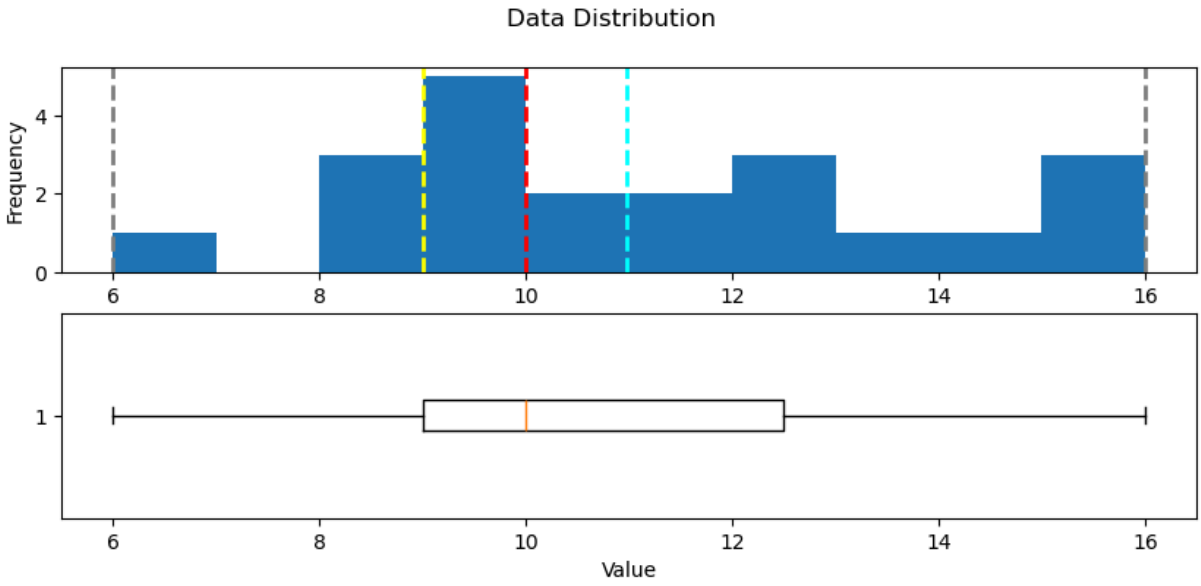


o is an outlier for this small sample, let's get rid of it:

In [10]:
```
# We will only get students who have studied more than one hour
col = df_students[df_students.StudyHours>1]['StudyHours']

# Call the function
show_distribution(col)
```

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

```
C:\Users\maria\AppData\Local\Temp\ipykernel_13788\547977180.py:60: UserWarning: Figu
reCanvasAgg is non-interactive, and thus cannot be shown
  fig.show()
```

