BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
WORK INTEGRATED LEARNING PROGRAMMES

COURSE HANDOUT

Part A: Content Design

Course Title	**Data Visualization and Interpretation**
Course No(s)	**ZG555**
Credit Units	5
Course Author	Febin.A.Vahab
Version No	1.0
Date	

Course Description

The course provides an insight on the best practices used in Data Visualization and also illustrates the best tools used to achieve the same

Course Objectives

No	Description
CO1	To introduce key techniques and theory used in visualization, including data models, graphical perception and techniques for visual encoding and interaction.
CO2	Solving various visualization problem using tools like Tableau, Python (Matplotlib)
CO3	Best Practices of Dashboard Design, Designing dashboards meeting the design principles for various requirements

Text Book(s)	
T1	Storytelling with Data, A data visualization guide for business professionals, by Cole Nussbaumer Knaflic; Wiley
T2	Data Visualisation : A Successful Design Process By Andy Kirk
T3	Visualize This: The Flowing Data Guide to Design, Visualization & Statistics, by Nathan Yau, Wiley
T4	Information Dashboard Design: Displaying data for at-a-glance monitoring, Stephen Few, second edition
T5	Tableau Your Data: Fast and Easy Visual Analysis with Tableau Software, by Daniel G Murray
T6	Matplotlib for Python Developers: Effective techniques for data visualization with Python, by Aldrin Yim, Claire Chung and Allen Yu
T7	Hands on Data Visualization with Bokeh: Interactive web plotting for Python using Bokeh, by Kevin Jolly
R1	Mastering Tableau, by David Baldwin

Learning Outcomes:

No	Learning Outcomes
LO1	Concepts and best practices of Data Visualization
LO2	Best practices of Information Dashboard Design
LO3	Data Visualization using Tableau
LO4	Data Visualization using Python (Matplotlib)

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Part B: Content Development Plan

Academic Term	First Semester 2019-20
Course Title	**Data Visualization and Interpretation**
Course No	**DSECL ZG555**
Credit	5
Content Developer	Febin.A.Vahab

Glossary of Terms

Module	**M**	Module is a standalone quantum of designed content. A typical course is delivered using a string of modules. M2 means module 2.
Contact Session	**CS**	Contact Session (CS) stands for a 2 hour long live session with students conducted either in a physical classroom or enabled through technology. In this model of instruction, instructor led sessions will be for 16 CS.
Recorded Lecture	**RL**	RL stands for Recorded Lecture or Recorded Lesson. It is presented to the student through an online portal. A given RL unfolds as a sequences of video segments interleaved with exercises.
Lab Exercises	**LE**	Lab exercises associated with various modules
Self-Study	**SS**	Specific content assigned for self study
Homework	**HW**	Specific problems/design/lab exercises assigned as homework

Modular Structure

Module Summary

No.	Title of the Module
M1	Data Visualizations and Practices
M2	Effective Dashboard Design
M3	Data Visualization with Tableau
M4	Data Visualization with Python – 1 (Matplotlib)
M5	Data Visualization with Python – 2 (Bokeh, Seaborn)

Detailed Structure

M1: Data Visualizations and Practices
Contact Session 1-3

Type	Description/Plan	Reference Text Book/Chapters

CS1	• Introduction
	• Exploiting the Digital age
	• Visualisation as a Discovery tool
	• Visualisation skills for the masses
	• The Visualisation methodology
	• Visualisation design objectives
	• Exploratory vs. explanatory analysis
	• Understanding the context for data presentations
	• 3 minute story
	• Effective Visuals
	  ○ Textuals
	  ○ Tabulars
	  ○ Graphicals
		T1
		Ch 1 and 2,T2 Ch1

	• Gestalt principles of visual perception • Visual Ordering • Decluttering	T1 Ch 3

CS2	• Preattentive attributes in text and graphs
	  ○ Size
	  ○ Color
	  ○ Position
	• Data Design concepts
	  ○ Affordances
	  ○ Accessibility
	  ○ Aesthetics
		T1 Ch 4
		
		T1 Ch 5

	• Storytelling • Visualization Design Lessons	T1 Ch 7 T1 Ch 8
CS3	**Taxonomy of Data Visualisation Methods** • Comparing Categories of Plots • Dot Plot • Bar Chart • Floating Bar • Histogram • Radial Chart • Glyph Chart	T2 Ch 5

	• Case Studies
	  ○ Visualizing Pattern Over time
	  ○ Visualizing Proportions
	  ○ Visualizing Relationships
		T3 , Ch 4, 5, 6

SELF STUDY		
• Data-Driven Documents (D3.js charts) ○ Exploring visual gallery		

○ Simple charts creation • https://d3js.org/ • Explore more D3 charts examples • Explore Google charts library • https://developers.google.com/chart/ • Good Enough to Great: A Quick Guide for Better Data Visualizations • https://www.tableau.com/learn/whitepapers/good-enough-great-quick-guide-better-data-visualizations

M2: Data Visualization with Tableau
Contact Session 4-7

Type	Description/Plan	Reference
CS4	• Exploring Tableau ○ User Interface ○ Tableau Prep ○ Data Connection ○ Data Preparation	T5 Ch 1, 2, 3 https://www.tableau.com/learn/training
CS5	• Visual Analytics ○ Data Analysis ○ Visuals	https://www.tableau.com/learn/training T5 Ch 3, 4
CS6	• Maps	T3 Ch 6
	• Dashboard and Stories	https://www.tableau.com/learn/training T5 Ch 8
CS7	• Beyond the Basic Chart Types ○ Bullet graphs ○ Pareto charts ○ Custom background images	R1 Ch 7
	• Visualization Best Practices and Dashboard Design	R1 Ch 10
SELF STUDY		
• Explore the different types of visuals that can be plotted with Tableau interface		

M3: Effective Dashboard Design
Contact Session 8-10

Type	Description/Plan	Reference
CS8	• Dashboard • Dashboard categorization and typical data • Characteristics of a Well-Designed Dashboard • Key Goals in the Visual Design Process	T4 Ch 2 and 5
	• Common Mistakes in Dashboard Design	T4 Ch 3
CS9	• Power of Visual Perception ○ Visually Encoding Data for Rapid Perception ○ Applying the Principles of Visual Perception to	T4 Ch 4

	Dashboard Design	
	• Effective Dashboard Display Media • Dashboards design for Usability	T4 Ch 6 T4 Ch 7
CS10	• Case Studies o Sample Sales Dashboard o Sample CIO Dashboard o Sample Telesales Dashboard o Sample Marketing Analysis Dashboard	T4 Ch 8
	• Bringing it all together with Dashboards o How Dashboard Facilitates Analysis and Understanding o How Tableau Improves the Dashboard-building process o The right way to build a Dashboard o Best Practices for Dashboard building	T5 Ch8
SELF STUDY		
• Explore any 2 dashboard design tools https://dzone.com/articles/20-free-and-open-source-data-visualization-tools • Build Your Competitive Edge: 12 Powerful Retail Dashboards https://www.tableau.com/learn/whitepapers/powerful-retail-dashboards • 10 Best Practices for Building Effective Dashboards https://www.tableau.com/learn/whitepapers/10-best-practices-building-effective-dashboards		

M4: Data Visualization with Python – 1 (Matplotlib)
Contact Session 11-14

Type	Description/Plan	Reference
CS11	• Merits of Matplotlib • The Lifecycle of a Plot • Pyplot • Matplotlib visuals basics	https://matplotlib.org/tutorials/index.html T6 Ch 1 and Ch2
CS12	• Plot styles types • Visual Decorations	http://www.labri.fr/perso/nrougier/teaching/matplotlib/ T6 Ch 3
CS13	• Advanced Matplotlib	T6 Ch4
CS14	**Matplotlib in the real word** • Plotting data from a database • Plotting data from a CSV file • Plotting extrapolated data using curve fitting • Plotting geographical data	T6 Ch9
SELF STUDY		
• Analysis of time series data using matplotlib		

- Plotting Univariate Distributions
- Plotting Bivariate Distributions

M5: Data Visualization with Python – 2 (Seaborn and Bokeh)
Contact Session 15-16

Type	Description/Plan	Reference

CS15	• Seaborn package
	  ○ Seaborn vs Matplotlib
	  ○ Data Loading
	  ○ Seaborn Basic Plots
		https://seaborn.pydata.org/
		
		https://www.datacamp.com/community/tutorials/seaborn-python-tutorial

	• Statistical plots with Seaborn	https://www.datacamp.com/courses/introduction-to-data-visualization-with-python
CS16	• Plotting using Glyphs • Plotting with different Data Structures	T7 Ch 1 T7 Ch 2
	• Using Annotations, Widgets, and Visual Attributes for Visual Enhancement • Building and Hosting Applications Using the Bokeh Server	T7 Ch 4 T7 Ch 5
SELF STUDY		
• Try out all the statistical plots mentioned in datacamp's tutorial • https://www.datacamp.com/courses/introduction-to-data-visualization-with-python • Try out the Bokeh tutorial • https://www.analyticsvidhya.com/blog/2015/08/interactive-data-visualization-library-python-bokeh/		

Evaluation Scheme:
Legend: EC = Evaluation Component; AN = After Noon Session; FN = Fore Noon Session

No	Name	Type	Duration	Weight	Day, Date, Session, Time
EC-1	Quiz	Online	-	5%	Dec
EC-1	Assignment	Online	-	25%	Dec,Feb/March
EC-2	Mid-Semester Test	Closed Book	1.5 hours	30%	Dec
EC-3	Comprehensive Exam	Open Book	2.5 hours	40%	March

Note: Assignment can be replaced by QUIZ also.
Syllabus for Mid-Semester Test (Closed Book): Topics in Session Nos. 1 to 7
Syllabus for Comprehensive Exam (Open Book): All topics (Session Nos. 1 to 16)

Important links and information:

CANVAS(LMS)

Students are expected to visit the CANVAS course page on a regular basis and stay up to date with the latest announcements and deadlines.

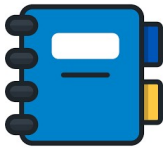
Contact sessions: Students should attend the online lectures as per the schedule provided on CANVAS.

Evaluation Guidelines:

1. EC1 consists of two assignments(Quiz/Assignment). Announcements will be made on the portal, in a timely manner.
2. For Closed Book tests: No books or reference material of any kind will be permitted.
3. For Open Book exams: Use of books and any printed / written reference material (filed or bound) is permitted. However, loose sheets of paper will not be allowed. Use of calculators is permitted in all exams. Laptops/Mobiles of any kind are not allowed. Exchange of any material is not allowed.
4. If a student is unable to appear for the Regular Test/Exam due to genuine exigencies, the student should follow the procedure to apply for the Make-Up Test/Exam which will be made available in CANVAS. The Make-Up Test/Exam will be conducted only at selected exam centres on the dates to be announced later.

It shall be the responsibility of the individual student to be regular in maintaining the self study schedule as given in the course handout, attend the online lectures, and take all the prescribed evaluation components such as Assignment/Quiz, Mid-Semester Test and Comprehensive Exam according to the evaluation scheme provided in the handout.

Agenda



- Visual Design Process
  - ✓ Dashboard Characteristics
  - ✓ Key goals in visual design process

1

Dashboard

A dashboard is visual display of the most important information needed to achieve one or more objectives, consolidated and arranged on a single screen, so that information can be seen at a glance.

--- Stephen Few

2

Dashboard (cont…)

Dashboard Challenge

- To display all required information on a single screen
  - Clearly and without distraction
  - In a manner that can be quickly examined and understood

3

Dashboard Characteristics

Characteristics of well designed dashboard

- Very well organized
- Summarized
- Customised
- Uses appropriate visual medium

4

Dashboard Characteristics (cont…)

Fundamental Principles for Dashboard display media

- Best suited for display of information
- Should be able to convey the same message when restricted to small area

5

Visual Design Process

Data-ink ratio

A large share of ink on a graphic should present data-information, the ink changing as the data change. Data-ink is non erasable core of a graphic, the non-redundant ink arranged in response to variation in the numbers represented.

-- Edward Tufte

6

Visual Design Process(cont…)

Data-ink ratio

= data-ink / total ink used to print the graphic

= proportion of a graphic's ink devoted to the non-redundant display of information

= 1 – proportion of a graphic that can be erased without loss of data-information

7

Visual Design Process(cont…)

Data-ink ratio

Maximize the data-ink ratio, within reason. Every bit on a graphic requires a reason. And nearly always that reason should be that the ink presents new information.

- Edward Tufte

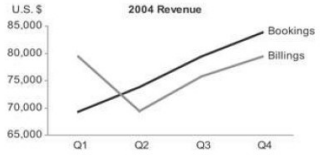
8

Visual Design Process(cont…)

Data-ink example

2005 YTD (U. S. $)

Region	Units	Bookings	Bookings %
Americas	3,888	229,392	43%
Europe	2,838	167,442	31%
Asia	1,788	105,492	20%
Other	509	30,031	6%
Total	$9,023	$532,357	100%



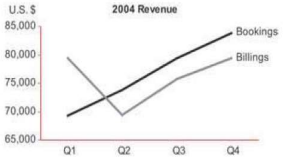
9

Visual Design Process(cont…)

Data-ink example

2005 YTD (U. S. $)

Region	Units	Bookings	Bookings %
Americas	3,888	229,392	43%
Europe	2,838	167,442	31%
Asia	1,788	105,492	20%
Other	509	30,031	6%
Total	$9,023	$532,357	100%

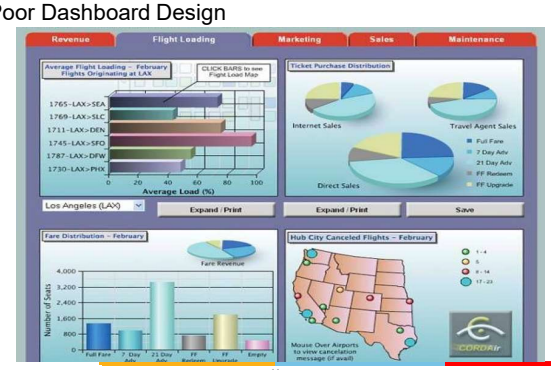


Note: Here, the non-data ink is highlighted in red

10

Visual Design Process(cont…)

Poor Dashboard Design



11

Visual Design Process(cont…)

Poor Dashboard Design

- Non-data pixels
  - Third dimension in pie and bars
  - Grid lines in bar graph
  - Background decoration
  - Background color variation in graphs

12

Visual Design Process(cont…)

Fundamental goals in Dashboard Design

- ❑ Reduce the non-data pixels
- ❑ Enhance the data pixels

13

Visual Design Process(cont…)

Reduce the non-data pixels

- ❑ Eliminate all unnecessary non-data pixels
- ❑ De-emphasize and regularise the non-data pixels that remain

14

Visual Design Process(cont…)

Eliminate all unnecessary non-data pixels:

- Unnecessary decoration

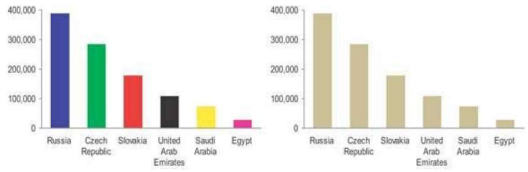
* You should eliminate graphics that provide nothing but decoration

15

Visual Design Process(cont…)

Eliminate all unnecessary non-data pixels

- Unnecessary color variation



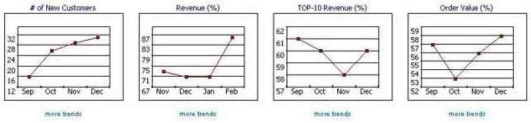
16

Visual Design Process(cont…)

innovate achieve lead

Eliminate all unnecessary non-data pixels

Enclosure and borders



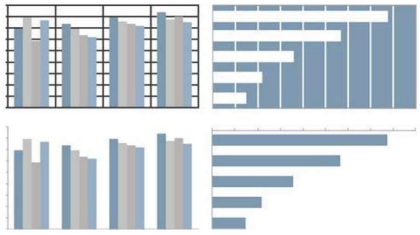
17

Visual Design Process(cont…)

innovate achieve lead

Eliminate all unnecessary non-data pixels

- Gridlines



* Grid lines in graphs are rarely useful. They are one of the most prevalent forms of distracting non-data pixels found in dashboards

18

Visual Design Process(cont…)

innovate achieve lead

Eliminate all unnecessary non-data pixels

- Gridlines

Salesperson	Jan	Feb	Mar
Robert Jones	2,834	4,838	6,131
Mandy Rodriguez	5,890	6,482	8,002
Terri Moore	7,398	9,374	11,748
John Donnelly	9,375	12,387	13,024
Jennifer Taylor	10,393	12,383	14,197
Total	$35,890	$45,464	$53,102

Salesperson	Jan	Feb	Mar
Robert Jones	2,834	4,838	6,131
Mandy Rodriguez	5,890	6,482	8,002
Terri Moore	7,398	9,374	11,748
John Donnelly	9,375	12,387	13,024
Jennifer Taylor	10,393	12,383	14,197
Total	$35,890	$45,464	$53,102

19

Visual Design Process(cont…)

innovate achieve lead

Eliminate all unnecessary non-data pixels

- Fill colors

Sell

Metric	Alerts	Result	Alert Spec	Last Update
QTD Sales ($MM)	2	$ 153.0	$ 166.0	1/21/02
QTD Average Daily Order Rate (ADOR) ($MM)	1	$ 16.1	$ 11.9	1/21/02
Previous Day's Orders ($MM)	0	$ 26.2	$ 11.9	1/21/02
QTD % e-Orders	3	53.0%	59.0%	1/21/02
Current Qtr Price vs Target ($/lb)	6	$ 1.27	$ 1.20	1/21/02

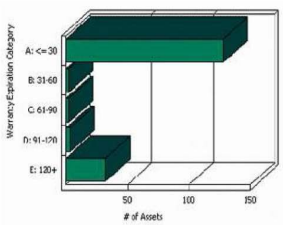
* Fill colors should be used to delineate rows in a table when this is necessary to help viewers' eyes track across the rows

20

Visual Design Process(cont…)

Eliminate all unnecessary non-data pixels

- 3D



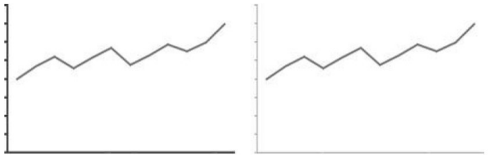
*3D should always be avoided when the added dimension of depth doesn't represent actual data

21

Visual Design Process(cont…)

De-emphasize and regularise the non-data pixels that remain

- Axis lines



22

Visual Design Process(cont…)

De-emphasize and regularise the non-data pixels that remain

- Lines, borders or fill colors

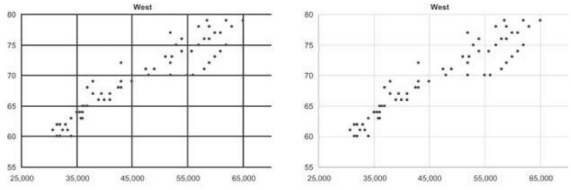
*Lines can be used effectively to delineate adjacent sections of the display from one another, but the weight of these lines can be kept to a minimum

23

Visual Design Process(cont…)

De-emphasize and regularise the non-data pixels that remain

- Grid lines (if necessary)



24

Visual Design Process(cont…)

innovate achieve lead

De-emphasize and regularise the non-data pixels that remain

- Grid lines and fill colors

Product	Jan	Feb	Mar	Q1 Total	Apr	May	Jun	Q2 Total	YTD Total
Product A	93,993	84,773	88,833	267,599	95,838	93,874	83,994	273,706	541,305
Product B	87,413	78,839	82,615	248,867	89,129	87,303	78,114	254,547	503,414
Product C	90,036	81,204	85,093	256,333	91,803	89,922	80,458	262,183	518,516
Product D	92,737	83,640	87,646	264,023	94,557	92,620	82,872	270,048	534,072
Product E	83,733	75,520	79,137	238,390	85,377	83,627	74,826	243,830	482,220
Total	447,913	403,976	423,323	1,275,212	456,705	447,346	400,264	1,304,314	2,579,526

Product	Jan	Feb	Mar	Q1 Total	Apr	May	Jun	Q2 Total	YTD Total
Product A	93,993	84,773	88,833	267,599	95,838	93,874	83,994	273,706	541,305
Product B	87,413	78,839	82,615	248,867	89,129	87,303	78,114	254,547	503,414
Product C	90,036	81,204	85,093	256,333	91,803	89,922	80,458	262,183	518,516
Product D	92,737	83,640	87,646	264,023	94,557	92,620	82,872	270,048	534,072
Product E	83,733	75,520	79,137	238,390	85,377	83,627	74,826	243,830	482,220
Total	447,913	403,976	423,323	1,275,212	456,705	447,346	400,264	1,304,314	2,579,526

*Grid lines and fill colors can be used in tables to clearly distinguish some columns from others, but this should be done in the muted manner seen below rather than the heavy handed manner seen above

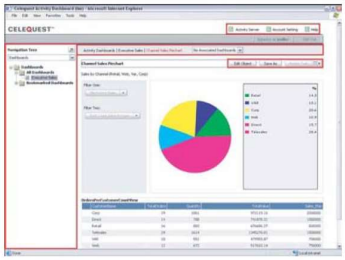
25

Visual Design Process(cont…)

innovate achieve lead

De-emphasize and regularise the non-data pixels that remain

- Buttons and user controls



26

Visual Design Process(cont…)

innovate achieve lead

De-emphasize and regularise the non-data pixels that remain

- Instructions / text

27

Visual Design Process(cont…)

innovate achieve lead

Enhance data pixels

- ❑ Eliminate all unnecessary data pixels
- ❑ Highlight the most important data pixels that remain

28

Visual Design Process(cont…)

Eliminate all unnecessary data pixels

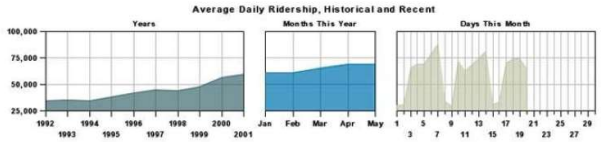
- ❑ Remove less relevant data
- ❑ Used condensed, summarised data

29

Visual Design Process(cont…)

Eliminate all unnecessary data pixels

- Use multi-foci displays



*These three time-series graphs displaying public transportation rider statistics contain there levels of detail: daily for the current month, monthly for the current year, and yearly for the last 10 years.

30

Visual Design Process(cont…)

Categories of the information

- ❑ Information that is always important
  - ❑ Key business measures
- ❑ Information that is only important at the moment
  - ❑ A measure that has fallen far behind its target

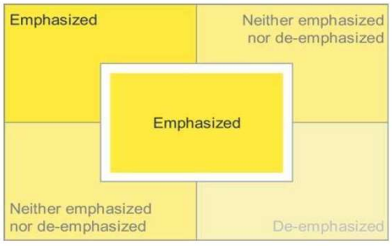
Both requires different ways of highlighting

31

Visual Design Process(cont…)

Highlight the most important data pixels that remain

- Most important information should be placed at top-left and center.



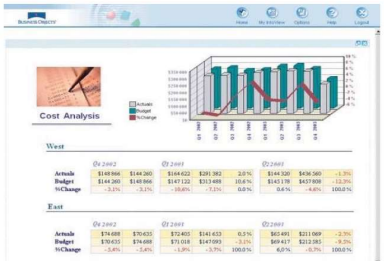
*Different degrees of visual emphasis are associated with different regions of a dashboard

32

Visual Design Process(cont…)

Highlight the most important data pixels that remain

- Waste of critical retail estate



*The most valuable real estate on this dashboard is dedicated to a company logo and meaningless decoration

33

Visual Design Process(cont…)

Approaches to static and dynamic highlighting of data

- ❑ Use Visual attributes that are greater than the norm
  - ❖ brighter or darker colors
- ❑ Use Visual attributes that simply contrast with the norm
  - ❖ Different color than the ones used in visual

34

Visual Design Process(cont…)

Use Visual attributes that are greater than the norm

Visual attribute	Useful expressions	Illustrations
Color intensity	A darker or more fully saturated version of any hue is naturally perceived as greater than a lighter or less-saturated version.	
Size	Bigger things clearly stand out as more important than smaller things.	
Line width	Thicker lines stand out as more important than thinner lines.	

35

Visual Design Process(cont…)

Use Visual attributes that simply contrast with the norm

Visual attribute	Useful expressions	Illustrations
Hue	Any hue that is distinct from the norm will stand out. [1]	
Orientation	Anything oriented differently than the norm will stand out.	
Enclosure	Anything enclosed by borders or surrounded by a fill color will stand out if different from the norm.	
Added marks	Anything with something distinctly added to it or adjacent to it will stand out.	

36

Visual Design Process(cont…)

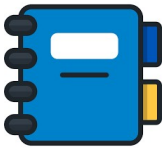
Example : Use of added marks

Metric	Actual	Variance
Revenue	$913,394	+$136,806
Profit	$193,865	-$73,055
Avg Order Size	$5,766	-$297
On Time Delivery	104%	+4%
New Customers	247	-62
Cust Satisfaction	4.73 / 5	+0.23

*Simple symbols can be used along with varying color intensities to dynamically highlight data

37

Agenda



- Dashboard Design for Usability

38

Dashboard Design for Usability

Usability aspects

- Information organization to support its meaning and use
- Consistency for quick and accurate interpretation
- Aesthetically pleasing viewing experience
- Design for a use as a launch pad
- Usability testing

39

Dashboard Design for Usability (cont…)

Information organization to support its meaning & use

Key points

- Grouping according to business functions, entities, roles
- Co-locating of objects belonging to same group
- Group delineation by least visible means
- Enhance meaningful comparison
- Discourage meaningless comparison

40

Dashboard Design for Usability (cont…)

Grouping according to business functions, entities, roles

- Information can be organized by
  - Business functions – like ordering, shipping, budgeting etc
  - Entities – like various departments , projects
  - Uses of data – like comparing sales metrics, cost metrics
- Learn
  - How information will be used
  - How the pieces ought to be arranged to best serve

41

Dashboard Design for Usability (cont…)

Co-locating of objects belonging to same group

- Use principle of similarity
- Place similar / related items closer to each other
- Still delineate them in simple manner

42

Dashboard Design for Usability (cont…)

Group delineation by least visible means

- Use white space effectively
- Otherwise, if space is issue, use subtle borders

43

Dashboard Design for Usability (cont…)

Group delineation by least visible means

Example – use white space

Product	Units Sold	Actual Revenue
Shirts	938	187,600
Blouses	1,093	114,765
Pants	3,882	62,112
Skirts	873	36,666
Dresses	72	2,088
Total	6,858	$403,231

Region	Units Sold	Actual Revenue
North	2,263	133,066
South	1,920	112,905
East	1,303	76,614
West	754	44,355
Canada	618	36,291
Total	6,858	$403,231

Channel	Units Sold	Actual Revenue
Direct	2,057	120,969
Distributor	1,921	119,903
Reseller	1,783	104,840
OEM	1,097	64,519
Total	6,858	$403,231

Warehouse	Units Sold	Actual Revenue
Virginia	2,537	149,195
California	1,920	112,905
Texas	1,372	80,646
Calgary	1,029	60,485
Total	6,858	$403,231

44

Dashboard Design for Usability (cont…)

Group delineation by least visible means

Example – use borders

Product	Units Sold	Actual Revenue
Shirts	938	187,600
Blouses	1,093	114,765
Pants	3,882	62,112
Skirts	873	36,666
Dresses	72	2,088
Total	6,858	$403,231

Region	Units Sold	Actual Revenue
North	2,263	133,066
South	1,920	112,905
East	1,303	76,614
West	754	44,355
Canada	618	36,291
Total	6,858	$403,231

Channel	Units Sold	Actual Revenue
Direct	2,057	120,969
Distributor	1,921	119,903
Reseller	1,783	104,840
OEM	1,097	64,519
Total	6,858	$403,231

Warehouse	Units Sold	Actual Revenue
Virginia	2,537	149,195
California	1,920	112,905
Texas	1,372	80,646
Calgary	1,029	60,485
Total	6,858	$403,231

45

Dashboard Design for Usability (cont…)

Enhance meaningful comparison

By

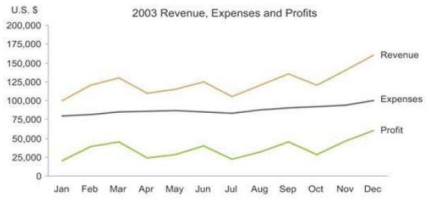
- Combining items in single table or graph
- Placing items close to each other
- Using different colors for different groups
- Use ratios, percentages i.e. actual values

46

Dashboard Design for Usability (cont…)

Enhance meaningful comparison

Example – Multiple measures in one graph



47

Dashboard Design for Usability (cont…)

Enhance meaningful comparison

Example – Using table for all measures

Product	Units Sold	Actual Revenue	% of Total	Forecast Revenue	% of Fcst
Product A	938	187,600	47%	175,000	107%
Product B	1,093	114,765	28%	130,000	88%
Product C	3,882	62,112	15%	50,000	124%
Product D	873	36,666	9%	40,000	92%
Product E	72	2,088	1%	50,000	4%
Total	6,858	$403,231	100%	$445,000	91%

48

Dashboard Design for Usability (cont…)

Discourage meaningless comparison

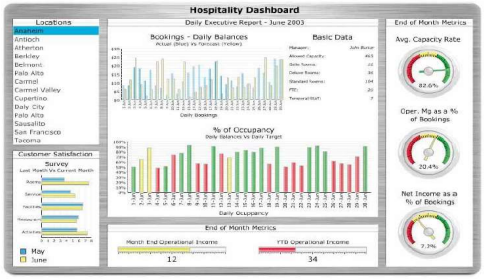
- By
  - Spatially separating items
  - Using different colors

49

Dashboard Design for Usability (cont…)

Discourage meaningless comparison

Example – useless comparisons



*This dashboard inadvertently encourages meaningless comparison

50

Dashboard Design for Usability (cont…)

Consistency for quick and accurate interpretation

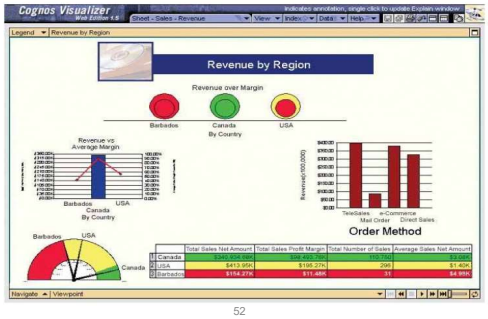
- Small difference triggers alarm
- Maintain consistency in
  - Visual appearances of display media
  - Choice of display media
- Use same mediums for same kind of comparisons
- Don't add variety just for sake of it

51

Dashboard Design for Usability (cont…)

Aesthetically pleasing viewing experience

- Example – ugly dashboard



52

Dashboard Design for Usability (cont…)

Aesthetically pleasing viewing experience

- Choose colors appropriately
  - Minimum bright colors to be used
  - Use less saturated colors
  - Use pale background colors
- Choose high resolution for clarity
  - Images with poor resolution are hard to read
- Choose right text
  - Use most legible font

53

Dashboard Design for Usability (cont…)

Design for use as a Launch Pad

- Design for interaction
  - Drill to details
  - Slicing and dicing of data
- Allow viewer to launch details section by clicking on data itself
- Use consistent launch actions

54

Dashboard Design for Usability (cont…)

Usability testing

- Difficult to get away with predetermined notions of user
- Present users with single prototype of most effective design
- Don't spoil users with the choices of design
- Present it to the actual users with real data for testing
- Take feedback and iterate over the design process again

55

Recap

Visual Design Process

- ✓ Characteristics of Dashboard
- ✓ Key goals of design process
  - Reduce the non-data pixels
  - Enhance the data pixels

56

Recap

Dashboard Design for Usability

- ✓ Organization of information to support meaning
- ✓ Consistency for quick & accurate interpretation
- ✓ Aesthetically pleasing viewing experience
- ✓ Design for use as a launch pad
- ✓ Usability testing

57

References

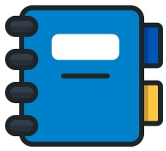
Information Dashboard Design

Stephen Few

- ❑ Chapter 5 : Eloquence Through Simplicity
- ❑ Chapter 7 : Designing Dashboard for Usability

58

Agenda



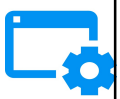
- ❑ Dashboards by examples
  - ❑ Sales Dashboard
  - ❑ CIO dashboard
  - ❑ Telesales, and
  - ❑ Marketing Analysis dashboard
- ❑ Best Practices for Tableau Dashboard

59

Sample Sales Dashboard

Critical Metrics

- ❑ Sales revenue
- ❑ Sales revenue in pipeline
- ❑ Profit
- ❑ Customer satisfaction rating
- ❑ Top 10 customers
- ❑ Market Share



60

Sample Sales Dashboard

innovate achieve lead

Dashboard 1

Critique of Sales Dashboard Example 1

SALES DASHBOARD - 19 December 2004

Performance: Good / Satisfactory / Poor

		Actual Q1	Actual Q2	Actual Q3	To Date Q4	Forecast Q4	Target Q4
Revenue Total		$ 154,057	$ 165,158	$ 199,738	$ 206,264	$ 225,205	$ 215,000
by Region	North America	$ 78,963	$ 78,138	$ 91,176	$ 91,441	$ 100,197	$ 107,500
	Europe	$ 30,811	$ 33,032	$ 39,948	$ 41,253	$ 45,374	$ 43,000
	Asia	$ 28,877	$ 37,472	$ 48,641	$ 52,944	$ 57,380	$ 39,775
	South America	$ 3,041	$ 3,435	$ 4,206	$ 5,035	$ 5,738	$ 5,375
	Middle East	$ 12,365	$ 13,081	$ 15,767	$ 15,592	$ 16,515	$ 19,350
by Product	Cabernet	$ 28,430	$ 30,228	$ 35,053	$ 38,728		$ 38,700
	Zinfandel	$ 13,876	$ 10,164	$ 17,876	$ 18,664		$ 19,350
	Merlot	$ 25,440	$ 24,977	$ 26,955	$ 26,865		$ 36,550
	Chardonnay	$ 68,634	$ 64,025	$ 104,063	$ 107,610		$ 98,900
	Sauvignan Blanc	$ 17,677	$ 35,763	$ 13,790	$ 12,399		$ 21,500
Profit		$ 31,999	$ 36,749	$ 42,431	$ 46,685		$ 53,750
Avg Order Size		405	421	435	449		430
Market Share		23%	20%	19%	17%		27%
Customer Satisfaction		3.18	2.95	2.82	2.67		3.29
On-Time Delivery		83%	73%	65%	68%		95%
New Customers		346	430	447	468		450

BITS Pilani

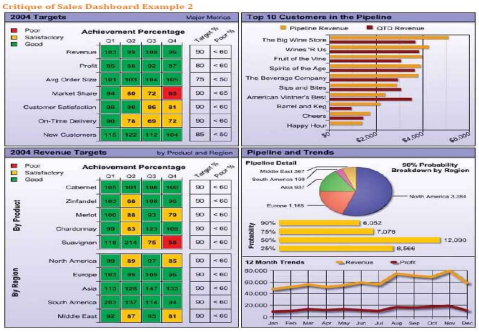
61

Sample Sales Dashboard

innovate achieve lead

Dashboard 2

Critique of Sales Dashboard Example 2



BITS Pilani

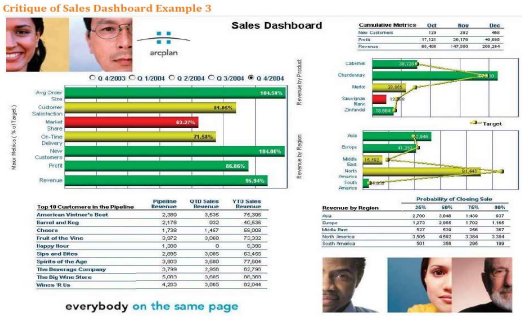
62

Sample Sales Dashboard

innovate achieve lead

Dashboard 3

Critique of Sales Dashboard Example 3



BITS Pilani

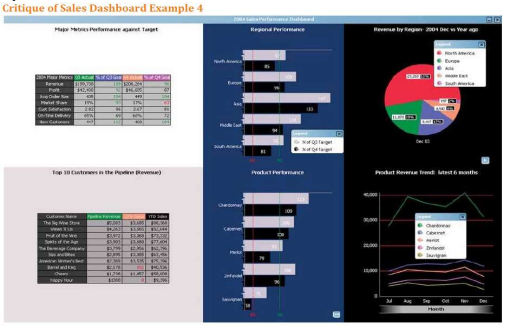
63

Sample Sales Dashboard

innovate achieve lead

Sample Dashboard 4

Critique of Sales Dashboard Example 4



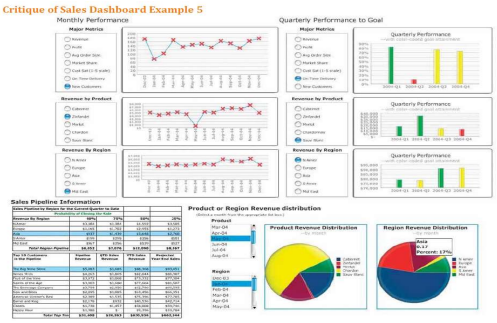
BITS Pilani

64

Sample Sales Dashboard

innovate achieve lead

Dashboard 5



BITS Pilani

65

Sample Sales Dashboard

innovate achieve lead

Dashboard 6

BITS Pilani

66

Sample CIO Dashboard

innovate achieve lead

Data to be reported:

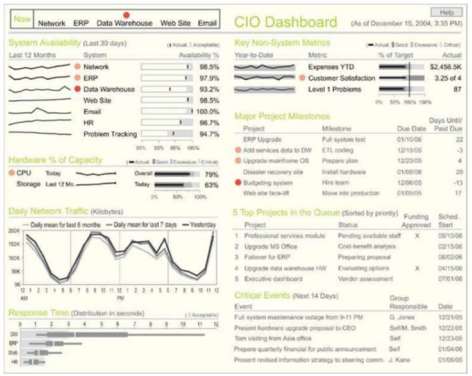
- System availability (uptime)
- Expenses
- Customer satisfaction
- Severe problem count
- CPU usage relative to capacity
- Storage usage relative to capacity
- Network traffic
- Application response time
- Major project milestones
- Top projects in the queue
- Other critical events

BITS Pilani

67

Sample CIO Dashboard

innovate achieve lead



BITS Pilani

68

Sample Telesales Dashboard

Data to be reported:

- Call wait time
- Call duration
- Abandoned calls (that is, callers who got tired of waiting and hung up)
- Call volume
- Order volume
- Sales representative utilization (representatives online compared to the number available)

69

Sample Telesales Dashboard

70

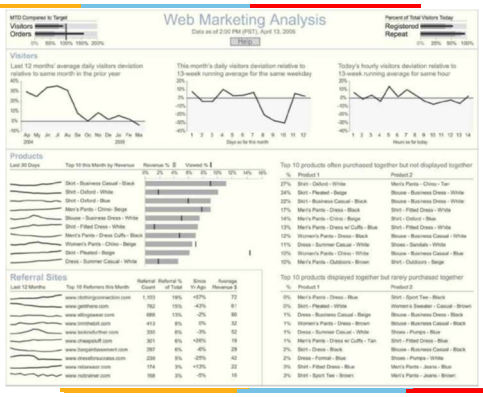
Sample Marketing Dashboard

Data to be reported (based on website activities):

- Number of visitors (daily, monthly, and yearly)
- Number of orders
- Number of registered visitors
- Number of times individual products were viewed on the site
- Occasions when products that were displayed on the same page were rarely purchased together
- Occasions when products that were not displayed on the same page were purchased together
- Referrals from other web sites that have resulted in the most visits

71

Sample Marketing Dashboard



72

Best Practices for Dashboard building in Tableau

- Size the dashboard to fit the in the worst-case available space.
- Employ 4-pane dashboard designs.
- Use Actions to filter instead of Quick Filters.
- Build cascading dashboard designs to improve load speed.
- Limit the use of color to one primary color scheme.
- Use small instructions near the work to make navigation obvious.
- Filter information presented in crosstabs to provide relevant details on-demand.
- Remove all non-data ink.
- Avoid One Size Fits All dashboards.

73

Best Practices for Dashboard building in Tableau

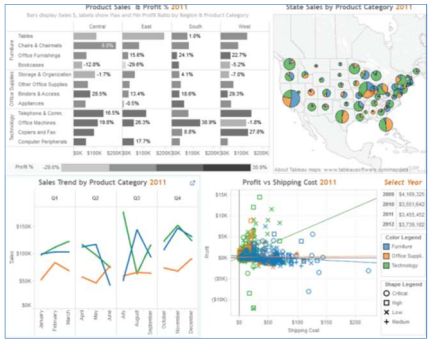
Size the dashboard to fit the in the worst-case available space.

- Determining the pixel height and width of the worst-case dashboard consumption environment.
- Tableau provides defaults for the typical sizes you will need or allows you to define a custom size.

74

Best Practices for Dashboard building in Tableau

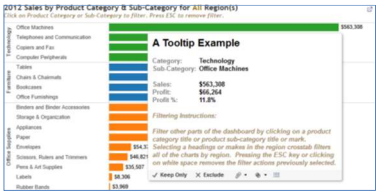
Employ 4-pane dashboard designs.



75

Best Practices for Dashboard building in Tableau

Use small instructions near the work to make navigation obvious.

