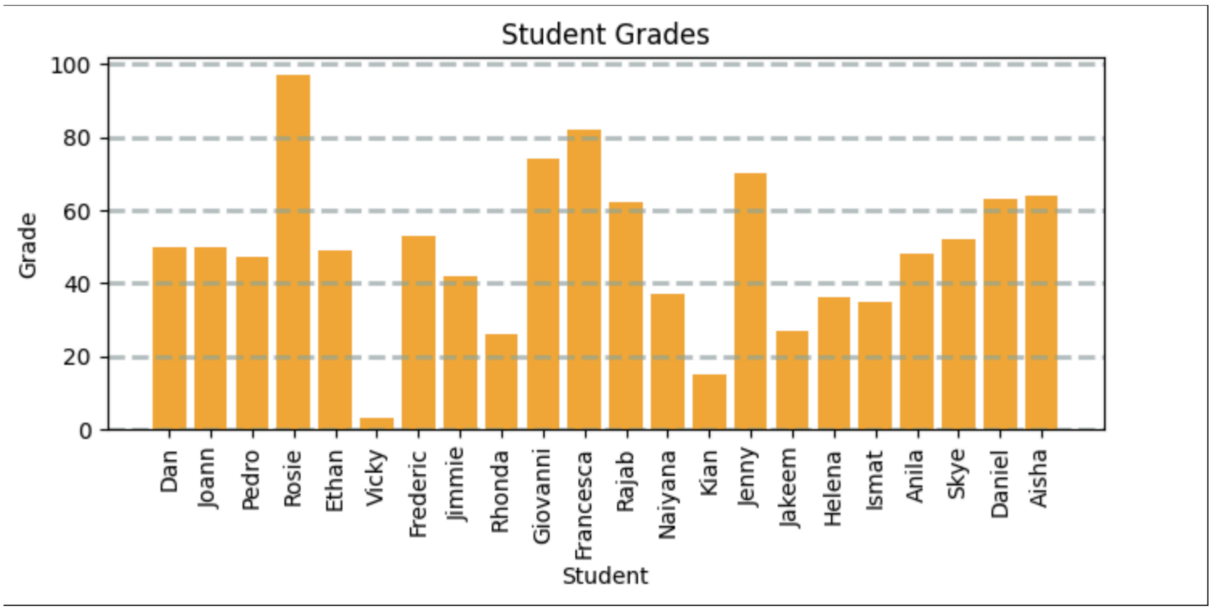
Matplotlib

(AI notes)

// Bar chart that shows the grade of each student:

```
from matplotlib import pyplot as plt

# Create a Figure
fig = plt.figure(figsize=(8,3))  // Creates the size of the figure

# Create a bar plot of name vs grade
plt.bar(x=df_students.Name, height=df_students.Grade, color='orange')
# Also specified the color of the bar chart

# Customize the chart
plt.title('Student Grades') // Adds a title to the chart
plt.xlabel('Student')       // Adds label Student to the x axis
plt.ylabel('Grade')         // Adds label Grade to the y axis
plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y',
alpha=0.7)   // Adds a grid to easily determine the values for the bars
// The grid is the horizontal good-looking lines (could be missing)
plt.xticks(rotation=90)     // Rotates the student names vertically to
be able to read them (not to be intercalated)

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

Result:

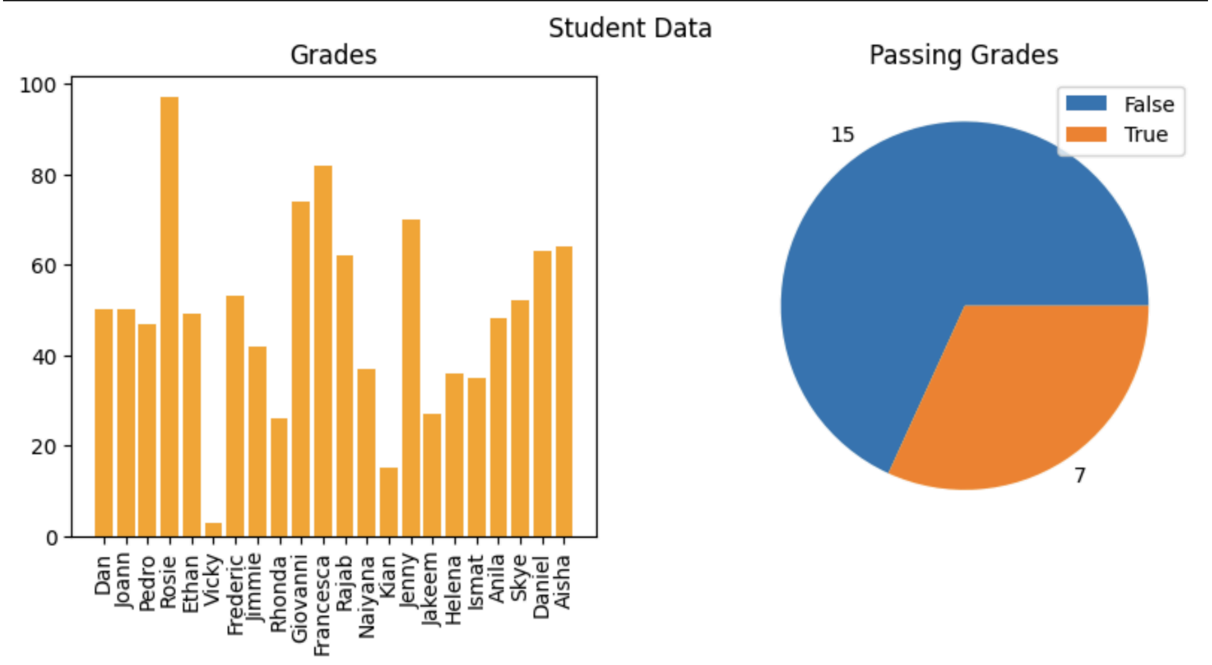


===

// A figure can contain multiple subplots. In the following example, there are two subplots, a bar chart and a pie chart:

```
# Create a figure for 2 subplots (1 row, 2 columns)
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()
```

Result:



==

// Instead of Matplotlib.pyplot, DataFrame itself has methods to plot its data:

```
df_students.plot.bar(x='Name', y='StudyHours', color='teal',
figsize=(6,4))
```

// Here is how you can create a histogram for a set of data (in our case, student grades):

```
# Get the variable to examine
var_data = df_students['Grade']

# Create a Figure
fig = plt.figure(figsize=(10,4))

# Plot a histogram
plt.hist(var_data)

# Add titles and labels
plt.title('Data Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

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

Good to know definitions:

- The mean: A simple average based on adding together all of the values in the sample set and then dividing the total by the number of samples.
- The median: The value in the middle of the range of all of the sample values.
- The mode: The most commonly occurring value in the sample set*.

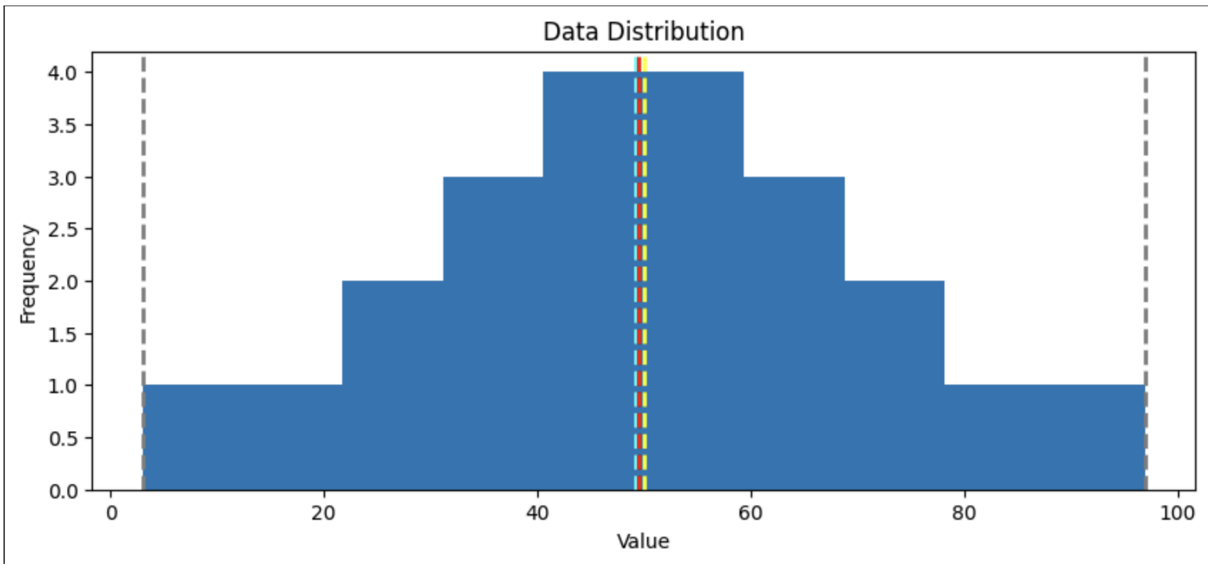
```
# Get the variable to examine
var = df_students['Grade']

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

// We display all of the above variables in a histogram:

```
# Create a Figure
fig = plt.figure(figsize=(10,4))

# Plot a histogram
plt.hist(var)

# Add lines for the statistics
plt.axvline(x=min_val, color = 'gray', linestyle='dashed', linewidth =
2)
```

```
plt.axvline(x=mean_val, color = 'cyan', linestyle='dashed', linewidth =
2)
plt.axvline(x=med_val, color = 'red', linestyle='dashed', linewidth =
2)
plt.axvline(x=mod_val, color = 'yellow', linestyle='dashed', linewidth
= 2)
plt.axvline(x=max_val, color = 'gray', linestyle='dashed', linewidth =
2)

# Add titles and labels
plt.title('Data Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

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

Result:



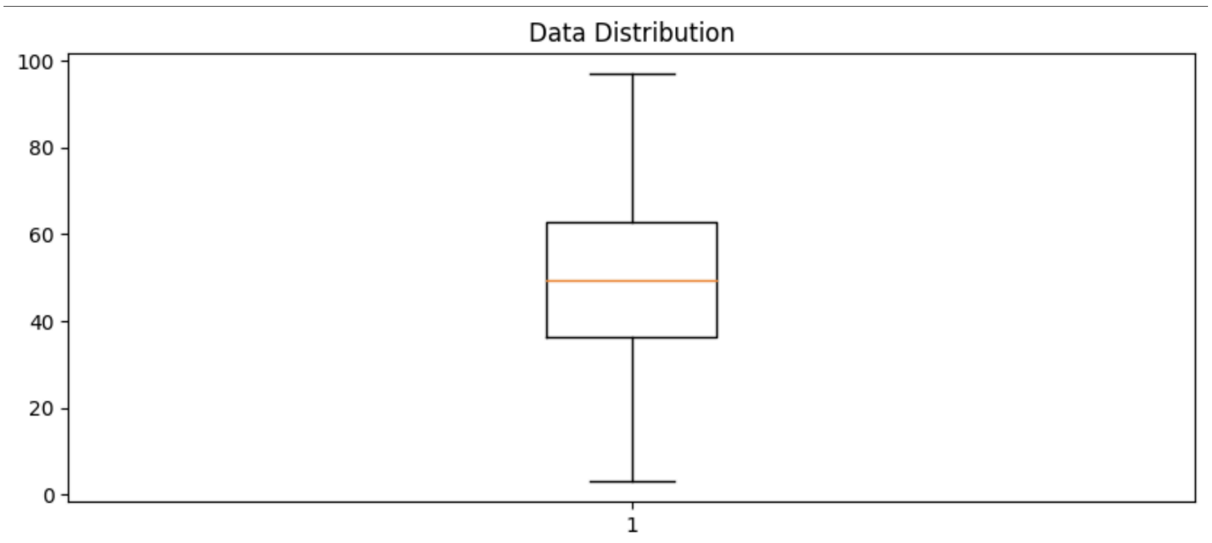
==

// Created a box plot for the student grades:

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

# Create a Figure
fig = plt.figure(figsize=(10,4))

# Plot a histogram
plt.boxplot(var)

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

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

Result:



==

Measures of variance – Good to know definitions

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

```
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(col_name, rng, var, std))
```

MinMax scaling technique of normalization. Scikit-Learn has implemented such a function that does this for you (normalizes all the values to be between 0 and 1):

```
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','StudyHours']])

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

The statistical correlation - is a value between -1 and 1 that indicates the strength of a relationship (for example, StudyHours and Grades):

```
df_normalized.Grade.corr(df_normalized.StudyHours)
```

To display the apparent correlation:

```
# Create a scatter plot
df_sample.plot.scatter(title='Study Time vs Grade', x='StudyHours',
y='Grade')
```

The **SciPy** package includes a **stats** class that provides a **linregress** method for calculating the **regression**. This returns (among other things) the coefficients you need for the slope equation: slope (m) and intercept (b) based on a given pair of variable samples you want to compare:

```
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()
```

Using the regression coefficients for prediction (to estimate the expected grade for a given amount of study):

```
# 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))
```