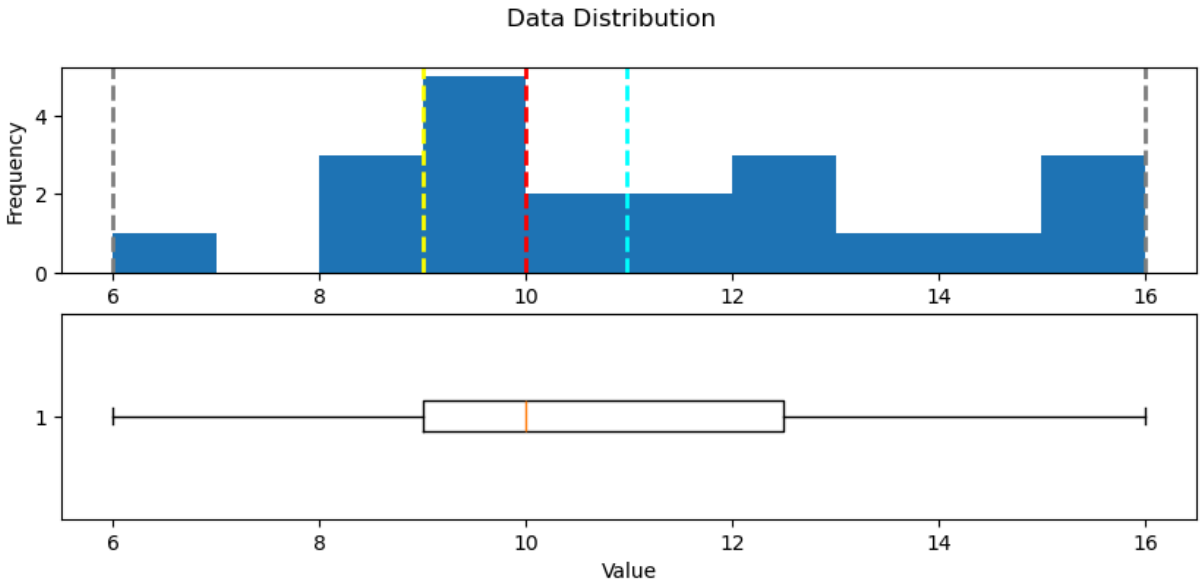


Alternatively, we can select upper or lower quantile of the data

In [21]:
```
# calculate the 0.01th percentile
q01 = df_students.StudyHours.quantile(0.01)

#Get the variables to examine
col = df_students[df_students.StudyHours>q01]['StudyHours']

#call the function
show_distribution(col)
```

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

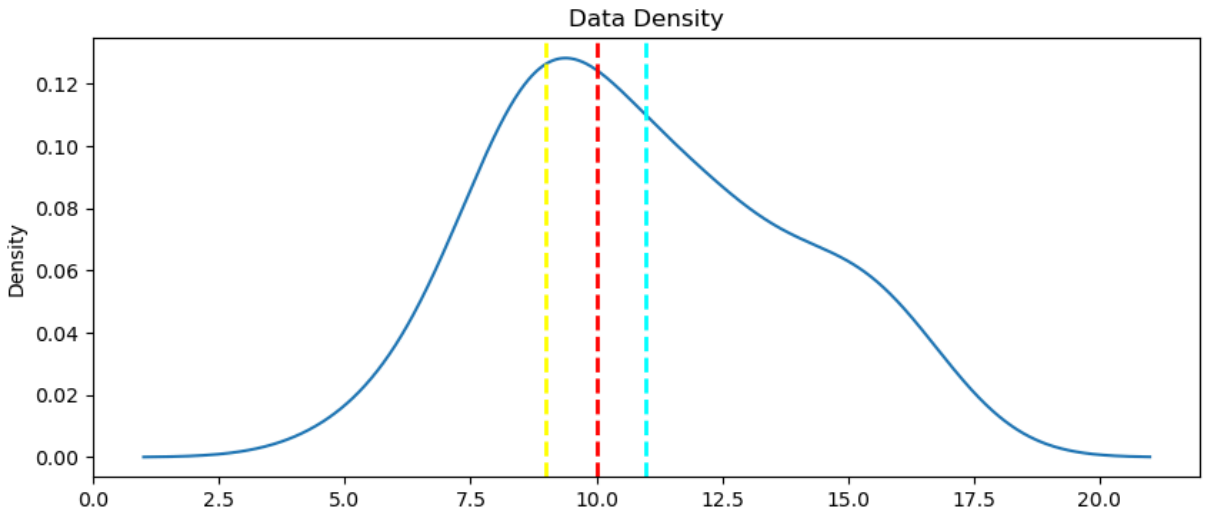
```
C:\Users\maria\AppData\Local\Temp\ipykernel_13788\547977180.py:60: UserWarning: Figu
reCanvasAgg is non-interactive, and thus cannot be shown
  fig.show()
```