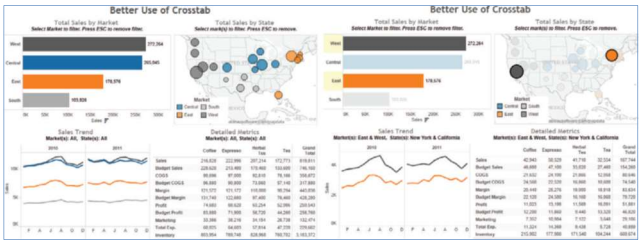


76

Best Practices for Dashboard building in Tableau

innovate achieve lead

Filter information presented in crosstabs to provide relevant details on-demand



BITS Pilani

77

References

innovate achieve lead

Information Dashboard Design

Stephen Few

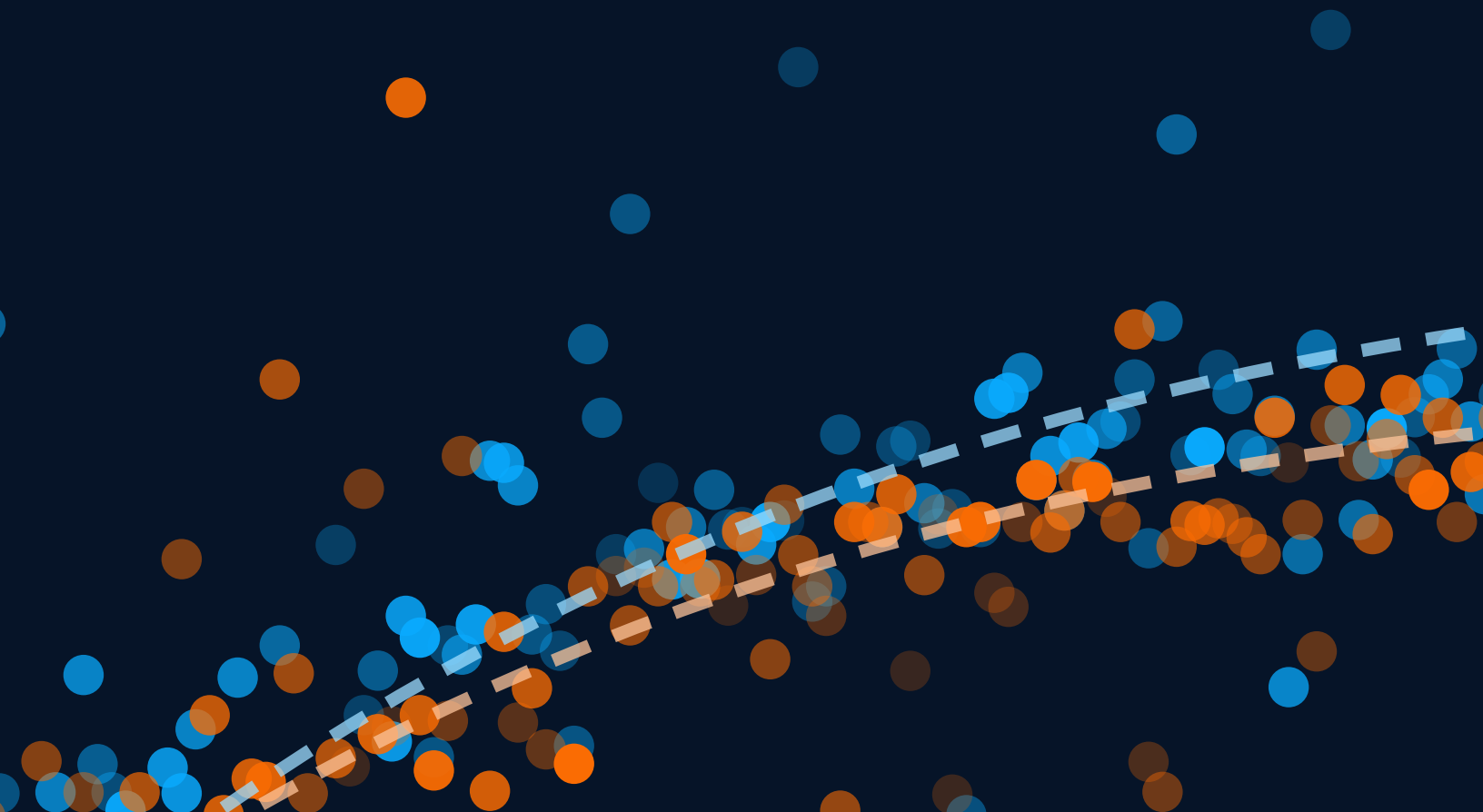
- Chapter 8 : Putting It All Together

BITS Pilani

78

+ a b | e a u

10 Best Practices for Building Effective Dashboards



A well-designed dashboard is a powerful launch point for data-driven conversations. Armed with the same collection of information, your business makes faster decisions based on a single source of truth.

A great dashboard's message and metrics are clear, its color enhances meaning, and it delivers the most relevant information to your audience. So how do you build dashboards for your organization that live up to this promise?

It really comes down to three things: thoughtful planning, informed design, and a critical eye for refining your dashboard.

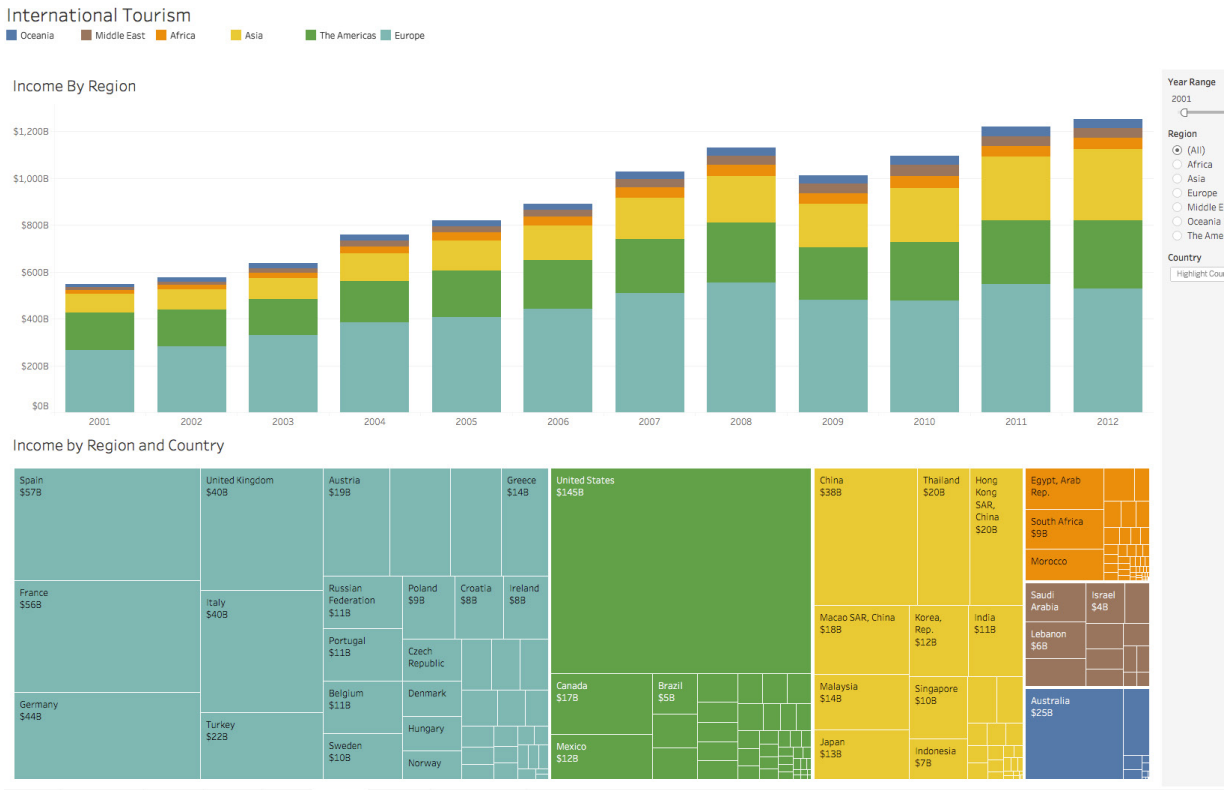
Contents

Thoughtful Planning

1. Know your audience

The best dashboards are built with their intended audience in mind. This doesn't happen by accident. Ask yourself, who am I designing this for? Is it a busy salesperson with 15 seconds to spare for key performance indicators, or is it a team reviewing quarterly dashboards over several hours?

It's also important to know your audience's level of expertise with the subject matter and data. For example, a beginner might need more action-oriented labelling for filters or parameters than an advanced user. If you don't know a lot about the audience, start by asking questions about their priorities and how they consume data to inform the best way to present the data. Remember that you can always create more dashboards. The best approach is to start simple.

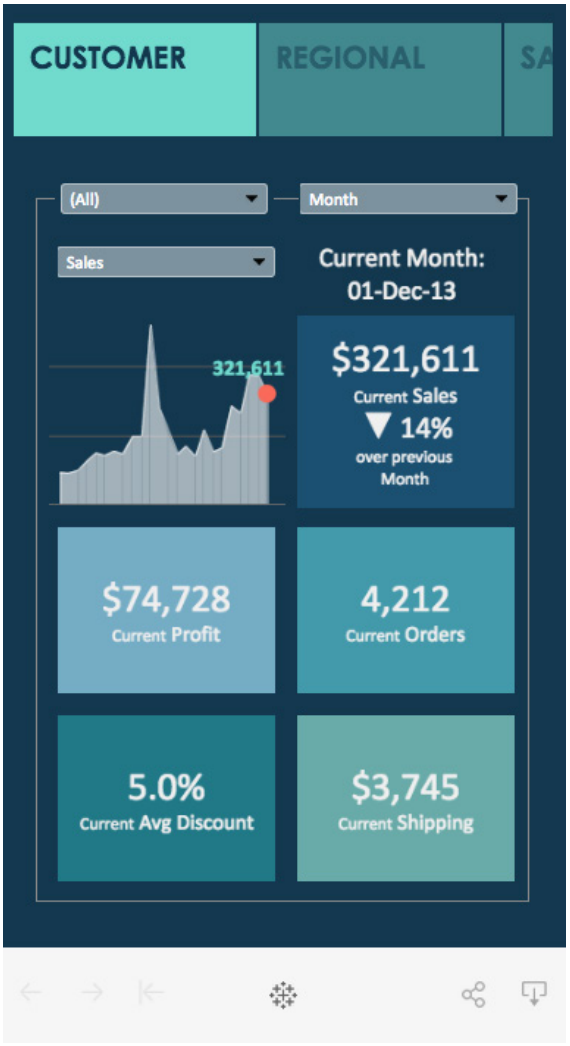


A financial team tasked with reviewing high-level information about international tourism income could easily use this dashboard.

2. Consider display size

If you build a dashboard solely for a desktop monitor, but your viewers primarily use their mobile phones to consume data, you're probably not going to have a very satisfied audience. Do some research up front to understand how your audience's habits might inform your dashboard design.

Surface the most important KPIs: Remember that your audience won't always be able to drill down on a small screen, so when you design for mobile phones or tablets, show only the most important metrics. In practice, this means creating dashboards with elements that are easy to click and have limited, intentional interactivity.



In this dashboard, there are no more than three interactions. This simplicity reduces confusion and helps with overall user experience on mobile.

Stack content vertically for phone screens: Most people use their phones in portrait mode. Unless you need to show a wide map view or timeline, prioritize optimizing your dashboard vertically for phones.

In Tableau, phone layouts are automatically generated whenever you create a new dashboard, arranging the dashboard's contents algorithmically in a phone-friendly manner. You can also choose the "Edit layout myself" option to manually add and arrange items to reflect changes to the Default dashboard. To see how your dashboards appear on different devices, review and add device layouts with Device Preview.

3. Plan for fast load times

Even the most beautiful dashboard won't have an impact if it takes too long to load. Sometimes long load times are caused by your data, your dashboard, or a combination of the two.

Some of the most critical decisions you make as an author begin in the data preparation stage before you even create your first view. Wherever possible, especially on production views, perform calculations in the database to reduce overhead. Aggregate calculations are great for calculated fields in Tableau, but perform row-level calculations in the database when you can.

Determine if you need to limit the amount of data surfaced in your dashboard, either by creating filters on a data source or creating an extract. Extracts are typically much faster than a live data source, and are especially great for prototyping. Keep in mind that extracts are not always the long-term solution. When querying against constantly-refreshing data, a live connection often makes more sense when operationalizing the view.

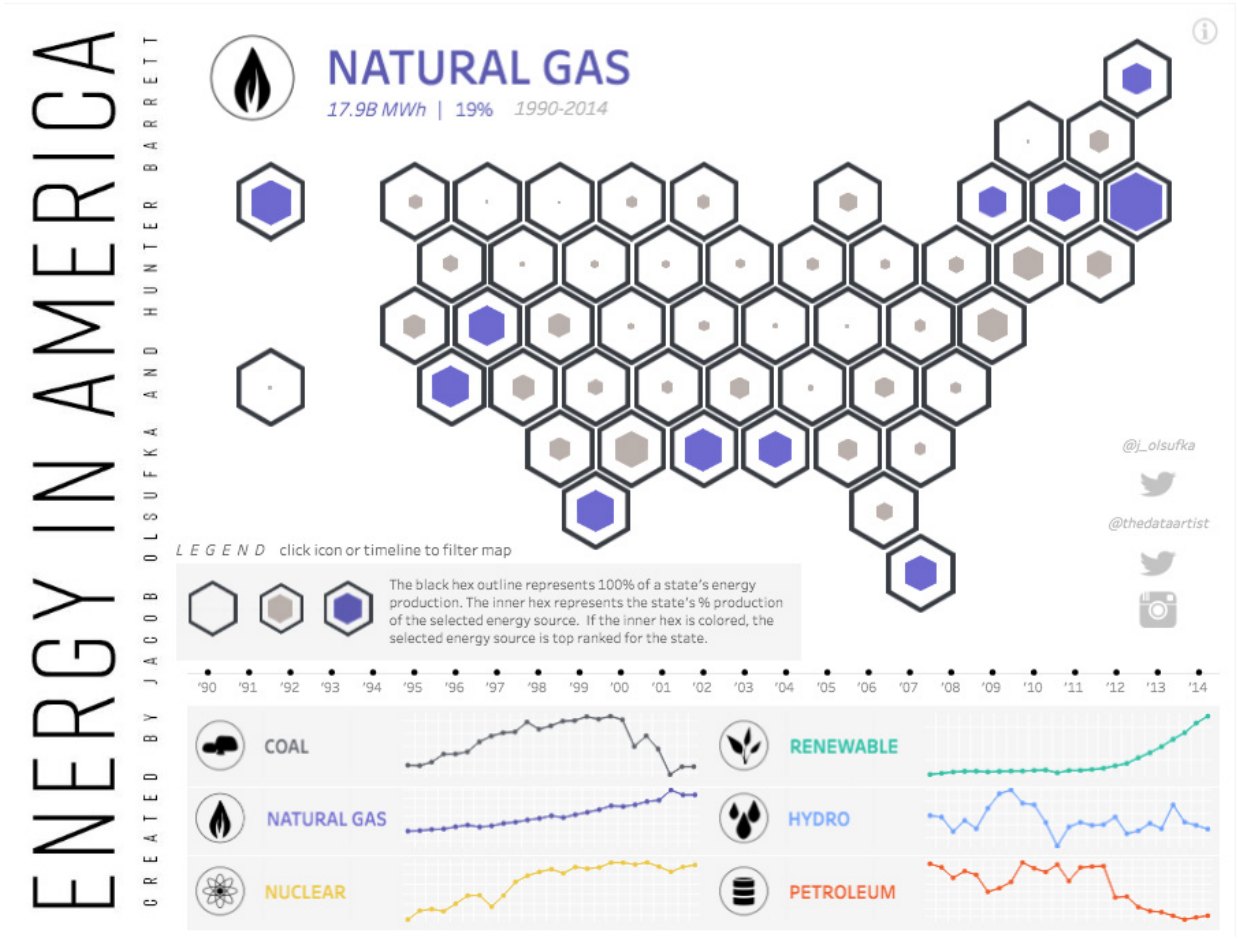
For more optimization tips, learn how to Optimize Workbook Performance in the Online Help. Knowing Tableau's Order of Operations may also help you shave time off your load times.

Informed Design

4. Leverage the sweetspot

Always consider how your audience will "read" your dashboard. Your dashboard should have a sensible "flow" and a logical layout of different pieces of information.

As you design your dashboard, consider the parts that form logical groups and use your design to group them together. Shading, lines, white space, and color are all useful ways to make the connections.



Jacob Olsufka grouped the hexes closely so we easily perceive the map of the United States. He also gave the legend and supporting text a common background, and grouped the social icons using proximity.

Most viewers scan web content starting at the top left of a web page. Once you know your dashboard's main purpose, be sure to place your most important view so that it occupies or spans the upper-left corner of your dashboard. In the dashboard above, the author decided that the header and the map view hold the key messages.

5. Limit the number of views and colors

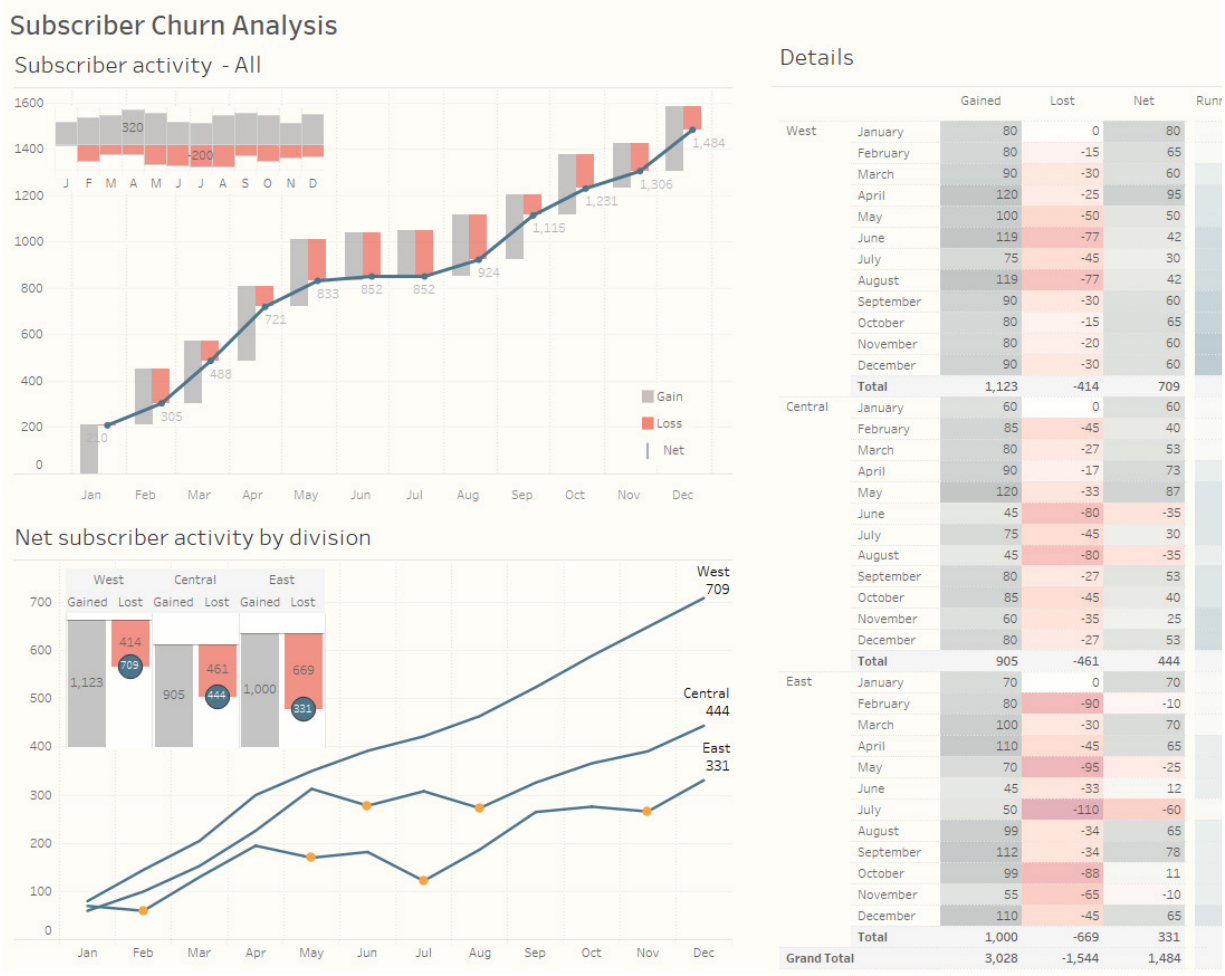
It's easy to get excited and want to cram your dashboard with every relevant view. But if you add too many, you'll sacrifice the big picture. In general, stick to two or three views. If you find that the scope needs to grow beyond that, create more dashboards or use a story—a sequence of visualizations that work together to guide the viewer through information.

Just like you can have too many views, you can also have too many colors. Color used correctly enhances analysis. Too many colors creates visual overload for your audience, slowing analysis and sometimes preventing it.

Here's a before-and-after for a dashboard about subscriber churn.

This "before" version uses harsh, more saturated colors and inconsistent shading, making it much harder for the viewer to see the relationship between the charts.

		Gained	Lost	Net
West	January	80	0	80
	February	80	-15	65
	March	90	-30	60
	April	120	-25	95
	May	100	-50	50
	June	119	-77	42
	July	75	-45	30
	August	119	-77	42
	September	90	-30	60
	October	80	-15	65
	November	80	-20	60
	December	90	-30	60
	Total	**1,123**	**-414**	**709**
Central	January	60	0	60
	February	85	-45	40
	March	80	-27	53
	April	90	-17	73
	May	120	-33	87
	June	45	-80	-35
	July	75	-45	30
	August	45	-80	-35
	September	80	-27	53
	October	85	-45	40
	November	60	-35	25
	December	80	-27	53
	Total	**905**	**-461**	**444**
East	January	70	0	70
	February	80	-90	-10
	March	100	-30	70
	April	110	-45	65
	May	70	-95	-25
	June	45	-33	12
	July	50	-110	-60
	August	99	-34	65
	September	112	-34	78
	October	99	-88	11
	November	55	-65	-10
	December	110	-45	65
	Total	**1,000**	**-669**	**331**
Grand Total		**3,028**	**-1,544**	**1,484**

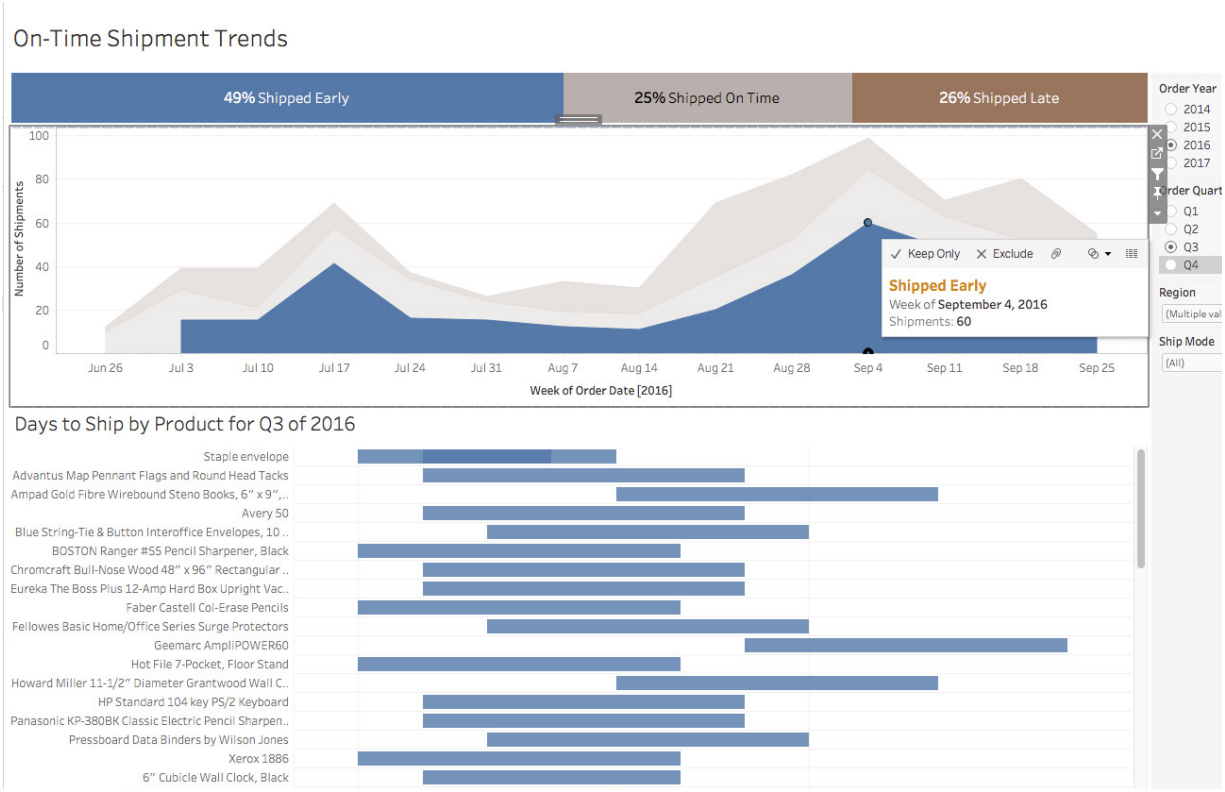


This revised version of the same dashboard has a modern design with minimal colors, creating a gentle formatting. Subscriber Churn, The Big Book of Dashboards

As addicting as it is to customize dashboards, avoid adding unnecessary objects that get in the way of your dashboard's capacity to quickly inform your audience.

6. Add interactivity to encourage exploration

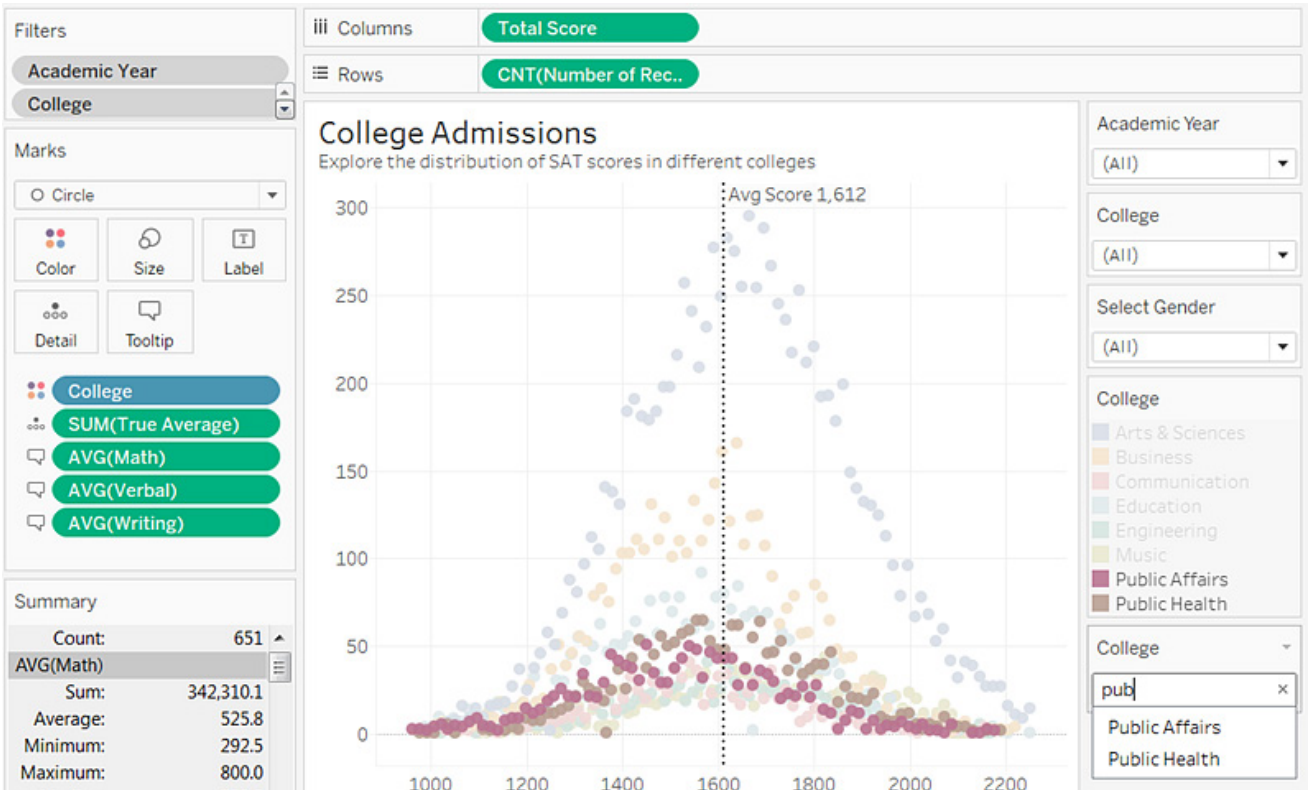
The power of dashboards lies in the author's ability to queue up specific views for side-by-side analysis. Filters super-charge that analysis and engage your audience. For example, you can have one view—your most important one—act as a filter on the other views in the dashboard. To do this, select Use as Filter from the view's shortcut menu.



This dashboard uses the area chart as a filter. When you click on the area chart, the bar chart below filters to only show the data classified as "Shipped Early," allowing the audience to dig into the data that is relevant to them.

You can also display filter cards for different types of data. For example, show filters as multi-select checkboxes, single select radio buttons, drop-down lists, etc. You can include a search box and edit the title of your filter to give your viewers clear instructions for interacting with the data.

Highlight actions are another powerful feature you can leverage, where a selection in one view highlights related data in the other views. For more advanced scenarios, you can use set actions or parameter actions to add deeper levels of interactivity.



This visualization uses Highlight Actions to increase interactivity. Searching "public" in the wildcard filter highlights the categories of colleges—in this case, public affairs and public health.

7. Format from largest to smallest

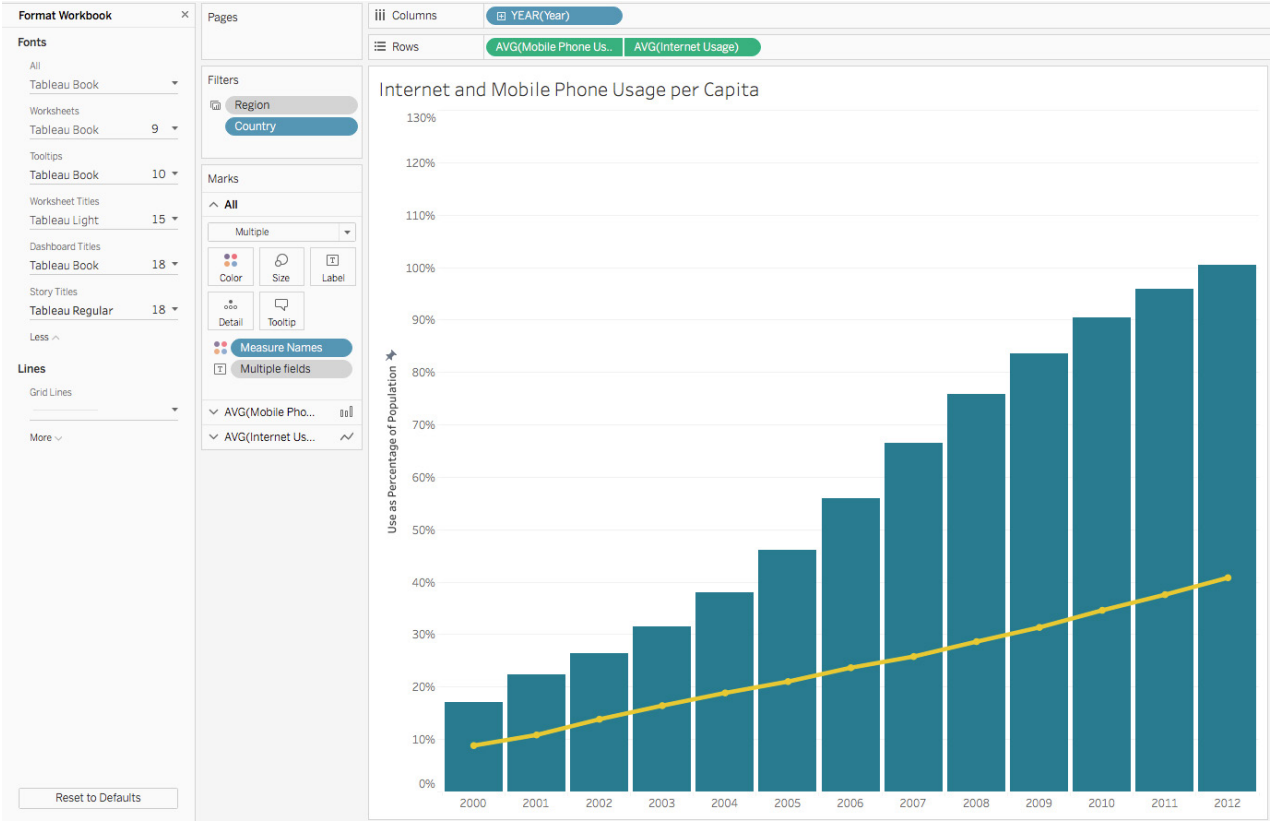
As you change the look and feel of your work, use a "largest to smallest" workflow. This will help you work quickly and keep you from accidentally overwriting your changes.

From a formatting perspective, the hierarchy of a dashboard looks like this:

1. Theme
2. Workbook
3. Worksheet

Start by confirming that you're using the right theme (Tableau's latest and greatest is always called Default). Choose one by going to Format > Workbook Theme.

The next step is to format at the workbook level. Here, you can change fonts, titles, and lines across your entire workbook.



Create consistency with formatting. Select Format > Workbook in Tableau to adjust the formatting for your entire workbook

Finally, move on to the worksheet level. For example, you might want to remove all the borders in a text table or add shading to every other column in a view. Save this step for last because when you make formatting changes at this level, they apply only to the view you're working on.

For tips on how to quickly give your dashboard a new look, including how to use your own custom fonts and colors, check out Rebrand a Dashboard in the Online Help.

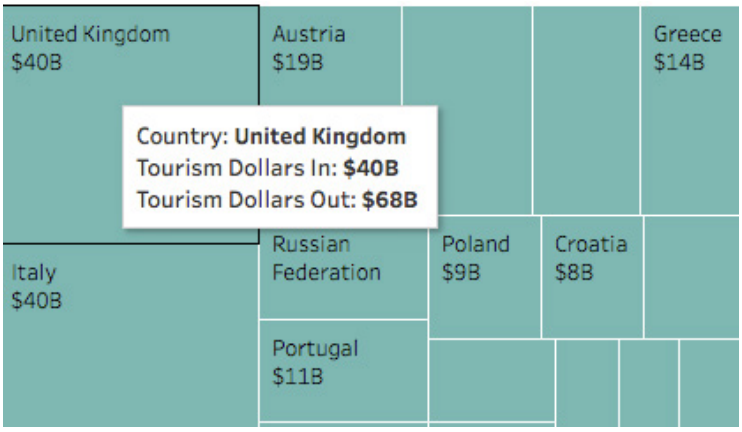
Refining your dashboard

8. Leverage tooltips, the story within your story

Once you're done with the main design work, take a look at your tooltips. Tooltips are a fantastic opportunity to reinforce the story you're trying to tell with your dashboard. They also add helpful context to your view. Tableau populates a view's tooltips automatically, but you can easily customize them by clicking Worksheet > Tooltip.

Just like you want to put your most important view in the upper left of your dashboard, you want the most important elements of your tooltip to be at the top.

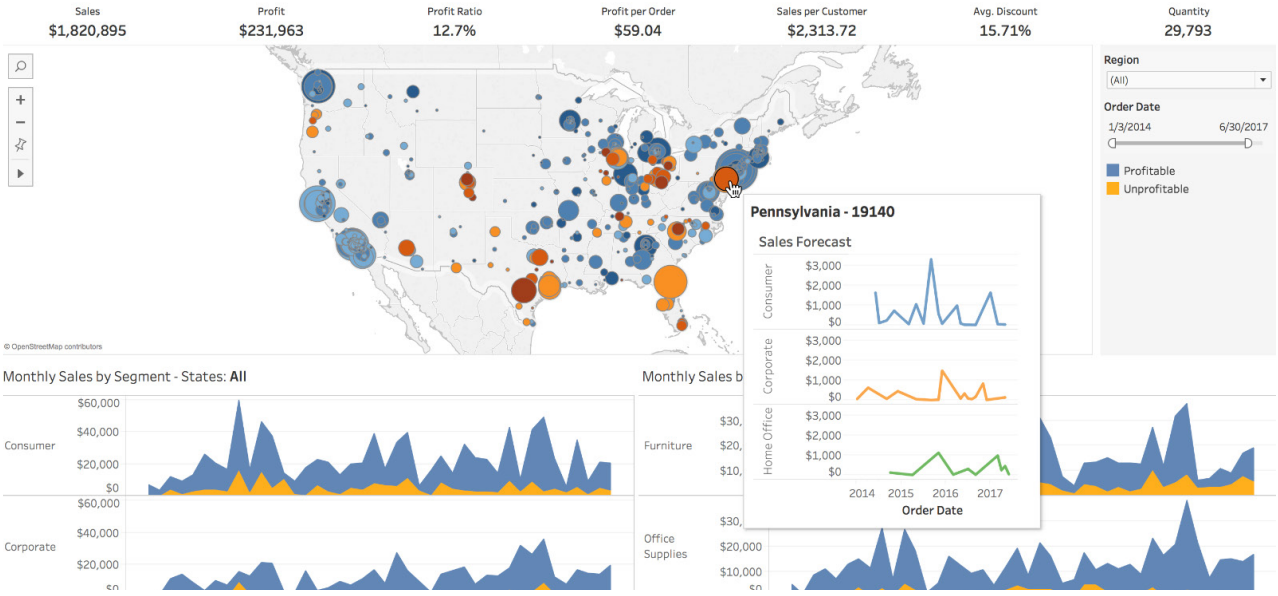
For example, let's say we're looking at a visualization showing international tourism by region and country.



At first glance, the tooltip on the left doesn't tell me what I need to know—what is the international tourism income for each country, related to its overall GDP?

When revised, this tooltip emphasizes the most important elements—the country, it's inbound and outbound tourism dollars, and the GDP of the country.

You can also use the Viz in Tooltip feature to augment your dashboards and stories with relevant data without introducing more clutter. With Viz in Tooltip, place visualizations of your own design into tooltips, revealing them on hover or selection of individual marks. The data in the viz is automatically filtered to the mark you hover on or select, giving you and your users precise views of the pertinent data. As always, make sure that the visualization is enhancing, not distracting from the contents of your dashboard. When in doubt, keep the rest of the dashboard simple and inform the user that they will get more context in the tooltip.

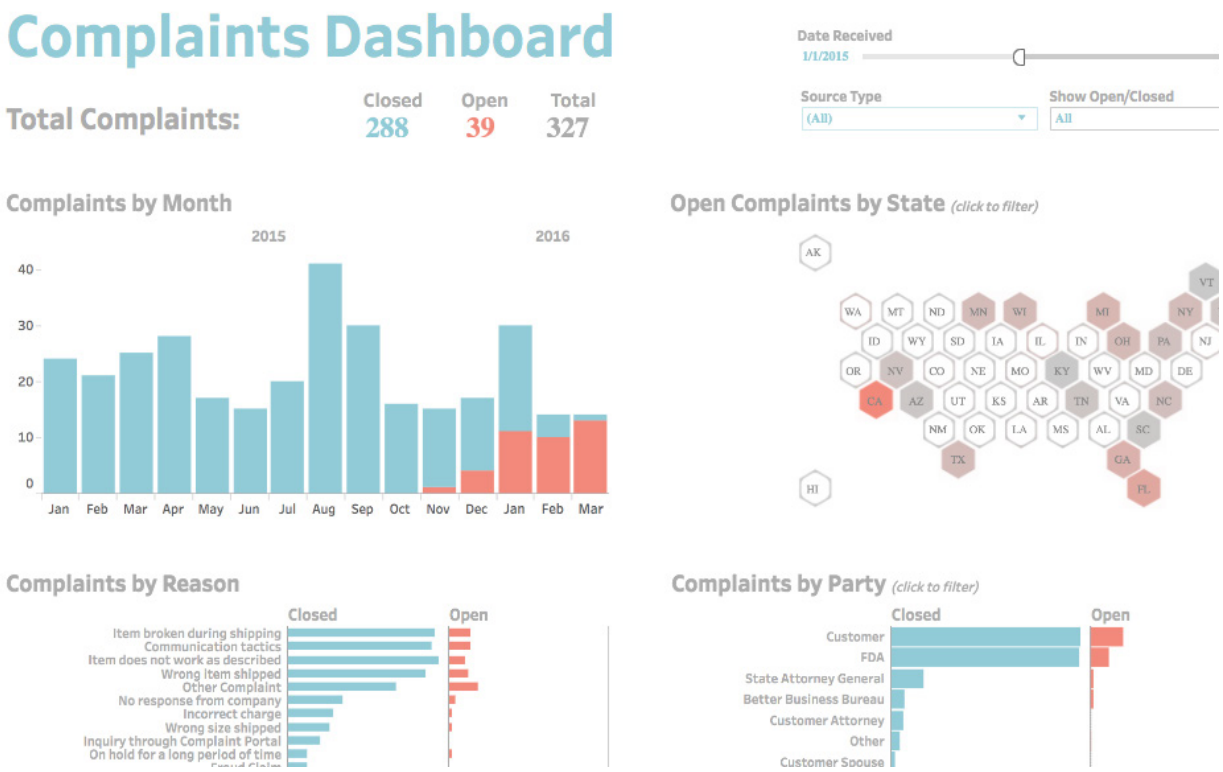


This dashboard leverages the Viz in Tooltip feature. When a user hovers over a mark on the map, the tooltip displays sales forecasts in each segment for that particular state. This adds context without crowding the dashboard.

9. Eliminate clutter

Take a step back and consider your dashboard from the perspective of someone who's never seen it. Every element should serve a purpose. If a title, legend, or axis label isn't necessary, consider getting rid of it.

If your dashboard needs more white space, consider a floating layout. If you go this route, give your dashboard a specific, fixed size so that the item that you floated stays put if the dashboard size changes.



This dashboard is a good example of simple, clean design. When you eliminate clutter and simplify your colors and layout, it is much easier to find hidden insights because you aren't trying to sift through all of the individual elements. Complaints Dashboard, The Big Book of Dashboards.

Simplifying your dashboard design is often an iterative process, so keep going back to existing dashboards with fresh eyes. To start, look at the latest dashboard you created: does it have too much on it? Is there anything you can remove or rearrange to add clarity?

10. Test your dashboard for usability

An important element of dashboard design is user testing. After you build a prototype, ask your audience how they're using the dashboard and if it helps them answer their pressing questions. Have they created their own versions of the dashboard? Are they digging into certain views and ignoring others? Use this information to tweak the existing dashboard or develop new ones.

As with any successful project, good testing is key. Learning how your dashboards are received will help inform future designs and influence how data is leveraged within your organization.

About Tableau

Tableau helps people transform data into actionable insights that make an impact. Easily connect to data stored anywhere, in any format. Quickly perform ad-hoc analyses that reveal hidden opportunities. Drag and drop to create interactive dashboards with advanced visual analytics. Then share across your organization and empower teammates to explore their perspective on data. From global enterprises to early-stage startups and small businesses, people everywhere use Tableau's analytics platform to see and understand their data.

