Line Graphs and Time Series: Takeaways

Syntax

- Importing the pyplot submodule:

```
import matplotlib.pyplot as plt
```

- Plotting a line graph:

```
plt.plot(x_coordinates, y_coordinates)
plt.show()
```

- Changing the scientific notation to plain notation:

```
plt.ticklabel_format(axis='both', style='plain')
```

- Adding a title and axes labels:

```
plt.title('New Reported Cases By Month (Globally)')
plt.xlabel('Month Number')
plt.ylabel('Number Of Cases')
```

- Plotting three graphs that share the same axes:

```
plt.plot(x_coordinates_1, y_coordinates_1)
plt.plot(x_coordinates_2, y_coordinates_2)
plt.plot(x_coordinates_3, y_coordinates_3)
plt.show()
```

- Adding a legend:

```
plt.plot(x_coordinates_1, y_coordinates_1, label='label')
plt.plot(x_coordinates_2, y_coordinates_2, label='label')
plt.legend()
plt.show()
```

- Plotting two graphs separately:

```
plt.plot(x_coordinates_1, y_coordinates_1)
plt.show()

plt.plot(x_coordinates_2, y_coordinates_2)
plt.show()
```

Concepts

- There are two kinds of data visualization:
    - Exploratory data visualization: We build graphs for *ourselves* to explore data and find patterns.
    - Explanatory data visualization: We build graphs for *others* to communicate and explain the patterns we've found through exploring data.

- The horizontal and vertical lines that intersect to make a graph are called **axes**. The horizontal line at the bottom is the **x-axis**, and the vertical line on the left is the **y-axis**. The point where the two lines intersect is called the **origin**.
- The two numbers that represent the distances of a point from the x- and y-axis are called **coordinates**. Point *A* above has two coordinates: seven and two. Seven is the **x-coordinate**, and two is the **y-coordinate**.
- A series of data points that is listed in time order is called a **time series**. To visualize a time series, we can use a **line graph**.
- We learned three types of growth associated with time series:
    - Linear: the growth is constant
    - Exponential: the growth starts slow, but then it becomes faster and faster
    - Logarithmic: the growth starts very fast, but then it becomes slower and slower
- These three types of change can also decrease:
    - Linear: the decrease is constant
    - Exponential: the decrease is slow in the beginning, but then it becomes faster and faster
    - Logarithmic: the decrease is very fast in the beginning, then it starts to slow down
- In practice, most of the line graphs we plot don't show any clear pattern. We need to pay close attention to what we see, and try to extract meaning from the graphs without forcing the data into common patterns we already know.

Resources

- Anatomy of a graph
- A pyplot tutorial from matplotlib
- A short article on line graphs by The Data Visualization Catalogue
- A fun and useful tutorial on graphing equations

Scatter Plots and Correlations: Takeaways

Syntax

- Transforming a Series to datetime:

```
dataframe['column_name'] = pd.to_datetime(dataframe['column_name'])
```

- Rotating x- and y-tick labels:

```
plt.xticks(rotation=45)
plt.yticks(rotation=30)
```

- Plotting a scatter plot:

```
plt.scatter(x_coordinates, y_coordinates)
plt.show()
```

- Measuring the Pearson's r between two Series:

```
dataframe['col_1'].corr(dataframe['col_2'])
```

- Measuring the Pearson's r between all columns of a DataFrame:

```
dataframe.corr()
```

- Measuring the Pearson's r for a subset of columns of a DataFrame:

```
dataframe.corr()[['col_1', 'col_2', 'col_3']]
```

Concepts

- The little lines on each axis to show unit lengths are called **ticks**, and the corresponding labels are **tick labels**. The x-axis has x-ticks, and the y-axis has y-ticks.
- In time series data, we sometimes see specific patterns occurring regularly at specific intervals of time — this is called **seasonality**.
- Weather, holidays, school vacations and other factors can often cause seasonality. Identifying seasonality can be useful for businesses.
- In a broad sense, when two columns are related in a specific way and to a certain degree, we call this relationship **correlation**. We can best visualize correlation using a **scatter plot**.
- Two positively correlated columns tend to change in the same direction. On a scatter plot, a positive correlations shows an upward trend.
- Two negatively correlated columns tend to change in opposite directions. On a scatter plot, a negative correlation shows a downward trend.
- Not all pairs of columns are correlated.
- We can measure correlation strength using **Pearson's r**. Pearson's r measures how well the points fit on a straight line.

- Pearson's r values lie between -1.00 and +1.00. When the positive correlation is perfect, the Pearson's r is equal to +1.00. When the negative correlation is perfect, the Pearson's r is equal to -1.00. A value of 0.0 shows no correlation.
- The sign of the Pearson's r only gives us the direction of the correlation, not its strength.
- When we're working with categorical variables that have been encoded with numbers, we need to interpret the sign of the correlation with caution.
- Correlation does not imply causation: proving causality requires more than just correlation, and we can't say that X is the cause of Y simply because columns X and Y are strongly correlated.