Let's look at the density of the distribution

In [24]:
```
def show_density(var_data):
    fig = plt.figure(figsize=(10,4))

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

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

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



This kind of distribution is called **right skewed**. The mass of the data is on the left side of the distribution, creating a long tail to the right because of the values at the extreme high end, which pull the mean to the right.

Measure of Variance

So now we have a good idea where the middle of the grade and study hours data distributions are. However, there's another aspect of the distributions we should examine: how much variability is there in the data ?

Typical statistics that measure variability in the data inclu

Range nge: The difference between the maximum and minimum. There's no built-in function for this, but it's easy to calculate using the minmax functions.
i**Variance** ance: The average of the squared difference from the mean. You can use the built-in var function to find
Standard Deviationiation: The square root of the variance. You can use the built-in std function to fin d this.

In [47]:
```
for col_name in ['Grade', 'StudyHours']:
    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
```

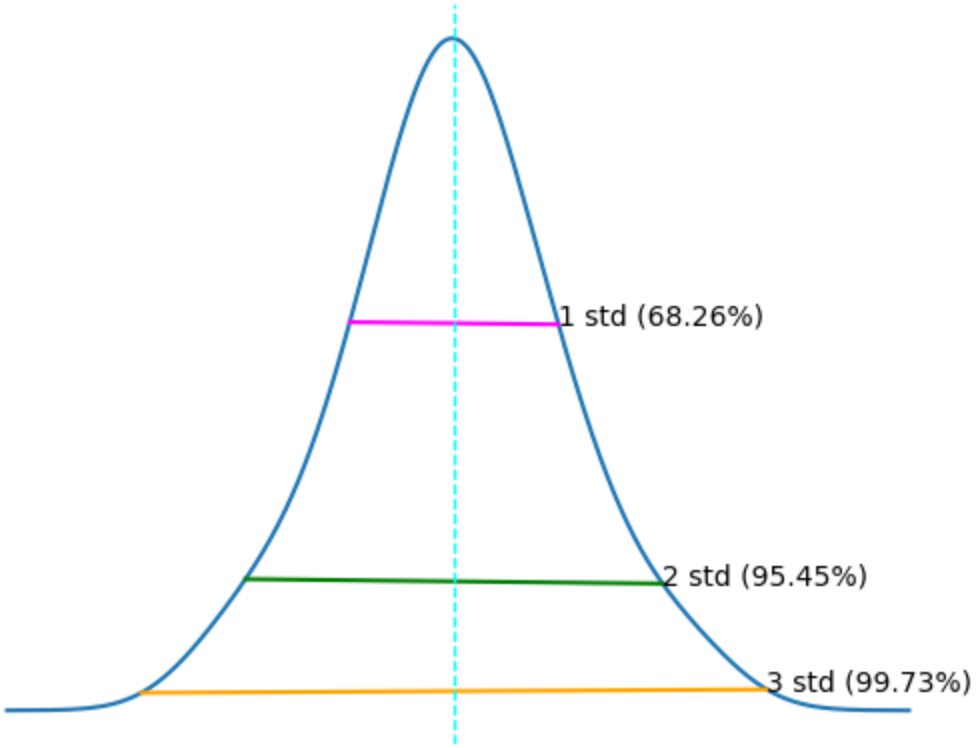
```
Grade:
 - Range: 94.00
 - Variance: 472.54
 - Std.Dev: 21.74

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

Of these statistics, the standard deviation is generally the most useful. It provides a measure of variance in the data on the same scale as the data itself (so grade points for the Grade distribution and hours for the StudyHours distribution). The higher the standard deviation, the more variance there is when comparing values in the distribution to the distribution mean; in other words, the data is more spread out.

When working with a normal distribution, the standard deviation works with the particular characteristics of a normal distribution to provide even greater insight. Run the following cell to see the relationship between standard deviations and the data in the normal distribution.

In [50]:
```
import scipy.stats as stats

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

# get the density
density = stats.gaussian_kde(col)

# 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()
```



The horizontal lines show the percentage of data within one, two, and three standard deviations of the mean (plus or minus).

In any normal distribution:

Approximately 68.26% of values fall within one standard deviation from the mean. Approximately 95.45% of values fall within two standard deviations from the mean. Approximately 99.73% of values fall within three standard deviations from the mean. So, because we know that the mean grade is 49.18, the standard deviation is 21.74, and distribution of grades is approximately normal, we can calculate that 68.26% of students should achieve a grade between 27.44 and 70.92.

