In this module, you will learn about data visualization and some of the best practices to keep in mind when creating plots and visuals. You will also learn about the history and the architecture of Matplotlib and learn about basic plotting with Matplotlib. In addition, you will learn about the dataset on immigration to Canada, which will be used extensively throughout the course. Finally, you will briefly learn how to read csv files into a pandas dataframe and process and manipulate the data in the dataframe, and how to generate line plots using Matplotlib.

**Learning Objectives**

* Describe the importance of data visualization
* Relate the history of Matplotlib and its architecture
* Apply Matplotlib to create plots using Jupyter notebooks
* Read csv files into a Pandas DataFrame; process and manipulate the data in the DataFrame; andgenerate line plots using Matplotlib

## Week 1 - Introduction to Data Visualization Tools

* Introduction to Data Visualization
* Introduction to Matplotlib
* Basic Plotting with Matplotlib
* Dataset on Immigration to Canada
* Line Plots
* Lab: Introduction to Matplotlib and Line Plots
* Quiz: Introduction to Data Visualization Tools

## Week 2 - Basic and Specialized Visualization Tools

* Area Plots
* Histograms
* Bar Charts
* Pie Charts
* Box Plots
* Scatter Plots
* Bubble Plots
* Lab: Basic Visualization Tools
* Lab: Specialized Visualization Tools
* Quiz: Basic Visualization Tools
* Quiz: Specialized Visualization Tools

## Week 3 - Advanced Visualizations and Geospatial Data

* Waffle Charts
* Word Clouds
* Seaborn and Regression Plots
* Introduction to Folium and Map Styles
* Maps with Markers
* Choropleth Maps
* Lab: Advanced Visualization Tools
* Lab: Creating Maps and Visualizing Geospatial Data
* Quiz: Advanced Visualization Tools
* Quiz: Visualizing Geospatial Data
* Peer-review Assignment

# Welcome

0:00

(Music)

Hello everyone and welcome to data visualization with Python. I'm Alex

Aklson, a data scientist at IBM, and I'm your instructor for this course.

Throughout this course we're gonna learn how to create meaningful, effective, and

aesthetically pleasing data visuals and plots in python using Matplotlib and a

couple of other libraries namely Seaborn and Folium. This course will consist of

three modules. In module 1, we will briefly discuss data visualization and some of

the best practices to keep in mind when creating data visuals. We will then learn

about Matplotlib: its history, architecture, and the three layers that

form its architecture. We will also learn about the data set that we will use

throughout the course in these lectures as well as the hands-on sessions. We will

essentially be working with a data set that was curated by the United Nations

on immigration from different countries to Canada from 1980 to 2013. Then we

will start learning how to use Matplotlib to create plots and visuals, and we will start off with line plots.

Now, we will generate the majority of our plots and visualizations in this course

using data stored in pandas dataframes. For those of you who don’t know what pandas is,

pandas is a python library for data manipulation and analysis.

So before we start building visualizations and plots,

we will take a brief crash course on pandas and learn

how to use it to read data from csv files like the one shown here

into what is called a pandas dataframe like the one shown here.

Now, if you are interested in learning more about the pandas library,

we actually cover it in much more detail in our next course

in this specialization which is Data Analysis with Python,

so make sure to complete the next course in this specialization.

In module 2, we will continue on with a few more

basic data visualizations such as area plots, histograms, and bar charts,

and learn how to use Matplotlib to create them and even create different versions of these plots.

We will also cover a set of specialized visualizations

such as pie charts, box plots, scatter plots, and bubble plots,

and we will learn how to create them still using Matplotlib.

In module 3, we will learn about more advanced visuals such as waffle charts

that provide a fine-grained view of the proportions of different categories in a dataset.

We will also learn about word clouds

that depict word frequency or importance in a body of text.

Also, in this module, we will explore another library, seaborn,

which is built on top of Matplotlib to simplify the process of creating plots and visuals,

and we will get a taste of its effectiveness through the creation of regression plots.

Finally, in this module, we will explore another library, folium,

which was built primarily to visualize geospatial data.

So, we will learn how create maps of different regions of the world,

superimpose markers of different shapes on top of maps, and learn how to create choropleth maps.

Before I conclude this video, let me stress one thing.

Data visualization is best learned through hands-on exercises and sessions.

Therefore, don’t worry if you find some of the videos to be short.

The labs and the hands-on sessions are very thorough and cover a lot of the concepts that are discussed

in the videos in much more detail, so it is very important that you complete the labs and the hands-on sessions,

although they are ungraded components of the course.

I hope that you remember this and keep it in mind as you progress in this course.

After completing this course, you’ll be able to

use different visualization libraries in Python namely,

Matplotlib, seaborn, and folium

to create expressive visual representations of your data for different purposes.

So, let’s get right into it.

# Introduction to Data Visualization

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-data-visualization/discussions/weeks/1)

0:00

hello everyone and welcome to the first

module of the data visualization with

Python course in this video we're gonna

introduce data visualization and go over

an example of transforming a given

visual into one which is more effective

attractive and impactive so let's get

started now one might ask why would I

need to learn how to visualize data well

data visualization is a way to show a

complex data in a form that is graphical

and easy to understand this can be

especially useful when one is trying to

explore the data and getting acquainted

with it also since a picture is worth a

thousand words then plots and graphs can

be very effective in conveying a clear

description of the data especially when

disclosing findings to an audience or

sharing the data with other pure data

scientists also they can be very

valuable when it comes to supporting any

recommendations you make to clients

managers or other decision-makers in

your field darkhorse analytics is a

company that spun out of a research lab

at the University of Alberta in 2008 and

has done fascinating work on data

visualization darkhorse analytics

specializes in quantitative consulting

in several areas including data

visualization and geo spatial analysis

their approach when creating a visual

revolves around three key points less is

more effective it is more attractive and

it is more impactive in other words any

feature or design you incorporate in

your plot to make it more attractive or

pleasing should support the message that

the plot is meant to get across and not

distract from it let's take a look at an

example so here is a pie chart of what

looks like people's preferences when it

comes to different types of pig meat the

charts message is almost half of the

people surveyed preferred bacon over the

other types of pig meat but I'm sure

that almost all of you agree that there

is a lot going on in this pie chart and

we're not even sure it features such as

the blue background or

3d orientation are meant to convey

anything in fact these additional

unnecessary features distract from the

main message and can be confusing to the

audience so let's apply darkhorse

analytics approach to transform this

into a visual that's more effective

attractive and impactive as I mentioned

earlier the message here is that people

are most likely to choose bacon over

other types of pig meat so let's get rid

of everything that can be distracting

from this core message the first thing

is let's get rid of the blue background

and the gray background let's also get

rid of borders as they do not convey any

extra information also let's drop the

redundant legend since the pie chart is

already color coded 3d isn't adding any

extra information so let's say bye to it

text bolding is also unnecessary and

let's get rid of the different colors

and the wedges whoa what just happened

well let's stick in the lines to make

them more meaningful now this looks a

little familiar yes this is a bar graph

after all one with horizontal bars and

finally let's emphasize bacon so that it

stands out among the other types of pig

meat now let's just oppose the pie chart

and the bar graph and compare which is

better and easy to understand I hope

that we anonymously agree that the bar

graph is the better of the two it is

simple cleaner less distracting and much

easier to read in fact pie charts have

recently come under fire from data

visualization experts who argue that

they are relevant only in the rarest of

circumstances bar graphs and charts on

the other hand are argued to be far

superior ways to quickly get a message

across but don't worry about this for

now we will come back to this point when

we learn how to create pie charts and

bar graphs with matplotlib

for more similar and interesting

examples check out darkhorse analytics

website they have a couple more examples

on how to clean bar graphs and maps of

geospatial data all these examples

reinforce the concept of less is more

effective attractive and impactive

# Introduction to Matplotlib

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-data-visualization/discussions/weeks/1)