Resources

- Anatomy of a graph
- A short article on scatter plots by The Data Visualization Catalogue
- A nice article on scatter plots by MathIsFun
- A nice article on correlation by MathIsFun

Bar Plots, Histograms, and Distributions: Takeaways

Syntax

- Plotting a vertical bar plot:

```
plt.bar(x=x_coordinates, height=heights)
plt.show()
```

- Plotting a horizontal bar plot:

```
plt.barh(y=y_coordinates, width=widths)
plt.show()
```

- Changing the x-tick labels of a graph (any graph):

```
plt.xticks(ticks=x_coordinates, labels=the_labels_you_want)
```

- Changing the y-tick labels of a graph (any graph):

```
plt.yticks(ticks=y_coordinates, labels=the_labels_you_want)
```

- Plotting a histogram:

```
plt.hist(column)
plt.show()
```

- Generating a frequency table:

```
Series.value_counts(bins=10)
```

- Generating a sorted grouped frequency table:

```
Series.value_counts(bins=10).sort_index()
```

Concepts

- We call the number of times that a unique value occurs the **frequency**. And we call the output of **Series.value_counts()** a **frequency table**.
- To create a readable frequency table for a numerical column, we group the values in intervals — in this case, we call the table **grouped frequency table**.
- **Bar plots** work well for visualizing frequency tables when the frequency tables are generated for categorical columns.
- **Histograms** work well for visualizing frequency tables when the frequency tables are generated for numerical columns.
- A histogram is a modified bar plot — the main visual difference is that there are no gaps between bars. Another equally important difference is that each bar represents an interval, not a single value.
- A histogram shows the distribution of the values, and if its shape is symmetrical, then we say we have a **symmetrical distribution**.

- If we draw a vertical line exactly in the middle of a symmetrical histogram, then we divide the histogram in two halves that are mirror images of one another.
- A common symmetrical distribution is the **normal distribution** — most values pile up in the middle of the range, and value frequencies decrease gradually toward the extremities of the range.
- Another common symmetrical distribution is the **uniform distribution**. The values are uniformly distributed — the unique values have equal frequencies.
- When we plot histograms in practice, we rarely see perfectly symmetrical distributions. However, these ideal cases we learned about serve as a baseline to help us describe and interpret the distributions we see in practice.
- **Skewed distributions** are not symmetrical, and they have the following characteristics:
    - The values pile up toward the end or the starting point of the range, making up the body of the distribution.
    - Then the values decrease in frequency toward the opposite end, forming the tail of the distribution.
- If the tail of a skewed distribution points to the right, then the distribution is right skewed (or positively skewed).
- If the tail of a skewed distribution points to the left, then the distribution is left skewed (or negatively skewed).

Resources

- A short article on bar plots by The Data Visualization Catalogue
- A short article on histograms by The Data Visualization Catalogue
- A few fun and useful tutorials from MathIsFun:
    - Bar Plots
    - Histograms
    - Frequency Distribution
    - Grouped Frequency Distribution
    - Normal Distribution

Pandas Visualizations and Grid Charts: Takeaways

Syntax

- To generate a histogram using Pandas:

```
Series.plot.hist()
plt.show()
```

- To generate a vertical bar plot using Pandas:

```
Series.plot.bar()
plt.show()
```

- To generate a horizontal bar plot using Pandas:

```
Series.plt.barh() # Horizontal bar plot
plt.show()
```

- To generate a line plot using Pandas(the index of the Series is the x-axis):

```
Series.plot.line()
plt.show()
```

- To generate a line plot using Pandas with a DataFrame:

```
DataFrame.plot.line(x='col_1', y='col_2')
plt.show()
```

- To generate a scatter plot using Pandas:

```
DataFrame.plot.scatter(x='col_1', y='col_2')
plt.show()
```

- To generate a grid chart with two columns and one row:

```
plt.figure(figsize=(8,3))
plt.subplot(1, 2, 1)
plt.subplot(1, 2, 2)
plt.show()
```

- To generate six line plots on a grid chart (two columns by three rows):

```
plt.figure(figsize=(10,12))
for i in range(1, 7):
    plt.subplot(3, 2, i)
    plt.plot(x_coordinates, y_coordinates)
plt.show()
```

Concepts

- We can generate graphs more quickly using Pandas visualization methods.
- Behind the curtains, Pandas uses Matplotlib to generate the graphs. This allows us to use Matplotlib code to customize the visualizations generated.

- A grid chart (also known as small multiples) is a collection of similar graphs that usually share the same x- and y-axis range.
- The main purpose of a grid chart is to ease comparison.

Resources

- A thorough introduction to Pandas visualization methods
- A Wikipedia article on grid charts

Relational Plots and Multiple Variables: Takeaways

Syntax

- Importing seaborn and activating the default style settings:

```
import seaborn as sns
sns.set_theme()
```

- Plotting a relational plot with five variables:

```
sns.relplot(data=data, x='Col_1', y='Col2',
            hue='Col_3', palette=palette,
            size='Col_4', sizes=sizes,
            style='Col_5', markers=markers)
plt.show()
```

- Separating a relational plot based on a sixth variable:

```
sns.relplot(data=data, x='Col_1', y='Col2',
            hue='Col_3', palette=palette,
            size='Col_4', sizes=sizes,
            style='Col_5', markers=markers,
            col='Col_6')
plt.show()
```

Concepts

- Seaborn is a Python library based on Matplotlib.
- We can visually represent a variable in several ways:
    - X- and y-coordinates
    - Color
    - Size
    - Shape
    - Spatial separation
- A relational plot is a graph that visually represents the relationships between three or more variables.
- Relational plots can also be formed using line plots.

Resources

- Seaborn relational plots
- Seaborn distribution plots
- Seaborn categorical plots
- Seaborn grid charts