The descriptive statistics we've used to understand the distribution of the student data variables are the basis of statistical analysis. Because they're such an important part of exploring your data, there's a built-in describe method of the DataFrame object that returns the main descriptive statistics for all numeric columns.

In [53]:
```
df_students.describe()
```

Out[53]:

	StudyHours	Grade
count	22.000000	22.000000
mean	10.522727	49.181818
std	3.487144	21.737912
min	1.000000	3.000000
25%	9.000000	36.250000
50%	10.000000	49.500000
75%	12.375000	62.750000
max	16.000000	97.000000

Comparing Data

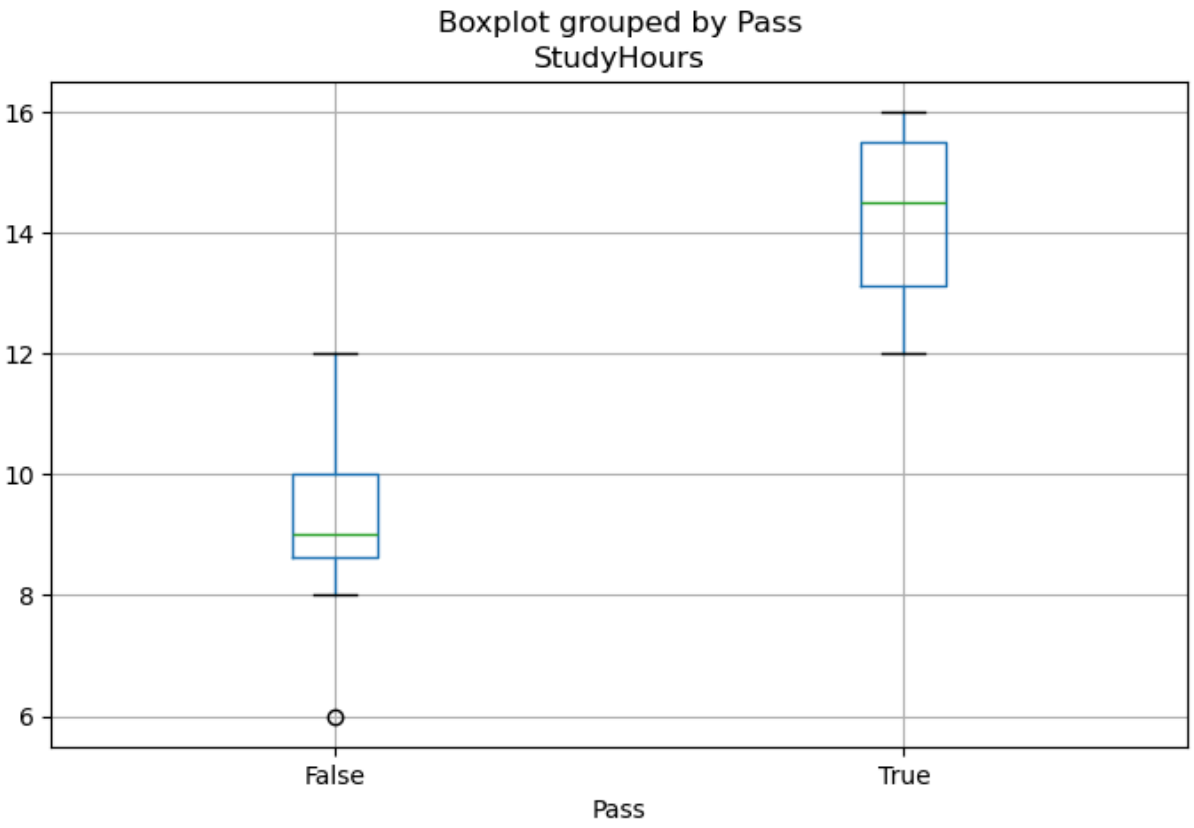
The data includes two numeric variables (StudyHours and Grade) and two categorical variables (Name and Pass). Let's start by comparing the numeric StudyHours column to the categorical Pass column to see if there's an apparent relationship between the number of hours studied and a passing grade.

To make this comparison, let's create box plots showing the distribution of StudyHours for each possible Pass value (true and false). columns.