0:01

In this video, we will start learning about Matplotlib. This video will focus

on the history of Matplotlib and its architecture. Matplotlib is one of the

most widely used, if not the most popular data visualization library in Python. It

was created by John Hunter, who was a neurobiologist and was part of a

research team that was working on analyzing Electrocorticography

signals, ECoG for short. The team was using a proprietary software for the

analysis. However they had only one license and were taking turns in using

it. So in order to overcome this limitation, John set out to replace the

proprietary software with a MATLAB based version that could be utilized by him

and his teammates, and that could be extended by multiple investigators. As a

result, Matplotlib was originally developed as an ECoG visualization tool,

and just like MATLAB, Matplotlib was equipped with a scripting interface for

quick and easy generation of graphics, represented by pyplot. We will learn

more about this in a moment. Now Matplotlib's architecture is composed of

three main layers: the back-end layer, the artist layer where much of the heavy

lifting happens and is usually the appropriate programming paradigm when

writing a web application server, or a UI application, or perhaps a script to be shared with other developers, and the scripting layer, which is the appropriate

layer for everyday purposes and is considered a lighter scripting interface

to simplify common tasks and for a quick and easy generation of graphics and

plots. Now let's go into each layer in a little more details.

So the back-end layer has three built-in abstract interface classes: FigureCanvas,

which defines and encompasses the area on which the figure is drawn. Renderer, an

instance of the renderer class knows how to draw on the figure canvas. And finally,

event, which handles user inputs such as keyboard strokes and

mouse clicks. Moving on to the artist layer. It is composed of one main object,

which is the artist. The artist is the object that knows how to take the

Renderer and use it to put ink on the canvas. Everything you see on a Matplotlib

figure is an artist instance. The title, the lines, the tick labels, the

images, and so on, all correspond to an individual artist. There are two types of

Artist objects. The first type is the primitive type, such as a line, a

rectangle, a circle, or text. And the second type is the composite type, such

as the figure or the axes. The top-level Matplotlib object that contains and

manages all of the elements in a given graphic is the figure artist, and the

most important composite artist is the axes because it is where most of the

Matplotlib API plotting methods are defined, including methods to create and

manipulate the ticks, the axis lines, the grid or the plot background. Now it

is important to note that each composite artist may contain other composite

artists as well as primitive artists. So a figure artist for example would

contain an axis artist as well as a rectangle or text artists. Now let's put

the artist layer to use and see how we can use it to generate a graphic. So

let's try to generate a histogram of 10,000 random numbers using the artist

layer. First we import the figure canvas from the backend backend underscore agg

and attach the figure artist to it. Note that agg stands for anti grain geometry

which is a high-performance library that produces attractive images. Then we

import the Numpy library to generate the random numbers. Next we create an axes

artist. The axes artist is added automatically to the figure axes

container, Fig.axes. And note here that (111)

is from the MATLAB convention so it creates a grid with one row and

one column and uses the first cell in that grid for the location of the new

axes. Then we call the axes method hist, to generate the histogram.

Hist creates a sequence of rectangle artists for each histogram bar and adds

them to the axes container. Here 100 means create 100 bins. Finally, we

decorate the figure with a title and we save it. Now this is the generated

histogram and so this is how we use the artist layer to generate a graphic. As

for the scripting layer, it was developed for scientists who are not professional

programmers, and I'm sure you agree with me based on the histogram that we just

created that the artist layer is syntactically heavy as it is meant for

developers and not for individuals whose goal is to perform quick exploratory

analysis of some data. Matplotlib's scripting layer is essentially the

Matplotlib.pyplot interface, which automates the process of defining a

canvas and defining a figure artist instance and connecting them. So

let's see how the same code that we used earlier using the artist layer to

generate a histogram of 10,000 random numbers would now look like. So first we

import the pyplot interface and you can see how all the methods associated

with creating the histogram and other artist objects and manipulating them

whether it is the hist method or showing the figure are part of the pyplot

interface. If you're interested in learning more about the history of

Matplotlib and its architecture, this link will take you to a chapter written

by the creators of Matplotlib themselves. It is definitely a recommended read.

# Basic Plotting with Matplotlib

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-data-visualization/discussions/weeks/1)

0:00

In this video, we will learn how to use Matplotlib to create plots, and we will

do so using the Jupyter notebook as our environment.

Now Matplotlib is a well-established data visualization library that is well

supported in different environments such as in Python scripts, in the iPython

shell, web application servers, in graphical user interface toolkits as

well as the Jupyter notebook. Now for those of you who don't know what the

Jupyter notebook is, it's an open source web application that allows you to

create and share documents that contain live code visualizations and some

explanatory text as well. Jupyter has some specialized support for Matplotlib

and so if you start a Jupyter notebook, all you have to do is import Matplotlib

and you're ready to go. In this course, we will be working mostly with the

scripting interface. In other words, we will learn how to create almost all of

the visualization tools using the scripting interface. As we proceed in the

course, you will appreciate the power of this interface when you find out that

you can literally create almost all of the conventional visualization tools

such as histograms, bar charts, box plots and many others using one function only:

the plot function. Let's start with an example. Let's first import the scripting

interface as plt, and let's plot a circular mark at the position (5, 5), so x

equals 5 and y equals 5. Notice how the plot was generated within the browser

and not in a separate window for example. If the plot gets generated in a new

window then you can enforce generating plots within the browser using what's

called a magic function. A magic function starts with % Matplotlib, and to enforce

plots to be rendered within the browser, you pass in inline as the backend.

Matplotlib has a number of different backends available. One limitation of

this backend is that you cannot modify a figure once

it's rendered. So after rendering the above figure, there is no way for us to

add, for example, a figure title or label its axes. You will need to generate a

new plot and add a title and the axes labels before calling the show

function. A backend that overcomes this limitation is the notebook backend.

With the notebook backend in place, if a plt function is called, it checks if an

active figure exists, and any functions you call will be applied to this active

figure. If a figure does not exist, it renders a new figure. So when we call the

plt.plot function to plot a circular mark at position (5, 5), the backend checks

if an active figure exists. Since there isn't an active figure, it generates a

figure and adds a circular mark to position (5, 5). And what is beautiful about

this back end is that now we can easily add a title for example or labels to the

axes after the plot was rendered, without the need to regenerate the figure.

Finally, another thing that is great about Matplotlib is that pandas also has

a built-in implementation of it. Therefore, plotting in pandas is as simple as

calling the plot function on a given pandas series or dataframe. So, say we

have a data frame of the number of immigrants from India and China to

Canada from 1980 to 1996 and say we're interested in generating a line plot of

these data. All we have to do is call the plot function on this dataframe which

we called India\_China\_df and set the parameter kind

to line and there you have it: a line plot of the data in the data frame.

Plotting a histogram of the data is not any different. So say we would like to

plot a histogram of the India column in our dataframe. All we have to do is call

the plot function on that column and set the parameter kind to hist, for histogram.

And there you have it. A histogram of the number of Indian immigrants

to Canada from 1980 to 1996. This concludes our video on basic plotting

with Matplotlib. See you in the next video.

# Dataset on Immigration to Canada

0:00

In this video, we will learn more about the dataset that we will be using

throughout the course. The population division of the United Nations compiled

immigration data pertaining to 45 countries. The data consist of the total

number of immigrants from all over the world to each of the 45 countries as

well as other metadata pertaining to the immigrants countries of origin. In this

course, we will focus on immigration to Canada and we will work primarily with

the data set involving immigration to the great white north Here is a snapshot

of the UN data on immigration to Canada in the form of an excel file. As you can

see, the first twenty rows contain textual data about the UN Department and other

irrelevant information. Row 21 contains the labels of the columns. Following that

each row represents a country and contains metadata about the country such

as what continent it resides in, what region it belongs to, and whether the

region is developing or developed. Each row also contains the total number of

immigrants from that country for the years 1980 all the way to 2013.

Throughout this course, we will be using pandas for any analysis of the data

before creating any visualizations. So in order to start creating different types

of plots of the data, whether for exploratory analysis or for presentation,

we will need to import the data into a pandas dataframe. To do that, we will

need to import the pandas library as well as the xlrd library, which is

required to extract data from Excel spreadsheet files. Then we call the

pandas function read\_excel to read the data into a pandas dataframe.

And let's name this dataframe df\_can. Notice how we're skipping

the first twenty rows to read only the data corresponding to each country. If you

want to confirm that you have imported your data correctly, in pandas, you can

always use the head function to display the first five rows of the dataframe. So,

if we call this function on our dataframe, df\_can, here is the

output. As you can see, the output of the head function looks correct with the

columns having the correct labels and each row representing a country and

containing the total number of immigrants from that country. This

concludes our video on the immigration to Canada dataset. I will see you in the

next video.

# Line Plots

0:00

In this video, things will start getting more exciting. We will generate our first

visualization tool: the line plot. So what is a line plot? As its name suggests, it

is a plot in the form of a series of data points connected by straight line

segments. It is one of the most basic type of chart and is common in many

fields not just data science. The more important question is when to use line

plots. The best use case for a line plot is when you have a continuous dataset

and you're interested in visualizing the data over a period of time. As an

example, say we're interested in the trend of immigrants from Haiti to Canada.

We can generate a line plot and the resulting figure will depict the trend

of Haitian immigrants to Canada from 1980 to 2013. Based on this line plot, we

can then research for justifications of obvious anomalies or changes. So in this

example, we see that there is a spike of immigration from Haiti to Canada in 2010.

A quick Google search for major events in Haiti in 2010 would return the tragic

earthquake that took place in 2010, and therefore this influx of immigration to

Canada was mainly due to that tragic earthquake. Okay, now, how can we generate

this line plot? Before we go over the code to do that, let's do a quick recap

of our dataset. Each row represents a country and contains metadata about the

country such as where it is located geographically, and whether it is

developing or developed. Each row also contains numerical figures of annual

immigration from that country to Canada from 1980 to 2013.

Now let's process the dataframe so that the country name becomes the index of

each row. This should make querying specific countries easier. Also let's add

an extra column which represents the cumulative sum of annual immigration from

each country from 1980 to 2013. So for Afghanistan, it is 58,639,

total, and for Albania it is 15,699,

and so on. And let's name our dataframe df\_canada. So now

that we know how our data is stored in the dataframe, df\_canada,

let's generate the line plot corresponding to immigration from Haiti.

First, we import Matplotlib as mpl and its scripting interface as plt. Then,

we call the plot function on the row corresponding to Haiti, and we set kind

equals line to generate a line plot. Note that we used years which is a list

containing string format of years from 1980 to 2013 in order to exclude the

column of total immigration that we added. Then to complete the figure, we

give it a title, and we label its axes. Finally, we call the show function to

display the figure. Note that this is the code to generate the line plot using the

magic function % matplotlib with the inline backend. And there you have it: a

line plot that depicts immigration from Haiti to Canada from 1980 to 2013.

In the lab session, we explore line plots in more detail so make sure to

complete this module's lab session. This concludes our video on line plots. I'll

see you in the next video.

# Area Plots

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-data-visualization/discussions/weeks/2)

0:00

In this video, we will learn about another visualization tool: the area plot,

which is actually an extension of the line plot that we learned about in an

earlier video. So what is an area plot? An area plot also known as an area chart or

graph is a type of plot that depicts accumulated totals using numbers or

percentages over time. It is based on the line plot and is commonly used when

trying to compare two or more quantities. So how can we generate an area plot with

Matplotlib? Before we go over the code to do that, let's do a quick recap of our

dataset. Recall that each row represents a country and contains metadata about

the country such as where it is located geographically and whether it is

developing or developed. Each row also contains numerical figures of annual

immigration from that country to Canada from 1980 to 2013.

Now let's process the dataframe so that the country name becomes the index of

each row. This should make retrieving rows pertaining to specific countries a

lot easier. Also, let's add an extra column which represents the cumulative

sum of annual immigration from each country from 1980 to 2013. So for

Afghanistan, it is 58,639, total, and for Albania, it is 15,699 and so

on, and let's name our data frame df\_canada. So now that we know

how our data is stored in the dataframe, df\_canada, let's try to

generate area plots for the countries with the highest number of immigration

to Canada. We can try to find these countries by sorting our dataframe in

descending order of cumulative total immigration from 1980 to 2013. We use the

sort\_values function to sort our dataframe in descending order and

here is the result. So it turns out that India followed by China then the UK, Philippines,

and Pakistan are the top five countries with the highest number of

immigration to Canada. So can we now go ahead and generate the area plots using

the first five rows of this dataframe? Not quite yet. First we need to create a

new dataframe of only these five countries and we need to exclude the

total column. More importantly, to generate the area plots for these

countries, we need the years to be plotted on the horizontal axis and the

annual immigration to be plotted on the vertical axis.

Note that Matplotlib plots the indices of a dataframe on the horizontal axis,

and with the dataframe as shown, the countries will be plotted on the

horizontal axis. So to fix this, we need to take the transpose of the dataframe.

Let's see how we can do this. After we sort our dataframe in descending order

of cumulative annual immigration, we create a new dataframe of the top five

countries and let's call it df\_top5. We then select only

the columns representing the years 1980 to 2013 in order to exclude the total

column before applying the transpose method. The resulting dataframe is

exactly what we want, with five columns where each column represents one of the

top five countries, and the years being the indices. Now we can go ahead and call

the plot function on dataframe df\_top5 to generate the area

plots. To do that, first we import Matplotlib as mpl and its

scripting interface as plt. Then we call the plot function on the dataframe df\_top5

and, we set kind equals area to generate an area plot.

Then to complete the figure we give it a title and we label its axes. Finally we

call the show function to display the figure. Note that here we're generating

the area plot using the inline backend. And there you have it: an area plot that

depicts the immigration trend of the five countries with

the highest immigration to Canada from 1980 to 2013. In the lab session, we

explore area plots in more details, so make sure to complete this module's lab

session. And with this, we conclude our video on area plots. I'll see you in the next video.

# Histograms

0:00

In this video, we will learn about another visualization tool: the histogram,

and we will learn how to create it using Matplotlib. Let's start by defining what

a histogram is. A histogram is a way of representing the frequency distribution

of a numeric dataset. The way it works is it partitions the spread of the

numeric data into bins, assigns each datapoint in the dataset to a bin, and then

counts the number of datapoints that have been assigned to each bin. So the

vertical axis is actually the frequency or the number of datapoints in each bin.

For example, let's say the range of the numeric values in the dataset is 34,129.

Now, the first step in creating the histogram is partitioning

the horizontal axis in, say, ten bins of equal width, and then we construct the

histogram by counting how many datapoints have a value that is between the

limits of the first bin, the second bin, the third bin, and so on. Say the number

of datapoints having a value between 0 and 3,413 is 175. Then we draw a bar of

that height for this bin. We repeat the same thing for all the other bins, and if

no datapoints fall into a bin then that bin would have a bar of height 0. So how

do we create a histogram using Matplotlib? Before we go over the code to

do that, let's do a quick recap of our dataset.

Recall that each row represents a country and contains metadata about the

country such as where it is located geographically and whether it is

developing or developed. Each row also contains numerical figures of annual

immigration from that country to Canada from 1980 to 2013.

Now let's process the dataframe so that the country name becomes the index of

each row. This should make retrieving rows pertaining to specific countries a

lot easier. Also let's add an extra column which

represents the cumulative sum of annual immigration from each country from 1980

to 2013. So for Afghanistan for example, it is 58,639,

total, and for Albania it is 15,699,

and so on. And let's name our dataframe df\_canada.

So now that we know how our data is stored in the dataframe df\_canada,

say we're interested in visualizing the distribution of

immigrants to Canada in the year 2013. The simplest way to do that is to

generate a histogram of the data in column 2013, and let's see how we can do

that with Matplotlib. First, we import Matplotlib as mpl and its scripting

interface as plt. Then we call the plot function on the data in column 2013 and

we set kind equals hist to generate a histogram. Then to complete the figure, we

give it a title and we label its axes. Finally, we call the show function to

display the figure. And there you have it: A histogram that depicts the

distribution of immigration to Canada in 2013, but notice how the bins are not

aligned with the tick marks on the horizontal axis. This can make the

histogram hard to read. So let's try to fix this in order to make our histogram

more effective. One way to solve this issue is to borrow the histogram

function from the Numpy library. So as usual we start by importing

Matplotlib and its scripting interface, but this time we also import the Numpy

library. Then we call the Numpy histogram function on the data in column 2013. What

this function is going do is it is going to partition the spread of the data in

column 2013 into 10 bins of equal width, compute the number of datapoints that

fall in each bin, and then return this frequency of each bin which we're

calling count here and the bin edges which we're calling bin\_edges.

We then pass these bin edges as an additional parameter in our plot

function to generate the histogram. And there you go: A nice looking histogram

whose bin edges are aligned with the tick marks on the horizontal axis. In the

lab session, we explore histograms in more details, so make sure to complete

this module's lab session. And with this, we conclude our video on histograms.

I'll see you in the next video.

# Bar Charts

0:00

In this video, we will learn about an additional visualization tool, namely the

bar chart, and learn how to create it using Matplotlib. A bar chart is a very

popular visualization tool. Unlike a histogram, a bar chart also known as a

bar graph is a type of plot where the length of each bar is proportional to

the value of the item that it represents. It is commonly used to compare the

values of a variable at a given point in time. For example, say we're interested in

visualizing in a discrete fashion how immigration from Iceland to Canada

looked like from 1980 to 2013. One way to do that is by building a bar chart where

the height of the bar represents the total immigration from Iceland to Canada

in a particular year. So how do we do that with Matplotlib. Before we go over

the code to do that, let's do a quick recap of our dataset. Recall that each

row represents a country and contains metadata about the country such as where

it is located geographically and whether it is developing or developed. Each row

also contains numerical figures of annual immigration from that country to

Canada from 1980 to 2013.

Now let's process the dataframe so that the country name becomes the index of

each row. This should make retrieving rows pertaining to specific countries a

lot easier. Also, let's add an extra column which represents the cumulative

sum of annual immigration from each country from 1980 to 2013.

So for Afghanistan for example, it is 58,639, total, and for Albania it is

15,699 and so on. And let's name our dataframe, df\_canada. So now

that we know how our data is stored in the dataframe, df\_canada,

let's see how we can use Matplotlib to generate a bar chart to visualize how

immigration from Iceland to Canada looked like from

1980 to 2013. As usual, we start by importing Matplotlib and its scripting interface.

Then we use the years variable to create a new dataframe; let's name it

df\_iceland, which includes the data pertaining to annual immigration

from Iceland to Canada and excluding the total column. Then we call the plot

function on df\_iceland, and we set kind equals bar to generate a bar

chart. Then to complete the figure we give it a title, and we label its axes.

Finally, we call the show function to display the figure. And there you have it:

A bar chart that depicts the immigration from Iceland to Canada from 1980 to 2013.

By examining the bar chart, we notice that immigration to Canada from Iceland

has seen an increasing trend since 2010. I'm sure that the curious among you are

already wondering who the culprit behind this increasing trend is. In the lab

session, we reveal the reason and we also learn how to create a bar chart

with horizontal bars, so make sure to complete this module's lab session. And

with this, we conclude our video on bar charts. I'll see you in the next video.

# Pie Charts

0:00

In this video, we will learn about another visualization tool: the pie chart,

and we will learn how to create it using Matplotlib.

So what is a pie chart? A pie chart is a circular statistical graphic divided

into slices to illustrate numerical proportion. For example, here is a pie

chart of the Canadian federal election back in 2015 were the Liberals in red

won more than 50% of the seats in the House of Commons. That is why the red

color occupies more than half of the circle. So how do we create a pie chart

with Matplotlib? Before we go over them code to do that, let's do a quick recap

of our dataset. Recall that each row represents a country and contains

metadata about the country such as where it is located geographically and whether

it is developing or developed. Each row also contains numerical figures of

annual immigration from that country to Canada from 1980 to 2013.

Now let's process the dataframe so that the country name becomes the index of

each row. This should make retrieving rows pertaining to specific countries a

lot easier. Also let's add an extra column which represents the cumulative

sum of annual immigration from each country from 1980 to 2013. So for

Afghanistan for example, it is 58,639, total, and for Albania, it is

15,699 and so on. And let's name our dataframe df\_canada. So now

that we know how our data is stored in the dataframe df\_canada, say

we're interested in visualizing a breakdown of immigration to Canada

continent wise. The first step is to group our data by

continent using the continent column, and we use pandas for this. We call the

pandas groupby function on df\_canada, and we sum the number

of immigrants from the countries that belong to the

same continent. Here is the resulting dataframe, and let's name it

df\_continents. The resulting dataframe has six rows, each

representing a continent and 35 columns representing the years from 1980 to 2013

plus the cumulative sum of immigration for each continent. And now we're ready

to start creating our pie chart. We start with the usual, importing Matplotlib as

mpl and its scripting layer the pyplot interface as plt. Then we call the

plot function on column total of the dataframe df\_continents and

we set kind equals pie to generate a pie chart. Then to complete the figure we

give it a title. Finally we call the show function to

display the figure. And there you have it: A pie chart that depicts each continent's

proportion of immigration to Canada from 1980 to 2013. In the lab session, we will

go through the process of creating a very professional-looking and

aesthetically pleasing pie chart and transform the pie chart that we just

created into one that looks like this. So make sure to complete this module's lab

session. One last comment on pie charts. There are some very vocal opponents to

the use of pie charts under any circumstances. Most argue that pie charts

fail to accurately display data with any consistency. Bar charts are much

better when it comes to representing the data in a consistent way and getting the

message across. If you're interested in learning more about the arguments

against pie charts, here is a link to a very interesting article that discusses

very clearly the flaws of pie charts. You can also find the link under the video.

And with this we conclude our video on pie charts. I'll see you in the next

video.

# Box Plots

0:00

In this video we will learn about another visualization tool, the Boxplot and how to

create one using matplotlib.

So, what is a boxplot? A boxplot is a way of statistically representing the distribution

of given data through 5 main dimensions.

The first dimension is minimum, which is the smallest number in the sorted data.

Its value can be obtained by subtracting 1.5 times the IQR where IQR is interquartile range

from the first quartile.

The second dimension is first quartile which is 25% of the way through the sorted data.

In other words, 1/4 of the data points are less than this value.

The third dimension is median, which is the median of the sorted data.

The fourth dimension is third quartile, which is 75% of the way through the sorted data.

In other words, 3/4 of the data points are less than this value.

And the final dimension is maximum, which is the highest number in the sorted data where

maximum equals third quartile summed with 1.5 multiplied by IQR.

Finally, boxplots also display outliers as individual dots that occur outside the upper

and lower extremes.

Now let's see how we can create a boxplot with Matplotlib.

Before we go over the code to do that. Let's do a quick recap of our data set.

Recall that each row represents a country and contains metadata about the country, such

as where it is located geographically, and whether it is developing or developed.

Each row contains numerical figures of annual immigration from that country to Canada from

1980 to 2013.

Now let's process the data frame so that the country name becomes the index of each row.

This should make retrieving rows pertaining to specific countries a lot easier.

Also, let's add an extra column which represents the cumulative sum of annual immigration from

each country from 1980 to 2013. So, for Afghanistan, for example, it is 58,639 total and for Albania

it is 15,699.

And so on. And let's name our data frame, DF\_Canada.

So now that we know how our data is stored in the Dataframe DF\_Canada, say we're interested

in creating a boxplot to visualize immigration from Japan to Canada.

As with other tools that we learned so far, we start by importing Matplotlib as MPL and

the pyplot interface as PLT.

Then we create a new data frame of the data pertaining to Japan and we're excluding the

column total using the years variable.

Then we transpose the resulting data frame to make it in the correct format to create

the Boxplot.

Let's name this new data frame DF\_Japan.

Following that, we call the plot function on DF\_Japan and we set kind equals box to

generate a boxplot.

Then, to complete the figure, we give it a title and we label the vertical axis.

Finally, we call the show function to display the figure and there you have it, a boxplot

that provides a pleasing distribution of Japanese immigration to Canada from 1980 to 2013.

In the lab session we explore boxplots in more detail and learn how to create multiple

boxplots as well as horizontal boxplots, so make sure to complete this module's lab session.

And with this we conclude our video on boxplots, see you in the next video.

(Music)

# Scatter Plots

0:00

In this video, we will learn about an additional visualization tool: the

scatter plot, and we will learn how to create it using Matplotlib. So what is a

scatter plot? A scatter plot is a type of plot that displays values pertaining to

typically two variables against each other. Usually it is a dependent variable

to be plotted against an independent variable in order to determine if any

correlation between the two variables exists. For example, here is a scatter

plot of income versus education, and by looking at the plotted data, one can

conclude that an individual with more years of education is likely to earn a

higher income than an individual with fewer years of education. So how can we

create a scatterplot with Matplotlib? Before we go over the code to do that,

let's do a quick recap of our dataset. Recall that each row represents a

country and contains metadata about the country such as where it is located

geographically and whether it is developing or developed. Each row also

contains numerical figures of annual immigration from that country to Canada

from 1980 to 2013.

Now let's process the dataframe so that the country name becomes the index of

each row. This should make retrieving rows pertaining to specific countries a

lot easier. Also let's add an extra column which represents the cumulative

sum of annual immigration from each country from 1980 to 2013. So for

Afghanistan for example, it is 58,639, total, and for Albania it is

15,699 and so on. And let's name our dataframe df\_canada. So now

that we know how our data is stored in the dataframe, df\_canada, say

were interested in plotting a scatter plot of the total annual immigration to

Canada from 1980 to 2013. To be able to do that,

we first need to create a new dataframe that shows each year and the

corresponding total number of immigration from all the countries worldwide as

shown here. Let's name this new dataframe, df\_total. In the lab

session, we will walk together through the process of creating df\_total

from df\_canada, so make sure to complete this module's lab

session. Then we proceed as usual. We import Matplotlib as mpl and its

scripting layer, the pyplot interface, as plt. Then we call the plot function on

the data frame df\_total, and we set kind equals scatter to generate a

scatter plot. Now unlike the other data visualization

tools where only passing the kind parameter was enough to generate the

plot, with scatter plots we also need to pass the variable to be plotted on the

horizontal axis as the x-parameter, and the variable to be plotted on the

vertical axis as the y-parameter. In this case, we're passing column year as the

x-parameter and column total as the y-parameter. Then to complete the figure we

give it a title and we label its axes. Finally, we call the show function to

display the figure. And there you have it: A scatter plot that shows total

immigration to Canada from countries all over the world from 1980 to 2013. The

scatter plot clearly depicts an overall rising trend of immigration with time. In

the lab session, we explore scatter plots in more details and learn about a very

interesting variation of this scatter plot, a plot called the bubble plot, and

we learn how to create it using Matplotlib. So make sure to complete this

module's lab session. And with this, we conclude our video on scatter plots. I'll

see you in the next video.

In this module, you will learn about advanced visualization tools such as waffle charts and word clouds and how to create them. You will also learn about seaborn, which is another visualization library, and how to use it to generate attractive regression plots. In addition, you will learn about Folium, which is another visualization library, designed especially for visualizing geospatial data. Finally, you will learn how to use Folium to create maps of different regions of the world and how to superimpose markers on top of a map, and how to create choropleth maps.

**Learning Objectives**

* Apply advanced visualization tools to create waffle charts and word clouds.
* Use Seaborn with Matplotlibto generate attractive regression plots.
* Explain how to use the Folium, for visualizing geospatial data.
* Use Folium to create maps and superpose markers.
* Create Choropleth Maps with Folium.

# Waffle Charts

0:00

In this video, we will learn about what some consider an advanced visualization

tool, namely the waffle chart. So what is a waffle chart? A waffle chart is a great

way to visualize data in relation to a whole or to highlight progress against a

given threshold. For example, say immigration from Scandinavia to Canada

is comprised only of immigration from Denmark, Norway, and Sweden, and we're

interested in visualizing the contribution of each of these countries

to the Scandinavian immigration to Canada. The main idea here is for a given

waffle chart whose desired height and width are defined, the contribution of

each country is transformed into a number of tiles that is proportional to

the country's contribution to the total, so that more the contribution the more

the tiles, resulting in what resembles a waffle when combined. Hence the name

waffle chart. Unfortunately, Matplotlib does not have a built-in function to

create waffle charts. Therefore, in the lab session, I'll walk you through the

process of creating your own Python function to create a waffle chart, so

it's really important that you complete this module's lab session. And with this,

we conclude our video on waffle charts. I'll see you in the next video.

# Word Clouds

0:00

In this video, we will learn about another advanced visualization tool: the

word cloud. So what is a word cloud? A word cloud is simply a depiction of the

importance of different words in the body of text. A word cloud works in a

simple way; the more a specific word appears in a source of textual data the

bigger and bolder it appears in the world cloud. So given some text data on

recruitment, for example, we generate a cloud of words like this. This cloud is

telling us that words such as recruitment, talent, candidates, and so on,

are the words that really stand out in these text documents. And assuming that

we didn't know anything about the content of these documents, a word cloud

can be very useful to assign a topic to some unknown textual data. Unfortunately,

just like waffle charts, Matplotlib does not have a built-in function to generate

word clouds. However, luckily a Python library for cloud word generation that

was created by Andreas Mueller is publicly available. So, in the lab session

we will learn how to use Mueller's word cloud generator, and we will also create

interesting word clouds superimposed on different background images. So make sure

to complete this module's lab session. And with this, we conclude our video on word

clouds. I'll see you in the next video.

# Seaborn and Regression Plots

0:00

In this video, we will learn about a new visualization library in Python, which is

Seaborn. Although Seaborn is another data

visualization library, it is actually based on Matplotlib.

It was built primarily to provide a high-level interface for drawing

attractive statistical graphics, such as regression plots, box plots, and so on.

Seaborn makes creating plots very efficient. Therefore with Seaborn you

can generate plots with code that is 5 times less than with Matplotlib. Let's

see how we can use Seaborn to create a statistical graphic. Let's look into

regression plots. Let's say we have a dataframe called df\_total of

total immigration to Canada from 1980 to 2013 with the year in one column and the

corresponding total immigration in another column, and say we're interested

in creating a scatter plot along with a regression line to highlight any trends

in the data. With Seaborn, you can do all this with literally one line of code. The

way to do this, we first import Seaborn and let's import it as sns. Then, we call

the Seaborn regplot function. We basically tell it to use the dataframe

df\_total and to plot the column year on the horizontal axis and

the column total on the vertical axis. And the output of this one line of code

is a scatter plot with a regression line and not just that, but also 95%

confidence interval. Isn't that really amazing? Seaborn's regplot function

also accepts additional parameters for any personal customization. So you can

change the color for example using the color parameter. Let's go ahead and

change the color to green. Also, you can change the marker shape as well using

the marker parameter. Let's go ahead and change the shape of our markers to a +

marker instead of the default circular marker. In the lab session, we

explore regression plots with Seaborn in more details, so

make sure to complete this module's lab session. And with this we conclude our

short introduction to Seaborn and regression plots. I'll see you in the

next video.

# Introduction to Folium

0:00

In this video, we will learn about a very interesting data visualization library

in Python which is Folium. Folium is a powerful data visualization library in

Python that was built primarily to help people visualize geospatial data. With

Folium, you can create a map of any location in the world as long as you

know its latitude and longitude values. You can also create a map and

superimpose markers as well as clusters of markers on top of the map for cool

and very interesting visualizations. You can also create maps of different styles

such as street level map, stamen map, and a couple others which we will look

into in just a moment. Creating a world map with Folium is pretty

straightforward. You simply call the map function and that is all. What is really

interesting about the maps created by Folium is that they are interactive, so

you can zoom in and out after the map is rendered, which is a super useful feature.

The default map style is the open street map, which shows a street view of an area

when you're zoomed in and shows the borders of the world countries when

you're zoomed all the way out. Now let's create a world map centred around

Canada. To do that, we pass in the latitude and the longitude values of

Canada using the location parameter and with Folium you can set the initial zoom

level using the zoom start parameter. Now I say initial because you can easily

change the zoom level after the map is rendered by zooming in or zooming out.

You can play with this parameter to figure out what the initial zoom level

looks like for different values. Now, let's set the zoom level for our map of

Canada to 4. And there you go. Here is a world map centred around Canada.

Another amazing feature of Folium is that you can create different map styles

using the tiles parameter. Let's create a stamen toner map of Canada. This

style is great for visualizing and exploring river meanders

and coastal zones. Another style is stamen terrain. Let's create a map of

Canada in stamen terrain. This style is great for visualizing hill shading and

natural vegetation colors. And with this we conclude our introduction to Folium.

I'll see you in the next video.

# Maps with Markers

0:00

In this video, we will continue working with the Folium library and learn how

to superimpose markers on top of a map for interesting visualizations. In the

previous video, we learned how to create a world map centred around Canada, so

let's create this map again and name it canada\_map this time. Ontario

is a Canadian province and contains about 40 percent of the Canadian

population. It is considered Canada's most populous province. Let's see how we

can add a circular mark to the centre of Ontario. To do that, we need to create

what is called a feature group. Let's go ahead and create a feature group named

Ontario. Now when a feature group is created, it is empty and that means

what's next is to start creating what is called children and adding them to the

feature group. So let's create a child in the form of a red circular mark located

at the centre of the Ontario province. We specify the location of the child by

passing in its latitude and longitude values. And once we're done adding

children to the feature group, we add the featured group to the map. And there you

have it: A red circular mark superimposed on top of the map and added to the

centre of the province of Ontario. Now, it would be nice if we could actually label

this marker in order to let other people know what it actually represents. To do

that, we simply use the marker function and the pop up parameter to pass in

whatever text we want to add to this marker. And there you go: Now our marker

displays Ontario when clicked on. In the lab session, we will look into a

real-world example and explore crime rate in San Francisco. We will create a

map of San Francisco and superimpose thousands of these markers on top of the

map. Not just that but I'll show you how you can also create clusters of markers

in order to make your map look less congested. This module's lab session is a

very interesting one so please make sure to complete it.

And with this, we conclude our video on adding markers to maps with Folium. I'll

see you in the next video.

# Choropleth Maps

0:00

In this video, we will learn how to create a special type of map called

choropleth map with Folium. I'm sure that most of you have seen maps similar to

this one and this one. These are what we call choropleth maps. So what is a

choropleth map? A choropleth map is a thematic map in which areas are shaded

or patterned in proportion to the measurement of the statistical variable

being displayed on the map, such as population density or per capita income.

The higher the measurement the darker the color. So, the map to the left is a

choropleth map of the world showing infant mortality rate per 1000 births.

The darker the color the higher the infant mortality rate. According to the

map, African countries have very high infant mortality rates with some of them

reporting a rate that is higher than 160 per 1000 births.

Similarly, the map to the right is a choropleth map of the US showing

population per square mile by state. Again, the darker the color the higher

the population. According to the map, states in the eastern part of the US

tend to be more populous than states in the western part, with California being

an exception. In order to create a choropleth map of a region of interest,

Folium requires a Geo JSON file that includes geospatial data of the region.

For a choropleth map of the world, we would need a Geo JSON file that lists

each country along with any geospatial data to define its borders and

boundaries. Here is an example of what the Geo JSON file would include about

each country. The example here pertains to the country Brunei. As you can see, the

file includes the country's name, it's ID, geometry shape, and the coordinates that

define the country's borders and boundaries. So let's see how we can

create a choropleth map of the world like this one showing immigration to

Canada. Before we go over the code to do that, let's do a

quick recap of our dataset. Recall that each row represents a country and

contains metadata about the country such as where it is located geographically

and whether it is developing or developed. Each row also contains

numerical figures of annual immigration from that country to Canada from 1980 to

2013. Now let's process the data, and let's add

an extra column which represents the cumulative sum of annual immigration from

each country from 1980 to 2013. So for Afghanistan for example,

it is 58,639, total, and for Albania it is 15,699,

and so on. And let's name our dataframe df\_Canada.

So now that we know how our data is stored in the dataframe,

df\_Canada, let's see how we can generate a choropleth map of the

world showing immigration to Canada. We should be experts now in creating world

maps with Folium. So let's go ahead and create a world map, but this time let's

use the mapbox bright tiles set. The result is a nice world map displaying

the name of every country. Now to convert this map into a choropleth map, we first

define a variable that points to our Geo JSON file. Then we apply the choropleth

function to our world map and we tell it to use the columns "Country" and "Total" in

our df\_Canada dataframe, and to use the country names to look up the

geospatial information about each country in the Geo JSON file. And there

you have it: A choropleth map of Canada showing the intensity of immigration

from different countries worldwide. In the lab session, we explore choropleth

maps in more details, so please make sure to complete this module's lab session. And

with this, we conclude our video on choropleth maps.

# Module Overview and Learning Objectives

As the saying goes, `A picture worth thousand words`. Data visualization through dashboards will help you uncover information from data that are hidden and democratize the understanding of the extracted information.

In this topic, you will create a dashboard with theme `US Domestic Airline Flights Performance`. You will do this using a US airline reporting carrier on-time performance dataset, plotly, and dash concepts learned throughout the course.

 In this module, you will learn

- How a dashboard can be used to answer critical business questions.

- What high-level overview of popular dashboarding tools available in python.

- How to use basic Plotly, plotly.graph\_objects, and plotly express.

- How to use Dash and basic overview of dash components (core and HTML).

- How to add different elements (like text box, dropdown, graphs, etc) to the dashboard.

- How to add interactivity to dash core and HTML components.

This module will help you get started with dashboard creation using the Plotly library. Hands-on labs will follow each concept to make you comfortable with using the library.

Reading lists will reference additional resources to learn more about the concepts covered.

 Enjoy the course!

# Dashboarding Overview

0:08

In this video, we are going to see how an interactive data application

can help improve business performance, and the tools available for building the application.

Play video starting at ::18 and follow transcript0:18

With real-time visuals on the dashboard, understanding business moving parts becomes easy.

Based on the report type and data, suitable graphs and charts can be created in one

central location.  This provides an easy way for stakeholders to understand what is going right,

wrong, and what needs to be improved. Getting the big-picture in one place can

help businesses make informed decisions. This improves business performance.

In general, the best dashboards answer critical business questions.

Play video starting at ::51 and follow transcript0:51

Let's say you are assigned a task to monitor and report the performance of domestic US flights.

Following are the yearly review report items. The top 10 airline carrier in the year 2019

in terms of number of flights The number of flights in 2019 split by month

Play video starting at :1:11 and follow transcript1:11

And the number of travelers from California state to other states split by distance group

Let’s look at the two ways of presenting the report.

For this type 1 report, the information is presented through tables, with inference

from tables documented for reference. For report type 2 we are presenting

the same report in the dashboard format. Hovering over each chart will provide details

about the data points. In the bottom sunburst chart, you can click on different numbers,

drill down into levels and get detailed information about each segment.

Can you observe the difference in the presentation of the findings? What if we need to get the report

on the real-time data, not the static data? Also, presenting the result using tables and

documents is time-consuming, less visually appealing, and more difficult to comprehend.

Play video starting at :2:5 and follow transcript2:05

A data scientist should be able to create and deliver a story around the finding

in a way stakeholders can easily understand. With that in mind, dashboards are the way to go.

Play video starting at :2:17 and follow transcript2:17

Let's take a look at web-based dashboarding tool options available in Python.

Dash is a python framework for building web analytic applications. It is written on top

of Flask, Plotly.js, and React.js. Dash is well-suited for building

data visualization apps with highly custom user interfaces.

Play video starting at :2:39 and follow transcript2:39

Panel works with visualizations from Bokeh, Matplotlib, HoloViews,

and many other Python plotting libraries, making them instantly viewable either individually or

when combined with interactive widgets that control them. Panel works equally well in Jupyter

Notebooks, for creating quick data-exploration tools. Panel can also be used in standalone

deployed apps and dashboards, allowing you to easily switch between those contexts as needed.

Voilà turns Jupyter notebooks into standalone web applications.

It can be used with separate layout tools like jupyter-flex or templates like voila-vuetify.

Streamlit can easily turn data scripts into shareable web apps with 3 main principles:

embrace python scripting, treat widgets as variables, and reuse data and computation.

There are other tools that can be used for dashboarding:

Play video starting at :3:36 and follow transcript3:36

Bokeh is a plotting library, a widget and app library. It acts as a server for both

plots and dashboards. Panel is one of the web-based dashboarding tools built on Bokeh.

ipywidgets provides an array of Jupyter-compatible widgets

and an interface supported by many Python libraries, but sharing as a dashboard requires

a separate deployable server like Voila. Matplotlib is a comprehensive library

for creating static, animated, and interactive visualizations in Python.

Bowtie allows users to build dashboards in pure Python.

Flask is a Python-backed web server that can be used to build arbitrary web sites,

including those with Python plots that function as flask dashboards.

Play video starting at :4:22 and follow transcript4:22

Learn more about the tools from the source link. In this course, we will be focusing on Dash.

# Additional Resources for Dashboards

For more information about Dashboards, visit the following links:

[Python dashboarding tools](https://pyviz.org/dashboarding/)

[John Snow's data journalism](https://www.theguardian.com/news/datablog/2013/mar/15/john-snow-cholera-map)

# Introduction to Plotly

0:08

In this video, we are going to provide an overview of the Plotly python library.

So, what is Plotly ? Plotly is an interactive, open source plotting

library that supports over 40 unique chart types. It is available in Python, R and Javascript.

Plotly python is build on top of Plotly Javascript library and includes chart

types like statistical, financial, maps, scientific, and 3-dimensional data .

The web based visualizations created using Plotly python can be displayed in Jupyter notebook,

saved to standalone HTML files, or served as part of pure Python-built web applications using Dash.

The focus of this lesson will be on two of the Plotly sub-modules:

Plotly Graph Objects and Plotly Express Plotly Graph Objects is the low-level

interface to figures, traces, and layout. The Plotly graph objects module provides

an automatically generated hierarchy of classes ( figures, traces, and layout) called graph objects.

Play video starting at :1:15 and follow transcript1:15

These graph objects are used for representing figures with

a top-level class plotly.graph\_objects.Figure. Plotly express is a high-level wrapper for Plotly.

It is a recommended starting point for creating most common figures provided by Plotly

using a more simple syntax. It uses graph objects internally.

Play video starting at :1:38 and follow transcript1:38

Let's see how to use plotly.graph\_objects sub-module by creating a simple line chart.

First, import the required packages. Here we are importing graph objects as go.

Play video starting at :1:53 and follow transcript1:53

Then, generate sample data using numpy.

Play video starting at :1:58 and follow transcript1:58

The Plotly.graph contains a JSON object which has a dictionary structure.

Play video starting at :2:4 and follow transcript2:04

Since we imported plotly graph object as go in the previous slide, `go` will be the JSON object.

Play video starting at :2:12 and follow transcript2:12

The Chart can be plotted by up-dating the values of the go object keywords.

We will create the figure by adding a scatter type trace.

Next the layout of the figure is updated using the ”update layout” method. Here, we are updating

the x-axis, y-axis, and chart title. This is the plotted figure.

Now, we will create the same line chart using Plotly express.

In Plotly express, the entire line chart can be created using

a single command. Visualization is automatically interactive.

Plotly express makes visualization easy to create and modify.

It's time to play with the plotly library. Next is going to be a lab session.

We will be using the airline reporting dataset from data asset exchange to

demonstrate how to use Plotly graph objects and Express for creating charts.

Play video starting at :3:8 and follow transcript3:08

Here is a quick overview of the airline reporting dataset.

Play video starting at :3:13 and follow transcript3:13

The Reporting Carrier On-Time Performance Dataset contains information on approximately 200 million

domestic US flights reported to the United States Bureau of Transportation Statistics.

The dataset contains basic information about each flight (such as date, time, departure airport,

arrival airport) and, if applicable, the amount of time the flight was delayed and information

about the reason for the delay. Next let's start the lab.

# Additional Resources for Plotly

To learn more about using Plotly to create dashboards, explore

[Plotly python](https://plotly.com/python/getting-started/)

[Plotly graph objects with example](https://plotly.com/python/graph-objects/)

[Plotly express](https://plotly.com/python/plotly-express/)

[API reference](https://plotly.com/python-api-reference/)

Here are additional useful resources:

[Plotly cheatsheet](https://images.plot.ly/plotly-documentation/images/plotly_js_cheat_sheet.pdf)

[Plotly community](https://community.plotly.com/c/api/5)

[Related blogs](https://plotlygraphs.medium.com/)

[Open-source datasets](https://developer.ibm.com/exchanges/data/)

# Introduction to Dash

0:08

In this video, we are going to see an overview of Dash library.

Dash is a Open-Source User Interface Python library for creating reactive, web-based

applications. It is enterprise-ready and a first-class member of Plotly’s open-source tools.

Dash applications are web servers running Flask and communicating JSON packets over HTTP requests.

Dash’s frontend renders components using React.js. It is easy to build a Graphical User Interface

using dash as it abstracts all technologies required to build the applications.

Dash is Declarative and Reactive. Dash output can be rendered in web

browser and can be deployed to servers. Dash uses a simple reactive decorator

for binding code to the UI. This is inherently mobile and cross-platform ready.

Let's say you are planning to create an application to answer a business question.

As a first step, you need to determine the layout of the application.

Decide which chart to use and where to place for example. This is called `layout` part in dash.

The second part is to add interactivity to the application.

There are two components of Dash First is `Core components`

We can import core components as dcc using this import statement

Next is `HTML Components` We can import html components as

html using this import statement Let's explore these further.

Play video starting at :1:44 and follow transcript1:44

The dash\_html\_components library has a component for every HTML tag.

You can compose your layout using Python structures with the dash-html-components library.

Play video starting at :2: and follow transcript2:00

The dash\_html\_components library provides classes for all of the HTML tags.

The keyword arguments describe the HTML attributes like style, className, and id.

Play video starting at :2:15 and follow transcript2:15

No knowledge of HTML or CSS is required but can help in styling the dashboards.

Let's see an example of how to use HTML components.

Play video starting at :2:27 and follow transcript2:27

We start by creating a dash application. From here we create division in our application

layout and then adding components to it. In the outer layout division, we first

provide a name for our application using the HTML heading component H1.

The style parameter is used to change the font color, size and border of the heading.

Next, we add paragraph content to the page using a HTML paragraph component P.

Play video starting at :2:58 and follow transcript2:58

Division can be created inside the outer division. Here we are providing division content as `This

is a new division` and styling it using style parameter components.

To put all this together, in the application layout create a HTML

division and add components. Multiple divisions can be added to the outer application layout.

The dash\_core\_components describe higher-level components that are

interactive and generated with JavaScript, HTML, and CSS through the React.js library.

Some example of core components are Creating a slider, input area, check items, and datepicker.

You can explore other components using the reference link provided at the end of the slide.

Play video starting at :3:46 and follow transcript3:46

Let's see how to add a slider and dropdown to the application

For the dropdown, we use the dcc.dropdown component. We will create a dropdown list

under the options parameter as a dictionary. `Label` will hold the dropdown display label name

and `value` will hold the value of the label. We can also provide a default dropdown

display label using `value` parameter. For the slider, we use the dcc.slider

component and provide min and max value of the slider. The `marks` parameter is used

for adding a slide marker and `value` parameter for adding default value.

# Additional Resources for Dash

To learn more about Dash, explore

[Complete dash user guide](https://dash.plotly.com/)

[Dash core components](https://dash.plotly.com/dash-core-components)

[Dash HTML components](https://dash.plotly.com/dash-html-components)

[Dash community forum](https://community.plotly.com/c/dash/16)

[Related blogs](https://medium.com/plotly/tagged/dash)

# Make dashboards interactive

0:07

In this video, we will see how to connect core and HTML components using callbacks.

Play video starting at ::14 and follow transcript0:14

A callback function is a python function that is

automatically called by Dash whenever an input component's property changes.

Callback function is decorated with `@app.callback` decorator.

So what this decorator tells Dash? Basically, whenever there is a change

in the input component value, callback function wrapped by the decorator is

called followed by the update to the output component children in the application layout.

Let’s look at the callback function skeleton. First, create a function that will perform

operations to return the desired result for the output component.

Decorate the callback function with @app.callback decorator.

This takes two parameters. Output : This sets result returned

from the callback function to a component id Input: This set input provided to the callback

function to a component id From here we will connect

input and output to desired properties. We will see this in action with an example using

the airline data. The use case here is to extract the top 10 airline carriers in the provided input

year selected by the number of flights. Based on the input year, the output will change.

Play video starting at :1:30 and follow transcript1:30

First, we import the required packages. As seen before, we will import pandas, dash,

dash core, and HTML components. The new entry here is dash dependencies.

From dash dependencies, we will import input and output that we will use in the callback function.

We read the airline data into the pandas dataframe. We load

our dataframe at the start of the app and can be read inside the callback function.

We will start designing the dash application layout by adding components.

First, we will provide the title to the dash app using the HTML heading component

H1 and style it using the style parameter. Next, we will add an HTML division and text

input Core component. In-Dash, the inputs

and outputs of the application are simply the properties of a particular component.

In this example, our input is the "value" property of the component that has the ID

“input-yr". By default, the value has 2010. We will update this value in

our callback function. Lastly, we will add a

division with a graph core component. The core component has `bar-plot` as id,

which we will update inside the callback function. Note the component ids.

We will add a callback decorator `app.callback`. Input to the callback will be the component with

id `input-yr` and property `value` Output to the callback will be the

component with id `bar-plot` and property `figure` Component\_id and component\_property keywords are

optional and are included here for clarity. Next, we will define the callback function

`get\_graph` The entered year will be the input.

Using the year we extract the required information from data. Finally,

the application layout graph is updated. Lastly, we will run the application.

This is the output of the code. Our initial input year is 2010. Note that as we

update the year, graph is updated for that year. The second example is a callback with two inputs.

It is similar to the one input callback except for a few changes.

We will add a division with one more text inputs with the component id input-ab

Play video starting at :3:54 and follow transcript3:54

Now we will add the new input with component id `input-ab` to the decorator inside the list.

Next, we will define callback function `get\_graph`. This takes the entered year

and the entered state as inputs parameters. Computation is performed to extract the

information and the application layout is updated with the graph.

This is the output of the code. Our initial input year is 2010

and state is AL which is Alabama. As I update the year and state, you can observe that the

graph is updated in parallel. Next let’s start the lab!

# Additional Resources for Interactive Dashboards

To learn more about making interactive dashboards in Dash, visit

[Python decorators reference 1](https://realpython.com/primer-on-python-decorators/)

[Python decorators reference 2](https://www.python.org/dev/peps/pep-0318/#current-syntax)

[Callbacks with example](https://dash.plotly.com/basic-callbacks)

[Dash app gallery](https://dash-gallery.plotly.host/Portal/)

[Dash community components](https://plotly.com/dash-community-components/)

# Lesson Summary

- Best dashboards answer critical business questions. It will help business make informed decisions, thereby improving performance. - Dashboards can produce real-time visuals. - Plotly is an interactive, open-source plotting library that supports over 40 chart types. - The web based visualizations created using Plotly python can be displayed in Jupyter notebook, saved to standalone HTML files, or served as part of pure Python-built web applications using Dash. - Plotly Graph Objects is the low-level interface to figures, traces, and layout whereas plotly express is a high-level wrapper for Plotly. - Dash is an Open-Source User Interface Python library for creating reactive, web-based applications. It is both enterprise-ready and a first-class member of Plotly’s open-source tools. - Core and HTML are the two components of dash. - The dash\_html\_components library has a component for every HTML tag. - The dash\_core\_components describe higher-level components that are interactive and are generated with JavaScript, HTML, and CSS through the React.js library. - A callback function is a python function that is automatically called by Dash whenever an input component's property changes. Callback function is decorated with `@app.callback` decorator. - Callback decorator function takes two parameters: Input and Output. Input and Output to the callback function will have component id and component property. Multiple inputs or outputs should be enclosed inside either a list or tuple.