Explore other resources

Product Demo

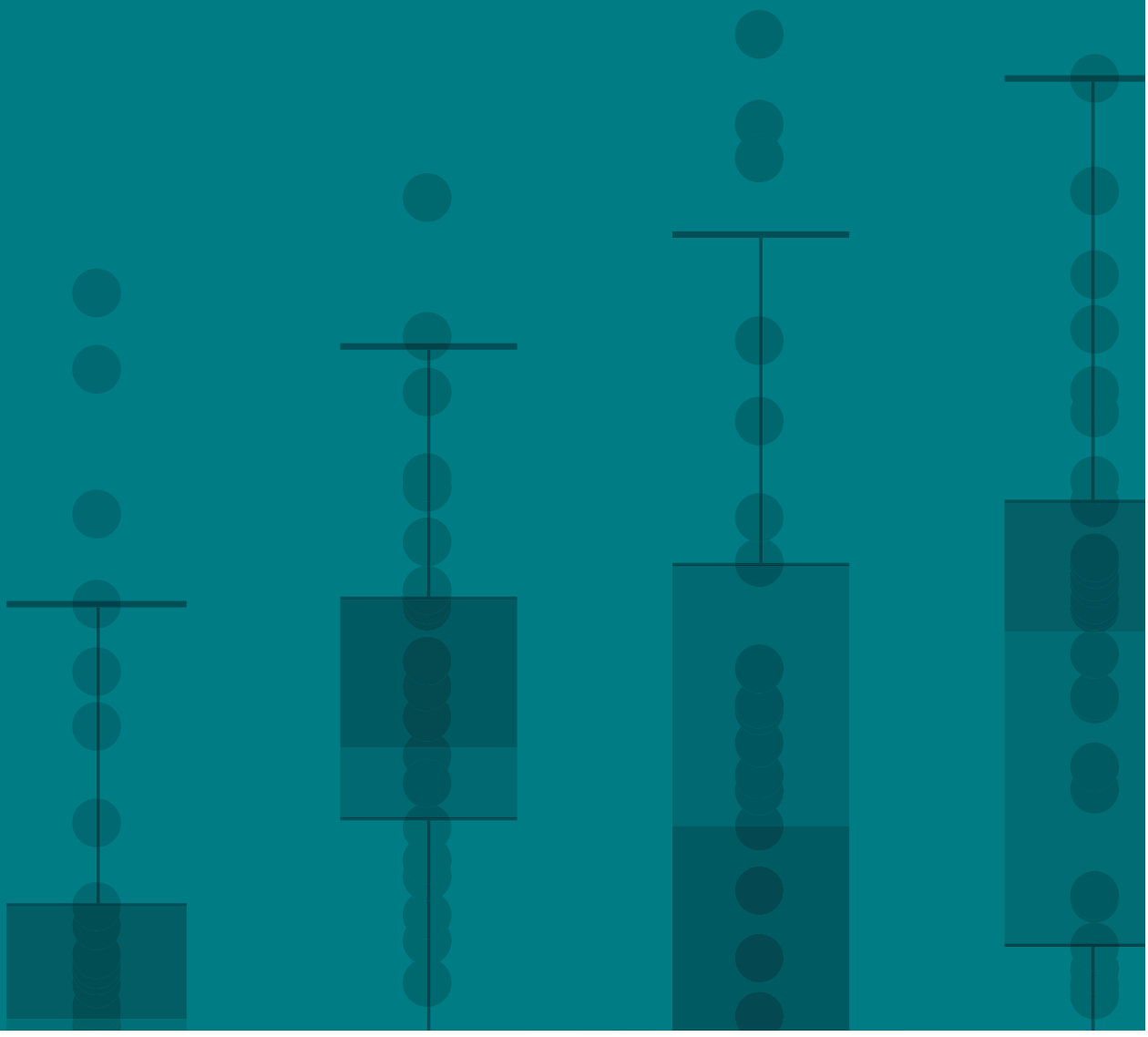
Training & Tutorials

Community & Support

Customer stories

Solutions

Build Your Competitive Edge:
12 Powerful Retail Dashboards



A common ask from our customers is to see more real-world retail and consumer goods examples. Last year, we responded by asking some of our incredible partners from around the world to provide their best retail and consumer good visualizations—which became the basis of our Top 10 Retail Dashboards for Better Performance white paper.

This year, we've reached out once again to our partner community and our partners have responded with amazing, retail-focused visualizations centered around store operations, merchandising, and marketing. These areas have been a big focus for our customers over the last year, and these dashboards have played a critical role in helping our customers build a competitive edge.

We hope these new visualizations inspire you to help raise the bar of analytics within your organization. We also hope it helps to reinforce the amazing capabilities and industry knowledge our partners have.

Table of Contents

1. Store Level Product Availability

Partner: **Atheon Analytics**
Find it on **Tableau Public**

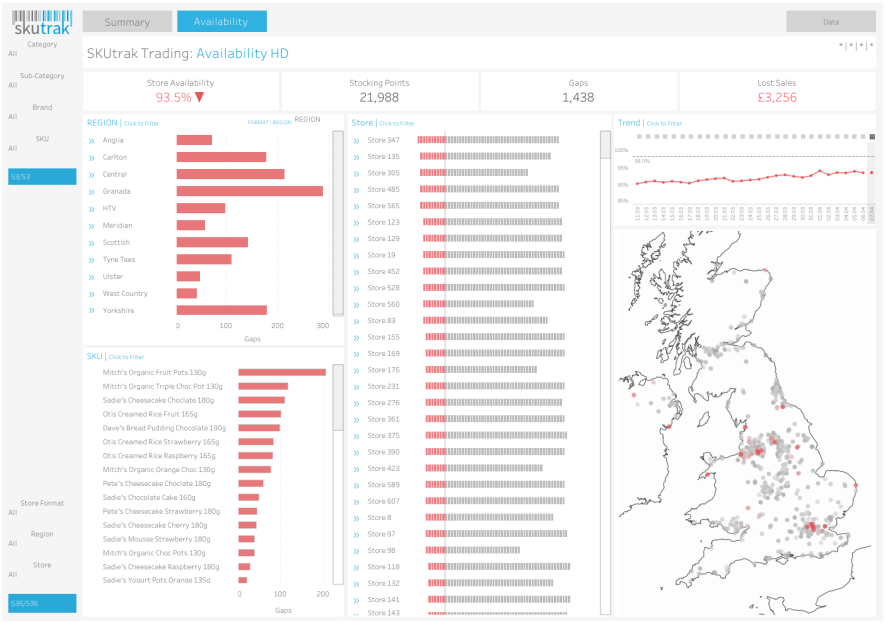
What is Store Level Product Availability and how does data visualization help address it?

Having the right product in the right store at the right time is one of the fundamentals of retailing. Having the best price or the best marketing strategy in the world doesn't matter if the customer cannot buy what they want, where they want, and when they want to buy it.

Availability is important—and costly—for brands and retailers. If a product is out of stock, the customer may substitute with another product, which means the brand will record a lost sale and the customer may permanently switch to the competing product. Customers can be fickle. If a customer's favorite products are not available, they may choose to do their shopping elsewhere, and the retailer loses out on not just the unavailable product, but the entire shopping basket of related goods.

Carry too much product, and brands and retailers will face product spoilage and increase costly waste, or tie up shelf space with under-performing products. Too much product also increases needed working capital, and our friends in the finance department won't appreciate that added investment.

The problem is that retailers generally look at product availability as a percentage. While percentage is a useful measure, it doesn't capture the size and magnitude of the problem—that is, missing distribution points. Resources are wasted fixing low distribution/low sales products with low availability, rather than focusing on what will have the largest economic impact.



SKU A

- Allocated to 200 stores and is Out of Stock in 40 Stores
- Availability percentage of 80% (200-40)/200.

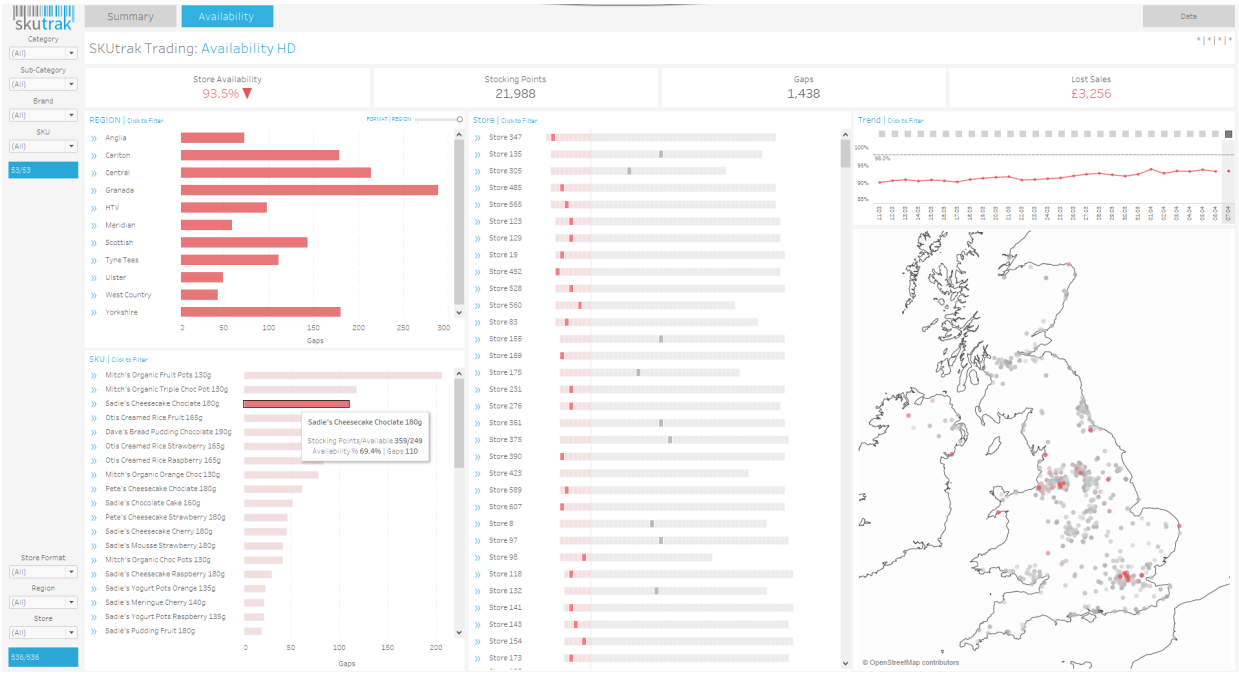
SKU B

- Allocated to 1000 stores and is Out of Stock in 100 Stores
- Availability percentage of 90% (1000-100)/1000.

SKU B is missing in over 2.5x as many stores (over twice as many customers can't purchase the product), but from a percentage measure looks to have better availability than SKU A.

The other problem is not being able to see which stores have the problem (and is there a pattern to the out of stock, such as stores serviced from a specific depot or warehouse).

> **Key takeaway: better insight into store level availability helps retail managers and brands**
>
> By analyzing the data visually and adding interactivity, both the retailer and brand (FMCG Supplier) can quickly find the biggest problems by product, geography, format, and store in a few clicks. The worst products, geographies, formats, and stores needing immediate attention are easy to identify.
>
> This visualization can be used every day to find and fix these issues. It's also useful for brand sales management to direct the field sales teams to stores with the biggest problems and maximize the return on investment. When this process is transformed via visual analytics, the customer, the brand, the retailer, and the financial departments all win.



Try it for yourself

2. Waste Manager

Partner: **Atheon Analytics**
Find it on **Tableau Public**

What is food waste and how does data visualization help identify it?

Waste on fresh products refers to products that are either thrown away or reduced in price due to expiring shelf life. Waste costs the retailer real money, and any savings on waste directly impacts profits. There is also corporate responsibility and sustainability pressure to reduce food waste. Properly identifying wasteful products and the cause of waste is a big problem for retailers and quick serve restaurants (QSRs). Due to the difficulty of consuming large text files in spreadsheets, retailers are forced to look at waste in aggregation, and not at store level, preventing retailers or QSRs from taking appropriate action.

Often, products with high waste are only problems in specific stores. Correcting these stores—either by changing product forecasts, facings, removing them from the menu or delisting altogether—can have a huge impact on waste reduction, but can also cause customer churn.

Let's look at High End Streak, a hypothetical example. High End Steak only comes in a pack-size of 12, with shelf life of 7 days, and sells at a retail price of £8.

100 Stores (with customer segmentation: high traffic, weekly shop) sell at full retail price 24 units a week, and have a zero theoretical waste:
- Sales value £9,600
- Waste (at retail) £0

60 Stores (with customer segmentation: low affluence, low traffic, weekly shop) sell at full price 8 units a week, and another 4 at 40% reduction due to expiring product dates.
- Sales value £4,900
- Waste (at retail) £760

80 Stores (with customer segmentation: high affluence, high traffic, convenience food-to-go) sell at full price only 4 units a week, and another 3 at 40% reduction and bin 5 (you can't send in half a pack).
- Sales value £4,700
- Waste (at retail) £4,000 (RTC £800 + Bin £3,200)

The total sales revenue for this organization was £19,200. Waste at retail) £4,760, and waste as a percentage of sales was 25%.

The organization has some choices to make:

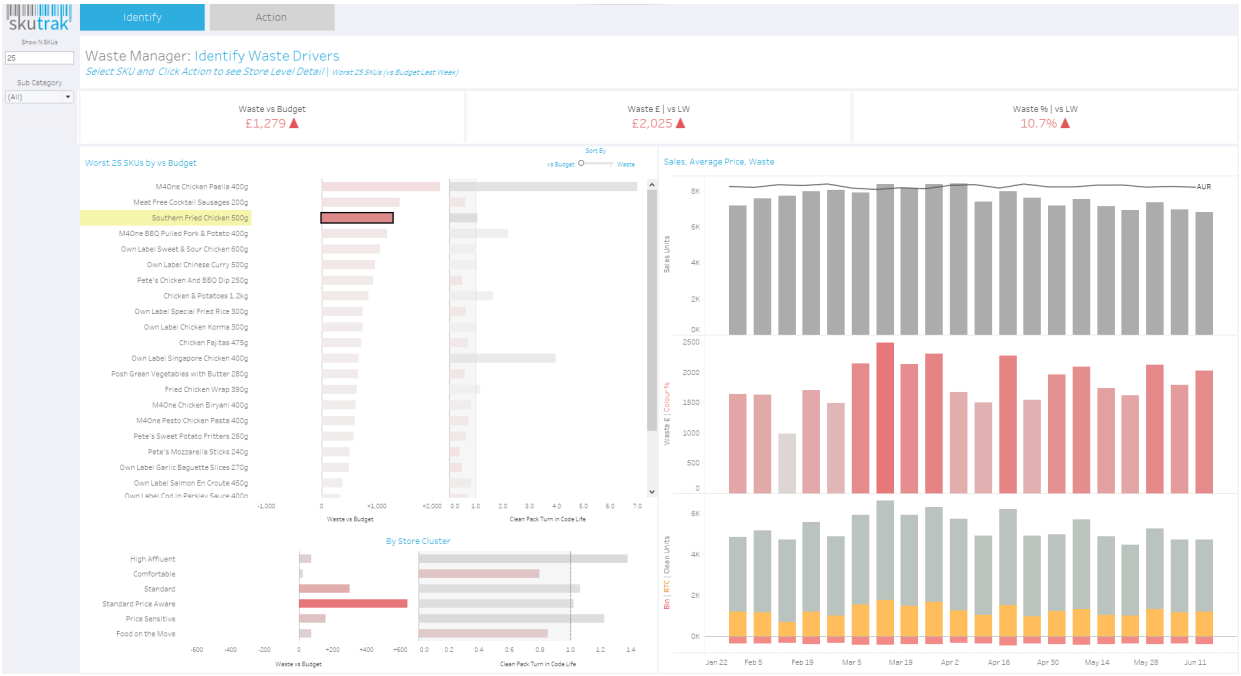
• Live with the waste to ensure customers will not be impacted as we hear that many customers love this product

• Remove the product from the product allocation—at the risk of annoying or losing customers (if we could see and make sense of the data at store level, we could remove it from the convenience stores selectively and maybe entice customers to choose ready-made meals, in the lowest traffic low/affluence stores)

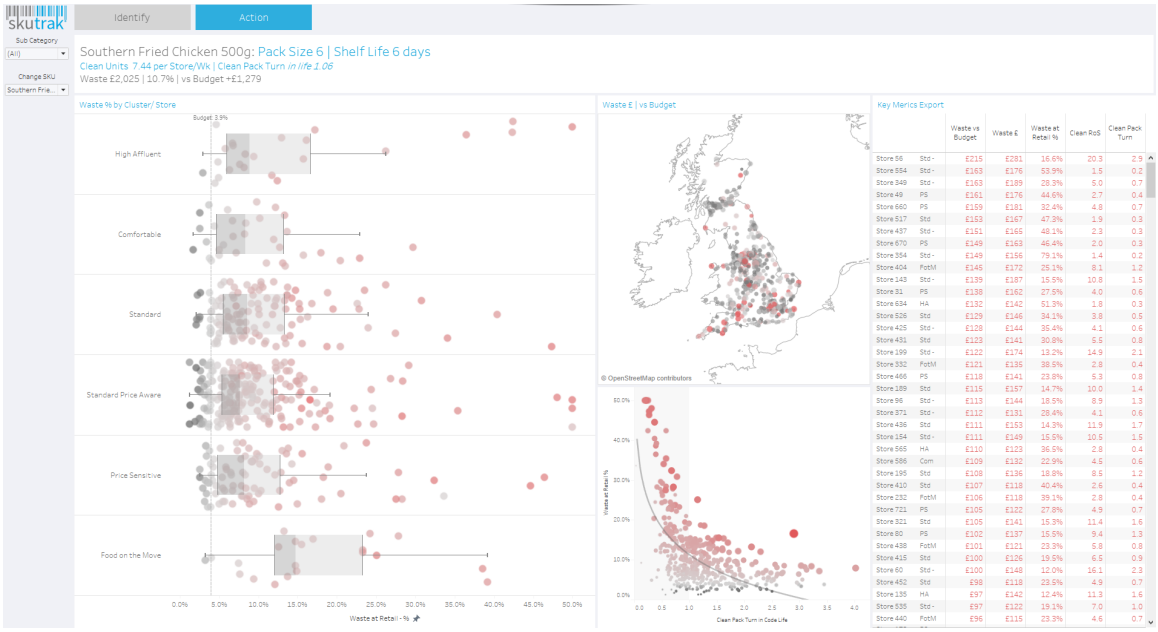
• Optimize the impact of waste by reducing the pack size to 6, so most convenience stores could sell through their inventory before the expiration date.

Key takeaway: visualizing waste provides powerful, actionable information

Waste Manager dashboard allows the retailer to quickly identify the products that are selling. Then by deconstructing the waste value into its constituents (clean sales, RTC sales, and binned) the cause of the problem can be seen. It also looks at the main cause of waste—sales velocity in some stores is too low to sell a whole pack at full retail within the product's life. In addition to seeing enterprise level data, the dashboard also shows break down by store cluster. This lets the retailer or QSR quickly identify whether the problem is the same in all stores, or in a particular store type. This is also shown over time, so it can be seen if the waste is a consistent problem, only when promoted, or related to some other event.



The Waste Manager dashboard then allows the user to interact with the data and see the clusters and stores that are driving waste. As stores are selected, the key information is shown in the table, which can then be exported or shared with the vendor or the merchandising team.



Try it for yourself

3. Retail Scorecard

Partner: **Interworks**
Find it **here**

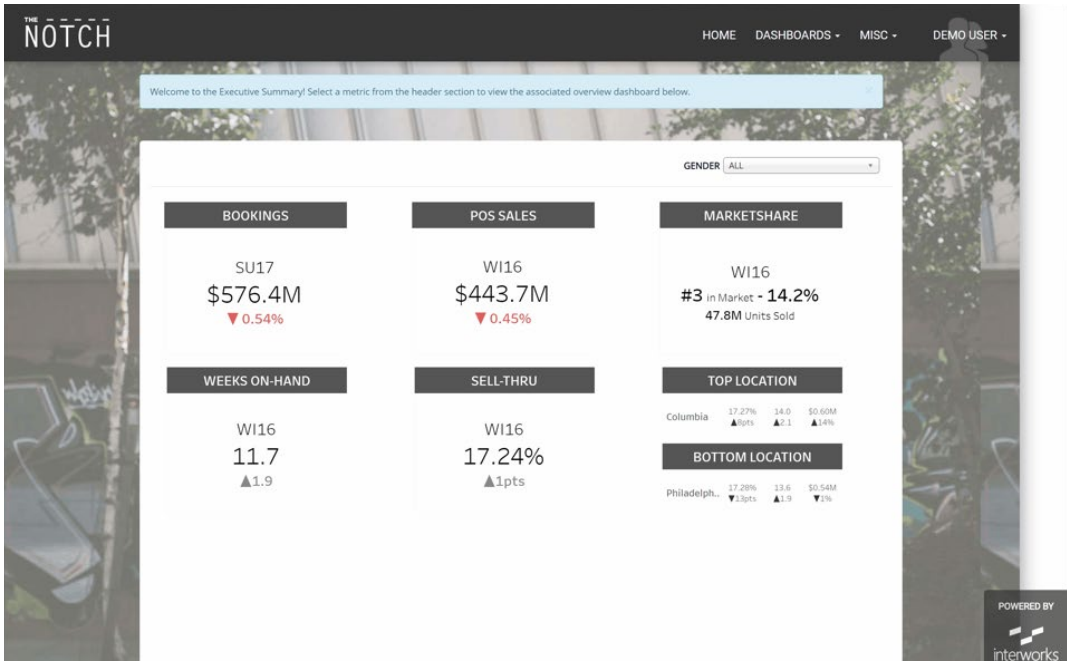
What is a retail scorecard and why is it important?

Retail data is typically siloed information that's difficult to access, analyze, or make actionable. With this retail scorecard key KPI screens are brought together in one place. Now, a general manager or VP can see their entire business across functions without having to navigate anywhere else. High level 'cards' contain snapshots of each KPI. Selecting a card accesses a drill-down dashboard specifically developed for that business function. A set of filters at the card level filters every card and every drill-down dashboard with one click.

Because time matters on the retail floor, JavaScript functions pre-load all the background dashboards to ensure each screen loads in less than one second. This is truly a next generation general manager or VP experience to view the entire business' health with detailed action items. This technique requires minimal expertise in Tableau Desktop and can be designed or maintained by any client team. Cards themselves are simple dashboards of their own.

Key takeaway: automated data analytics means more time to focus on the bottom line

This system is simple to build, applicable across all retail scenarios, built with speed-to-insight in mind and represents an achievement that most retailers strive to achieve.



Try it for yourself

4. Retail Executive Overview

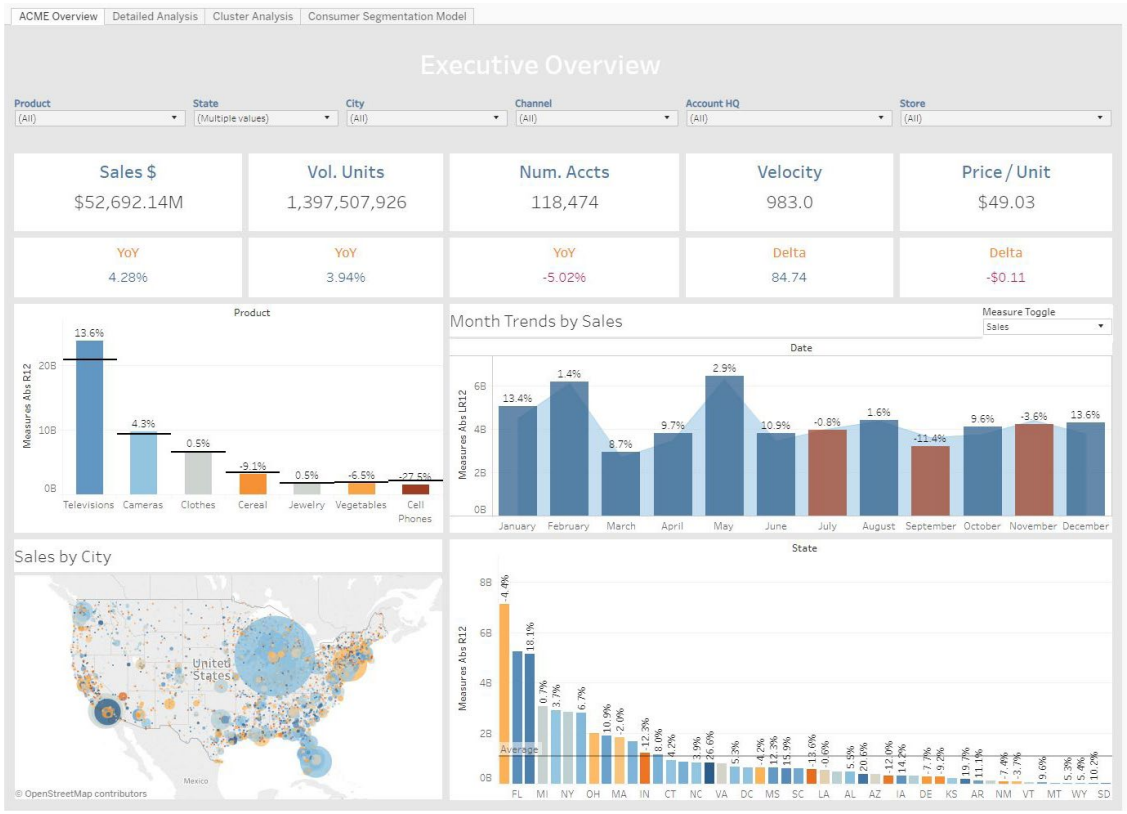
Partner: **Keyrus**
Find it on **Tableau Public**

What does a Retail Executive Overview capture?

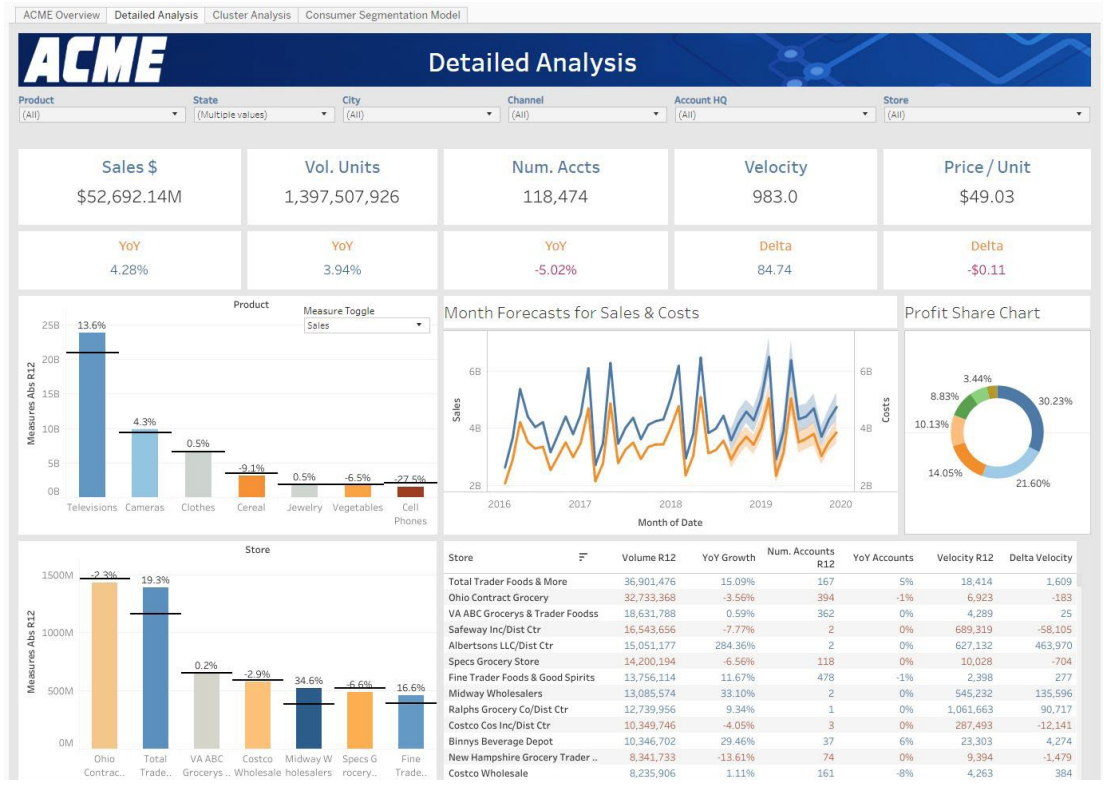
As we talk with our retail customers, many executives share excitement in being able to finally see a holistic view of their retail data. This dashboard shows high-level KPIs that convey the overall health of the organization.

Key takeaway: a high-level overview provides the KPIs executives need, at a glance

Executives can quickly see trends, year-over-year performance by product, and geographic sales performance.



Executives are also empowered to ask additional questions of the data. They can use filters and built-in performance forecasting for any combination of product, state, city, or store.



Try it for yourself

5. Weather Response Predicted Demand

Partner: **North Highland**
Find it on **Tableau Public**

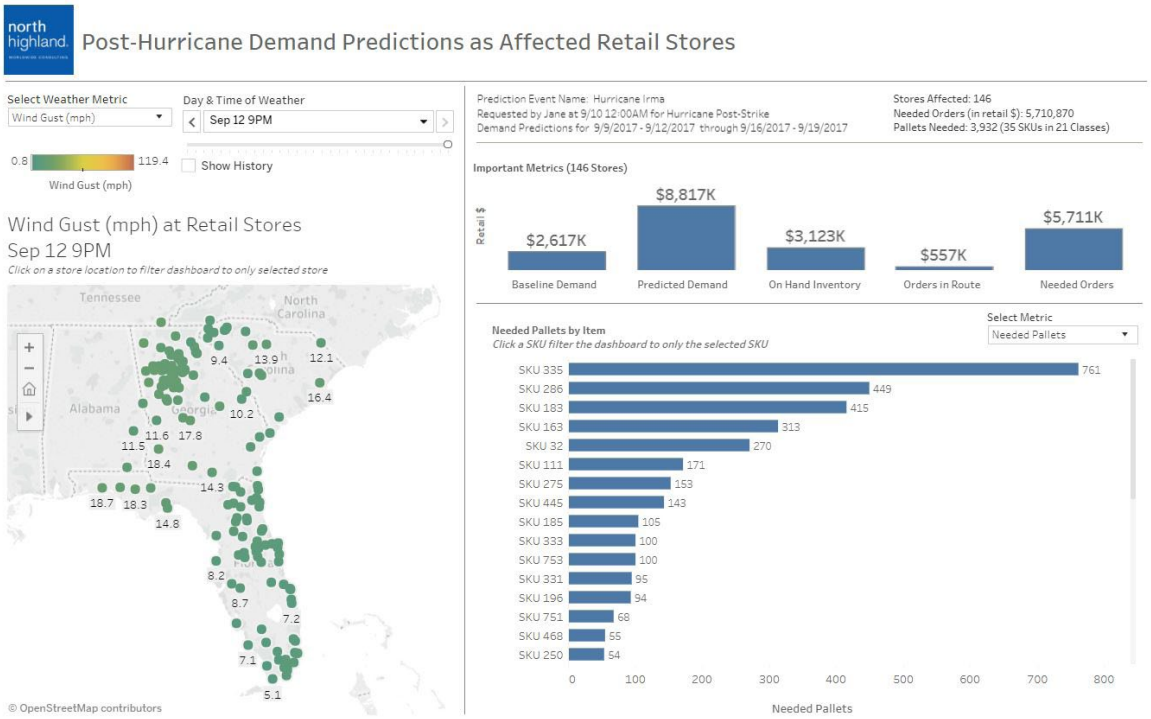
What is a Weather Response Predicted Demand dashboard and why is it important?

Not only do retailers have to deal with changing consumers, digitalization, and increased competition, but they also have to deal with extreme weather events.

In response to emergency weather events, North Highland's Predicted Demand dashboard enables mass retailers to predict extreme spikes in product demand, as well as the specific quantities, products, and locations that will be impacted. Insights from the dashboard enable retailers to funnel this new demand into their existing supply chain processes for quick execution in times of need.

Powered by machine learning, North Highland's Tableau dashboard analyzes and distills insights from over 73 billion historical NOAA weather data points, regional demographics and psychographics, retail sales history, and inventory history to equip retailers with a succinct snapshot of anticipated demand during emergency weather events including floods, hurricanes, and tornadoes.

> **Key takeaway: On-demand weather data can be used in tandem with inventory data, to anticipate demand**
>
> The dashboard offers a clear summary of baseline demand, predicted demand, on-hand inventory, orders in route, and the remaining gap in needed orders—enabling the informed allocation of inventory and resources to ensure that supplies are delivered to the right place at the right time.
>
> For large retailers with nationwide store footprints, the dashboard also enables users to drill in by region, product category, or timeframe to zero in on demand at the SKU, store, and day level—allowing decision makers to quickly mobilize on next steps and accelerate emergency response and recovery efforts, helping customers when they need it most.



Try it for yourself

6. Retail Store Heat Mapping

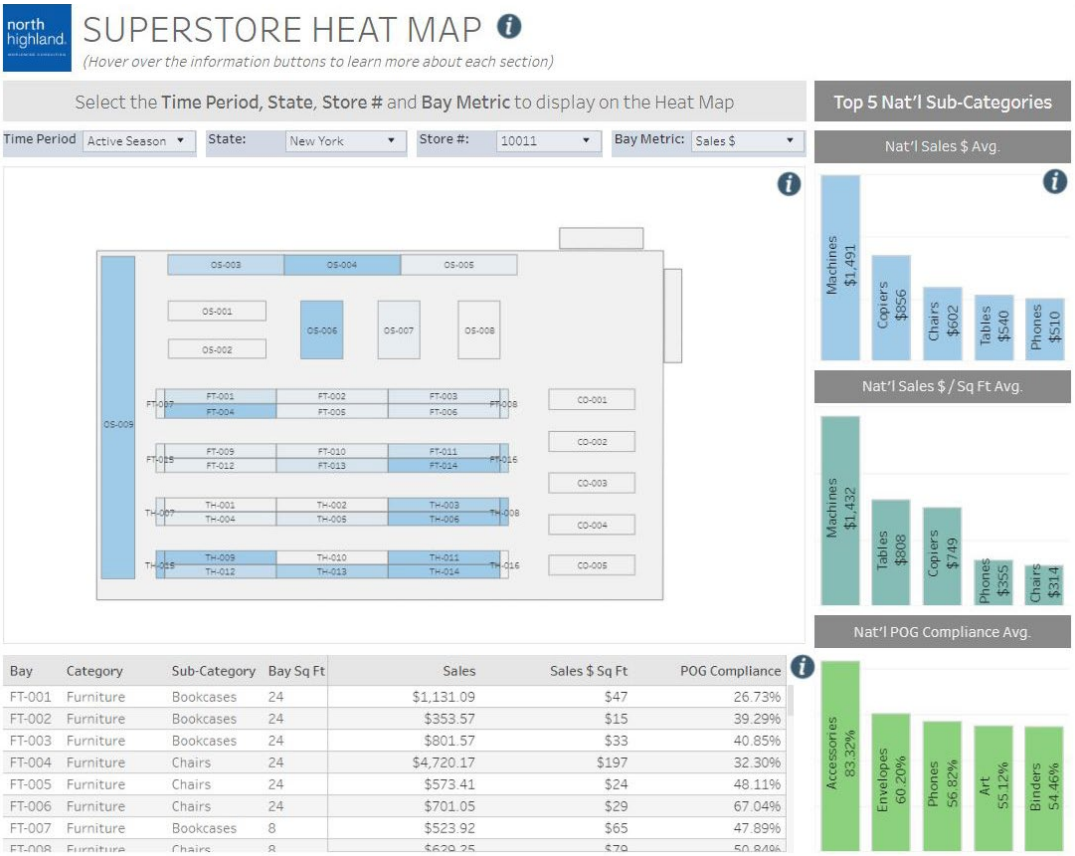
Partner: **North Highland**
Find it on **Tableau Public**

How can retail store heat mapping data improve sales?

Gone are the numbers only, grid-based sales flash reporting. Modern retailers are using visual analytics to gain greater insight within the physical space of brick and mortar stores.

Store heat-mapping analytics give retailers the ability to understand a wide variety of information about physical store layout and performance without having field teams physically travel to each store or consume valuable time from store staff.

> **Key takeaway: Heat maps allow retailers to optimize store layouts without the manual guesswork of the past**
>
> Leveraging industry-standard mapping technology, the solution reads store CAD maps and overlays analysis such as sales per square foot, product locator services, planogram compliance, seasonal reset location and performance, shelf/fixture profitability, and sales velocity.



Try it for yourself

7. Regional Manager Dashboard

Partner: **Automated Insights**

Find it **here**

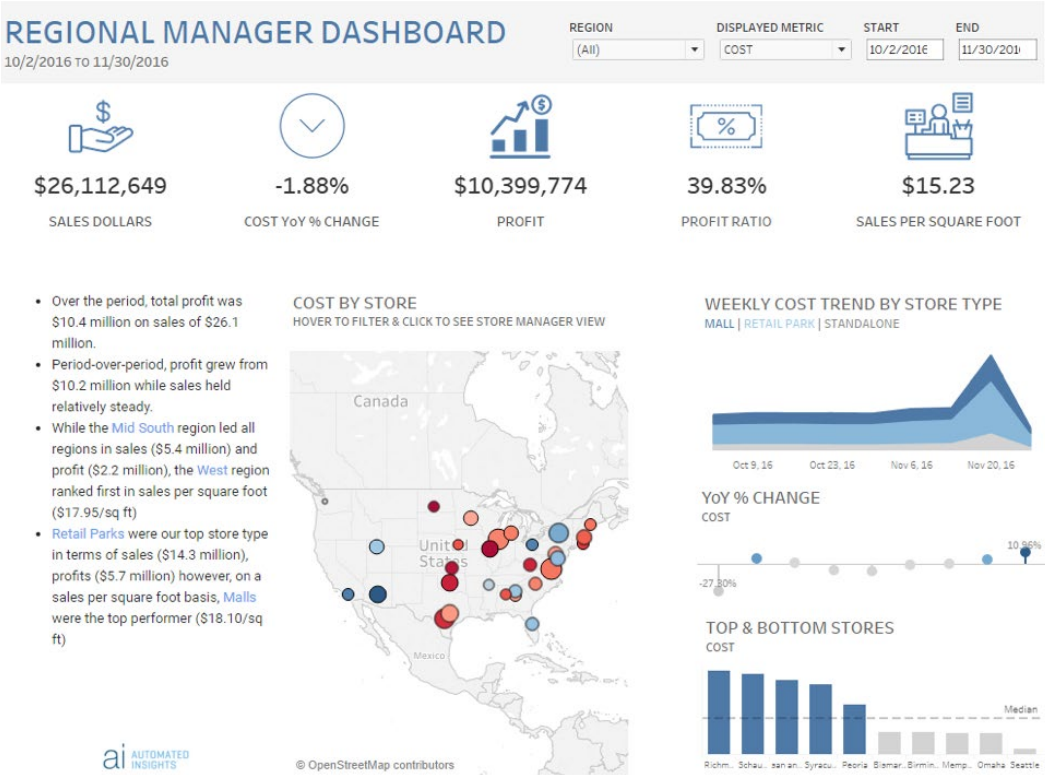
How does a Regional Manager dashboard help drive better decision-making?

Analytics have become one of the most powerful tools for retailers to derive operational insights from their point of sale databases. Using Tableau, analysts can build guided visual analytics that are quick and easy to digest and broadly distribute data across massive retail networks. Adding written analytics is a great way to explain the context behind visualizations and ensure the way information is consumed is consistent with the strategy and goals set forth by leadership. Combining visual analytics with a written explanation reduces the training required to interpret the information.

Whether it's sharing inventory analytics with your supply chain or distributing sales KPIs to brick-and-mortar locations, there's a lot to gain from creating a data-driven culture. This dashboard gives retailers the big picture of nationwide sales and merchandising by not only showing what's going on, but also describing the broader context of why it's happening or how it's relevant. The combination of both visual and written analytics ensures dashboard consumers are able to explore data in a meaningful way to identify trends, events, and KPIs efficiently without misinterpretation.

Key takeaway: by embedding automated written analysis, regional managers can receive role-based insights that provide complete context and drive action.

Regional managers can see their store performance in context of the greater enterprise and are able to drill into underperforming stores and focus on the exact departments and products that need attention.



Try it for yourself

8. Store Manager Dashboard

Partner: **Narrative Science**
Find it **here**

What is a Store Manager Dashboard and why is it important?

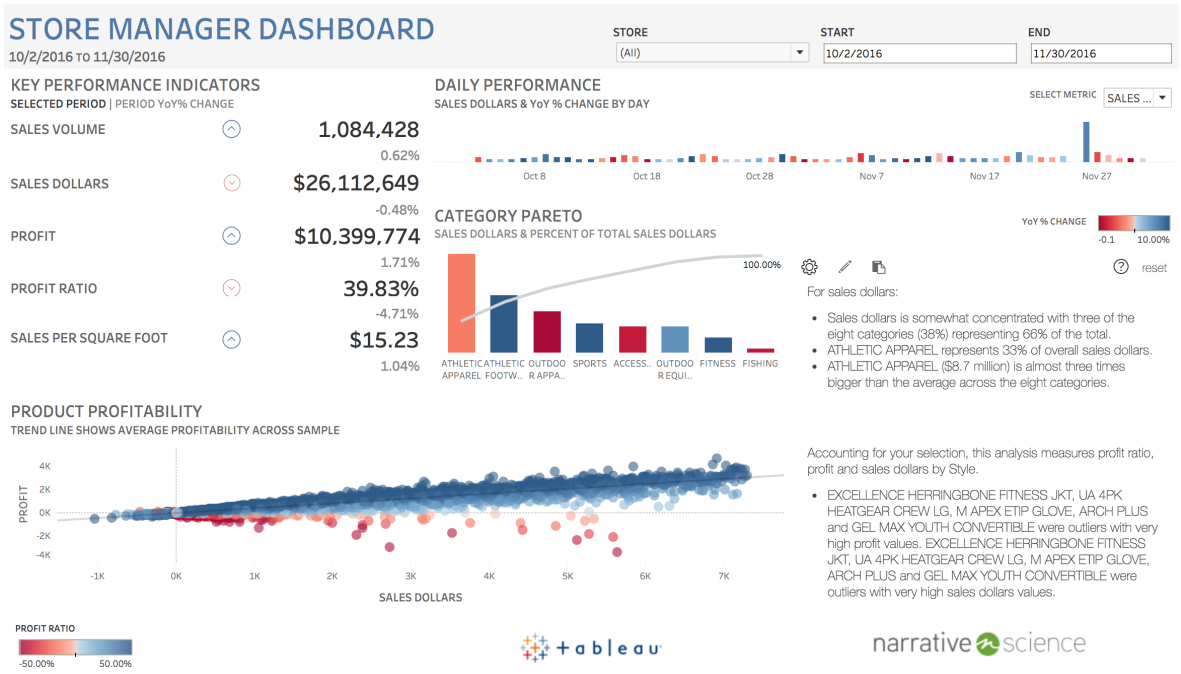
Store managers struggle with legacy BI reports that are tens of thousands of rows of data in Excel or non-interactive PDF reports. With store labor constantly being evaluated and reduced, operational efficiency has to be a focus. Getting from problem to analysis to action in today's fast-paced retail environment has to happen in seconds—not minutes.