In [60]:
```
df_sample = df_students[df_students['StudyHours']>1]
```

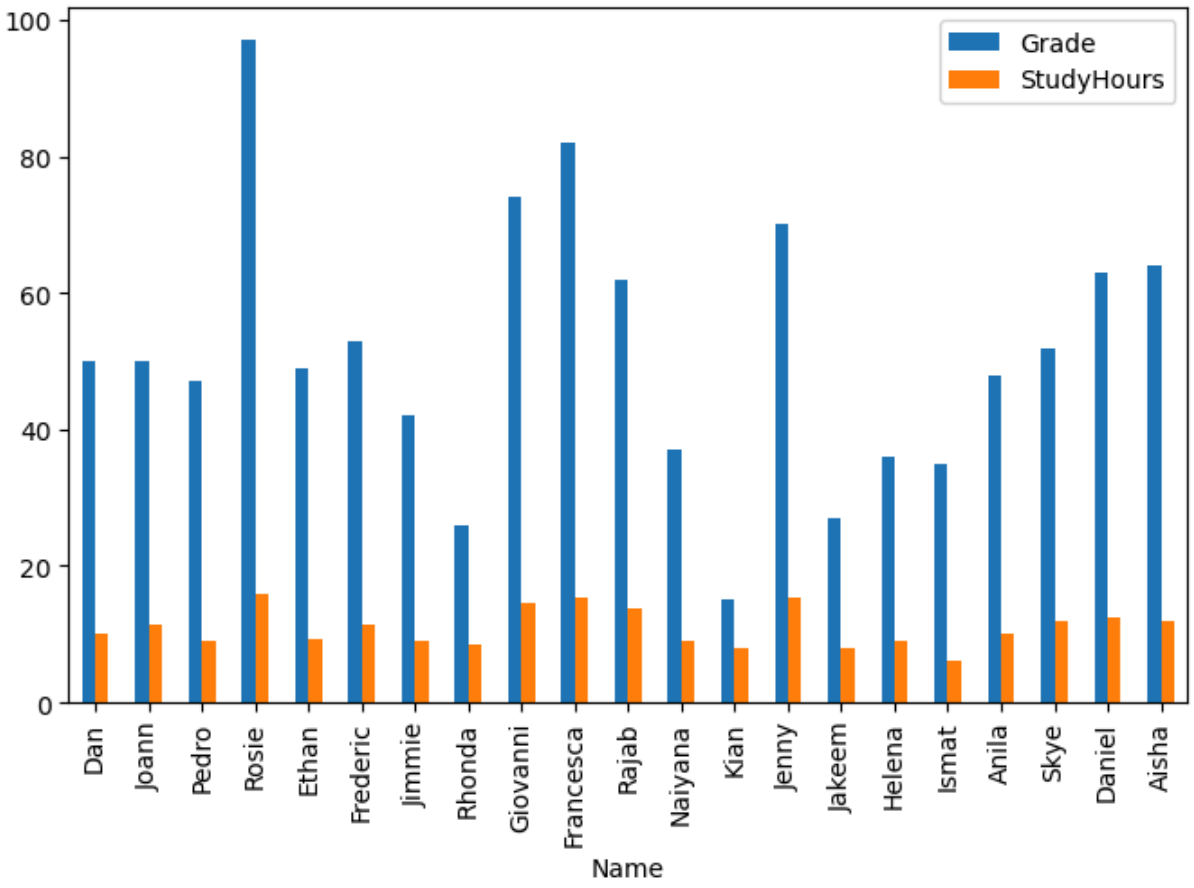
In [62]:
```
df_sample.boxplot(column='StudyHours', by='Pass', figsize=(8,5))
```

Out[62]:
```
<Axes: title={'center': 'StudyHours'}, xlabel='Pass'>
```



Comparing the StudyHours distributions, it's immediately apparent (if not particularly surprising) that students who passed the course tended to study for more hours than students who didn't. So if you wanted to predict whether or not a student is likely to pass the course, the amount of time they spend studying may be a good predictive indicator.

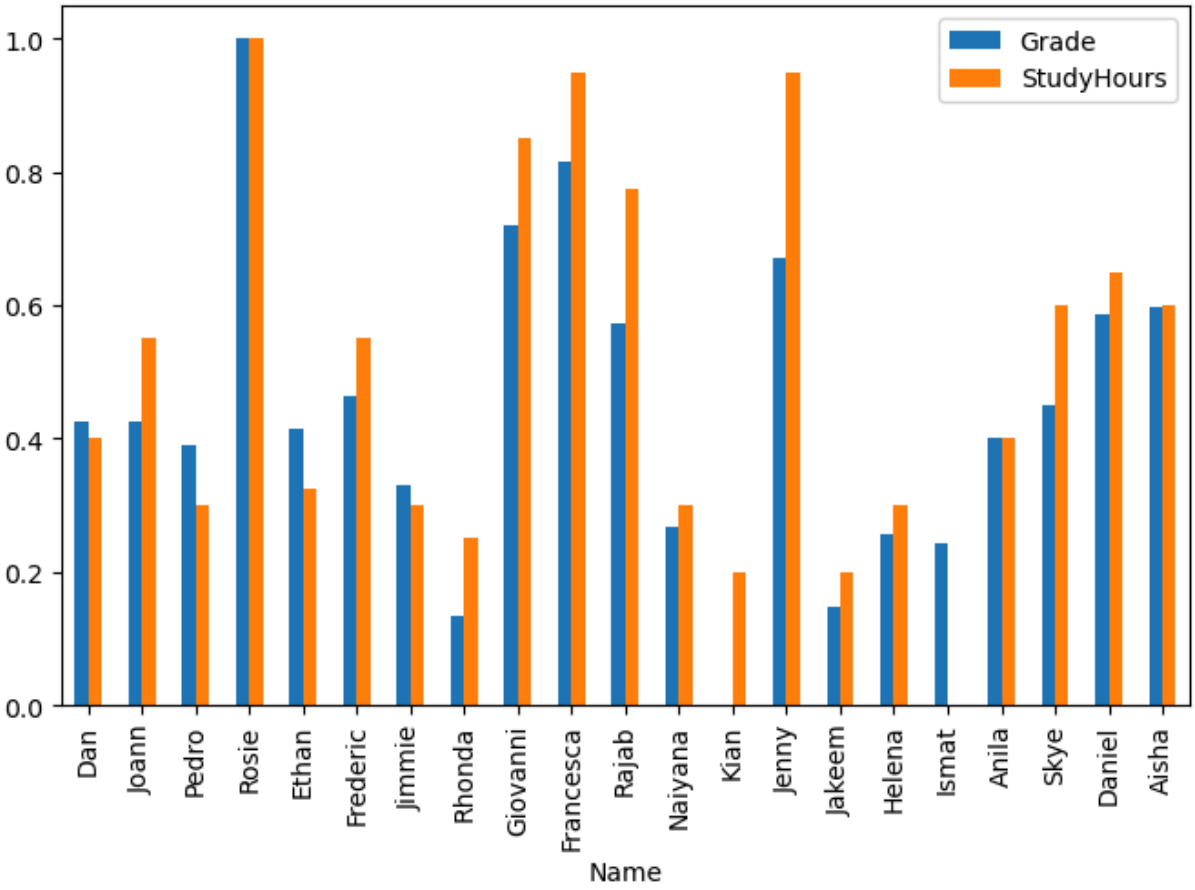
In [67]:
```
# comparing numeric variables
df_sample.plot(x='Name', y=['Grade', 'StudyHours'], kind='bar', figsize=(8,5));
```



The chart shows bars for both grade and study hours for each student, but it's not easy to compare because the values are on different scales. A grade is measured in grade points (and ranges from 3 to 97), and study time is measured in hours (and ranges from 1 to 16).

A common technique when dealing with numeric data in different scales is to normalize the data so that the values retain their proportional distribution but are measured on the same scale. To accomplish this, we'll use a technique called MinMax scaling that distributes the values proportionally on a scale of 0 to 1. You could write the code to apply this transformation, but the Scikit-Learn library provides a scaler to do it for you.