Narratives for Tableau instantly provides plain-English explanations of charts and graphs, making it possible for store managers to identify and communicate key insights faster. This natively-built Tableau dashboard extension provides a seamless experience, analyst-quality insights, and the ability to consume a narrative based on a store manager's interaction with the dashboard.

Key takeaway: store managers need actionable data—and they can obtain it quickly with this dashboard

In the example below, the Store Manager of fictional store Gravitas Sporting Goods located in Oklahoma City can go to their dashboard to see the primary driver of the decline in sales was due to weak performance in the Athletic Apparel category. Store managers see the visual analysis, read the narrative, and have confidence they have the right information to correct the issue.

From there, they can dig into the product performance within Athletic Apparel and come up with a plan for underperforming products.



Try it for yourself

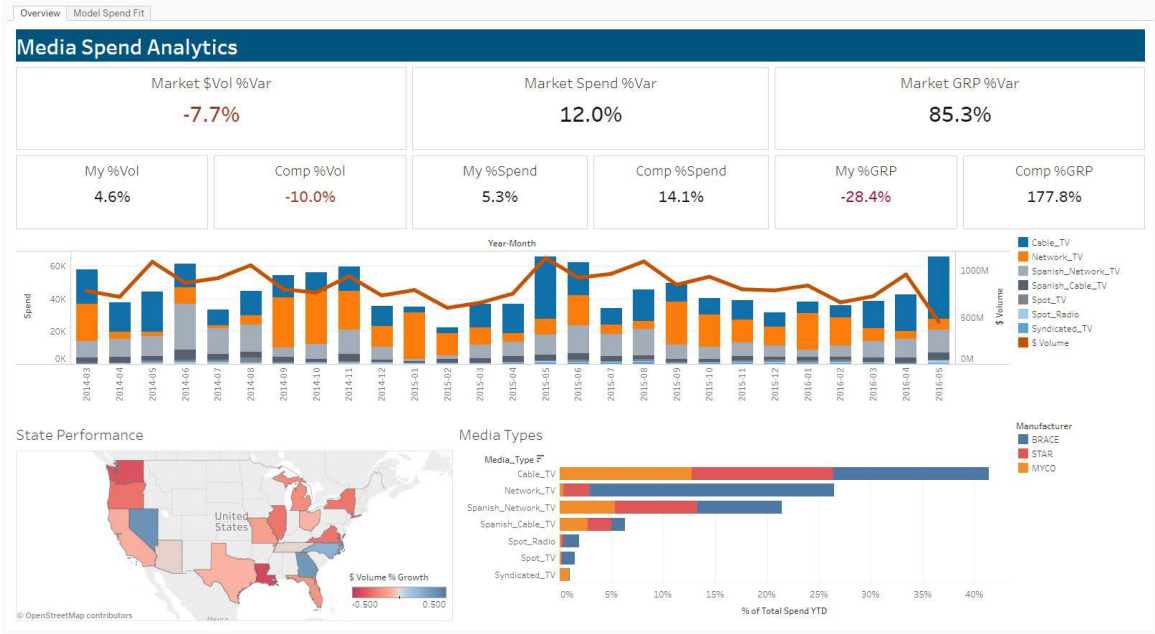
1. Marketing Mix Models and ROI Engines

Partner: **Keyrus**
Find it on **Tableau Public**

What insights do Marketing Mix Models and ROI Engines provide?

Frequently, marketing performance measurement and sales activity are analyzed separately, because it's hard for many organizations to combine them. But measuring marketing spend and actual sales performance together can help identify marketing campaigns that have direct sales impact. Applying machine learning to the underlying combined datasets helps optimize the marketing mix and achieve maximum ROI.

> **Key takeaway: combing two distinct, and often siloed data sources, can help to identify marketing and sales effectiveness.**
>
> Retailers can use machine learning to maximize marketing mix to drive marketing ROI, ensuring marketing campaigns have a tangible impact on increasing revenues.



Try it for yourself

2. Marketing Consumer Segmentation

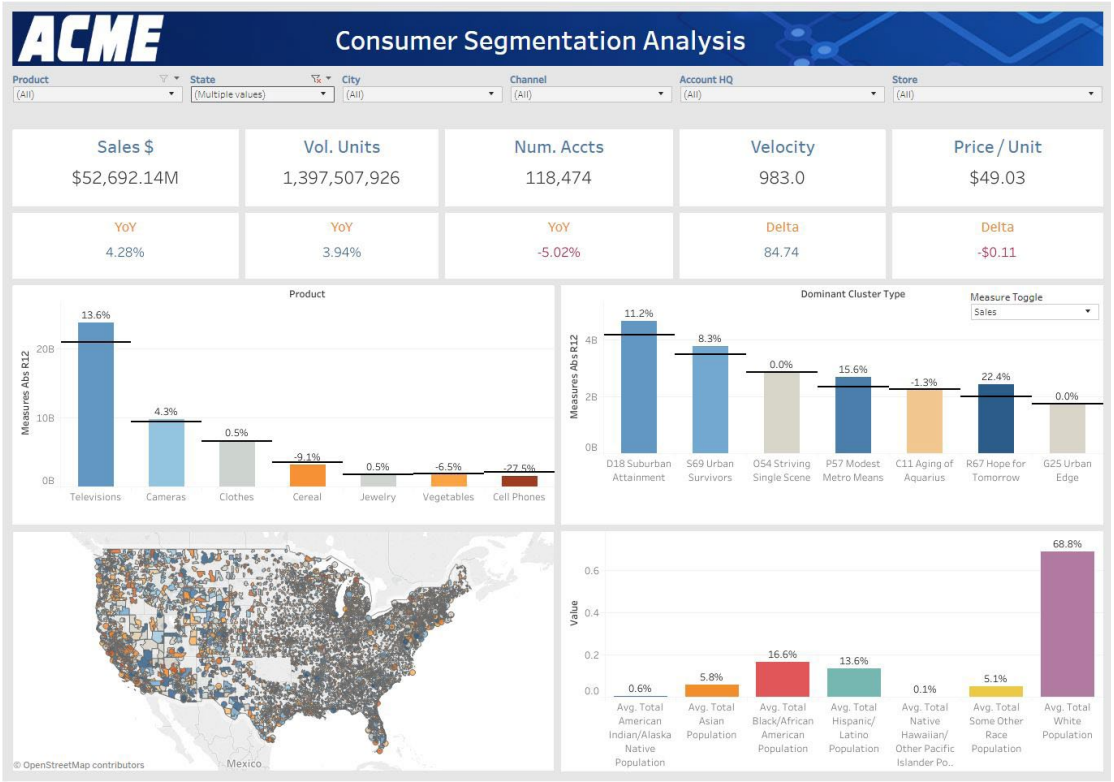
Partner: **Keyrus**
Find it on **Tableau Public**

What is Marketing Consumer Segmentation and why is it important?

With razor thin margins, every dollar of retail spending needs to be analyzed, scrutinized, justified, and optimized. Consumer brands need to know who their customer is, what they like, what they want, and where they are. With more pressure on marketing budgets, it is essential to optimize spending and know what consumer segments need to be prioritized.

Key takeaway: granular insight improves the quality of decision making

Marketing executives can measure KPIs by designated market area (DMA), sales impact, and dominant cluster type. With hyper focus and no guesswork, marketing can maximize every spend, expand key customer segments, or winning back key consumers whose loyalty may be wavering.



Try it for yourself

3. Digital Content Optimization

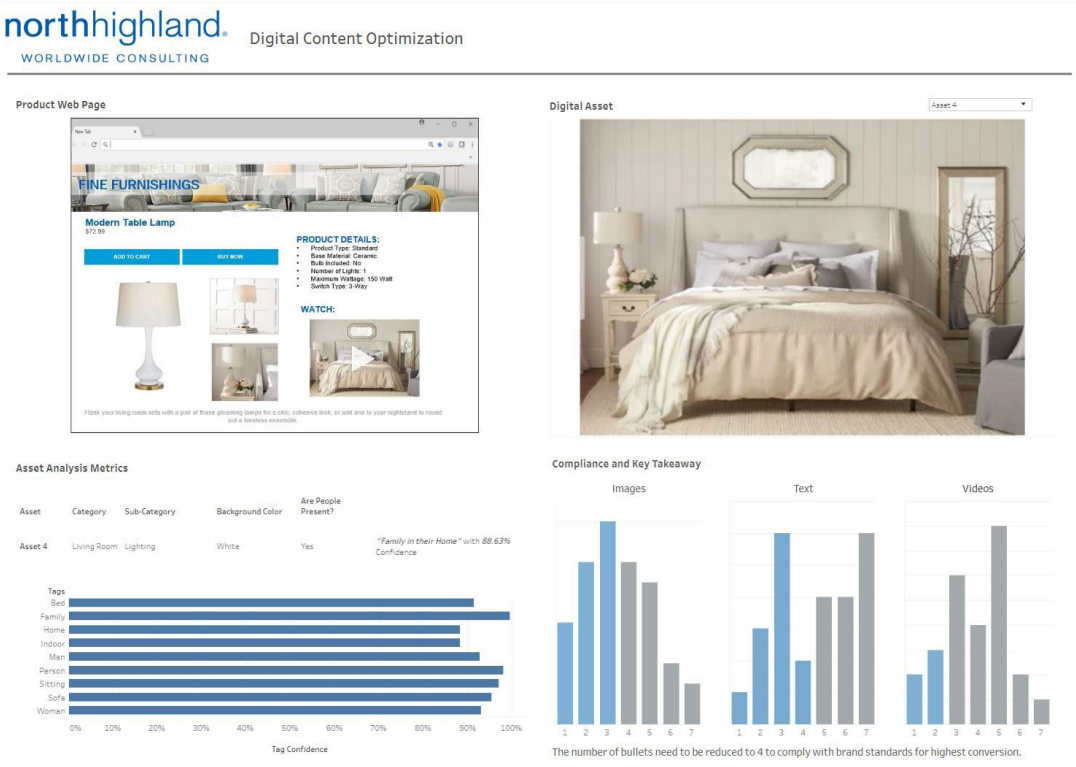
Partner: **North Highland**
Find it on **Tableau Public**

What is a Digital Content Optimization dashboard?

Consumers shop online in ever-growing numbers, so in order to keep their attention—and retain their loyalty—content has to be optimized for speed, engagement, and conversion. North Highland's Digital Content Optimization dashboard provides optimized, tactical recommendations that drive e-commerce performance, grounded in insights from the visual analysis of specific digital assets, including individual text, video, and image components.

This Tableau-powered dashboard determines the optimal combination of assets that results in the highest conversion of visitors to buyers, considering each digital asset on a specific product information page across all products in a product category. The dashboard leverages what we know about the assets already: image attributes such as type of image and count of images, text attributes such as number of words, bullets, paragraphs and mentions of specific topics, and video attributes such as length and count. It also supplements this information with what we can learn from cognitive recognition tools—such as objects and actions within images, the presence of people, language, and sentiment within videos—to fully understand what the customer is responding to. Once it's understood how these factors relate to sales, the solution recommends the optimal mix of digital assets to maximize conversion.

> **Key takeaway: understanding—and responding to—your online customers' habits and preferences is essential to driving revenue**
>
> The opportunity for digital optimization is large for retailers. One mass retailer realized 100x ROI from increased purchase conversion in the first year following use of North Highland's solution and dashboard. Another company realized over $27M revenue opportunity for one of its brands, leading to actionable improvement recommendations that will be applied to other brands.
>
> Ultimately, North Highland's dashboard solution helps leaders right-size their asset capabilities and marketing spend to help inform and define future state action plans against recommendations that align to content and brand standards.



Try it for yourself

4. Voice of the Customer

Partner: **VoiceBase**
Find it on **Tableau Public**

What is Voice of the Customer and why is it important?

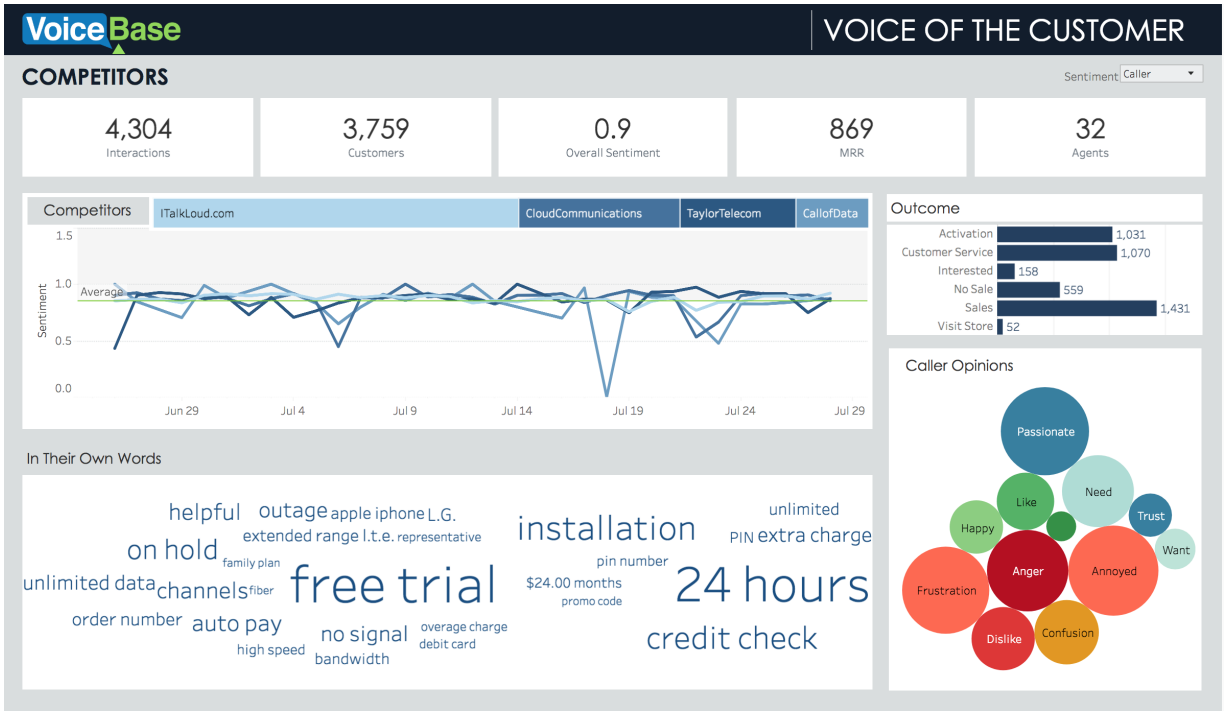
In today's digital marketplace, retailers must gain a better understanding of customers' needs, wants, and concerns by continually gathering, analyzing, and acting on customer feedback. And listening to the customer is more important than ever: according to a Walker study, by the year 2020 customer experience will overtake price and product as the key brand differentiator. Voice of the Customer (VoC) dashboards help retailers deliver enhanced customer experiences, engage employees and drive business change. The customer voice can be loudly heard in emails, support chats, voice calls and sales interactions, chatbots and messaging applications, tweets, social platforms, video, and surveys.

One primary channel for Voice of the Customer analysis is speech analytics, which transcribes and analyzes millions of customer calls to discover actionable insights, close more sales, and improve business performance. Speech analytics has historically been limited to the call center.

VoiceBase's open data architecture allows departments outside of the call center to harness rich insights available through voice calls and create true 'Voice of the Customer' dashboards with Tableau.

Key takeaway: VoC can help you understand your customer and how they view your products, services, and brand

The dashboard below shows the VoC Competitor analysis using VoiceBase transcribed calls to improve competitive positioning and enhance customer experience. You'll see KPIs of how customers describe competitive brands, including emotions, call volume trends and organic words used with sales and service representatives. This allows you to be able to quickly see how these trends affect the call outcome.



Try it for yourself

Summary

Today's retailers need an edge in order to thrive—not merely survive—in a turbulent landscape where consumer behavior is constantly changing, and customer-centric experiences are essential. To make more informed decisions, savvy retailers are increasingly embracing a data-driven approach.

Using data-driven dashboards, it's possible for retailers to see and understand their data in new ways, and build strategic and competitive advantage with granular, actionable insights. As you've seen in the dashboard examples in this paper, the width and breadth of what retailers can do with visual analytics is impressive: from assessing store level availability to optimizing store layouts, it's clear that having the right data at the right time is key to maximizing profitability and scaling for the future.

Connect with our Partners

Many thanks to our generous partners for sharing these dashboards with us. Please reach out to them to learn more about how you can implement Tableau dashboards like these and harness the power of your data.

Partner	Contact	Email	Phone
Atheon Analytics	Simon Runc	simon.runc@atheon.co.uk	+44-08444-145-501
Automated Insights	Peter Benson	peter@automatedinsights.com	919-824-7671
Interworks	Derrick Austin	derrick.austin@interworks.com	405-533-1039
Keyrus	Razvan Nistor	Razvan.nistor@keyrus.us	646-664-4872
Narrative Science	Shawn Parks	sparks@narrativescience.com	312-477-0590
North Highland	Dan Kopp	dan.kopp@northhighland.com	770-314-9418
VoiceBase	Emily Blazensky	emily@voicebase.com	408 702-7160

Resources

7 Tips and Tricks from Dashboard Experts

Tableau Retail and Wholesale Analytics solutions page

Top 10 Retail Dashboards for Better Performance

About Tableau

Over 80% of the top 100 retailers and over 7,000 retail and consumer goods companies around the world trust Tableau to help them understand their data and create actionable insights. On the Tableau platform, it's easy to explore your data, build dashboards, and perform ad hoc analyses in just a few clicks.

Download a free trial and experience the power of Tableau for yourself.

tableau

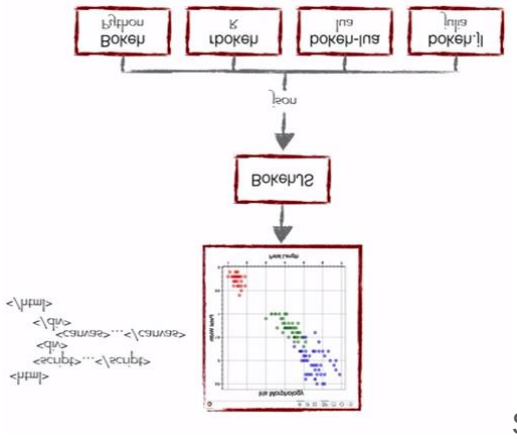
Interactive Data Visualization using Bokeh (in Python)

Introduction

Recently, I was going through a video from SciPy 2015 conference, "Building Python Data Apps with Blaze and Bokeh", recently held at Austin, Texas, USA. I couldn't stop thinking about the power these two libraries provide to data scientists using Python across the globe. In this article, I will introduce you to the world of possibilities in data visualization using Bokeh and why I think this is a must learn/use library for every data scientist out there.

What is Bokeh?

Bokeh is a Python library for interactive visualization that targets web browsers for representation. This is the core difference between Bokeh and other visualization libraries. Look at the snapshot below, which explains the process flow of how Bokeh helps to present data to a web browser.



Source: Continuum Analytics

As you can see, Bokeh has multiple language bindings (Python, R, lua and Julia). These bindings produce a JSON file, which works as an input for BokehJS (a Javascript library), which in turn presents data to the modern web browsers.

Bokeh can produce elegant and interactive visualization like D3.js with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone who would like to quickly and easily create interactive plots, dashboards, and data applications.

What does Bokeh offer to a data scientist like me?

I started my data science journey as a BI professional and then worked my way through predictive modeling, data science and machine learning. I have primarily relied on tools like

QlikView & Tableau for data visualization and SAS & Python for predictive analytics & data science. I had near zero experience of using JavaScript.

So, for all my data products or ideas, I had to either outsource the work or had to pitch my ideas through wire-frames, both of which are not ideal for building quick prototypes. Now, with Bokeh, I can continue to work in Python ecosystem, but still create these prototypes quickly.

Benefits of Bokeh:

- Bokeh allows you to build complex statistical plots quickly and through simple commands
- Bokeh provides you output in various medium like html, notebook and server
- We can also embed Bokeh visualization to flask and django app
- Bokeh can transform visualization written in other libraries like matplotlib, seaborn, ggplot
- Bokeh has flexibility for applying interaction, layouts and different styling option to visualization

Challenges with Bokeh:

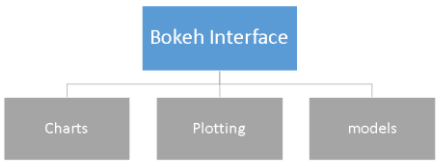
- Like with any upcoming open source library, Bokeh is undergoing a lot of development. So, the code you write today may not be entirely reusable in future.
- It has relatively less visualization options, when compared to D3.js. Hence, it is unlikely in near future that it will challenge D3.js for its crown.

Given the benefits and the challenges, it is currently ideal to rapidly develop prototypes. However, if you want to create something for production environment, D3.js might still be your best bet.

To install Bokeh, please follow the instruction given here.

Visualization with Bokeh

Bokeh offers both powerful and flexible features which imparts simplicity and highly advanced customization. It provides multiple visualization interfaces to the user as shown below:



- **Charts:** a *high-level* interface that is used to build complex statistical plots as quickly and in a simplistic manner.
- **Plotting:** an *intermediate-level* interface that is centered around composing visual glyphs.

- **Models:** a *low-level* interface that provides the maximum flexibility to application developers.

In this article, we will look at first two interfaces charts & plotting only. We will discuss models and other advance feature of this library in next post.

Charts

As mentioned above, it is a high level interface used to present information in standard visualization form. These forms include box plot, bar chart, area plot, heat map, donut chart and many others. You can generate these plots just by passing data frames, numpy arrays and dictionaries.

Let's look at the common methodology to create a chart:

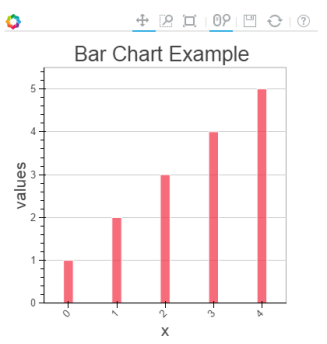
1. Import the library and functions/ methods
2. Prepare the data
3. Set the output mode (Notebook, Web Browser or Server)
4. Create chart with styling option (if required)
5. Visualize the chart

To understand these steps better, let me demonstrate these steps using example below:

Charts Example-1: Create a bar chart and visualize it on web browser using Bokeh

We will follow above listed steps to create a chart:

```
#Import library
from bokeh.charts import Bar, output_file, show #use output_notebook to visualize it
in notebook
# prepare data (dummy data)
data = {"y": [1, 2, 3, 4, 5]}
# Output to Line.HTML
output_file("lines.html", title="line plot example") #put output_notebook() for noteb
ook
# create a new line chat with a title and axis labels
p = Bar(data, title="Line Chart Example", xlabel='x', ylabel='values', width=400, hei
ght=400)
# show the results
show(p)
```



In the chart above, you can see the tools at the top (zoom, resize, reset, wheel zoom) and these tools allows you to interact with chart. You can also look at the multiple chart options (legend, xlabel, ylabel, xgrid, width, height and many other) and various example of charts here.

Chart Example-2: Compare the distribution of sepal length and petal length of IRIS data set using Box plot on notebook

To create this visualization, firstly, I'll import the iris data set using sklearn library. Then, follow the steps as discussed above to visualize chart in ipython notebook.

```
#IRIS Data Set
from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
df=pd.DataFrame(iris.data)
df.columns=['petal_width','petal_length','sepal_width','sepal_length']
#Import library
from bokeh.charts import BoxPlot, output_notebook, show
data=df[['petal_length','sepal_length']]
# Output to Notebook
output_notebook()
# create a new line chat with a title and axis labels
p = BoxPlot(data, width=400, height=400)
# show the results
show(p)
```

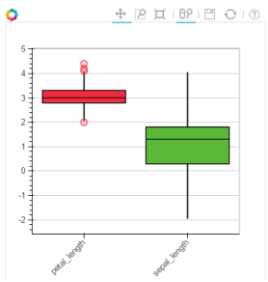


Chart Example-3: Create a line plot to bokeh server

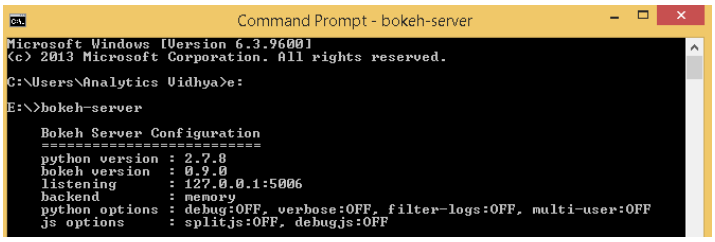
Prior to plotting visualization to Bokeh server, you need to run it.

If you are using a conda package, you can use run command **bokeh-server** from any directory using command. Else, **python ./bokeh-server** command should work in general. For more detail on this please refer this link "Deploying Bokeh Server".

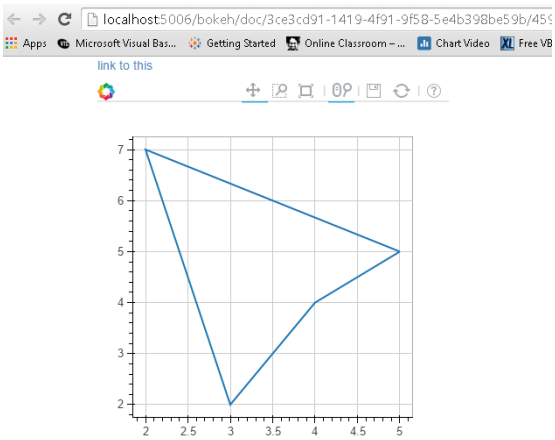
There are multiple benefits of Plotting visualization on Bokeh server:

- Plots can be published to larger audience
- Visualize large data set interactively
- Streaming data to automatically updating plots
- Building dashboards and apps

To start plotting on Bokeh server, I have executed the command **bokeh-server** to initialize it followed by the commands used for visualization.




```
from bokeh.plotting import figure, output_server, show
output_server("line")
p = figure(plot_width=400, plot_height=400)
# add a line renderer
p.line([5, 2, 3, 4, 5], [5, 7, 2, 4, 5], line_width=2)
show(p)
```



Plotting

Plotting is an *intermediate-level* interface that is centered around composing visual glyphs. Here, you create a visualization by combining various visual elements (dot, circles, line, patch & many others) and tools (hover tool, zoom, Save, reset and others).

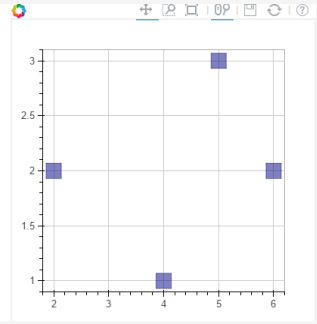
Bokeh plots created using the bokeh.plotting interface comes with a default set of tools and visual styles. For plotting, follow the below steps:

1. Import library, methods or functions
2. Select the output mode (notebook, web browser, server)
3. Activate a figure (similar like matplotlib)
4. Perform subsequent plotting operations, it will affect the generated figure.
5. Visualize it

To understand these steps better, let me demonstrate these steps using examples below:

Plot Example-1: Create a scatter square mark on XY frame of notebook

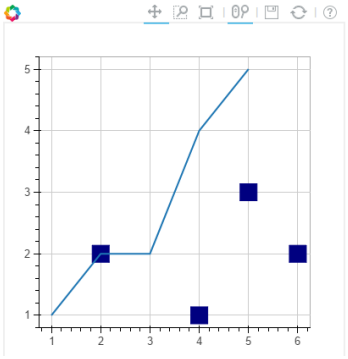
```
from bokeh.plotting import figure, output_notebook, show

# output to notebook
output_notebook()
p = figure(plot_width=400, plot_height=400)
# add square with a size, color, and alpha
p.square([2, 5, 6, 4], [2, 3, 2, 1, 2], size=20, color="navy")
# show the results
show(p)
```



Similarly, you can create various other plots like line, wedges & arc, ovals, images, patches and many others, refer this link to see various example.

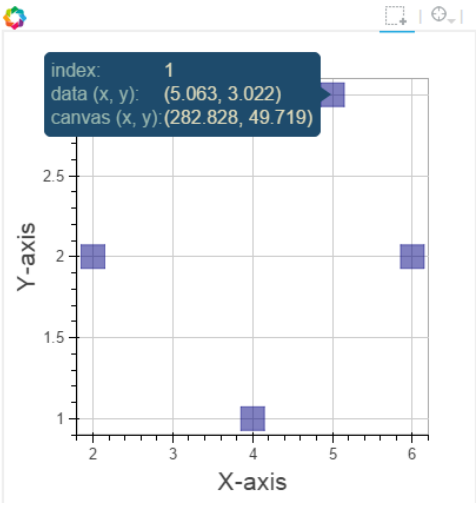
Plot Example-2: Combine two visual elements in a plot

```
from bokeh.plotting import figure, output_notebook, show
# output to notebook
output_notebook()
p = figure(plot_width=400, plot_height=400)
# add square with a size, color, and alpha
p.square([2, 5, 6, 4], [2, 3, 2, 1, 2], size=20, color="navy")
p.line([1, 2, 3, 4, 5], [1, 2, 2, 4, 5], line_width=2) #added a line plot to existing
figure
# show the results
show(p)
```



Plot Example-3: Add a hover tool and axis labels to above plot

```
from bokeh.plotting import figure, output_notebook, show
from bokeh.models import HoverTool, BoxSelectTool #For enabling tools
# output to notebook
output_notebook()
#Add tools
TOOLS = [BoxSelectTool(), HoverTool()]
p = figure(plot_width=400, plot_height=400, tools=TOOLS)
# add a square with a size, color, and alpha
p.square([2, 5, 6, 4], [2, 3, 2, 1, 2], size=20, color="navy", alpha=0.5)
#Visual Elements
p.xaxis.axis_label = "X-axis"
p.yaxis.axis_label = "Y-axis"
# show the results
show(p)
```



Plot Example-4: Plot map of India using latitude and longitude data for boundaries

Note: I have data for polygon of latitude and longitude for boundaries of India in a csv format. I will use that for plotting.

Here, we will go with patch plotting, let's look at the commands below:

```
#Import libraries
import pandas as pd
from bokeh.plotting import figure, show, output_notebook
#Import Latitude and lanogitude co-ordinates
India=pd.read_csv('E:/India.csv')
del India['ID']
India.index=['IN0','IN1','IN2','IN3','IN4','IN5']
#Convert string values to float as co-ordinates in dataframe are string
for j in range(0,len(India)):
 a = India['lats'][j]
 India['lats'][j] = [float(i) for i in a[1:len(a)-1].split(",")]
for j in range(0,len(India)):
 a = India['lons'][j]
 India['lons'][j] = [float(i) for i in a[1:len(a)-1].split(",")]
# Output option
output_notebook()
# Create your plot
p = figure(plot_height=400, plot_width=400, toolbar_location="right",x_axis_type=None
, y_axis_type=None)
p.patches(xs=India['lons'], ys=India['lats'], fill_color="white",line_color="black",
line_width=0.5)
#Visualize your chart
show(p)
```

Python For Data Science *Cheat Sheet*

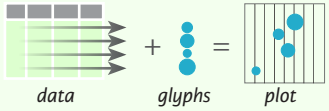
Bokeh

Learn Bokeh Interactively at www.DataCamp.com, taught by Bryan Van de Ven, core contributor

Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data:

Python lists, NumPy arrays, Pandas DataFrames and other sequences of values

2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]                          Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",      Step 2
              x_axis_label='x',
              y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)   Step 3
>>> output_file("lines.html")                    Step 4
>>> show(p)                                      Step 5
```

1 Data

Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'],
                                [32.4,4,66, 'Asia'],
                                [21.4,4,109, 'Europe']]),
                      columns=['mpg','cyl', 'hp', 'origin'],
                      index=['Toyota', 'Fiat', 'Volvo'])
```

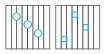
```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
                x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

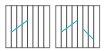
3 Renderers & Visual Customizations

Glyphs



Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
              color='blue', size=1)
```



Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Customized Glyphs

Also see Data

Selection and Non-Selection Glyphs

```
>>> p = figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

Hover Glyphs

```
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```

Colormapping

```
>>> color_mapper = CategoricalColorMapper(
                   factors=['US', 'Asia', 'Europe'],
                   palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
              color=dict(field='origin',
                         transform=color_mapper),
                         legend='Origin'))
```

Rows & Columns Layout

Rows

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2,p3)
```

Columns

```
>>> from bokeh.layouts import columns
>>> layout = column(p1,p2,p3)
```

Nesting Rows & Columns

```
>>>layout = row(column(p1,p2), p3)
```

Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Linked Plots

Also see Data

Linked Axes

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

Linked Brushing

```
>>> p4 = figure(plot_width = 100, tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200, tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

Legends

Legend Location

Inside Plot Area

```
>>> p.legend.location = 'bottom_left'
```

Outside Plot Area

```
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" , [p1, r1]),("Two" , [r2])], location=(0, -30))
>>> p.add_layout(legend, 'right')
```

Legend Orientation

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

4 Output

Output to HTML File

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Notebook Output

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

Embedding

Standalone HTML

```
>>> from bokeh.embed import file_html
>>> html = file_html(p, CDN, "my_plot")
```

Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

5 Show or Save Your Plots

```
>>> show(p1)
>>> show(layout)
```

```
>>> save(p1)
>>> save(layout)
```

Statistical Charts With Bokeh

Also see Data

Bokeh's high-level `bokeh.charts` interface is ideal for quickly creating statistical charts

Bar Chart

```
>>> from bokeh.charts import Bar
>>> p = Bar(df, stacked=True, palette=['red','blue'])
```

Box Plot

```
>>> from bokeh.charts import BoxPlot
>>> p = BoxPlot(df, values='vals', label='cyl',
                legend='bottom_right')
```

Histogram

```
>>> from bokeh.charts import Histogram
>>> p = Histogram(df, title='Histogram')
```

Scatter Plot

```
>>> from bokeh.charts import Scatter
>>> p = Scatter(df, x='mpg', y ='hp', marker='square',
                xlabel='Miles Per Gallon',
                ylabel='Horsepower')
```

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

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

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

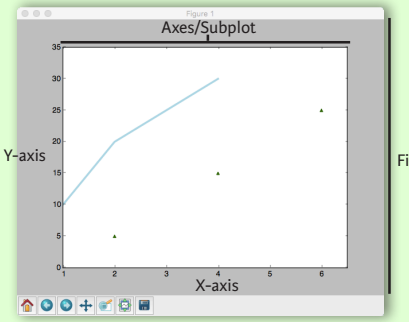
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot 2D vector fields
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6))
```
Create a figure and one subplot

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```
(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters

Return a dict of params or use with `with` to temporarily set the style

Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                    font_scale=1.5,
                    rc={"lines.linewidth":2.5})
```
Set context to `"talk"`
Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

```
>>> sns.set_palette("husl",3)
>>> sns.color_palette("husl")
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui)
```
Define the color palette
Use with `with` to temporarily set palette
Set your own color palette

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
```
Subplot grid for plotting conditional relationships

```
>>> sns.factorplot(x="pclass",
                   y="survived",
                   hue="sex",
                   data=titanic)
```
Draw a categorical plot onto a Facetgrid

```
>>> sns.lmplot(x="sepal_width",
               y="sepal_length",
               hue="species",
               data=iris)
```
Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
```
Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions

```
>>> i = sns.JointGrid(x="x",
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
```
Grid for bivariate plot with marginal univariate plots

```
>>> sns.jointplot("sepal_length",
                  "sepal_width",
                  data=iris,
                  kind='kde')
```
Plot bivariate distribution

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species",
                  y="petal_length",
                  data=iris)
```
Scatterplot with one categorical variable

```
>>> sns.swarmplot(x="species",
                  y="petal_length",
                  data=iris)
```
Categorical scatterplot with non-overlapping points

Bar Chart

```
>>> sns.barplot(x="sex",
                y="survived",
                hue="class",
                data=titanic)
```
Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

```
>>> sns.countplot(x="deck",
                  data=titanic,
                  palette="Greens_d")
```
Show count of observations

Point Plot

```
>>> sns.pointplot(x="class",
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                           "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```
Show point estimates and confidence intervals as rectangular bars

Boxplot

```
>>> sns.boxplot(x="alive",
                y="age",
                hue="adult_male",
                data=titanic)
```
Boxplot

```
>>> sns.boxplot(data=iris,orient="h")
```
Boxplot with wide-form data

Violinplot

```
>>> sns.violinplot(x="age",
                   y="sex",
                   hue="survived",
                   data=titanic)
```
Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width",
                y="sepal_length",
                data=iris,
                ax=ax)
```
Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,
                        kde=False,
                        color="b")
```
Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1)
```
Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
                      "Sex")
```
Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

```
>>> h.set(xlim=(0,5),
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```
Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax,yticks=[0,5])
>>> plt.tight_layout()
```
Add plot title
Adjust the label of the y-axis
Adjust the label of the x-axis
Adjust the limits of the y-axis
Adjust the limits of the x-axis
Adjust a plot property
Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

```
>>> plt.show()
>>> plt.savefig("foo.png")
>>> plt.savefig("foo.png",
                transparent=True)
```
Show the plot
Save the plot as a figure
Save transparent figure

Close & Clear

Also see Matplotlib

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```
Clear an axis
Clear an entire figure
Close a window

Python Seaborn Tutorial For Beginners

This Seaborn tutorial introduces you to the basics of statistical data visualization

Seaborn: Python's Statistical Data Visualization Library

One of the best but also more challenging ways to get your insights across is to visualize them: that way, you can more easily identify patterns, grasp difficult concepts or draw the attention to key elements. When you're using Python for data science, you'll most probably will have already used Matplotlib, a 2D plotting library that allows you to create publication-quality figures. Another complimentary package that is based on this data visualization library is Seaborn, which provides a high-level interface to draw statistical graphics.

Today's post will cover some of the most frequently asked questions users had while they started out working with the Seaborn library. How many of the following questions can you answer correctly?

1. Seaborn vs Matplotlib?
2. How To Load Data To Construct Seaborn Plots
   - Loading A Built-in Data Set
   - Loading Your Pandas DataFrame
3. How To Show Seaborn Plots
4. How To Use Seaborn With Matplotlib Defaults
5. How To Use Seaborn's Colors As A colormap in Matplotlib?
6. How To Scale Seaborn Plots For Other Context
7. How To Temporarily Set The Plot Style
8. How To Set The Figure Size in Seaborn
9. How To Rotate Label Text
10. How To Set `xlim` or `ylim` in Seaborn
11. How To Set Log Scale

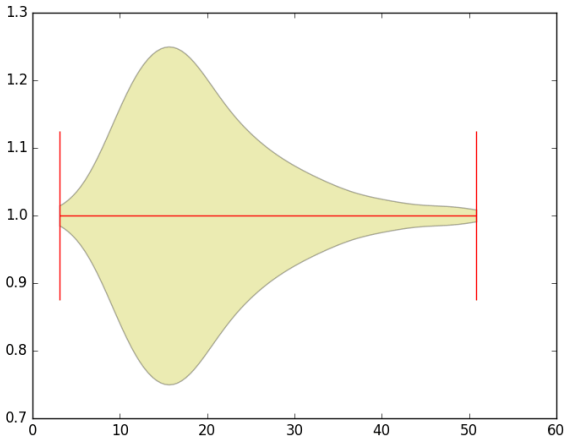
12. How To Add A Title

Seaborn vs Matplotlib

As you have just read, Seaborn is complimentary to Matplotlib and it specifically targets statistical data visualization. But it goes even further than that: Seaborn extends Matplotlib and that's why it can address the two biggest frustrations of working with Matplotlib. Or, as Michael Waskom says in the "introduction to Seaborn": "If matplotlib "tries to make easy things easy and hard things possible", seaborn tries to make a well-defined set of hard things easy too."

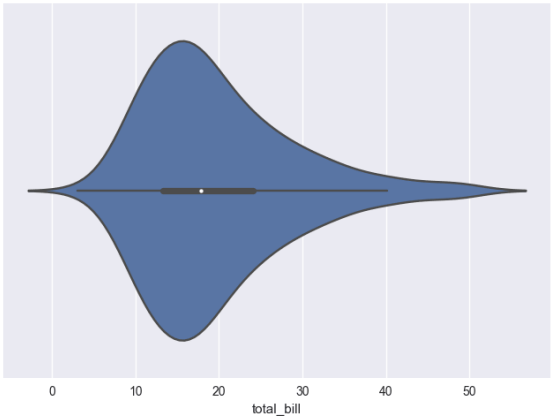
One of these hard things or frustrations had to do with the default Matplotlib parameters. Seaborn works with different parameters, which undoubtedly speaks to those users that don't use the default looks of the Matplotlib plots.

```
# Import the necessary libraries
import matplotlib.pyplot as plt
import pandas as pd
# Initialize Figure and Axes object
fig, ax = plt.subplots()
# Load in data
tips = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/master/tips.csv")
# Create violinplot
ax.violinplot(tips["total_bill"], vert=False)
# Show the plot
plt.show()
```




```
# Import the necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
# Load the data
```

```
tips = sns.load_dataset("tips")
# Create violinplot
sns.violinplot(x = "total_bill", data=tips)
# Show the plot
plt.show()
```



The Matplotlib defaults that usually don’t speak to users are the colors, the tick marks on the upper and right axes, the style,…

The examples above also makes another frustration of users more apparent: the fact that working with DataFrames doesn’t go quite as smoothly with Matplotlib, which can be annoying if you’re doing exploratory analysis with Pandas. And that’s exactly what Seaborn addresses: the plotting functions operate on DataFrames and arrays that contain a whole dataset.

As Seaborn compliments and extends Matplotlib, the learning curve is quite gradual: if you know Matplotlib, you’ll already have most of Seaborn down.

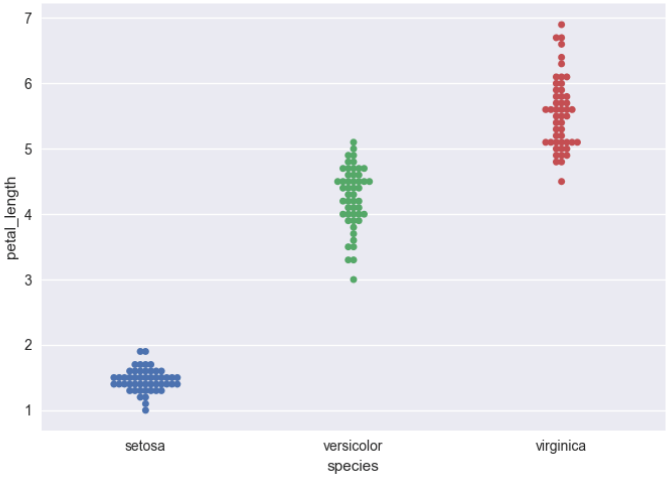
How To Load Data To Construct Seaborn Plots

When you’re working with Seaborn, you can either use one of the built-in data sets that the library itself has to offer or you can load a Pandas DataFrame. In this section, you’ll see how to do both.

Loading A Built-in Seaborn Data Set

To start working with a built-in Seaborn data set, you can make use of the `load_dataset()` function. To get an overview or inspect all data sets that this function opens up to you, go here. Check out the following example to see how the `load_dataset()` function works:

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
# Load iris data
iris = sns.load_dataset("iris")
# Construct iris plot
sns.swarmplot(x="species", y="petal_length", data=iris)
# Show plot
plt.show()
```



As an anecdote, it might be interesting for you to know that the import convention `sns` comes from the fictional character Samuel Norman "Sam" Seaborn on the television serial drama The West Wing. It's an inside joke by the core developer of Seaborn, namely, Michael Waskom.

Loading Your Pandas DataFrame Getting Your Data

Of course, most of the fun in visualizing data lies in the fact that you would be working with your own data and not the built-in data sets of the Seaborn library. Seaborn works best with Pandas DataFrames and arrays that contain a whole data set.

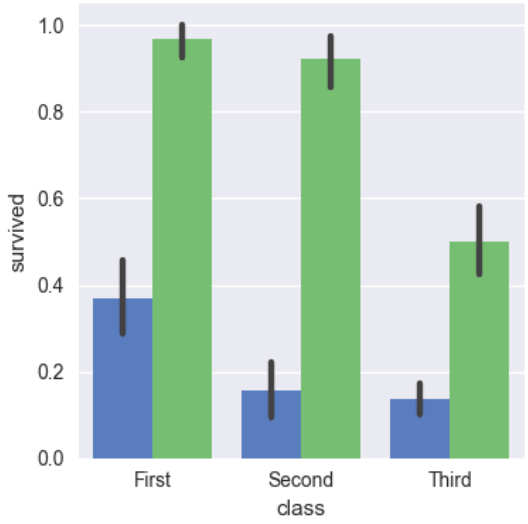
Remember that DataFrames are a way to store data in rectangular grids that can easily be overviewed. Each row of these grids corresponds to measurements or values of an instance, while each column is a vector containing data for a specific variable. This means that a

DataFrame's rows do not need to contain, but can contain, the same type of values: they can be numeric, character, logical, etc. Specifically for Python, DataFrames come with the Pandas library, and they are defined as a two-dimensional labeled data structures with columns of potentially different types.
The reason why Seaborn is so great with DataFrames is, for example, because labels from DataFrames are automatically propagated to plots or other data structures, as you saw in the first example of this tutorial, where you plotted a violinplot with Seaborn. There, you saw that the x-axis had a legend `total_bill`, while this was not the case with the Matplotlib plot. This already takes a lot of work away from you.
But that doesn't mean that all the work is done -quite the opposite. In many cases, you'll need to still manipulate your Pandas DataFrame so that the plot will render correctly. If you want to know more, check out DataCamp's Pandas Tutorial on DataFrames in Python or the Pandas Foundations course.

How To Show Seaborn Plots

Matplotlib still underlies Seaborn, which means that the anatomy of the plot is still the same and that you'll need to use `plt.show()` to make the image appear to you. You might have already seen this from the previous example in this tutorial. In any case, here's another example where the `show()` function is used to show the plot:

```
# Import necessarily libraries
import matplotlib.pyplot as plt
import seaborn as sns
# Load data
titanic = sns.load_dataset("titanic")
# Set up a factorplot
g = sns.factorplot("class", "survived", "sex", data=titanic, kind="bar", palette="muted", legend=False)

# Show plot
plt.show()
```



Note that in the code chunk above you work with a built-in Seaborn data set and you create a factorplot with it. A factorplot is a categorical plot, which in this case is a bar plot. That's because you have set the `kind` argument to `"bar"`. Also, you set which colors should be displayed with the `palette` argument and that you set the `legend` to `False`.

How To Use Seaborn With Matplotlib Defaults

As you read in the introduction, the Matplotlib defaults are something that users might not find as pleasing than the Seaborn defaults. However, there are also many questions in the opposite direction, namely, those use Seaborn and that want to plot with Matplotlib defaults.

Before, you could solve this question by importing the `apionly` module from the Seaborn package. This is now deprecated (since July 2017). The default style is no longer applied when Seaborn is imported, so you'll need to explicitly call `set()` or one or more of `set_style()`, `set_context()`, and `set_palette()` to get either Seaborn or Matplotlib defaults for plotting.

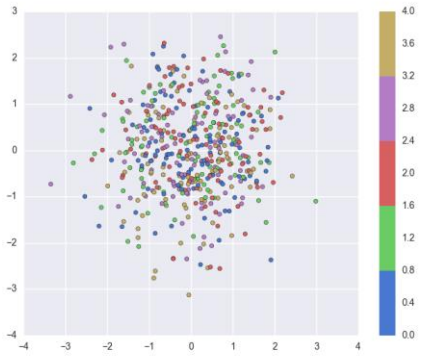
```
# Import Matplotlib
import matplotlib.pyplot as plt
# Check the available styles
plt.style.available
# Use Matplotlib defaults
plt.style.use("classic")
```

How To Use Seaborn's Colors As A colormap in Matplotlib?

Besides using Seaborn with Matplotlib defaults, there's also questions on how to bring in Seaborn colors into Matplotlib plots. You can make use of `color_palette()` to define a color map that you want to be using and the number of colors with the argument `n_colors`. In this case, the example will assume that there are 5 labels assigned to the data points that are defined in `data1` and `data2`, so that's why you pass `5` to this argument and you also make a list with length equal to `N` where 5 integers vary in the variable `colors`.

```
# Import the necessary libraries
import seaborn as sns
```

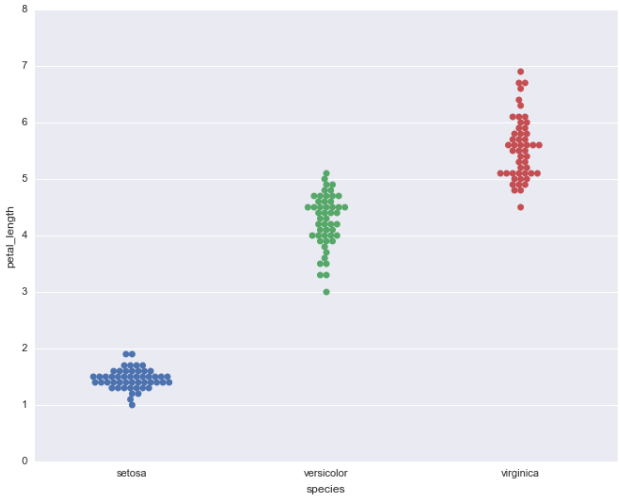
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
# Define a variable N
N = 500
# Construct the colormap
current_palette = sns.color_palette("muted", n_colors=5)
cmap = ListedColormap(sns.color_palette(current_palette).as_hex())
# Initialize the data
data1 = np.random.randn(N)
data2 = np.random.randn(N)
# Assume that there are 5 possible labels
colors = np.random.randint(0,5,N)
# Create a scatter plot
plt.scatter(data1, data2, c=colors, cmap=cmap)
# Add a color bar
plt.colorbar()
# Show the plot
plt.show()
```



How To Scale Seaborn Plots For Other Contexts

If you need your plots for talks, posters, on paper or in notebooks, you might want to have larger or smaller plots. Seaborn has got you covered on this. You can make use of `set_context()` to control the plot elements:

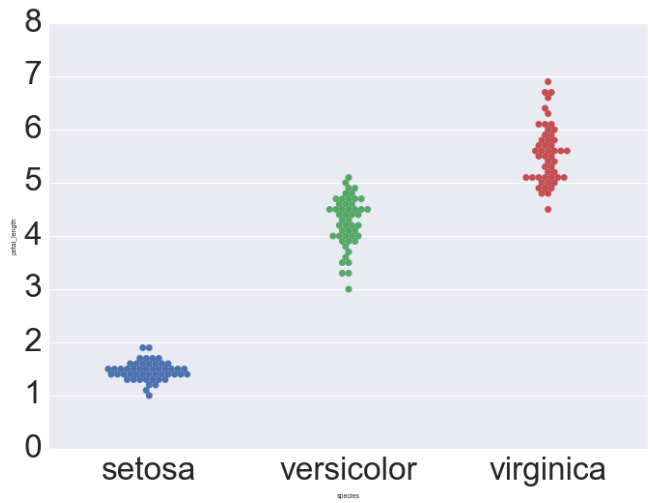
```
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
# Reset default params
sns.set()
# Set context to `"paper"`
sns.set_context("paper")
# Load iris data
iris = sns.load_dataset("iris")
# Construct iris plot
sns.swarmplot(x="species", y="petal_length", data=iris)
# Show plot
plt.show()
```



The four predefined contexts are `"paper"`, `"notebook"`, `"talk"` and `"poster"`. **Tip**: try changing the context in the DataCamp Light chunk above to another context to study the effect of the contexts on the plot.

You can also pass more arguments to `set_context()` to scale more plot elements, such as `font_scale` or more parameter mappings that can override the values that are preset in the Seaborn context dictionaries. In the following code chunk, you overwrite the values that are set for the parameters `font.size` and `axes.labelsize`:

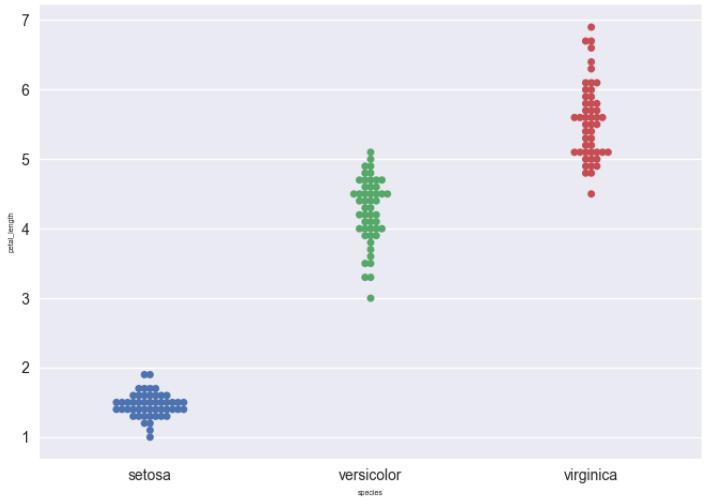
```
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
# Set context to `"paper"`
sns.set_context("paper", font_scale=3, rc={"font.size":8,"axes.labelsize":5})
# Load iris data
iris = sns.load_dataset("iris")
# Construct iris plot
sns.swarmplot(x="species", y="petal_length", data=iris)
# Show plot
plt.show()
```



Note that in the first code chunk, you have first done a reset to get the default Seaborn parameters back. You did this by calling `set()`. This is extremely handy if you have experimented with setting other parameters before, such as the plot style.

Additionally, it's good to keep in mind that you can use the higher-level `set()` function instead of `set_context()` to adjust other plot elements:

```
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
# Reset default params
sns.set(rc={"font.size":8,"axes.labelsize":5})
# Load iris data
iris = sns.load_dataset("iris")
# Construct iris plot
sns.swarmplot(x="species", y="petal_length", data=iris)
# Show plot
plt.show()
```



One of the hardest things about data visualizations is customizing the graphs further until they meet your expectations and this stays the same when you're working with Seaborn. That's why it's good to keep in mind the anatomy of the Matplotlib plot and also what this means for the Seaborn library.

As for Seaborn, you have two types of functions: axes-level functions and figure-level functions. The ones that operate on the Axes level are, for example, `regplot()`, `boxplot()`, `kdeplot()`, …, while the functions that operate on the Figure level are `lmplot()`, `factorplot()`, `jointplot()` and a couple others.

This means that the first group is identified by taking an explicit `ax` argument and returning an `Axes` object, while the second group of functions create plots that potentially include `Axes` which are always organized in a "meaningful" way. The Figure-level functions will therefore need to have total control over the figure so you won't be able to plot an lmplot onto one that already exists. When you call the Figure-level functions, you always initialize a figure and set it up for the specific plot it's drawing.

You can easily see this when you make a boxplot and an lmplot, for example:

```
>>> sns.boxplot(x="total_bill", data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot object at 0x117e8da20>



>>> sns.lmplot('x', 'y', data, size=7, truncate=True, scatter_kws={"s": 100})
<seaborn.axisgrid.FacetGrid object at 0x11fa03438>
```

However, you see that, once you've called `lmplot()`, it returns an object of the type `FacetGrid`. This object has some methods for operating on the resulting plot that know a bit about the structure of the plot. It also exposes the underlying figure and array of axes at the `FacetGrid.fig` and `FacetGrid.axes` arguments.

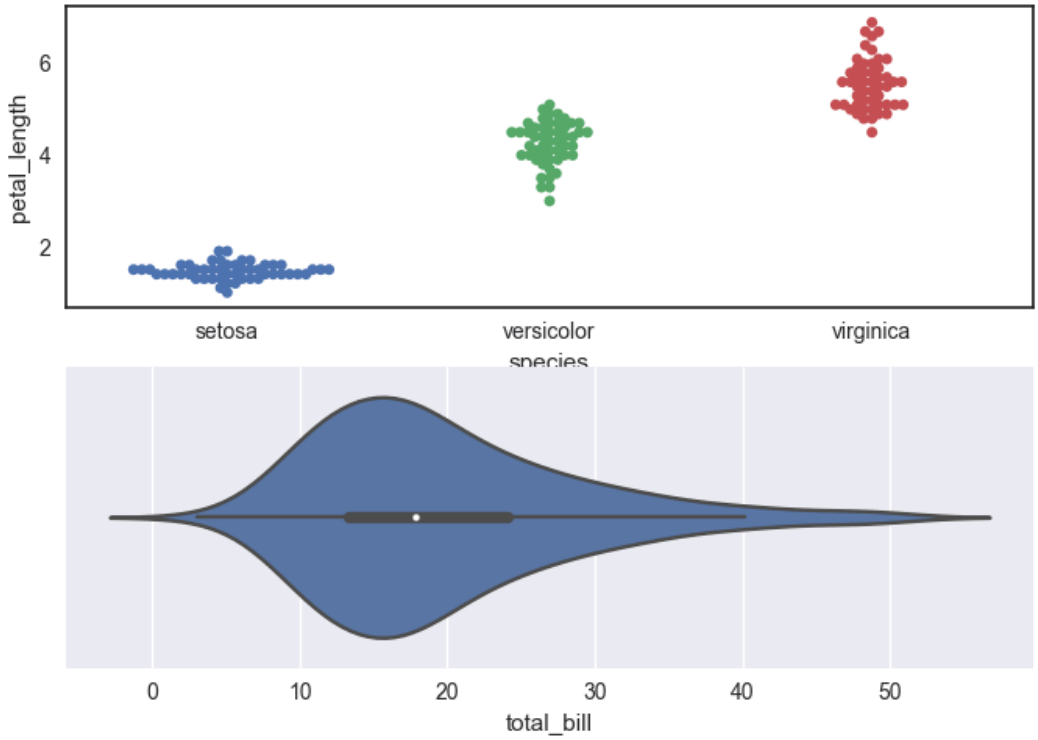
When you're customizing your plots, this means that you will prefer to make customizations to your regression plot that you constructed with `regplot()` on `Axes` level, while you will make customizations for `lmplot()` on Figure level.

Let's see how this works in practice by covering some of the following, most frequently asked questions:

How To Temporarily Set The Plot Style

You can use `axes_style()` in a `with` statement to temporarily set the plot style. This, in addition to the use of `plt.subplot()`, will allow you to make figures that have differently-styled axes, like in the example below:

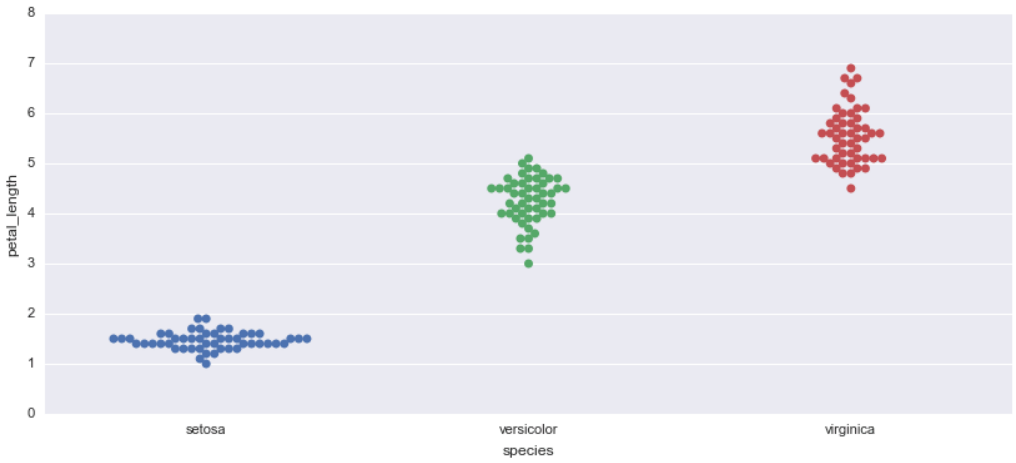
```
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
# Load data
iris = sns.load_dataset("iris")
tips = sns.load_dataset("tips")
# Set axes style to white for first subplot
with sns.axes_style("white"):
    plt.subplot(211)
    sns.swarmplot(x="species", y="petal_length", data=iris)
# Initialize second subplot
plt.subplot(212)
# Plot violinplot
sns.violinplot(x = "total_bill", data=tips)
# Show the plot
plt.show()
```



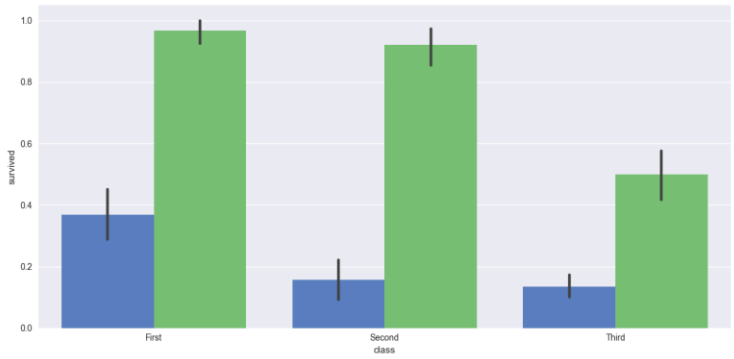
How To Set The Figure Size in Seaborn

For axes level functions, you can make use of the `plt.subplots()` function to which you pass the `figsize` argument.

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
# Initialize Figure and Axes object
fig, ax = plt.subplots(figsize=(10,4))
# Load in the data
iris = sns.load_dataset("iris")
# Create swarmplot
sns.swarmplot(x="species", y="petal_length", data=iris, ax=ax)
# Show plot
plt.show()
```




```
# Import the libraries
import matplotlib.pyplot as plt
import seaborn as sns
# Load data
titanic = sns.load_dataset("titanic")
# Set up a factorplot
g = sns.factorplot("class", "survived", "sex", data=titanic, kind="bar", size=6, aspect=2, palette="muted",
legend=False)
# Show plot
plt.show()
```

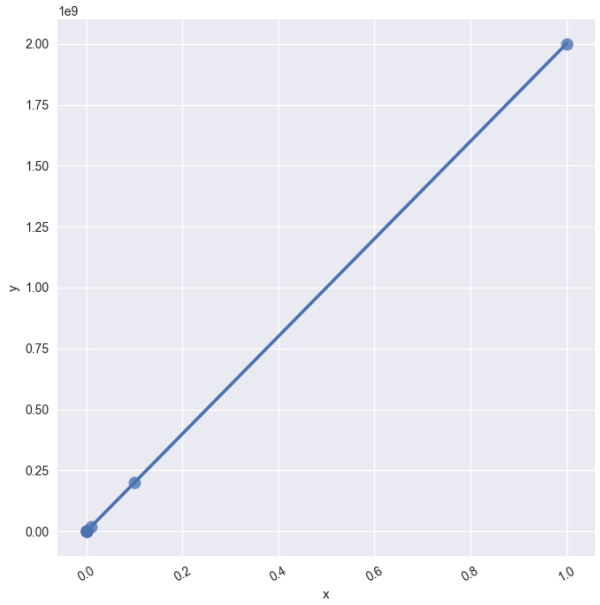


How To Rotate Label Text in Seaborn

To rotate the label text in a Seaborn plot, you will need to work on the Figure level. Note that in the code chunk below, you make use of one of the FacetGrid methods, namely, `set_xticklabels`, to rotate the text label:

```
# Import the necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
# Initialize the data
```

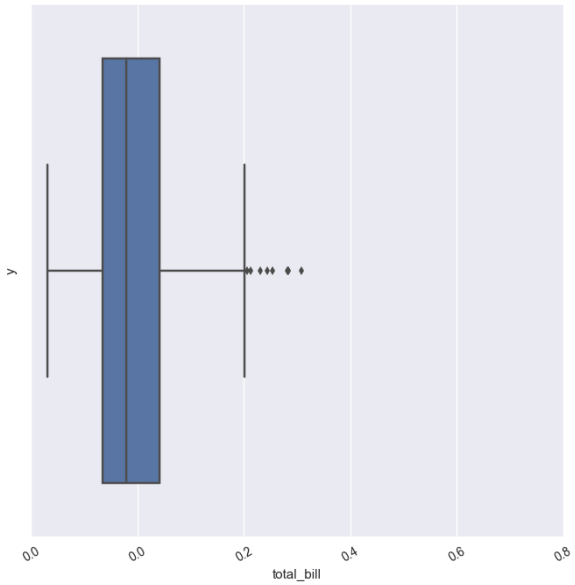
```
x = 10 ** np.arange(1, 10)
y = x * 2
data = pd.DataFrame(data={'x': x, 'y': y})
# Create an lmplot
grid = sns.lmplot('x', 'y', data, size=7, truncate=True, scatter_kws={"s": 100})
# Rotate the labels on x-axis
grid.set_xticklabels(rotation=30)
# Show the plot
plt.show()
```



How To Set `xlim` or `ylim` in Seaborn

For a boxplot, which works at the Axes level, you'll need to make sure to assign your boxplot to a variable `ax`, which will be a `matplotlib.axes._subplots.AxesSubplot` object, as you saw above. With the object at Axes level, you can make use of the `set()` function to set `xlim`, `ylim`,… Just like in the following example:

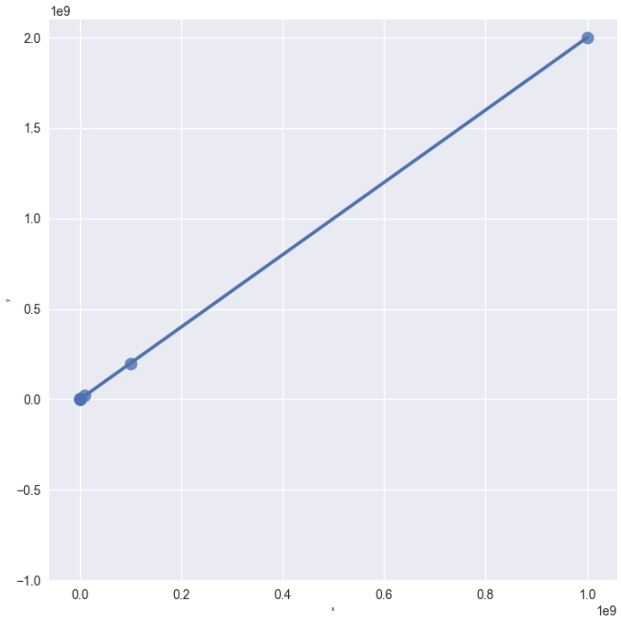
```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
# Load the data
tips = sns.load_dataset("tips")
# Create the boxplot
ax = sns.boxplot(x="total_bill", data=tips)
# Set the `xlim`
ax.set(xlim=(0, 100))
# Show the plot
plt.show()
```



Note that alternatively, you could have also used `ax.set_xlim(10,100)` to limit the x-axis.

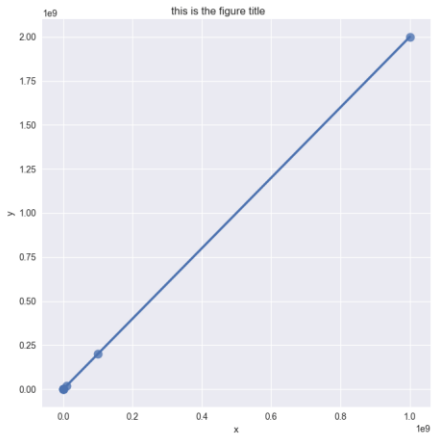
Now, for functions at Figure-level, you can access the `Axes` object with the help of the `axes` argument. Let's see how you can use the `ax` argument to your advantage to set the `xlim` and `ylim` properties:

```
# Import the necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
# Initialize the data
x = 10 ** np.arange(1, 10)
y = x * 2
data = pd.DataFrame(data={'x': x, 'y': y})
# Create lmplot
lm = sns.lmplot('x', 'y', data, size=7, truncate=True, scatter_kws={"s": 100})
# Get hold of the `Axes` objects
axes = lm.ax
# Tweak the `Axes` properties
axes.set_ylim(-1000000000,)
# Show the plot
plt.show()
```



Likewise, `FacetGrid` exposes the underlying figure with the help of the `fig` argument.

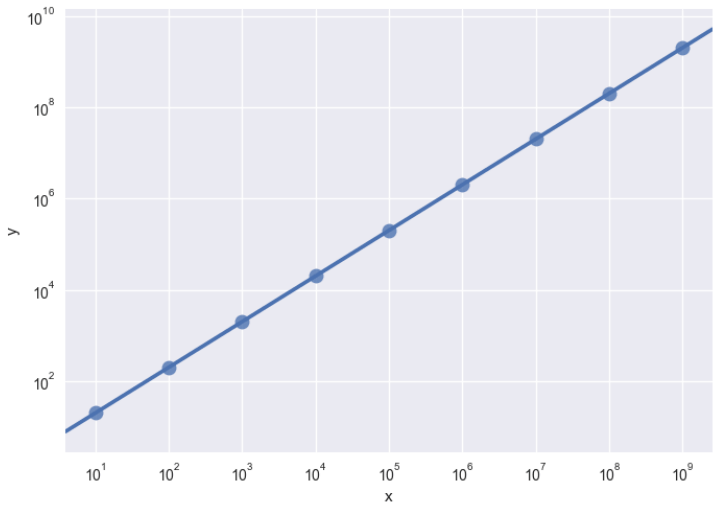
```
# Import the necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
# Initialize the data
x = 10 ** np.arange(1, 10)
y = x * 2
data = pd.DataFrame(data={'x': x, 'y': y})
# Create lmplot
lm = sns.lmplot('x', 'y', data, size=7, truncate=True, scatter_kws={"s": 100})
# Access the Figure
fig = lm.fig
# Add a title to the Figure
fig.suptitle('this is the figure title', fontsize=12)
# Show the plot
plt.show()
```



How To Set Log Scale

You can modify the scale of your axes to better show trends. That's why it might be useful in some cases to use the logarithmic scale on one or both axes. For a simple regression with `regplot()`, you can set the scale with the help of the `Axes` object.

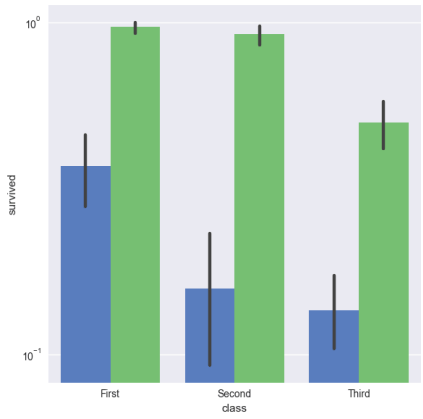
```
# Import the necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
# Create the data
x = 10 ** np.arange(1, 10)
y = x * 2
data = pd.DataFrame(data={'x': x, 'y': y})
# Initialize figure and ax
fig, ax = plt.subplots()
# Set the scale of the x-and y-axes
ax.set(xscale="log", yscale="log")
# Create a regplot
sns.regplot("x", "y", data, ax=ax, scatter_kws={"s": 100})
# Show plot
plt.show()
```



When you're working with Figure level functions, you can set the `xscale` and `yscale` properties with the help of the `set()` method of the `FacetGrid` object:

```
# Import the libraries
import matplotlib.pyplot as plt
import seaborn as sns
# Load data
titanic = sns.load_dataset("titanic")
```

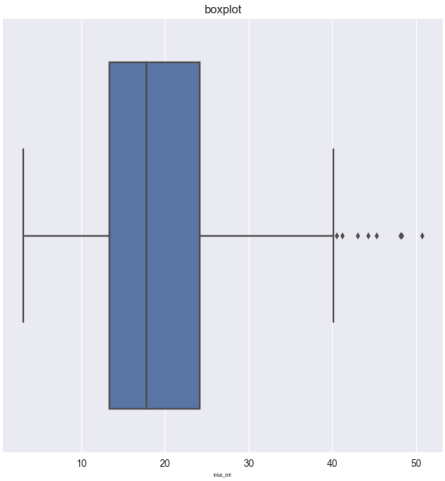
```
# Set up a factorplot
g = sns.factorplot("class", "survived", "sex", data=titanic, kind="bar", size=6, palette="muted", legend=False)
# Set the `yscale`
g.set(yscale="log")
# Show plot
plt.show()
```



How To Add A Title

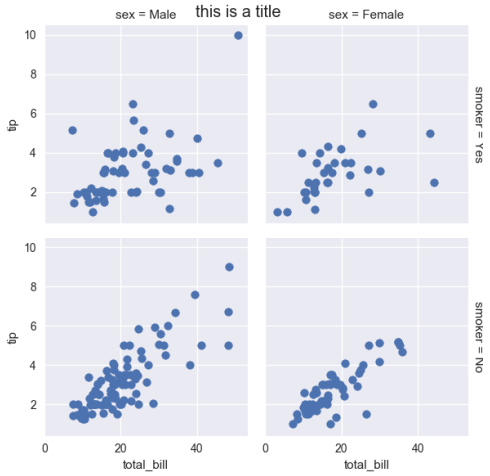
To add titles to your Seaborn plots, you basically follow the same procedure as you have done in the previous sections. For Axes-level functions, you'll adjust the title on the `Axes` level itself with the help of `set_title()`. Just pass in the title that you want to see appear:

```
# Import the libraries
import matplotlib.pyplot as plt
import seaborn as sns
tips = sns.load_dataset("tips")
# Create the boxplot
ax = sns.boxplot(x="total_bill", data=tips)
# Set title
ax.set_title("boxplot")
# Show the plot
plt.show()
```



For Figure-level functions, you can go via `fig`, just like in the factorplot that you have made in one of the previous sections, or you can also work via the Axes:

```
# Import the necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
# Load the data
tips = sns.load_dataset("tips")
# Create scatter plots
g = sns.FacetGrid(tips, col="sex", row="smoker", margin_titles=True)
g.map(sns.plt.scatter, "total_bill", "tip")
# Add a title to the figure
g.fig.suptitle("this is a title")
# Show the plot
plt.show()
```



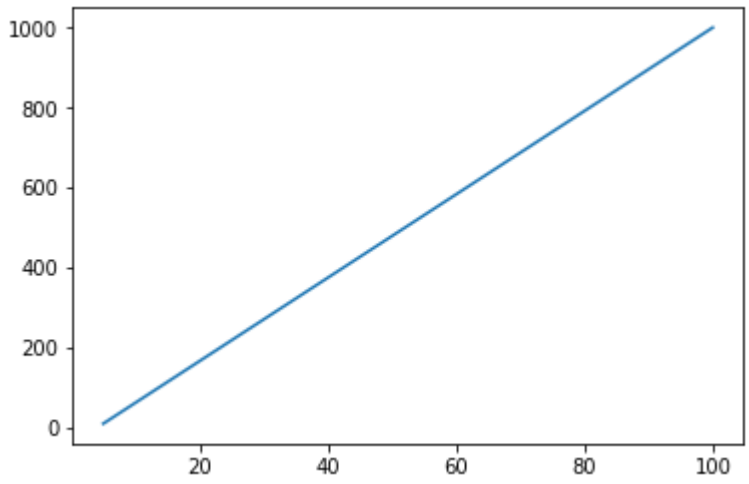
Basic Plotting: Introduction to `matplotlib`

In this section, we will:

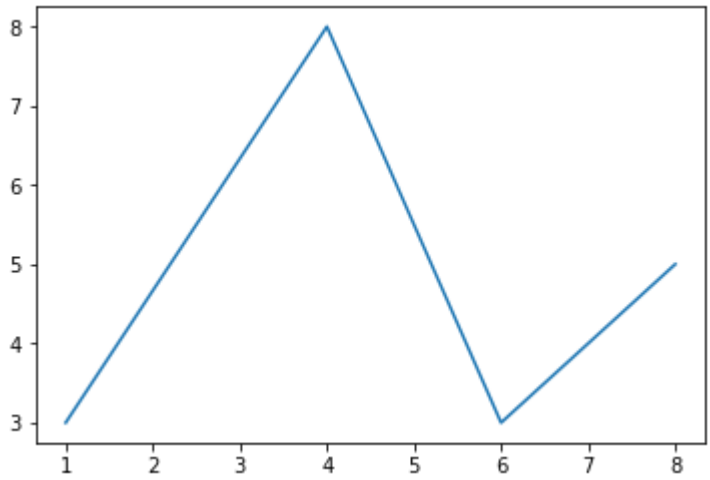
- Create basic plots using `matplotlib.pyplot`
- Put axis labels and titles
- Create multiple plots (subplots) in the same figure
- Change the scales of x and y axes
- Create common types of plots: Histograms, boxplots, scatter plots and bar charts
- Working with images

Basic Plotting, Axes Labels and Titles

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Plotting two 1-D numpy arrays
x = np.linspace(5, 100, 100)
y = np.linspace(10, 1000, 100)

plt.plot(x, y)
plt.show()
```



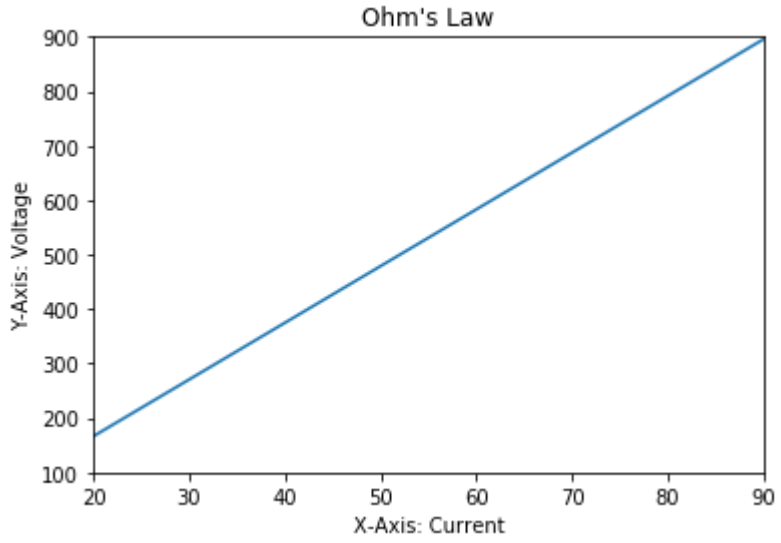
In [5]:
```
# let us plot our first basic chart
plt.plot([1, 4, 6, 8], [3, 8, 3, 5])
plt.show()
```



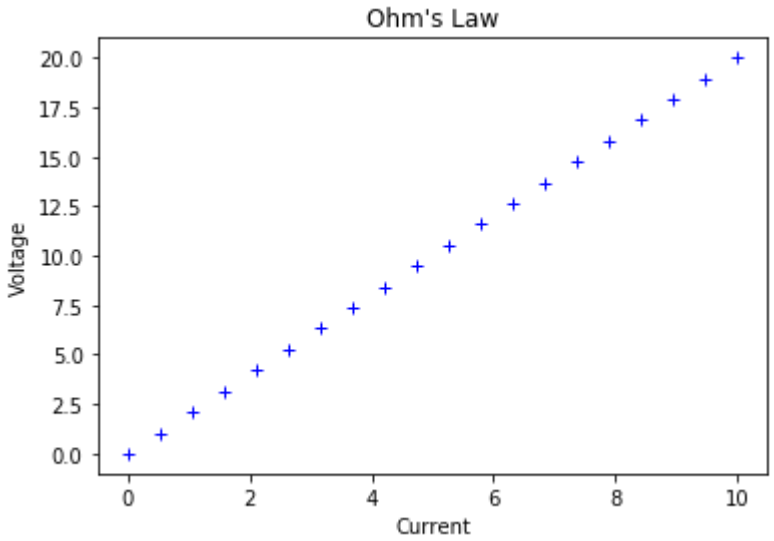
Let's see how to put labels and the x and y axes and the chart title.

Also, you can specify the limits of x and y labels as a range using `xlim([xmin, xmax])` and `ylim([ymin, ymax])` .

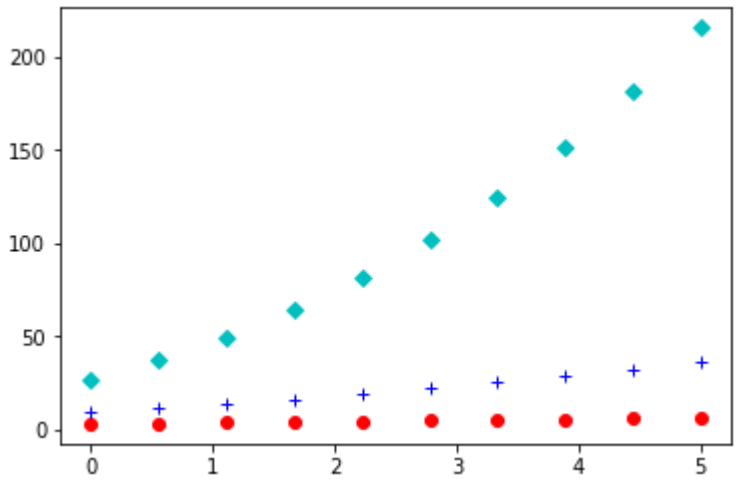
In [6]:
```
# Axis labels and title
plt.plot(x, y)

# x and y labels, and title
plt.xlabel("X-Axis: Current")
plt.ylabel("Y-Axis: Voltage")
plt.title("Ohm's Law")

# Define the range of labels of the axis
# Arguments: plt.axis(xmin, xmax, ymin, ymax)
plt.xlim([20, 90])
plt.ylim([100, 900])
plt.show()
```



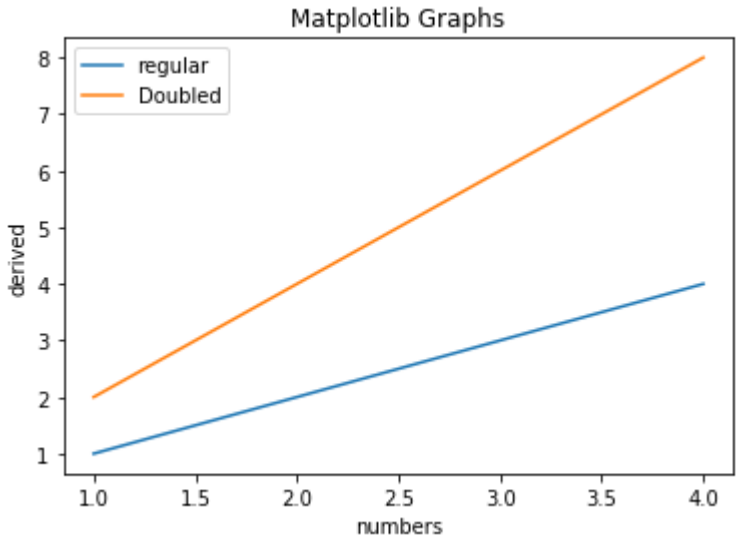
In [7]:
```
# Change the colors and line type

# initialising x and y arrays
x = np.linspace(0, 10, 20)
y = x*2

# color blue, line type '+'
plt.plot(x, y, 'b+')

# put x and y labels, and the title
plt.xlabel("Current")
plt.ylabel("Voltage")
plt.title("Ohm's Law")

plt.show()
```



In [8]:
```
# Plotting multiple lines on the same plot

x = np.linspace(0, 5, 10)
y = np.linspace(3, 6, 10)

# plot three curves: y, y**2 and y**3 with different line types
plt.plot(x, y, 'ro', x, y**2, 'b+', x, y**3, 'cD')
plt.show()
```

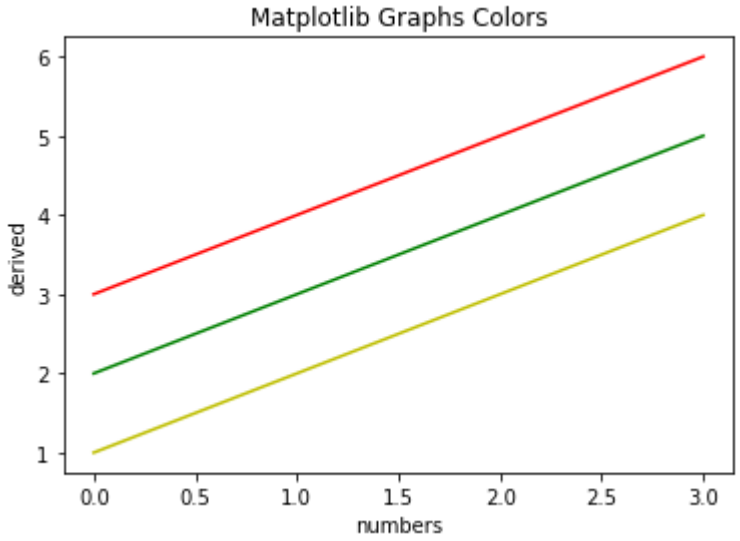


In [9]:
```
# Plotting a complete chart

x = np.arange(1, 5)

plt.title("Matplotlib Graphs")

plt.plot(x, x*1, label = "regular")
plt.plot(x, x*2, label = "Doubled")

plt.grid(False)
plt.xlabel('numbers')
plt.ylabel('derived')

plt.legend()

plt.show()
```

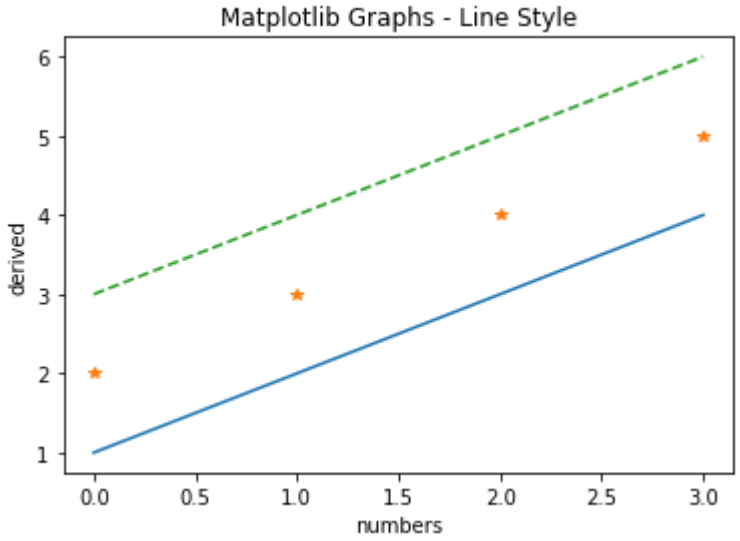


Decoration with Plot styles and Types

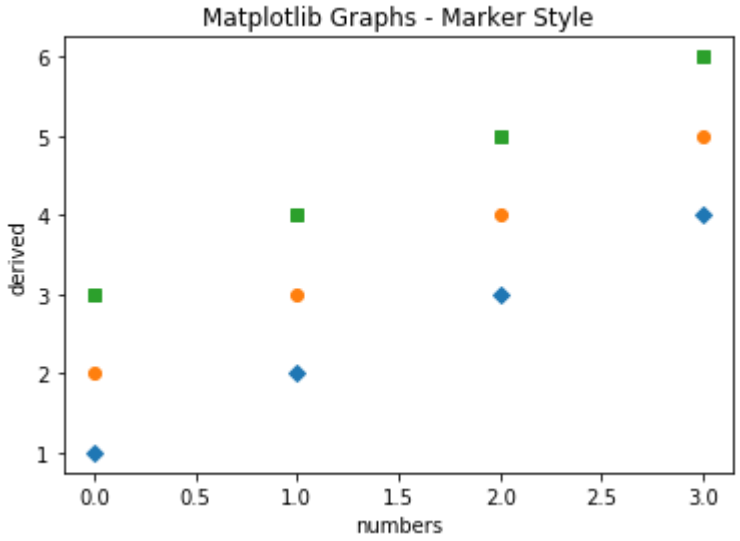
In [10]:
```
x = np.arange(1, 5)

plt.title("Matplotlib Graphs Colors")

plt.plot(x, 'y')
plt.plot(x + 1, 'g')
plt.plot(x + 2, 'r')

plt.grid(False)
plt.xlabel('numbers')
plt.ylabel('derived')

plt.show()
```



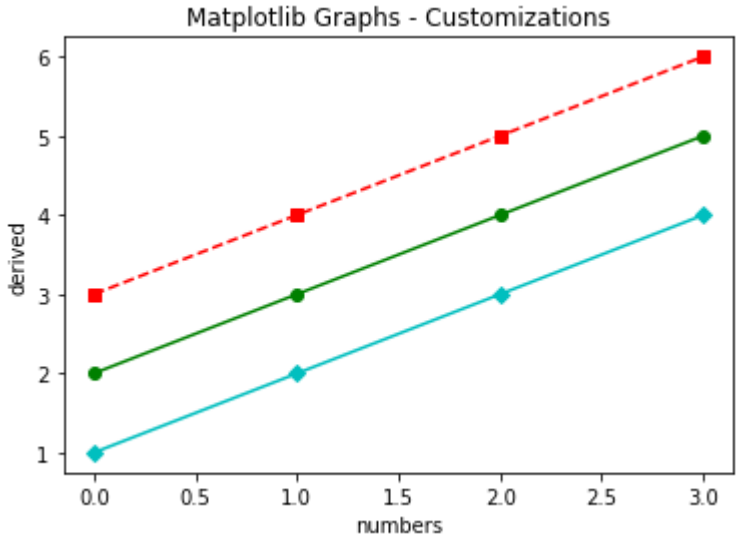
In [11]:
```
x = np.arange(1, 5)

plt.title("Matplotlib Graphs - Line Style")

plt.plot(x, '-')
plt.plot(x + 1, '*')
plt.plot(x + 2, '--')

plt.grid(False)
plt.xlabel('numbers')
plt.ylabel('derived')

plt.show()
```



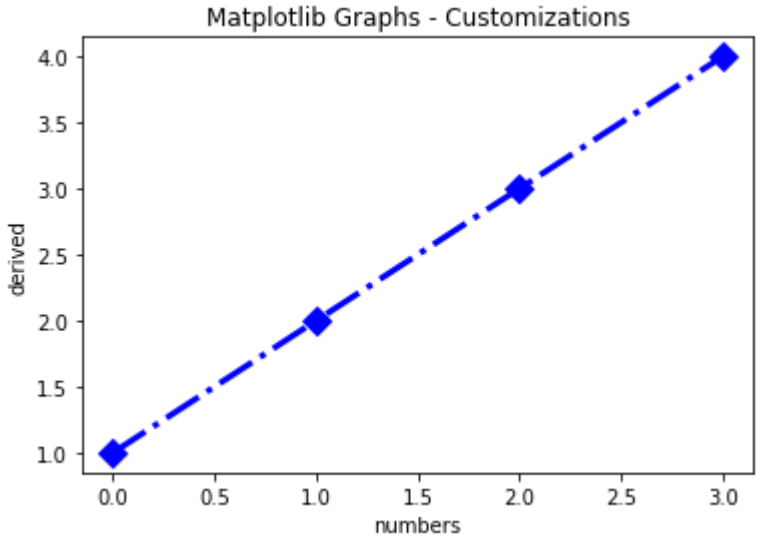
In [12]:
```
x = np.arange(1, 5)

plt.title("Matplotlib Graphs - Marker Style")

plt.plot(x, 'D')
plt.plot(x + 1, 'o')
plt.plot(x + 2, 's')

plt.grid(False)
plt.xlabel('numbers')
plt.ylabel('derived')

plt.show()
```



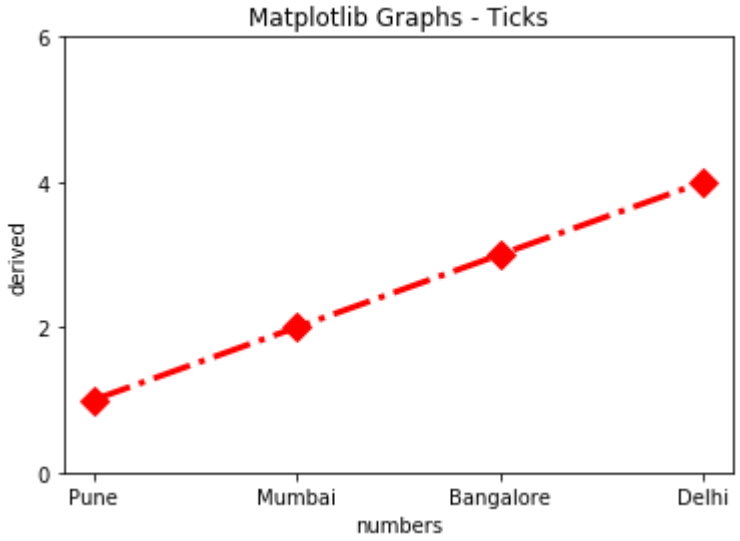
In [13]:
```
x = np.arange(1, 5)

plt.title("Matplotlib Graphs - Customizations")

plt.plot(x, 'cD-')
plt.plot(x + 1, 'go-')
plt.plot(x + 2, 'rs--')

plt.grid(False)
plt.xlabel('numbers')
plt.ylabel('derived')

plt.show()
```



In [14]:
```
x = np.arange(1, 5)

plt.title("Matplotlib Graphs - Customizations")

plt.plot(x, color="blue", linestyle="dashdot", linewidth=3, marker="D", markersi

plt.grid(False)
plt.xlabel('numbers')
plt.ylabel('derived')

plt.show()
```



In [15]:
```
x = np.arange(1, 5)

plt.title("Matplotlib Graphs - Ticks")

plt.plot(x, color="red", linestyle="dashdot", linewidth=3, marker="D", markersize
plt.xticks(range(len(x)),['Pune', 'Mumbai', "Bangalore", "Delhi"])
plt.yticks(range(0, 8, 2))
plt.grid(False)
plt.xlabel('numbers')
plt.ylabel('derived')

plt.show()
```



Advanced Matplotlib

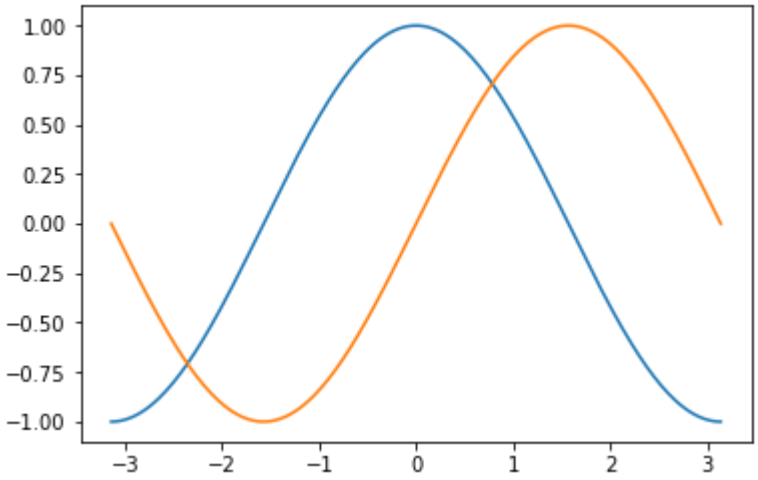
In this section, we want to draw the cosine and sine functions on the same plot. Starting from the default settings, we'll enrich the figure step by step to make it nicer.

In [17]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

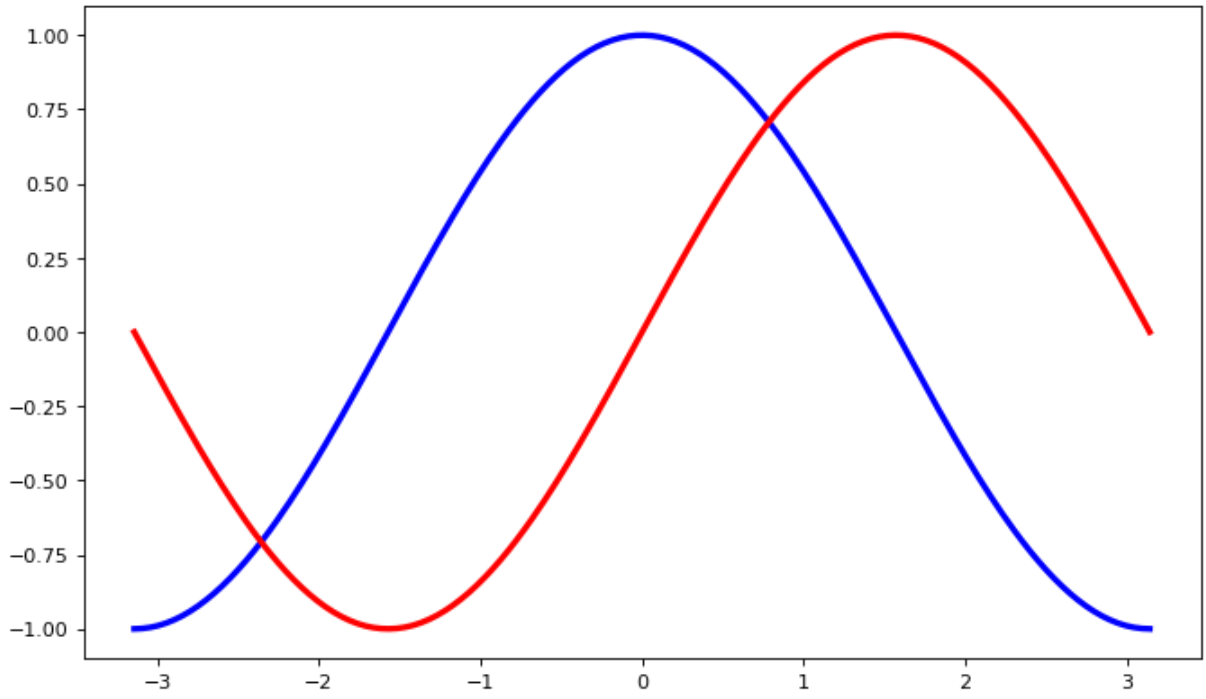
In [18]:
```
X = np.linspace(-np.pi, np.pi, 256, endpoint = True)
C, S = np.cos(X), np.sin(X)
```

X is now a NumPy array with 256 values ranging from -π to +π (included). C is the cosine (256 values) and S is the sine (256 values).

In [19]:
```
plt.plot(X,C)
plt.plot(X,S)
plt.show()
```



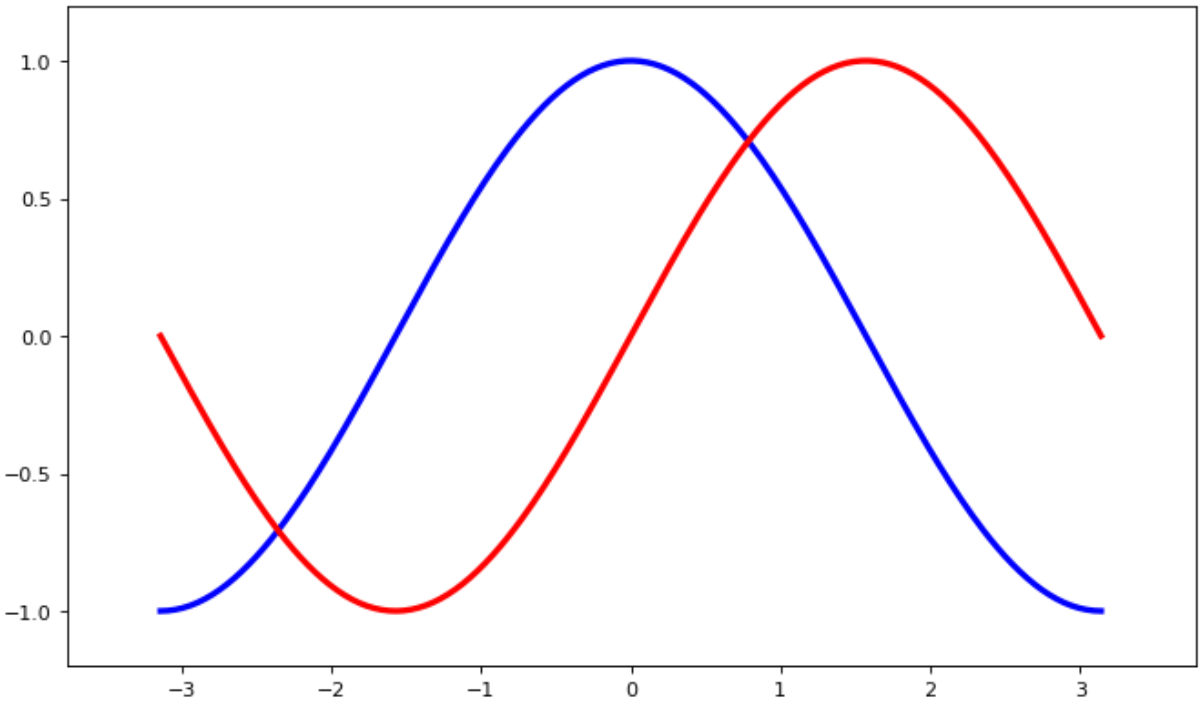
In [20]:
```
plt.figure(figsize=(10,6), dpi = 80)
plt.plot(X,C, color = 'blue', linewidth = 3, linestyle = '-')
plt.plot(X,S, color = 'red', linewidth = 3, linestyle = '-')
plt.show()
```



Setting limits

current limits of the figure are a bit too tight and we want to make some space in order to clearly see all the data points

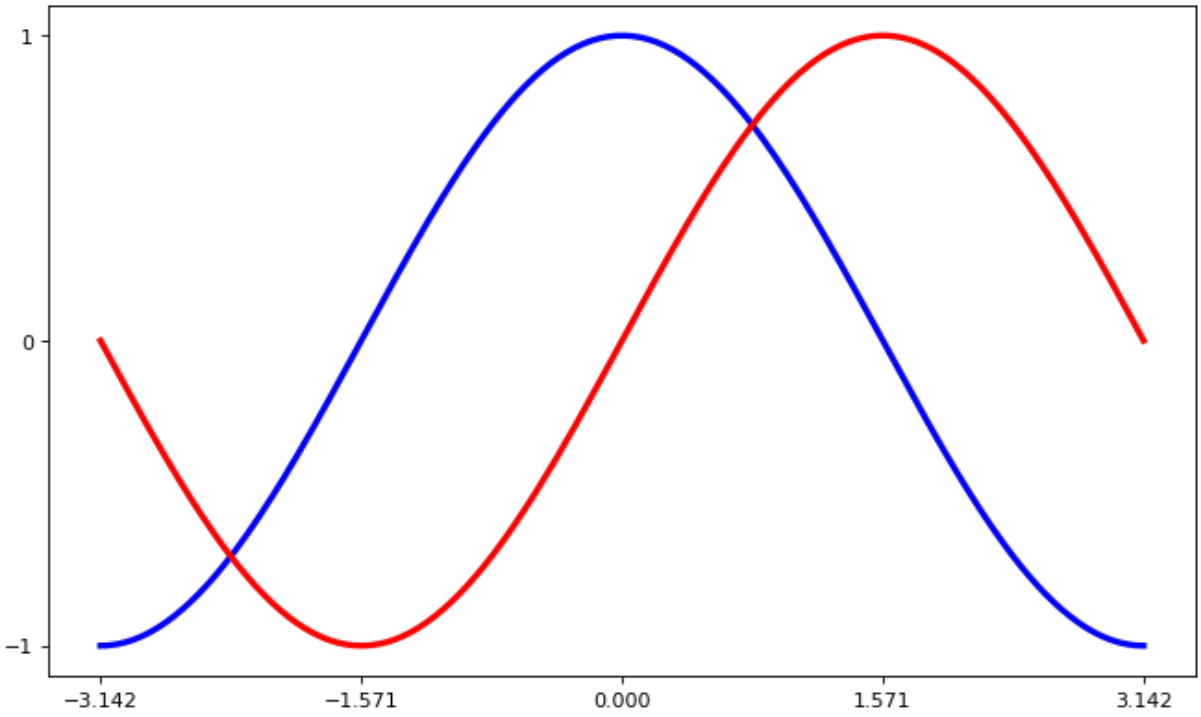
In [21]:
```
plt.figure(figsize=(10,6), dpi = 80)
plt.plot(X,C, color = 'blue', linewidth = 3, linestyle = '-')
plt.plot(X,S, color = 'red', linewidth = 3, linestyle = '-')

plt.xlim(X.min()*1.2, X.max()*1.2)
plt.ylim(C.min()*1.2, C.max()*1.2)

plt.show()
```



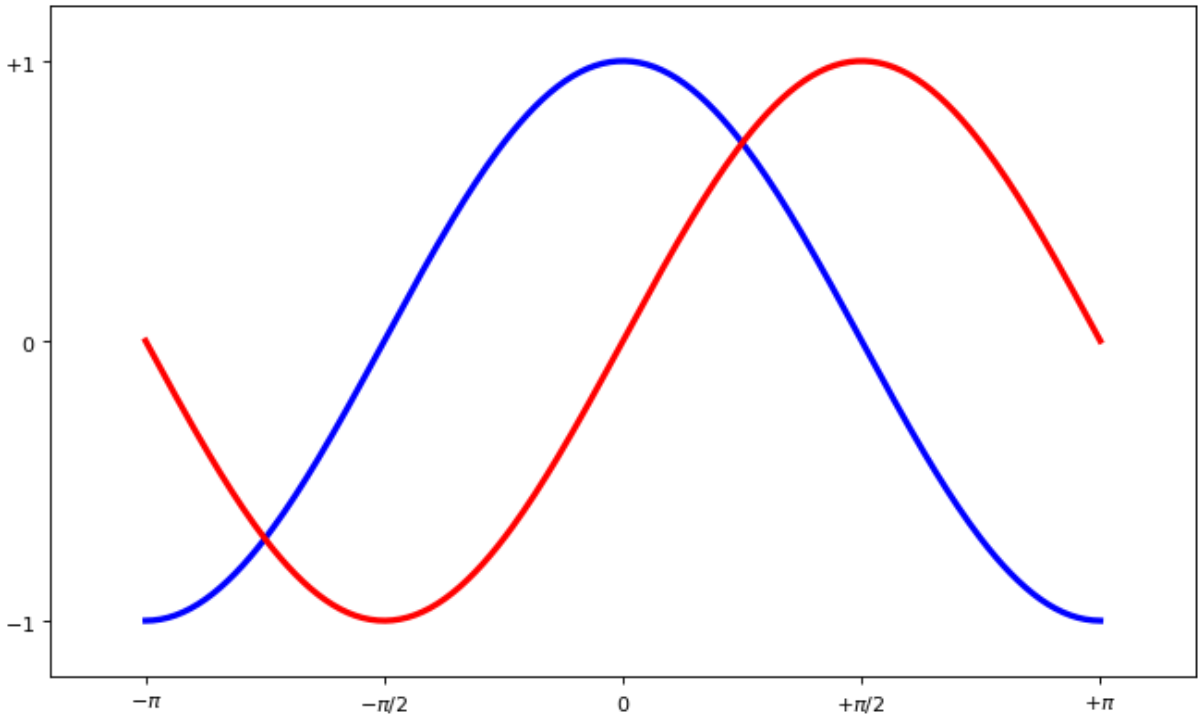
Setting ticks

Current ticks are not ideal because they do not show the interesting values (+/-π,+/-π/2) for sine and cosine. We'll change them such that they show only these values.

In [22]:
```
plt.figure(figsize=(10,6), dpi = 80)
plt.plot(X,C, color = 'blue', linewidth = 3, linestyle = '-')
plt.plot(X,S, color = 'red', linewidth = 3, linestyle = '-')

plt.xticks( [-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, +1])

plt.show()
```



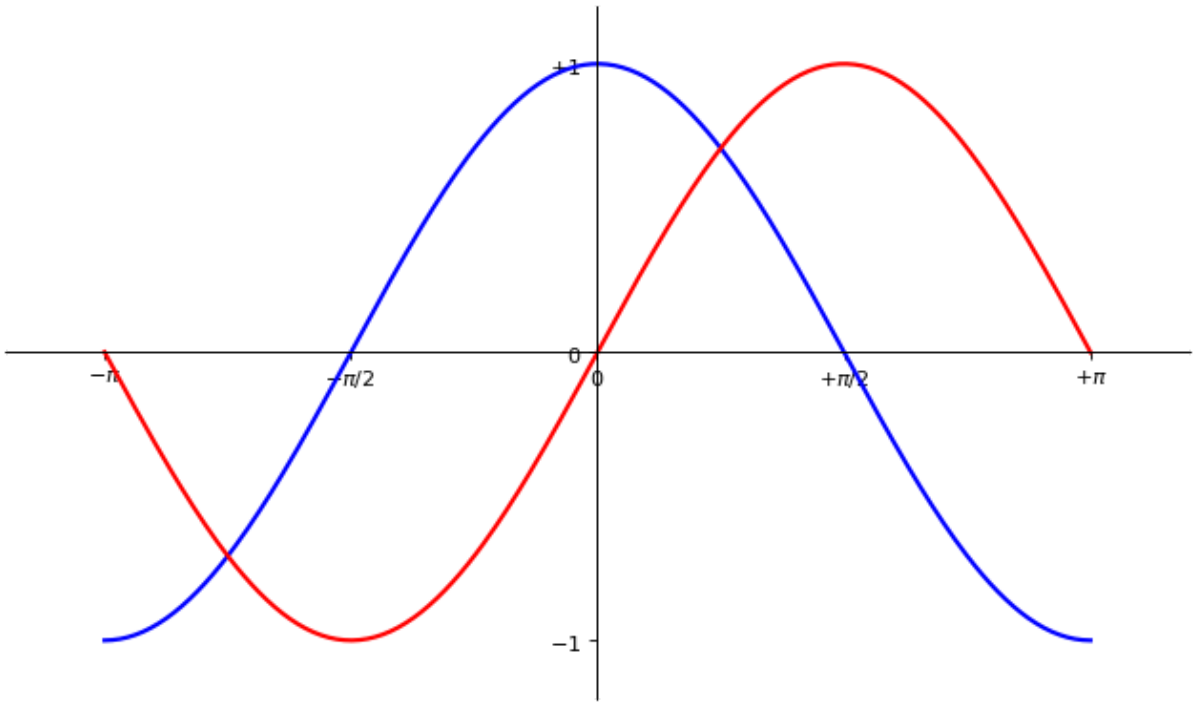
In [23]:
```
plt.figure(figsize=(10,6), dpi = 80)
plt.plot(X,C, color = 'blue', linewidth = 3, linestyle = '-')
plt.plot(X,S, color = 'red', linewidth = 3, linestyle = '-')

plt.xlim(X.min()*1.2, X.max()*1.2)
plt.ylim(C.min()*1.2, C.max()*1.2)

# We want to mention the pi values instead of numerical values in the x-axis
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       ['$-\pi$', '$-\pi/2$', '$0$', '$+\pi/2$', '$+\pi$'])

plt.yticks([-1, 0, +1],
       ['$-1$', '$0$', '$+1$'])

plt.show()
```



Moving spine/ axes

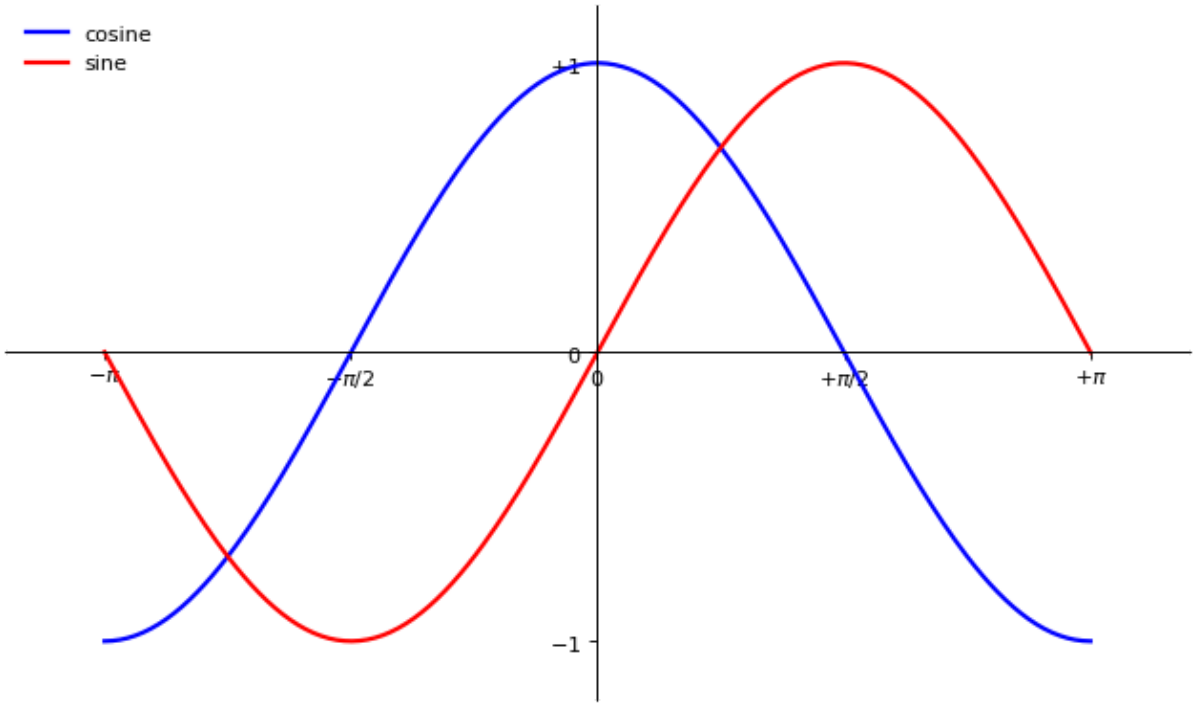
Spines are the lines connecting the axis tick marks and noting the boundaries of the data area. They can be placed at arbitrary positions and until now, they were on the border of the axis. We'll change that since we want to have them in the middle. Since there are four of them (top/bottom/left/right), we'll discard the top and right by setting their color to none and we'll move the bottom and left ones to coordinate 0 in data space coordinates.

In [24]:
```
plt.figure(figsize=(10,6), dpi = 80)
plt.plot(X,C, color = 'blue', linewidth = 2, linestyle = '-')
plt.plot(X,S, color = 'red', linewidth = 2, linestyle = '-')

plt.xlim(X.min()*1.2, X.max()*1.2)
plt.ylim(C.min()*1.2, C.max()*1.2)

# We want to mention the pi values instead of numerical values in the x-axis
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       ['$-\pi$', '$-\pi/2$', '$0$', '$+\pi/2$', '$+\pi$'])

plt.yticks([-1, 0, +1],
       ['$-1$', '$0$', '$+1$'])

ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

#ax.spines['right'].set_visible(False)
#ax.spines['top'].set_visible(False)

ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

plt.show()
```

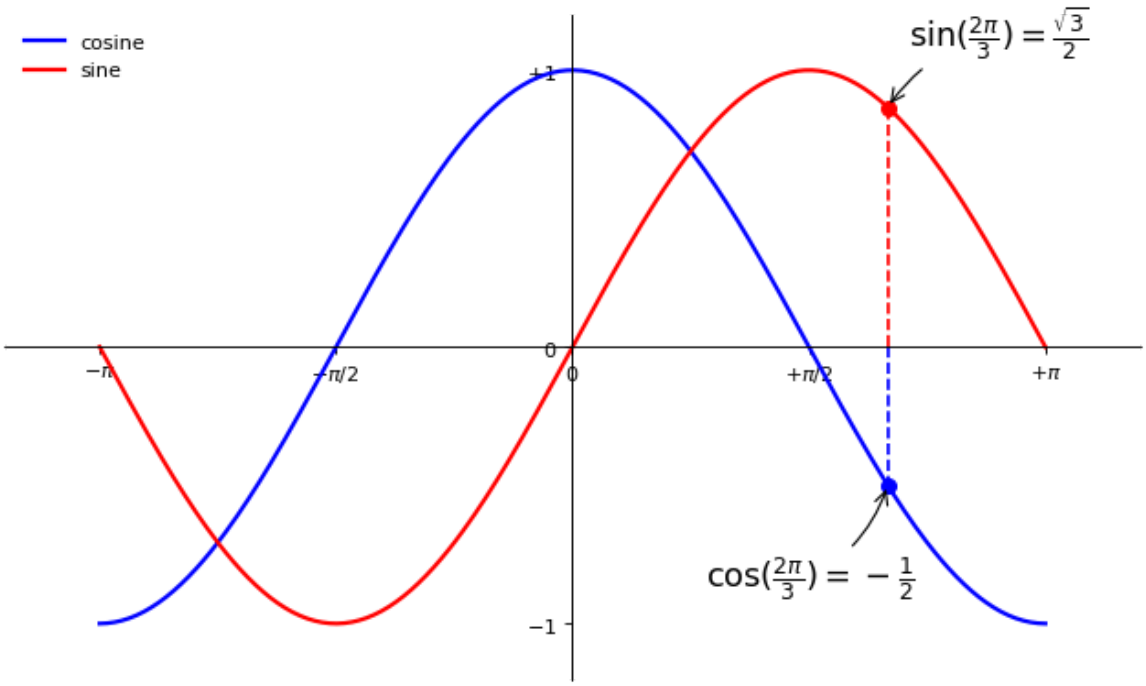


Adding legends

In [25]:
```
plt.figure(figsize=(10,6), dpi = 80)
plt.plot(X,C, color = 'blue', linewidth = 2, linestyle = '-', label = 'cosine')
plt.plot(X,S, color = 'red', linewidth = 2, linestyle = '-', label = 'sine')

plt.xlim(X.min()*1.2, X.max()*1.2)
plt.ylim(C.min()*1.2, C.max()*1.2)

# We want to mention the pi values instead of numerical values in the x-axis
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       ['$-\pi$', '$-\pi/2$', '$0$', '$+\pi/2$', '$+\pi$'])

plt.yticks([-1, 0, +1],
       ['$-1$', '$0$', '$+1$'])

ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

#ax.spines['right'].set_visible(False)
#ax.spines['top'].set_visible(False)

ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

plt.legend(loc='upper left', frameon = False)

plt.show()
```



In [26]:
```
plt.figure(figsize=(10,6), dpi = 80)
plt.plot(X,C, color = 'blue', linewidth = 2, linestyle = '-', label = 'cosine')
plt.plot(X,S, color = 'red', linewidth = 2, linestyle = '-', label = 'sine')

plt.xlim(X.min()*1.2, X.max()*1.2)
plt.ylim(C.min()*1.2, C.max()*1.2)

# We want to mention the pi values instead of numerical values in the x-axis
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       ['$-\pi$', '$-\pi/2$', '$0$', '$+\pi/2$', '$+\pi$'])

plt.yticks([-1, 0, +1],
       ['$-1$', '$0$', '$+1$'])

ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

#ax.spines['right'].set_visible(False)
#ax.spines['top'].set_visible(False)

ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

t = 2*np.pi/3
plt.plot([t,t],[0,np.cos(t)], color ='blue', linewidth=1.5, linestyle="--")
plt.scatter(t,np.cos(t), 50, color ='blue')

plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
             xy=(t, np.sin(t)), xycoords='data',
             xytext=(+10, +30), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t,t],[0,np.sin(t)], color ='red', linewidth=1.5, linestyle="--")
plt.scatter(t,np.sin(t), 50, color ='red')

plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
             xy=(t, np.cos(t)), xycoords='data',
             xytext=(-90, -50), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.legend(loc='upper left', frameon = False)

plt.show()
```



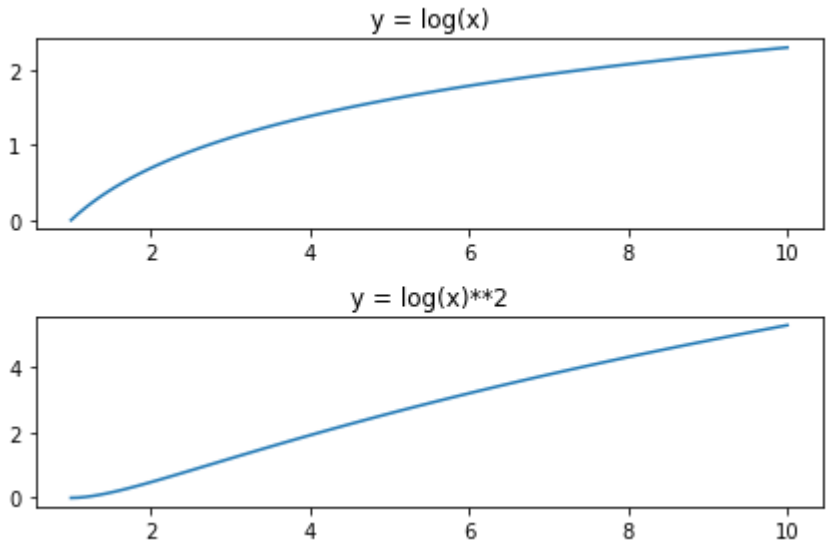
Figures and Subplots

You often need to create multiple plots in the same figure, as we'll see in some upcoming examples.

`matplotlib` has the concept of **figures and subplots** using which you can create *multiple subplots inside the same figure*.

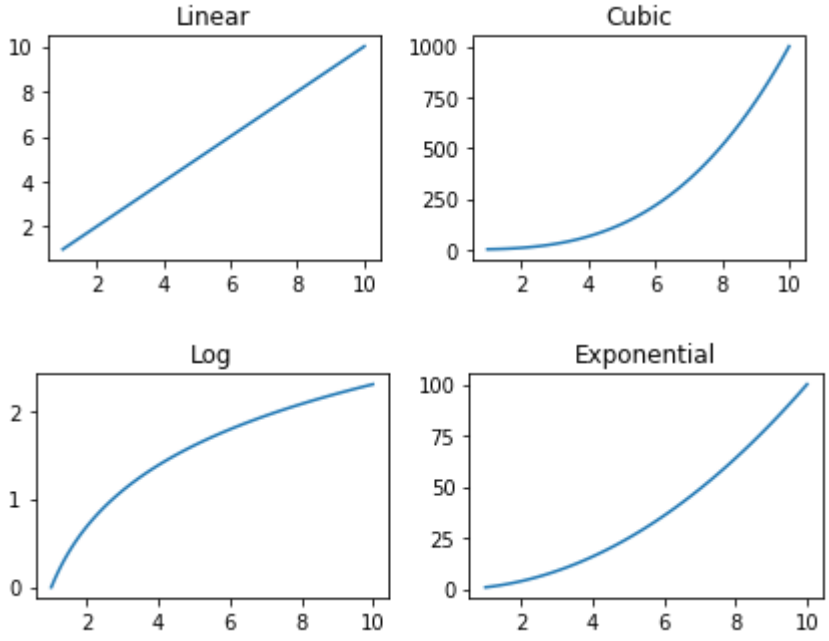
To create multiple plots in the same figure, you can use the method `plt.subplot(nrows, ncols, nsubplot)` .

In [27]:
```
x = np.linspace(1, 10, 100)
y = np.log(x)

# initiate a new figure explicitly
plt.figure(1)

# Create a subplot with 1 row, 2 columns

# create the first subplot in figure 1
plt.subplot(2,1,1)     # equivalent to plt.subplot(1, 2, 1)
plt.title("y = log(x)")
plt.plot(x, y)

# create the second subplot in figure 1
plt.subplot(212)
plt.title("y = log(x)**2")
plt.plot(x, y**2)

plt.tight_layout()
plt.show()
```



Let's see another example - say you want to create 4 subplots in two rows and two columns.

In [28]:
```
# Example: Create a figure having 4 subplots
x = np.linspace(1, 10, 100)

# Optional command, since matplotlib creates a figure by default anyway
plt.figure(1)

# subplot 1
plt.subplot(2, 2, 1)
plt.title("Linear")
plt.plot(x, x)


# subplot 2
plt.subplot(2, 2, 2)
plt.title("Cubic")
plt.plot(x, x**3)

plt.tight_layout()

# subplot 3
plt.figure(2)
plt.subplot(2, 2, 1)
plt.title("Log")
plt.plot(x, np.log(x))

# subplot 4
plt.subplot(2, 2, 2)
plt.title("Exponential")
plt.plot(x, x**2)

plt.tight_layout()
plt.show()
```

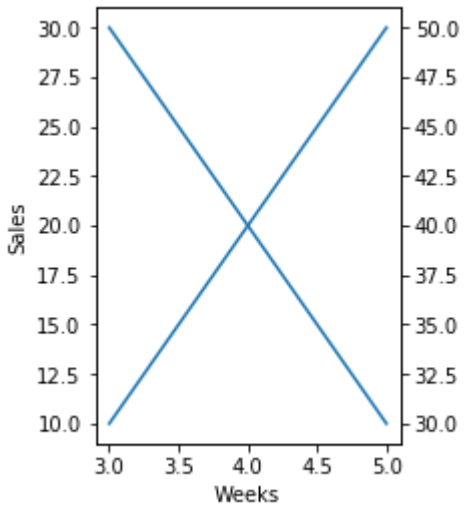


You can see the list of colors and shapes here:
https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot

(https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot)

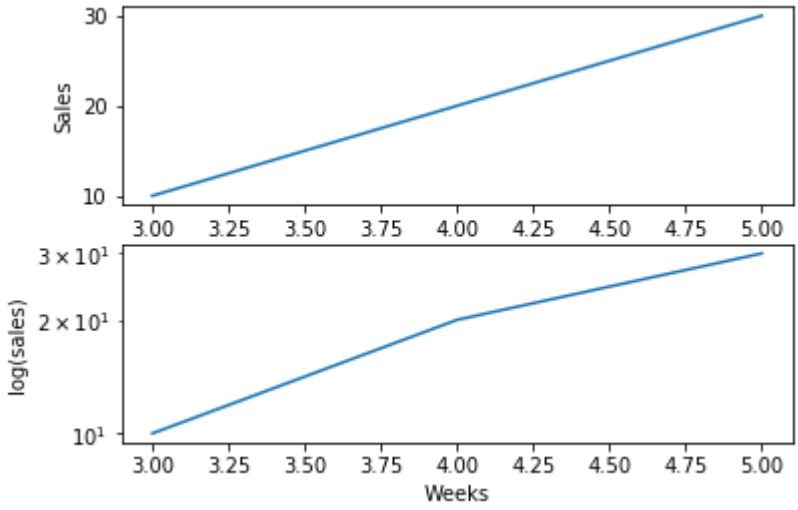
Dual Y Axis

In [29]:
```
fig = plt.figure()

plot1 = fig.add_subplot(1, 2, 1)
plot1.plot([3, 4, 5], [10, 20, 30])
plot1.set_xlabel("Weeks")
plot1.set_ylabel("Sales")

plot2 = plot1.twinx()
plot2.plot([3, 4, 5], [50, 40, 30])
plt.show()
```



Logarithmic Axis

In [30]:
```
fig = plt.figure()

plot1 = fig.add_subplot(2, 1, 1)
plot1.plot([3, 4, 5], [10, 20, 30])
plot1.set_xlabel("Weeks")
plot1.set_ylabel("Sales")

plot2 = fig.add_subplot(2, 1, 2)
plot2.plot([3, 4, 5], [10, 20, 30])
plot2.set_ylabel("log(sales)")
plot2.set_xlabel("Weeks")
plot2.set_yscale('log')
plt.show()
```



Types of Commonly Used Plots

Let's now use the retail store's sales data to create some commonly use plots such as:

- Boxplots
- Histograms
- Scatter plots
- Bar plots

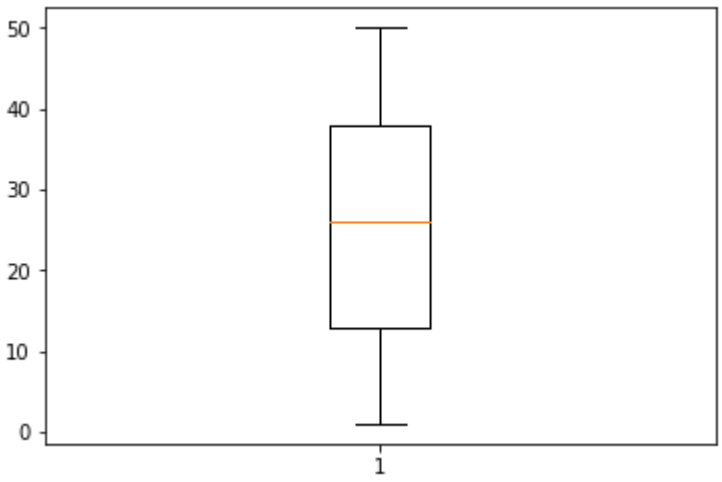
In [31]:
```
# Example: Globals sales data
df = pd.read_csv("market_fact.csv")
df.head()
```

Out[31]:

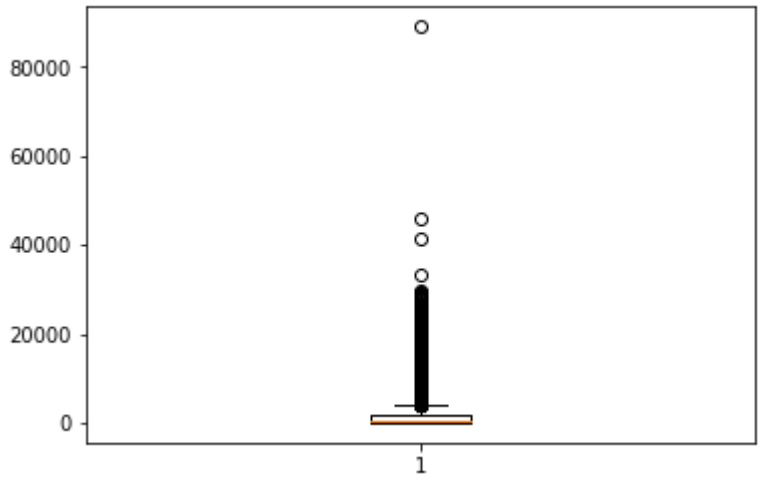
	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	
1	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56	
2	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90	
3	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34	
4	Ord_5485	Prod_17	SHP_7664	Cust_1818	4233.15	0.08	35	1219.87	

Boxplot

In [32]:
```
# Boxplot: Visualise the distribution of a continuous variable
plt.boxplot(df['Order_Quantity'])
plt.show()
```



In [33]:
```
# Boxplot of Sales is quite unreadable, since Sales varies
# across a wide range
plt.boxplot(df['Sales'])
plt.show()
```



As you can see, the boxplot of `Sales` is pretty unreadable, since Sales varies across a wide range as shown below.

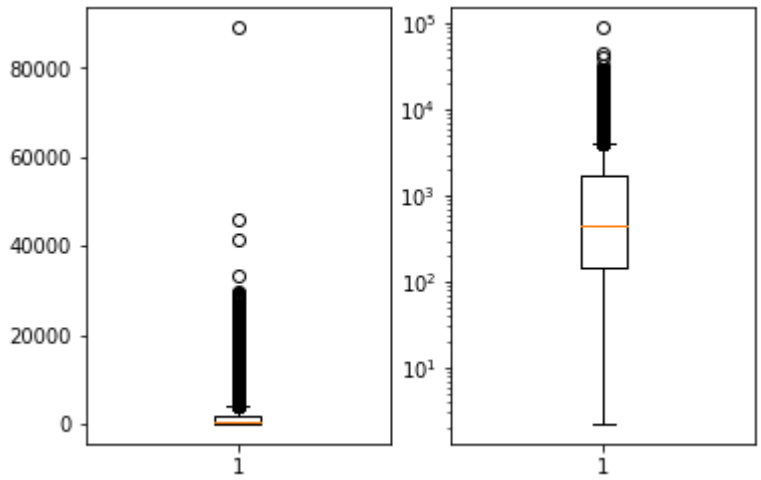
In [34]:
```
# Range of sales: min is 2.24, median is 449, max is 89061
df['Sales'].describe()
```

Out[34]:
```
count     8399.000000
mean      1775.878179
std       3585.050525
min          2.240000
25%        143.195000
50%        449.420000
75%       1709.320000
max      89061.050000
Name: Sales, dtype: float64
```

The solution to this problem is to **change the scale of the axis** (in this case, the y axis) so that the range can fit into the size of the plot.

One commonly used technique is to transform an axis into the **logarithmic scale**. You can transform the scale of an axis using `plt.yscale('log')` .

In [35]:
```
# Usual (linear) scale subplot
plt.subplot(1, 2, 1)
plt.boxplot(df['Sales'])


# Log scale subplot
plt.subplot(1, 2, 2)
plt.boxplot(df['Sales'])
plt.yscale('log')
plt.show()
```

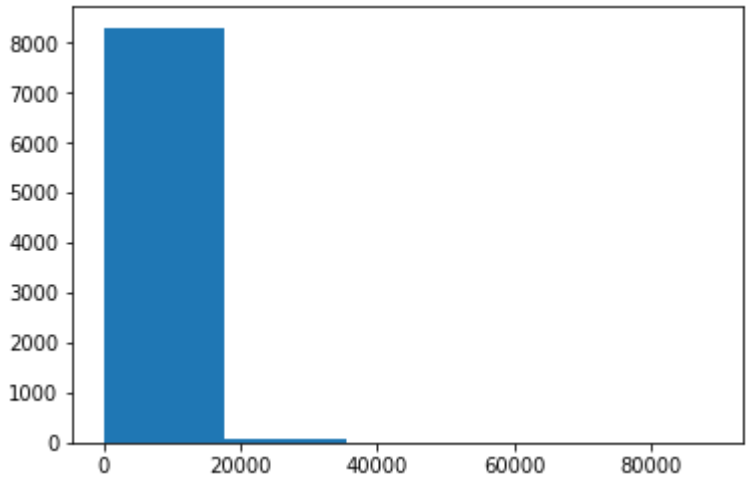


Clearly, the log scale subplot is far more readable - you can infer that the minimum sales is around 0, the median is approximtely in the middle of 100 and 1000, and the max is reaching 100,000.

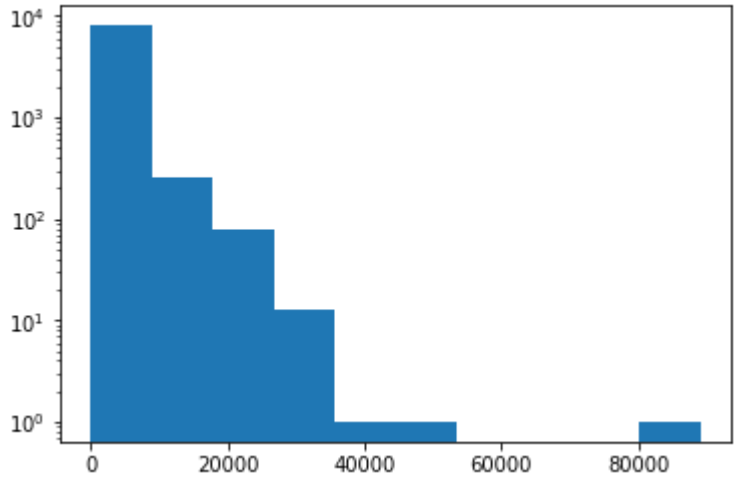
Histogram

Histograms are useful for visualising distribution of single variables.

In [36]:
```
# Histograms
num_bins = 5 #default number of bins is 10 if not specified separately
plt.hist(df['Sales'],num_bins)
plt.show()
```



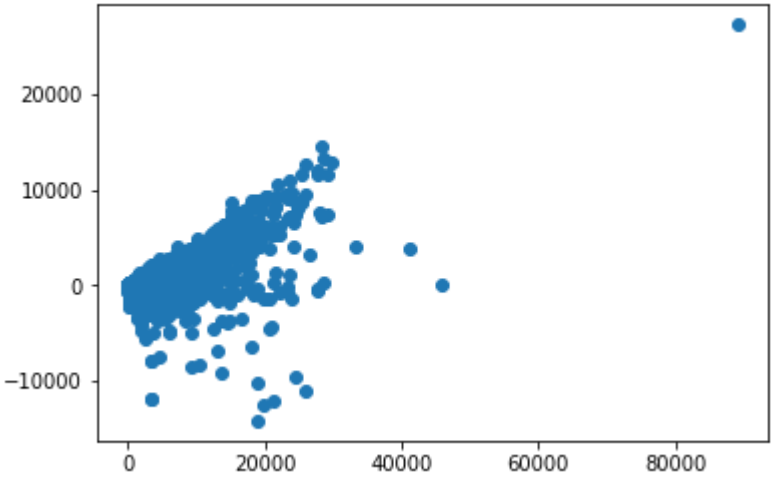
In [37]:
```
# The histogram can be made more readable by using a log scale

plt.hist(df['Sales'])
plt.yscale('log')
plt.show()
```



Scatter Plot

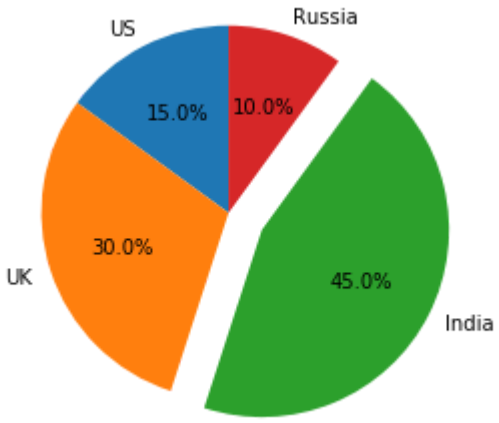
Scatter plots are used to visualise two variables, one on each axis.

In [38]:
```
# Scatter plots with two variables: Profit and Sales
plt.scatter(df['Sales'], df['Profit'])
plt.show()
```



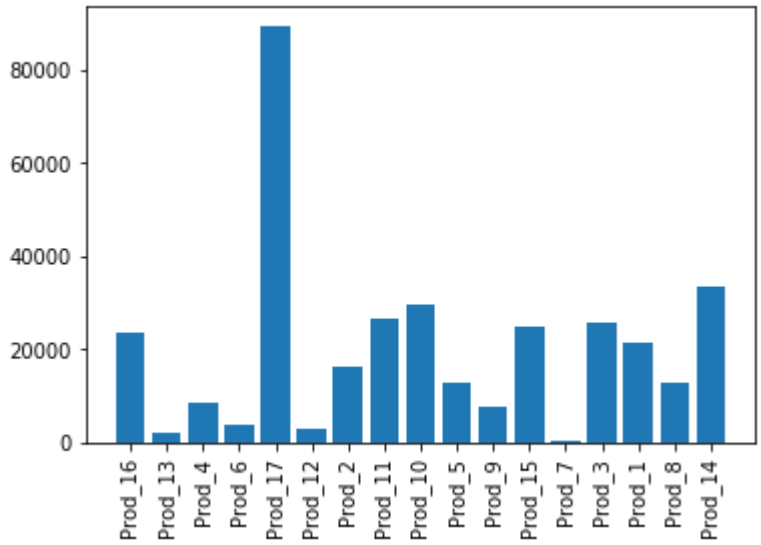
Pie chart

In [39]:
```
# Pie chart, where the slices will be ordered and plotted counter-clockwise

#Prepare data
country_labels = 'US', 'UK', 'India', 'Russia'
counry_sales = [15, 30, 45, 10]
explode = (0, 0, 0.2, 0)  # only "explode" the 2nd slice

fig1, ax1 = plt.subplots()
ax1.pie(counry_sales, explode=explode, labels=country_labels, autopct='%1.1f%%',
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```

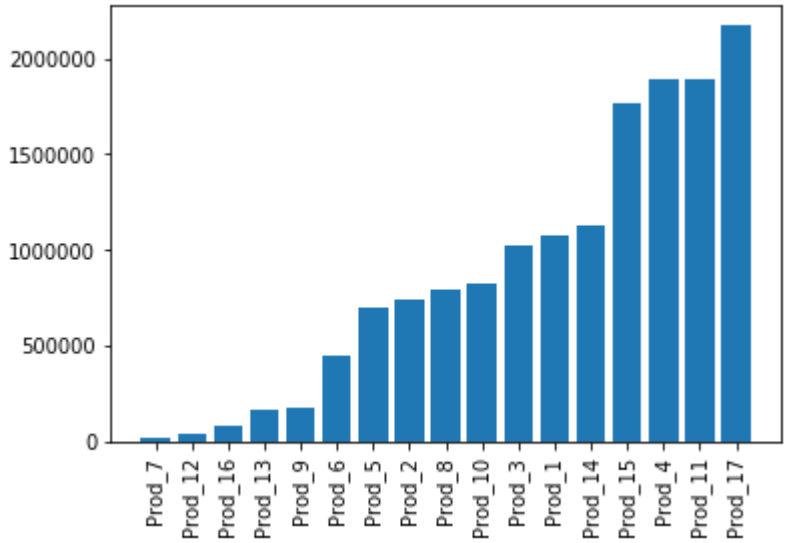


Bar Chart

In [40]:
```
plt.bar(df['Prod_id'], df['Sales']) # to produce a horizontal bar chart, use "bar
plt.xticks(rotation = 90)
plt.show()
```



In [41]:
```
# produce the same graph in ascending order.
result = df.groupby(["Prod_id"])['Sales'].aggregate(np.sum).reset_index().sort_va
plt.bar(result['Prod_id'],result['Sales'])
plt.xticks(rotation = 90)
plt.show()
```



In [42]:
```
df.head()
```

Out[42]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	
1	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56	
2	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90	
3	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34	
4	Ord_5485	Prod_17	SHP_7664	Cust_1818	4233.15	0.08	35	1219.87	

In [43]:
```
df.describe()
```

Out[43]:

	Sales	Discount	Order_Quantity	Profit	Shipping_Cost	Product_Base_Ma
count	8399.000000	8399.000000	8399.000000	8399.000000	8399.000000	8336.000
mean	1775.878179	0.049671	25.571735	181.184424	12.838557	0.512
std	3585.050525	0.031823	14.481071	1196.653371	17.264052	0.135
min	2.240000	0.000000	1.000000	-14140.700000	0.490000	0.350
25%	143.195000	0.020000	13.000000	-83.315000	3.300000	0.380
50%	449.420000	0.050000	26.000000	-1.500000	6.070000	0.520
75%	1709.320000	0.080000	38.000000	162.750000	13.990000	0.590
max	89061.050000	0.250000	50.000000	27220.690000	164.730000	0.850

In [44]:
```
# plotting two variables on the same bar chart
result = df.groupby(["Prod_id"])['Order_Quantity'].aggregate(np.sum).reset_index
result1=df.groupby(["Prod_id"])['Shipping_Cost'].aggregate(np.sum).reset_index()
_x=np.arange(len(result['Prod_id']))
plt.bar(_x-0.1,result['Order_Quantity'],width=0.2, color='b', align='center')
plt.bar(_x+0.1,result1['Shipping_Cost'],width=0.2, color='g', align='center')
plt.xticks(_x,result['Prod_id'],rotation = 90)
plt.yscale('log')
plt.gca().legend(('Order Quantity','Shipping Costs'))
plt.show()
```

A few good resources for practicing matplotlib visualization

1. Official documentation (https://matplotlib.org/users/pyplot_tutorial.html)
2. Matplotlib tutorial showing a variety of plots (https://github.com/rougier/matplotlib-tutorial)
3. Matplotlib reference site (https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot)

Data Visualization Session 13

Visualising Distributions of Data

In this section, we will see how to:

- Visualise univariate distributions
- Visualise bivariate distributions

We will also start using the `seaborn` library for data visualisation. Seaborn is a python library built on top of `matplotlib`. It creates much more attractive plots than `matplotlib`, and is often more concise than `matplotlib` when you want to customize your plots, add colors, grids etc.

Let's start with univariate distributions.

Visualising Univariate Distributions

We have already visualised univariate distributions before using boxplots, histograms etc. Let's now do that using seaborn. We'll use the sales data for the upcoming few exercises.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# the commonly used alias for seaborn is sns
import seaborn as sns

# set a seaborn style of your taste
sns.set_style("whitegrid") #few other commonly used styles are "darkgrid", "white

# data
df = pd.read_csv("./global_sales_data/market_fact.csv")
df.head()
```

Histograms and Density Plots

Histograms and density plots show the frequency of a numeric variable along the y-axis, and the value along the x-axis. The `sns.distplot()` function plots a density curve. Notice that this is aesthetically better than vanilla `matplotlib`.

In [ ]:
```
# simple density plot
sns.distplot(df['Shipping_Cost']) #unlike matplotlib the default bins value is d
plt.show()
```

You can also plot what is known as the **rug plot** which plots the actual data points as small vertical

bars. The rug plot is simply specified as an argument of the `distplot()` .

In [ ]:
```
# rug = True
# plotting only a few points since rug takes a long while
sns.distplot(df['Shipping_Cost'][:200], rug=True)
plt.show()
```

Simple density plot (without the histogram bars) can be created by specifying `hist=False` .

In [ ]:
```
sns.distplot(df['Sales'], hist=False)
plt.show()
```

Since seaborn uses matplotlib behind the scenes, the usual matplotlib functions work well with seaborn. For example, you can use subplots to plot multiple univariate distributions.

In [ ]:
```
# subplots

# subplot 1
plt.subplot(2, 2, 1)
#plt.title('Sales')
sns.distplot(df['Sales'])

# subplot 2
plt.subplot(2, 2, 2)
#plt.title('Profit')
sns.distplot(df['Profit'])

# subplot 3
plt.subplot(2, 2, 3)
#plt.title('Order Quantity')
sns.distplot(df['Order_Quantity'])

# subplot 4
plt.subplot(2, 2, 4)
#plt.title('Shipping Cost')
sns.distplot(df['Shipping_Cost'])
plt.tight_layout()
plt.show()
```

Boxplots

Boxplots are a great way to visualise univariate data because they represent statistics such as the 25th percentile, 50th percentile, etc.

In [ ]:
```
# boxplot
sns.boxplot(df['Order_Quantity'])
#plt.title('Order Quantity')

plt.show()
```

In [ ]:
```
# to plot the values on the vertical axis, specify y=variable
sns.boxplot(y=df['Order_Quantity'])
plt.title('Order Quantity')
plt.show()
```

Visualising Bivariate Distributions

Bivariate distributions are simply two univariate distributions plotted on x and y axes respectively. They help you observe the relationship between the two variables.

They are also called joint distributions and are created using `sns.jointplot()` .

In [ ]:
```
# joint plots of Profit and Sales

sns.jointplot('Sales', 'Profit', df)
plt.show()

# same as sns.jointplot(df['Sales'], df['Profit'])
```

Notice that both the distributions are heavily skewed and all the points seem to be concentrated in one region. That is because of some extreme values of Profits and Sales which matplotlib is trying to accomodate in the limited space of the plot.

Let's remove that point and plot again.

In [ ]:
```
# remove points having extreme values
df = df[(df.Profit < 100) & (df.Sales < 200)]

sns.jointplot('Sales', 'Profit', df, kind="scatter", space = 0, color="b")
plt.show()
```

You can adjust the arguments of the jointplot() to make the plot more readable. For e.g. specifying `kind=hex` will create a 'hexbin plot'.

In [ ]:
```
# plotting low Sales value orders
# hex plot
df = pd.read_csv("./global_sales_data/market_fact.csv")
df = df[(df.Profit < 100) & (df.Profit > -100) & (df.Sales < 200)]
sns.jointplot('Sales', 'Profit', df, kind="hex", color="k")
plt.show()
```

The bottom-right region of the plot represents orders where the Sales is high but the Profit is low, i.e. even when the store is getting high revenue, the orders are still making losses. These are the kind of orders a business would want to avoid.

Data Visualization Session 14

Plotting Pairwise Relationships

You'll find it helpful to plot pairwise relationships between multiple numeric variables. For e.g., here we have taken the prices of some popular cryptocurrencies such as bitcoin, litecoin, ethereum, monero, neo, quantum and ripple.

Now, the crypto enthusiasts would know that the prices of these currencies vary with each other. If bitcoin goes up, the others will likely follow suit, etc.

Now, say you want to trade in some currencies. Given a set of cryptocurrencies, how will you decide when and which one to buy/sell? It will be helpful to analyse past data and identify some trends in these currencies.

In [14]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

btc = pd.read_csv("crypto_data/bitcoin_price.csv")
ether = pd.read_csv("crypto_data/ethereum_price.csv")
ether.head()
```

Out[14]:

	Date	Open	High	Low	Close	Volume	Market Cap
0	Nov 07, 2017	298.57	304.84	290.77	294.66	5407,66,000	28,533,300,000
1	Nov 06, 2017	296.43	305.42	293.72	298.89	5793,59,000	28,322,700,000
2	Nov 05, 2017	300.04	301.37	295.12	296.26	3376,58,000	28,661,500,000
3	Nov 04, 2017	305.48	305.48	295.80	300.47	4164,79,000	29,175,300,000
4	Nov 03, 2017	288.50	308.31	287.69	305.71	6463,40,000	27,547,400,000

In [15]:
```
# reading cryptocurrency files
btc = pd.read_csv("crypto_data/bitcoin_price.csv")
ether = pd.read_csv("crypto_data/ethereum_price.csv")
ltc = pd.read_csv("crypto_data/litecoin_price.csv")
monero = pd.read_csv("crypto_data/monero_price.csv")
neo = pd.read_csv("crypto_data/neo_price.csv")
quantum = pd.read_csv("crypto_data/qtum_price.csv")
ripple = pd.read_csv("crypto_data/ripple_price.csv")

# putting a suffix with column names so that joins are easy
btc.columns = btc.columns.map(lambda x: str(x) + '_btc')
ether.columns = ether.columns.map(lambda x: str(x) + '_et')
ltc.columns = ltc.columns.map(lambda x: str(x) + '_ltc')
monero.columns = monero.columns.map(lambda x: str(x) + '_mon')
neo.columns = neo.columns.map(lambda x: str(x) + '_neo')
quantum.columns = quantum.columns.map(lambda x: str(x) + '_qt')
ripple.columns = ripple.columns.map(lambda x: str(x) + '_rip')

btc.head()
```

Out[15]:

	Date_btc	Open_btc	High_btc	Low_btc	Close_btc	Volume_btc	Market Cap_btc
0	Nov 07, 2017	7023.10	7253.32	7023.10	7144.38	2,326,340,000	117,056,000,000
1	Nov 06, 2017	7403.22	7445.77	7007.31	7022.76	3,111,900,000	123,379,000,000
2	Nov 05, 2017	7404.52	7617.48	7333.19	7407.41	2,380,410,000	123,388,000,000
3	Nov 04, 2017	7164.48	7492.86	7031.28	7379.95	2,483,800,000	119,376,000,000
4	Nov 03, 2017	7087.53	7461.29	7002.94	7207.76	3,369,860,000	118,084,000,000

In [19]:
```
# merging all the files by date
m1 = pd.merge(btc, ether, how="inner", left_on="Date_btc", right_on="Date_et")
m2 = pd.merge(m1, ltc, how="inner", left_on="Date_btc", right_on="Date_ltc")
m3 = pd.merge(m2, monero, how="inner", left_on="Date_btc", right_on="Date_mon")
m4 = pd.merge(m3, neo, how="inner", left_on="Date_btc", right_on="Date_neo")
m5 = pd.merge(m4, quantum, how="inner", left_on="Date_btc", right_on="Date_qt")
crypto = pd.merge(m5, ripple, how="inner", left_on="Date_btc", right_on="Date_ri

crypto.head()
```

Out[19]:

	Date_btc	Open_btc	High_btc	Low_btc	Close_btc	Volume_btc	Market Cap_btc	Date_et	Ope
0	Nov 07, 2017	7023.10	7253.32	7023.10	7144.38	2,326,340,000	117,056,000,000	Nov 07, 2017	29
1	Nov 06, 2017	7403.22	7445.77	7007.31	7022.76	3,111,900,000	123,379,000,000	Nov 06, 2017	29
2	Nov 05, 2017	7404.52	7617.48	7333.19	7407.41	2,380,410,000	123,388,000,000	Nov 05, 2017	30
3	Nov 04, 2017	7164.48	7492.86	7031.28	7379.95	2,483,800,000	119,376,000,000	Nov 04, 2017	30
4	Nov 03, 2017	7087.53	7461.29	7002.94	7207.76	3,369,860,000	118,084,000,000	Nov 03, 2017	28

5 rows × 49 columns

In [26]:
```
# Subsetting only the closing prices column for plotting
clos = crypto[["Close_btc", "Close_et", 'Close_ltc', "Close_mon","Close_neo", "C
clos.head()
```

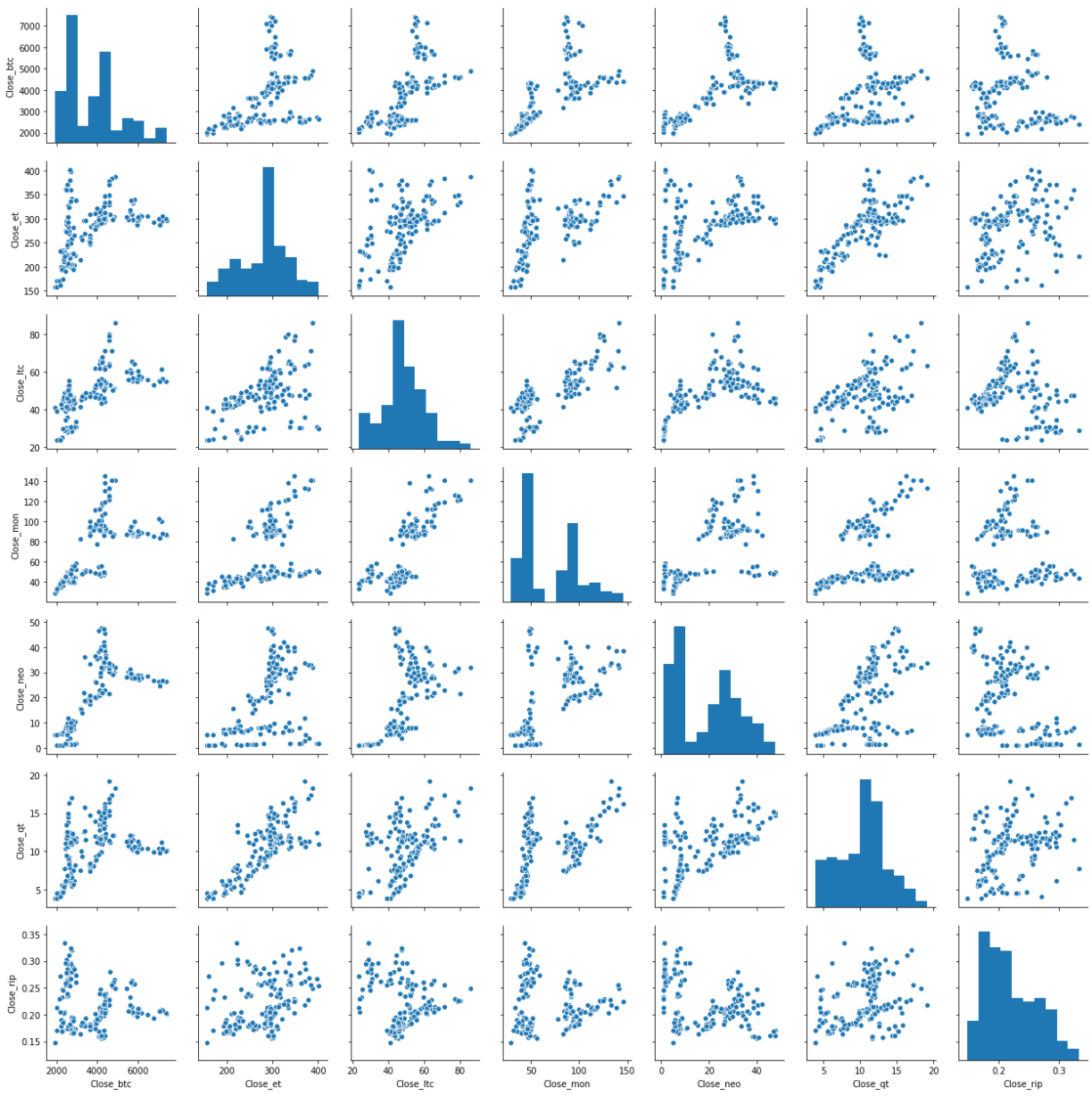
Out[26]:

	Close_btc	Close_et	Close_ltc	Close_mon	Close_neo	Close_qt	Close_rip
0	7144.38	294.66	61.30	99.76	26.23	11.21	0.210354
1	7022.76	298.89	55.17	102.92	26.32	10.44	0.205990
2	7407.41	296.26	54.75	86.35	26.38	10.13	0.202055
3	7379.95	300.47	55.04	87.30	26.49	10.05	0.203750
4	7207.76	305.71	56.18	87.99	26.82	10.38	0.208133

Pairwise Scatter Plots

Now, since we have multiple numeric variables, `sns.pairplot()` is a good choice to plot all of them in one figure.

In [27]:
```
# pairplot
sns.pairplot(clos)
plt.show()
```



In [28]:
```
# You can also observe the correlation between the currencies
# using df.corr()
cor = clos.corr()
round(cor, 3)
```

Out[28]:

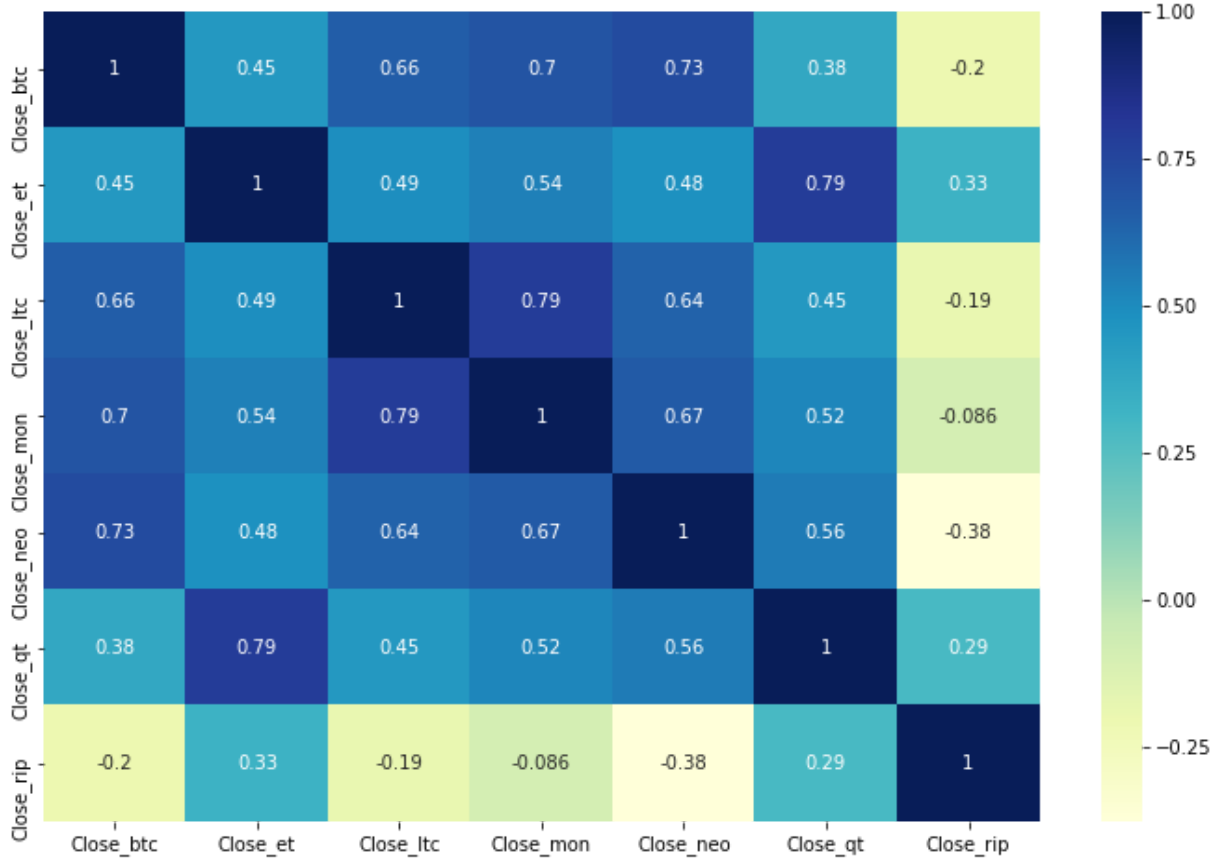
	Close_btc	Close_et	Close_ltc	Close_mon	Close_neo	Close_qt	Close_rip
Close_btc	1.000	0.449	0.658	0.697	0.735	0.382	-0.198
Close_et	0.449	1.000	0.490	0.539	0.482	0.791	0.332
Close_ltc	0.658	0.490	1.000	0.793	0.641	0.448	-0.187
Close_mon	0.697	0.539	0.793	1.000	0.669	0.518	-0.086
Close_neo	0.735	0.482	0.641	0.669	1.000	0.557	-0.375
Close_qt	0.382	0.791	0.448	0.518	0.557	1.000	0.292
Close_rip	-0.198	0.332	-0.187	-0.086	-0.375	0.292	1.000

The dataframe above is a **correlation matrix** of cryptocurrencies. Try finding some important relationships between currencies. Notice that quantum and ethereum are highly correlated (0.79).

Heatmaps

It will be helpful to visualise the correlation matrix itself using `sns.heatmap()` .

In [29]:
```
# figure size
plt.figure(figsize=(12,8))

# heatmap
sns.heatmap(cor, cmap="YlGnBu", annot=True)
plt.show()
```



The orange boxes show the most correlated currencies. Specifically, **ethereum-quantum (0.79)** and **monero-ltc (0.79)** are the most correlated pairs. Also, **neo-btc (0.73)** are quite highly correlated.

Please note that this data is from a specific time period only.

Thus, from a risk-minimisation point of view, you should not invest in these pairs of cryptocurrencies, since if one goes down, the other is likely to go down as well (but yes, if one goes up, you'll become filthy rich).

Data Visualization Session 13

Plotting Categorical Data

In this section, we will:

- Plot distributions of data across categorical variables
- Plot aggregate/summary statistics across categorical variables

Plotting Distributions Across Categories

We have seen how to plot distributions of data. Often, the distributions reveal new information when you plot them across categorical variables.

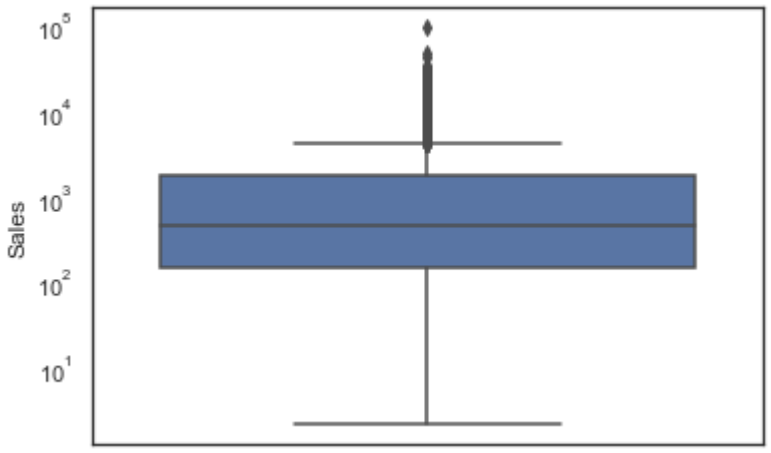
Let's see some examples.

In [31]:
```
# loading libraries and reading the data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# set seaborn theme if you prefer
sns.set(style="white")

# read data
market_df = pd.read_csv("./global_sales_data/market_fact.csv")
customer_df = pd.read_csv("./global_sales_data/cust_dimen.csv")
product_df = pd.read_csv("./global_sales_data/prod_dimen.csv")
shipping_df = pd.read_csv("./global_sales_data/shipping_dimen.csv")
orders_df = pd.read_csv("./global_sales_data/orders_dimen.csv")
```

Boxplots

We had created simple boxplots such as the ones shown below. Now, let's plot multiple boxplots and see what they can tell us the distribution of variables across categories.

In [32]:
```
# boxplot of a variable
sns.boxplot(y=market_df['Sales'])
plt.yscale('log')
plt.show()
```



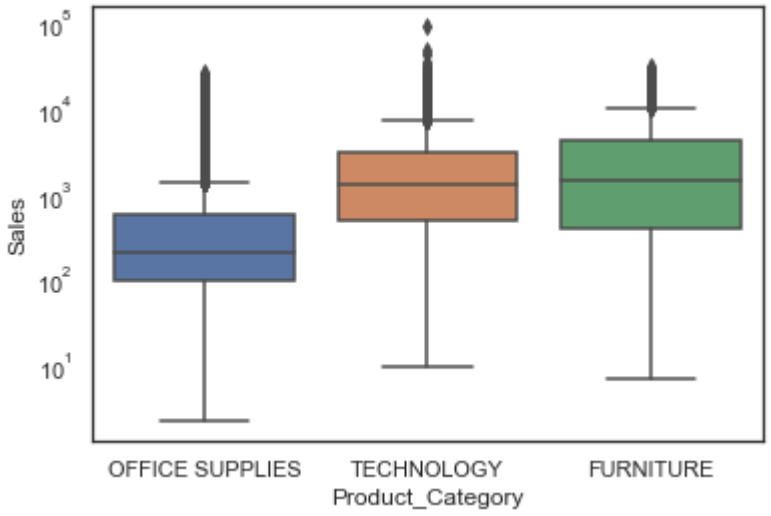
Now, let's say you want to **compare the (distribution of) sales of various product categories**. Let's first merge the product data into the main dataframe.

In [33]:
```
# merge the dataframe to add a categorical variable
df = pd.merge(market_df, product_df, how='inner', on='Prod_id')
df.head()
```

Out[33]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping_C
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	3
1	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.12	3
2	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.58	3
3	Ord_3730	Prod_16	SHP_5175	Cust_1314	459.08	0.04	34	61.57	3
4	Ord_4143	Prod_16	SHP_5771	Cust_1417	207.21	0.06	24	-78.64	6

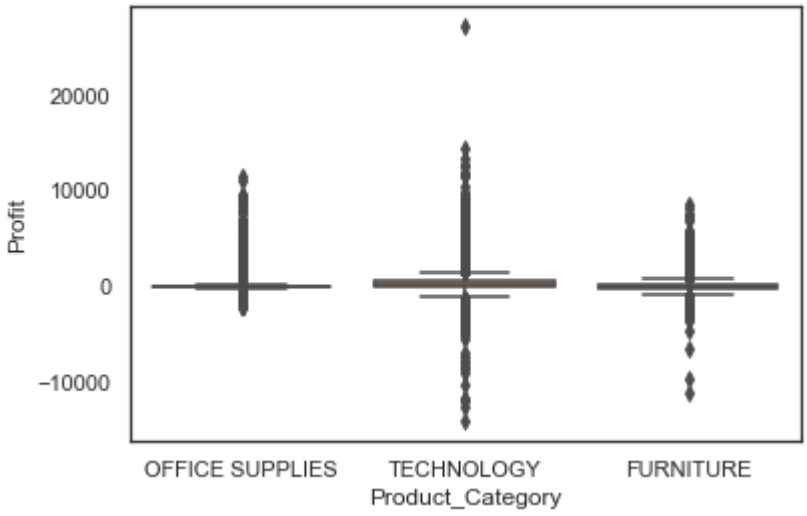
In [34]:
```
# boxplot of a variable across various product categories
sns.boxplot(x='Product_Category', y='Sales', data=df)
plt.yscale('log')
plt.show()
```



So this tells you that the sales of office supplies are, on an average, lower than the other two categories. The sales of technology and furniture categories seem much better. Note that each order can have multiple units of products sold, so Sales being higher/lower may be due to price per unit or the number of units.

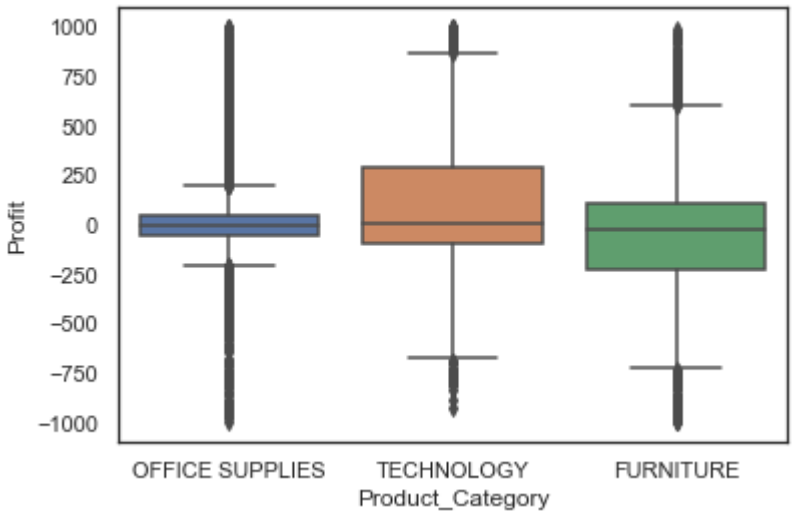
Let's now plot the other important variable - Profit.

In [35]:
```
# boxplot of a variable across various product categories
import seaborn as sns
sns.boxplot(x='Product_Category', y='Profit', data=df)

plt.show()
```



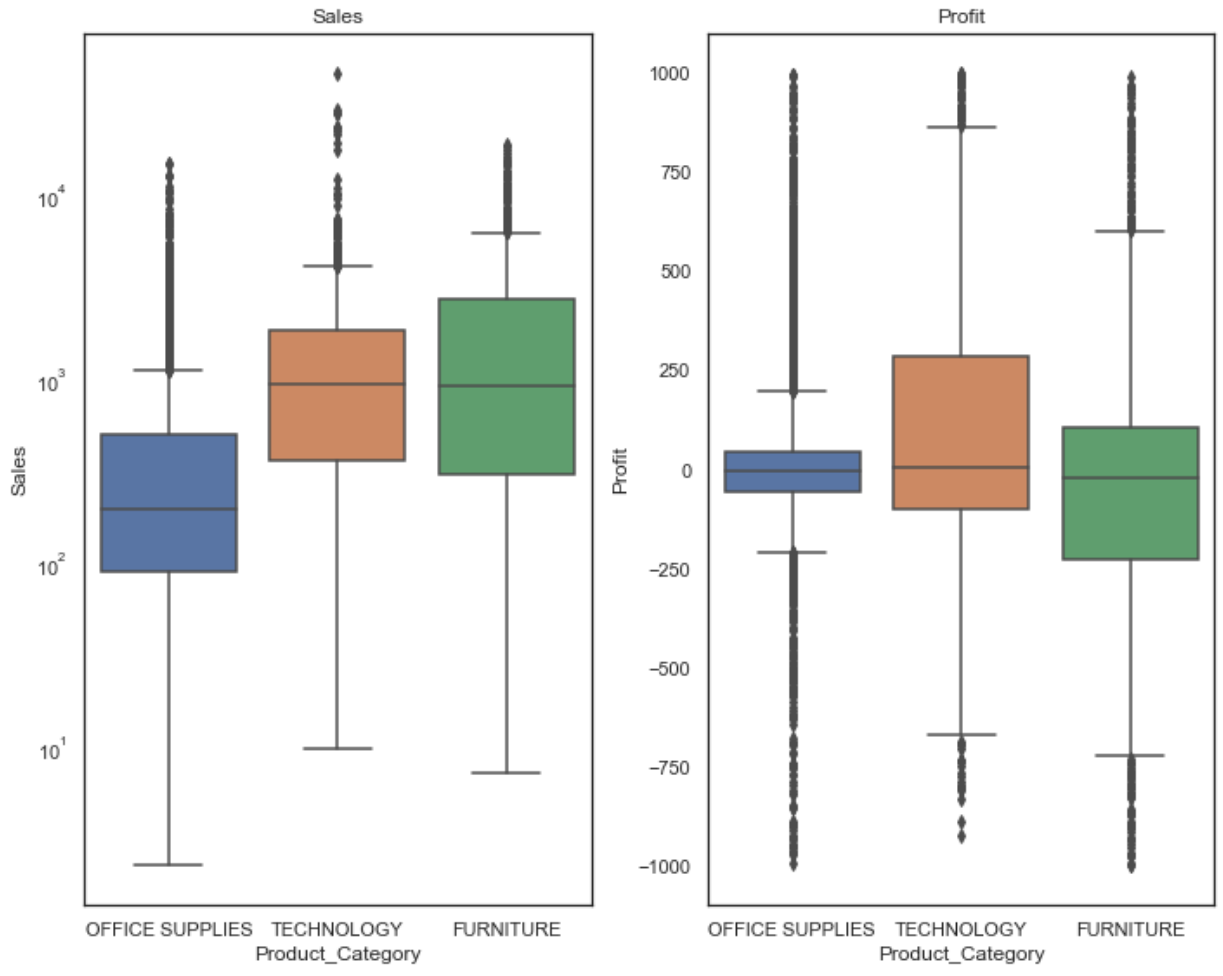
Profit clearly has some *outliers* due to which the boxplots are unreadable. Let's remove some extreme values from Profit (for the purpose of visualisation) and try plotting.

In [36]:
```
df = df[(df.Profit<1000) & (df.Profit>-1000)]

# boxplot of a variable across various product categories
sns.boxplot(x='Product_Category', y='Profit', data=df)
plt.show()
```



You can see that though the category 'Technology' has better sales numbers than others, it is also the one where the **most loss making transactions** happen. You can drill further down into this.

In [37]:
```
# adjust figure size
plt.figure(figsize=(10, 8))

# subplot 1: Sales
plt.subplot(1, 2, 1)
sns.boxplot(x='Product_Category', y='Sales', data=df)
plt.title("Sales")
plt.yscale('log')

# subplot 2: Profit
plt.subplot(1, 2, 2)
sns.boxplot(x='Product_Category', y='Profit', data=df)
plt.title("Profit")

plt.tight_layout()
plt.show()
```



Now that we've compared Sales and Profits across product categories, let's drill down further and do the same across **another categorical variable** - Customer_Segment.

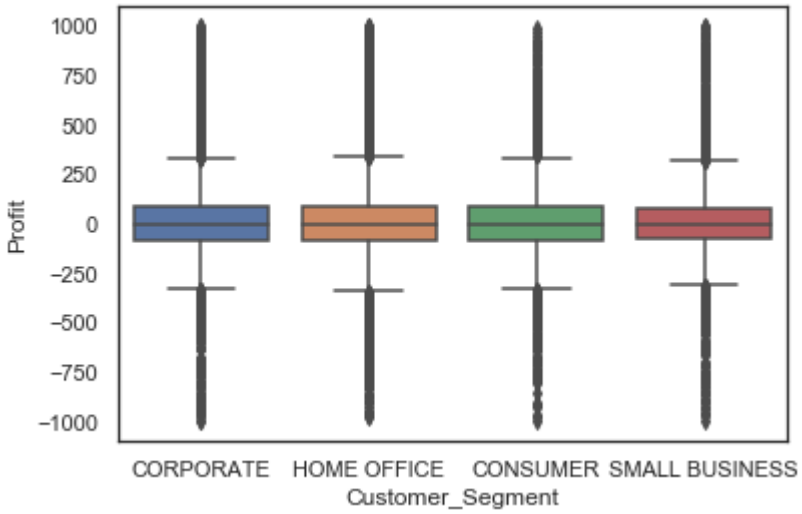
We'll need to add the customer-related attributes (dimensions) to this dataframe.

In [38]:
```
# merging with customers df
df = pd.merge(df, customer_df, how='inner', on='Cust_id')
df.head()
```

Out[38]:

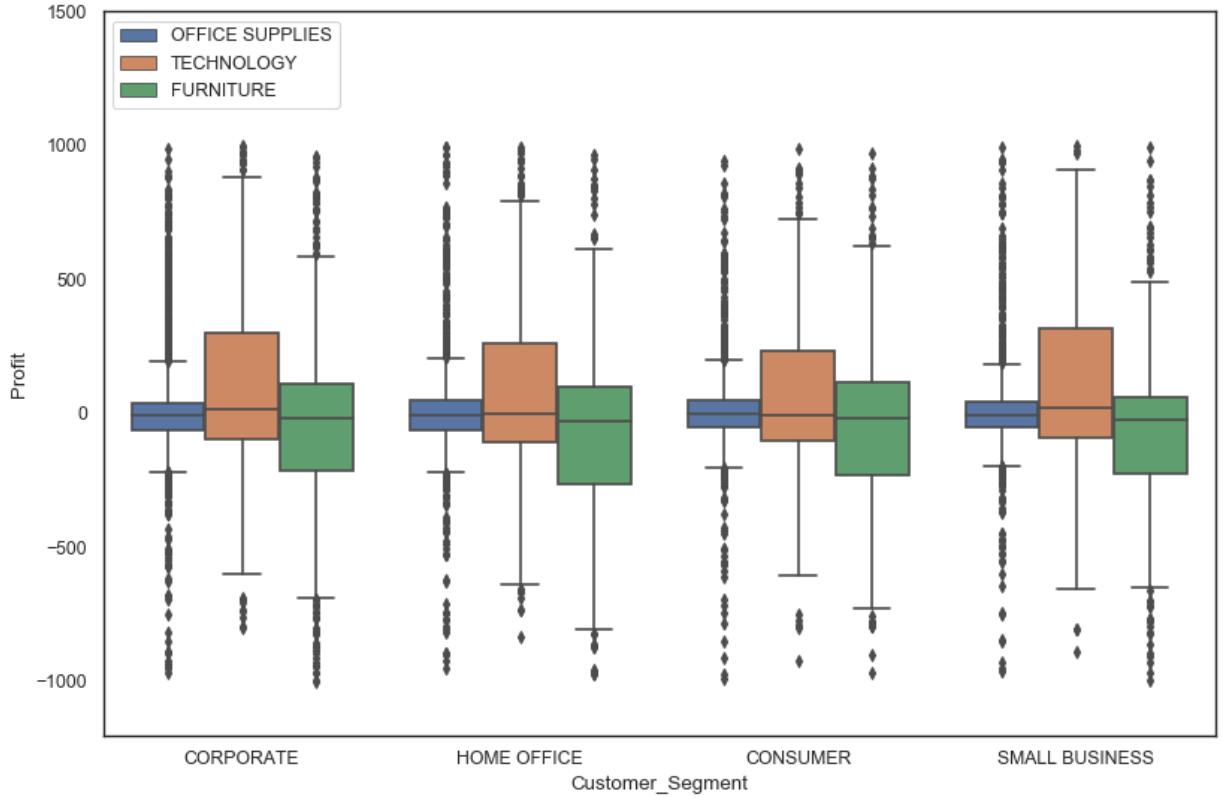
	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping_
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	
1	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56	
2	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34	1
3	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64	
4	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.12	

In [39]:
```
# boxplot of a variable across various product categories
sns.boxplot(x='Customer_Segment', y='Profit', data=df)
plt.show()
```



You can **visualise the distribution across two categorical variables** using the `hue=` argument.

In [40]:
```
# set figure size for larger figure
plt.figure(num=None, figsize=(12, 8), dpi=80, facecolor='w', edgecolor='k')

# specify hue="categorical_variable"
sns.boxplot(x='Customer_Segment', y='Profit', hue="Product_Category", data=df)
plt.legend(loc='upper left')
plt.ylim(-1200,1500)
plt.show()
```

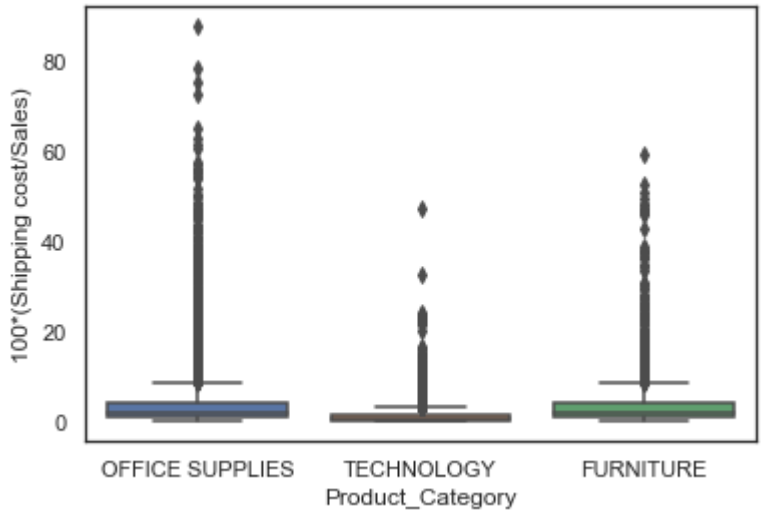


Across all customer segments, the product category `Technology` seems to be doing fairly well, though `Furniture` is incurring losses across all segments.

Now say you are curious to know why certain orders are making huge losses. One of your hypothesis is that the *shipping cost is too high in some orders*. You can **plot derived variables** as well, such as *shipping cost as percentage of sales amount*.

Data Visualization Session 14

In [41]:
```
# plot shipping cost as percentage of Sales amount
sns.boxplot(x=df['Product_Category'], y=100*df['Shipping_Cost']/df['Sales'])
plt.ylabel("100*(Shipping cost/Sales)")
plt.show()
```



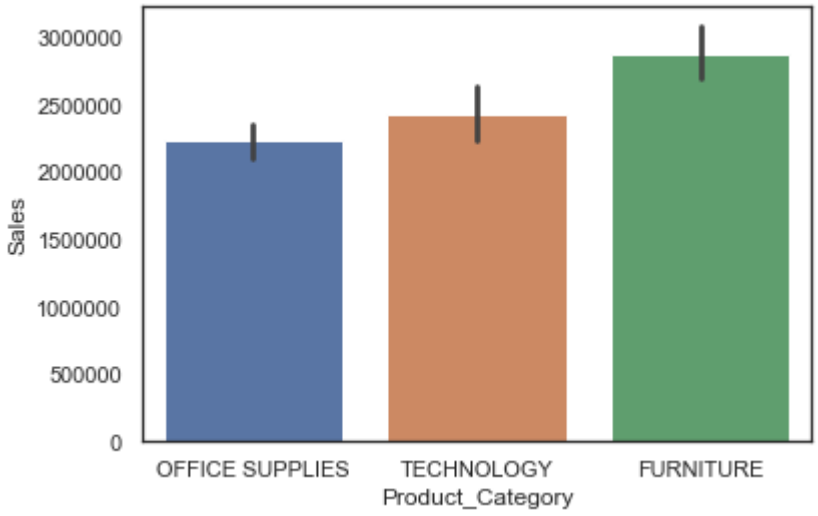
Plotting Aggregated Values across Categories

Bar Plots - Mean, Median and Count Plots

Bar plots are used to **display aggregated values** of a variable, rather than entire distributions. This is especially useful when you have a lot of data which is difficult to visualise in a single figure.

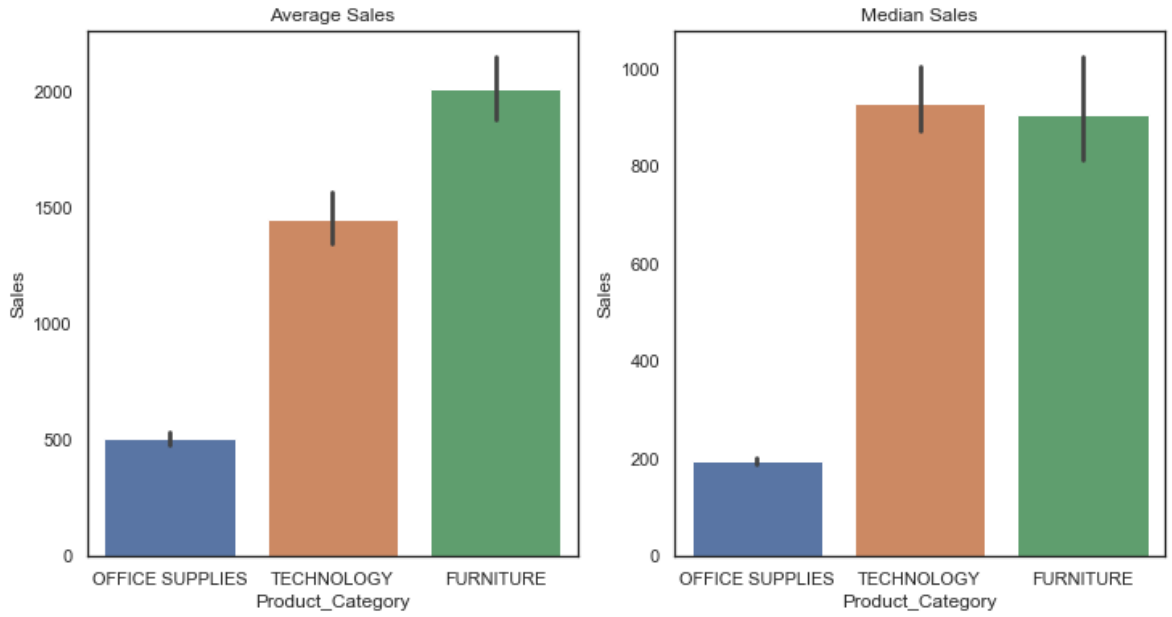
For example, say you want to visualise and *compare the average Sales across Product Categories*. The `sns.barplot()` function can be used to do that.

In [45]:
```
# bar plot with default statistic=mean
sns.barplot(x='Product_Category', y='Sales', data=df, estimator = np.sum)
plt.show()
```



Note that, **by default, seaborn plots the mean value across categories**, though you can plot the count, median, sum etc. Also, barplot computes and shows the confidence interval of the mean as well.

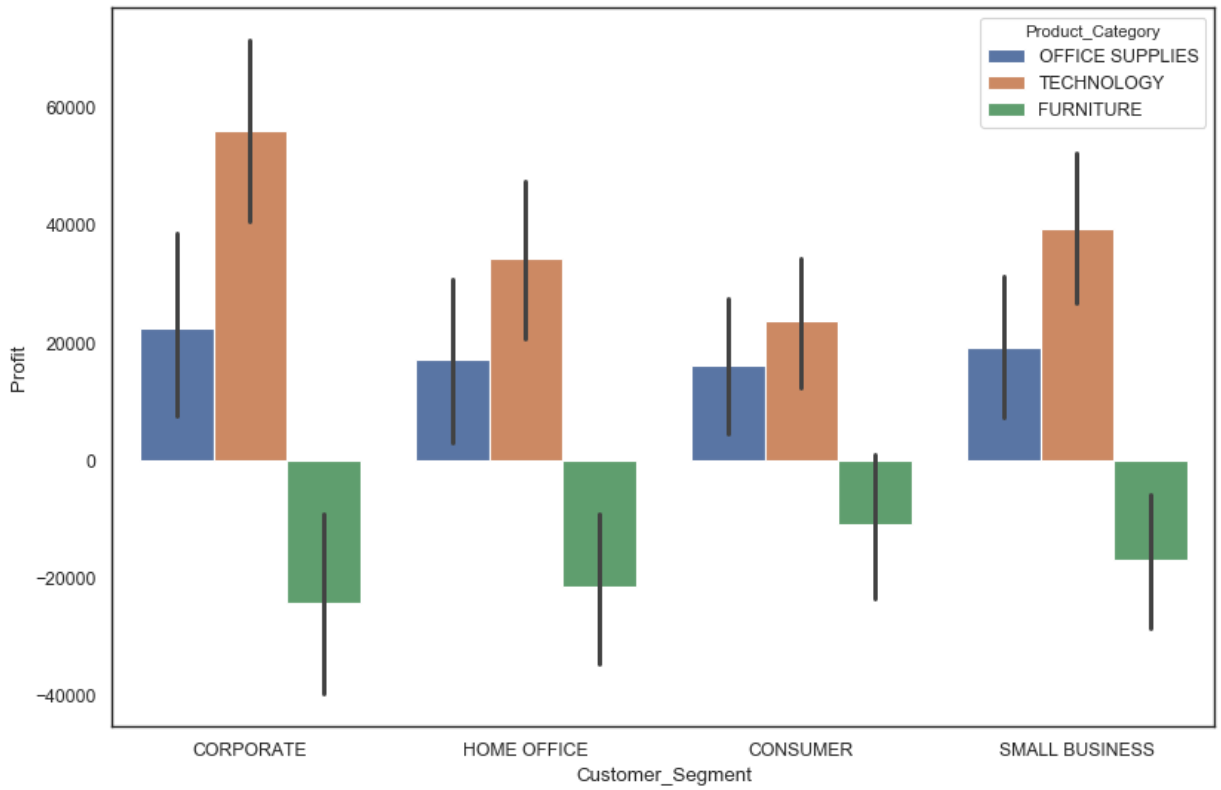
In [44]:
```
# Create 2 subplots for mean and median respectively

# increase figure size
plt.figure(figsize=(12, 6))

# subplot 1: statistic=mean
plt.subplot(1, 2, 1)
sns.barplot(x='Product_Category', y='Sales', data=df)
plt.title("Average Sales")

# subplot 2: statistic=median
plt.subplot(1, 2, 2)
sns.barplot(x='Product_Category', y='Sales', data=df, estimator=np.median)
plt.title("Median Sales")

plt.show()
```



Look at that! The mean and median sales across the product categories tell different stories. This is because of some outliers (extreme values) in the `Furniture` category, distorting the value of the mean.

You can add another categorical variable in the plot.

In [46]:
```
# set figure size for larger figure
plt.figure(num=None, figsize=(12, 8),dpi=80)

# specify hue="categorical_variable"
sns.barplot(x='Customer_Segment', y='Profit', hue="Product_Category", data=df, es
plt.show()
```



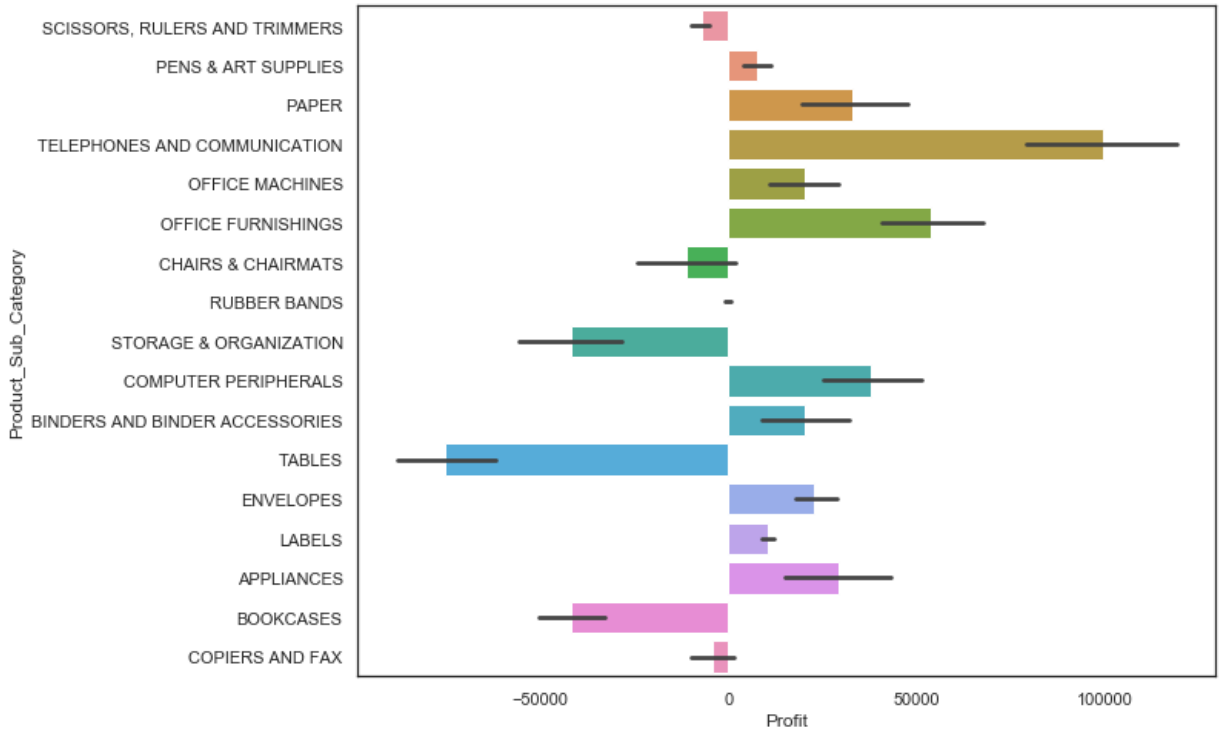
The plot neatly shows the median profit across product categories and customer segments. It says that:

- On an average, only Technology products in Small Business and Corporate (customer) categories are profitable.
- Furniture is incurring losses across all Customer Segments

Compare this to the boxplot we had created above - though the bar plots contains 'lesser information' than the boxplot, it is more revealing.

When you want to visualise having a large number of categories, it is helpful to plot the categories across the y-axis. Let's now *drill down into product sub categories*.

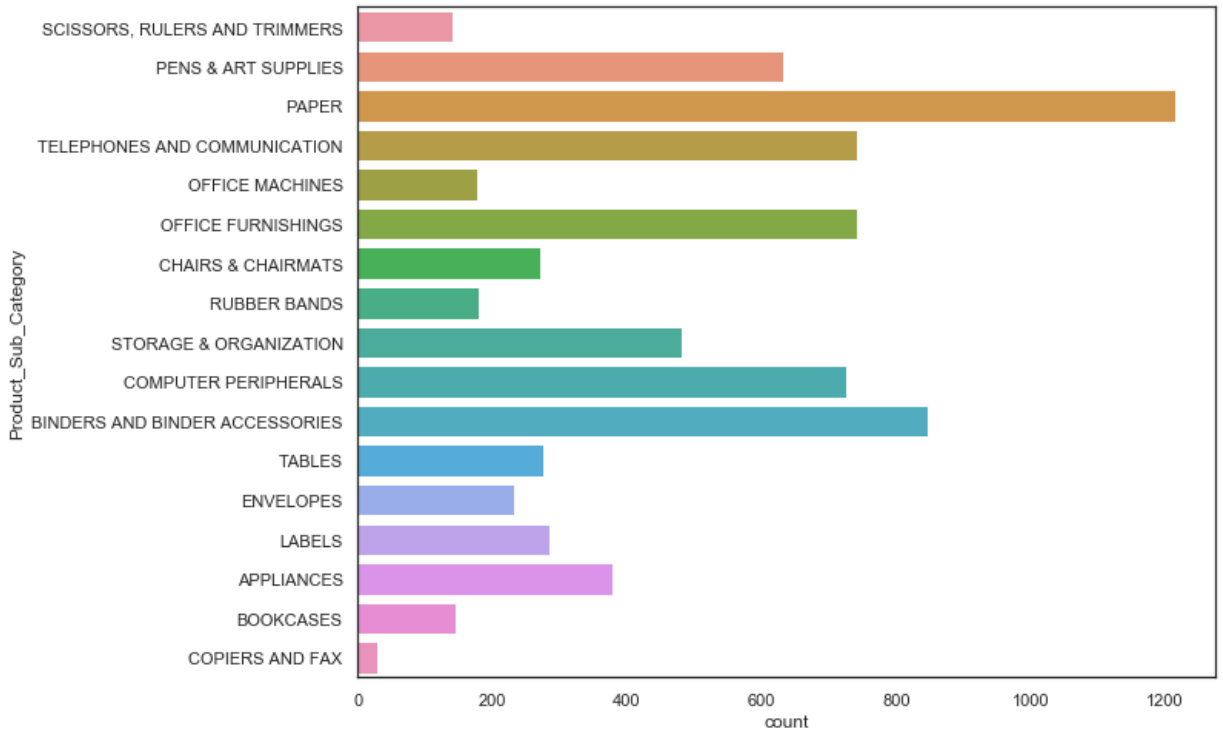
In [48]:
```
# Plotting categorical variable across the y-axis
plt.figure(figsize=(10, 8))
sns.barplot(x='Profit', y="Product_Sub_Category", data=df, estimator=np.sum)
plt.show()
```



The plot clearly shows which sub categories are incurring the heaviest losses - Copiers and Fax, Tables, Chairs and Chairmats are the most loss making categories.

You can also plot the **count of the observations** across categorical variables using `sns.countplot()` .

In [49]:
```
# Plotting count across a categorical variable
plt.figure(figsize=(10, 8))
sns.countplot(y="Product_Sub_Category", data=df)
plt.show()
```



Note that the three most loss making categories - Storage, Tables and bookcases - has a significant number of orders.

Additional Stuff on Plotting Categorical Variables

1. Seaborn official tutorial on categorical variables (https://seaborn.pydata.org/tutorial/categorical.html)

Data Visualization Session 14

Visualising Time Series Data

In the section, we will explore ways to visualise data gathered over time. We will:

- Plot simple time series plots
- Derive variables such as month and year and use them for richer visualisations

In [20]:
```
# loading libraries and reading the data

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# set seaborn theme if you prefer
sns.set(style="whitegrid")

# read data
market_df = pd.read_csv("./global_sales_data/market_fact.csv")
customer_df = pd.read_csv("./global_sales_data/cust_dimen.csv")
product_df = pd.read_csv("./global_sales_data/prod_dimen.csv")
shipping_df = pd.read_csv("./global_sales_data/shipping_dimen.csv")
orders_df = pd.read_csv("./global_sales_data/orders_dimen.csv")
```

Visualising Simple Time Series Data

Let's say you want to visualise numeric variables such as `Sales`, `Profit`, `Shipping_Cost` etc. over time.

In [22]:
```
market_df.head()
```

Out[22]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	
1	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56	
2	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90	
3	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34	
4	Ord_5485	Prod_17	SHP_7664	Cust_1818	4233.15	0.08	35	1219.87	

In [21]:
```
orders_df.head()
```

Out[21]:

	Order_ID	Order_Date	Order_Priority	Ord_id
0	3	13-10-2010	LOW	Ord_1
1	293	01-10-2012	HIGH	Ord_2
2	483	10-07-2011	HIGH	Ord_3
3	515	28-08-2010	NOT SPECIFIED	Ord_4
4	613	17-06-2011	HIGH	Ord_5

Since the `Order_Date` variable is in the orders dataframe, let's merge it.

In [23]:
```
# merging with the Orders data to get the Date column
df = pd.merge(market_df, orders_df, how='inner', on='Ord_id')
df.head()
```

Out[23]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90	
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64	
3	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56	
4	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34	

In [24]:
```
# Now we have the Order_Date in the df
# It is stored as a string (object) currently
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8399 entries, 0 to 8398
Data columns (total 13 columns):
Ord_id                 8399 non-null object
Prod_id                8399 non-null object
Ship_id                8399 non-null object
Cust_id                8399 non-null object
Sales                  8399 non-null float64
Discount               8399 non-null float64
Order_Quantity         8399 non-null int64
Profit                 8399 non-null float64
Shipping_Cost          8399 non-null float64
Product_Base_Margin    8336 non-null float64
Order_ID               8399 non-null int64
Order_Date             8399 non-null object
Order_Priority         8399 non-null object
dtypes: float64(5), int64(2), object(6)
memory usage: 918.6+ KB
```

Since `Order_Date` is a string, we need to convert it into a `datetime` object. You can do that using `pd.to_datetime()` .

In [25]:
```
# Convert Order_Date to datetime type
df['Order_Date'] = pd.to_datetime(df['Order_Date'])

# Order_Date is now datetime type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8399 entries, 0 to 8398
Data columns (total 13 columns):
Ord_id                 8399 non-null object
Prod_id                8399 non-null object
Ship_id                8399 non-null object
Cust_id                8399 non-null object
Sales                  8399 non-null float64
Discount               8399 non-null float64
Order_Quantity         8399 non-null int64
Profit                 8399 non-null float64
Shipping_Cost          8399 non-null float64
Product_Base_Margin    8336 non-null float64
Order_ID               8399 non-null int64
Order_Date             8399 non-null datetime64[ns]
Order_Priority         8399 non-null object
dtypes: datetime64[ns](1), float64(5), int64(2), object(5)
memory usage: 918.6+ KB
```

Now, since on each day, multiple orders were placed, we need to aggregate `Sales` using a metric such as mean, median etc., and then create a time series plot.

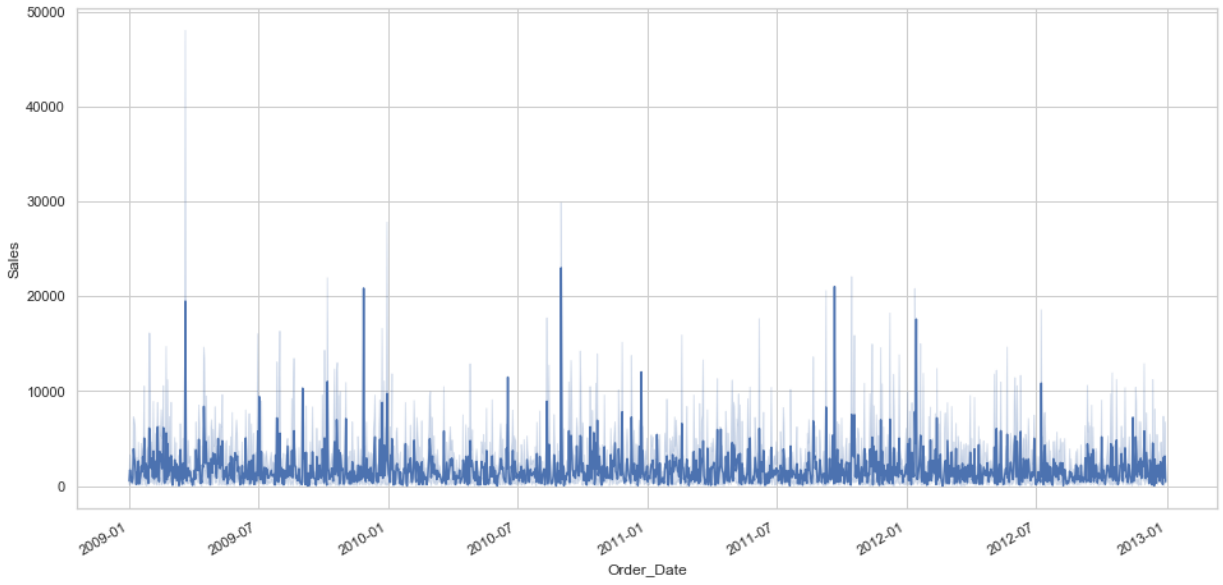
We will group by `Order_Date` and compute the sum of `Sales` on each day.

In [26]:
```
# aggregating total sales on each day
time_df = df.groupby('Order_Date')['Sales'].sum()
print(time_df.head())

print(type(time_df))
```

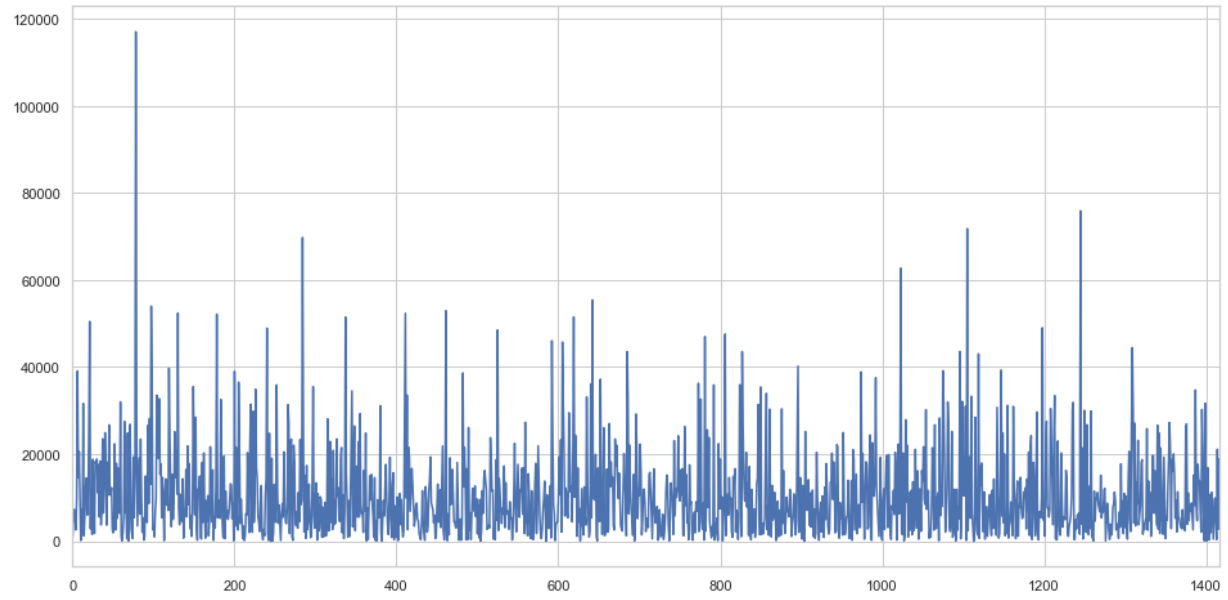
```
Order_Date
2009-01-01    1052.8400
2009-01-02    5031.9000
2009-01-03    7288.1375
2009-01-04    6188.4245
2009-01-05    2583.3300
Name: Sales, dtype: float64
<class 'pandas.core.series.Series'>
```

We can now create a time-series plot using `sns.tsplot()` .

In [28]:
```
# time series plot
import matplotlib.dates as mdates
# figure size
fig, ax = plt.subplots(figsize=(16, 8))
# lineplot
sns.lineplot(x='Order_Date',y='Sales',data=df,ax=ax)
ax.format_xdata = mdates.DateFormatter('%Y-%m-%d')
fig.autofmt_xdate()
plt.show()
```



In [27]:
```
# time series plot

# figure size
plt.figure(figsize=(16, 8))

# tsplot
sns.tsplot(time_df)
plt.show()
```



Using Derived Date Metrics for Visualisation

It is often helpful to use derived variables from date such as month and year and using them to identify hidden patterns.

In [29]:
```
# extracting month and year from date

# extract month
df['month'] = df['Order_Date'].dt.month

# extract year
df['year'] = df['Order_Date'].dt.year

df.head()
```

Out[29]:

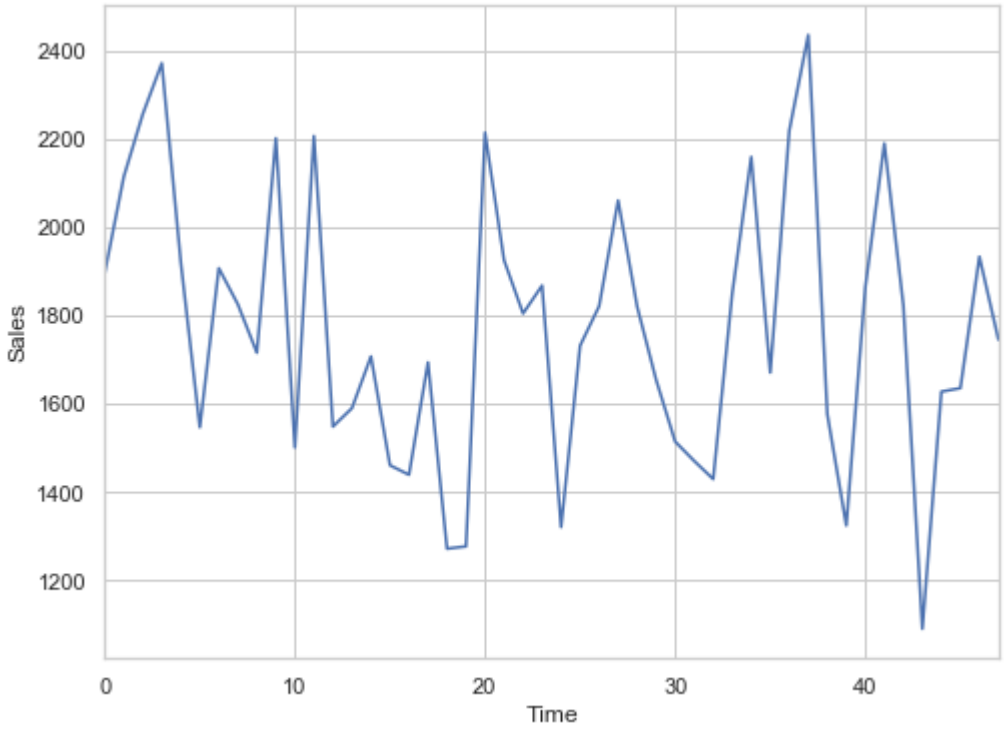
	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipping
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90	
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64	
3	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56	
4	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34	

Now you can plot the average sales across years and months.

In [30]:
```
# grouping by year and month
df_time = df.groupby(["year", "month"]).Sales.mean()
df_time.head()
```

Out[30]:
```
year  month
2009  1        1898.475090
      2        2116.510723
      3        2258.661599
      4        2374.155868
      5        1922.317055
Name: Sales, dtype: float64
```

In [31]:
```
plt.figure(figsize=(8, 6))
# time series plot
sns.tsplot(df_time)
plt.xlabel("Time")
plt.ylabel("Sales")
plt.show()
```



There is another way to visualise numeric variables, such as `Sales`, across the year and month. We can pivot the `month` column to create a wide-format dataframe, and then plot a heatmap.

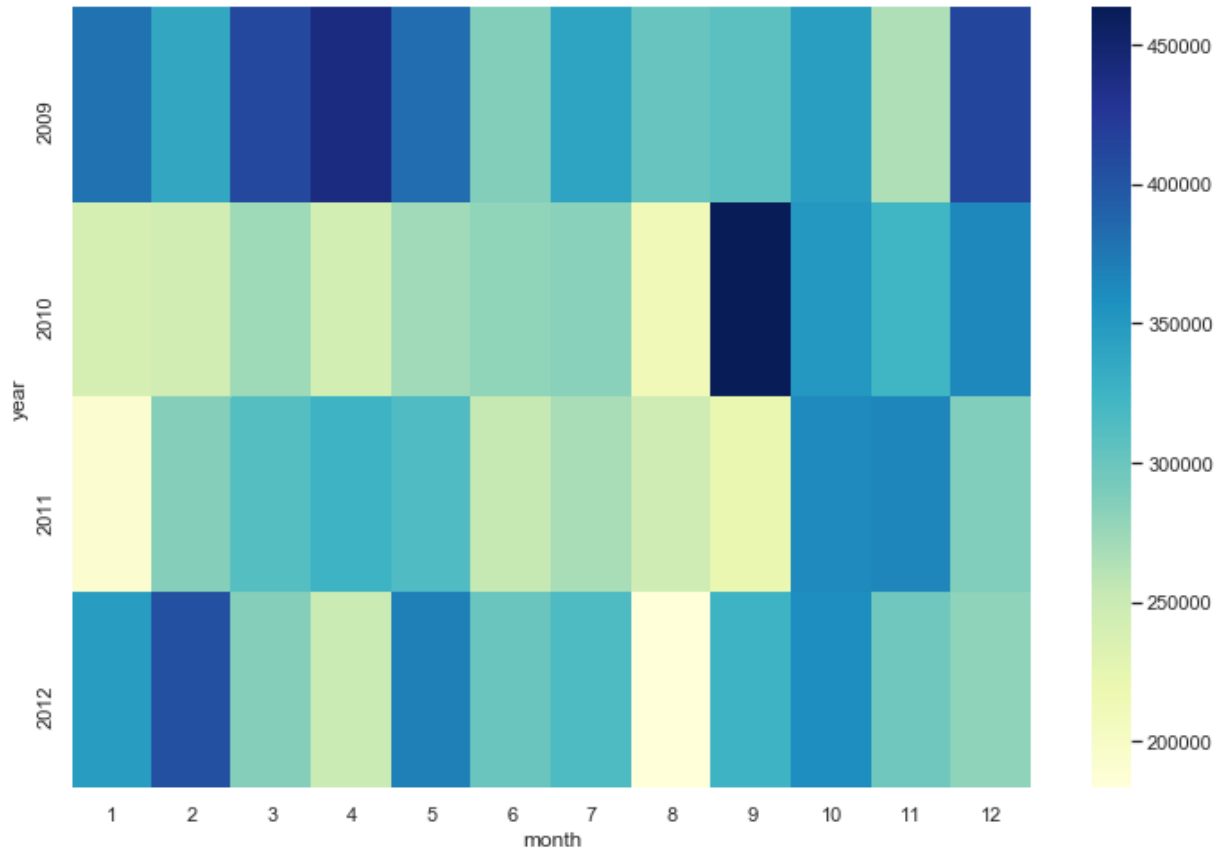
In [32]:
```
# Pivoting the data using 'month'
year_month = pd.pivot_table(df, values='Sales', index='year', columns='month', a
year_month.head()
```

Out[32]:

month	1	2	3	4	5	6	7
year							
2009	379695.0180	336525.2050	411076.411	439218.8355	382541.0940	286397.2530	339817.0355
2010	240197.9760	243504.4415	273429.591	242681.2995	272234.3790	279740.5190	283801.7100
2011	191642.3765	286007.3815	311709.165	325909.2735	315211.8710	253306.7455	268256.9980
2012	346449.7220	404735.7155	285669.409	249147.6945	369417.4365	300335.2720	315713.9180

You can now create a heatmap using `sns.heatmap()` .

In [33]:
```
# figure size
plt.figure(figsize=(12, 8))

# heatmap with a color map of choice
sns.heatmap(year_month, cmap="YlGnBu")
plt.show()
```



Addtional Reading on Time Series Plots and Heatmaps

1. Seaborn heatmaps (documentation) (https://seaborn.pydata.org/generated/seaborn.heatmap.html)