In [70]:
```
from sklearn.preprocessing import MinMaxScaler

# Get a scaler object
scaler = MinMaxScaler()

# Create a new dataframe for the scaled values
df_normalized = df_sample[['Name', 'Grade', 'StudyHours']].copy()

# Normalize the numeric columns
df_normalized[['Grade','StudyHours']] = scaler.fit_transform(df_normalized[['Grade'

# Plot the normalized values
df_normalized.plot(x='Name', y=['Grade','StudyHours'], kind='bar', figsize=(8,5));
```



Correlation between Grades and Study Hourse

In [73]:
```
df_students.Grade.corr(df_students.StudyHours)
```

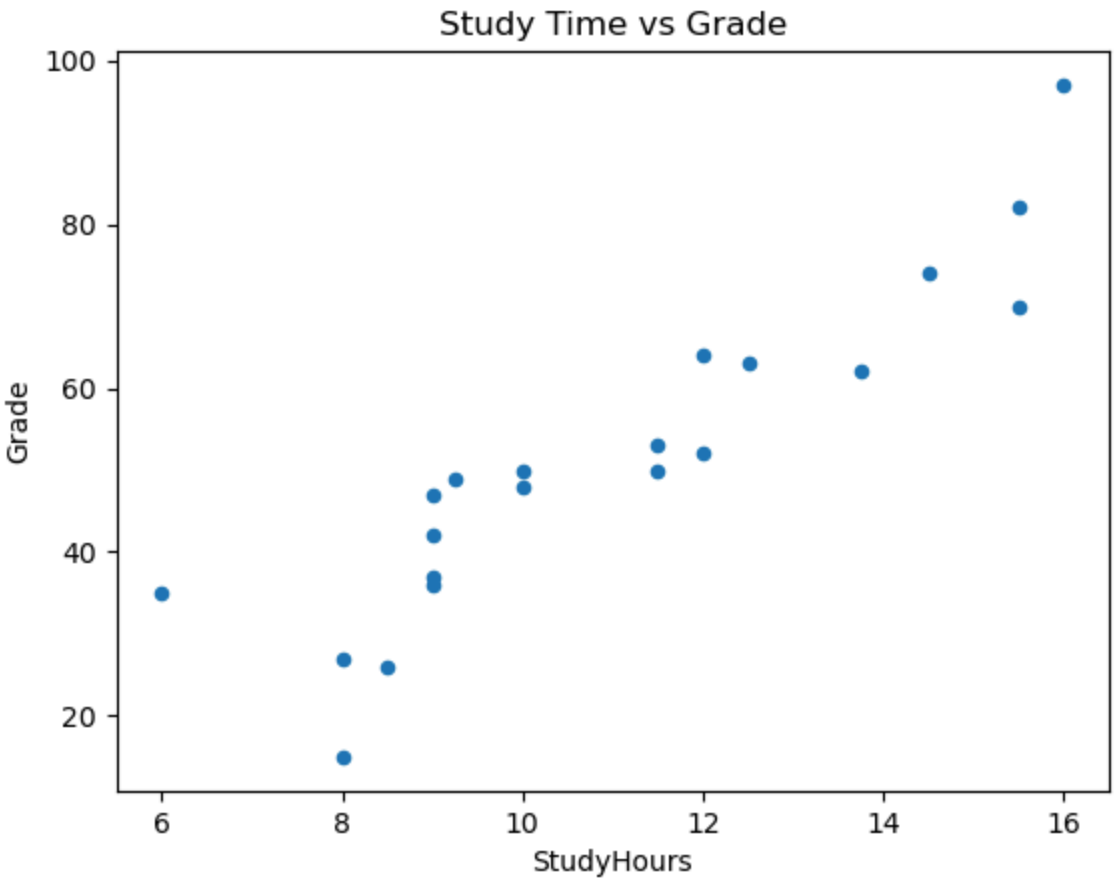
Out[73]:
```
0.9254280509006747
```

In [75]:
```
df_normalized.Grade.corr(df_normalized.StudyHours)
```

Out[75]:
```
0.9117666413789677
```

In [77]:
```
# Create a scatter plot
df_sample.plot(kind='scatter', title='Study Time vs Grade', x='StudyHours', y='Grad
```

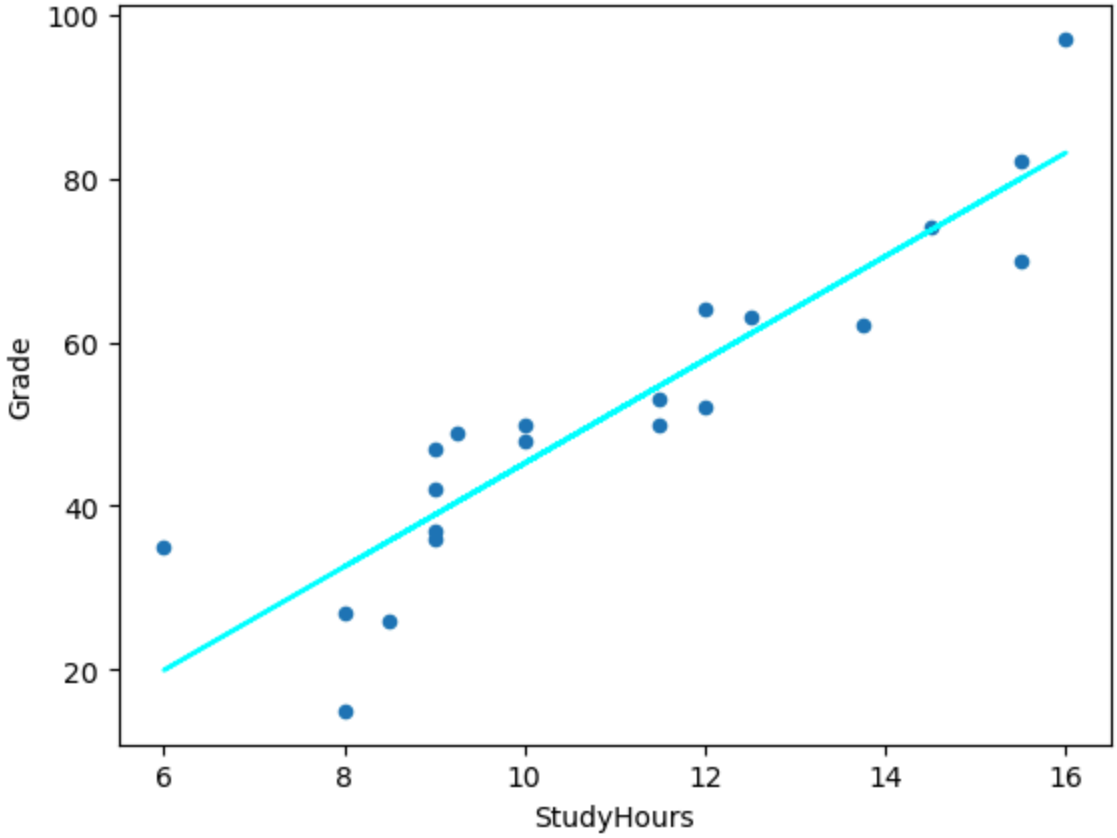
Out[77]:
```
<Axes: title={'center': 'Study Time vs Grade'}, xlabel='StudyHours', ylabel='Grad
e'>
```



Drawing a regression line using scipy stats:

In [80]:
```
from scipy import stats

#
df_regression = df_sample[['Grade', 'StudyHours']].copy()

# Get the regression slope and intercept
m, b, r, p, se = stats.linregress(df_regression['StudyHours'], df_regression['Grade
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['StudyHours']) + b

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

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

# Plot the regression line
plt.plot(df_regression['StudyHours'],df_regression['fx'], color='cyan')

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

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



Making Prediction

In [84]:
```
# 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(stud
```

```
Studying for 14 hours per week may result in a grade of 70
```

In [ ]: