Rhys Tutt

[Email address]

My Analytics handbook

# Data Science





Communication – not everything can be learned from the data, much is from experience.

Presentation – gatekeeper between insights and people.

Domain knowledge – industry knowledge.

Real-life practise

Programming

Creativity – cultivate new ideas and insights.



“By 2018, the US alone could face a shortage of 140-190k people with deep analytical skills” – Report by McKinsey & Company.

Most valuable skills in data science:  
> Communication  
> Business accumin  
> Translating technical skills back to the bottom line and business. E.g. monetizing or user experience. Targeting audience better.  
> Translating insights into business improvements.

# **Business Insights**

## Concepts

1. Strategy - what the business is trying to achieve.

2. Identify which business areas are important to improving strategy. Usually customer, finance and operations.

3. Identify your unanswered business questions - work out exactly what you need to know so you can focus on it.

4. Find the data to answer questions.

5. Identify what data you already have and if you don't have it work out how to collect it.

6. Work out if the costs and effort are justified - once you know costs you can work out if benefits outweigh them. Make a clear case for the investment that outlines long-term value of data to business strategy.

7. Collect the data.

8. Analyze the data.

9. Present and distribute the insights.

10. Incorporate the learning into the business.

https://www.forbes.com/sites/bernardmarr/2016/06/14/data-driven-decision-making-10-simple-steps-for-any-business/2/#5545d725589b

---

1. Choose the right data.

2. Build models that predict and optimize business outcomes.

Performance improvements and competitive advantage arise from analytics models that allow managers to predict and optimize outcomes. More important, the most effective approach to building a model usually starts, not with the data, but with identifying a business opportunity and determining how the model can improve performance. hypothesis-led modeling.

3. Transform your company’s capabilities

http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/three-keys-to-building-a-data-driven-strategy

---

Context: What are you trying to achieve? Who is invested in the project's results? Are there any larger goals or deadlines that can help prioritize the project?

Need: What specific needs could be addressed by intelligently using data? What will this project accomplish that was impossible before?

Vision: What will meeting the need with data look like? Is it possible to mock up the final result? What is the logic behind the solution?

Outcome: How and by whom will the result be used and integrated into the company? How will the success of the project be measured?

1. Sketch: Find your inspiration

2. Prepare: Do your data homework

Filtering: Cut the noise and focus on the most interesting topic.

Sorting: Rank data by importance.

Grouping and segmentation: Summarize data and segment different groups.

Visualizing: Bring data to life using intuitive visuals.

3. Visualize: Bring data to life

Aesthetics

Focus on trends, not data points.

The best insights often come from looking not at singular data points but at trends, especially when they change direction.

Examine different time ranges.

Search for strong relationships

Often, the most powerful and insightful discoveries in data analysis are the relationships between variables, or correlation and dependence to statisticians.

Try different perspectives

Be skeptical

https://www.thinkwithgoogle.com/marketing-resources/data-measurement/data-to-insights-blueprint-for-your-business/

---

1. Insights that support a specific decision or provide additional understanding of a specific situation or issue and will directly benefit the business.

2. Insights that enhance market understanding or strategy definition without directly impacting a specific decision.

Business insights of type #1 should be derived using a deductive method. You begin with a specific issue or question; find a model or framework (aka, theory) that addresses the question; determine the possible answers (aka, hypotheses); use data to find the “right” answers; and develop a recommended solution to the original question.

Here’s a common example. When a company wants to reduce customer attrition and build customer loyalty we apply a standard empirical model of customer loyalty (experiences -> attitudes -> loyalty -> behavior); collect data and validate each relationship (hypothesis) proposed by the model; and then create priorities and recommendations.



The second type of insight is the result of an inductive, exploratory research process. This process usually begins with Inductive research modela fuzzy question, or an issue that doesn’t have well-developed theories or frameworks. The goal is to learn something new or provide better definition of an issue by observing specific instances or data; looking for patterns; developing hypotheses based on those patterns; and looping back through the process until you can develop meaningful, supportable conclusions.



The conclusions or insights derived from this process should impact a company’s strategies. But this type of research is difficult and messy, and the observations-patterns-hypotheses cycle may not produce meaningful insights that impact the business. Consequently, businesses often avoid this research, but they do so at their own peril.

A majority of business insights will, and should, come from deductive processes. However, the true “eureka” moments, the insights that define new and successful strategies, the analyses that uncover undervalued customer segments or new product niches, will most often come from inductive insights.

<http://blog.walkerinfo.com/blog/customer-feedback-analysis/extracting-business-insights-two-types-of-insights>

## Podcast Tips

CRM - Customer relationship management

Understanding if a customer purchases when will you see them again and how do you get them back.

RFM - scorecard based on recency, frequency and monetary each out of 5

Customer segmentation - geodemographic or behavioural segmentation.

E.g. devices, purchases on a certain day (e.g. Friday on android, sunday on desktop), seasonality.

Geodemographic segmentation - static and transactional.

Cost of acquisition of new customers, cost of re-acquisition of existing customers.

Drop off points - why are they leaving.

Are there particular segments more profitable than others?

80/20 rule - 20% of customers creating 80% of profit.

The zero moment of truth - when is the moment you really convince them to purchase.

How do you segment your business - make infographic - age, gender, location.

Time series graph of purchases since date opened

Products preferred by first time customers.

People who've come to the site multiple times but not purchased, give them a specific message.

## Initial Questions

What is the problem I’m trying to solve?

Why is it happening?

What are the key metrics/ indicators driving the action?

What is the next course of action after the insights/model? How would we apply it and test against the norm, e.g. AB testing.

## Data Science Project Ideas

<https://www.datascienceweekly.org/articles/aspiring-data-scientist-here-are-some-at-work-project-ideas>

<https://customerthink.com/five_big_ideas_profit_from_analytics_big_data/>

## My Ideas

> Look at who the customers are and what their behaviour is. E.g. bulk of purchases coming from which demographic.

> Look at what count of players are contributing to what % of deposits. E.g. 10 players contributing to 25% of deposits etc etc.

> Model of big customers and compare to how it looks historically and their current attrition.

> Is there a specific market or area they're coming from. Purchases per player info, is there high value in an area per customer or are there tons of smaller customers. What kind of promos do they respond to.

> Where is the profit coming, where are the losses coming, then analyze those areas to see why, what or who is causing it.

> Look at demographics and behaviour of groups.

> Look at operations we have in place already which could be improved to incentivize deposits.

> Look at promos and marketing we have in place, what's effective etc.

> When are we losing the customers, look at their overall experience etc but e.g. at what time periods do we need to focus on customers.

> Life cycle of customers and understanding them properly so you can figure out how to channel them.

> Look at different customer brackets and what they respond to. E.g. smaller bracket chase bonuses whereas bigger players don’t care as much etc.

> Think about the grouping levels you want, then think about what you’re measuring and the time period, then think about the different variables you want to introduce. E.g. user satisfaction - which could be measured as margin or netwin.

> Outliers can vastly effect the numbers, so often better to break down into buckets e.g. depositing buckets; or use measures like avg deposit, median, standard deviation.

> Keep in mind month-to-date stats.

> Often first step is to visualize data with histograms/boxplots etc and see how it’s distributed and go from there. 80/20 rule.

> Data exploration should be very visual.

> When do people open emails? e.g. 3pm

> Devices better on certain days

> Remember time between deposits (average duration) as input variable.

> Always need to be analysing increases and situations to see % of new players, existing and does this present more of a case for acquisition or cross selling?

> ALWAYS looking at who are the customers!

> Determine good customer or seasoned customer behaviour, market to those customers.

> Always looking for similarities and trends between observations.

> Random custom analytic project ideas – multiple logins no purchase, rejections to inactivity, good players under different details, what are the leaving touchpoints and reasons (e.g. margin, experience)

## Terminology

Calculate profit = NetWin + Progressive Wins - Bonus

ROI is GW/Cost and ROP Purch/Cost

The more you spend on marketing the more deposits you need but then the ROP requirement decreases.

## Visualization Tips

Only data should speak, reduce the title sizes and other features. So focus is on the insights.

Each object should have 0.5cm gap.

Colours are important.

Colour.adobe

Titles are very important – 30-40% of info of chart comes from title.

## Books

Lean Startup

The Truthful Art: Data, Charts, and Maps for Communication - <https://www.goodreads.com/book/show/26401716-the-truthful-art>

The Functional Art: An introduction to information graphics and visualization (Voices That Matter)

Head First Data Analysis: A learner's guide to big numbers, statistics, and good decisions

Data Smart: Using Data Science to Transform Information into Insight  
<https://www.goodreads.com/book/show/17682206-data-smart>

Doing Data Science: Straight Talk from the Frontline

The Big Book of Dashboards: Visualizing Your Data Using Real-World Business Scenarios - <https://www.goodreads.com/book/show/31396339-the-big-book-of-dashboards>

The Visual Display of Quantitative Information by Edward Tufte - <https://www.goodreads.com/book/show/17744.The_Visual_Display_of_Quantitative_Information>

Dear Data by Giorgia Lupi and Stefanie Posavec

Life 3.0: Being Human in the Age of Artificial Intelligence

# **Customer Analytics**

Customer analytics is the collection, management, analysis and strategic leverage of an organizations granular data about the behaviour of its customers.

It focuses more on observed behavioural patterns than demographics or attitudes; is inherently granular focusing on individual-level behaviour rather than aggregate patterns.

Descriptive takes data collected and tries to establish patterns; predictive analytics analyses the past behaviour of customers to make future predictions; prescriptive tries to give recommendations based on the descriptive and predictive analytics to change consumer behaviour.

Application Tips:  
> Track page browsing, products considered, cookies for past experience, can link to offline.  
> Don’t just take last click – take entire path for attribution.  
> GRP: Reach \* Frequency and it’s monetization counterpart. E.g. ESPN wanting to know if launching mobile will cannibalize sales on computer as ads are better on computer.  
> Look for indirect value of promotions, e.g. sending a discount for group activity and others take up offer which you can answer if you track that persons contact.  
> FB and online have greater short term effects; TV has a greater carry-on effect.  
> Track where customers navigate and tailor experience to that.   
> Can also track customers paths in stores, can determine whether short walks are more valuable or not and then determine where to put items and the checkout.  
> Planned and unplanned purchases.  
> Do more social media scraping – American Express doing this to see whether you’re going to churn.  
> Predict based on activity within different periods, e.g. 1st month, 2nd month, 3rd month.

<http://knowledge.wharton.upenn.edu/article/finding-the-right-tool-to-unlock-the-power-of-data/>

## Descriptive Analytics

* Links market to firm through info.
* Information needed for actionable insights.
* Principles for systematically collecting and interpreting data to aid decision makers.

Types:

* Exploratory research (ambiguous problem) - why are our sales declining. Commonly answered with focus groups and internet communities.
* Descriptive research (aware of problem) - what kinds of people are buying our products and who buys our competitors products. Segmentation.
* Causal research (problem clearly defined) - will buyer purchase more with a change of our website. Determined through AB testing and field experiments.

NPS – net promotor score – how likely are you recommend…?

3 Factors Necessary for Causation (important for AB testing):  
- Correlation (evidence of association)  
- Temporal antecedence (one must occur before the dependent outcome)  
- No third factor driving both.

## Predictive Analytics

### Regression

Demand analysis – price vs quantity; as price goes up demand goes down. This allows you to change variables and see what effect that would have, e.g. price sensitivity. Can then predict optimal pricing which maximizes overall profit.

Could run this for various variables – marketing vs new customers; promotions vs sales.

Ideal tool for understanding the drivers of demand and for demand prediction.

Limited that if you don’t have the existing data you can’t predict further forward.

### Customer Centricity

Customer lifetime value.

Take averages and statements of cohorts.

RFM – you’re not really looking at patterns eg. 1 1 1 0 1 1 1; rather if they satisfy the criteria.

Build probabilistic models – e.g. buy till you die model – % of propensities to keep making an action (for example purchasing).

## Prescriptive Analytics

Find optimal price by profit-cost on graph.

Willingness to Pay (WTP) – how much a consumer would pay for an additional item; find it by calculating the area below the demand curve for each item.

Can then use both of these to create optimal bundle.

Be careful drawing conclusions from descriptive data, need to run experiments and test this conclusion is correct due to other outside causal factors.

Consumer theory.

Industrial organization – game theory.

## RFM Segmentation

RFM segmentation is a technique used to get to know the customers better and to be able to divide them into groups which will make marketing targeting more effective and cost-efficient.

RFM stands for Recency, Frequency and Monetary and it does exactly this – gives these three dimensions to our customer base based on their transactions (how recently they purchased, how frequently they purchased and how much money they spent).

The premise is that customers who have bought more recently, purchased more often and have spent more money are more likely to respond to a marketing offering, than customers who have purchased less recently, not so often and for very little money. RFM analysis can be used to segment and label the customers (active, churning, churned, never activated) and thus the proper marketing campaign can be directed to the right group of customers.

**Recency –** represents the “freshness” of customer activity. Naturally, we would like to identify active and inactive customers. The basic definition for this attribute is the number of days since their last order or interaction.  
**Frequency –** how often the customer buys. By default this could be the total number of orders in the last year or during the whole of customer’s lifetime.  
**Monetary** value indicates how much the customer is spending. In many cases this is just the sum of all the order values.

Recency is usually considered the most valuable metric.

Good website - <https://www.optimove.com/learning-center/rfm-segmentation>

<https://www.researchgate.net/publication/281655061_Modelling_customer_churn_using_segmentation_and_data_mining>

<https://www.putler.com/rfm-analysis/>

## Customer Segmentation

Geodemographic – where they are located; demographic (B2C) – race, age, gender, religion, income, education, occupation; demographic (B2B) – company size, industry, role, time working for the company; physcographic – less tangible principles such as lifestyle, values, social class, personality; behavioural – usage, loyalites, awareness, occasions, liking, purchase patterns.

Often when I personally do behavioural segmentation I’ll look at segmenting the players by their everyday behaviour. E.g. looking at days purchased, average sum, count and amount per day.

You can combine this with lifetime behaviour or combine it with multiple different methods of segmentation.

Then this can greatly improve retention and email marketing.

<http://www.thebridgecorp.com/customer-segmentation/>

## Cohort Analysis

**Cohort –** this is a group of people or events who share a common characteristic over time

**Cohort analysis –** this is the study of activity / behavior of a particular cohort or a group of them over time (or other iteration).

‘Take a dataset which has uniquely identified entity (customer in this case), define unique cohorts by which the items can be grouped (in this case we will use the year and month of first purchase), and follow the behavior of the cohorts over time (in this case the sum of the OrderValue for each cohort per timeslice)’.

A cohort is usually a group who you have similar acquisition characteristics (e.g. time).

One of the most important measures is customer retention.

## Lean Analytics Tips

10: Ratios are good for comparing factors that are opposed or have some kind of tension. A good metric changes behavior.

12: Types of Metrics - Qual / Quant, Exploratory / Reporting, Leading / Lagging, Correlated / Causal ( If you find a relationship between something you want and something you control, you can change the future )

14: We care about active users, because it probably lead-indicates our churn rate. This requires a lot of different queries to diff DBs. (18) Analyze patterns of engagement and desirable behavior, find commonalities.

16: Knowledge Matrix - Known Knowns are facts that may be wrong and should be checked against data. Known Unknowns are questions we can answer with automated reporting. Unknown Knowns are intuitions we should quantify. Unknown Unknowns are located through exploratory reporting and where we can locate unfair advantages and new insights.

18: Pivot hard or go home, and be prepared to burn bridges.

19: Churn is a lagging indicator. Cohort analysis (comparing customer groups over time) is the road to leading indicators.

20: Are the metrics we track helping us make better decisions faster?

22: A company assumed "active" was 4x a week, when it turned out to be only 1x a week (to great success). Things they tried: Clarified signup flow, added more explanatory copy. Daily email notifications, transactional emails tied to actions on-site.

24: A segment is a group that shares a common characteristic. Segment visitors then compare segments to each other to understand differences in metrics. Look for disproportionate relationships.

26: Cohort analysis allows patters to emerge across customer lifecycles.

27: We can test anything, but focus on the critical steps and assumptions.

34: Venn Diagram: Expertise, Desire, Monetizable. E+D = Learn to M. E+M = Improve D. D+M = Learn to say no. All 3 = Victory!

52: Use Google Analytics multi-channel conversion visualizer to see which referral sources are combining to influence visitors

57: Find days where unsubscribe rate is high, then find out why. Need to tweak the unsubscribe process to get better resolution on this. Find the action, not the result.

92: Backupify focuses on monthly recurring revenue. They watch churn, but are not going to focus on it until they hit the 10MM revenue level.

97: Properly calculating churn: Select a time period. Average the number of customers at the beginning and the end of the period. Divide the number of cancellations by this number. To increase data integrity, measure churn daily ( using the method in 57: )

125: User Generated Content: Use the forums as a source of UGC and find ways to repackage it and syndicate it.

154: QS is in Stage 5, we have revenue and we are beginning to branch into new verticals. We need an ecosystem to help us cross the gap from niche site to industry staple.

159: Find out what's actually important to people. Get inside their head. Delve for this information aggressively.

211: Refresh the 3-year plan every 18 months. Align the entire company around the vision.

213: The best companies warehouse every possible data point about their site's interactions and use only the data they need. Rally ( a software co ) records everything from kernel-level performance to HTTP-based user gesture interactions between the browser and software. They can then correlate changes in site performance to user behavior and vice versa.

256: metrics for stage 5 (scale). Attention is a precious commodity. Don't waste the visitor's attention on stuff that doesn't matter. Internally, compare the metrics that matter across channels, regions, and other segments to find efficiencies and inefficiencies.

258: Get a better understanding of what new visitors really do.

260: Limit the company's vision to a 3-pronged strategy.

261: For each C-Level strategic assumption, what are the 3 line-level tactics that can be used to survey, test, prototype, then fill/kill quickly?

262: Enable (both emotionally and technically) anyone on staff to run a split test. Give the line level a wide range of flexibility.

263: Scale stage summary: Focus on the health of the ecosystem and ability to enter new markets. Pay attention to compensation, API traffic, channel relationships, and competitors. These are no longer frivolities or distractions.

274: Hypothesis to test: most people unsubscribe because they don't need our service anymore, not because we're crappy. Limited data bears this out but we need to conduct FAR better exit surveys.

281: Product pricing has nothing to do with cost, and everything to do with what the customer pays and how they derive value from the product.

283: Try an intentionally absurdly priced package to anchor high prices, as well as to see if anyone actually bites.

286: We have no real way to measure virality. We can try to use the affiliate program but I suspect many will refer by word of mouth if given an easy way to do so. How many do they refer and how quickly?

287: Sharing by email accounts for 80% of social sharing, usually between small groups of people. Remember Emerson Spartz' theory about "bridge nodes". Which groups of people are likely to be conduits between other peer groups? (326)

288: 3pm is when people are most likely to open an email. If software permits, time newsletters on a per user level, based on signup time.

290: Site load time matters a great deal. Spend a lot of time to get this down.

302: Negative Churn is the long-tail of brand awareness. We might convert some "Long time listeners, first time callers" after a year, or longer. Focus on customers that have been on our mailing list for a long time but are not subscribers.

304: Top SaaS companies increase revenue per customer by 20% a year. Can we upsell the Buylist product enough to hit this target? Can our price increase for new subscribers help us hit this level?

324: What kind of content do different traffic sources expect? Twitter Time on Site is disproportionately high. Why is this?

325: Find outlier content and promote it more heavily. Unlocked Insider articles are great for this, as are "nexus pages" like our BOTG page.

326: Most sharing is intimate. Each share generates an average of 9 visitors, per the book's data. Can we find this number for our site? Book data; 5::1 Twitter, 36::1 on reddit. Sharing happens from a groundswell of small interactions between colleagues and friends rather than a one-to-many broadcast. See above (287) about bridge nodes.

334: Engagement rations: 90% lurk, 9% contribute sometimes, 1% engage heavily. Make participation easy, and a side-effect of site usage.

## Practical Example of Data Modelling

DATA MODELLING GENUINE NEW PLAYERS

1. New accounts who deposit in first month.

2. Accounts where they haven't had any account or deposits on the MID.

3. Look at links to other MIDs and delete where there are previous deposits.

4. Get count users, look at rolling month sum of deposits and attrition %.

5. Financials - purchases, bonus, cashins, profit/loss.

6. Account info - bannertag, which casinos are they coming on?

7. Demographics - countries, DOB, method of purchase,

8. Behaviour - promoID %, margin during different periods, gameplay info (max bet, slots)

9. Overall Stats - Make up of accounts, AVG deposit per user, SUM deposit per user, % of SUM coming from how many players.

## CLV

If the vertical line for every customer indicate a purchase, the main idea is that since the first customer purchased more frequently in the past but stop purchasing for a long time, the probability of being alive is lower in compare to customer 2 who purchase less frequent with large gap between purchases.



From Reza -

<https://university.custora.com/for-data-scientists/clv/advanced/pareto-nbd>

CLVTools package – need to check this out

## Customer Churn Predictions

Customer Churn – Logistic Regression with R - <http://www.treselle.com/blog/customer-churn-logistic-regression-with-r/>

R and Tableau - <https://dzone.com/articles/predict-customer-churn-using-r-and-tableau>

A comparison of machine learning techniques for customer churn prediction - <http://www.sciencedirect.com/science/article/pii/S1569190X15000386>

## Attrition

Looks based on different cohorts as totals based on day/month 0 – sum purchases & count users over 7/14 days etc and get total % sums.

Also proportion the return by cohort on each period, e.g. day 0 – 46, day 2 – 12% etc.

This way you can start to to see which cohorts are more valuable and sticking around longer or which need extra attention.

## Readings

NPS - <https://www.jacada.com/blog/is-net-promotor-score-still-the-best-metric-for-measuring-customer-satisfaction>

<http://knowledge.wharton.upenn.edu/article/what-marketers-need-to-know-about-binge-buying/>

<http://knowledge.wharton.upenn.edu/article/predicting-and-monetizing-the-lifespan-of-a-tweet/>

<https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2701093>

<https://s3.amazonaws.com/kw-wdp/epubs/20150912-WDP-Fader-CustomerCentricity_Chapter4_CourseraCustomerAnalytics001.pdf>

<https://www.forbes.com/forbes/welcome/?toURL=https://www.forbes.com/sites/blakemorgan/2016/01/26/customer-centricity-with-whartons-dr-peter-fader/&refURL=&referrer=#12de5f5b4580>

# **Marketing Analytics**

<https://rjmetrics.com/resources/the-ultimate-30-minute-guide-to-marketing-analytics/>

## ROI

Return on Investment (ROI) is what you get back for whatever you lay out. But it’s a money-metric, and that means it’s about as depersonalized as you can get. So in a people-intensive field like planned giving, ROI may actually be skewing your marketing vision. Consider instead Return on Prospect (ROP). It’s different because it takes into account the fact that prospects create value for you in two ways: by contributing today; and by resolving to contribute in the future.

ROI = (Incremental Profit – Campaign Cost) / Campaign Cost

This revenue versus cost ROI ratio, however, becomes more complicated for marketing initiatives as a campaign’s timeframe gets longer.

Marketing activities aimed at brand awareness often have an aggregate effect that — if measured on too-short a time-frame — will contradict their overall impact on ROI. Measuring and optimizing leading metrics on each marketing channel provides early indicators of success (or failure) of marketing campaigns to better track and predict their cumulative impact on customers’ purchasing decisions in the long run.

The compounding ROI of customer lifetime value - a predictive metric that forecasts the total future value a customer is projected to drive for a company over the course of their entire life. When you reconsider the revenue versus cost ROI formula, it’s clear that an understanding of CLV is integral to determining the upfront costs to attract and convert new leads, and the investment required to keep and grow existing customers.

The typical number of touches it takes to close a deal varies across the board, but in almost every case it’s greater than one. While the revenue from a single sale is ultimately known, how do you assign value to the host of touch points that influenced the deal along the way? Whether you use single-touch or multi-touch attribution, the branching of attribution across multiple touch points complicates the net revenue to cost ROI calculation.

## Campaign Analysis

UTM codes gives detailed info on where a customer comes from – e.g. our btag. This shows the source (e.g. FB, google), medium (social cpc, email), content (particular advert) and campaign (the campaign name or specific promotion).

From this you can understand your ROI and can work out your cost per lead (CPL) – divide spend by total number of leads.

Look at email analytics – e.g. open rate, click through rate, click through to purchase rate.

Some campaigns might need to be measured in a different way as they may be more aimed at brand awareness, could require text mining or natural language processing (NLP).

Use CRM software to understand how many new opportunities, leads and completed sales resulted from your campaign.

Social intelligence – volume of mentions, reach, engagement, news coverage, share of voice, purchase intent, sentiment and emotional response, brand associations.

<https://www.brandwatch.com/blog/complete-guide-campaign-analysis/>

### Common Terms & Measures

Pixel – A short snippet of java script code that collects information about the visitor abd their behaviour on the website.  
CPL – Cost per lead (e.g. pixel fire) when player first click on advertisement and gets redirected. Shown a lead form and incur cost when player fills in lead form. (Cost/PixelFire)  
CPR – Cost per registration. (Cost/Real Reg)  
CPA – Cost per action, usually measured as p9. (Cost/P9)  
CTR – Click-through rate. (Clicks/Impressions)  
CPM – Cost per impression (Costs/Impressions)  
CPC – Cost per click (Cost/RawClicks)  
Stdl – Started downloads  
ROP – Return on purchase; similar idea to ROI return on investment. (Purchases/cost)  
Cost – Pixel Fire \* Cost Per Lead  
P1/PP – Purchased > 1  
PP2+ - deposited more than once  
Pixel to p9 (24) – p9/PixelFire  
Pixel to reg – RealReg/PixelFire  
Day0 ROP – USDPurchasesDay0/Cost  
Day2 ROP – USDPurchasesDay2/Cost  
SUBID – a string of alphanumeric characters generated at the end of a redirect URL, which records a user-defined variable  
User Agent String – comes from web browser and gives information about the browser, operating system, device, etc. Many companies have apps for this like 51 degrees.

## Attribution Modelling

An attribution model is the rule, or set of rules, that determines how credit for sales and conversions is assigned to touchpoints in conversion paths. For example, the Last Interaction model in Analytics assigns 100% credit to the final touchpoints (i.e., clicks) that immediately precede sales or conversions. In contrast, the First Interaction model assigns 100% credit to touchpoints that initiate conversion paths.

<https://support.google.com/analytics/answer/1662518?hl=en>

## Randomized Control Trials

Used to analyse causal effects – running experiments to see if campaigns have no effect or little effect so you can stop the campaign.

Measuring what would happen if you turned everything off.

To analyse whether marketing campaign caused someone to buy – have two very similar populations and with one you send the marketing material, the other you don’t. Users must be randomly selected but very similar.

Basically AB testing.

## Game Theory

Game theory is a tool used to analyze strategic behavior by taking into account how participants expect others to behave. Game theory is used to find the optimal outcome from a set of choices by analyzing the costs and benefits to each independent party as they compete with each other.

Although it’s generally agreed that game theory could be an excellent tool, it’s actual use in present day marketing practice is very rare. This is because many see game theory as being too empirical to work with marketing, which by its nature is highly subjective.

The issue is that game theory analyses the behaviour of rational players, whose decisions can be predicted, and any deviation from the norm easily explained. On the other hand, marketing exists to control consumer behaviour, which is usually irrational and affected by many different, often unidentifiable factors - such as feelings and desires - which can’t be predicted or quantified.

Game theory also ignores the marketing department’s role in creating and protecting a brand image. The uncertainty of public opinion means that a decision that’s technically the most logical or rational might not actually be the best approach in terms of publicity.

It’s therefore impossible to apply game theory fully to marketing. However, it can still be used as part of a wider marketing strategy, as a "mixed" approach.

Customer journey – the trick is to rationalise the consumers’ decisions as much as possible, leading them down a controlled path where their actions can be predicted and responded to in the most effective way possible.

<https://www.campaignlive.co.uk/article/playing-game-theory-marketing-tool/1309753>

<http://blogs.cornell.edu/info2040/2017/09/18/how-e-commerce-uses-game-theory-to-capture-consumer-dollars/>

# **Management**

## Initial Approach

1. Understanding
   * Regular processes, reports and generally how things are currently done.
   * Engage with staff to get their opinions on what’s being done right and wrong, new ideas and also what they want to achieve.
2. Optimization
   * Quick wins.
   * Create new project flow and procedures.
   * Improve the current processes before anything, also frees up time for people to focus on projects you want.
3. Improvement & development (business and personal).

* Improving current processes.
* New metrics to monitor.
* Involves speaking to team and outside of team.
* Innovations, drive culture of thinking outside the box.
* Developing personal skills.

## WHY & WHAT

WHY – Not just making observations but we’re always finding out and explaining why they occurred, so linking observations with what’s happening in the business.

WHAT – What effect will events have, what can we do next and do we need to monitor anything rather than looking after the fact.

## Coaching

Personal development plan.

<https://www.thebalance.com/coaching-for-improved-performance-1918713>

## Ideas

Managing by exception

# **Data Storytelling**

## Basics

3 minute story – try to tell your entire story concisely within 3 minutes.

The big idea – making your point in a single sentence. This must articulate your point of view, convey what’s at stake and be a complete sentence.

Storyboarding – issue, demonstrate issue, ideas for overcoming issue, describe program, show pre-data and results, give recommendation.

Always remember context.

Align text with top left, sometimes just make some words bold for effect.

Don’t be afraid of white space.

Get rid of redundant info or clutter, even removing gridlines and axis can be good.

Label bars with text values inside the bar.

Use simple text for effect especially when just a simple comparison.

Can use subtitles and captions.

## Gestalt Principles of Visual Perception

Leverage alignment of elements and maintain white space.

Use contrast strategically.

Clutter is your enemy.

Example – remove chart border, remove gridlines or make them very thin and light grey, often remove data markers on line chart, clean up axis labels with commas and remove trailing decimals where possible and abbreviate months, at times remove legend and label data directly in consistent colour for proximity.

The more work the user has to do the more likely you lose their attention.

Remove axis labels all together if you want people to focus on the comparison rather than the numbers.

## Preattentive Attributes

Use preattentive attributes to focus attention.

Colour only certain bars to make a point and leave others more opaque colours.

Can even put same graph in presentations with different iterations of the same visual with different colours and annotations.

Do this more for explanatory than exploratory.

One tactic is to push everything to background (make everything grey) then start working on it depending on what you want to show.

Data labels and points only on specific points.

Use size for levels of importance.

Use colour sparingly, start with grey and then use colours intentionally to highlight. Often use blue or a coral/dark red.

Look away and then look at chart to see where eyes are drawn.

If you’re using grey with one colour, retain this same colour scheme so people get used to what colour to focus on. Can shift that scheme with different topic.

## Clutter & Context

Identify distractions by thinking about clutter and context.

Not all data are equally important; when detail isn’t needed summarise, would eliminating this change anything? Push necessary but non-message impacting items to the background.

Lines are the best for trends.

Keep things simple, label explicitly and if there’s a conclusion you wish to draw then say it in text.

## Storytelling

3 part story.

This only comes after you have the big idea and the 3 minute story.

Again storyboarding with post-it notes is important – explain issue, demonstrate issue, ideas for overcoming issue, describe program, show pre-data and results, give recommendation with call to action.

## Powerpoint Presentation

Quintus, Sphere & Strategy, Phlox looks good (roadmap)

# **Statistics**

## Probability and Data in R

### Basics

LO 1. Identify variables as numerical and categorical.

If the variable is numerical, further classify as continuous or discrete based on whether or not the variable can take on an infinite number of values or only non-negative whole numbers, respectively.

If the variable is categorical, determine if it is ordinal based on whether or not the levels have a natural ordering.

LO 2. Define associated variables as variables that show some relationship with one another. Further categorize this relationship as positive or negative association, when possible.

LO 3. Define variables that are not associated as independent.

Test yourself: Give one example of each type of variable you have learned.

LO 4. Identify the explanatory variable in a pair of variables as the variable suspected of affecting the other, however note that labeling variables as explanatory and response does not guarantee that the relationship between the two is actually causal, even if there is an association identified between the two variables.

LO 5. Classify a study as observational or experimental, and determine and explain whether the study’s results can be generalized to the population and whether the results suggest correlation or causation between the quantities studied.

If random sampling has been employed in data collection, the results should be generalizable to the target population.

If random assignment has been employed in study design, the results suggest causality.

LO 6. Question confounding variables and sources of bias in a given study.

LO 7. Distinguish between simple random, stratified, and cluster sampling, and recognize the benefits and drawbacks of choosing one sampling scheme over another.

Simple random sampling: Each subject in the population is equally likely to be selected.

Stratified sampling: First divide the population into homogenous strata (subjects within each stratum are similar, across strata are different), then randomly sample from within each strata.

Cluster sampling: First divide the population into clusters (subjects within each cluster are non-homogenous, but clusters are similar to each other), then randomly sample a few clusters, and then randomly sample from within each cluster.

LO 8. Identify the four principles of experimental design and recognize their purposes: control any possible confounders, randomize into treatment and control groups, replicate by using a sufficiently large sample or repeating the experiment, and block any variables that might influence the response.

LO 9. Identify if single or double blinding has been used in a study.

Test yourself:

Describe when a study’s results can be generalized to the population at large and when causation can be inferred.

Explain why random sampling allows for generalizability of results.

Explain why random assignment allows for making causal conclusions.

Describe a situation where cluster sampling is more efficient than simple random or stratified sampling.

Explain how blinding can help eliminate the placebo effect and other biases.

### Data and Studies Basics

Variables:  
- Numerical (quantitative) – continuous (infite number) or discrete (specific set of numeric values, e.g. a count which can be only a whole number).  
- Categorical (qualitative) – regular categorical or ordinal (have inherent ordering).

When two variables show connection they are called associated (dependent).

Studies:   
- Observational – collect data in a way that doesn’t interfere with how the data arise. Retrospective is past data and prosepective data are collected throughout the study.  
- Experiment – randomly assign subjects to treatments to establish causal connections. Because you’re doing random assignment the other variables you can’t control should be equally represented.

Confounding variables – extraneous variables that affect both the explanatory and response variable, and make it seem like there’s a relationship between them.

CORRELATION DOES NOT IMPLY CAUSATION

A few sources of sampling bias:  
- Convenience sample – individuals who are easily accessible are more likely to be included.  
- Non-response – if only a non-random fraction of people respond; e.g. people who work business hours.  
- Voluntary response – only people who volunteer to respond because they have strong opinions.

Sampling methods:  
- Simple random sample  
- Stratified sample – divide population into strata then sample from within each stratum (e.g. divide population into male / female, then randomly sample from within that).  
- Cluster sample – divide into clusters, randomly sample a few clusters then sample all these observations.  
- Multistage sample – divide into strata and then cluster.

Principles of experimental design:  
(1) Control – compare treatment of interest to a control group.  
(2) Randomize – randomly assign subjects to treatments.  
(3) Replicate – collect a sufficiently large sample or replicate the entire study.  
(4) Block – block for variables known or suspected to affect the outcome.

Explanatory variables (factors) are conditions we can impose on experimental units (e.g. light and sound), blocking variables are characteristics that experimental units come with that we would like to control for (e.g. gender to make sure both genders are represented equally).

Stratified sampling allows for controlling for possible confounders in the sampling stage, while blocking allows for controlling for such variables during random assignment.

Experimental terminology:  
- Placebo – fake treatement used as control group  
- Blinding – experimental units don’t know which group they’re in.  
- Placebo effect – showing change despite being on placebo.  
- Double-blind – both experimental units and researchers don’t know the group assignment.

First you take a sample then you assign them to the different groups.  
Random sampling – generalizability  
Random assignment – assigning subjects with various characteristics that may affect the outcome (confounding variables) equally, as you can then make causal conclusions.



Suggested reading: Chapter 1, Sections 1.1 - 1.5

Practice exercises: End of chapter exercises in Chapter 1: 1.1, 1.3, 1.11, 1.13, 1.17, 1.19, 1.25, 1.27, 1.31

<https://www.openintro.org/stat/textbook.php?stat_book=os>

### Numerical Data

Scatterplots: x = explanatory, y = response. As these are only observational this only shows correlation NOT causation, as we would need to conduct a randomized controlled experiment.

Examine relationship:  
- Direction – Increasing (positive) or decreasing (negative).  
- Shape – linear or curved.  
- Strength – strong (not much scatter) or weak.  
- Outliers

Histograms – provide a view of the data density and show the shape of distribution (left skewed = long left tail so bars on right, symettrical or right skewed = positive).



Unimodal – normal; bimodal – two distinct groups; uniform – no trend.

Center: mean (the arithmetic average), median (the midpoint), mode (the most frequent observation). If these are taken of the sample then they are sample statistics which are point estimates for the population parameters. Mean is usually closer to the tail.  
Spread: standard deviation (variability around the mean /roughly the average deviation around the mean / square root of the variance), variance (the average squared deviation from the mean), range (max-min), interquartile range (middle 50% of the distribution). IQR is often a better measure of spread as it doesn’t rely on the endpoints which can be affected by outliers.  
Variability vs diversty – diversity would have more different data points and variability would have a bigger gap between points in the data.

Robust statistics are measures on which extreme observations have little effect. Median is more robust than the mean, IQR is more roubst than the SD or range. Robust statistics are better for skewed data with extreme observations; non-robust are useful for symettrical data.

COME BACK TO THIS:

Note that an observed difference in sample statistics suggesting dependence between variables may be due to random chance, and that we need to use hypothesis testing to determine if this difference is too large to be attributed to random chance. Set up null and alternative hypotheses for testing for independence between variables, and evaluate the data support for these hypotheses using a simulation technique.

Since we’re randomly splitting the cards into two groups, we would expect similar averages in the two groups, yielding a difference of 0 in the averages.

### Transforming Data

A re-scaling of the data using a function.

Goals of transformations:  
- To see the data structure differently.  
- To reduce the skew and assist in modelling.  
- To straighten a nonlinear relationship in a scatterplot.

Keep in mind it makes the results harder to interpret.

(Natural) log transformation is often applied when much of the data clusters near zero and all observations are positive. Can also do this to make the relationship between variables more linear and hence easier to model with simple methods.

Square root or inverse transformations can also work.

### Categorical Data

Present the data in a frequency table or bar plot; often more useful to see the relative frequencies than raw counts.

Contingency table – relative frequencies of categorical groups.

Segmented bar plots are useful, but again frequency segmented bar plots are more useful.

Mosaic plots can be useful for seeing the marginal distribution by width of bars and can then see the frequencies by each segment in the bar.

### Introduction to Inference

Hypothesis testing framework:  
- Start with null hypothesis that represents status quo.  
- Set an alternative hypothesis that represents the research question, ie. What we’re testing for.  
- Conduct hypothesis test under assumption null hypothesis is true, via simulation or theoretical methods. If test results suggest the data do not provide convincing evidence for the alternative hypothesis, reject it and stick with null hypothesis. If they do then reject and go in favour of alternative.

In a hypothesis test the burden of proof is on the alternative hypothesis.

E.g. male vs female promotions – split them into two groups of males and females.  
We would expect difference in results to be 0.  
Then see how many males get promotions vs how many females – note difference in %.  
Then re-do these steps, if the simulations look like the data then the proportions of promoted are due to chance. If the results of simulations do not look like the data then it wasn’t due to chance and the variables are dependent.  
Can then build a dotplot or similar of the simulated differences.

P-value: the probability of observing data at least as extreme as the original study under the assumption the null hypothesis is true.

### Probability

Frequentist interpretation: The probability of the outcome is the proportion of times the outcome would occur if we observed the random process an infinite number of times.

Bayesian interpretation: subjective degree of belief.

Law of large numbers: as more observations are collected the proportion of occurrences with a particular outcome converges to the probability of the outcome. Common misconception is the law of averages, even if 10 heads in a row the likelihood of a tail on the next flip is still 50%.

Disjoint events - mutually exclusive; non-disjoint events – can happen at the same time.   
Disjoint probability you simply add together and non-disjoint you must subtract any events that overlap.

A sample space is a collection of all possible outcomes of a trial.

Probability distribution lists all possible outcomes in the sample space and the probabilities with which they occur.  
> Events must be disjoint  
> Each probability must be between 0 and 1  
> Probabilities must total 1

Complementary events are two mutually exclusive events whose probability adds up to 1.

The sum of two disjoint events will not necessarily adding to 1 whereas complementary will always add up to 1.

Independence: two processes are independent if knowing the outcome of one provides no useful information about the outcome of the other.  
To calculate two independent events both happening you multiply their probability together.



NEED TO LOOK MORE AT BAYES THEOREM & CONDITIONAL PROBABILITY.

Often drawing our probability trees can help.

Posterior probability is P(hypothesis|data) – it tells us the probability of a hypothesis we set forth given the data we just observed.

Bayesian approach allows us to take advantage of prior information.

### Simulating Probability Outcomes

coin\_outcomes <- c("heads", "tails")  
sim\_fair\_coin <- sample(coin\_outcomes, size = 100, replace = TRUE)  
table(sim\_fair\_coin)

sim\_unfair\_coin <- sample(coin\_outcomes, size = 100, replace = TRUE, prob = c(0.2, 0.8))

### Normal Distribution

Unimodal and symmetric (bell curve); follows strict guidelines about how variably the data are distributed around the mean. 68% within 1 stdev, 95% within 2 stdev & 99.7% within 3 stdev.



Standardized (Z) score of an observation is the number of standard deviations it falls above or below the man.  


Z score of mean = 0

Unsual observations: Z > 2

Percentile is the percentage of observations that fall below a given data point. When the distribution is normal, Z scores can be used to calculate percentiles.

In R we can use Pnorm()  
E.g. a student scores 1800 on test. Pnorm(1800, mean = 1500, sd = 300); we can see they scored better than 84% of other students.

In R we can use qnorm() for cutoff values.  
E.g. finding someone who scored in the top 10%. Qnorm(0.9, mean = 1500, sd = 300)

Evaluating the normal distribution – looking for straight lines on a scatter plot. The closer the points are to a perfect straight line, the more condiment we can be that the data follow the normal model.



### Binomial Distribution

Bernouilli random variable – when an individual trial has only two possible outcomes.

The binomial distribution describes the probability of having exactly k successes in n independent Bernouilli trial with probability of success p.



Binomial conditions: trials must be independent, the number of trials (n) must be fixed, each trial outcome must be classified as a success or failure, the probability of success (p) must be the same for the trial.

In R:

We can use the choose function - choose(9,2)

If we had a case we’re testing of 10 cases, 8 were a success and the success probability is 13%:  
dbinom(8, size = 10, p = 0.13)

### Normal Approximation to Binomial

Z = observation – SD / SD  
P(Z>1.29) = 1 – (where the z score lands on table)  
OR  
In R:  
sum(dbinom(70:245), size = 245, p = 0.25)  
# 70:245 is because we want to know the chance of this happening in these users, 245 is sample size an p is probability of the event.

How do you know if it’s a normal unimodal binomial distribution?   
Success-failure rule: A binomial distribution with at least 10 expected successes and 10 expected failures closely follows a normal distribution.

## Ratios/Percentages

So if we have an experiment group 60% of population and control group 40% of population, which is a 3:2 split. To measure any gains, an increase of purchases would need to be more than this ratio. E.g. if control group deposited $10,000, then 10,000/2 = $5,000. So for any actual gain you would need the experimental group to have purchased more than $15,000 (5,000x3).

Normalize data in ratios you could also divide the results of the experiment by 3 and multiply by 2.

<https://sciencing.com/calculate-improvement-percentage-8588140.html>

<https://en.wikipedia.org/wiki/Relative_change_and_difference>

## Distributions

Continuous variables can take any value within a range, discrete variables have a fixed set (categorical).



Variance is the measure of how spread out your data is; calculate the distance of each from the mean squared and take the average. Standard deviation doesn’t square the distance.

A quantity expressing by how much the members of a group differ from the mean value for the group OR the Standard Deviation is a measure of how spread out numbers are.



The middle is the mean and so 68.9% appears with standard deviation of 1 from the mean on a normal distribution.

A small standard deviation means that the values in a statistical data set are close to the mean of the data set, on average, and a large standard deviation means that the values in the data set are farther away from the mean, on average.

A small standard deviation can be a goal in certain situations where the results are restricted, for example, in product manufacturing and quality control.

But in situations where you just observe and record data, a large standard deviation isn’t necessarily a bad thing; it just reflects a large amount of variation.

Standard deviation and the mean are heavily affected by outliers.

Skewness - Looking at the fact there’s more outliers on the left or right.



Mean is the average, median can be more useful with outliers, mode used for discrete values. The values are always displayed in this order on left skewed data and vice versa on right skewed.



## Central Limit Theorem



Parameters are set for the whole population; statistics are performed on a sample to make an estimate on the population.

Sampling distribution takes various samples of the population and takes the mean of each.

Central limit theorem states that the sampling distribution of the sample mean will be a normal distribution.

Also the mean of the sampling distribution of the sample mean will be the same as the mean of the original population.

## Z-Score

## P-Values

P-value tests the null hypothesis, e.g. the devil's advocate of whether the same changes are seen without the variable.

A p of .05 means there’s a less than 5 percent chance that in the world where the null hypothesis is true, the results you’re seeing would be due to random chance. This sounds nitpicky, but it’s critical. It’s the misunderstanding that leads people to be unduly confident in p-values. The false-positive rate for experiments at p=.05 can be much, much higher than 5 percent.

## Law of large numbers

As sample sizes grow your measurement will grow closer to the expected measure. E.g. coin toss 0.5.

## Good Sources

<https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/>

<http://www.investopedia.com/articles/financial-theory/09/regression-analysis-basics-business.asp>

<http://www.mathsisfun.com/data/standard-deviation.html>

Cohort Analysis - <http://analyzecore.com/2015/12/10/cohort-analysis-retention-rate-visualization-r/>

# Business Science

Always tie your problem to financial costs – direct (e.g. training, hiring, etc) & indirect (harder to quantify – e.g. productivity).

Business science problem framework & CRISP-DM – data science framework to give a critical logical step checklist, conveys plan to stakeholders and keeps the project on track. Apply/integrate BSPF in the CRISP-DM steps.

CRISP-DM: Cross-Industry Standard Process for Data Mining. Gives a structured approach to data mining projects, generic for many problems. It is iterative, provides data narrative through documentation at different stages, as data develops the business gains a good understanding and continuous integration is built in.

CRISP-DM process cycle:   
1. Business understanding – most critical and fun part. Combines discovery (often with subject matter experts) & project management.  
2. Data understanding – 20% creative data collection.  
3. Data preparation – 60% data cleaning.  
4. Modeling – prediction (classification, regression, clustering & anomaly detection), model understanding and model assessment.  
5. Evaluation – degree to which data science insights meet business objectives. All depends on ROI.  
6. Deployment – automated reports, dashboards & interactive decision making tools.

Business science problem framework:  
- The goal is to make good decisions through systematic decision making without emotion.  
- Identify, measure & analyse problems = improvement.  
- Systematic decision making – identifying the drives > business understanding > systematic decision making.  
- CRISP-DM is high level, whereas as DSPF lets us dive into details.

## Data Science Project Setup

Create good project/directory structure in CRISP-DM format.  
E.g. 00\_Data, 01\_Business\_Understanding, 02\_Data\_Understanding, 03\_Data\_Preparation, etc.

library(fs)

make\_project\_dir <- function() {  
 dir\_names <- c(  
 "00\_Data",   
 "00\_Scripts",   
 "01\_Business\_Understanding",  
 "02\_Data\_Understanding",  
 "03\_Data\_Preparation",  
 "04\_Modeling",  
 "05\_Evaluation",  
 "06\_Deployment")  
 dir\_create(dir\_names)  
 dir\_ls()  
}

pkgs <- c(  
 "h2o", # High performance machine learning  
 "lime", # Explaining black-box models  
 "recipes", # Creating ML preprocessing recipes  
 "tidyverse", # Set of pkgs for data science: dplyr, ggplot2, purrr, tidyr, ...  
 "tidyquant", # Financial time series pkg - Used for theme\_tq ggplot2 theme  
 "glue", # Pasting text  
 "cowplot", # Handling multiple ggplots  
 "GGally", # Data understanding – visualizations  
 "skimr", # Data understanding - summary information  
 "fs", # Working with the file system - directory structure  
 "readxl", # Reading excel files  
 "writexl" # Writing to excel files  
)

## Business Understanding

Need to expose the cost ideas to the rest of the business.

Understand the drivers

Measure the drivers

Need to start thinking of other things at play, collecting data iteratively and developing KPIs.

Descriptive features, Employment features, compensation features, survey results, performance data, work-life features, training & education, time based features. So break down data collection activities into strategic areas.

KPI - organisations goal for how business should run can be internal data or an industry standard.

Uncover problems & opportunities – show and understand aggregated costs.

## Functional Workflow

**count\_to\_pct** <- function(data, ..., col = n) {  
 grouping\_vars\_expr <- quos(...)  
 col\_expr <- enquo(col)

ret <-   
data %>%   
 group\_by(!!! grouping\_vars\_expr) %>%   
 mutate(pct = (!! col\_expr) / sum(!! col\_expr)) %>%   
 ungroup()

return(ret)

}

**assess\_attrition** <- function(data, attrition\_col, attrition\_value, baseline\_pct){  
 attrition\_col\_expr <- enquo(attrition\_col)

data %>%   
 filter((!! attrition\_col\_expr) %in% attrition\_value) %>%   
 arrange(desc(pct)) %>%   
 mutate(  
 Above\_Industry\_Rate = case\_when(  
 pct > baseline\_pct ~ "Yes",  
 TRUE ~ "No"  
 )  
 ) %>%   
 mutate(  
 Cost\_Of\_Attrition = calculate\_attrition\_cost(n = n, salary = 80000)  
 )  
}

**plot\_attrition** <- function(data, ..., .value = cost,  
 fct\_reorder = TRUE,  
 fct\_rev = FALSE,  
 include\_lbl = TRUE,  
 colour = palette\_light()[[1]],  
 units = c("0","K","M")) {

# Inputs   
 group\_vars\_expr <- quos(...)  
 if (length(group\_vars\_expr) == 0)  
 group\_vars\_expr <- quos(rlang::sym(colnames(data)[[1]])) # Takes first column name of data if nothing supplied.

value\_expr <- enquo(.value)  
 value\_name <- quo\_name(value\_expr)

units\_val <- switch(units[[1]],  
 "M" = 1e6,  
 "K" = 1e3,  
 "0" = 1)  
 if (units[[1]] == "0") units <- ""

# Data Manipulation

usd <- scales::dollar\_format(prefix = "$", largest\_with\_cents = 1e3)

data\_manipulated <-  
 data %>%   
 mutate(name = str\_c(!!! group\_vars\_expr, sep = ': ') %>% as.factor()) %>%   
 mutate(value\_text = str\_c(usd(!! value\_expr / units\_val),  
 units[[1]], sep = ""))

if (fct\_reorder) {  
 data\_manipulated <- data\_manipulated %>%   
 mutate(name = fct\_reorder(name, !! value\_expr)) %>%   
 arrange(name)  
}

if (fct\_rev) {  
 data\_manipulated <- data\_manipulated %>%   
 mutate(name = fct\_rev(name)) %>%   
 arrange(name)  
 }

# Visualisation

g <-   
 data\_manipulated %>%   
 ggplot(aes\_string(x = value\_name, y = "name")) +  
 geom\_segment(aes(xend = 0, yend = name), colour = colour) +  
 geom\_point(aes\_string(size = value\_name), colour = colour) +  
 scale\_x\_continuous(labels = scales::dollar) +  
 theme\_tq() +  
 scale\_size(range = c(3, 5)) +  
 theme(legend.position = 'none')

if (include\_lbl) {  
 g <- g +  
 geom\_label(aes\_string(label = "value\_text", size = value\_name),  
 hjust = "inward", colour = colour)   
 return(g)  
 }

}

dept\_job\_role\_tbl %>%   
 count(Department, JobRole, Attrition) %>%   
 count\_to\_pct(Department, JobRole) %>%   
 assess\_attrition(Attrition, attrition\_value = "Yes", baseline\_pct = 0.088) %>%   
 mutate(  
 Cost\_Of\_Attrition = calculate\_attrition\_cost(n = n, salary = 80000)  
 ) %>%   
 plot\_attrition(Department, JobRole, .value = Cost\_Of\_Attrition,  
 units = "M")

## Explore Features

library(skimr)

skim() - quick summary with a little distribution histogram. Good for high level understanding. Separates data types.

Can often see the numerics which should be categrical based on the distribution with big gaps.

Character - if lots of unique data then think of using other category.

**Exploring Character Data:**

train\_raw\_tbl %>%   
 select\_if(is.character) %>%   
 map(unique)

train\_raw\_tbl %>%   
 select\_if(is.character) %>%   
 map(~table(.) %>% prop.table())

**Exploring Numeric Data:**

Discrete – numeric features that aren’t continuous and are actually categorical. Typically should be converted to categorical data types.

Numeric variables lower in levels are likely to be discrete.

train\_raw\_tbl %>%   
 select\_if(is.numeric) %>%   
 map\_df(~unique(.) %>% length()) %>%   
 gather() %>%   
 arrange(value) %>%  
 filter(value <= 10)

**GGally – ggpairs**

Top is usually the most important.

Lower Triangle:

1. Histogram for numeric-categorical pairs.
2. Scatter for numeric-numeric pairs.
3. Bars for categorical-categorical pairs.

Upper Triangle:

1. Box-plot for numeric-categorical pairs.
2. Correlation value for numeric-numeric pairs.
3. Bars for categorical-categorical pairs.

train\_raw\_tbl %>%   
 select(Attrition, Age, Gender, MaritalStatus, NumCompaniesWorked, Over18, DistanceFromHome) %>%   
 ggpairs()

train\_raw\_tbl %>%   
 select(Attrition, Age, Gender, MaritalStatus, NumCompaniesWorked, Over18, DistanceFromHome) %>%   
 ggpairs(aes(colour = Attrition), lower = "blank", legend = 1,  
 diag = list(continuous = wrap("densityDiag", alpha = 0.5))) +  
 theme(legend.position = "bottom")

**With no colour**

dataset %>%  
 ggpairs(lower = "blank", diag = list(continuous = wrap("densityDiag", alpha = 0.5)))

**Function**

plot\_ggpairs <- function(data, color = NULL, density\_alpha = 0.5) {  
 color\_expr <- enquo(color)  
 if (rlang::quo\_is\_null(color\_expr)) {  
 g <- data %>%  
 ggpairs(lower = "blank")   
 } else {  
 color\_name <- quo\_name(color\_expr)  
 g <- data %>%  
 ggpairs(mapping = aes\_string(color = color\_name),   
 lower = "blank", legend = 1,  
 diag = list(continuous = wrap("densityDiag",   
 alpha = density\_alpha))) +  
 theme(legend.position = "bottom")  
 }  
 return(g)  
 }

train\_raw\_tbl %>%  
 select(Attrition, contains("employee"), contains("department"), contains("job")) %>%  
 plot\_ggpairs(Attrition)

## Data Preparation

View()

Handy functions:  
one\_of  
left\_join  
str\_replace\_all

Use lists to collect objects that need to iterated over, use purrr functions to iterate.

Fill() – replace missing values (NAs) with the closest entry previous if direction = “down” or next if direction = “up”.

definitions\_tbl <- definitions\_raw\_tbl %>%  
 fill(X\_\_1, .direction = "down") %>%  
 filter(!is.na(X\_\_2)) %>%  
 separate(X\_\_2, into = c("key", "value"), sep = " '", remove = TRUE) %>%  
 rename(column\_name = X\_\_1) %>%  
 mutate(key = as.numeric(key)) %>%  
 mutate(value = value %>% str\_replace(pattern = "'", replacement = ""))

Split() – splits a data frame into multiple data frames within a list. Supply a column name as a vector, e.g. split(.$column\_name)

definitions\_list <- definitions\_tbl %>% split(.$column\_name) %>%  
 map(~ select(., -column\_name)) %>%  
 map(~ mutate(., value = as\_factor(value)))

for (i in seq\_along(definitions\_list)) {  
 list\_name <- names(definitions\_list)[i]  
 colnames(definitions\_list[[i]]) <- c(list\_name, paste0(list\_name, "\_value"))  
}

reduce() iteratively applies use specified function to successive binary sets of objects. E.g. reduce(left\_join).

data\_merged\_tbl <- list(HR\_Data = train\_raw\_tbl) %>%  
 append(definitions\_list, after = 1) %>%  
 reduce(left\_join) %>%  
 select(-one\_of(names(definitions\_list))) %>%  
 set\_names(str\_replace\_all(names(.), pattern = "\_value", replacement = "")) %>%  
 select(sort(names(.)))

Factoring character data

data\_processed\_tbl <- data\_merged\_tbl %>%  
 mutate\_if(is.character, as.factor) %>%  
 mutate(  
 BusinessTravel = BusinessTravel %>% fct\_relevel("Non-Travel", "Travel\_Rarely", "Travel\_Frequently"),  
 MaritalStatus = MaritalStatus %>% fct\_relevel("Single", "Married", "Divorced")  
 )

**Plot Hist Facet**

plot\_hist\_facet <- function(data, bins = 10, ncol = 5,  
 fct\_reorder = FALSE, fct\_rev = FALSE,   
 fill = palette\_light()[[3]],   
 color = "white", scale = "free") {

data\_factored <- data %>%  
 mutate\_if(is.character, as.factor) %>%  
 mutate\_if(is.factor, as.numeric) %>%  
 gather(key = key, value = value, factor\_key = TRUE)

if (fct\_reorder) {  
 data\_factored <- data\_factored %>%  
 mutate(key = as.character(key) %>% as.factor())  
 }

if (fct\_rev) {  
 data\_factored <- data\_factored %>%  
 mutate(key = fct\_rev(key))  
 }

g <- data\_factored %>%  
 ggplot(aes(x = value, group = key)) +  
 geom\_histogram(bins = bins, fill = fill, color = color) +  
 facet\_wrap(~ key, ncol = ncol, scale = scale) +   
 theme\_tq()

return(g)  
}

train\_raw\_tbl %>%  
 select(Attrition, everything()) %>%  
 plot\_hist\_facet(bins = 10, ncol = 5, fct\_rev = F)

### Pre-Processing With Recipes

(1) create the steps   
(2) prepare the recipe - can see the changes it will make   
(3) bake the new data.

If you prepare the object before baking you can review info such as means and all the steps.

Other steps: step\_num2factor, YeoJohnson

Center & Scale: when in doubt, do it. Make sure you do centering before scaling.  
Centering subtracts out means, scaling takes the range and makes sure all data has the same range.

step\_holiday - convert date data into one or more binary indicator variables for common holidays

Check Skewness, can then use the names in recipes (e.g. step\_YeoJohnson(skewed\_names))

skewed\_names <-  
 list\_datasets[[1]] %>%  
 select\_if(is.numeric) %>%  
 map\_df(skewness) %>%  
 gather(factor\_key = T) %>%  
 mutate(value = abs(value)) %>%  
 filter(value >= 10) %>%  
 arrange(desc(value)) %>%  
 pull(key) %>% as.character()

rec\_obj <-  
 recipe(VipPlayer ~ ., data = list\_modelling[[1]]) %>%  
 step\_YeoJohnson(skewed\_names) %>%  
 step\_center(all\_numeric()) %>%  
 step\_scale(all\_numeric()) %>%  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%  
 prep(data = list\_modelling[[1]])

list\_modelling <-  
 list\_modelling %>%  
 map(~bake(rec\_obj, newdata = .))

### Correlation Analysis

A correlation analysis is a good way to determine if you are getting good features prior to modelling. Always begin with this, it helps guide the process and saves time for everyone involved.

Can also do this on dummied data so you can focus on your Outcome\_Yes.

Might not detect everything going on but it’s a good starting barometer. It’s a linear style of modelling so if it’s exponential it won’t show as much of a correlation.

get\_cor <- function(data, target, use = "pairwise.complete.obs",  
 fct\_reorder = FALSE, fct\_rev = FALSE) {

feature\_expr <- enquo(target)  
 feature\_name <- quo\_name(feature\_expr)

data\_cor <- data %>%  
 mutate\_if(is.character, as.factor) %>%  
 mutate\_if(is.factor, as.numeric) %>%  
 cor(use = use) %>%  
 as.tibble() %>%  
 mutate(feature = names(.)) %>%  
 select(feature, !! feature\_expr) %>%  
 filter(!(feature == feature\_name)) %>%  
 mutate\_if(is.character, as\_factor)

if (fct\_reorder) {  
 data\_cor <- data\_cor %>%   
 mutate(feature = fct\_reorder(feature, !! feature\_expr)) %>%  
 arrange(feature)  
 }

if (fct\_rev) {  
 data\_cor <- data\_cor %>%   
 mutate(feature = fct\_rev(feature)) %>%  
 arrange(feature)  
 }

return(data\_cor)  
}

train\_tbl %>%  
 get\_cor(Attrition\_Yes, fct\_reorder = T, fct\_rev = T)

plot\_cor <- function(data, target, fct\_reorder = FALSE, fct\_rev = FALSE,   
 include\_lbl = TRUE, lbl\_precision = 2, lbl\_position = "outward",  
 size = 2, line\_size = 1, vert\_size = 1,   
 color\_pos = palette\_light()[[1]],   
 color\_neg = palette\_light()[[2]]) {

feature\_expr <- enquo(target)  
 feature\_name <- quo\_name(feature\_exp

data\_cor <- data %>%  
 get\_cor(!! feature\_expr, fct\_reorder = fct\_reorder, fct\_rev = fct\_rev) %>%  
 mutate(feature\_name\_text = round(!! feature\_expr, lbl\_precision)) %>%  
 mutate(Correlation = case\_when(  
 (!! feature\_expr) >= 0 ~ "Positive",  
 TRUE ~ "Negative") %>% as.factor())

g <- data\_cor %>%  
 ggplot(aes\_string(x = feature\_name, y = "feature", group = "feature")) +  
 geom\_point(aes(color = Correlation), size = size) +  
 geom\_segment(aes(xend = 0, yend = feature, color = Correlation), size = line\_size) +  
 geom\_vline(xintercept = 0, color = palette\_light()[[1]], size = vert\_size) +  
 expand\_limits(x = c(-1, 1)) +  
 theme\_tq() +  
 scale\_color\_manual(values = c(color\_neg, color\_pos))

if (include\_lbl) g <- g + geom\_label(aes(label = feature\_name\_text), hjust = lbl\_position)

return(g)   
}

## Modeling

Packages <- c(“cowplot”, “fs”, “glue”)

If it’s performing well on the validation set but then not as well on the cross fold validation set then it’s probably overfitting.

**Auto ML**

automl\_models\_h2o <- h2o.automl(  
 x = x,  
 y = y,  
 training\_frame = train\_h2o,  
 validation\_frame = valid\_h2o,  
 leaderboard\_frame = test\_h2o,  
 max\_runtime\_secs = 30,  
 nfolds = 5  
)

h2o.getModel("StackedEnsemble\_BestOfFamily\_0\_AutoML\_20180503\_035824") %>%  
 h2o.saveModel(path = "04\_Modeling/h2o\_models/")

stacked\_ensemble\_h2o <- h2o.loadModel("04\_Modeling/h2o\_models/StackedEnsemble\_BestOfFamily\_0\_AutoML\_20180503\_035824")

predictions <- h2o.predict(stacked\_ensemble\_h2o, newdata = as.h2o(test\_tbl))  
predictions\_tbl <- predictions %>% as.tibble()

# **Extracts any H2O model name** by a position so can more easily use h2o.getModel()

extract\_h2o\_model\_name\_by\_position <- function(h2o\_leaderboard, n = 1) {  
 model\_name <- h2o\_leaderboard %>%  
 as.tibble() %>%  
 slice(n) %>%  
 pull(model\_id)

print(model\_name)

return(model\_name)  
}

automl\_models\_h2o@leaderboard %>%  
 extract\_h2o\_model\_name\_by\_position(2) %>%  
 h2o.getModel()

# **Visualize the H2O leaderboard** to help with model selection

plot\_h2o\_leaderboard <- function(h2o\_leaderboard, order\_by = c("auc", "logloss"),   
 n\_max = 20, size = 4, include\_lbl = TRUE) {

# Setup inputs  
 order\_by <- tolower(order\_by[[1]])

leaderboard\_tbl <- h2o\_leaderboard %>%  
 as.tibble() %>%  
 mutate(model\_type = str\_split(model\_id, "\_", simplify = T) %>% .[,1]) %>%  
 rownames\_to\_column(var = "rowname") %>%  
 mutate(model\_id = paste0(rowname, ". ", as.character(model\_id)) %>% as.factor())

# Transformation  
 if (order\_by == "auc") {

data\_transformed\_tbl <- leaderboard\_tbl %>%  
 slice(1:n\_max) %>%  
 mutate(  
 model\_id = as\_factor(model\_id) %>% reorder(auc),  
 model\_type = as.factor(model\_type)  
 ) %>%  
 gather(key = key, value = value,   
 -c(model\_id, model\_type, rowname), factor\_key = T)  
 } else if (order\_by == "logloss") {  
 data\_transformed\_tbl <- leaderboard\_tbl %>%  
 slice(1:n\_max) %>%  
 mutate(  
 model\_id = as\_factor(model\_id) %>% reorder(logloss) %>% fct\_rev(),  
 model\_type = as.factor(model\_type)  
 ) %>%  
 gather(key = key, value = value, -c(model\_id, model\_type, rowname), factor\_key = T)  
 } else {  
 stop(paste0("order\_by = '", order\_by, "' is not a permitted option."))  
 }

# Visualization  
 g <- data\_transformed\_tbl %>%  
 ggplot(aes(value, model\_id, color = model\_type)) +  
 geom\_point(size = size) +  
 facet\_wrap(~ key, scales = "free\_x") +  
 theme\_tq() +  
 scale\_color\_tq() +  
 labs(title = "Leaderboard Metrics",  
 subtitle = paste0("Ordered by: ", toupper(order\_by)),  
 y = "Model Postion, Model ID", x = "")

if (include\_lbl) g <- g + geom\_label(aes(label = round(value, 2), hjust = "inward"))

return(g)  
}

**K-Fold Cross Validation**

More stable models with increased generalisation.

Splits training set into k folds, e.g. 5 training sets & 5 validation (test) sets in 80/20 split.

Same parameters but not necessarily same model on each fold. Then gets the average AUC from each model. This process can then be performed again with new modelling parameters (grid search – iterative hyperparameter tuning), so then we can assess which parameters to use in the final model.

5-fold cross validation: 6 models generated. First 5 are CV models to get performance of model parameters, 6th is generated using model parameters on entire training dataset.

For all except stacked ensembles – h2o.cross\_validation\_models(deeplearning\_h2o)

h2o.auc(deeplearning\_h2o, train = T, valid = T, xval = T)

**Grid (Hyperparameter) Search**

Creating grid or matrix of numerous combinations of model parameters, then run model with all combinations to see which gets best results.

Hyperparameter is just a fancy name for model parameters.

Grid search methods: Cartesian (the normal way) or Random.

Epochs - number of times each batch within deep learning model is trained. Important for generalising.

Hidden - designating how many units in each layer

deeplearning\_h2o <- h2o.loadModel("04\_Modeling/h2o\_models/DeepLearning\_0\_AutoML\_20180503\_035824")

h2o.performance(deeplearning\_h2o, newdata = as.h2o(test\_tbl))

deeplearning\_grid\_01 <- h2o.grid(  
 algorithm = "deeplearning",  
 grid\_id = "deeplearning\_grid\_01",  
 # h2o.deeplearning()  
 x = x,  
 y = y,  
 training\_frame = train\_h2o,  
 validation\_frame = valid\_h2o,  
 nfolds = 5,   
 hyper\_params = list(  
 hidden = list(c(10, 10, 10), c(50, 20, 10), c(20, 20, 20)),  
 epochs = c(10, 50, 100)  
 )  
)

deeplearning\_grid\_01

h2o.getGrid(grid\_id = "deeplearning\_grid\_01", sort\_by = "auc", decreasing = TRUE)

deeplearning\_grid\_01\_model\_3 <- h2o.getModel("deeplearning\_grid\_01\_model\_3")

deeplearning\_grid\_01\_model\_3 %>% h2o.auc(train = T, valid = T, xval = T)

deeplearning\_grid\_01\_model\_3 %>%  
 h2o.performance(newdata = as.h2o(test\_tbl))

Good resource on GBM tuning in h2o - <https://blog.h2o.ai/2016/06/h2o-gbm-tuning-tutorial-for-r/>

## Assessing Model Performance

AUC: Area Under the Curve. Way of measuring binary classifier by comparing False Positive Rate to True Positive Rate. Refers to a Receiver Operating Characteristics (ROC) plot.

Logloss: Logarithmic loss. Measures performance of classifier by comparing class probability to actual value (1 or 0).   
Compares the prediction probability to the 1/0 actual value computing the mean error. This is a great way to measure the true performance of a classifier.

AUC is used most in data science but logloss is often the best to use.

Predictions:  
predictions <- h2o.predict(stacked\_ensemble\_h2o, newdata = as.h2o(test\_tbl))  
predictions\_tbl <- predictions %>% as.tibble()

Performance:  
performance\_h2o <- h2o.performance(stacked\_ensemble\_h2o, newdata = as.h2o(test\_tbl))  
performance\_h2o@metrics

**Classifier Summary Metrics**

h2o.auc(stacked\_ensemble, train = T, valid = T, xval = T)  
h2o.auc(performance\_h2o)  
h2o.giniCoef(performance\_h2o)  
h2o.logloss(performance\_h2o)

Usually use AUC, logloss & precision/recall.

h2o.confusionMatrix(stacked\_ensemble\_h2o)  
h2o.confusionMatrix(performance\_h2o)

Confusion matrix – focus on threshold, precision & recall.  
Top row: Prediction; Left-column: Actual.

**Precision, Recall, F1 & Effect of Threshold**

Threshold: value that determines class probability of 0 or 1. With the metric function we can look at how metrics change with different thresholds.

Important Measures That Vary By Threshold:  
F1: Optimal balance between precision & recall. Typically the threshold that maximises F1 is used as threshold/cutoff for turning class probability into 0/1, but not always the case.  
Precision: Measures false positives (e.g. predicted to do something but don’t). So out of all the times it predicted ‘yes’, how many were correct.  
Recall: Measures false negatives (e.g. predicted not to do something but actually do). So it provides a metric for under-picking ‘yes’, so how often did we miss people by incorrectly saying they would be ‘Yes’.

Recall is typically more important than precision in business. We would rather give up some false positives (inadvertently targeting stayers) to gain false negatives (accurately predict leavers).

F1 optimises balance between precision and recall but that’s not always the best because there are different costs associated with false positives and false negatives. Typically false positives cost the company more, this is where expected value comes in.

performance\_tbl <- performance\_h2o %>%  
 h2o.metric() %>%  
 as.tibble()

performance\_tbl %>%  
 filter(f1 == max(f1))

Visualise Precision & Recall vs Threshold

performance\_tbl %>%  
 ggplot(aes(x = threshold)) +  
 geom\_line(aes(y = precision), color = "blue", size = 1) +  
 geom\_line(aes(y = recall), color = "red", size = 1) +  
 geom\_vline(xintercept = h2o.find\_threshold\_by\_max\_metric(performance\_h2o, "f1")) +  
 theme\_tq() +  
 labs(title = "Precision vs Recall", y = "value")

### For Data Scientists

ROC Plot

Plotted tpr vs fpr

load\_model\_performance\_metrics <- function(path, test\_tbl) {

model\_h2o <- h2o.loadModel(path)  
 perf\_h2o <- h2o.performance(model\_h2o, newdata = as.h2o(test\_tbl))

perf\_h2o %>%  
 h2o.metric() %>%  
 as.tibble() %>%  
 mutate(auc = h2o.auc(perf\_h2o)) %>%  
 select(tpr, fpr, auc)  
}

model\_metrics\_tbl <- fs::dir\_info(path = "04\_Modeling/h2o\_models/") %>%  
 select(path) %>%  
 mutate(metrics = map(path, load\_model\_performance\_metrics, test\_tbl)) %>%  
 unnest()

model\_metrics\_tbl %>%  
 mutate(  
 path = str\_split(path, pattern = "/", simplify = T)[,3] %>% as\_factor(),  
 auc = auc %>% round(3) %>% as.character() %>% as\_factor()  
 ) %>%  
 ggplot(aes(fpr, tpr, color = path, linetype = auc)) +  
 geom\_line(size = 1) +  
 theme\_tq() +  
 scale\_color\_tq() +  
 theme(legend.direction = "vertical") +  
 labs(  
 title = "ROC Plot",  
 subtitle = "Performance of 3 Top Performing Models"  
 )

Precision vs Recall Plot

load\_model\_performance\_metrics <- function(path, test\_tbl) {

model\_h2o <- h2o.loadModel(path)  
 perf\_h2o <- h2o.performance(model\_h2o, newdata = as.h2o(test\_tbl))

perf\_h2o %>%  
 h2o.metric() %>%  
 as.tibble() %>%  
 mutate(auc = h2o.auc(perf\_h2o)) %>%  
 select(tpr, fpr, auc, precision, recall)  
}

model\_metrics\_tbl <- fs::dir\_info(path = "04\_Modeling/h2o\_models/") %>%  
 select(path) %>%  
 mutate(metrics = map(path, load\_model\_performance\_metrics, test\_tbl)) %>%  
 unnest()

model\_metrics\_tbl %>%  
 mutate(  
 path = str\_split(path, pattern = "/", simplify = T)[,3] %>% as\_factor(),  
 auc = auc %>% round(3) %>% as.character() %>% as\_factor()  
 ) %>%  
 ggplot(aes(recall, precision, color = path, linetype = auc)) +  
 geom\_line(size = 1) +  
 theme\_tq() +  
 scale\_color\_tq() +  
 theme(legend.direction = "vertical") +  
 labs(  
 title = "Precision vs Recall Plot",  
 subtitle = "Performance of 3 Top Performing Models"  
 )

### For Business

Gain & Lift - emphasises how much model improves results. Results based metrics which help communicated modeling results in terms everyone cares about.

Gain example: without model we'd expect global attrition rate of first ten to be 16% of 1.6 people wherea we can predict 90%.

Lift: ratio between what we gained vs expectation. E.g. using above example 9/1.6 = 5.6x in first 10 cases.

Lift is basically meaning your model is .. times better targeting than random guessing.

Grouping into cohorts is at the heart of the Gain/Lift Chart. E.g. when we do this we can immediately show if a candidate has a high probability of leaving, how likely they are to leave.

**Manual Calculation**

ranked\_predictions\_tbl <-  
 predictions\_tbl %>%  
 select(predict:Yes, Attrition) %>%  
 arrange(desc(Yes))

calculated\_gains\_lift <-  
 ranked\_predictions\_tbl %>%  
 mutate(ntile = ntile(Yes, n = 10)) %>%  
 group\_by(ntile) %>%  
 summarise(  
 cases = n(),  
 responses = sum(Attrition = 'Yes')  
 ) %>%  
 arrange(desc(ntile)) %>%  
 mutate(group = row\_number()) %>%  
 select(group, cases, responses) %>%  
 mutate(  
 cumulative\_responses = cumsum(responses),  
 pct\_responses = responses / sum(responses),  
 gain = cumsum(pct\_responses),  
 cumulative\_pct\_cases = cumsum(cases) / sum(cases),  
 lift = gain / cumulative\_pct\_cases,  
 gain\_baseline = cumulative\_pct\_cases,  
 lift\_baseline = gain\_baseline / cumulative\_pct\_cases  
 )

**H2o Gain & Lift**

Group – cohort similar to our ntile, first 16.  
Cumulative\_data\_fraction – cumulative pct cases, just accumulates as we get more cases.   
Cumulative\_capture\_rate – this is the cumulative percentage GAIN we will use.  
Cumulative\_lift = LIFT

gain\_lift\_tbl <- performance\_h2o %>%  
 h2o.gainsLift() %>%  
 as.tibble()

gain\_transformed\_tbl <-   
 gain\_lift\_tbl %>%   
 select(group, cumulative\_data\_fraction, cumulative\_capture\_rate, cumulative\_lift) %>%  
 select(-contains("lift")) %>%  
 mutate(baseline = cumulative\_data\_fraction) %>%  
 rename(gain = cumulative\_capture\_rate) %>%  
 gather(key = key, value = value, gain, baseline)

gain\_transformed\_tbl %>%  
 ggplot(aes(x = cumulative\_data\_fraction, y = value, color = key)) +  
 geom\_line(size = 1.5) +  
 theme\_tq() +  
 scale\_color\_tq() +  
 labs(  
 title = "Gain Chart",  
 x = "Cumulative Data Fraction",  
 y = "Gain"  
 )

lift\_transformed\_tbl <-   
 gain\_lift\_tbl %>%   
 select(group, cumulative\_data\_fraction, cumulative\_capture\_rate, cumulative\_lift) %>%  
 select(-contains("capture")) %>%  
 mutate(baseline = 1) %>%  
 rename(lift = cumulative\_lift) %>%  
 gather(key = key, value = value, lift, baseline)

lift\_transformed\_tbl %>%  
 ggplot(aes(x = cumulative\_data\_fraction, y = value, color = key)) +  
 geom\_line(size = 1.5) +  
 theme\_tq() +  
 scale\_color\_tq() +  
 labs(  
 title = "Lift Chart",  
 x = "Cumulative Data Fraction",  
 y = "Lift"  
 )

# **Excel**

## General Tips

Issues with Date Conversion - rather than opening your CSV file, create a new workbook, then on the DATA tab, select Get External Data → From Text. This gets to the interface where you can specify how to interpret your text data, including how to handle dates.

Can also use LEFT and MID functions to extract different parts of the dates and then put it all together.

## Charts

Change min & max axis – Click the numbers on the axis – Vertical Axis Options – Axis Options – Bounds.

Design – Add Chart Element

Numbers need to be vertical for charts, e.g. line charts.

Always fix the charts and slicers not to move with cells – Properties – Don’t move size with cells.

Hold Alt when re-sizing and it will snap to the grid lines.

## Pivot Tables

Pivot table – Analyze – Fields, Items & Sets – Calculated Fields

Click in pivot table – Design – Report Layout – Repeat All Item Labels

Analyze – Insert Slicer; can also link multiple tables and charts to slicer.

Pivot Table Options - Data - Number of items to retain per field: None

Analyze – Fields – Calculated Fields

## Other Tips / Ideas

Highlight whole row and click View – Freeze Panes (will freeze all rows above highlighted row).

Insert – Symbol – Windings – Tick at bottom

Create a file with a pivot table that links to external source data, can then update either the files or user can just use the pivot skeleton and select new data source.

## Formulas

CTRL + ALT + SHIFT + F9 to recheck all formula dependencies and then recalculate all formulas.

Nested IF from Bannertag  
=IF(P2=Q2,"Match",IF(Q2="","Unknown",IF(Q2="Unknown","Unknown","No Match")))

Datedif  
=DATEDIF(D2,TODAY(),"d")

IF Datedif  
=IF(DATEDIF(D2,TODAY(),"m")>=6,"6+ Months",IF(DATEDIF(D2,TODAY(),"m")>=3,"3-6 Months","Less Than 3 Months"))

IF First 4 are the same  
=IF((LEFT(C2,4)=LEFT(C3,4)),"Similar", "Not Similar")

Get characters up until character  
=LEFT(A1,FIND("\_",A1&":")-1)

Nested IF for finding numbers in field  
=IF(ISNUMBER(SEARCH("1",B2)),"Yes",IF(ISNUMBER(SEARCH("",B2)),"Yes","No"))

## Macros

**Heatmap**

Sub test()  
Dim spread As Double, ratio As Double, red As Double, \_  
 green As Double, blue As Double  
For i = 2 To 916  
 Set Rng = Range(Cells(i, 4), Cells(i, 8000))  
 minn = Application.WorksheetFunction.Min(Rng)  
 maxx = Application.WorksheetFunction.Max(Rng)  
 spread = maxx – minn  
 If (spread > 0) Then  
 For j = 4 To 8000  
 If (Cells(i, j) <> "") Then  
 blue = 0  
 qreen = 0  
 red = 0  
 ratio = (Cells(i, j) - minn) / spread  
 If ratio < 0.25 Then  
 blue = 1  
 green = 4 \* ratio  
 ElseIf ratio < 0.5 Then  
 green = 1  
 blue = 1 + 4 \* (minn - Cells(i, j) + 0.25 \* spread) / spread  
 ElseIf ratio < 0.75 Then  
 green = 1  
 red = 4 \* (Cells(i, j) - minn - 0.5 \* spread) / spread  
 Else  
 red = 1  
 green = 1 + 4 \* (minn - Cells(i, j) + 0.75 \* spread) / spread  
 End If  
 Cells(i, j).Interior.Color = RGB(red \* 255, green \* 255, blue \* 255)  
 End If  
 Next j  
 Else  
 For j = 4 To 8000  
 If (Cells(i, j) <> "") Then  
 blue = 1  
 qreen = 0  
 red = 0  
 Cells(i, j).Interior.Color = RGB(red \* 255, green \* 255, blue \* 255)  
 End If  
 Next j  
 End If  
Next i  
End Sub

## Good Sources

<http://www.pryor.com/blog/working-with-multiple-data-series-in-excel/>

Multiple series in one line chart - <https://peltiertech.com/multiple-series-in-one-excel-chart/>

# **Power BI/DAX**

[https://www.daxpatterns.com/dynamic-segmentation/](https://eur04.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwww.daxpatterns.com%2Fdynamic-segmentation%2F&data=02%7C01%7C%7C8cf4a6daa21444004e9008d6032a36ab%7C84df9e7fe9f640afb435aaaaaaaaaaaa%7C1%7C0%7C636699877178979661&sdata=xAFIbhbj07BrV1%2Fc1iXcrNqnHJA%2BfFKjLfxFFy4IF2E%3D&reserved=0)

## Power Pivot

Allows you to pull all the data down and store it in Excel. You can then link multiple tables with joins and build dashboards on this.

Get External Data – call a proc or look at specific tables.

In my past case – created a WHILE loop proc to insert all company info into table twice per day. Then created another proc to select from this which we used in the Power Pivot connection. Then can create filters and dashboard in excel.

Get data into database model – then create relationships, Power Pivot has row context.

Formatting in the pivots follows your formatting in the tables, likewise you want to try and create explicit measurements in the actual data model as new columns rather than doing the sums in the pivot (implicit).

Create calculated fields in the bottom section – known as Measures or Explicit Formulas. Then Power Pivot knows filter context of these totals.

IMPORTANT – the relationships are treated like cross joins so you must have measures for these to work.

Measures = calculating new columns from existing ones; calculated fields = sums of the existing columns.

Sorting in the pivot – need to do it on more granular levels before doing it on totals or it won’t work. Then go back to slicers and choose A-Z rather than order by data source.

### Formulas

Always follow this convention:   
=RELATED(tablename[column])

Calculated field  
Total Net Revenue: = SUM(tablename[column])

Split string into different columns – this splits the column by “|”, the second of these is telling it to split and gets the 2nd element.   
=PATHITEM( SUBSTITUTE( SubidTable[BannerTag], "-", "|" ), 1 )  
=PATHITEM( SUBSTITUTE( SubidTable[BannerTag], "|", "|" ), 2 )  
=PATHITEM( SUBSTITUTE( SubidTable[BannerTag], "-", "|",1 ), 2 ) #the 1 is the instance

Solve divide by zeros:  
DIVIDE(total1,total2,0)

# **SQL**

Custom colours – CustomObsidian  
<http://www.jimmcleod.net/blog/index.php/2012/06/26/beautify-your-management-studio-2012/>

Look at Visual Studio Code or Apex SQL.

## Basics

### Syntax

INSERT INTO (existing table name)  
SELECT (DISTINCT)  
- datepart, monthname, year  
- sum()  
INTO (table name)  
FROM  
JOIN (INNER JOIN & LEFT JOIN)  
- on  
WHERE  
- like '%', 'w\_x%', '[adr]%', '[^a-f]%'  
- in ('name', 'name, 'name')  
- > = < <> !=  
GROUP BY  
HAVING  
ORDER BY (Default ASC oldest to newest / smallest to biggest)

### Shortcuts

Ctrl+N = New query  
Ctrl+L = Estimated Execution Plan  
Ctrl+M = Actual Execution Plan  
Ctrl+R = Results  
Alt + F1 = exec sp\_help  
F4 = Properties

## General Info

### Databases

OLTP (On-line Transaction Processing) - is characterized by a large number of short on-line transactions (INSERT, UPDATE, DELETE)

OLAP (On-line Analytical Processing) - is characterized by relatively low volume of transactions. Queries are often very complex and involve aggregations.

### Table Structure

A partition is simply a way DBAs can sub-divide rows in a table (and the table’s indexes) into separate sections, generally for the purpose of horizontally spreading rows across more than one filegroup.

Partitions contain data rows stored in either the physical form of a heap (a table without a clustered index) or a B-Tree structure (a table with a clustered index). We will go into a more detailed explanation of these structures in the next section.



An extent is a collection of eight contiguous 8K pages (for a total of 64K).

Every SQL Server page (8,192 bytes) begins with a 96-byte header used to store metadata about the page. This includes the page’s number, page type, amount of free space on the page, the allocation unit ID of the object that owns the page, among other metadata.

Each page can store a maximum of 8,060 bytes. The number of rows that can be stored on a page depends on the size of the rows.



A heap is simply a table without a clustered index. (We will talk about clustered indexes later in this article). When rows are added to a heap, they are not stored in any particular order on a data page, and data pages are not stored in any particular sequence within a database. In other words, rows are stored wherever there is room available. This means that the data pages that contain the rows of the heap may be scattered throughout a database, in no particular order.

Since a table can’t exist as a bunch of scattered pages, SQL Server provides a way to link them all together so that they act as a single table. This is done using what are called Index Allocation Map (IAM) pages. IAM pages manage the space allocated to heaps (among other tasks), and is what is used to connect the scattered pages (and their rows) into a table.

### Indexes

An index is made up of a set of pages (index nodes) that are organized in a B-tree structure.

Query against an indexed column navigates down and gets more granular as it goes.

The leaf node will contain either the entire row of data or a pointer to that row, depending on whether the index is clustered or nonclustered.

Indexes can help or hinder performance, especially when performing lots of UPDATES, INSERTS or DELETES.

If a heap has a non-clustered index on it (as the primary key), and data is inserted into the table, two writes have to occur. One write for inserting the row, and one write for updating the non-clustered index. On the other hand, if a table has a clustered index as the primary key, inserts take only one write, not two writes

Tips:   
> Try to have as few duplicates in key columns as possible, if composite then this means both combined (index selectivity).   
> In composite indexes, use the most unique values and most common comparison expressions first.  
> Create nonclustered indexes on columns used frequently in your statement’s predicates and join conditions.  
> Consider indexing columns used in exact-match queries.



Clustered

A clustered index stores the actual data rows at the leaf level of the index.

Because a clustered index is both an index and the rows of the table, whenever you execute a query against the data using the key column as your selection criteria, you can quickly return the data because the data is part of the index.

There can be only one clustered index on a table or view. In addition, data in a table is sorted only if a clustered index has been defined on a table.

Logical ordering doesn’t equal physical ordering – pages may have to be split etc depending on row size.

Tips: Use keys (usually on PK) which you use most commonly to identify and join, these INCLUDE the whole data in the row. The columns should be fairly stable order.

Nonclustered

Unlike a clustered index, the leaf nodes of a nonclustered index contain only the values from the indexed columns and row locators that point to the actual data rows, rather than contain the data rows themselves. So basically, a non-clustered index includes the key value and a bookmark (pointer) that tells SQL Server where to find the actual row in the clustered index.

If referencing a clustered table, the row locator points to the clustered index, using the value from the clustered index to navigate to the correct data row. If referencing a heap, the row locator points to the actual data row.

SQL Server uses a Key Lookup to retrieve non-key data from the data page when a nonclustered index is used to resolve the query. That is, once SQL Server has used the nonclustered index to identify each row that matches the query criteria, it must then retrieve the column information for those rows from the data pages of the table using the clustered index.

As the number of rows in the result set increases, so does the number of Key lookups. At some point, the cost associated with the Key Lookup will outweigh any benefit provided by the nonclustered index.

An index that contains all information required to resolve the query is known as a “Covering Index”; it completely covers the query. This is basically including additional, non-key columns in the leaf level of the nonclustered index.

Only the key columns in the leaf nodes, don't put columns that are already in the clustered index. Usually use what's in your join or predicate and then include other columns (covering index). Included columns are not part of the key but data is stored in leaf nodes.

While a non-clustered index can be added to a heap to speed up some queries, when the non-clustered index is non-covering, the use of a RID bookmark lookup is required.

You can create more than one nonclustered index per table or view.

### Query Tuning

Use both Statistics IO and Execution plan together.

Statistics IO

STATISTICS IO provides detailed information about the impact that your query has on SQL Server. The ideal solution is to use the least number of logical reads to perform your operation. The fewer the logical reads, the faster the response and the lesser the impact on the Server.

**Scan Count -** This number tells us that the optimizer has chosen a plan that caused this object to be read repeatedly. This number is used as a gauge later on in the process and you will see what object it is being scanned when I go over the execution plan. This number does not change unless you alter the query.

**Logical Reads -** This number tells us the actual number of pages read from the data cache. This is the number to focus on because it does not change unless you change the actual query structure or index structures. Most common changes are the joins within the WHERE clause, parameter values, or index structures.

Physical Reads - This is the number of pages actually read from the disk. These are the pages that weren’t already in cache and so it is an interesting figure to monitor as it has a direct effect on the performance of the query. SQL Server does all of its work within its caches. If there is a requested page that is not in cache, it will read it from disk and place it in cache, then use that page. Because the physical reads change based upon memory pressure and not query design, I tend to ignore this figure.

Read-Ahead Reads - This number tells us how many of the physical reads were satisfied by SQL Servers ‘Read-ahead’ mechanism. Ignore this.

LOB Logical Reads - Should the query you are tuning at the time request large objects, you will see this number grow. Pay attention to this number, just like the Logical Reads show above.

LOB Physical Reads - This is the number of physical reads the server performed to fetch the necessary pages to satisfy the query. Again, being physical we have no control over this. Ignore it.

LOB Read-Ahead Reads - This represents the number of physical reads satisfied by the Read-Ahead mechanism. Nothing you can affect, without tuning the physical server, nothing to look to tune.

Execution Plans

The Predicate section shows you the all the pieces this index uses to qualify a row. It will show you any known values, value ranges, and joins to other tables (and on what columns).

The Output List shows you what columns the index will be returning. If the index can satisfy all of the columns requested, it is considered a ‘Covered’ index, otherwise the index will need to get the additional columns from the underlying structure.

The final section is Seek Predicates, which shows the actual columns, values, and criteria (<,>,=) used to satisfy the seek. If this where an Index Scan you would not see the Seek Predicates section.

Difference between seek predicate and predicate: Seek Predicate is the operation that describes the b-tree portion of the Seek. Predicate is the operation that describes the additional filter using non-key columns. Based on the description, it is very clear that Seek Predicate is better than Predicate as it searches indexes whereas in Predicate, the search is on non-key columns – which implies that the search is on the data in page files itself.



### Query Performance

Three basic rules for writing T-SQL:

1. Write to your data structures and also take advantage of foreign key constraints and other structures.
2. Write for your indexes.
3. Write for the optimizer: The query optimizer is an amazing piece of software. But, you can overwhelm it by writing code that isn’t configured best to support it, such as nesting views within views within views, etc. Take the time to understand how the optimizer works and write your code in such a way that you help it, not hurt it.

## Apply

Apply operator serves us execute sub query for each row in a main query however the sub query is a function or else with no nullable parameter.

The APPLY operator can be a useful tool when you want to evoke a table-valued function for each row returned by a table expression (the outer table). You simply use the operator to join the outer table to the function.

Great when you have more than one record, e.g. multiple records for same MasterID in temp table, will only do the sum once per row.

CROSS APPLY (Inner) & OUTER APPLY (Outer) – syntax looks similar to exists.

SELECT t.account,  
 pur.Purchases  
FROM #TempTable t  
CROSS APPLY (  
 SELECT p.account,  
 SUM ( p.purchaseamount ) as Purchases  
 FROM purchases p  
 WHERE t.account = p.account  
 GROUP BY p.account  
 ) pur

## Altering Tables

CREATE INDEX index\_name   
ON table\_name (column\_name)

ALTER TABLE #Players  
ADD CONSTRAINT Pk\_User PRIMARY KEY CLUSTERED ( GamingServerID, UserID )

ALTER TABLE table\_name  
 ALTER COLUMN column\_name column\_type;

## Windowed Functions

ROW\_NUMBER () OVER ( PARTITION BY UserID ORDER BY UserID )  
ROW\_NUMBER () OVER ( ORDER BY ( SELECT 1 ) ) – If you don’t want an inherent order.

## Text/String Functions

SUBSTRING ( expression ,start , length ) - expression is the selection, start is where the returned characters starts, length is how many characters.

SELECT LEFT (emailaddress, CHARINDEX ( '@', emailaddress ) – 1 )   
It seeks the position of the @, and takes the number of characters up to but not including (that's the - 1) the @ symbol.

SELECT SUBSTRING(@FullName, 1,CHARINDEX('java2s', @FullName) - 1)

Get text before the comma:  
CASE WHEN Hardware LIKE '%,%' THEN SUBSTRING(Hardware , 0, charindex(',', Hardware , 0)) ELSE Hardware END

TRIM/RTRIM/LTRIM - gets rid of blank spaces.

Count number of times as character occurs in a string:  
SELECT LEN(@CoreTag) - LEN(REPLACE(@CoreTag, '\_', ''))

String function I use for tags:  
 SELECT @BannerTag as BannerTag,  
 util.[dbo].[fn\_GetNthDelimitedValue](@BannerTag,'-',1,0) as CoreTag,  
 LEFT( SUBSTRING(@BannerTag,charindex('-',@BannerTag,1)+1,CHARINDEX('|',@BannerTag,1)-CHARINDEX('-',@BannerTag,1)-1) ,100) as SubID1,  
 util.[dbo].[fn\_GetNthDelimitedValue](@BannerTag,'|',2,0) as SubID2,  
 util.[dbo].[fn\_GetNthDelimitedValue](@BannerTag,'|',3,0) as SubID3,  
 util.[dbo].[fn\_GetNthDelimitedValue](@BannerTag,'|',4,0) as SubID4,  
 util.[dbo].[fn\_GetNthDelimitedValue](@BannerTag,'|',5,1) as SubID5

**STUFF** results into one column

STUFF ( character\_expression , start , length , replaceWith\_expression )

*(1)*

REPLACE ( STUFF ( ( SELECT DISTINCT ' / ' + d2.TypeofMatch  
FROM #Duplicates d2  
WHERE d2.GamingServerID = d.GamingServerID AND d2.UserID = d.UserID  
FOR XML PATH ('')  
), 1, 2, ''), '&amp;', '&') as TypeOfMatch

*(2)*

STUFF (( SELECT DISTINCT ', ' + RTRIM ( d2.DupAccountNo )  
FROM #Duplicates d2  
WHERE d2.GamingServerID = d.GamingServerID  
AND d2.UserID = d.UserID  
FOR XML PATH ('')  
), 1, 1, '') as DuplicateAccounts

*(3)*

STUFF ( ( SELECT DISTINCT '/' + IpCountry  
FROM #iptmp i2  
WHERE i2.USERID = i1.USERID  
AND i2.CasinoId = i1.CasinoId  
FOR XML PATH('')  
 ), 2, 1000 )

*(4)*

SELECT lm.MasterID,  
 lm.LinkedMasterID,  
 STUFF((  
 SELECT DISTINCT  
 TOP 20  
 '/ ' + RTRIM ( lm2.LinkType )  
 FROM #LinkedMids lm2  
 WHERE lm.LinkedMasterID = lm2.LinkedMasterID  
 FOR  
 XML PATH('')  
 ), 1, 1, '')  
 FROM #LinkedMids lm  
 GROUP BY lm.MasterID,  
 lm.LinkedMasterID

---

STRING\_AGG ( expression, separator ) [ <order\_clause> ]  
E.g. SELECT STRING\_AGG (name, ',') WITHIN GROUP (ORDER BY name ASC)

## Rolling Sums & Date Diffs

1. Rolling sum by calendar month is easy – DATEPART month and year, then assign row\_number based on these and join back to itself.  
   E.g.   
   INTO TEMP TABLE -  
   DATEPART ( MM,… ) as Month,

DATEPART ( YYYY,…) as Year  
ROW\_NUMBER () OVER ( ORDER BY d.year, d.month )  
  
THEN – JOIN TEMP TABLE BACK TO ITSELF  
> SELECT nw2.Year,

nw2.Month,

nw2.NetWin,

SUM ( nw1.NetWin ) as CumulativeNetWin

INTO #CumulativeNetWin

FROM #NetWin nw1

JOIN #NetWin nw2

ON nw1.RN <= nw2.RN

GROUP BY nw2.Year,

nw2.Month,

nw2.NetWin

ORDER BY nw2.Year,

nw2.Month

1. If you need this to be specific to a date rather than calendar month or week, could create a temp table and fill up the info based on each user. Then do grouping at the end by Month/Year.  
   E.g. SUM ( CASE WHEN pur.PRTime <= DATEADD ( MM, 1, nrp.DateOpened ) THEN pur.Amount\_USD END ) as Month1, SUM ( CASE WHEN pur.PRTime <= DATEADD ( MM, 2, nrp.DateOpened ) THEN pur.Amount\_USD END ) as Month2
2. Recursive CTEs.
3. ROWS UNBOUNDED PRECEDING – SQL 2012 and later.

**Date Diff:**

Created a date diff between each row but only between transactions on the same day. To do this I created a table with ROW\_NUMBER and ROW\_NUMBER by DOY.

IF OBJECT\_ID ('tempdb..#TempTable') IS NOT NULL  
 DROP TABLE #TempTable

SELECT p.CustomerKey,  
 p.PRTime,  
 ROW\_NUMBER () OVER ( PARTITION BY p.CustomerKey  
 ORDER BY p.PRTime ) as RN,  
 ROW\_NUMBER () OVER ( PARTITION BY p.CustomerKey, DATEPART ( DAYOFYEAR, p.PRTime )  
 ORDER BY p.PRTime ) as DOYRN  
 INTO #TempTable  
FROM #CurrentWeekPurchases p

IF OBJECT\_ID ('tempdb..#TimeBetweenPurchases') IS NOT NULL  
 DROP TABLE #TimeBetweenPurchases

SELECT tt1.CustomerKey,  
 tt1.PRTime,  
 tt1.RN,  
 tt1.DOYRN,  
 DATEDIFF ( MINUTE, x.PRTime, tt1.PRTime ) as DateDiff  
 INTO #TimeBetweenPurchases  
FROM #TempTable tt1  
OUTER APPLY (  
 SELECT TOP 1 tt2.PRTime  
 FROM #TempTable tt2  
 WHERE tt2.RN < tt1.RN  
 AND tt2.CustomerKey = tt1.CustomerKey  
 ORDER BY tt2.rn DESC  
 ) x  
WHERE tt1.DOYRN <> 1  
ORDER BY tt1.CustomerKey,  
 tt1.RN

## Rolling Retention/Cohort Analysis

You can calculate rolling retention against any cohort window with any behavioural condition, assuming that you collect necessary customer events. Using SQL and your own DB, you have infinite flexibility in defining, tweaking and improving your retention analysis to drive growth.

Rolling retention is defined as the percentage of returning users measure at a regular interval, typical weekly or monthly, grouped by their sign-up week/month, also known as cohort. By grouping users based on when they signed up, you can gain insight on how your product/marketing/sales initiatives have impacted retention: For example, suppose you had a major launch and had many sign-ups over the following few days. How well did these new users stick around compared to, say, pre-launch users that signed up a week prior? Or, say you sent out offers for a discount to dormant users from 2 months ago: How many of them came back and stayed on the product? Rolling retention helps you answer these questions at a glance.

1. Group the bucket into period e.g. daily, weekly or monthly. Will use month in example.
2. Normalizing bucket periods – use FIRST\_VALUE in a windowed function to get a new column of the first month.
3. Minus the relative date grouping from the first value to get your week number.
4. SUM case whens.   
   E.g. SUM(CASE WHEN week\_number = 0 THEN 1 ELSE 0 END) AS week\_0  
    SUM(CASE WHEN week\_number = 1 THEN 1 ELSE 0 END) AS week\_1



This gives you the raw user counts for each cohort week over week. If you want to convert the columns into percentage, then divide week\_1 through week\_9 with week\_0.

Another method which isn’t exact that I have used is getting the MAX purchase date of each user and then doing a datediff from date opened. Issue is you could deposit in the 3rd month and then the 6th and it take the 6th as your retention month.

## Dates & MTD Options

First day of previous month: SELECT DATEADD(mm, DATEDIFF(mm, 0, GETDATE()) - 1, 0)

Last day of previous month: SELECT DATEADD(DAY, -(DAY(GETDATE())), GETDATE())

First day of current month: SELECT DATEADD(mm, DATEDIFF(mm, 0, GETDATE()), 0)

Last day of current month: SELECT DATEADD (dd, -1, DATEADD(mm, DATEDIFF(mm, 0, GETDATE()) + 1, 0))

SELECT DATEADD(month, DATEDIFF(month, 0, GETUTCDATE() ), 0) AS StartOfMonth

DECLARE @DatePrevFrom DATE = util.dbo.fn\_FirstOfMonth ( DATEADD ( MM, -1, GETUTCDATE() ) )

DECLARE @DatePrevTo DATE = DATEADD ( MM, -1, GETUTCDATE())

DECLARE @DateFrom DATE = util.dbo.fn\_FirstOfMonth ( GETUTCDATE() )

DECLARE @DateTo DATE = GETUTCDATE()  
  
In 2012 and later you can now use EOMONTH

DATEPART ( interval, @Date )  
-- yyyy, mm, dd  
-- dw (weekday)

Use datepart to find variables of how many specific days they purchased on the account.

Date Formatting:  
CONVERT(VARCHAR(10), r.Date, 103) + ' ' + convert(VARCHAR(8), r.Date, 14)

Find how many days there are this month (useful when projections need to be done):  
DECLARE @DaysThisMonth INT = (

SELECT DATEPART ( DAY, DATEADD (dd, -1, DATEADD(mm, DATEDIFF(mm, 0, GETDATE()) + 1, 0)) )

)

**MTD**

Better way to write this:

DECLARE @DateFrom DATETIME = '20171201'  
DECLARE @Day INT = ( SELECT DATEPART ( DAY, GETUTCDATE() ) )  
WHERE dd.Date > @DateFrom  
 AND DAY ( dd.Date ) < @Day

**Dynamic Date Variables (depending on GMT time)**

DECLARE @DateFrom DATETIME  
DECLARE @DateTo DATETIME

IF DATEPART ( HOUR, GETUTCDATE() ) < 12  
 BEGIN  
 SET @DateFrom = CAST ( DATEADD ( dd, -16, GETUTCDATE() ) as DATE )  
 SET @DateTo = CAST ( DATEADD ( dd, -1, GETUTCDATE() ) as DATE )  
 END

IF DATEPART ( HOUR, GETUTCDATE() ) >= 12  
 BEGIN  
 SET @DateFrom = CAST ( DATEADD ( dd, -15, GETUTCDATE() ) as DATE )  
 SET @DateTo = CAST ( GETUTCDATE() as DATE )  
 END

To round to nearest 10 you use ROUND ( …, -1 ); e.g. round ( datepart(minute, ...) , - 1)

**Date yyyy-mm-dd (ISO8601)**

SELECT CONVERT(char(10), GetDate(),126)  
ONVERT(CHAR(23),CONVERT(DATETIME,sp.prtime,101),121) # with time  
OR  
SELECT FORMAT(GetDate(), 'yyyy-MM-dd') (SQL 2012)

## Variables

Use variables to automate cleaning up and then NEWID() to randomly select the rows. Then used this for a DELETE statement.

DECLARE @RecordCleanup INT  
DECLARE @CountCountrol INT = ( SELECT COUNT ( \* ) FROM #Results WHERE Disabled = 0 )  
DECLARE @CountDisabled INT = ( SELECT COUNT ( \* ) FROM #Results WHERE Disabled = 1 )

SELECT @Difference = @CountCountrol - @CountDisabled

SELECT TOP ( @Difference ) \*  
FROM #Results r  
WHERE r.Disabled = 0  
ORDER BY NEWID()

## Random Samples

SELECT TOP 60 PERCENT \*  
FROM #Results r   
ORDER BY NEWID()

## Query Hints

WITH ( FORCESEEK, INDEX = (drag & drop) )

INNER LOOP JOIN / INNER HASH JOIN / INNER MERGE JOIN

## While Loop

Give all the rows of temp table a row\_number then proceed with the following:

DECLARE @Counter INT = ( SELECT COUNT (\*) FROM #TempTable )

WHILE @Counter > 0

BEGIN

SELECT @User = pd.User,  
 @Detail1 = pd.Detail1,  
 @Detail2 = pd.Detail2  
FROM #PlayerDetails pd  
WHERE pd.ID = @Counter

SELECT DISTINCT  
 @GamingServerID,  
 @UserID,  
 ….  
 ‘TypeofMatch  
FROM Table2  
WHERE Table2.Detail1 = @Detail1  
 AND Table2.Detail2 = @Detail2  
 AND Table2.User <> @UserID

SET @Counter = @Counter - 1

END

---

DECLARE @Counter INT = ( SELECT COUNT (\*) FROM #Temptablewithrownumber )  
WHILE @Counter > 0  
BEGIN  
 SELECT @Company = s.CompanyID  
 FROM #Temptablewithrownumber s  
 WHERE @Counter = s.OrderOfCompletion

SET @Counter = @Counter - 1

END

---

Attrition Loop

DECLARE @Counter INT = 15 -- No. of days to run attrition figures.  
DECLARE @DayAdd INT = 0 -- Use this for individual day attrition  
--DECLARE @DayAdd INT = 1 -- Starts at day 0

WHILE @Counter > 0

BEGIN   
 -- Individual Day Attrition

INSERT INTO #PurchaseAttrition  
 (  
 …  
 )

SELECT r.UserID,  
 r. StatusID,  
 @DayAdd,  
 COALESCE ( SUM ( p.Amount ), 0 ) as SumPurchases,  
 COALESCE ( COUNT ( p.Amount ), 0 ) as CountPurchases  
 FROM #Results r  
 LEFT JOIN Purchases p  
 ON ...  
 AND p.prtime >= DATEADD ( dd, @DayAdd, r.DateOpened ) -- @DayAdd date opened  
 AND p.prtime < DATEADD ( dd, 1, DATEADD ( dd, @DayAdd, r.DateOpened ) ) -- Add 1 day to @DayAdd date opened  
 GROUP BY r.UserID,  
 r. StatusID

SET @DayAdd = @DayAdd + 1  
 SET @Counter = @Counter - 1

END

## CTE

;  
WITH Sales\_CTE (SalesPersonID, SalesOrderID, SalesYear)   
AS   
(   
 SELECT SalesPersonID, SalesOrderID, YEAR(OrderDate) AS SalesYear   
 FROM Sales.SalesOrderHeader   
 WHERE SalesPersonID IS NOT NULL   
)   
, New\_CTE ( ... )  
AS  
(  
)  
UPDATE #  
SET …  
FROM #  
JOIN New\_CTE

## SQL Server System

Little tip to get all column names:   
SELECT name + ','   
FROM tempdb.sys.columns   
WHERE object\_id = object\_id('tempdb..#Results')

SELECT COLUMN\_NAME + ','  
FROM ScratchPad.INFORMATION\_SCHEMA.COLUMNS  
WHERE TABLE\_NAME = N'TableName'

Check SP dependencies on tables:  
SELECT COLUMN\_NAME + ','  
FROM sys.procedures  
WHERE OBJECT\_DEFINITION(OBJECT\_ID) LIKE '%TableName%'  
ORDER BY name

Find tables or column names:  
USE …  
SELECT TOP 1000  
 sc.name,  
 so.name,  
 so.xtype  
FROM SYSCOLUMNS sc  
JOIN SYSOBJECTS so   
 ON so.id = sc.id   
WHERE   
 sc.name LIKE '%managementcolorid%'  
 --so.name LIKE '%fx%'  
 AND so.xtype = 'U'  
ORDER by sc.name,  
 so.name

Look at constraints & referencing tables:

SELECT c.CONSTRAINT\_NAME,  
 cu.TABLE\_NAME AS ReferencingTable, cu.COLUMN\_NAME AS ReferencingColumn,  
 ku.TABLE\_NAME AS ReferencedTable, ku.COLUMN\_NAME AS ReferencedColumn  
FROM INFORMATION\_SCHEMA.REFERENTIAL\_CONSTRAINTS c  
INNER JOIN INFORMATION\_SCHEMA.CONSTRAINT\_COLUMN\_USAGE cu  
 ON c u.CONSTRAINT\_NAME = c.CONSTRAINT\_NAME  
INNER JOIN INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE ku  
 ON ku.CONSTRAINT\_NAME = c.UNIQUE\_CONSTRAINT\_NAME  
WHERE ku.TABLE\_NAME = 'tb\_Session'

## Tips/Things to Remember

Keep in mind when you can’t create indexes – need to re-write the query for the indexes, for example only pulling out certain columns and then joining again OR putting all the data into a temp table first (e.g. when setting a value to 0 or 1). An example was a query using where >= Amount was in the WHERE clause, I needed to move it to HAVING MAX >= Amount and it used the Index Seek.

Need to always think of the grouping you’re using, especially on queries which are going to be run for longer periods. E.g. if you’re deleting based on MID then the results will be inconsistent over time.

Also always need to think of the date formats in diff tables, e.g. of some are grouped by day you need to cast it as date.

5 cards in timeframe:  
Players having MAX card rego in last day, then count cards > 5 & DATEDIFF(HOUR, MIN(c.USCCDATEADDED), MAX(c.USCCDATEADDED)) Interval.

CASE WHEN SUMS  
SUM ( CASE WHEN Status = 1 THEN Amount END )

STDEV ( ) for standard deviation to see how much values vary

Permissions: GRANT SELECT ON … TO …

Convert negatives to positives with ABS()

TRUNCATE TABLE – Similar to delete, can be used to clear out a table but it’s faster as it doesn’t log the deletion of each row.

Custom order by:  
ORDER BY CASE Cluster   
 WHEN 'High' THEN 1  
 WHEN 'Mid-Range Low-Freq' THEN 2  
 WHEN 'Mid-Range High-Freq' THEN 3  
 WHEN 'Low' THEN 4  
 END

Need to always think of grouping levels, e.g. if a record could appear multiple times need to think of how you’ll group these. E.g. weekly stats have same masterIDs but grouping by the financial month (first day of month). And often looking at numbers till the end of that month.

Can call a stored procedure with a temp table:  
- Within the stored proc say if object\_id IS NULL CREATE TABLE  
- Can then execute that stored procedure already having created the input temp table, only need to the parameters and insert into a proc results temp table.

Compare adjacent records for each MasterID / Date and set IsConsecutive(n) = 1 if TxHour(n+1) - TxHour(n) = 1

## Detecting Outliers

**Standard Deviation Thresholds (Z-Scores)**

This is a bit subjective, but you can identify the rows whose values are furthest from the average. I would do this by calculating the z-score and looking at the largest/smallest z-scores.

The z-score is the value minus the average divided by the standard deviation. Here is an example of the calculation:

# 1

select t.\*,  
 (price - avg\_price) / nullif(std\_price, 0) as z\_price  
from t join  
 (select product, avg(price) as avg\_price, stdev(price) as std\_price  
 from t  
 group by product  
 ) tt  
 on t.product = tt.product  
order by abs(z\_price) desc;

# 2 two-tailed 5% threshold

with data as (  
 select  
 date\_trunc('day', created\_at)::date as day,  
 count(1) as value  
 from disk\_usage  
 group by 1  
), data\_with\_stddev as (  
 select  
 day,  
 value,  
 (value - avg(value) over ())  
 / (stddev(value) over ()) as zscore  
 from data  
 order by 1  
)  
select \* from data\_with\_stddev where abs(stddev) >= 1.645

**Percentage Thresholds**

With percentage thresholds, our alerts will continually adjust to recent trends. E.g. we can set an alert at 2x the current average.

To get the percentage difference vs. the mean for each data point, we use a window function to divide each row’s value by the average value for the entire table above.

with user\_count as (  
 select  
 DATEPART(day, created\_at) as day,  
 count(1) as value  
 from users  
 group by 1  
),   
user\_count\_with\_pct as (  
 select  
 day,  
 value,  
 value / (avg(value) over ()) as pct\_of\_mean   
 from user\_count  
 order by 1  
)  
select \* from user\_count\_with\_pct where pct >= 2.0

## Good Sources

Further learning - <https://www.khanacademy.org/computing/computer-programming/sql/further-learning-in-sql/a/further-learning-in-sql-what-to-learn-next>

Indexes and DB structure - <https://www.simple-talk.com/sql/database-administration/brads-sure-guide-to-indexes/>

Query tuning - <https://www.simple-talk.com/sql/performance/simple-query-tuning-with-statistics-io-and-execution-plans/>

Estimated vs Actual Rows - <https://www.brentozar.com/archive/2013/08/query-plans-what-happens-when-row-estimates-get-high/>

Rolling Retention - <https://blog.treasuredata.com/blog/2016/07/22/rolling-retention-done-right-in-sql/> & <https://www.simple-talk.com/sql/t-sql-programming/calculating-values-within-a-rolling-window-in-transact-sql/>

Retention & reengagement - <http://blog.forcerank.it/sql-for-calculating-churn-retention-reengagement>

Outlier Detection - <https://www.periscopedata.com/blog/outlier-detection-in-sql>

# **GA/BigQuery**

## GA

Audience - User Explorer - Add Segment

Behaviour - Site Content - All Pages

## Basics

Arrays are similar to lists.

An "Array" is one consecutive memory area that contains one element after the other in some "packed" way (without holes).

A list may also hold a number of elements but the implementation detail on how this is done internally are basically unknow and typically irrelevant. You can add elements to a list, remove elements from the list, find elements in the list or iterate over all the elements in the list.

Typically a list does organize it's elements indirectly - so lists do NOT contain the elements in place (as in an array) but lists contain references to those elements and the elements are stored somewhere else. That enables a list to contain one and the same element multiple times, which is impossible in an array (unless you store references in your array - which you could)

In Legacy SQL:  
Repeated = ARRAY<T>  
Record = STRUCT<T>

ARRAYs of ARRAYs are not allowed. Queries that would produce an ARRAY of ARRAYs will return an error. Instead a STRUCT must be inserted between the ARRAYs using the SELECT AS STRUCT construct.

Can use: SAFE. before a function to make sure it won't give an error. E.g. SELECT SAFE.SUBSTR('foo', 0, -2)

A SELECT \* REPLACE statement does not change the names or order of columns. However, it can change the value and the value type.

## Queries

**My Query**

#standardSQL

SELECT   
 fullVisitorId,  
 userId,  
 clientId,  
 date,  
 EXTRACT(DATE FROM TIMESTAMP\_MILLIS ( visitStartTime ) ),  
 TIMESTAMP\_MILLIS ( visitStartTime ),  
 hits.page.pagePath,  
 hits.hitNumber,  
 hits.time,  
 dim.index,  
 dim.value  
FROM  
 `j-first-project-57563.131442491.ga\_sessions\_20180627` t  
LEFT JOIN  
 UNNEST ( t.hits ) as hits  
LEFT JOIN  
 UNNEST ( hits.CustomDimensions) as dim  
LIMIT 1000

**-- Part 2:** Just get all players who hit disconnect page path and then get their epaid from any other page path.

#standardSQL

WITH disc\_tbl AS  
 (  
 SELECT   
 fullVisitorId,  
 userId,  
 clientId,  
 hits.page.pagePath,  
 MIN ( TIMESTAMP\_SECONDS ( visitStartTime ) ) as SessionTime  
 FROM  
 `j-first-project-57563.131442491.ga\_sessions\_20180627` t  
 LEFT JOIN  
 UNNEST ( t.hits ) as hits  
 WHERE  
 date = '20180627'  
 AND hits.page.pagePath = '/TrackedEvent/Lobby/PlayerSessionDisconnected'  
 GROUP BY  
 fullVisitorId,  
 userId,  
 clientId,  
 hits.page.pagePath  
 )

SELECT   
 \*  
FROM  
 (  
 SELECT   
 dis.fullVisitorId,  
 dis.userID,  
 dis.clientId,  
 dis.pagePath,  
 dis.SessionTime,  
 hits.page.pagePath as PagePath2,  
 dim.index,  
 dim.value,  
 ROW\_NUMBER () OVER ( PARTITION BY dis.fullVisitorId ) as RowNumber  
 FROM  
 disc\_tbl dis  
 JOIN  
 `j-first-project-57563.131442491.ga\_sessions\_20180627` ses  
 ON dis.fullVisitorId = ses.fullVisitorId  
 LEFT JOIN  
 UNNEST ( ses.hits ) as hits  
 LEFT JOIN  
 UNNEST ( hits.CustomDimensions) as dim  
 WHERE  
 dim.index = 83  
 ) sisq  
WHERE  
 sisq.RowNumber = 1

1) Unnest within SELECT:

SELECT visitId, ( SELECT COUNT( hitNumber ) FROM UNNEST( hits ) ) AS view\_count  
FROM `google.com:analytics-bigquery.LondonCycleHelmet.ga\_sessions\_20130910`

2) Unnest within JOINs:

SELECT visitId, COUNT( hitNumber ) AS view\_count  
FROM `google.com:analytics-bigquery.LondonCycleHelmet.ga\_sessions\_20130910`  
LEFT JOIN UNNEST( hits )   
GROUP BY visitId

## Connecting R

<https://www.blendo.co/blog/access-data-google-bigquery-python-r/>

<https://github.com/r-dbi/bigrquery>

## Resources

<https://bigquery.cloud.google.com>

<https://analytics.google.com/analytics/web/>

<https://cloud.google.com/bigquery/docs/reference/standard-sql/enabling-standard-sql>

<https://cloud.google.com/bigquery/docs/reference/standard-sql/migrating-from-legacy-sql#differences_in_repeated_field_handling>

Unnest in select

<https://robertsahlin.com/flatten-google-analytics-custom-dimensions-with-a-bigquery-udf/>

<https://www.periscopix.co.uk/blog/bigquery-example-queries/>

<https://cloud.google.com/dataprep/docs/html/EXAMPLE---Flatten-and-Unnest-Transforms_57344993>

<https://stackoverflow.com/questions/47039541/bigquery-sql-is-it-better-to-unnest-in-select-or-join>

<https://chartio.com/resources/tutorials/how-to-flatten-data-using-google-bigquerys-legacy-vs-standard-sql/>

<https://cloud.google.com/bigquery/data-types>

<https://www.quora.com/What-is-the-difference-between-an-array-a-list-and-a-linked-list>

<https://www.recommendedagencies.com/periscopix/news/892677/bigquery-example-queries>

# **SSRS/Visual Studio**

## Basics

N3 format – decimal with three numbers, e.g. ROP.

Custom date text box properties:  
dd/MM/yyy

Dealing with dates:   
=Today.AddDays(-7)

Can use regex or isnumeric in the proc to distinguish between input variables. E.g. distinguishing between MasterID And AccountNo.

To make the report scrollable, set height to 0.

Can specify a maximum value for a parameter within the SQL proc e.g. BEGIN RETURN END, then set the Tablix “No Rows” message to indicate such to the user.

Use SWITCH instead of Nested Ifs.

CHAR(10) - SSRS adds new line

Freeze headers:  
Right click the row in the Tablix – Tablix Properties – Keep header visible while scrolling  
-  
If it is a matrix (has groups), go to the grouping section and right click the arrow for Advanced Mode. Then on the top row group change FixedData to True.

Tablix Properties - Keep together on one page if possible.

## Expressions & Formulas

Expression for ROP colour by Alex:  
=IIF(IsNothing(Fields!BenchmarkROP.Value), "No Color", IIF(Sum(Fields!Cost.Value) > 500, IIF(iif(Sum(Fields!Cost.Value) = 0, 0, Sum(Fields!USDPurchasesDay0.Value) / iif(Sum(Fields!Cost.Value) = 0, 1, Sum(Fields!Cost.Value))) > Fields!BenchmarkROP.Value, "LightGreen","Pink"), "No Color"))

My most up-to-date fill expression:  
=IIF(IsNothing(Fields!Day0BenchmarkROP.Value), "No Color",   
IIF(Fields!PixelCost.Value > 500,   
IIF(Fields!Day0ROP.Value >= Fields!Day0BenchmarkROP.Value, "LightGreen","Pink")  
,Nothing))

Background colour based on another field:  
=IIF(IsNothing(Fields!Day0BenchmarkROP.Value), "No Color", IIF(Fields!Day0ROP.Value >= Fields!Day0BenchmarkROP.Value,"LightGreen","Pink"))

Text box properties – expression for font colour:  
=IIF(Fields!ScoreRating.Value = "URGENT", "White",  
IIF(Fields!ScoreRating.Value = "High", "Red","Black"))

Expression for fill:  
=IIF(Fields!ScoreRating.Value = "URGENT", "Red", Nothing)

Base fill of another field containing colours. Text box properties – fill fx = field

Can put action on a text box to open new report with parameters, just have to specify path.

Divide by Zero:  
=IIf(ReportItems!Textbox85.Value = 0, 0, ReportItems!Textbox85.Value / IIf(ReportItems!Textbox84.Value = 0, 1, ReportItems!Textbox84.Value))  
---  
Count under 30%  
=IIf(sum(Fields!CountUnder30.Value) = 0, 0, sum(Fields!CountUnder30.Value) / IIf(sum(Fields!Reg.Value) = 0, 1, sum(Fields!Reg.Value)))  
Then you can copy and paste these up to the other grouped rows with sum:  
=IIf(sum(Fields!CountUnder30.Value) = 0, 0, sum(Fields!CountUnder30.Value) / IIf(sum(Fields!Reg.Value) = 0, 1, sum(Fields!Reg.Value)))

Total grouped aggregations:  
sum(a)/sum(b)

Can create grouping and then do custom sort based:  
=iif(Fields!MonitorGroup.Value = "tes1","1",  
iif(Fields!MonitorGroup.Value = "test2", "2",  
iif(Fields!MonitorGroup.Value = "test3", "3",  
iif(Fields!MonitorGroup.Value = "test4","4",  
iif(Fields!MonitorGroup.Value = "test5","5",""  
)))))

<https://docs.microsoft.com/en-us/sql/reporting-services/lesson-6-adding-grouping-and-totals-reporting-services>

# **R**

## Updating R

if(!require(installr)) {

install.packages("installr"); require(installr)} #load / install+load installr

updateR()

## Importing Data

What you need:

1. Raw data
2. Tidy data set
3. Code book describing each variable and its values in tidy data set (e.g. units)
4. Exact steps explaining how you went from raw to processed

setwd("C:\\Users\\rtutt\\Desktop\\R")

Downloading files from the internet:  
fileUrl <- “http…”  
download.file(fileUrl, destfile = “./data/cameras.csv”)

read.table(“”, sep = “,”, header = TRUE)  
nrows – how many rows to read of file  
quote = “” means no quotes

CSV  
Data <- read.csv("Section6-Homework-Data.csv", na.strings=c(“”),header = TRUE)  
Import CSV <- read.csv(file.choose(),na.strings= c("NA", "#DIV/0!", ""), header = TRUE)

Excel  
library(xlsx)  
colIndex <- 2:3  
rowIndex <- 1:4  
read.xlsx(“”, sheetIndex = 1, header = TRUE, colIndex = colIndex, rowIndex = rowIndex)  
write.xlsx()

XML  
library(XML)  
fileUrl <- “http:….”  
doc <- xmlTreeParse(fileUrl, useInternal = TRUE)  
rootNode <- xmlRoot(doc)  
xmlName(rootNode)

JSON  
library(jsonlite)  
jsonData <- fromJSON(https://api....”)  
names(jsonData)  
toJSON(data, pretty = TRUE)

## Initial Data Preparation

View()

Check out the data frame ( nrow, ncol, str, head, tail, summary )

Assign colnames / rownames

Assign factors & levels to categorical variables ( movies$Year <- factor(movies$Year ) OR factor(dataset$Purchased, levels = c(“0”,”1”),labels = c(“NotPurchased”,”Purchased”))

Gsub(“pattern”, “replacement”, dataset, ignore.case = TRUE/FALSE) to remove characters. An example using it with numeric commas: as.numeric(gsub(",","",value$Count))

Change colnames:  
names(comparedf) <- gsub(x = names(comparedf), pattern = '.x', replacement = 'ASX')  
names(comparedf) <- gsub(x = names(comparedf), pattern = '.y', replacement = 'TLS')

Remove factors by converting to character then numeric - as.numeric(as.character(…))

Locate missing data - data[!complete.cases(data),]

Can also plot missing data -  
library(Amelia)  
missmap(data.raw,legend = TRUE,y.cex = 0.1, x.cex = 0.5)

Replace missing data by removing, replacing with factual data or function to replace with mean.

Can subset, filter & add calculated columns using [] and $.

Can join two DFs together ( <- merge(stats, mydf, by.x = "Country.Code", by.y = "Code") )

Take random sample of 100 rows: dataset [sample(nrow(dataset), 100),]

TrainingDataset <- TrainingDataset[,colSums(is.na(TrainingDataset)) == 0]

Referenced

<https://stackoverflow.com/questions/12454487/remove-columns-from-dataframe-where-some-of-values-are-na>

nzv <- nearZeroVar(trainDF, saveMetrics = T)   
trainDFclean <- trainDF[, !nzv$nzv]

Use gc() to clear now unused memory, or, better only create the object you need in one session.

rm()

memory.limit()

# Rounding numbers to decimal places:

formatC(x, digits = 8, format = "f")

subset(dataset1, !(column1 %in% dataset2$column))  
  
Tidying Data:   
dataset\_final <-  
 dataset %>%  
 select\_if(is.numeric) %>%  
 select(-one\_of(names(dataset\_final))) %>%  
 cbind(dataset\_final)

## Dealing with Missing Data

1. Predict with 100% accuracy (e.g. city is missing but state is NSW)  
2. Leave record as is.  
3. Remove record entirely (if the missing field is important to analysis and can’t research value).  
4. Replace with median or mean (median better if there’s big outliers, can’t do this for categorical like Year).  
5. Fill in by exploring correlations and similarities  
6. Introduce dummy variable for “missing”

**NA Replacement**

df[is.na(df)] <- 0

mutate\_all(funs(replace(., is.na(.), 0))))

**Replace missing values:**

df[complete.cases(df),]

na.omit(df)

Dealing with NAs:  
- use which operator around your filter for non missing data: fin[which(fin$Revenue == 9746272),]  
- use is.na function to find missing data. fin[is.na(fin$Expenses),]

Remove rows that have NA value in column: fin <- fin[!is.na(fin$Industry),]

Resetting the data frame index: rownames(fin) <- 1:nrow(fin) OR rownames(fin) < NULL

Replace missing data with factual analysis: fin[is.na(fin$State) & fin$City == "New York","State"] <- "NY" (found missing state rows, then narrowed down and replace the state).

Replace missing data with average:   
dataset$Age = ifelse(is.na(dataset$Age),  
 ave(dataset$Age, FUN = function(x) mean(x, na.rm = TRUE)),  
 dataset$Age)

Replace missing data with median from their class:  
med\_empl\_retail <- median(fin[fin$Industry == 'Retail', "Employees"], na.rm = TRUE)  
fin[is.na(fin$Employees) & fin$Industry == 'Retail', 'Employees'] <- med\_empl\_retail

Replace missing data with derived values:  
fin[is.na(fin$Profit), 'Profit'] <- fin[is.na(fin$Profit), 'Revenue'] - fin[is.na(fin$Profit), 'Expenses']

bioconductor package - impute.knn

## Factors

We can convert a number of categorical variables to factors at once:  
dataset[,c(1:10)] <- lapply(select(dataset, default.payment.next.month:PAY\_6), function(var) as.factor(var))

Levels orders the factors and then we can lable them based on this new order.

customers$Cluster <- factor(customers$Cluster,  
 levels = c(4,1,3,2),  
 labels = c("Low","Medium","High","Top"))

Assign factors & levels to categorical variables ( movies$Year <- factor(movies$Year ) OR factor(dataset$Purchased, levels = c(“0”,”1”),labels = c(“NotPurchased”,”Purchased”))

Remove factors by converting to character then numeric - as.numeric(as.character(…))

Change the order of categories by changing the actual factor variable, e.g. factor(levels, labels)

Drop levels of factors after filtering and re-apply them:  
droplevels(df)  
levels(df$currency)

**Map Function**

dataset\_factors <-  
 dataset %>%  
 select\_if(is.factor) %>%  
 map\_df(~fct\_lump((.), n = 9))

dataset <-  
 dataset %>%  
 select\_if(negate(is.factor)) %>%  
 cbind(dataset\_factors)

**Lump factors into 6 categories**

library(forcats)

mutate(package = as.factor(package) %>% fct\_lump(n = 5, other\_level = "Other"))

dataset %>%   
 mutate(OperatingSystem = fct\_lump(OperatingSystem, n = 20, other\_level = "Other")) %>%   
 toprankcompare(OperatingSystem)

**Other Forcats Functions**

mutate(fct\_rev())

fct\_reorder & fct\_relevel

**Tidying Data with Factors Example**

dataset\_tidy <-  
 dataset %>%  
 gather(key = Country, value = Percent, -Month) %>%  
 mutate(Percent = 100 \* Percent,  
 Month = as.Date(as.character(Month)),  
 Country = as.factor(Country) %>% fct\_reorder(Percent) %>% fct\_rev())

**Re-ordering factors by values**

fn = factor(f, levels=unique(f[order(a,b,f)]), ordered=TRUE)

Can re-order at dplyr level:

order <-   
 countrytopsummary %>%   
 group\_by(Country) %>%   
 summarise(TotalPixels = sum(Pixels)) %>%   
 arrange(desc(TotalPixels)) %>%   
 mutate(Country = factor(Country, unique(Country)))

Dplyr level with ggplot2

dataset %>%   
 group\_by(Province) %>%   
 summarise(TotalPlayers = n(),  
 TopPlayers = sum(PurchaseRank == 'Top')) %>%   
 mutate(TotalPercentage = 100 \* TotalPlayers / sum(TotalPlayers),  
 TopConversion = 100 \* TopPlayers/TotalPlayers) %>%   
 arrange(desc(TopConversion)) %>%   
 mutate(Province = factor(Province, unique(Province))) %>%   
 ggplot(aes(x = Province, y = TopConversion)) +  
 geom\_bar(stat = "identity")

My Example (used both above):

Create another dplyr table (above) factor ordered by total value, then re-ordered other factors by this:  
order <- levels(order$Country)  
countrytopsummary$Country <- factor(countrytopsummary$Country,  
 levels = order)

Another Alternative (if data already grouped at right level)

cty\_mpg <- cty\_mpg[order(cty\_mpg$mileage), ] # sort  
cty\_mpg$make <- factor(cty\_mpg$make, levels = cty\_mpg$make) # to retain the order in plot.

Can also re-order using reorder function if values are the same:

p2 <- ggplot(df, aes(x = reorder(Category, -Count), y = Count)) +  
 geom\_bar(stat = "identity")

cc.df$origin <- reorder(cc.df$origin, cc.df$count)

cc.df$origin <- reorder(cc.df$origin, -cc.df$count)

Another Example

topbrowsers <-  
 dataset %>%  
 group\_by(Browser) %>%  
 summarise(Pixels = sum(Pixels)) %>%  
 arrange(desc(Pixels)) %>%  
 top\_n(10) %>%  
 droplevels()

topbrowsers <-  
 topbrowsers %>%  
 arrange(desc(Pixels)) %>%  
 mutate(Browser = factor(Browser, unique(Browser)))

order <- levels(topbrowsers$Browser)

dataset %>%  
 mutate(Browser = factor(Browser, levels = order))

<https://stackoverflow.com/questions/10758243/r-order-a-factor-based-on-value-in-one-or-more-other-columns>

## Formatting & Scales

Rounding numbers to decimal places:

formatC(x, digits = 8, format = "f")

Library(scales)

dollar(), percent()

mutate\_all(funs(prettyNum(., big.mark=",")))

mutate\_each(funs(as.character(scales::dollar(.))))

library(scales)  
mtcars %>%   
 mutate\_at(names(.), funs(dollar(.)))

Sometimes need to separate the mutating rather than doing all at once:  
mutate(… = formatC(…, digits = 2, format = "f"),  
 … = prettyNum(…, big.mark=","),  
 … = prettyNum(…, big.mark=","))

mutate(TotalUpsellBonus = dollar(TotalUpsellBonus))

mutate\_if(is.numeric, funs(round(., 2))) %>%   
mutate\_if(is.numeric, funs(prettyNum(., big.mark=",")))

Library(scales)  
scale\_y\_continuous(labels=comma)

<https://cran.r-project.org/web/packages/scales/scales.pdf>

## Manipulating Data

Can subset a data frame by the values in another:  
subset(dataset, Country %in% countrytop15$Country)

### dplyr

dplyr::left\_join()

Good practice to ungroup after grouping.

cran <- tbl\_df(mydf)

training <- rename(training, 'New' = 'Old')

top\_n(5, column)

mutate(NewMid = gsub(1, "New Mid", NewMid))

mutate(StDevPurchases = as.numeric(gsub('NULL', 0, Score))) # Pattern, replacement, column

Count - replaces group\_by() %>% summarise(n())

Contains(), starts\_with(), ends\_with()  
E.g. train\_raw\_tbl %>%  
 select(Attrition, contains("employee"), contains("department"), contains("job"))

Select column names by another data frame: select(-one\_of(names(dataset\_final)))

Relative Frequency Proportions  
dataset %>%   
 group\_by(Attrition) %>%   
 summarise(Number = n()) %>%   
 mutate(Freq = Number / sum(Number) )  
  
-- This one is good for visualizaing relative frequencies, e.g. with bar charts

training %>%   
 group\_by(JobRole, Attrition) %>%   
 summarise(Number = n()) %>%   
 mutate(Freq = Number / sum(Number)) %>%   
 filter(Attrition == 'Yes') %>%   
 arrange(desc(Freq))

Cumulative sum to flag specific high usage cases

mutate(  
 pct\_cum = cumsum(pct),  
 high\_usage = case\_when(  
 pct\_cum <= 0.8 ~ "Yes",  
 TRUE ~ "No"  
 ))

Turn 0’s into NA and drop

dataset <- dataset %>%   
 mutate(Quantity = replace(Quantity, Quantity <= 0, NA),  
 UnitPrice = replace(UnitPrice, UnitPrice <= 0, NA))

dataset <- dataset %>%   
 drop\_na()

Replace NULLs with 0  
dataset <-  
 dataset %>%  
 mutate(PurchStDevRatio = as.numeric(gsub('NULL', 0, PurchStDevRatio)))

Replacing Values  
mutate(Country = replace(Country, Country == "EIRE","Republic of Ireland"))

Conditional Count  
CountPurchasingUsers = sum(SumPurchases > 0)

Conditional Sum  
TopPurchases = sum(SumPurchasesUSD[which(PurchaseRank == 'Top')])

Rename within chaining  
 %>%  
rename('Upsell Bonus Uptake' = 'UpsellBonusUptake')

Rename by index  
select/rename(NewName = 1)

Select only columns of certain data type  
select\_if(dataset, is.factor)  
select\_if(dataset, is.numeric)  
select\_if(negate(is.factor))

quanti <- select\_if(dataset, is.numeric)  
quanti <- cbind.data.frame(quanti, dataset$DEFAULT)

Distinct  
dataset %>% distinct(PromoID)

select - subset of columns  
select(cran, variable1:variable2) # sequence of columns  
select(cran, -(variable1:variable5))

filter - subset of rows  
filter(dataset, column1 == "data", column2 == 'data')  
filter(dataset, column1 == "data" | column2 == "data") # OR  
filter(dataset, !is.na(column))

filter - use %in% instead of ==

You can use multiple categories e.g. c("cat1","cat2")

Filter dataframe by another dataframe  
dataset %>%   
 filter(!(UserID %in% outliernetwin$UserID)

Filtering out based on multiple conditions  
Enclose the whole statement in brackets with ! before it:  
dataset\_daily %>% filter(!(FailedLoginDate == '2018-06-12' & Baseline == 'Baseline'))

Filtering by multiple & OR conditions

dataset %>%  
 filter((a == 'Yes' & b== '') |  
 (a == 'Yes' & b== ''))

Filtering with text contains  
df %>%  
 filter(str\_detect(letters, "a|f|o") )

Case When

#1 - %% means contains, the TRUE is kind of like an else  
x <- 1:50  
case\_when(  
 x %% 35 == 0 ~ "fizz buzz",  
 x %% 5 == 0 ~ "fizz",  
 x %% 7 == 0 ~ "buzz",  
 TRUE ~ as.character(x)  
)

#2 -   
starwars %>%  
 select(name:mass, gender, species) %>%  
 mutate(  
 type = case\_when(  
 height > 200 | mass > 200 ~ "large",  
 species == "Droid" ~ "robot",  
 TRUE ~ "other"  
 )  
 )

#3 -  
mutate(  
 MinsUntilLogin = case\_when(  
 MinsDifference <= 1 ~ "0-1",  
 MinsDifference >= 2 & MinsDifference <= 19 ~ "2-19",  
 TRUE ~ "20+")  
 )

Percent Rank  
percent\_rank(x)  
OR  
mutate(percrank=rank(value)/length(value))

arrange - order rows according to variable  
arrange(dataset, desc(column), column2)

mutate - new variable based on the value of one or more variables already in a dataset.  
mutate(dataset, newcolumn = size / 2^20, newcolumn2 = size\_mb / 2^10)

summarize() - collapses the dataset to a single row and can be grouped per value of a specific variable.  
summarize(dataset, avg\_bytes = mean(size))  
groupeddata <- group\_by(dataset, variable1)  
summarize(groupeddata, mean(column))  
n() – count  
n\_distinct() - count distinct  
pack\_sum <- summarize(groupeddata,  
                      count = n(),  
                      unique = n\_distinct(column),  
                      avg\_bytes = mean(size))

quantile(pack\_sum$count, probs = 0.99) #top 1% quantile  
top\_counts <- filter(pack\_sum, count > 679)  
top\_counts\_sorted <- arrange(top\_counts, desc(count))  
View(top\_counts\_sorted)

Chaining/piping (Ctrl+shift+m)

result3 <-  
  cran %>%  
  group\_by(package) %>%  
  summarize(count = n(),  
            unique = n\_distinct(ip\_id),  
            countries = n\_distinct(country),  
            avg\_bytes = mean(size)  
  ) %>%  
  filter(countries > 60) %>%  
  arrange(desc(countries), avg\_bytes)

-- Another example of my query looking at top 5 by increase using dplyr and tidyr:

increase <- dataset %>%   
 filter(Change == 'Increase') %>%   
 arrange(desc(Ratio)) %>%   
 top\_n(5, Ratio)

increasegather <- gather(select(increase,Country,'10', '11'),Month, Count, '10', '11', -Country)

Formatting: mutate\_all(funs(prettyNum(., big.mark=",")))

### tidyr

**gather (wide to long format)**

The most important function in tidyr is gather(). It should be used when you have columns that are not variables and you want to collapse them into key-value pairs.

Common to change data from wide to long format, especially for time series - e.g. needs to be in format index, data, time. Can't be index1, index2, index 3 etc.

Often needs to be done for ggplot aes & faceting.

gather(data, key, value, ... or - ...) # new key column name, new value column name and then the ... is the data to make long or not.

--- example I used where all rows were summed in horizontal format

gather(dataset, Week, X.1:X.12)

gather(dataset, Segment, CountPurchasingUsers:LowQualityUsers, -Year, -Month) # new column is called Segment, groups by Year & Month, Creates new categorical variable Segment.

gather(select(dataset, Year, Month, TotalPurchases, TotalCashins, TotalPurchasesNewRegos, TotalCashinsNewRegos), Type, Amount, -Year, -Month)

# Key is the new id, value is the value, 3rd is which columns we’re looking at.

temperature\_tall <-  
 temperature\_wide %>%  
 gather(key = "id\_sensor", value = "temperature", starts\_with("temp")) %>%  
 mutate(id\_sensor = str\_replace(id\_sensor, "temperature\_", "")) %>%  
 print()

# Another Example from Business Science

Tbl with columns – group, cumulative\_data\_fraction, gain, baseline

Gather(key = key, value = value, gain, baseline)

Becomes tbl - – group, cumulative\_data\_fraction, key (gain/baseline), value

**spread (from long to short)**

spread(data, colname to spread, new col name)

Example of changing format from long to wide - e.g. with a table with day 0, 1, 2 going downwards for each status

attritionresults %>%  
 select(Status, PurchaseDay, CountUsers) %>%  
 spread(PurchaseDay, CountUsers)

*Example on Segmentation*

Used this one when I had a description for each player in one column value (Already Active, Dormant etc) to then create sparse columns e.g. AlreadyActive 0 or 1, Dormant 0 or 1.

Start by creating a new column where the value for each line is 1, then use spread and the 1 will only go on the column with a player segment.

dataset\_clean <-  
 dataset %>%  
 mutate(Value = 1) %>%  
 spread(PlayerSegment, Value) %>%  
 mutate\_if(is.numeric, funs(replace(., is.na(.), 0)))

<http://www.cookbook-r.com/Manipulating_data/Converting_data_between_wide_and_long_format/>

**Separate & Unite**

separate(data, col, into, sep = "-") # column, new column name(s) and the character to separate on. Often used to break date-time values into their individual pieces (day, month, year, etc.)

unite(data, col, ..., sep = "\_") opposite of separate

dataset\_new <- separate(dataset, yr\_month, c("year","month"), sep = "\_")

Tidyr - need to specify both columns.   
E.g. separate(col = Type, into = c("Type","Day"), sep = "\_")

## Writing Programatically

### Basic Refresher

Three parts: arguments, body and environment.

Return value is the last executed expression, or the first executed return() statement.

Functions can be treated like usual R objects; they can also be called anonmylously on one line.

My\_fun <- function(arg1, arg2) {  
body  
}

A couple of examples:

ratio <- function(x, y) {  
 x / y  
}

ratio(3, 4)

---

f <- function(x) {  
 return(x + 1)  
 }

When you call a function, a new environment is made for the function to work. The new environment is populated with the argument values, then objects are looked for outside.

So you can name objects within a function, these wouldn't exist outside it.

When writing functions you should never depend on objects not in the argument.

x <- 5  
f <- function(x) {  
 y <- 5  
 x + y  
}  
f(5)  
= 10

Gives you the type of second element from the list x  
typeof(tricky\_list[["x"]][[2]])

**Tip for loops, use seq\_along() instead of 1:ncol()**

for (i in 1:ncol(df)) {  
 print(median(df[[i]]))  
 }

Is the same as:

for (i in seq\_along(df)) {  
 print(median(df[[i]]))  
}

**Store the output from a loop in a vector; you could also use output not output[[i]] this was just used for generalizability**

output <- vector("double", ncol(df))

for (i in seq\_along(df)) {  
 output[[i]] <- median(df[[i]])  
}

output

### Writing Functions

If you copy and pasted twice it's time to write a function.

Generally name functions lower case descriptive verbs with underscores.

Arguments - use x, y, z for vectors; df for data frame, i or j for numeric indices; n is length or num rows; p is number of columns.

Data arguments supply the data to compute on.

Detail arguments control the details of how the computation is done.

rescale01 <- function(x) {  
 rng <- range(x, na.rm = TRUE)  
 (x - rng[1]) / (rng[2] - rng[1])  
}

... allow multiple unnamed arguments.

Write in tidyeval framework (data, ..., col)

Put tidyeval expressions in their own parenthesis to make sure they execute first.

### Functional Programming

Can only use enquo inside functions & quo outside of them.

De-bug functions by setting all the arguments as objects and then stepping down through function.

map(.x, .f, ...)

map(VECTOR\_OR\_LIST\_INPUT, FUNCTION\_TO\_APPLY, OPTIONAL\_OTHER\_STUFF)

map() iterates function across columns of data frame

Basically just loops over a vector “.x”, does something to each element “.f” and returns the results.

“.x” is always a vector.

Map\_dbl(df, mean) # basically the same as sapply

Map() – returns a list; map\_dbl(), map\_lgl(), map\_int(), map\_chr()

You can call an existing function you defined in map or you can define an anonymous function within map on the fly.  
E.g. map(df, rescale01)  
map(df, function(x), sum(is.na(x)))  
Or a shortcut: map(df, ~ sum(is.na(.))) # tilda defines function and the . is a placeholder for the x

Another shortcut is for list or string subsetting, e.g. map\_dbl(list, “objectA”) # extracts all of object a from list. Shortcut for [[

map(cyl, function(df) lm(mpg ~ wt, data = df))  
Same as map(cyl, ~ lm(mpg ~ wt, data = .))

map\_dbl(cyl, ~ mean(.$disp))

models <- mtcars %>%   
 split(mtcars$cyl) %>%  
 map(~ lm(mpg ~ wt, data = .))

Map(df, mean) is the same as:  
col\_mean <- function(df) {  
 output <- numeric(length(df))  
 for (i in seq\_along(df)) {  
 output[[i]] <- mean(df[[i]])  
 }  
 output  
}

# Absolute deviations raised to power

f <- function(x, power) {  
 # Edit the body to return absolute deviations raised to power  
 abs(x - mean(x)) ^ power  
}

safely() will still run your function on all elements from map but show the errors. Will return a list with two elements: result and error.

Map(long\_list, safely(log))

Possibly() always succeeds but you give it a default value to return on error.

Quietly() captures printed output, messages and warnings instead of capturing errors.

Map2() iterates over two arguments.  
e.g. rnorm(5, mean = 1), rnorm(10, mean = 2)  
Becomes map2(list(5, 10), list(1, 2), rnorm)

Pmap() iterates over many arguments.  
E.g. pmap( list (n = list(5, 10),   
 mean = list(1, 2),  
 sd = list(0.1, 0.5)), rnorm)  
Or pmap(list(mu, n, sd), rnorm)

Invoke\_map() iterates over functions and arguments.  
E.g. invoke\_map(list(rnorm, runif, rexp), n = 5)

The above 3 also have a whole family of functions – e.g. map2\_dbl(), pmap\_dbl(), etc.

safe\_readLines <- safely(readLines)  
html <- map(urls, safe\_readLines)  
transpose(html) # turns the list inside out.  
transpose(html)[["result"]]

walk() - call functions for their side effects not return value (e.g. printing output, plotting and saving files).

plots <- cyl %>% map(~ ggplot(., aes(mpg, wt)) + geom\_point())  
paths <- paste0(names(plots), ".pdf")  
walk2(paths, plots, ggsave)

walk(sims, hist) #walk through histograms

breaks\_list <- list(  
 Normal = seq(6, 16, 0.5),  
 Uniform = seq(0, 5, 0.25),  
 Exp = seq(0, 1.5, 0.1)  
)  
walk2(sims, breaks\_list, hist)

Create function for bins for histogram

find\_breaks <- function(x) {  
 rng <- range(x, na.rm = TRUE)  
 seq(rng[1], rng[2], length.out = 30)  
}  
find\_breaks(sims[[1]])

# find good breaks for each of the objects  
nice\_breaks <- map(sims, find\_breaks)

# apply to all and create plots  
walk2(sims, nice\_breaks, hist)

# pwalk is like pmap, you can use multiple arguments.  
nice\_breaks <- map(sims, find\_breaks)  
nice\_titles <- c("Normal(10, 1)", "Uniform(0, 5)", "Exp(5)")  
pwalk(list(x = sims, breaks = nice\_breaks, main = nice\_titles), hist, xlab = "")

walk() functions return the object you passed to them. This means they can easily be used in pipes.

E.g. sims %>%  
 walk(hist) %>%  
 map(summary) # gives a summary for each

Turn df column into vector:  
y <- select(df, UQ(y)) %>% map(as.numeric)

### Robust Functions

Throwing errors - stopifnot()

stop"specify error message”

both\_na <- function(x, y) {  
 if (!length(x) == length(y)) {  
 stop("x and y must have the same length", call. = FALSE)  
 }   
 sum(is.na(x) & is.na(y))  
}

safe\_readLines <- safely(readLines)  
html <- map(urls, safe\_readLines)

# Initialize some objects  
safe\_readLines <- safely(readLines)  
html <- map(urls, safe\_readLines)  
res <- transpose(html)[["result"]]  
errs <- transpose(html)[["error"]]

is\_ok <- map\_lgl(errs, is.null)

# Extract the successful results  
subset(res, !is\_ok)

# Extract the input from the unsuccessful results  
urls <- subset(res, !is\_ok)  
urls

Unstable types – e.g. df[1,] will return first row, however if it’s a 1 row data frame it will return a vector.  
Need to aim for type-stable functions, can avoid this by being aware of [, and sapply. Use map instead of sapply! E.g. map\_chr  
Can also use an error message, if(any())

Non-standard evaluation can be an issue(e.g. dplyr, ggplot2, subset) especially as you can often evaluate objects from the global environment not just the chosen data frame.  
Better to use standard subsetting within functions (i.e.[]) rather than filter().

big\_x <- function(df, threshold) {  
 if(!"x" %in% names(df)){  
 stop("df must contain variable called x", call. = FALSE)  
 }  
 if('threshold' %in% names(df)){  
 stop("df must not contain variable called threshold", call. = FALSE)  
 }  
 dplyr::filter(df, x > threshold)  
}

Pure functions – their output only depends on their inputs and they don’t affect the outside world except through their return value (common examples – argument defaults that depend on global options).   
?options; getOption(“digits”); options(digits = 5)

The return value of a function should never depend on a global option.

A classic example of hidden dependence is read.csv() – stringsAsFactors.

### Practical Applications

First is with lists and second is combining with mutate() on data frames.

Often used with the nest and split functions.

For dplyr & tidyeval you need to use enquo() !!, quos() !!!

For ggplot2 which isn’t in tidyeval you use quo\_name() on above already enquo/quo variable. And then aes\_string().

**Summary of Each Variable**

Quick look at all variables:  
dataset %>%  
 map(~summary(.))

**Dim of multiple data frames**:

listdf <- list(training, testing, validating)

listdf %>%   
 map(~dim(.))

--

iris0 <-  
 iris %>%   
 group\_by(Species) %>%   
 nest() %>%   
 mutate(gg1 = purrr::map(data, ~ ggplot(., aes(Sepal.Length, Sepal.Width)) + geom\_point())) %>%  
 mutate(gg2 = purrr::map(data, ~ ggplot(., aes(Sepal.Length, Petal.Width)) + geom\_point())) %>%  
 mutate(g = purrr::map2(gg1, gg2, ~ gridExtra::grid.arrange(.x, .y)))

#### Functions with dplyr

ChurnPercentage <- function(data, ...){  
 data %>%   
 group\_by(..., Churn) %>%   
 summarise(Count = n()) %>%   
 mutate(Percentage = Count / sum(Count))  
}  
ChurnPercentage(dataset, PurchDaysSince)

OR

**compare1** <- function(df, x, y){  
 x <- enquo(x)  
 y <- enquo(y)  
 df %>%   
 group\_by(!! x, !! y) %>%   
 summarise(Count = n()) %>%   
 mutate(Percentage = Count / sum(Count)) %>%  
 filter(UQ(y) == 'Yes') %>%   
 arrange(desc(Percentage))  
}  
compare1(df = dataset, x = CutVar1, y = Churn7)

OR

**compare2** <- function(df, x1, x2, y){  
 x1 <- enquo(x1)  
 x2 <- enquo(x2)  
 y <- enquo(y)  
 df %>%   
 group\_by(!! x1, !! x2, !! y) %>%   
 summarise(Count = n()) %>%   
 mutate(Percentage = Count / sum(Count)) %>%   
 filter(UQ(y) == 'Yes') %>%   
 arrange(desc(Percentage))  
}  
compare2(df = dataset, x1 = PurchDaysSince, x2 = CutVar1, y = Churn7)

OR

**compare3** <- function(df, x1, x2, x3, y){   
 x1 <- enquo(x1)  
 x2 <- enquo(x2)  
 x3 <- enquo(x3)  
 y <- enquo(y)

df %>%   
 group\_by(!! x1, !! x2, !! x3, !! y) %>%   
 summarise(Count = n()) %>%   
 mutate(Percentage = Count / sum(Count)) %>%   
 filter(UQ(y) == 'Yes') %>%   
 arrange(desc(Percentage))  
}  
compare3(df = dataset, x1 = PurchDaysSince, x2 = CutVar1, x3 = CutVar2, y = Churn7)

#### GGPlot Theme

theme\_rt <- function(panel\_border = TRUE){

theme\_obj <-  
 theme\_tq() +  
 theme(  
 plot.title = element\_text(colour = "#666666", size = 12, face = "bold"),  
 plot.subtitle = element\_text(colour = "#666666", size = 10),  
 axis.title = element\_text(colour = "#666666"),  
 axis.title.x = element\_text(hjust = 0.5),  
 axis.title.y = element\_text(hjust = 0.5),  
 axis.text = element\_text(colour = "#666666"),  
 axis.ticks = element\_blank(),  
 plot.caption = element\_text(colour = "#666666", hjust = 0, size = 10),  
 legend.position = "top",  
 legend.justification = 0  
 )   
 if(panel\_border) {   
 return(theme\_obj)  
 } else {  
 theme\_obj +  
 theme(panel.border = element\_blank())   
 }  
}

#### Table Theme

table\_rt <- function (data, rows = 2, palette = 1, light\_to\_dark = TRUE) {

# Palette 1: Blue

if(palette == 1) {  
 if(light\_to\_dark) {  
 tt <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = blues9[1:rows], col="black"),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="navyblue", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))  
 } else {  
 tt <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = blues9[rows:1], col="black"),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="navyblue", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))  
 }  
 }

# Palette 2: Grey

if(palette == 2) {  
 rows <- rows + 606  
 if(light\_to\_dark) {  
 tt <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = colors()[607:rows], col="black", alpha = 0.5),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="black", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))  
 } else {  
 tt <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = colors()[rows:607], col="black", alpha = 0.5),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="black", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))  
 }  
 }

# Output

data %>%  
 mutate\_if(is.numeric, funs(round(., 2))) %>%  
 mutate\_if(is.numeric, funs(prettyNum(., big.mark = ","))) %>%  
 grid.table(theme = tt, rows = NULL)  
}

#### Map Plots

**(1)**

listplots <-  
 mtcars %>%   
 select(wt, disp, hp, drat) %>%   
 names() %>%  
 map(~ggplot(mtcars, aes\_string(x = .)) + geom\_histogram() + labs(title = .))

do.call(grid.arrange, c(listplots, nrow = length(listplots)))

**(2)**

list\_violins <-  
 dataset\_final\_numeric %>%  
 names() %>%  
 map(~ggplot(dataset\_final\_numeric, aes\_string(x = "dataset\_final$Outcome", y = (.))) +  
 geom\_violin()  
 )

do.call(grid.arrange, list\_violins, nrow = length(list\_violins))

**(3)**

Can use split with ggplot when you have more than variable to facet, e.g. AB Test & OS, then cowplot or grid.arrange.

E.g. using my purch\_segment\_plotting function

list\_plots <-  
 dataset %>%  
 mutate(OperatingSystem = fct\_lump(OperatingSystem, n = 3)) %>%  
 group\_by(AbGroup, OperatingSystem, PurchaseSegment) %>%  
 summarise(CountUsers = n()) %>%  
 mutate(Pct = 100 \* CountUsers / sum(CountUsers)) %>%  
 split(.$OperatingSystem) %>%  
 map(~purch\_segment\_plotting(., fill\_var = AbGroup) +  
 expand\_limits(y = 100))

NEED TO LOOK INTO HOW TO ADD TITLE OF LIST OBJECT TO GGPLOT

#### Count Pct to Plotting

count\_to\_pct <- function(data, ..., dep\_var = VipPlayer) {

grouping\_vars <- quos(...)  
 dep\_var <- enquo(dep\_var)

data %>%  
 group\_by(!!! grouping\_vars, !! dep\_var) %>%  
 summarise(Count = n()) %>%  
 mutate(Pct = 100 \* Count/sum(Count),  
 AboveAvg = ifelse(Pct >= 3.1, "Yes","No")) %>%  
 filter(!! dep\_var == 'Yes') %>%  
 arrange(desc(Pct))  
}

plot\_function <- function(data, var\_expr, abline\_val = 0) {

var\_expr <- enquo(var\_expr)  
 xaxis <- quo\_name(var\_expr)

data <-  
 data %>%  
 mutate(name = fct\_reorder(!! var\_expr, desc(Pct))) %>%  
 arrange(name)

data %>%  
 ggplot(aes\_string(x = paste0("fct\_reorder(name, desc(Pct))"), y = "Pct", fill = "AboveAvg")) +  
 geom\_bar(stat = "identity", alpha = 0.9) +  
 geom\_hline(yintercept = abline\_val, colour = palette\_light()[[1]],linetype = "dotted", size = 1.1) +  
 theme\_rt() +  
 scale\_fill\_manual(values = c(palette\_light()[[8]], palette\_light()[[5]])) +  
 guides(fill = FALSE) +  
 labs(x = xaxis)  
}

#### Split & Map

Create new dfs by column & apply a function over a column from each dataframe or every column from each dataframe.

dataset %>%  
 split(.$CardType) %>%  
 map(~select(., SumPurchases) %>% summary())

dataset %>%  
 split(.$CardType) %>%  
 map(~summary(.$SumPurchases))

dataset %>%  
 split(.$CardType) %>%  
 map(~summary(.))

dataset %>%  
 split(.$CardType) %>%  
 map(~cut2(.$SumPurchases, cuts = c(0, 1, 11, 50, 500)) %>% table() %>% prop.table())

#### Purchase Throughput

Split, map, cut2, cbind of 1-10 deps per player and then proportions.

purchprops <-  
 purchases %>%  
 split(.$RowNumber) %>%  
 map\_df(~cut2(.$pramount, cuts = c(0, 10, 20, 50, 200)) %>% table() %>% prop.table() \* 100  
 )

purchprops <- cbind(c("0-9","10-19","20-49","50-199","200+"),purchprops)

purchprops %>%  
 select('Purchase Segment' = 'c("0-9", "10-19", "20-49", "50-199", "200+")',  
 1:11) %>%  
 mutate\_if(is.numeric, funs(round(., 2))) %>%  
 table\_rt()

do.call("rbind", purchprops) -- will put them all back together.

# Ratios of segment to segment

purchases$PurchaseSegment <- cut2(purchases$pramount, cuts = c(0, 10, 20, 50, 200))

purchases$PurchaseSegment <- factor(purchases$PurchaseSegment,  
 labels = c("0-9", "10-19", "20-49", "50-199", "200+"))

purchases %>%  
 split(.$RowNumber) %>%  
 reduce(left\_join, by = "UserID") %>%  
 select(UserID, contains ("Segment")) %>%  
 select(  
 1,  
 Purchase1 = 2,  
 Purchase2 = 3,  
 Purchase3 = 4,  
 Purchase4 = 5,  
 Purchase5 = 6,  
 Purchase6 = 7,  
 Purchase7 = 8,  
 Purchase8 = 9,  
 Purchase9 = 10,  
 Purchase10 = 11  
 )

# Purchases 2 onwards if they've deposit 10 times.

purchase\_throughput %>%  
 filter(!is.na(Purchase10)) %>%  
 filter(Purchase2 != "0-9") %>%  
 group\_by(Purchase2, Purchase3, Purchase4, Purchase5, Purchase6, Purchase7, Purchase8, Purchase9, Purchase10) %>%  
 summarise(Count = n()) %>%  
 ungroup() %>%  
 group\_by(Purchase2) %>%  
 mutate(Pct = 100 \* Count/sum(Count)) %>%  
 arrange(desc(Pct)) %>%  
 View()

Could've also used left join e.g.

purchases\_1 <- purchases %>% filter(RowNumber == 1)  
purchases\_2 <- purchases %>% filter(RowNumber == 2)  
purchases\_3 <- purchases %>% filter(RowNumber == 3)  
purchases\_4 <- purchases %>% filter(RowNumber == 4)

purchases\_1 %>%  
 inner\_join(purchases\_2, by = "UserID") %>%  
 inner\_join(purchases\_3, by = "UserID") %>%  
 inner\_join(purchases\_4, by = "UserID") %>%  
 glimpse()

#### User Attriton

user\_attrition\_fun <- function ( data, ..., day0\_baseline = FALSE, table\_output = TRUE) {

# Initial Manipulation  
 grouping\_vars <- quos(...)  
 data <-  
 data %>%  
 group\_by(!!! grouping\_vars) %>%  
 summarise(  
 Day0 = sum(Day0),  
 Day2 = sum(Day2),  
 Day6 = sum(Day6),  
 Day14 = sum(Day14),  
 Day30 = sum(Day30),  
 Day60 = sum(Day60),  
 Day90 = sum(Day90)  
 )

# Choose Baseline  
 if(day0\_baseline) {  
 data <-  
 data %>%  
 mutate(  
 Day0Perc = 100,  
 Day2Perc = 100 \* Day2/Day0,  
 Day6Perc = 100 \* Day6/Day0,  
 Day14Perc = 100 \* Day14/Day0,  
 Day30Perc = 100 \* Day30/Day0,  
 Day60Perc = 100 \* Day60/Day0,  
 Day90Perc = 100 \* Day90/Day0  
 )  
 } else {  
 data <-  
 data %>%  
 mutate(  
 Day0Perc = 100,  
 Day2Perc = 100 \* Day2/Day0,  
 Day6Perc = 100 \* Day6/Day2,  
 Day14Perc = 100 \* Day14/Day6,  
 Day30Perc = 100 \* Day30/Day14,  
 Day60Perc = 100 \* Day60/Day30,  
 Day90Perc = 100 \* Day90/Day60  
 )  
 }

# Re-Formatting  
 data <-  
 data %>%  
 mutate\_if(is.numeric, funs(round(., 2))) %>%  
 mutate\_if(is.numeric, funs(prettyNum(., big.mark = ',')))

# Output  
 if(table\_output){  
 data %>%  
 rename(  
 'Day 0 %' = Day0Perc,  
 'Day 2 %' = Day2Perc,  
 'Day 6 %' = Day6Perc,  
 'Day 14 %' = Day14Perc,  
 'Day 30 %' = Day30Perc,  
 'Day 60 %' = Day60Perc,  
 'Day 90 %' = Day90Perc  
 ) %>%  
 select(-Day0, -Day2, -Day6, -Day14, -Day30, -Day60, -Day90) %>%  
 grid.table(theme = tt3, rows = NULL)  
 } else {  
 return(data)  
 }  
}

dataset %>%  
 user\_attrition\_fun(OperatingSystem, day0\_baseline = TRUE, table\_output = FALSE)

#### Puch Segment Plotting

Remember that facet\_wrap you can just pass in the quoted name without ~ and note the labels.

purch\_segment\_plotting <- function (data, y\_var = Pct, fill\_var = PurchaseSegment, label\_round = 2, ggtitle = '', ggsubtitle = '') {

# Editing for labels  
 y\_var <- enquo(y\_var)

data <-  
 data %>%  
 mutate(  
 PctLabels = paste0(round(!! y\_var, label\_round), '%')  
 )

# Plotting  
 y\_plot\_var <- quo\_name(y\_var)  
 fill\_plot\_var <- quo\_name(enquo(fill\_var))

data %>%  
 mutate(PurchaseSegment = fct\_rev(PurchaseSegment)) %>%  
 ggplot(aes\_string(x = "PurchaseSegment", y = y\_plot\_var, fill = fill\_plot\_var)) +  
 geom\_bar(stat = "identity") +  
 geom\_text(aes(label = PctLabels,  
 size = 1, hjust = 0, vjust = -0.5)) +  
 coord\_flip() +  
 facet\_wrap(fill\_plot\_var) +  
 theme\_rt() +  
 guides(fill = FALSE) +  
 scale\_fill\_tq() +  
 theme(legend.position='none') +  
 labs(title = ggtitle,  
 subtitle = ggsubtitle,  
 x = 'Purchase Segments',  
 y = 'Proportions %')  
}

#### Cuts Analysis

cuts\_analysis <- function(data, ...){  
 grouping\_vars <- quos(...)

data$PurchaseSegment <- cut2(data$SumPurchases, cuts = c(0, 1, 11, 50, 250, 500))  
 data$PurchaseSegment <- factor(data$PurchaseSegment, labels = c("0", "1-11","11-50","50-250","250-500","500+"))

data <-  
 data %>%  
 filter(PurchaseSegment != 0) %>%  
 group\_by(!!! grouping\_vars) %>%  
 count(!!! grouping\_vars, PurchaseSegment) %>%  
 mutate(Percentage = (n/sum(n)))

return(data)

}

# Example of cool plot I made, also did another where I rbinded a baseline to include.

dataset %>%  
 mutate(BN = as.factor(as.character(BN))) %>%  
 mutate(BN = fct\_lump(BN, n = 11)) %>%  
 cuts\_analysis(BN) %>%  
 mutate(Percentage = 100 \* Percentage) %>%  
 ggplot(aes(x = Segmenty = Percentage, fill = BN)) +  
 geom\_bar(stat = "identity", alpha = 0.8) +  
 facet\_wrap(~BN) +  
 geom\_text(aes(label=paste0(round(Percentage,0), "%"), size=1, hjust=0.5, vjust=-0.15)) +  
 theme\_rt() +  
 scale\_fill\_tq() +  
 guides(fill = FALSE) +  
 theme(legend.position='none') +  
 labs(title = 'Test Title',  
 subtitle = 'bla bla \n',  
 caption = 'Test Caption') +  
 expand\_limits(y = (77))

#### Cut2

CutFun <- function(df, cutcol, numcuts = 5){  
 df %>%   
 mutate(CutVar = cut2(cutcol, g = numcuts))  
}  
dataset <- CutFun(dataset, dataset$CountPurchases, 10)

#### Browser Analysis

browserlist <-  
 users %>%  
 split(.$BrowserName)

**factor\_fun** <- function(data, var\_expr, n = 12) {  
 var\_expr <- enquo(var\_expr)

data %>%  
 mutate(Var = fct\_lump(!! var\_expr, n)) %>%  
 count(Var) %>%  
 mutate(Prop = n/sum(n)) %>%  
 arrange(desc(n))

}

# Hardware

hardwareplotlist <-  
 browserlist %>%  
 map(~factor\_fun(., RegHardware)) %>%  
 map(~rename(., RegHardware = Var)) %>%  
 map(~filter(., !RegHardware == 'Other')) %>%  
 map(~ggplot(., aes(x = fct\_reorder(RegHardware, n), y = n)) +  
 geom\_bar(stat = "identity", alpha = 0.8, colour = "black", fill = "navyblue") +  
 coord\_flip() +  
 theme\_rt() +  
 scale\_fill\_tq() +  
 guides(fill = FALSE) +  
 scale\_y\_continuous(labels = comma) +  
 labs(x = names(.))  
 )

Grid.arrange…

# Cut function

browserlist %>%  
 map(~cut2(.$Age, cuts = c(18, 30, 35, 40, 50))) %>%  
 map(~table(.) %>% prop.table())

# Df mapping

browserlist %>%  
 map(~cut2(.$Age, cuts = c(18, 30, 35, 40, 50))) %>%  
 map\_df(~table(.) %>% prop.table()) %>%  
 cbind(AgeSegments) %>%  
 select(AgeSegments, 'Chr, 'FB) %>%  
 mutate\_if(is.numeric, percent) %>%  
 grid.table(theme = tt3, rows = NULL)

# Mapping

browserlist %>%  
 map(~group\_by(., AgeSegment) %>%  
 summarise(Count = n(),  
 SumPurchaes = sum(USDPurchasesDay6),  
 AvgPur = mean(USDPurchasesDay6)))

#### Recipes: Model Pre-Processing

rec\_obj <-  
 recipe(Outcome ~ ., data = list\_modelling[[1]]) %>%  
 step\_center(all\_numeric()) %>%  
 step\_scale(all\_numeric()) %>%  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%  
 prep(data = list\_modelling[[1]])

list\_modelling <-  
 list\_modelling %>%  
 map(~bake(rec\_obj, newdata = .))

#### Proportions

# Map prop table across multiple columns:

dataset\_final %>%  
 select\_if(is.factor) %>%  
 map(~table(.) %>% prop.table())

# Prop table function

prop\_function <- function(data, ...) {  
 var <- quos(...)  
 data %>%  
 select(!!! var) %>%  
 table() %>% prop.table()  
}

dataset %>%  
 prop\_function(VipPlayer)

#### Null Replacement

nullreplacementfun <- function(data, daynum) {

data[is.na(data[[daynum]]),][[daynum]] <- as.integer(0)

return(data)

}

#### Normalise

normalise\_fun <- function(x) {

controlnum <-  
 summary\_tbl %>%  
 filter(AbGroup == 'Control') %>%  
 select(CountUsers) %>%  
 as.numeric()

testnum <-  
 summary\_tbl %>%  
 filter(AbGroup == 'Test') %>%  
 select(CountUsers) %>%  
 as.numeric()

normalise <- testnum / controlnum

x \* normalise  
 }

summary\_tbl %>%  
 select(-NetwinPP) %>%  
 filter(AbGroup == 'Control') %>%  
 mutate\_if(is.numeric, funs(normalise\_fun)) %>%  
 rbind(  
 summary\_tbl %>%  
 select(-NetwinPP) %>%  
 filter(AbGroup == 'Test')  
 )

#### Log Comparison

Log10\_fun <- function(data, outcome\_var, independent\_var){

outcome\_var <- enquo(outcome\_var)  
 independent\_var <- enquo(independent\_var)

data %>%  
 select(!! outcome\_var, !! independent\_var) %>%  
 filter(!! independent\_var > 0) %>%  
 mutate(  
 Outcome = !! outcome\_var %>% as.factor() %>% as.numeric(),  
 PreLog = !! independent\_var,  
 Log = log(!! independent\_var)  
 ) %>%  
 select\_if(is.numeric) %>%  
 select(- !! independent\_var) %>%  
 correlate() %>%  
 focus(Outcome) %>%  
 fashion()  
}

#### Top Rank Compare

toprankconversion <- function(df, x){  
 x <- enquo(x)

df %>%   
 group\_by(!! x) %>%   
 summarise(TotalPlayers = n(),  
 TopPlayers = sum(PurchaseRank == 'Top')) %>%   
 mutate(TotalPercentage = TotalPlayers / sum(TotalPlayers),  
 TopConversion = TopPlayers/TotalPlayers) %>%   
 arrange(desc(TopConversion)) %>%   
 mutate(TotalPercentage = percent(round(TotalPercentage, 4)),  
 TopConversion = percent(round(TopConversion, 4))) %>%   
 select(!! x, 'Total Players' = TotalPlayers, 'Total Percentage' = TotalPercentage, 'Top Players' = TopPlayers, 'Top Conversion' = TopConversion)  
}  
toprankconversion(dataset, Gender) %>%   
 mutate\_if(is.numeric, funs(prettyNum(., big.mark=","))) %>%   
 grid.table(rows = NULL, theme = tt3)

#### Correlation Fun

**(Can’t get it working), keeps correlating with the first column.**

correlatefun <- function(df, y){   
 y <- enquo(y)  
 y <- select(df, UQ(y)) %>% sapply(as.numeric)  
 df1 <- df[,-1]

df1 %>%  
 correlate() %>%  
 focus(y)  
 # rename('feature' = 'rowname') %>%   
 # arrange(abs(y))  
 #mutate(feature = as\_factor(feature))  
}

#### CorrelationPlot

(must pass in cor\_analysis df with Outcome & Feature)

cor\_analysis <-   
 datasetfilter[,-1] %>%   
 mutate(Outcome = ifelse(datasetfilter$Churn7 == 'Yes', 1, 0)) %>%   
 correlate() %>%   
 focus(Outcome) %>%   
 rename('Feature' = 'rowname') %>%   
 arrange(abs(Outcome)) %>%   
 mutate(Feature = as\_factor(Feature))

corrplot <- function(df, title = 'Correlation Analysis'){  
 df %>%  
 ggplot(aes(x = Outcome, y = fct\_reorder(Feature, desc(Outcome)))) +  
 geom\_point() +  
 geom\_segment(aes(xend = 0, yend = Feature),  
 color = palette\_light()[[2]],  
 data = df %>% filter(Outcome > 0)) +  
 geom\_point(color = palette\_light()[[2]],  
 data = df %>% filter(Outcome > 0)) +  
 # Negative Correlations - Prevent churn  
 geom\_segment(aes(xend = 0, yend = Feature),  
 color = "chartreuse4",  
 data = df %>% filter(Outcome < 0)) +  
 geom\_point(color = "chartreuse4",  
 data = df %>% filter(Outcome < 0)) +  
 # Vertical lines  
 geom\_vline(xintercept = 0, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 geom\_vline(xintercept = -0.25, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 geom\_vline(xintercept = 0.25, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 # Aesthetics  
 theme\_bw() +  
 labs(title = title,  
 subtitle = "Positive Correlations (contribute to Outcome), Negative Correlations (prevent Outcome)",  
 y = "Feature Importance")  
 }

correlationplot(cor\_analysis, title = "title Correlation")

**Log Correlation Comparison Function (just can’t figure out the rename part)**

logcompare <- function(df, x, y){  
 x <- enquo(x)  
 y <- enquo(y)  
 varname <- names(df %>% select(!! y))

df %>%  
 select(!! y,!! x) %>%   
 mutate(  
 Outcome = UQ(y) %>% as.factor() %>% as.numeric(),  
 Log = log(UQ(x))  
 ) %>%  
 select(!! x, Outcome, Log) %>%  
 correlate() %>%  
 focus(3) %>%  
 fashion() #%>%   
 #mutate\_(paste(varname) = 'Outcome')  
}

logcompare(df = dataset, x = PurchRegularityRatio, y = Churn7)

**Percentile**

library(dplyr)  
library(tidyr)  
library(broom)  
library(purrr)

mtcars %>%  
 nest(-cyl) %>%  
 mutate(Quantiles = map(data, ~ quantile(.$mpg))) %>%   
 unnest(map(Quantiles, tidy))

#### Spread

spreadfun <- function(data, ...) {  
 data %>%  
 select(Status, ...) %>%  
 spread(...)  
}

#### Time Series Plot

Must have column with baseline for dates prior to change.

ts\_plot <- function (data, x, y, ggtitle = '') {

x <- quo\_name(enquo(x))  
 y <- quo\_name(enquo(y))

data %>%  
 ggplot(aes\_string(x = x, y = y, colour = "Baseline")) +  
 geom\_point(alpha = 0.8, size = 2.5) +  
 geom\_line(size = 1.2) +  
 theme\_rt() +  
 scale\_colour\_manual(values = c(palette\_dark()[[1]], palette\_dark()[[5]])) +  
 labs(  
 title = ggtitle  
 )  
}

---

ts\_plot <- function(data, x, y){

# x <- substitute(x)  
 # y <- substitute(y)

data %>%  
 ggplot(aes(x, y)) +  
 geom\_point(colour = palette\_light()[[1]], size = 2) +  
 geom\_smooth(method = "loess", colour = palette\_light()[[1]], size = 1.1) +  
 theme\_rt() +  
 expand\_limits(y = 0) +  
 scale\_y\_continuous(labels = comma)  
}

ts\_plot(dataset, dataset$PurchaseDate, dataset$SumPurchases)

#### Simple Division Map

newreg\_division <- function(x) {  
 x / dataset$NewRegos  
}

dataset %>%  
 select(NewRegos:TotalPurchasesNewRegos, PurchasingUsers:LowQualityUsers) %>%  
 map\_df(~newreg\_division(.))

#### Split T-Tests

mtcars %>%  
 split(as.character(.$cyl)) %>%  
 map(~t.test(.x$mpg))

But, purrr::by\_slice() pairs nicely with dplyr::group\_by().

mtcars %>%  
 dplyr::select(cyl, mpg) %>%  
 group\_by(as.character(cyl)) %>%  
 by\_slice(~ t.test(.x$mpg))

Or, you could skip purrr entirely using dplyr:::summarise().

mtcars %>%  
 dplyr::select(cyl, mpg) %>%  
 group\_by(as.character(cyl)) %>%  
 summarise(t\_test = data\_frame(t.test(.$mpg)))

If the nested data.frame is confusing, broom can help us get an easy data.frame summary of the results.

mtcars %>%  
 group\_by(as.character(cyl)) %>%  
 by\_slice(~tidy(t.test(.x$mpg))) %>%  
 unnest()

<https://stackoverflow.com/questions/35558766/purrr-map-a-t-test-onto-a-split-df>

### Resources

Writing an r package - <https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/>

<https://stackoverflow.com/questions/45407316/writing-own-function-using-dplyr-and-group-by-how-to-continue-with-changed-col>

map, nest & many models - <http://omaymas.github.io/Climate_Change_ExpAnalysis/>

<http://ijlyttle.github.io/isugg_purrr/presentation.html>

map & ggplot2 -

<https://stackoverflow.com/questions/46570609/iteratively-apply-ggplot-function-within-a-map-function>

<https://stackoverflow.com/questions/42518156/use-purrrmap-to-apply-multiple-arguments-to-a-function>

<https://www.r-bloggers.com/make-ggplot2-purrr/>

mutate in functions - <https://stackoverflow.com/questions/26003574/dplyr-mutate-use-dynamic-variable-names>

Percentiles - <https://stackoverflow.com/questions/30488389/using-dplyr-window-functions-to-calculate-percentiles>

### Apply



apply(X, Margin, Function) – X is the object, margin is rows (1) or columns (2) and then the function you want to apply. The apply function uses loops.  
E.g. apply(dataset, 1, mean)

Use lapply to apply to all objects within a list. E.g. transpose all the objects of a list (switch the rows and columns) using the t function.  
E.g. mynewlist <- lapply(dataset, t)  
OR   
add new numbers to each month row – lapply(dataset, rbind, MonthNum = 1:12)  
OR  
rowMeans/rowSums/colMeans/colSums – lapply(dataset, rowMeans)

## Working with Dates

ISO 8601: "yyyy-mm-dd"

Must be fixed digits, doesn't need a separator but if you do it must be "-"

Can subtract a date from another or ask logical arguments, e.g. as.Date("yyyy-mm-dd") + 1

Date objects are stored as days since 1970-01-01

library(readr) -  has read\_csv() which will recognise dates in a few formats.  
library(anytime) - anytime() sole goal is to automatically parse strings as dates regardless of format.

# Not sure about the dummy grouping part:

ggplot(releases, aes(x = date, y = type)) +  
  geom\_line(aes(group = 1, color = factor(major))) +  
  scale\_x\_date(date\_breaks = "10 years", date\_labels = "%Y")

# Example of looking at latest date, e.g. how long has it been since last release:

last\_release\_date <- max(releases$date)  
last\_release <- filter(releases, date == last\_release\_date)  
Sys.Date() - last\_release\_date

Datetimes

POSIXlt - list with named componenets  
POSIXct - seconds since 1970-01-01 00:00:00; better for data frames.  
as.POSIXct("YYY-MM-DDTHH:MM:SS") - local time  
as.POSIXct("YYY-MM-DD HH:MM:SSZ") – UTC  
as.POSIXct("YYY-MM-DD HH:MM:SSZ", tz = "UTC")  
as.POSIXct("2010-10-01 12:12:00", tz = "America/Los\_Angeles")

dataset$Date<-as.POSIXct(dataset$Date,format='%Y-%m-%d')  
Or use as.POSIXlt for GMT  
as.POSIXct(dataset$DateTimeHour,format='%d/%m/%Y %H:%M')

scale\_x\_datetime(breaks = date\_breaks("1 day"))

Current date/time:  
Sys.time()  
Sys.Date()  
as.POSIXlt(Sys.time(), "GMT")  
Doesn’t seem to work right now: format(as.POSIXlt(Sys.time(), "GMT"),'%d/%m/%Y %H:00:00')

Can then remove current day (doesn’t seem to work right now)  
filter(dataset, !Date == Sys.Date())

Shortened to hour:  
format(Sys.time(), '%d/%m/%Y %H:00:00')

Often with regression analysis we convert dates to numeric numbers since the event started. E.g. delta\_time is change in time since even started, delta\_temperature is change in temp since event started.

mutate(  
 delta\_time = as.numeric(instant) - as.numeric(instant[[1]]),  
 delta\_temperature = temperature - temperature[[1]]  
 )

### Lubridate

library(lubridate)

ymd(“2013-02-27”)  
dmy(“27/2/13”)  
mdy\_hm(z)  
parse\_date\_time(c(“Feb 27th”, “27th Feb 2017”), order = c(“mdy”, “dmy”))

Short dates - dOmY", "OmY", "Y"

a = day of week; I = 12 hour time, a/p = am or pm

parse\_date\_time(x, orders = "amdyIp")

Lubridate grouping by times:  
hour(Sys.time())

dates <- as.Date(as.character(dates), "%Y%m%d")

library(lubridate)  
ymd/dmy/mdy()

ymd(Sys.Date())

ymd\_hms(Sys.time())

# For dates if you just want ym you will need to add a day.  
E.g. dataset$DatePeriod <- as.Date(as.yearmon((as.character(dataset$DatePeriod))))

# Use make\_date() to combine year, month and mday

akl\_hourly <- akl\_hourly\_raw %>%   
 mutate(date = make\_date(year = year, month = month, day = mday))

# Parse datetime\_string   
akl\_hourly <- akl\_hourly %>%   
 mutate(  
 datetime\_string = paste(date, time, sep = "T"),  
 datetime = ymd\_hms(datetime\_string)  
 )

Extracting parts of datetime – year, month, day, hour, min, second, wday, yday, tz, quarter, semester()

Can set the property by putting it on left side, e.g. year(x) <- 2017

Logical values (TRUE/FALSE) – leap\_year, am, pm, dst.

Quick look at date distribution:  
year(release\_time) %>% table()  
wday(releases$datetime, label = TRUE, abbr = FALSE) %>% table()

# How often is the hour before 12 (noon)?  
mean(hour(release\_time) < 12)

**Rounding dates**:  
round\_date(unit = “hour”); ceiling\_date(); floor\_date()  
round\_date(r\_3\_4\_1, unit = "5 minutes")

lubridate::hour(purchases$PurchaseTime)

# Create day\_hour, datetime rounded down to hour  
akl\_hourly <- akl\_hourly %>%  
  mutate(  
    day\_hour = floor\_date(datetime, unit = "hour")  
  )

# Find day\_hours with n != 2    
akl\_hourly %>%   
  count(day\_hour) %>%  
  filter(n != 2) %>%   
  arrange(desc(n))

days\_in\_month(x)

# If only interested in week days:  
wday(date) %in% 2:6.

# any() - will give true or false  
rainy\_days <- akl\_day %>%   
  group\_by(month, day) %>%  
  summarise(  
    any\_rain = any(rainy)  
  )

# get **difference** between now and latest date.  
latest\_date <- filter(dataset, date == max(date)  
difftime(Sys.Date(), latest\_date, units = "weeks") # or today, now()

**Time Spans**:  
Period – Human concept of a time span, e.g. period of one day is just until the same time the next date. *today() + days(1)*  
Duration – Stopwatch concept of a time span, e.g. datetime + duration of one day (datetime + 86400 secs) *ddays()*

Add **sequences** of dates:  
# Sequence of two weeks from 1 to 26  
every\_two\_weeks <- 1:26 \* weeks(2)  
# Create datetime for every two weeks for a year  
today\_8am + every\_two\_weeks

---  
month\_seq <- 1:12 \* months(1)  
today() + month\_seq  
---  
%m+% and %m-% roll back to last existing date if NA.  
today %m+% month\_seq

**Intervals**

Once you have an interval you can find out certain properties like its start, end and length with int\_start(), int\_end() and int\_length()

monarchs <- monarchs %>%  
 mutate(reign = datefrom %--% dateto)

monarchs %>%  
 mutate(length = int\_length(reign) %>%   
 arrange(desc(length)) %>%  
 select(name, length, dominion)

Can test if a value is within an interval:  
y2001 <- ymd("2001-01-01") %--% ymd("2001-12-31")  
ymd("2002-03-30") %within% y2001 (returns FALSE)  
int\_overlaps() will return true if two intervals overlap.

Can extract the duration and period from an interval:  
monarchs <- monarchs %>%  
 mutate(  
 duration = as.duration(reign),  
 period = as.period(reign)  
)

**Time Zones**

Time zone must come from OlsonNames()

Force\_tz() – change the timezone without changing the clock time.   
E.g. game2\_local <- force\_tz(game2, tzone = "America/Edmonton")

with\_tz() – view the same instant in a different time zone, doesn’t change the underlying time just how it is displayed.  
E.g. with\_tz(game2\_local, tzone = "Pacific/Auckland")

as.period(game2\_local %--% game3\_local)

**Fast parsing** with lubridate::fast\_strptime

microbenchmark(  
 ymd\_hms = ymd\_hms(dates),  
 fasttime = fastPOSIXct(dates),  
 fast\_strptime = fast\_strptime(dates,   
 format = "%Y-%m-%dT%H:%M:%SZ"),  
 times = 20)

### Plots by Date

mm - y

scale\_x\_date(date\_breaks = "month", date\_labels = "%b %y")

ggplot(akl\_daily, aes(x = yday, y = max\_temp)) +  
  geom\_line(aes(group = year), alpha = 0.5)

# Examine distribtion of max\_temp by month  
library(ggridges)  
ggplot(akl\_daily, aes(x = max\_temp, y = month, height = ..density..)) +  
  geom\_density\_ridges(stat = "density")

# hms plot  
ggplot(akl\_hourly, aes(x = time, y = temperature)) +  
 geom\_line(aes(group = make\_date(year, month, mday)), alpha = 0.2)

### Other Packages

Hms  
as.hms()

library(fasttime)  
fastPOSIXct(dates)

Cheatsheet:  
<https://www.rstudio.com/resources/cheatsheets/>

<https://stackoverflow.com/questions/29687704/convert-local-datetime-to-utc-in-r>

## Working with Strings

Should use “ instead of ‘ as much as possible.

Caan use a backslash to use an apostrophe or quote inside a string.

writeLines(strings, sep = " ")

The function cat() is very similar to writeLines(), but by default separates elements with a space, and will attempt to convert non-character objects to a string.

A sequence in a string that starts with a \ is called an escape sequence and allows us to include special characters in our strings.

formatC() has a format argument that takes a code representing the required format. The most useful are:  
"f" for fixed,  
"e" for scientific, and  
"g" for fixed unless scientific saves space

tolower(names(data))

strsplit(names(data), "\\."

sub("\_","", names(data),) #removes characters

# Grepl – to find strings  
coal <- grepl("coal", SCC$Short.Name, ignore.case = TRUE)  
subSCC <- SCC[coal,]

Regular Expressions:  
^I think = start of the line  
morning$ = end of the line  
[Bb][Uu][Ss][Hh] = all version of the word  
^[Ii] am = start combining them  
^[0-9][a-zA-Z] = range  
[^?.]$ = indicates no matching the indicated class  
9.11 = “.” indicates any character  
flood|fire = or

Text contains:  
ProfitLoss = ifelse(grepl("Profit",NetWinSegment), "profit","loss"))

Split by underscore, store in matrix not list & extract first colum:  
mutate(model\_type = str\_split(model\_id, "\_", simplify = T) %>% .[,1])

Case\_when with text contains:  
mutate(Hardware = case\_when(  
 grepl("iPad", Hardware) ~ "iPad",  
 grepl("iPhone", Hardware) ~ "iPhone",  
 TRUE ~ "Other"  
 )  
)

### stringr

Filter – column contains text:

df %>%  
 filter(str\_detect(letters, "a|f|o") )

mtcars %>%  
 filter(str\_detect(type, 'Toyota|Mazda'))

str\_c – concatenate, similar to paste0. Prints NA for missing values.

str\_replace\_na() – replaces missing strings with any string you choose.

str\_length()

str\_sub() - extracts parts of strings based on their location.   
E.g. str\_sub(x, 1, 4) asks for the substring starting at the first character, up to the fourth character.  
Both start and end can be negative integers, in which case, they count from the end of the string. For example, str\_sub(x, -4, -1), asks for the substring starting at the fourth character from the end, up to the first character from the end, i.e. the last four characters.

str\_split() – Use the simplify = TRUE argument when you want to split each string into the same number of pieces as a matrix.  
date\_ranges <- c("23.01.2017 - 29.01.2017", "30.01.2017 - 06.02.2017")  
str\_split(date\_ranges, fixed(" - "))  
str\_split(date\_ranges, fixed(" - "), n = 2, simplify = TRUE)  
both\_names <- c("Box, George", "Cox, David")  
both\_names\_split <- str\_split(both\_names, fixed(", "), n = 2, simplify = TRUE)

(dataset)  
names(iris) <- str\_replace\_all(names(dataset), "[.]", "\_")

dataset <- c("Go to Heaven for the climate, Hell for the company.")  
str\_extract\_all(dataset, "[H][a-z]+ ")  
#"Heaven " "Hell "

tq\_transmute\_fun\_options()$xts %>%  
 stringr::str\_subset("^apply")

**Paste**

paste(newgeodataset$location, ': ', "P/P - ", prettyNum(newgeodataset$TotalCount, big.mark=","), '; ', "Top Pl Conv - ", percent(newgeodataset$TopConversion/100))

paste("Clusters of clients")

paste0(as.character(c(seq(15, 0, -5), seq(5, 15, 5))), "m")

paste(“….”)

paste0("H", semester(date)))

## Web Data

read.csv & red.delim (tsv)

download.file(url = csv\_url, destfile = "feed\_data.csv")

### APIs

Application Programming Interfaces - lets you make queries for specific bits of data, can be used to expose data automatically.

APIs are server components to make it easy for your code to interact with a service and get data from it. R features many "clients" - packages that wrap around connections to APIs so you don't have to worry about the details.

Use a client if you can, google "CRAN[name of website]"

E.g.   
library(pageviews)  
hadley\_pageviews <- article\_pageviews(project = "en.wikipedia", "Hadley Wickham")

API Access tokens to control & identify use so APIs are not overwhelmed.

E.g.  
library(birdnik)   
vector\_frequency <- word\_frequency(api\_key, "vector")

### HTTP requests

Conversation between your machine & server. Main methods - GET & POST. Many others like HEAD & DELETE.

Using httr to interact with APIs directly

library(httr)  
get\_result <- GET("http://httpbin.org/get")  
post\_result <- POST("http://httpbin.org/post", body = "this is a test")/  
pageview\_response <- GET(url)  
pageview\_data <- content(pageview\_response)

Every call has a response status code - if it starts with 2 or 3 it's fine, if 4 then your code is broken and 5 their code is broken.

Handling errors

request\_result <- GET(fake\_url)   
if(http\_error(request\_result)){   
warning("The request failed")  
} else {   
content(request\_result)  
}

URLs are directory based or parameter based.

Constructing directory-based URLs can be done via paste(), which takes an unlimited number of strings

directory\_url <- paste("http://swapi.co/api", "people", 1, sep = "/")  
result <- GET(directory\_url)

Constructing parameter-based URLs can also be done with paste(), but the easiest way to do it is with GET() and POST() themselves, which accept a query argument consisting of a list of keys and values.

query\_params <- list(nationality = "americans",  
 country = "antigua")

parameter\_response <- GET("https://httpbin.org/get", query = query\_params)

Respectful API usage - user agent & rate-limiting

GET(url, user\_agent("my@email.address this is a test"))  
urls <- c("http://fakeurl.com/api/1.0/",  
 "http://fakeurl.com/api/2.0/")

for(url in urls){  
result <- GET(url)  
Sys.sleep(5)  
}

Tying it all together

get\_pageviews <- function(article\_title){  
url <- paste(  
 "https://wikimedia.org/api/rest\_v1/metrics/pageviews/per-article/en.wikipedia/all-access/all-agents",  
article\_title,  
"daily/2015100100/2015103100",  
sep = "/"  
 )   
response <- GET(url, user\_agent("my@email.com this is a test"))   
if(http\_error(response)){  
stop("the request failed")  
}  
content(response)  
}

### Handling JSON and XML

Two most common formats of data returned by APIs.

**JSONS** - objects (name & value) & arrays.

Basic Example:  
resp\_json <- rev\_history("Hadley Wickham")  
http\_type(resp\_json)  
content(resp\_json, as = "text")  
content(resp\_json, as = "parsed")

library(jsonlite)  
fromJSON(content(resp\_json, as = "text"))

The output from parsing JSON is a list. One way to extract relevant data from that list is to use a package specifically designed for manipulating lists, rlist.

list.select() extracts sub-elements by name from each element in a list.

list.stack() will stack the elements of a list into a data frame.

library(rlist)  
 list.stack(  
 list.select(movies\_list, title, year)  
)

Can also use dplyr

revs <- content(resp\_json)$query$pages$`41916270`$revisions  
revs %>%  
 bind\_rows %>%  
 select(user, timestamp)

**XML**

library(xml2)  
resp\_xml <- rev\_history("Hadley Wickham", format = "xml")  
http\_type(resp\_xml)  
rev\_text <- content(resp\_xml, as = "text")  
rev\_xml <- read\_xml(rev\_text)  
xml\_structure(rev\_xml)

## Packages

packages <- c("tidyquant", "purrr","Hmisc","ggthemes","scales","gridExtra","grid","knitr","rvest", "tidyverse","forcats","Amelia","corrplot", "party","corrr", "corrplot", "recipes","caret","rpart","h2o","DMwR")

lapply(packages, require, character.only = TRUE)

CLVTools package – need to check this out

library(dplyr)  
library(tidyr)  
library(sqldf)  
library(lsr)  
library(dummies)  
library(Hmisc)  
library(tidyquant)  
library(tibble)

library(ggplot2)  
library(ggExtra)  
library(gridExtra)  
library(ggmap)  
library(ggthemes)  
library(plotly)  
library(googleVis)  
library(scales)  
library(directlabels)

library(cluster)  
library(mclust)  
library(rsample)  
library(recipes)  
library(DMwR)  
library(corrr)  
library(h20)  
library(caret)  
library(keras)  
library(caretEnsemble)  
library(randomForest)  
library(e1071)  
library(class)  
library(xgboost)  
library(party)  
library(rvest)

library(lime)  
library(yardstick)  
library(reprtree)  
library(DiagrammeR)  
library(Ckmeans.1d.dp)  
library(MLmetrics)  
library(ROCR)  
library(rattle)  
library(microbenchmark) # can measure run times.

options(repos='http://cran.rstudio.org')  
have.packages <- installed.packages()  
cran.packages <- c('devtools','plotrix','randomForest','tree')  
to.install <- setdiff(cran.packages, have.packages[,1])  
if(length(to.install)>0) install.packages(to.install)

library(devtools)

if(!('reprtree' %in% installed.packages())){  
 install\_github('araastat/reprtree')  
}  
for(p in c(cran.packages, 'reprtree')) eval(substitute(library(pkg), list(pkg=p)))

library(reprtree)

### “sqldf” (using SQL)

sqldf("SELECT  
 day  
 , avg(temp) as avg\_temp  
 FROM dataset  
 GROUP BY  
 day;")

## RMarkdown

Can make an html doc or slide presentation.

Headings ##, ###, ####

Bullet points “-“

```{r, include = FALSE, message = FALSE, echo = FALSE, comment = “”, eval = FALSE, fig.align = ‘center’, results = ‘hide’}

Can post to github – pages then go to the website and rename:  
Username.github.io/remove “blob/gh-pages” with path to html

<https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf>

## Shiny

A web application framework for R – allows you to create a graphical interface so users can interact with visualizations, models and algorithms without needing to know R themselves.

Shiny uses Bootstrap.

<http://shiny.rstudio.com/tutorial/>

## R Interface to Python

Using package “reticulate”.

<https://rstudio.github.io/reticulate/index.html>

<https://cran.r-project.org/web/packages/reticulate/index.html>

## Future Packages



Rcpp - Rcpp provides a powerful API on top of R, make function in R extremely faster.

library(car) – for confidence ellipse

Caret – for modelling.

vowpal wabbit – linear models.

Good for understanding models for what is driving metrics.

Pandas

Shiny – turn R into web apps.

Regression analysis - use LASSO from lars (or glmnet) package

“glmnet” - Lasso and elastic-net regression methods with cross validation

“Forecast” – time series models

“RColorBrewer”

“lubridate” – dealing with dates.

“ggvis” – interactive web based graphics

“reshape2”

“prophet” - Tool for producing high quality forecasts for time series data that has multiple seasonality with linear or non-linear growth.

knitr - Easy dynamic report generation in R.

rmarkdown

latex

rstats

## Good Sources

Practical regression in R - <https://cran.r-project.org/doc/contrib/Faraway-PRA.pdf>

Regression - <http://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-tutorial-and-examples>

Good packages - <http://blog.yhat.com/posts/10-R-packages-I-wish-I-knew-about-earlier.html>

<https://www.datacamp.com/>

<https://www.codecademy.com/>

<http://tryr.codeschool.com/>

Kaggles:  
<https://www.kaggle.com/cloudly/credit-default>   
<https://www.kaggle.com/hendraherviawan/customer-segmentation-using-rfm-analysis-r>

# **R: Statistical Modelling**

## Correlation

Boxplot as discretized/conditioneed scatterplots:  
ggplot(data = ncbirths,   
       aes(x = cut(weeks, breaks = 5), y = weight)) +   
  geom\_boxplot()

Correlation coefficient between - 1 and 1; near 1 is nearly perfect positive correlation. 0.5 is moderate, 0.2 is weak and 0 is no relationship (knowing x will give no indication of y).

Sign -> direction

Magnitude -> strength

Only captures a linear relationship though, common to have strong non-linear relationship. E.g. can have quadratic relationship.

Usually talking about Pearson product-moment correlation.

Example 1:  
mammals %>%  
 summarize(N = n(),   
 r = cor(BodyWt, BrainWt),   
 r\_log = cor(log(BodyWt), log(BrainWt)))

Example 2:  
bdims %>%  
  group\_by(sex) %>%  
  summarize(N = n(), r = cor(hgt, wgt))

Correlation doesn't mean causation. Can only show bivariate relationships (2 variables).

Spurious correlation - remarkable but non-sensical where patterns are actually random noise. Often caused by the role of time.

filter(abs(cor) > 0.2)

cor(datasetfilter$PurchAmountRatio, ifelse(datasetfilter$Churn7 == 'Yes', 1, 0))  
cor(log10(datasetfilter$PurchAmountRatio), ifelse(datasetfilter$Churn7 == 'Yes', 1, 0))  
# Or use my log compare function.

## Regression

Least squares - determine best fit line.

General statistical model: Response = f(explanatory) + noise   
Some mathematical function f that can translate values of one variable into values of another, except there is some randomness in the process.

Response = intercept + (slope \* explanatory) + noise

Least squares - easy/deterministic/unique, residuals sum to zero, line must pass through x, y.

Linear model - y-hat is expected value given corresponding x; beta-hats are estimates of true, unknown betas; residuals are estimates of true, unknown epsilons; error is misleading, rather call it noise.

bdims\_summary %>%  
 mutate(slope = r \* sd\_y / sd\_x,   
 intercept = mean\_y - slope \* mean\_x)

Regression to the mean - the basic idea that extreme random observations will tend to be less extreme upon a second trial. E.g. if a father is taller than the average, they're child tends to also be taller than the mean but by less.

geom\_abline(slope = 1, intercept = 0) +   
 geom\_smooth(method = "lm", se = FALSE)

### Interpreting LM Models

Residuals - look for a symettrical distribution across these points on the mean value zero. If not symmetrical it means that the model predicts certain points that fall far away from the actual observed points.

Coefficients - Theoretically, in simple linear regression, the coefficients are two unknown constants that represent the intercept and slope terms in the linear model. Ultimately, want to find an intercept and a slope such that the resulting fitted line is as close as possible to the 50 data points in our data set.  
 > Estimate - Intercept is the average value of x in our dataset; slope shows for every unit of x a corresponding value of y.  
 > Standard error - Average amount coefficient estimates vary from actual average value of our response variable. Shows how much the values could vary if we ran model again and again. Can be used to compute confidence intervals and statistically test hypothesis of existince of relationship.  
 > T-value - how many STDevs our coefficient estimate is away from 0. Far away from 0 means we can reject null hypothesis and declare a relationship exists. Want it to be far from 0 and large relative to the standard error.  
 > Pr(>t) - Small p value indicates relationship. Typically 5% or lower is a good cut-off point.

Residual standard error - quality of a linear regression fit.

Multiple R-squared, Adjusted R-squared - Provides a measure of how well the model is fitting the actual data, proportion of variance. 0 says it explains no variance in response and 1 indicates it does explain the observed variance in the response variable. A good level would depend on the case. In multiple regression settings, the R2 will always increase as more variables are included in the model. That’s why the adjusted R2 is the preferred measure as it adjusts for the number of variables considered.

F-Statistic - good indicator of whether a relationship between predictor and response variable exists. The further the F-statistic is from 1 the better it is. However, how much larger the F-statistic needs to be depends on both the number of data points and the number of predictors. When number of data points is large then only a little bit larger than 1 is sufficient to reject null hypothesis; reverse is true if number of data points is small then we want large f-statistic.

<https://feliperego.github.io/blog/2015/10/23/Interpreting-Model-Output-In-R>

### Interpreting Coefficients

lm(response var ~ explanatory, data = dataset)

Pay careful attention to units and scales.

Intercept isn’t usually very interesting, the slope is more interesting. E.g. for every unit of one variable (shown) the other has x units.

## Time Series Analysis

library(tibbletime)

ve\_package\_frequency\_tbl <- variance\_explained\_tbl %>%  
 select(date, package, function\_name) %>%  
 mutate(package = as.factor(package) %>% fct\_lump(n = 5, other\_level = "Other")) %>%  
 arrange(date) %>%  
 as\_tbl\_time(index = date) %>%  
 collapse\_by(period = "6 m", clean = TRUE) %>%  
 count(date, package) %>%  
 count\_to\_pct(date) %>%  
 mutate(biannual = paste0("H", semester(date)))

### Xts

Xts objects are a matrix of observations combined with an index of corresponding dates and times.

The main xts constructor takes a number of arguments, but the two most important are x for the data and order.by

Xts also allows you to bind arbitrary key-value attributes to your data. This lets you keep metadata about your object inside your object. To add these at creation, you simply pass additional name = value

At the core of both xts and zoo is a simple R matrix with a few additional attributes. The most important of these attributes is the index.

The index holds all the information we need for xts to treat our data as a time series.

When working with time series, it will sometimes be necessary to separate your time series into its core data and index attributes for additional analysis and manipulation. The core data is the matrix portion of xts. You can separate this from the xts object using coredata().

The index portion of the xts object is available using the index() function.

E.g. hayek <- xts(x = data, order.by = dates, born = bday)

### Apply Functions

**Plot function:**

ts\_plot <- function(data, x, y){

# x <- substitute(x)  
 # y <- substitute(y)

data %>%  
 ggplot(aes(x, y)) +  
 geom\_point(colour = palette\_light()[[1]], size = 2) +  
 geom\_smooth(method = "loess", colour = palette\_light()[[1]], size = 1.1) +  
 theme\_rt() +  
 expand\_limits(y = 0) +  
 scale\_y\_continuous(labels = comma)  
}

ts\_plot(dataset, dataset$PurchaseDate, dataset$SumPurchases)

Can apply functions daily, weekly, monthly, yearly, quarterly.

purch\_weekly <-  
 dataset %>%  
 tq\_transmute(  
 select = SumPurchases,  
 mutate\_fun = apply.weekly,  
 FUN = mean,  
 na.rm = TRUE,  
 col\_rename = "mean\_purchases"  
 )

ts\_plot(purch\_weekly, purch\_weekly$PurchaseDate, purch\_weekly$mean\_purchases)

**Apply Custom Stat function with quantiles:**

probs <- c(0, 0.025, 0.25, 0.5, 0.75, 0.975, 1)

custom\_stat\_fun <- function(x, na.rm = TRUE, ...) {  
 # x = numeric vector  
 # na.rm = boolean, whether or not to remove NA's  
 # ... = additional args passed to quantile  
 c(mean = mean(x, na.rm = na.rm),  
 stdev = sd(x, na.rm = na.rm),  
 quantile(x, na.rm = na.rm, ...))  
}

purch\_weekly\_st <-  
 dataset %>%  
 tq\_transmute(  
 select = SumPurchases,  
 mutate\_fun = apply.weekly,  
 FUN = custom\_stat\_fun,  
 na.rm = TRUE,  
 probs = probs  
 )

Can then plot the trends with IQR to see volatility and trend line as median.

purch\_weekly\_st %>%  
 ggplot(aes(x = Date, y = `50%`)) +  
 # Ribbon  
 geom\_ribbon(aes(ymin = `25%`, ymax = `75%`),  
 color = palette\_light()[[1]], fill = palette\_light()[[1]], alpha = 0.5) +  
 # Points  
 geom\_point(colour = palette\_light()[[1]], size = 2) +  
 geom\_smooth(method = "loess", se = FALSE, colour = palette\_light()[[1]], size = 1.1) +  
 # Aesthetics  
 labs(title = "Median Weekly Purchases", x = "",  
 subtitle = "Range of 1st and 3rd quartile to show volatility",  
 y = "Median Purchases By Week") +  
 expand\_limits(y = 0) +  
 scale\_color\_tq(theme = "dark") +  
 theme\_rt() +  
 theme(legend.position="none") +  
 scale\_y\_continuous(labels = comma)

**Conversion trend for weekly calcs (not rolling):**

custom\_stat\_fun <- function(x, na.rm = TRUE, ...) {  
 c(mean = mean(x, na.rm = na.rm),  
 stdev = sd(x, na.rm = na.rm),  
 quantile(x, na.rm = na.rm, ...))  
}

# Quantile Weekly Conversions

probs <- c(0, 0.025, 0.25, 0.5, 0.75, 0.975, 1)  
 weekly\_pixel\_to\_reg <-  
 dataset %>%  
 filter(Country == 'India') %>%  
 tq\_transmute(  
 select = RegToP9,  
 mutate\_fun = apply.weekly,  
 FUN = custom\_stat\_fun,  
 na.rm = TRUE,  
 probs = probs  
 )

<https://www.coursera.org/learn/practical-time-series-analysis>

### Rolling Functions

Rolling functions across windows.

Moving average allows us to visualize how averages change over time and cut through noise to detect a trend. Also we can vary the sensitivity of the window calculation.

Combining a rolling mean with a rolling standard deviation can help detect regions of abnormal volatility and consolidation. This is the concept behind Bollinger Bands in the financial industry. The bands can be useful in detecting breakouts in trend for many time series, not just financial.

Combining a rolling mean with a rolling standard deviation can help detect regions of abnormal volatility and consolidation. This is the concept behind Bollinger Bands in the financial industry. The bands can be useful in detecting breakouts in trend for many time series, not just financial.

# Rolling stats - make sure to ungroup

custom\_stat\_fun <- function (x, na.rm = TRUE) {

m <- mean(x, na.rm = na.rm)  
 s <- sd(x, na.rm = na.rm)  
 hi <- m + 2\*s  
 lo <- m - 2\*s

ret <- c(mean = m, stdev = s, hi.95 = hi, lo.95 = lo)

return(ret)

}

df\_rollstats <-  
 dataset\_daily %>%  
 ungroup() %>%  
 filter(!(FailedLoginDate == '2018-06-12' & Baseline == 'Baseline')) %>%  
 tq\_mutate(  
 select = Pct,  
 mutate\_fun = rollapply,  
 width = 7,  
 align = "right",  
 by.column = FALSE,  
 FUN = custom\_stat\_fun,  
 na.rm = TRUE  
 )

[http://www.business-science.io/timeseries-analysis/2017/07/02/tidy-timeseries-analysis.html](https://eur01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.business-science.io%2Ftimeseries-analysis%2F2017%2F07%2F02%2Ftidy-timeseries-analysis.html&data=02%7C01%7C%7Cb6641a994c114f7a8f1408d5b70fa702%7C84df9e7fe9f640afb435aaaaaaaaaaaa%7C1%7C0%7C636616200209588479&sdata=G4v0AzYbP69LQudHtUAmOm89NynKfqLzCakoK2I7%2Bps%3D&reserved=0)

## Financial Modelling

### Bollinger Bands

Combining a rolling mean with a rolling standard deviation can help detect regions of abnormal volatility and consolidation. This is the concept behind Bollinger Bands in the financial industry. The bands can be useful in detecting breakouts in trend for many time series, not just financial.

The price of the stock is bracketed by an upper and lower band along with a 21-day simple moving average. STdev is a measure of volatility, when the markets become more volatile, the bands widen; during less volatile periods, the bands contract.

The closer the prices move to the upper band, the more overbought the market, and the closer the prices move to the lower band, the more oversold the market. When stock prices continually touch the upper Bollinger Band®, the prices are thought to be overbought; conversely, when they continually touch the lower band, prices are thought to be oversold, triggering a buy signal.

The squeeze - low volatility when bands come close together. Potential sign of future increased volatility and trading opportunites. Further apart the more likely there will be a decrease in volatility and exiting a trade. Not trading signals though, no indication on when the change may take place or which direction.

Breakouts - approximately 90% of price action occurs between the two bands. Any breakouts above or below is a major event but again doesn't provide a trading signal.

When using Bollinger Bands®, designate the upper and lower bands as price targets. If the price deflects off the lower band and crosses above the 20-day average (the middle line), the upper band comes to represent the upper price target. In a strong uptrend, prices usually fluctuate between the upper band and the 20-day moving average. When that happens, a crossing below the 20-day moving average warns of a trend reversal to the downside.

One of the most useful and commonly used tools in spotlighting extreme short-term prices in a security. Buying when stock prices cross below the lower Bollinger Band® often helps traders take advantage of oversold conditions and profit when the stock price moves back up toward the center moving-average line.

Constriction - when bands move together.

After constriction, bullish expansion is where bands move in opposite directions but prices are closer to upper band and berish expansion is prices closer to lower band.

Bullish Move Up - lower band moves up with upper band; but move won't be as extreme or lengthy as others. So still a good entry but more short term one.

Bearish move down - upper band moving down with lower band, so volatility not expanding.

Ideally we want to see expansion as this means a big move. Once the lower band starts to follow the upper band it's going to slow the move down.

Strategies:

1. Inside Band Trading - buy at the bottom of range near bottom band (not a breakout) and sell around middle.

2. Pull back to the middle - strong up move and then pull back to the middle. Buy in middle and sell at upper band.

3. Surfing the bands - prices should only break out of bands 10% of time.

4. Bollinger band reversals - fake breakouts.

Time Frames:

Short term analysis – for stops.

Middle term – Primary analysis (20 period)

Long term – 20 period bollinger bands on weekly chart for enviromental info.

### Trend lines

Can see the overall trend of a stocks price, can be used to help predict support and resistance.

As the price nears a major support/resistance level, there are two different scenarios that can occur: The price will bounce off the trendline and continue in the direction of the prior trend, or it will move through the trendline, which can then be used as a sign that the current trend is reversing or weakening.

Some traders will only connect closing prices while others may choose to use a mix of close, open and high prices.

The more prices that touch the trendline the stronger and more influential the line is believed to be.

Trend is expected to remain upward until price closes below the trend line.

The more times the price touches the trendline, the more influential the line is said to be. The price action illustrated by the arrow on the far right, would be used by traders as confirmation that the trendline is valid. In this case, traders would look to enter a long position as close to the trendline as possible.

Once a technical trader has entered a position near the trendline, he or she would keep the position open until the price moved below the support of the trendline.

Once a technical trader has entered a position near the trendline, he or she would keep the position open until the price moved below the support of the trendline. Most traders will constantly adjust their stop-loss orders by moving them higher, as the trendline continues to slope upward.

### MACD

Moving Average Convergence Divergence - trend-following momentum indicator, shows the relationship between two moving averages of prices.

Calculated by subtracting the 26-day exponential moving average

Moving averages are in their nature, lagging indicators, whereas EMA puts more weight on the latest data.

Crossovers - when the MACD falls below the signal line it's time to sell. When MACD rises above signal line it's a bullish signal which suggests likelihood of upward momentum. Many wait for confirmed cross to enter position to avoid getting "faked out".

Divergence - When the security price diverges from the MACD, it signals the end of the current trend. For example, a stock price that is rising and a MACD indicator that is falling could mean that the rally is about to end. Conversely, if a stock price is falling and the MACD is rising, it could mean that a bullish reversal could occur in the near-term.

Trading divergence is the most popular.

Dramatic Rise - When the MACD rises dramatically - that is, the shorter moving average pulls away from the longer-term moving average - it is a signal that the security is overbought and will soon return to normal levels.

Also watch for MACD above zero as this signals the short-term average relative to long-term average. When zero, short term average is above the long term average signalling upward momentum. Conversely opposite is true.

library(TTR)

The MACD is a special case of the general oscillator applied to price. The MACD can be used as a general oscillator applied to any series. Time periods for the MACD are often given as 26 and 12, but the original formula used exponential constants of 0.075 and 0.15, which are closer to 25.6667 and 12.3333 periods.

macd <- MACD( ttrc[,"Close"], 12, 26, 9, maType="EMA" )

MACD calculates the percentage difference between the two moving averages by default. If you want to use the raw difference, set percent = FALSE in your MACD call.

### RSI

Relative strength index (RSI) is a momentum indicator that compares the magnitude of recent gains and losses over a specified time period to measure speed and change of price movements of a security.

It is primarily used to attempt to identify overbought or oversold conditions in the trading of an asset.

rsiMA1 <- RSI(price, n=14, maType="WMA", wts=ttrc[,"Volume"])

### Investing Resources

Investing Course -

https://academy.investopedia.com/products/technical-analysis?aca\_ref=inline

Bollinger bands –

https://www.investopedia.com/terms/b/bollingerbands.asp?lgl=rira-baseline-vertical

<https://www.investopedia.com/articles/technical/102201.asp>

Trend lines –

https://www.investopedia.com/articles/trading/06/trendlines.asp

EMA -

https://www.investopedia.com/terms/e/ema.asp

MACD -

https://www.investopedia.com/terms/m/macd.asp

https://www.investopedia.com/articles/forex/05/macddiverge.asp

RSI -

https://www.investopedia.com/terms/r/rsi.asp

### My Stock Analysis

tq\_index("SP500")

apple <- tq\_get("AAPL", get = "stock.prices")

apple %>%  
 arrange(desc(date))

r quandl

<https://www.quandl.com/data/ASX-Australian-Securities-Exchange/usage/quickstart/api>

# For London

ticker <- "VOD.L"

a <- getSymbols(ticker, src="google", from = as.Date("2010-01-01"), to = as.Date("2017-05-16"))

# Stock Info ----

stock <- getSymbols("TLS.AX", from = as.Date("2017-06-01"), to = as.Date("2018-05-15"), auto.assign = FALSE)

df <- as.data.frame(stock)

df <- cbind(date=as.Date(rownames(df)),df)

rownames(df) <- NULL

colnames(df) <- c("Date","Open","High","Low","Close","Volume","Adjusted") # I do this so the data frame is more applicable universally instead of having the ASX code in front of each column name e.g. TLS.AX.Open.

glimpse(df)

# Initial Charts ----

df %>%  
 ggplot(aes(x = Date, y = Close)) +  
 geom\_point(colour = "grey40", alpha = 0.5) +  
 geom\_line(colour = palette\_light()[[1]], size = 1) +  
 geom\_smooth(method = "loess", colour = palette\_light()[[3]]) +  
 theme\_rt() +  
 labs(title = 'Overall: Telstra in a Downard Trend',  
 subtitle = "Despite a few small breaks, the closing prices are consistently falling below the overall trend line.  
In my view, we'll need a few more breakouts before any suggestion of a trend reversal.  
 ")

# Mention key events e.g. ACCC decision etc.

# Rolling Avg Chart ----

df\_rollmean <- df %>%  
 tq\_mutate(  
 # tq\_mutate args  
 select = Close,  
 mutate\_fun = rollapply,  
 # rollapply args  
 width = 28,  
 align = "right",  
 FUN = mean,  
 # mean args  
 na.rm = TRUE,  
 # tq\_mutate args  
 col\_rename = "mean\_28"  
 ) %>%  
 tq\_mutate(  
 # tq\_mutate args  
 select = Close,  
 mutate\_fun = rollapply,  
 # rollapply args  
 width = 84,  
 align = "right",  
 FUN = mean,  
 # mean args  
 na.rm = TRUE,  
 # tq\_mutate args  
 col\_rename = "mean\_84"  
 )

df\_rollmean %>%  
 ggplot(aes(x = Date, y = Close)) +  
 # Data  
 geom\_point(alpha = 0.1) +  
 geom\_line(aes(y = mean\_28), colour = palette\_light()[[1]], size = 1.1) +  
 geom\_line(aes(y = mean\_84), colour = palette\_light()[[2]], size = 1.1) +  
 # Aesthetics  
 labs(title = "Daily Closing Prices: Telstra", x = "",  
 subtitle = "28 and 84 Day Moving Average  
 ") +  
 scale\_color\_tq() +  
 theme\_rt() +  
 theme(legend.position="none")

# Adv Chart ----

custom\_stat\_fun\_2 <- function(x, na.rm = TRUE) {  
 # x = numeric vector  
 # na.rm = boolean, whether or not to remove NA's  
 m <- mean(x, na.rm = na.rm)  
 s <- sd(x, na.rm = na.rm)  
 hi <- m + 2\*s  
 lo <- m - 2\*s  
  
 ret <- c(mean = m, stdev = s, hi.95 = hi, lo.95 = lo)  
 return(ret)  
}

df\_rollstats <- df %>%  
 tq\_mutate(  
 select = Close,  
 mutate\_fun = rollapply,  
 # rollapply args  
 width = 21,  
 align = "right",  
 by.column = FALSE,  
 FUN = custom\_stat\_fun\_2,  
 # FUN args  
 na.rm = TRUE  
 )

df\_rollstats %>%  
 ggplot(aes(x = Date)) +  
 geom\_point(aes(y = Close), colour = "grey40", alpha = 0.6) +  
 geom\_ribbon(aes(ymin = lo.95, ymax = hi.95), colour = palette\_light()[[1]], alpha = 0.3) +  
 geom\_line(aes(y = mean), colour = palette\_light()[[1]], size = 1.2, alpha = 0.9) +  
 theme\_rt() +  
 labs(title = "Telstra: 28 Day Moving Avg & 95% Confidence Interval Bands",  
 subtitle = 'Also known as Bollinger Bands.  
 ',  
 x = "")

# Correlation with ASX 200 Index ----

# One more thing we can do is look at the rolling correlation between the ASX index & Telstra share price to see if this is something we could use to forecast the Telstra share price. Already from a quick chart we can tell this is probably not going to be a good measure, but we'll look further all the same.

# I can't figure out how to download this data from the API, I've tried a bunch of different options e.g. ^AORD from Google finance or ASX codes XJO/VAS.

# For now I've just manually dowmloaded the monthly data from here - https://www.marketindex.com.au/asx200 and then run average by month and run a rolling correlation.

asx200 <- read.csv("asx200.csv")

glimpse(asx200)

asx200$Date <- as.Date(as.character(asx200$Date))

# Merge datasets on same dates (end of months)

comparedf <- merge(asx200, df, by = 'Date')  
comparedf$Volume <- NULL  
comparedf$Adjusted <- NULL

# Replace column names

names(comparedf) <- gsub(x = names(comparedf), pattern = '.x', replacement = 'ASX')  
names(comparedf) <- gsub(x = names(comparedf), pattern = '.y', replacement = 'TLS')

# Moderate correlation between the two, but not very strong.

comparedf %>%  
 select(CloseASX,CloseTLS) %>%  
 correlate()

tlsplot <-  
 comparedf %>%  
 select(Date, CloseASX, CloseTLS) %>%  
 ggplot(aes(x = Date)) +  
 geom\_line(aes(y = log10(CloseTLS)), colour = palette\_light()[[1]], size = 1.1) +  
 theme\_rt() +  
 labs(title = 'TLS Share Price vs ASX Top 200 Index  
 ',  
 y = 'Telstra Share Price (Log10)',  
 x = '')

tlsasx <-  
 comparedf %>%  
 select(Date, CloseASX, CloseTLS) %>%  
 ggplot(aes(x = Date)) +  
 geom\_line(aes(y = log10(CloseASX)), colour = palette\_light()[[2]], size = 1.1) +  
 theme\_rt() +  
 labs(y = 'ASX Top 200 Index (Log10)')

grid.arrange(tlsplot, tlsasx, ncol = 1)

# Running Correlation - can see they were more correlated previously, but became basically un-correlated towards the end of last year. This indicates that some key events occurred which caused this change to the TLS share price.

# This makes sense when you consider the turbulent share price history of Telstra recently - earnings under pressure with NBN, dividend cuts, $10m in fines in March this year. These key events caused the share price to rapidly decrease in a manner that didn't fit with the ASX index as a whole.

rolling\_cor <-  
 comparedf %>%  
 select(Date, CloseASX, CloseTLS) %>%  
 tq\_mutate\_xy(  
 x = CloseASX,  
 y = CloseTLS,  
 mutate\_fun = runCor,  
 n = 3,  
 use = "pairwise.complete.obs",  
 col\_rename = "rolling\_corr"  
 )

rolling\_cor <- rolling\_cor[!is.na(rolling\_cor$rolling\_corr),]

rolling\_cor %>%  
 ggplot(aes(Date, rolling\_corr)) +  
 geom\_point(colour = palette\_light()[[1]], size = 2.5) +  
 geom\_hline(yintercept = 0.623, colour = palette\_light()[[2]]) +  
 theme\_rt() +  
 labs(title = 'ASX vs TLS: Rolling 3 Month Correlation',  
 subtitle = 'Dynamic vs Static Correlation (Red Line)  
 ',  
 y = 'Correlation')

# Massive thanks to Matt Dancho (Business Science) whose series on time series analysis paved the way for me.

r crypto

### Quandl & Quantmod

install.packages("Quandl")

library(Quandl)

Quandl.api\_key("azowxXrXky6AfvxZBbJz")

ASX/{CODE}{MONTH}{YEAR}  
{CODE} is the futures ticker code, as listed given in the CSV files provided below  
{MONTH} is the single-letter month code  
{YEAR} is a 4-digit year

Quandl:

https://blog.quandl.com/using-quandl-in-r

https://www.analyticsvidhya.com/blog/2017/09/comparative-stock-analysis/

https://chrisconlan.com/download-historical-stock-data-google-r-python/

Quantmod:

https://ntguardian.wordpress.com/2017/03/27/introduction-stock-market-data-r-1/

https://www.quantmod.com/examples/charting/

## Outliers

Possible remove most extreme 2 percentile of results.

Sometimes cap outlier values:

attrition %>%  
 mutate(  
 SumPurchases = case\_when(  
 SumPurchases > 5000 ~ 5000,  
 TRUE ~ SumPurchases  
 )  
 )

### “outliers” Package

outliernetwin <- dataset[scores(dataset$NetWin, type = "z", prob = 0.98),]  
outlierpurchases <- dataset[scores(dataset$SumPurchases, type = "z", prob = 0.999),]

dataset %>%   
 filter(!(UserID %in% outliernetwin$UserID),  
 !(UserID %in% outlierpurchases$UserID))

**outlier** (x, opposite = FALSE)  
Outliers gets the extreme most observation from the mean. If you set the argument opposite=TRUE, it fetches from the other side. If argument is a dataframe, then outlier is calculated for each column by sapply. The same behavior is applied by apply when the matrix is given.

**scores** (x, type="z", prob=0.95) # beyond 95th %ile based on z-scoresThere are two aspects the the scores() function.Compute the normalised scores based on “z”, “t”, “chisq” etc. Z score is mean/stdev; so essentially it represents the distance between the score and population mean in units of standard deviation.Find out observations that lie beyond a given percentile based on a given score  
scores(x, type="z", prob=0.95) # beyond 95th %ile based on z-scores

### Script

Turkey’s method which uses the interquartile approach and removes outliers outside 1.5 IQR.

outlierKD <- function(dt, var) {

var\_name <- eval(substitute(var),eval(dt))

tot <- sum(!is.na(var\_name))

na1 <- sum(is.na(var\_name))

m1 <- mean(var\_name, na.rm = T)

par(mfrow=c(2, 2), oma=c(0,0,3,0))

boxplot(var\_name, main="With outliers")

hist(var\_name, main="With outliers", xlab=NA, ylab=NA)

outlier <- boxplot.stats(var\_name)$out

mo <- mean(outlier)

var\_name <- ifelse(var\_name %in% outlier, NA, var\_name)

boxplot(var\_name, main="Without outliers")

hist(var\_name, main="Without outliers", xlab=NA, ylab=NA)

title("Outlier Check", outer=TRUE)

na2 <- sum(is.na(var\_name))

message("Outliers identified: ", na2 - na1, " from ", tot, " observations")

message("Proportion (%) of outliers: ", (na2 - na1) / tot\*100)

message("Mean of the outliers: ", mo)

m2 <- mean(var\_name, na.rm = T)

message("Mean without removing outliers: ", m1)

message("Mean if we remove outliers: ", m2)

response <- readline(prompt="Do you want to remove outliers and to replace with NA? [yes/no]: ")

if(response == "y" | response == "yes"){

dt[as.character(substitute(var))] <- invisible(var\_name)

assign(as.character(as.list(match.call())$dt), dt, envir = .GlobalEnv)

message("Outliers successfully removed", "\n")

return(invisible(dt))

} else{

message("Nothing changed", "\n")

return(invisible(var\_name))

}

}

## AB Test

<https://blog.hubspot.com/marketing/how-to-do-a-b-testing>

<https://blog.hubspot.com/marketing/marketers-guide-understanding-statistical-significance>

<https://www.r-bloggers.com/hey-i-just-did-a-significance-test/>

<http://rstatistics.net/statistical-tests-with-r/>

# **R: Exploratory Analysis & Visualization**

<https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/>



## Basics

Principals of Analytic Graphics:  
1. Show comparisons – if something is good, compared to what? (e.g. control group)  
2. Show causality, mechanism, explanation. (e.g. why?)  
3. Multivariate data to give richer story. Be careful of Simpson’s paradox “trend that appears in different groups of data disappears when these groups are combined”. Remember facets.  
4. Integration of multiple modes of evidence  
5. Describe and document evidence with appropriate labels, scales, souces etc.  
6. Content is king

Exploratory graphs:  
- Colour/size are primarily used for information.  
- Always want some underlying questions to explore.  
- meant to be quick and dirty

Three plotting systems: base model (constructed piecemeal through series of function calls), lattice (specified by one functon) and ggplot2 (mixes elements of base and lattice).

Quantile(dataset$column) gives you the different quartiles.

# Cut data into quartiles, use these to cut the dataset into the various cutpoints, then facet plot by these.  
cutpoints <- quantile(diamonds$carat,seq(0,1,length=4),na.rm=TRUE)  
diamonds$car2 <- cut(diamonds$carat, cutpoints)  
g + geom\_point(alpha = 1/3) + facet\_grid(cut ~ car2)

Vector formats are good for line drawings and plots with solid colors using a modest number of points, while bitmap formats are good for plots with a large number of points, natural scenes or web-based plots.

Do univariate and bivariate inspections of your data in scatterplots to get an idea of distribution and any outliers with ggplot.

## Presenting Tables

Library(knitr)

Kable(df)

kable(temp,"html") %>% kable\_styling("striped",full\_width=T) %>% column\_spec(1:2,bold=T,background="white") %>% row\_spec(c(1,3,4,5,6,7,8,10,11,16,17,18,20),bold=F,color="white",background="#ffb6c1")



library(gridExtra)  
grid.table(valuesummary)

Grid arrange tables:

t1 <- ttheme\_default()  
tt2 <- ttheme\_minimal()  
tt3 <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = blues9[1:4], col=NA),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="navyblue", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))

b <- grid.text("PIXEL TO REG", y = unit(1, "npc"), x = unit(0.5, "npc"), vjust = 1, hjust = .5, gp = gpar(fontfamily = "Impact", col = "#A9A8A7", cex = 8, alpha = 0.2))

grid.arrange(  
 top = b,  
 tableGrob(summarytable, theme=tt3),  
 tableGrob(countrytable, theme=tt3),  
 nrow = 1  
)

# To save the files:  
png("test.png", height = 50\*nrow(df), width = 200\*ncol(df))  
Table  
dev.off()

**My favourite:**

tt3 <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = blues9[1:2], col="black"),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="navyblue", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))

grid.table(cov\_table\_a, rows = NULL)  
tableGrob(summarytable, theme=tt3, rows = NULL)

tt4 <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = c(blues9[1],blues9[1],blues9[1],blues9[1],blues9[1],blues9[2],blues9[2],blues9[2],blues9[2],blues9[2]), col="black"),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="navyblue", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))

**Tables with ggplot2**

One idea I used was matching the table colours with the bar chart fill colours.

ttprov <- ttheme\_minimal(core=list(bg\_params = list(fill = c(palette\_light()[[3]], palette\_light()[[3]], palette\_light()[[2]], palette\_light()[[2]]), alpha = 0.2, col="black"),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="navyblue", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))

tbl <- tableGrob(SummaryTable, rows=NULL, theme= ttprov)  
# Plot chart and table into one object  
grid.arrange(plt, tbl,  
 nrow=2,  
 as.table=TRUE,  
 heights=c(3,1))

<https://www.r-bloggers.com/plotting-tables-alsongside-charts-in-r/>

<https://stackoverflow.com/questions/16680063/how-can-i-add-a-table-to-my-ggplot2-output>

<https://cran.r-project.org/web/packages/gridExtra/vignettes/tableGrob.html>

<https://stackoverflow.com/questions/31796219/grid-table-and-tablegrob-in-gridextra-package>

<https://stackoverflow.com/questions/18414001/gridextra-colour-different-rows-with-tablegrob>

## Base Plotting System

barplot(yearlyemissions$Emissions, names.arg = yearlyemissions$year, col = 4,  
 main = 'Total Emissions per Year', xlab = 'Year', ylab = 'Total PM2.5 Emissions')

Histogram:   
- hist(dataset$column)  
- rug(dataset$variable) puts the point underneath the axis  
- breaks controls how many bars  
- abline(v = median(dataset$column, lwd = 4)

Scatterplot: with(dataset plot(x, y, col = “red”, pch = 2))

Boxplot:   
- boxplot(x ~ y, dataset)  
- abline(h = 12) can set a line in the boxplot to visualize average or similar.

Pch is the plotting symbol (default is open circle); lty is the line type; col is the plotting colour.

Par() can be used to specify global parameters that affect all plots in R session. E.g. bg is background colour. par(mfrow = c(1, 2), mar = c(5, 4, 2, 1)) #mfrow first argument is rows then columns

Title(main=””)

mtext("Ozone and Weather in New York City", outer = TRUE) # Main title

pdf(file="myplot.pdf")  
dev.off()

boxplot(Ozone ~ Month, airquality, xlab = "Month", ylab = "Ozone (ppb)", col.axis = "blue", col.lab = "red")

legend("topright", pch = c(17,8), col = c("blue", "red"), legend = c("May", "Other Months"))

axis(2, axTicks(2), format(axTicks(2), scientific = F))

## Lattice Plotting System

Good for visualizing multiple variables across levels of a factor variable.

xyplot(Life.Exp ~ Income | region, data = state, layout = c(4,1)) – e.g. 4 is the number of columns and 1 is rows.

## GGPlot2

Grammar of graphics.

Ultimate list: <http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html>

<https://www.r-graph-gallery.com/portfolio/ggplot2-package/>

<http://www.sthda.com/english/articles/32-r-graphics-essentials/133-plot-one-variable-frequency-graph-density-distribution-and-more/>

Save your plots - ggsave(file = paste(PromoName, ".png",sep=""))

**Go-To Template:**theme\_tq() +  
 theme(#panel.border = element\_blank(),  
 plot.title = element\_text(colour = "#666666", size = 12, face = "bold"),  
 plot.subtitle = element\_text(colour = "#666666", size = 10),  
 axis.title = element\_text(colour = "#666666"),  
 axis.title.x = element\_text(hjust = 0.5),  
 axis.title.y = element\_text(hjust = 0.5),  
 axis.text = element\_text(colour = "#666666"),  
 axis.ticks = element\_blank(),  
 plot.caption = element\_text(colour = "darkgrey", hjust = 0, size = 8),  
 legend.position = "top",  
 legend.justification = 0)

**Lately been using TQ colours**:  
scale\_fill\_manual(values = c(palette\_light()[[3]], palette\_light()[[2]])

My Template #2:  
ggplot(dataset, aes(x = RegulatedMarketAccountStatusDescription, fill = factor(CutPurchases, labels = c("1-4","5-19","20-172")))) +  
 geom\_histogram(stat = "count") +  
 labs(title = 'Experian Manual Follow Up Results: 1st Oct - 10th Oct',  
 fill = 'Purchase Segment:',  
 x = '',  
 caption = ‘') +  
 scale\_y\_continuous(breaks = seq(0, 100, 5)) +  
 theme\_economist() +  
 theme(plot.caption = element\_text(hjust = 0.5)) +  
 scale\_fill\_tableau()

**My template #3 (simple with annotations and certain points emphasized):**  
ggplot() +  
 geom\_line(data = datasetval, aes(x= Week, y = AvgDeposits), size= 1, colour = "steelblue1") +  
 geom\_point(data = filter(datasetval, Week == 1 | Week == 2 | Week == 3 | Week == 4), aes(x = Week, y = AvgDeposits), colour = "navyblue", size = 2) +  
 geom\_text(data = filter(datasetval, Week == 1 | Week == 2 | Week == 3 | Week == 4),   
 aes(x = Week, y = AvgDeposits, label= format(round(AvgDeposits)), hjust = 1.2, vjust = -0.6 ),  
 colour = "navyblue") +  
 scale\_x\_reverse(breaks = seq(0,12,1)) +  
 scale\_y\_continuous(labels = comma) +  
 theme\_tufte(ticks = FALSE) +  
 labs(title = "Average of Total Purchases Prior to Trigger (%)",  
 subtitle = 'Dramatic Increase at Week 1',  
 x = "Week",  
 y = "Avg Deposits",  
 caption = "  
 Note: Average taken from players who purchased in the 12 weeks prior to Exclusion trigger event") +  
 annotate("text", size = 3.5, x = 8.5, y = 22, colour = "#666666", label = "Average deposits are fairly low and stable \nuntil they begin increasing around Week 4 \nand then more rapidly around Week 2 & 1") +  
 theme(plot.title = element\_text(face = "bold", colour = "#666666"),  
 plot.subtitle = element\_text(colour = "#666666", size = 11),  
 axis.title = element\_text(colour = "#666666"),  
 axis.text = element\_text(colour = "darkgrey"),  
 plot.caption = element\_text(colour = "darkgrey", hjust = 0, size = 8))

Add extra stats around charts:  
library(ggExtra)  
ggMarginal(a, type = "histogram", fill = "lightblue", colour = "black", margins = "x", “y”, “both”)  
ggMarginal(a, type = "boxplot", fill = "lightblue", colour = "black")  
ggMarginal(a, type = "density", fill = "lightblue", colour = "black")

facet\_grid(drv ~ cyl, margins = TRUE) # margins gives totals over each row  
Change the order of categories by changing the actual factor variable, e.g. factor(levels, labels)

facet\_wrap(~variable, scales = “free”, ncol = 3)

### Multiple Variables at Once

quanti <- select\_if(dataset, is.numeric)  
quanti <- cbind.data.frame(quanti, DEFAULT = dataset$DEFAULT)

quali <- select\_if(dataset, is.factor)

# df.melt = melt(quanti, id.vars = 'DEFAULT')

tidydata <- gather(quanti, variable, value, -DEFAULT)

ggplot(tidydata, aes(x = value)) +  
 geom\_density(aes(fill = DEFAULT)) +  
 facet\_grid(.~ variable, scales = "free")

### Scatterplot, Line & Smooth

Geom\_point(colour = “black”, fill = “”, pch = 21)

qplot(displ, hwy, data = mpg, geom=c("point", "smooth"),facets=.~drv)

ggplot() +  
 geom\_point(data = training, aes(x=age,y=wage,colour=jobclass))

Scatterplots with geom\_point(alpha=0.5, position = “jitter”) & sometime confidence ellipse - data.ellipse(x, y, levels=c(0.5, 0.975))

geom\_smooth**:** Aids the eye in seeing patterns in the presence of overplotting. When there’s lots going on it allows you to see trends.

ggplot(data=dat1, aes(x=time, y=total\_bill, group=sex, colour=sex)) +  
 geom\_line() +  
 geom\_point()

Simple way to join geom\_line colours into one line - geom\_path(aes(group = 1))

# Plot with regression smoothers  
ggplot(data = training, aes(x=age, y=wage,colour=education)) +  
 geom\_point() +  
 geom\_smooth(method='lm',formula=y~x, se = TRUE, alpha = 0.2)   
OR y~ splines::bs(x, 3)

# Logistic Smoother  
ggplot(training, aes(x = Cashins, y = as.numeric(DEPOSITING) - 1)) +  
 geom\_jitter() +  
 geom\_smooth(formula = y ~ splines::ns(x, 5), method = "glm", method.args = list(family = "binomial"))

Other methods: lm, glm, loess, gam

### Bar

Either show frequency distributions or raw counts (stat = “identity”).

Can make bar charts instead of histogram with aes( x & y ) + stat = “identity”. Can then make diverging bar, lollipop or dot charts. Can also make ordered bar charts.

ggplot(data=dat, aes(x=time, y=total\_bill, fill=time)) +   
 geom\_bar(colour="black", fill="#DD8888", width=.8, stat="identity") +   
 guides(fill=FALSE) +  
 xlab("Time of day") + ylab("Total bill") + ggtitle("Average bill for 2 people")

Horizontal bar chart – coord\_flip()

Re-order bar char charts with scale\_x\_discrete(limits = c(“”,””,””)

Can also re-order at factor level or use re-order, e.g.:  
p2 <- ggplot(df, aes(x = reorder(Category, -Count), y = Count)) +  
 geom\_bar(stat = "identity")

Can add captions to geom\_bar:  
geom\_text(aes(label=round(Emissions,0), size=1, hjust=0.5, vjust=-0.5)) +  
theme(legend.position='none')

**Matt Dancho’s bar with annotations & trend line**

ggplot(aes(date, pct)) +  
 geom\_bar(stat = "identity", fill = palette\_light()[[1]], color = "white") +

geom\_text(aes(x = date, y = pct, label = biannual),   
 vjust = -1, color = palette\_light()[[1]], size = 3) +  
 geom\_text(aes(x = date, y = pct, label = scales::percent(pct)),   
 vjust = 2, color = "white", size = 3) +  
 geom\_smooth(method = "lm", se = FALSE) +  
 scale\_y\_continuous(labels = scales::percent) +  
 scale\_fill\_tq() +  
 theme\_tq() +  
 labs(  
 title = 'How "Tidy" Is DRobs Code?',  
 subtitle = "Variance Explained Blog",  
 x = "Date (Bi-Annual Aggregation)", y = "% of Total R Functions [n / sum(n)]"  
 ) +  
 expand\_limits(y = 1)

**Diverging:**

mtcars <- mtcars[order(mtcars$mpg\_z), ] # sort  
mtcars$`car name` <- factor(mtcars$`car name`, levels = mtcars$`car name`)

ggplot(mtcars, aes(x=`car name`, y=mpg\_z, label=mpg\_z)) +   
 geom\_bar(stat='identity', aes(fill=mpg\_type), width=.5) +  
 scale\_fill\_manual(name="Mileage",   
 labels = c("Above Average", "Below Average"),   
 values = c("above"="#00ba38", "below"="#f8766d")) +   
 labs(subtitle="Normalised mileage from 'mtcars'",   
 title= "Diverging Bars") +   
 coord\_flip()

**My Diverging:**

ggplot(filter(netwinsegset, Cluster == 'High'),   
 aes(x = NetWinSegment,   
 y = ifelse(ProfitLoss == "profit", Percentage, -Percentage),  
 fill = ProfitLoss  
 )) +  
 geom\_bar(stat = "identity") +  
 facet\_grid(.~ABTest) +  
 coord\_flip(ylim = c(-25,50)) +  
 scale\_fill\_manual(values = c(palette\_light()[[2]],"chartreuse4")) +  
 scale\_y\_continuous(breaks = seq(-50,50,25)) +  
 theme\_tq() +  
 geom\_hline(yintercept = 0, color = "black", size = 1, linetype = 1) +  
 geom\_hline(yintercept = -25, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 geom\_hline(yintercept = 25, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 guides(fill = FALSE) +  
 labs(title = 'High',  
 x = 'Net Win Bucket',  
 y = 'Percentage Of Players')

<https://stackoverflow.com/questions/5208679/order-bars-in-ggplot2-bar-graph>

### Lollipop Chart

Good alternative to bar charts.

ggplot(data, aes(x=x, y=y)) +  
 geom\_segment( aes(x=x, xend=x, y=0, yend=y), color="grey") +  
 geom\_point( color="orange", size=4)

Change baseline by changing the y value.

Conditional colouring – create two types first:  
data=data %>% mutate(mycolor = ifelse(y>0, "type1", "type2"))  
geom\_segment( aes(x=x, xend=x, y=0, yend=y, color=mycolor), size=1.3, alpha=0.9) +

Conditional colouring – with ifelse:  
geom\_segment( aes(x=x, xend=x, y=0, yend=y ), color=ifelse(data$x %in% c("A","D"), "orange", "grey"), size=ifelse(data$x %in% c("A","D"), 1.3, 0.7) ) +  
geom\_point( color=ifelse(data$x %in% c("A","D"), "orange", "grey"), size=ifelse(data$x %in% c("A","D"), 5, 2) )

**Diverging:**

mtcars <- mtcars[order(mtcars$mpg\_z), ] # sort  
mtcars$`car name` <- factor(mtcars$`car name`, levels = mtcars$`car name`)

ggplot(mtcars, aes(x=`car name`, y=mpg\_z, label=mpg\_z)) +   
 geom\_point(stat='identity', fill="black", size=6) +  
 geom\_segment(aes(y = 0,   
 x = `car name`,   
 yend = mpg\_z,   
 xend = `car name`),   
 color = "black") +  
 geom\_text(color="white", size=2) +  
 labs(title="Diverging Lollipop Chart",   
 subtitle="Normalized mileage from 'mtcars': Lollipop") +   
 ylim(-2.5, 2.5) +  
 coord\_flip()

**My Example 1:**

ggplot(filter(netwinsegset, Cluster == 'High'),   
 aes(x = NetWinSegment,   
 y = ifelse(ProfitLoss == "profit", Percentage, -Percentage),  
 fill = ProfitLoss  
 )) +  
 geom\_segment(aes(y = 0, yend = ifelse(ProfitLoss == "profit", Percentage, -Percentage), x = NetWinSegment, xend = NetWinSegment)) +  
 geom\_point() +  
 facet\_grid(.~ABTest) +  
 coord\_flip(ylim = c(-25,50)) +  
 scale\_fill\_manual(values = c(palette\_light()[[2]],"chartreuse4")) +  
 scale\_y\_continuous(breaks = seq(-50,50,25)) +  
 theme\_tq()

### Population Pyramid

Horizontal bar chart, good way to compare two factors side by side (e.g. Gender) in segments.

brks <- seq(-15000000, 15000000, 5000000)  
lbls = paste0(as.character(c(seq(15, 0, -5), seq(5, 15, 5))), "m")

ggplot(email\_campaign\_funnel, aes(x = Stage, y = Users, fill = Gender)) + # Fill column  
 geom\_bar(stat = "identity", width = .6) + # draw the bars  
 scale\_y\_continuous(breaks = brks, labels = lbls) + # Labels  
 coord\_flip()

My Example

ggplot(demotbl2, aes(x = AgeSegment, fill = Gender,  
 y = ifelse(test = Gender == "M",   
 yes = -Percentage, no = Percentage))) +  
 geom\_bar(stat = "identity", width =.6, alpha = 0.8) +  
 scale\_y\_continuous(labels = abs, limits = max(demotbl2$Percentage) \* c(-1,1)) +  
 coord\_flip() +  
 scale\_fill\_manual(values = c("dodgerblue4","firebrick")) +  
 geom\_hline(yintercept = 0, colour = "black", size = 1.2)Other Plots

### Boxplot

ggplot(data = fin, aes(x = Industry, y = Growth, colour = Industry)) +  
 geom\_jitter() +  
 geom\_boxplot(size = 1, alpha = 0.9, outlier.colour = NA) # as outliers are duplicated by jitter.

Can cut variable and create boxplot (breaks refers to intervals, so 6 breaks = 7 boxplots)  
ggplot(data = dataset,   
 aes(x = cut(xnvarvame, breaks = 6), y = yvarname)) +   
 geom\_boxplot()

boxplot.stats(Mileage[Brands=="CEAT"])

Interpreting boxplots - can see the means; narrower boxes mean less variance. Wider distributions = more volatility and risk.



So if it’s more than 1.5x inter quartile range it won’t be included.

### Histogram & Density Plots

Histograms must have continuous numerical variable, can’t be a factor. If it’s factor would instead use geom\_bar.

geom\_histogram() for continuous variables, if it’s factor would instead use geom\_bar. Only x value.

Histogram:  
s <- ggplot(data=movies, aes(x = BudgetMillions))  
s + geom\_histogram(binwidth = 20, aes(fill=Genre), colour = "Black")

# Density plots for continuous variables   
ggplot(data = training, aes(x=wage,colour=education)) +  
 geom\_density()

# Histogram with density plot

a + geom\_histogram(aes(y = ..density.., color = sex),   
 fill = "white",  
 position = "identity")+  
 geom\_density(aes(color = sex), size = 1) +  
 scale\_color\_manual(values = c("#868686FF", "#EFC000FF"))

### Violin Plot

A blend of boxplots and density plots.

X is a factor and Y is continuous.

geom\_violin(trim=FALSE)+  
 geom\_boxplot(width=0.1, fill="white")

-- My Template

ggplot(data = datasetsub, aes(x = CHURNED, y = CountPlayerNetWinRatio)) +  
 geom\_violin(aes(fill = CHURNED), alpha = 0.8) +  
 geom\_boxplot(width = 0.3, fill = "white", alpha = 0.5) +  
 theme\_tq() +  
 theme(panel.border = element\_blank(),  
 plot.title = element\_text(colour = "#666666", size = 12, face = "bold"),  
 plot.subtitle = element\_text(colour = "#666666", size = 10),  
 axis.title = element\_text(colour = "#666666"),  
 axis.title.x = element\_text(hjust = 0.5),  
 axis.title.y = element\_text(hjust = 0.5),  
 axis.text = element\_text(colour = "#666666"),  
 axis.ticks = element\_blank(),  
 plot.caption = element\_text(colour = "darkgrey", hjust = 0, size = 8),  
 legend.position = "top",  
 legend.justification = 0) +  
 coord\_cartesian(ylim = c(0, 150)) +  
 scale\_fill\_manual(values = c(palette\_light()[[1]], "firebrick")) +  
 guides(fill = FALSE) +  
 labs(title = 'Player NetWin Sessions Ratio & Churn',  
 subtitle = 'Correlation between smaller NetWin ratio and churning',  
 x = 'Churn',  
 y = 'NetWin Ratio')

<http://www.sthda.com/english/wiki/ggplot2-violin-plot-quick-start-guide-r-software-and-data-visualization#add-summary-statistics-on-a-violin-plot>

### Other Plots

Mosaic plot - <https://cran.r-project.org/web/packages/ggmosaic/vignettes/ggmosaic.html>

Candlestick - <https://cran.r-project.org/web/packages/tidyquant/vignettes/TQ04-charting-with-tidyquant.html#candlestick-chart>

Missing Map:  
library(Amelia)  
missmap(data.raw,legend = TRUE,y.cex = 0.1, x.cex = 0.5)

DotPlot:  
a + geom\_dotplot(aes(fill = sex), binwidth = 1/4) +  
 scale\_fill\_manual(values = c("#00AFBB", "#E7B800"))

Ridges Density:

ggplot(iris, aes(x = Sepal.Length, y = Species)) +  
 geom\_density\_ridges(scale = 0.9)

ggplot(  
 lincoln\_weather,   
 aes(x = `Mean Temperature [F]`, y = `Month`)  
 ) +  
 geom\_density\_ridges\_gradient(  
 aes(fill = ..x..), scale = 3, size = 0.3  
 ) +  
 scale\_fill\_gradientn(  
 colours = c("#0D0887FF", "#CC4678FF", "#F0F921FF"),  
 name = "Temp. [F]"  
 )

### Examples of My Plots

**Count Deposit Ratio Plot**

dataset %>%  
 filter(SumPurchases > 0) %>%  
 mutate(  
 Purchase\_1 = case\_when(  
 CountPurchases >= 1 ~ 1,  
 TRUE ~ 0  
 ),  
 Purchase\_2 = case\_when(  
 CountPurchases >= 2 ~ 1,  
 TRUE ~ 0  
 ),  
 Purchase\_3 = case\_when(  
 CountPurchases >= 3 ~ 1,  
 TRUE ~ 0  
 ),  
 Purchase\_4 = case\_when(  
 CountPurchases >= 4 ~ 1,  
 TRUE ~ 0  
 ),  
 Purchase\_5 = case\_when(  
 CountPurchases >= 5 ~ 1,  
 TRUE ~ 0  
 )  
 ) %>%  
 group\_by(Description) %>%  
 summarise(  
 Purchase\_1 = sum(Purchase\_1),  
 Purchase\_2 = sum(Purchase\_2),  
 Purchase\_3 = sum(Purchase\_3),  
 Purchase\_4 = sum(Purchase\_4),  
 Purchase\_5 = sum(Purchase\_5)  
 ) %>%  
 mutate(  
 Purchase\_Perc\_1 = 100 \* ( Purchase\_1 / Purchase\_1 ),  
 Purchase\_Perc\_2 = 100 \* ( Purchase\_2 / Purchase\_1 ),  
 Purchase\_Perc\_3 = 100 \* ( Purchase\_3 / Purchase\_2 ),  
 Purchase\_Perc\_4 = 100 \* ( Purchase\_4 / Purchase\_3 ),  
 Purchase\_Perc\_5 = 100 \* ( Purchase\_5 / Purchase\_4 )  
 ) %>%  
 select(-Purchase\_1:-Purchase\_5) %>%  
 gather(key = Purchase, value = Percent, -Description) %>%  
 # mutate(Purchase = str\_split(Purchase, n = 1))  
 mutate(Purchase = as.numeric(str\_sub(Purchase, -1, -1))) %>%  
 ggplot(aes(x = Purchase, y = Percent, colour = Description)) +  
 geom\_point(size = 3, alpha = 0.8) +  
 geom\_line(size = 1.2) +  
 geom\_text(aes(label = paste0(round(Percent),'%')), vjust = -1.5, hjust = -0) +  
 expand\_limits(y = 103) +  
 scale\_colour\_manual(values = c(palette\_light()[[2]], palette\_light()[[3]])) +  
 theme\_rt() +  
 labs(  
 title = 'Slightly better deposit ratios on Indians going through to agent'  
 )

### Ablines & Edits

Good for emphasising points.

geom\_hline(yintercept = 50, colour = "red", linetype = "dotted", size = 1.2)

geom\_vline(xintercept = median(dataset$Age), colour = "red", linetype = "dotted", size = 1.3)

abline(a=0, b= 1, lty = 5, col = 4)  
abline(v = 0.5, lty = 3)

geom\_rect(aes(xmin = -Inf, xmax = Inf, ymin = 50, ymax = 100), fill = "firebrick", alpha = 0.005)

Vline with date axis:  
geom\_vline(xintercept = as.numeric(as.Date("2018-06-13")), colour = "red", linetype = "dotted", size = 1.3)

Vline with date time axis:  
geom\_vline(xintercept = as.numeric(as\_datetime("2018-06-21 17:00:00")), colour = "red", linetype = "dotted", size = 1.3)

# Good example with a few on a line chart:

ggplot(data = xgpay0, aes(x = PAY\_0, y = Freq)) +  
 geom\_line(size = 1.3, colour = "navyblue") +  
 geom\_point(data = filter(xgpay0, Freq > 50), size = 2, colour = "red") +  
 geom\_hline(yintercept = 50, colour = "red", linetype = "dotted", size = 0.9) +  
 geom\_rect(aes(xmin = -Inf, xmax = Inf, ymin = 50, ymax = 100), fill = "firebrick", alpha = 0.005) +  
 scale\_x\_continuous(breaks = seq(-2,8,1)) +  
 scale\_y\_continuous(breaks = seq(0,100,25)) +  
 coord\_cartesian(ylim = c(0,100))

# Good examples with my **country pixel plots** which I then threw in an infographic:

summaryplot<-  
 ggplot(summary, aes(x = DateTimeHour, y = PixelToReg)) +  
 geom\_line(size = 1.2, colour = "navyblue") +  
 coord\_cartesian(ylim = c(40,75), xlim = c(min(summary$DateTimeHour),max(summary$DateTimeHour))) +  
 scale\_x\_datetime(breaks = date\_breaks("2 hour")) +  
 geom\_hline(yintercept = 50, colour = "red", linetype = "dotted", size = 1.2) +  
 geom\_hline(yintercept = 60, colour = "limegreen", linetype = "dotted", size = 1.2) +  
 geom\_rect(aes(xmin = min(summary$DateTimeHour), xmax = max(summary$DateTimeHour), ymin = 0, ymax = 50), fill = "firebrick", alpha = 0.01) +  
 geom\_rect(aes(xmin = min(summary$DateTimeHour), xmax = max(summary$DateTimeHour), ymin = 50, ymax = 60), fill = "#666666", alpha = 0.01) +  
 geom\_rect(aes(xmin = min(summary$DateTimeHour), xmax = max(summary$DateTimeHour), ymin = 60, ymax = 100), fill = "limegreen", alpha = 0.01) +  
 theme\_tq()

countryplot <-   
 ggplot(filter(canindsummary, !PixelToReg == Inf),   
 aes(x = DateTimeHour, y = PixelToReg)) +  
 geom\_line(size = 1.2, colour = "navyblue") +  
 facet\_grid(Country ~ .) +  
 coord\_cartesian(ylim = c(min(canindsummary$PixelToReg),max(canindsummary$PixelToReg)), xlim = c(min(summary$DateTimeHour),max(summary$DateTimeHour))) +  
 scale\_x\_datetime(breaks = date\_breaks("2 hour")) +  
 geom\_hline(yintercept = 50, colour = "red", linetype = "dotted", size = 1.2) +  
 geom\_hline(yintercept = 60, colour = "limegreen", linetype = "dotted", size = 1.2) +  
 geom\_rect(aes(xmin = min(summary$DateTimeHour), xmax = max(summary$DateTimeHour), ymin = 0, ymax = 50), fill = "firebrick", alpha = 0.01) +  
 geom\_rect(aes(xmin = min(summary$DateTimeHour), xmax = max(summary$DateTimeHour), ymin = 50, ymax = 60), fill = "#666666", alpha = 0.01) +  
 geom\_rect(aes(xmin = min(summary$DateTimeHour), xmax = max(summary$DateTimeHour), ymin = 60, ymax = 100), fill = "limegreen", alpha = 0.01) +  
 theme\_tq()

# Example for diverging charts:

# Vertical lines  
 geom\_vline(xintercept = 0, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 geom\_vline(xintercept = -0.25, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 geom\_vline(xintercept = 0.25, color = palette\_light()[[5]], size = 1, linetype = 2) +

### Colours

**Lately been using TQ colours**:  
scale\_fill\_manual(values = c(palette\_light()[[3]], palette\_light()[[2]])

TQ Colours:

[[1]] = dark navy blue

[[2]] = red

[[3]] = dark green

[[4]] = beige

[[5]] = pale blue

[[6]] = dark blue

[[7]] = light green

[[8]] = pink

[[9]] = orange

"light", "dark", or "green"

Colours: #FF9999 (salmon), steelblue1, firebrick, olivedrab, dodgerblue4, maroon, #CC79A7 (pinkish colour), darkorchid1, chartreuse4 (military green), darkseagreen4, darkseagreen, aquamarine4

Diverging: values = c(palette\_light()[[2]],"chartreuse4")

Shades: azure(1-4); lightblue

fill = palette\_light()[[1]]

scale\_fill\_brewer(palette = "Dark2)

-- Nice colours from Kaggle:  
scale\_fill\_manual(values = c("#386cb0","#ef3b2c", "#fdb462"))

scale\_fill\_tq(..., theme = "light", "dark", "green")

Colors()

Change background colours:  
theme(plot.background = element\_rect(fill = 'grey'))  
theme(panel.background = element\_rect(fill = 'grey75'))

Categorical variable colours can be changed with built in palette:   
scale\_color\_brewer()  
scale\_color\_manual(breaks = c(“”), values = c("darkgrey","grey","darkblue","grey"), labels = c(“))

Mapping continuous variables to gradient colour:  
scale\_color\_gradient()  
scale\_color\_gradient(low="darkkhaki", high="darkgreen")



Change background colours:  
theme(plot.background = element\_rect(fill = 'grey'))  
theme(panel.background = element\_rect(fill = 'grey75'))

Good colour palette -  
<https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf>

### Annotations & Axis

Two-line label using “\n”

Custom annotation for different fonts in title  
title <- paste(“….”)  
subhead <- paste(“…”\_=)  
ggplot + annotation\_custom(grob=textGrob(title, just="left",   
 gp=gpar(fontsize=10, fontface="bold")),  
 xmin=9.8, xmax=9.8, ymin=11.7) +  
 annotation\_custom(grob=textGrob(subheader, just="left",   
 gp=gpar(fontsize=8, lineheight=1)),  
 xmin=9.8, xmax=9.8, ymin=10.4)

annotate("text", size = 3.5, x = 8.5, y = 22, colour = "#666666", label = "Average deposits are fairly low and stable \nuntil they begin increasing around Week 4 \nand then more rapidly around Week 2 & 1")

**Add captions to geom\_point**:  
geom\_text(aes(label=format(round(Emissions), big.mark=",",scientific=FALSE) , size=2, hjust = .5, vjust=-1))+  
theme(legend.position='none')

**Add captions to geom\_bar**:  
geom\_text(aes(label=round(Emissions,0), size=1, hjust=0.5, vjust=-0.5)) +  
theme(legend.position='none')

Change grid lines:   
theme(  
panel.background = element\_rect( fill = “”, colour = “”),  
panel.grid.major = element\_line() OR element\_blank(),  
panel.grid.minor = element\_line() OR element\_blank()

Can force the axis to start at a particular value rather than using coord\_cartesiaan  
expand\_limits(x = 0) or expand\_limits(y = 0)

Changing scale of axis:  
scale\_y\_continuous(breaks = round(seq(min(training\_set$Month.12), max(training\_set$Month.12), by = 0.1),1))  
scale\_y\_continuous(breaks = seq(0, 20000, 2000))  
expand\_limits(y = 0)

library(plyr)  
scale\_y\_continuous(breaks = seq(0, round\_any(max(counts$Count),100), 500))

Zoom in on area:  
coord\_cartesian(xlim = c(0,50),ylim= c(-5000,25000))

**Flip x-axis labels vertically**

theme(axis.text.x = element\_text(angle = 90, hjust = 1))

Sometimes if you want to make sure the y-axis goes down to 0 and up to a certain point you can add coord\_cartesian as well as your scale continuous.

Library(scales)  
scale\_y\_continuous(labels=comma)

scale\_y\_reverse()

Insert an abline - geom\_vline(xintercept = 0); geom\_hline(yintercept = 20)

Add % sign after axis:  
scale\_y\_continuous(labels=function(x) paste0(x,"%"))  
scale\_y\_continuous(labels=percent)

Remove legend and label lines directly:  
library(directlabels)  
geom\_dl(aes(label = Segment), method = "last.polygons", cex =1)  
---  
library(ggrepel)  
geom\_label\_repel(aes(label = Segment), nudge\_x = 1, na.rm = TRUE)  
geom\_text\_repel(data = filter(newregosegments, Month == min(newregosegments$Month)),   
 aes(label = Segment), size = 4, segment.colour = NA, nudge\_y = -71)

### Themes

Be careful of using randomForest package as it overrides the margin call.

# Example of my theme

theme\_bw() +  
 theme(panel.border = element\_blank(),  
 plot.title = element\_text(colour = "#666666", size = 12, face = "bold"),  
 plot.subtitle = element\_text(colour = "#666666", size = 10),  
 axis.title = element\_text(colour = "#666666"),  
 axis.title.x = element\_text(hjust = 0.1),  
 axis.title.y = element\_text(hjust = 0.9),  
 axis.text = element\_text(colour = "darkgrey"),  
 axis.ticks = element\_blank(),  
 plot.caption = element\_text(colour = "darkgrey", hjust = 0, size = 8))

**# Example 2:**

theme\_tq() +  
 theme(panel.border = element\_blank(),  
 plot.title = element\_text(colour = "#666666", size = 12, face = "bold"),  
 plot.subtitle = element\_text(colour = "#666666", size = 10),  
 axis.title = element\_text(colour = "#666666"),  
 axis.title.x = element\_text(hjust = 0.5),  
 axis.title.y = element\_text(hjust = 0.5),  
 axis.text = element\_text(colour = "#666666"),  
 axis.ticks = element\_blank(),  
 plot.caption = element\_text(colour = "darkgrey", hjust = 0, size = 8),  
 legend.position = "top",  
 legend.justification = 0) +  
 scale\_fill\_manual(values = c("azure3", "firebrick"))

# White background with grey gridlines.  
theme\_bw() +  
theme(panel.border = element\_blank())

theme\_tufte(ticks = FALSE) – very simple

theme\_light()

# Add border to legend  
legend.background = element\_rect(fill = "grey95")

Matt Dancho’s theme:  
theme\_tq()  
fill = palette\_light()[[1]]

library(ggthemes)  
+ theme\_...()  
economist, pander, solarized, fivethirtyeight, hc  
+ scale\_colour\_...()  
+ scale\_fill\_...(‘tableau10’)

library(devtools)  
devtools::install\_github('cttobin/ggthemr')  
library(ggthemr)  
ggthemr("dust" OR “solarized” OR “flat”)  
ggthemr\_reset()

theme(axis.title.x = element\_text(colour="DarkGreen", size=20),  
 axis.title.y = element\_text(colour="Red", size=20),  
 axis.text.x = element\_text(size=10),  
 axis.text.y = element\_text(size=10),   
 legend.title = element\_text(size=30),  
 legend.text = element\_text(size=20),  
 legend.position = c(1,1), #puts it in the top right  
 legend.justification = c(1,1), #this anchors the top right of the legend to the top right of the chart.   
 plot.title = element\_text(colour = "DarkBlue",  
 size = 25,  
 family = "Courier")) #font

# Remove legend: guides(fill = FALSE)

theme(panel.background = element\_rect(fill = "white", colour = "white"),  
 axis.ticks = element\_blank()) +  
 guides(fill = FALSE)

### Arrangement

#### CowPlot

p <- cowplot::plot\_grid(a, b, c, d, ncol = 2)   
p\_title <- ggdraw() +  
 draw\_label("title example", size = 18, fontface = "bold",  
 colour = palette\_light()[[1]])

plot\_grid(p\_title, p, ncol = 1, rel\_heights = c(0.04,1))

#### Grid Arrange

Often use library(gridExtra) & grid.arrange with multiple charts or zoomed in charts.  
grid.arrange(b, c, ncol = 2)  
hlay <- rbind(c(1,1,1,2,NA),  
 c(1,1,1,2,2),  
 c(1,1,1,2,2))  
grid.arrange(canmap1, canmap2, layout\_matrix = hlay)  
---  
grid.arrange(top=textGrob("Month-To-Date Comparison",gp=gpar(fontsize=20,font=3,col = "navyblue")), p1, p2, p3, layout\_matrix = hlay)

---

My infographic:

library(grid)  
library(gridExtra)

hlay <- rbind(c(1,1,1,2,2,2),  
 c(1,1,1,2,2,2),  
 c(3,3,3,3,3,3),  
 c(3,3,3,3,3,3),  
 c(3,3,3,3,3,3))

a <- grid.text("MTD SUMMARY", y = unit(1, "npc"), x = unit(0.5, "npc"), vjust = 1, hjust = .5, gp = gpar(fontfamily = "Impact", col = "#A9A8A7", cex = 8, alpha = 0.2))

grid.arrange(top=a, totalfinchart, newregosfinchart, segmentsbarchart, layout\_matrix = hlay)

grid.text("1st-14th", y = unit(1, "npc"), x = unit(0.5, "npc"), vjust = 2, hjust = .5, gp = gpar(fontfamily = "Impact", col = "darkblue", cex = 3, alpha = 0.5))

---

Grid arrange tables:

t1 <- ttheme\_default()  
tt2 <- ttheme\_minimal()  
tt3 <- ttheme\_minimal(  
 core=list(bg\_params = list(fill = blues9[1:4], col=NA),  
 fg\_params=list(fontface=3)),  
 colhead=list(fg\_params=list(col="navyblue", fontface=4L)),  
 rowhead=list(fg\_params=list(col="orange", fontface=3L)))

b <- grid.text("PIXEL TO REG", y = unit(1, "npc"), x = unit(0.5, "npc"), vjust = 1, hjust = .5, gp = gpar(fontfamily = "Impact", col = "#A9A8A7", cex = 8, alpha = 0.2))

grid.arrange(  
 top = b,  
 tableGrob(summarytable, theme=tt3),  
 tableGrob(countrytable, theme=tt3),  
 nrow = 1  
)

<https://cran.r-project.org/web/packages/gridExtra/vignettes/tableGrob.html>

<http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3/#put-two-potentially-unrelated-plots-side-by-side-pushviewport>

<https://cran.r-project.org/web/packages/gridExtra/vignettes/arrangeGrob.html>

## Plotting Predictors

Split into training & test sets, only do plotting in training set.

Looking for imbalance in outcome/predictors, outliers, groups of points not explained by a predictor and skewed variables.

summary(dataset)  
dim(training)  
dim(testing)

featurePlot(x=training[,c("age", "education", "jobclass")],  
 y=training$wage,  
 plot="pairs")

Look at them with scatterplots, lines, smoothers, histograms & density plots.

# Cut2, making factors (Hmisc package), e.g. cutting wage into 3 groups of quantile groups. Then use these groups in plots, put them next to each other. If you selected g2 it will split in half using median.

library(Hmisc)  
library(gridExtra)

cutWage <- cut2(training$wage,g=3)  
table(cutWage)  
p1 <- ggplot(data=training, aes(x=cutWage,y=age,fill=cutWage))+  
 geom\_boxplot()  
p2 <- ggplot(data=training, aes(x=cutWage,y=age,fill=cutWage))+  
 geom\_boxplot() +  
 geom\_jitter()  
grid.arrange(p1,p2,ncol=2)

# Tables - using the cutWage grouping to analyse further and also get a proportions table.  
t1 <- table(cutWage, training$jobclass)  
t1  
prop.table(t1, 1)

cut2(x, cuts = c(..), m = min, g = groups) # Name them with levels.

# Correlation Plot

library(ggcorrplot)

data(mtcars)  
corr <- round(cor(mtcars), 1)

ggcorrplot(corr, hc.order = TRUE,   
 type = "lower",   
 lab = TRUE,   
 lab\_size = 3,   
 method="circle",   
 colors = c("tomato2", "white", "springgreen3"),   
 title="Correlogram of mtcars",   
 ggtheme=theme\_bw)

## Transforming Data

The relationship between two variables may not be linear, sometimes a careful transformation of one or both of the variables can reveal a clear relationship. The coord\_trans() function transforms the coordinates of the plot. Alternatively, the scale\_x\_log10() and scale\_y\_log10() functions perform a base-10 log transformation of each axis.  
ggplot(data = mammals, aes(x = BodyWt, y = BrainWt)) +  
 geom\_point() +   
 coord\_trans(x = "log10", y = "log10")  
OR  
ggplot(data = mammals, aes(x = BodyWt, y = BrainWt)) +  
 geom\_point() +  
 scale\_x\_log10() + scale\_y\_log10()

## Plotly

# Scatterplot & 3D Scatterplot

plot\_ly(data, x, y, mode = "markers")  
plot\_ly( x, y, z, type = "scatter3d", mode = "markers") # these must be objects/vectors

# My Example:

plot\_ly(x = AmountHistoryRatio, y = NumBets, z = VelocityHistoryRatio, color = Excluded, type = "scatter3d", mode = "markers", colors = “Set1”, opacity = 0.8) %>%  
 layout(scene = list(xaxis = list(title = 'Amount:History'),  
 yaxis = list(title = 'Num Bets'),  
 zaxis = list(title = 'Velocity:History')))

# Time series line graph

Need to switch from short to long format - e.g. needs to be in format index, data, time. Can't be index1, index2, index 3 etc.

plot\_ly(dataset, x = time, y, color)

# Other Plots

plot\_ly(type = "histogram", "box"  
plot\_ly(z, type = "heatmap","surface") #latter gives 3d

ggplotly(ggplot object)

```{r , plotly = TRUE, echo = FALSE}  
py <- plotly(username="r\_user\_guide", key="mw5isa4yqp")  
ggplotly(gg, session = "knitr")  
```

## Interactive Maps

### GoogleVis

```{r, dependson="gv", results='asis', cache = TRUE}

G <- gvisGeoChart(dataset, locationvar = "Country",  
 colorvar = "Count", options = list(width = 600, height = 400, region = 150))  
print(G,"chart")

```

gvisGeoChart(dataset, locationvar = "Country", colorvar = "Count", options = list(width = 1500, height = 850)) #about full screen html

gvisTable

Can use gvisMerge(plot1, plot2, horizontal = TRUE/FALSE

<https://www.rdocumentation.org/packages/googleVis/versions/0.6.0/topics/gvisGeoChart>

### GGMap

Use google api for geocoding lon/lat and then map data points to map.

Good for seeing where players are coming from and trying to see if more decent players from one specific area. Also good idea to run for top customers and see if you can find certain areas where they are often located.

city <- as.character(dataset17$City)  
lonlat <- geocode(unique(city))

cities <- cbind(unique(syd), lonlat)  
colnames(cities) <- c('City','lon','lat')  
dataset17 <- merge(dataset17, cities, by.x = 'City', by.y = 'City')  
dataset17 <- dataset17[,c(2:38,1)]

Then I made a count Excel sheet grouped by City/Segment and merged this in too so I can use the size variable based on the count.

map <- get\_map(location = 'Turkey', zoom = 5)  
ggmap(map, fullpage = TRUE) +  
 geom\_point(data = lowvalue17, aes(x = lon, y = lat, size = Count.of.PlayerKey) +  
 scale\_size\_area(breaks = c(2,4,6,8))

Have done this before where I look more narrowly, e.g. “NT-Northwest Territories X0E0K0, Canada” or just “NT-Northwest Territories, Canada”

location <- as.character(Canadadataset$GoogleGeocode2)  
lonlat <- geocode(unique(location))  
geo <- cbind(unique(location), lonlat)  
colnames(geo) <- c('Location', 'lon', 'lat')  
geoCanadadataset <- merge(Canadadataset, geo, by.x = 'GoogleGeocode2', by.y = 'Location')

map <- get\_map(location = 'Canada', zoom = 3)

ggmap(map, fullpage = TRUE) +  
 geom\_point(data = geoCanadadataset, aes(x = lon, y = lat, size = CountbyGeocode2), colour = "red") +  
 scale\_size\_area() +  
 ggtitle('Top 3,000 Canadian Players Geolocations') +  
 labs(size = 'Count') +  
 theme(plot.title = element\_text(hjust = 0.5, colour = "NavyBlue", size = 16))

### Leaflet

my\_map <- leaflet() %>%  
 addTiles() %>%  
 addMarkers(lat = 39.29, lng = 76.58,  
 popup = "Rhys")

# Create data frame with lat & lon

RhysIcon <- makeIcon(  
 iconUrl = "image.png",  
 iconWidth = 31\*215/230, IconHeight = 31,  
 iconAnchorX = 31\*215/230/2, iconAnchorY = 16  
)

df %>%  
 leaflet() %>%   
 addTiles() %>%   
 addMarkers(icon = RhysIcon, popup = description)

IF too many points close together  
addMarkets(clusterOptions = markerClusterOptions())

addCircleMarkers(color = ...) %>%   
 addLegend(labels = datasetnames, colors = c("blue","red","green"))

addCircles(weight = 1, radius = sqrt(dataset$count) \* 30)

# Different tile map options  
addProviderTiles("Esri.NatGeoWorldMap")

<http://leaflet-extras.github.io/leaflet-providers/preview/>

**My Template**

datasetgeo <-   
 datasetgeo %>%   
 group\_by(GeocodeBroad) %>%   
 summarise(TotalCount = n(),  
 TopCount = sum(PurchaseRank == 'Top')) %>%   
 mutate(Percentage = 100 \* TotalCount/sum(TotalCount),  
 TopConversion = 100 \* TopCount/TotalCount) %>%   
 arrange(desc(Percentage))

location <- as.character(datasetgeo$GeocodeBroad)  
lonlat <- geocode(location)  
geos <- cbind(location, lonlat)

failures <- droplevels(geos[is.na(geos$lon),])  
location2 <- as.character(failures$location)  
lonlat2 <- geocode(location2)

geos2 <- cbind(location2, lonlat2)

geos <- geos[!is.na(geos$lon),]  
geos2 <- geos2[!is.na(geos2$lon),]

lonlattbl <- rbind(geos, geos2, geos3)

newgeodataset <- merge(datasetgeo, lonlattbl, by = 'location')  
newgeodataset$labels <- paste(newgeodataset$location, ': ', "P/P - ", prettyNum(newgeodataset$TotalCount, big.mark=","), '; ', "Top Pl Conv - ", percent(newgeodataset$TopConversion/100))

my\_map <-  
 leaflet() %>%   
 addProviderTiles("Esri.NatGeoWorldMap") %>%   
 addCircleMarkers(lat = newgeodataset$lat,   
 lng = newgeodataset$lon,   
 popup = newgeodataset$labels,   
 radius = (newgeodataset$Percentage),  
 color = ifelse(newgeodataset$TopConversion >= 5, "blue",palette\_light()[[2]])  
 )  
my\_map

library(htmlwidgets)  
saveWidget(my\_map, file="map.html")

<https://rstudio.github.io/leaflet/>

## Infographics

grid.table()

library(grid)  
library(gridExtra)

hlay <- rbind(c(1,1,1,2,2,2),  
 c(1,1,1,2,2,2),  
 c(3,3,3,3,3,3),  
 c(3,3,3,3,3,3),  
 c(3,3,3,3,3,3))

a <- grid.text("MTD SUMMARY", y = unit(1, "npc"), x = unit(0.5, "npc"), vjust = 1, hjust = .5, gp = gpar(fontfamily = "Impact", col = "#A9A8A7", cex = 8, alpha = 0.2))

grid.arrange(top=a, totalfinchart, newregosfinchart, segmentsbarchart, layout\_matrix = hlay)

grid.text("1st-14th", y = unit(1, "npc"), x = unit(0.5, "npc"), vjust = 2, hjust = .5, gp = gpar(fontfamily = "Impact", col = "darkblue", cex = 3, alpha = 0.5))

<https://www.r-bloggers.com/r-how-to-layout-and-design-an-infographic/>

<https://www.noupe.com/design/fantastic-information-architecture-resources.html>

<https://stackoverflow.com/questions/2162131/how-can-i-learn-to-create-beautiful-infographics-with-connection-to-my-r-knowle>

## Clustering

Clustering is similar to classification, but the basis is different. In Clustering you don’t know what you are looking for, and you are trying to identify some segments or clusters in your data. When you use clustering algorithms on your dataset, unexpected things can suddenly pop up like structures, clusters and groupings you would have never thought of otherwise.

Clustering is fairly sensitive to scaling.

Only for continuous data, so need to convert factors.

Can do threshold/rule based segmentation, e.g. just two pre-set dimensions, but typically things will meet your initial assumptions. More value when you run this against a number of dimensions.

### Hierarchical Clustering

A simple way of quickly examining and displaying multi-dimensional data. This technique is usually most useful in the early stages of analysis when you're trying to get an understanding of the data, e.g., finding some pattern or relationship between different factors or variables.

Agglomerative (bottom-up) approach – find two closest points, merge them together, then find next closest, etc.

Produces a tree showing how close things are to each other.

How you define close is the most important step, must pick a distance/similarity that makes sense for your problem.  
Continuous: Euclidean distance or correlation similarity; Binary: Manhattan distance.  
  
distxy <- dist(dataFrame[,1:3])  
hClustering <- hclust(distxy)  
plot(hClustering)  
plot(as.dendrogram(hClustering))  
abline(h=1.5, col="blue")  
# can use myplclust  
mypclust(hclustering, lab.col = unclass(dataFrame$factorvariable)

heatmap() runs a quick hierarchical clustering visualization on a matrix.  
heatmap(dataMatrix, col = cm.colors(25))

### K-Means Clustering

Iterative process:  
Step 1: Choose the number K of clusters.  
Step 2: Select at random K points, the centroids (not necessarily from your dataset).  
Step 3: Assign each data point to the closest centroid (Euclidian distance) > that forms K clusters.  
Step 4: Compute and place the new centroid of each cluster.  
Step 5: Essentially step 3 again, reassign each data point to the new closest centroid. If any reassignment then back to step 4, if not finished.



Random Initialization Trap – solve this using K means ++ for more specific centroids assignment.

Choose the right number of clusters using WCSS, the lesser the WCSS the better the fit. Looking for the elbow of the graph and make a judgement call for what is optimal for your analysis.



# SDS Method

dataset <- read.csv("Mall\_Customers.csv")  
X <- dataset[,4:5]

# Using the elbow method to find optimal number of clusters

set.seed(6)  
wcss <- vector()  
for (i in 1:10) wcss[i] <- sum(kmeans(X, i)$withinss)  
plot(1:10, wcss, type = 'b', main = paste("Clusters of clients"), xlab = "Number of clusters", ylab = "WCSS") #type = b, means "both" lines and points.

# Applying k-means to the mall dataset

set.seed(29)  
kmeans <- kmeans(X, centers = 5, iter.max = 300, nstart = 10) #Choosing the dataset, the number of clusters, the maximum iterations allowed, how many random sets.

# Visualizing the clusters (2 dimensions)

library(cluster)

clusplot(X,  
 kmeans$cluster,  
 lines = 0, # so no distance lines appear  
 shade = TRUE,   
 color = TRUE,  
 labels = 2,  
 plotchar = FALSE,  
 span = TRUE, #plot ellipsis  
 main = paste('Cluster of clients'),  
 xlab = "Annual Income",  
 ylab = "Spending Score")

# Coursera Method

kmeansObj <- kmeans(dataFrame, centers = 3)  
kmeansObj$cluster  
table(kmeansObj$cluster, dataset$outcome)  
plot(x, y, col = kmeansObj$cluster, pch = 19, cex = 2)  
points(kmeansObj$centers, col = 1:3, pch = 3, cex = 3, lwd = 3)

plot(x, y, col = kmObj$cluster, pch = 19, cex = 2)  
points( kmObj$centers, col = c("black","red","green"), pch = 3, cex = 3, lwd = 3)  
---  
plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2)

---

table(kmeansObj$cluster)  
dataset$Cluster <- kmeansObj$cluster

Heatmaps of matrix  
kmeansObj2 <- kmeans(dataMatrix, centers = 3)  
image(t(dataMatrix)[, order(kmeansObj$cluster)], yaxt = “n”)

Can then plot the clusters to see which features drive the location of the centre for each cluster, so then we know which features are important in classifying people in each cluster.

Plot(kmeansObj$cluster[1, 1:10], pch = 19, ylab = “Cluster Centre”, xlab = “”)  
# first number is the cluster and second is the features

Customer segmentation - <http://www.business-science.io/business/2016/08/07/CustomerSegmentationPt1.html>

<https://www.optimove.com/learning-center/customer-segmentation-via-cluster-analysis>

RFM segmentation & clustering - <http://www.kimberlycoffey.com/blog/2016/8/k-means-clustering-for-customer-segmentation>

## SVD (Singular Value Decomposition)

Trying to find the best matrix with fewer variables (lower rank) that explains the data.

Scales matter and patterns are often hard to find.

U matrix associated with the scaled data matrix. This is the first LEFT singular vector and it's associated with the ROW means of the clustered data.  
V matrix is the first right singular vector and is associated with the column means of the clustered data.  
The diagonal entries of D are like weights for the U and V columns accounting for the variation in the data.

Plot how much of the variance it explains.

Can multiply each of the components to get the real numbers.

Svd1 = svd(scale(matrix))  
plot(svd1$u[,1], col = sub1$class, pch = 19)  
plot(svd1$u[,2], col = sub1$class, pch = 19)

variance <- prop.table(svd1$d^2)  
plot(variance)

plot(svd1$d^2/sum(svd1$d^2), type = "b", pch = 16, xlab = "principal components",   
 ylab = "variance explained")

# New clustering with maximum contributor to see which variable tells the most variance.  
maxContrib <- which.max(svd1$v[,2])  
distanceMatrix <- dist(sub1[, c(10:12, maxContrib)])  
hclustering <- hclust(distanceMatrix)  
mypclust(hclustering, lab.col = unclass(sub1$factorvariable)  
names(dataset)[maxContrib]

CASE STUDY: plot svd1$u and can see separation of one class in the plot. To figure out why that is we need to find which of the variables contributes to the variation of that component by looking at the RIGHT singular vectors svd1$v. In particular, the second one as the cluster stood out in the second column of svd1$u.  
Plot the second column but can’t see anything.   
maxCon <- which.max(svd1$v[,2])  
mdist <- dist(sub1[,c(10:12,maxCon)])  
hclustering <- hclust(mdist)  
myplclust(hclustering, lab.col = unclass(sub1$activity))  
names(sub1[maxCon]) # shows you which column is the biggest contributor to the clustering.  
--  
kClust <- kmeans(sub1[,-c(562,563)], centers = 6)  
table(kClust$cluster, sub1$activity)  
kClust <- kmeans(sub1[,-c(562,563)], centers = 6, nstart = 100) #nstart is the random starts number  
laying <- which(kClust$size==29)  
plot(kClust$centers[laying, 1:12],pch=19,ylab="Laying Cluster") # We see the first 3 columns dominate this cluster center.  
names(sub1[,c(1:3)])  
walkdown <- which(kClust$size==49)  
plot(kClust$centers[walkdown,1:12], pch = 19, ylab = "Walkdown Cluster") # clustering by the first twelve columns to see if we can see any pattern.

## Case Study

read.table

strsplit(cnames, "|", fixed = TRUE)

mean(is.na(x0))

Sometimes when plots are way too narrow you can perform a log10 on them

negative <- x1<0  
mean(negative, na.rm = TRUE)

dates <- pm1$Date  
dates <- as.Date(as.character(dates), "%Y%m%d")  
hist(dates[negative], "month")

Split cnt0 into several data frames with the numbers per county site.  
sapply(split(cnt0, cnt0$county.site), nrow)

Time Series Plot - 2 graphs one row  
par(mfrow = c(1,2), mar = c(4,4,2,1))  
plot(dates0, x0sub, pch = 20)  
abline(h = median(x0sub, na.rm = TRUE),lwd=2)  
Putting 2 plots on the same range:   
rng <- range(x0sub, x1sub, na.rm = TRUE)  
ylim = rng

Get mean of each sample by state  
mn0 <- with(pm0, tapply(Sample.Value, State.Code, mean, na.rm = TRUE))

merge(d0,d1, by = "state")

## Chart Websites

<http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/81-ggplot2-easy-way-to-mix-multiple-graphs-on-the-same-page/>

<https://www.r-graph-gallery.com/portfolio/ggplot2-package/>

<http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/>

<http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3/#change-the-plot-background-not-the-panel-color-plot.background>

<http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3/>

<http://sape.inf.usi.ch/sites/default/files/ggplot2-colour-names.png>

Mosaic plots - <https://cran.r-project.org/web/packages/ggmosaic/vignettes/ggmosaic.html>

# **R: Machine Learning**



3 main concepts: Regression, Classification & Clustering



Regression is used to predict continuous values. Classification is used to predict which class a data point is part of (discrete value).

Classification: you are given some new data, you have to set new label for them.  
For example, a company wants to classify their prospect customers. When a new customer comes, they have to determine if this is a customer who is going to buy their products or not.

Clustering: you're given a set of history transactions which recorded who bought what.  
By using clustering techniques, you can tell the segmentation of your customers.



## Prediction Model Basics

saveRDS(results, "rfeResults.rds")

Question > Data > Features > Algorithms > Parameters > Evaluation

The input data is the most important part.

Features – expert application knowledge; can do semi-supervised learning in Caret.

Predictions are about accuracy trade-offs and the right balance.

Out of sample error (generalization error) – the error rate you get on a new data set, more important.

Design:   
- split into training, testing, validation (optional).  
- Training set – pick features and prediction function using cross-validation  
- If no validation set, then apply the model only 1x to the test. Only use it once otherwise you’re basically using the test set to train the model.  
- If validation set, then apply to test set and refine & apply the model only 1x to validation.  
- So basically you want to keep the test set on its own to be used only one time as an estimate so it’s unbiased and your model doesn’t become tuned on the test set.  
- Also need to set benchmarks.

Rule thumb for splitting:  
- Large sample size (60% training, 20% test, 20% validation)  
- Medium sample size (60% training, 40% test\_  
- Small sample size (do cross validation, report caveat of small sample size)

Types of errors for binary outcomes:  
- True positive – correctly identified  
- False positive – incorrectly identified (type I)  
- True negative – correctly rejected  
- False negative – incorrectly rejected (type II)

Key Quantities:  
- Sensitivity – positive test / positive outcome – TP / (TP + FN)  
- Specificity – negative test / negative outcome – TN / (FP+TN)  
- Positive predictive value – positive outcome / positive test – TP / (TP+FP) – e.g. if the test gave you a positive result, what are the chances you actually have positive outcome.  
- Negative predictive value – negative outcome / negative test – TN / (TP+FP+FN+TN)  
- Accuracy – correct outcome – (TP+TN) / (TP+FP+FN+TN)



If you’re predicting a rare event, you need to know the sample and how rare that event is.

For continuous data – most common use Root Mean Squared Error (sensitive to outliers though). Common error measures:

Unsupervised prediction – when you don’t know the labels for prediction. To build a predictor – create & name clusters; build predictor for clusters. Then, in a new dataset, predict the clusters.

IDEAS: Need to think about how you would use the model. E.g. you can predict with 80% accuracy those customers who will still be depositing, this is interesting but then what you want to focus on is those customers your model predicted WOULD still be depositing but didn’t. Can you create a new class and then cluster these in some way?  
OR do you change the query as in churn predictions and predict based churning (e.g. leaving).

 

For probabilities you use an ROC curve; AUC of 0.5 is random guessing (45 degree), in general AUC of 0.8 is considered “good”.

Cross validation – split training/test sets, build a model on the training set and only at the end evaluate on the test set. Repeat and average the estimated errors. Useful for picking variables to include, picking the type of prediction function to use, picking the parameters in the prediction function and comparing different predictors.

Collaborative filtering – using similarities to give predictions, e.g. similar ratings on movies so recommending a movie they haven’t watched.

Content filtering – using information to make predictions, e.g. same director/actor or genre.

## Predicting with Tree Basics

When predicting with trees you continue iteratively until the leaf predicts a positive result.

1. Start with all variables in one group.
2. Find variable/split that best separates outcomes.
3. Divide data into two groups (“leaves”) on that spit (“node”).
4. Within each split, find best variable/split that separates the outcomes.
5. Continue until groups are too small or sufficiently “pure”.

Errors:   
Misclassification – 0 = perfect purity, 0.5 = no purity  
Gini index – 0 = perfect purity, 0.5 = no purity  
Deviance/information gain = 0 = perfect purity, 1 = no purity

Data transformations may be less important.

## Data Splitting

library(caret)

inTrain <- createDataPartition(y = dataset$DEPOSITING,  
 p = 0.7, list = FALSE)

training <- dataset[inTrain,]

testval <- dataset[-inTrain,]

inTest <- createDataPartition(y = testval$DEPOSITING,  
 p= 0.5, list = FALSE)

testing <- testval[inTest,]  
validating <- testval[-inTest,]

listdf <- list(training, testing, validating)

listdf %>%   
 map(~dim(.))

---

inTrain <- createDataPartition(y=spam$type,  
 p=0.75, list = FALSE)

training <- spam[inTrain,]  
testing <- spam[-inTrain,]

modelFit <- train(type ~., data=training, method="glm")

## Training the Model

Metric Options:  
- Continuous outcomes – Root mean squared error (RMSE) or RSquared  
- Categorical outcomes – Accuracy (fraction correct) or Kappa (a measure of concordance)

trainControl for more control of your model – Resampling:  
Method:  
- boot = bootstrapping  
- cv = cross validation  
- repeatedcv = repeated cross validation  
- LOOCV = leave one out cross validation  
Number:  
- For cross validation  
- Number of subsamples to take  
Repeats:  
- Number of times to repeat subsampling  
- If big this can slow things down

Setting overall seed ensures the random numbers generated are the same each time for parallel fitting.

## Pre-Processing

Especially if the data is extremely skewed; training and test must be processed in the same way.

Don’t transform factor variables.

Dealing with categorical variables:  
- Common to combine those with a low frequency (e.g. < 5%)  
- Can also combine levels having similar response rate.

From Reza:

*The clustering method majority of the time apply on your standrdized data.(each varibale is going to have a mean of zero and variance of one) to give more weights to single variable you can play by how the variable standrizing process will be implemented. More information can be found in https://stats.stackexchange.com/questions/77850/assign-weights-to-variables-in-cluster-analysis. Some really simpler solution could be to have several copy of the variable of more important. so for example you can have number of purchase1 ,number of purchase 2,.... with the same value. with this approach you force the model to put more weight on number of purchase in compare to other variable(this approach will work but I suggest try to find bettter methods) there are many pages with suggestion regarding weighting the variable before KNN and K-means*

### Treating Imbalanced Classes

Library(DMwR)

Usually you treat class imbalance if the ratio is around 1:10 but sometimes it helps if the ratio is slightly smaller too.

Unbalanced classification problems cause numerous issues to many learning algorithms. These problems are characterized by the uneven proportion of cases that are available for each class of the problem. SMOTE (Chawla et. al. 2002) is a well-known algorithm to fight this problem. The general idea of this method is to artificially generate new examples of the minority class using the nearest neighbors of these cases. Furthermore, the majority class examples are also under-sampled, leading to a more balanced dataset.

-- This makes the targets 50/50  
(need the data frame argument as it gives an error if you pass a tibble)

trainingsm <- SMOTE(Attrition ~ ., as.data.frame(trainingObj), perc.over = 100, perc.under = 200)

<http://amunategui.github.io/smote/>

<https://www.kaggle.com/mansoori/attrition-analysis-and-prediction-auc-0-92>

<https://www.kaggle.com/aljaz91/ibm-s-attrition-tackling-class-imbalance-with-gbm>

### Correlation

Correlation and absolute value  
  
M <- abs(cor(training[,-58]))  
diag(M) <- 0 #don't want those who have a correlation of 1 as that's themselves, so setting them to 0  
which(M > 0.8,arr.ind=T)

library(lsr)  
correlate(training[,-30],corr.method="pearson", p.adjust.method="holm")

library(corrplot)  
Customer\_Cor <- Customer %>% select(sum:pets)  
M <- cor(Customer\_Cor)  
diag(M) <- 0  
corrplot(M, method="square")   
corrplot(M, method="square", type = "upper", order = "AOE",)  
corrplot.mixed(M, lower.col = "black", number.cex = .7)  
which(M > 0.8,arr.ind=T)

# If you get an error with corrplot, use as.matrix()

<https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html>

Can use Kendall’s correlation matrix for analysing the correlation of categorical variables.

Can also try using ggcor

ggcorr(data = bgg.useful %>%  
 filter(stats.usersrated >= 100) %>%  
 select(starts\_with("details"),  
 +stats.average,  
 +stats.averageweight,  
 -ends\_with(".factor"),  
 -details.name,  
 -details.yearpublished) ,  
 label = TRUE,  
 geom = "tile",  
 low = corr.palette[1],  
 mid = corr.palette[2],  
 high = corr.palette[3],  
 midpoint=0,  
 size = 2.5) +   
 ggtitle("Correlations between Details Metrics")

### Center/Scaling

Usually the calculation you use is (value – mean) / standard deviation.

library(caret)

# Standardizing - Imputing NA data  
preProcess(training[,-58],method="knnImpute")

# Center & scaling, can then use the same pre-processing from training on test  
preObj <- preProcess(training[,-58], method=c("center","scale"))  
trainCapAveS <- predict(preObj,training[,-58])$capitalAve  
testCapAveS <- predict(preObj,testing[,58])$capitalAve

# Can use box-cox transforms on continuous data  
preObj <- preProcess(training[.-58],method=c("BoxCox"))

# Can include this within the train function  
set.seed(32343)  
modelFit <- train(type~.,data=training,  
 preProcess = c("center","scale"),method="glm")

--- Another example which I have performed:

preObj <- preProcess(training[,c("…")], method = c("center", "scale"))

training <- predict(preObj, training)  
validating <- (predict(preObj, validation))  
testing <- predict(preObj, testing)

### Dummy Variables

library(dummies)  
trainingdumm <- dummy.data.frame(trainingdumm, names=c("…",”…”), sep="\_")

### Feature Creation

Step 1 – extracting the best features to summarize your date. Balancing act of summarization vs information loss; better to err on the side of more features. )

Step 2 – transforming your covariates into new covariates. More important for regression and SVMs than classification. Should only be done in training set and found through exploratory analysis; new covariates should be added to the data frames.

Common to add dummy variables of factors – dummyVars(wage ~ jobclass, data = training)

Removing variables with little variability – nearZeroVar(training, saveMetrics = TRUE)

Can fit curve regression lines with splines.

### PCA

#### Pre-processing

PCA is a method to reduce high dimensional data to its essential elements (not lose information) and explain the variability in the data.

Most useful for linear-type models but can make it harder to interpret predictors.

Outliers can have a big effect, make sure you check plots first.

See variables which are highly correlated to each other (e.g. above 80%).

Can combine the variables by adding them or subtracting them, e.g. in this case you can see the most variability is in the x axis so adding them together works best.

# Correlation and absolute value  
M <- abs(cor(training[,-58]))  
diag(M) <- 0 #don't want those who have a correlation of 1 as that's themselves, so setting them to 0  
which(M > 0.8,arr.ind=T)

# Look at columns of correlated predictors  
plot(spam[,34],spam[,32])

A weighted combination of predictors might be better, pick this combination to capture the most information possible. Benefits - reduces number of predictors, reduces noise.

--- Method I have used

prepca <- preProcess(training[,c("Purchases", "Bonus")], method = "pca", pcaComp = 1)  
training <- predict(prepca, training)  
testing <- predict(prepca, testing)  
validation <- predict(prepca, validation)



Statistical goal - Find a new set of multivariate variables that are uncorrelated and explain as much variance as possible.  
Data Compression goal – Best matrix created with fewer variables that explains the original data.

# Principal Components – can condense a large number of quantitive variables down.

smallSpam <- spam[,c(34,32)]  
prComp <- prcomp(smallSpam)  
plot(prComp$x[,1],prComp$x[,2])  
prComp <- prcomp(log10(spam[,-58]+1))

# PCA with Caret

preProc <- preProcess(log10(spam[,-58]+1),method="pca",pcaComp=2)  
spamPC <- predict(preProc, log10(spam[,-58]+1))  
ggpplot(aes(x=spamPC[,1],y= spamPc[,2],colour = spam$type)) +  
 geom\_point()

# Preprocessing with PCA

preProc <- preProcess(log10(training[,-58]+1), method = "pca", pcaComp = 2)  
trainPC <- predict(preProc, log10(training[,-58]+1))  
modelFit <- train(training$type ~ ., method = "glm",data = trainPC)

# Have to use same PCA on test data set

testPC <- predict(preProc, log10(testing[,-58]+1))  
confusionMatrix(testing$type,predict(modelFit,testPC))

# Alternative (caret does it for you)

modelFit <- train(training$type ~ ., method="glm",preProcess="pca",data=training)  
confusionMatrix(testing$type,predict(modelFit,testing))

# Method I’ve used to see which variables contribute the most to variance, also shows which columns are contributing to the most variance.

pca <- prcomp(datasetfilt[,-1], center = TRUE, scale = TRUE)  
summary(pca)  
print(pca)  
plot(pca, type = "l")

loadings <- eigen(cov(datasetfilt[,-1]))$vectors  
explvar <- loadings^2  
explvar

<https://stats.stackexchange.com/questions/87037/which-variables-explain-which-pca-components-and-vice-versa>

#### Feature Extraction - SDS

From the independent variables in your dataset, PCA extracts new independent variables that explain the most variances of the dataset regardless of the dependent variable. As the dependent variable isn’t considered, PCA is an unsupervised model.

Example case – clustering was done to establish customer segments based on what wine they like and then want to build a logistic regression model so when new wines are released we can predict who to recommend them to. If we want clear visualizations of the prediction regions and prediction boundary of classification model, we need to use dimensionality reduction to bring this down to only 2 independent variables that explain the most the variance so we can represent in one plot.

Need to feature scale with PCA; after data pre-processing you apply PCA then build the model.

library(caret)  
library(e1071)

pca = preProcess(x = training\_set[-14],  
 method = "pca",  
 pcaComp = 2)

# thresh = threshold of how much variance you want the variables to explain, e.g. 0.6  
# pcaComp = number of PCA components to keep, e.g. only want 2.

training\_set <- predict(pca, training\_set)  
training\_set <- training\_set[,c(2,3,1)] # re-order columns so dependent variable is last.

test\_set <- predict(pca, test\_set)  
test\_set <- test\_set[,c(2,3,1)]

Now you can build the model, make the predictions and confusion matrix. Then have to change the plot codes as there are 3 levels for dependent variable instead of 2 as we have done before.

library(ElemStatLearn)  
set = training\_set  
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)  
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)  
grid\_set = expand.grid(X1, X2)  
colnames(grid\_set) = c('PC1', 'PC2')  
y\_grid = predict(classifier, newdata = grid\_set)

plot(set[, -3],  
 main = 'SVM (Training set)',  
 xlab = 'PC1', ylab = 'PC2',  
 xlim = range(X1), ylim = range(X2))  
contour(X1, X2, matrix(as.numeric(y\_grid), length(X1), length(X2)), add = TRUE)  
points(grid\_set, pch = '.', col = ifelse(y\_grid == 2, 'deepskyblue', ifelse(y\_grid == 1, 'springgreen3', 'tomato')))  
points(set, pch = 21, bg = ifelse(set[, 3] == 2, 'blue3', ifelse(set[, 3] == 1, 'green4', 'red3')))



<https://tgmstat.wordpress.com/2013/11/28/computing-and-visualizing-pca-in-r/>

## Feature Engineering

All comes down to how much you understand the data, with your human intuition and creativity.

Indicator variables: Can create binary indicator variables if an observation meets a certain condition and isolate key properties. E.g. a model for housing prices over a period that spans through the market crash.

Interaction features: combination of two or more features; can be products, sums or differences between two features. Ask yourself “could I combine this information in any way that might be even more useful?”  
E.g. think about school example – number of schools in area and also a scoring variable; what’s important is number of good schools so you multiply the two together.

Sparse classes: Combing categorical variables with few observations. As a general guideline combine them until each class has around 50 observations.

Decompose categorical variables: Instead of red, blue or unknown. Make a new variable for “Has\_Colour” or binary features for each colour.

Decompose datetimes: if you suspect relationships between time of day or other attributes break these down further – e.g. “Hour\_of\_day” or “Part\_of\_day” (morning, afternoon, etc) or time of month or seasons.

Reframing numerical qualities: e.g. domain knowledge that something happens at a certain point, e.g. higher taxation at certain amount so you create new variables “Above\_amount”. Or you could create variables aggregated over periods.

Dummy variables.

Remove unused or redundant features.

We may have a SUM field and instead want to do this by time period or by season where we would need to step backwards and use the real data.

<https://elitedatascience.com/feature-engineering>

<https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/>

<https://www.analyticsvidhya.com/blog/2015/11/8-ways-deal-continuous-variables-predictive-modeling/>

### Discretize Count Variables

Numeric features like age, years worked, length of time in a position, count purchases; rather than individual observations we group them into cohorts.

### Log Transformations on Numeric Variables

Especially where we can see observations bunched within a small part in histogram.

If the feature has 0s we can manually change these values to 1 or for example set all values between 1-10 to 0 and then take a logarithm of those remaining.

In the cases of positive data, you could find the minimum (above 0) and treat all the 0 values as a desired insignificant (or a practically negligible) value that is smaller than the minimum. In plots, this is shown as the background.

Make the tranformation log(1+t) instead of log(t). After you transform back, subtract 1. Or you can use square root transformation. Can only do this if t has positive values.

library(corrr)

list\_datasets[[1]] %>%  
 select(VipPlayer, Wagered) %>%  
 mutate(  
 VipPlayer = VipPlayer %>% as.factor() %>% as.numeric(),  
 Wagered = Wagered,  
 LogWagered = log(Wagered)  
 ) %>%  
 correlate() %>%  
 focus(VipPlayer) %>%  
 fashion()

<https://www.researchgate.net/post/Log_transformation_of_values_that_include_0_zero_for_statistical_analyses2>

### One Hot Encoding

One-hot encoding is the process of converting categorical data to sparse data, which has columns of only zeros and ones

### PreProcessing With Recipe

rec\_obj <- recipe(Churn ~ ., data = train\_tbl) %>%  
 step\_discretize(tenure, options = list(cuts = 6)) %>%  
 step\_log(TotalCharges) %>%  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%  
 step\_center(all\_predictors(), -all\_outcomes()) %>%  
 step\_scale(all\_predictors(), -all\_outcomes()) %>%  
 prep(data = train\_tbl)

x\_train\_tbl <- bake(rec\_obj, newdata = train\_tbl)  
x\_test\_tbl <- bake(rec\_obj, newdata = test\_tbl)

Can also use all\_numeric() if you’re not making dummy categorical variables.

## Feature Selection

Random Forest package has its own built in variable importance function – “importance”. With RF, the Gini importance index is defined as the averaged Gini decrease in node impurities over all trees in the forest (it follows from the fact that the Gini impurity index for a given parent node is larger than the value of that measure for its two daughter nodes, see e.g. (2)).

The caret R package provides tools automatically report on the relevance and importance of attributes in your data and even select the most important features for you.

### Rank Features with Caret

library(caret)  
importance <- varImp(model, scale=FALSE)  
# summarize importance  
print(importance)  
plot(importance)

### Feature Correlation w/ Outcome (corrr)

All columns must be numeric, if you have factors then one hot encode them.

corrr\_analysis <- trainingObj[,-1] %>%  
 mutate(CHURNED = ifelse(trainingObj$CHURNED == 'Yes',1,0)) %>%  
 correlate() %>%  
 focus(CHURNED) %>%  
 rename(‘feature’ = ‘rowname’) %>%  
 arrange(abs(CHURNED)) %>%  
 mutate(feature = as\_factor(feature))

corrchurn <- corrr\_analysis %>%   
 top\_n(20, CHURNED)

corrstay <- corrr\_analysis %>%   
 top\_n(20, desc(CHURNED))

correlation <- corrchurn %>%   
 rbind(corrstay)

correlation %>%  
 ggplot(aes(x = CHURNED, y = fct\_reorder(feature, desc(CHURNED)))) +  
 geom\_point() +  
 geom\_segment(aes(xend = 0, yend = feature),   
 color = palette\_light()[[2]],   
 data = correlation %>% filter(CHURNED > 0)) +  
 geom\_point(color = palette\_light()[[2]],   
 data = correlation %>% filter(CHURNED > 0)) +  
 # Negative Correlations - Prevent churn  
 geom\_segment(aes(xend = 0, yend = feature),   
 color = palette\_light()[[1]], #”"chartreuse4"  
 data = correlation %>% filter(CHURNED < 0)) +  
 geom\_point(color = palette\_light()[[1]], #"chartreuse4"  
 data = correlation %>% filter(CHURNED < 0)) +  
 # Vertical lines  
 geom\_vline(xintercept = 0, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 geom\_vline(xintercept = -0.25, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 geom\_vline(xintercept = 0.25, color = palette\_light()[[5]], size = 1, linetype = 2) +  
 # Aesthetics  
 theme\_bw() +  
 labs(title = "Churn Correlation Analysis",  
 subtitle = "Positive Correlations (contribute to churn), Negative Correlations (prevent churn)",  
 y = "Feature Importance")

# sometimes I change the negative correlations color to “chartreuse4”

### Auto Feature Selection in Caret

Automatic feature selection methods can be used to build many models with different subsets of a dataset and identify those attributes that are and are not required to build an accurate model.

A popular automatic method for feature selection provided by the caret R package is called Recursive Feature Elimination or RFE.

# define the control using a random forest selection function  
control <- rfeControl(functions=rfFuncs, method="cv", number=10)  
results <- rfe(training[,-21], training[,21], sizes = c(1:20), rfeControl = control)

print(results)  
predictors(results)  
plot(results, type=c("g", "o"))

---

ctrl <- rfeControl(functions = lmFuncs,  
 method = "repeatedcv",  
 repeats = 5,  
 verbose = FALSE)

lmProfile <- rfe(x, y,  
 sizes = subsets,  
 rfeControl = ctrl)

<https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/>

## Regression



Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable (s) (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables.

Main benefits:  
> It indicates the significant relationships between dependent variable and independent variable.  
> It indicates the strength of impact of multiple independent variables on a dependent variable.

Regression analysis also allows us to compare the effects of variables measured on different scales, such as the effect of price changes and the number of promotional activities. These benefits help market researchers / data analysts / data scientists to eliminate and evaluate the best set of variables to be used for building predictive models.

### Standard Preparation - SDS

setwd("C:/Users/rtutt/Desktop/R/Machine Learning/Machine Learning A-Z Template Folder/Part 1 - Data Preprocessing/Section 2 -------------------- Part 1 - Data Preprocessing --------------------")

dataset <- read.csv("Data.csv")  
# dataset <- dataset[, 2:3]

# Split dataset into training and test  
install.packages('caTools')  
library('caTools')  
set.seed(123)  
split <- sample.split(dataset$Purchased, SplitRatio = 0.8)  
training\_set <- subset(dataset, split == TRUE)  
test\_set <- subset(dataset, split == FALSE)

# Feature Scaling – must put values of variables on same scale to avoid one value dominating Euclidian distance using standardisation or normalisation.  
# training\_set[, 2:3] <- scale(training\_set[, 2:3])  
# test\_set[, 2:3] <- scale(test\_set[, 2:3])

### Regression Template - SDS

# Fitting the Regression Model

… Create Regressor here

# Predicting a new result

y\_pred <- predict(regressor, data.frame(Level = 6.5))  
OR  
y\_pred = predict(regressor, newdata = test\_set)

# Visualizing the Training set results

ggplot() +  
 geom\_point(aes(x = training\_set$Month.4, y = training\_set$Month.12, colour = training\_set$Period), size = 3) +  
 geom\_line(aes(x = training\_set$Month.4, y = predict(regressor, newdata = training\_set)),  
 colour = 'blue') +  
 scale\_y\_continuous(breaks = round(seq(min(training\_set$Month.12), max(training\_set$Month.12), by = 0.1),1)) +  
 ggtitle('Month 12 vs Month 4 Deposit Multiplier') +  
 xlab('Month 4') +  
 ylab('Month 12') +  
 labs(colour = "Training - Rego Date")

# Visualizing the Continuous Regression Model results.

library("ggplot2")  
ggplot() +  
 geom\_point(aes(x = dataset$Level, y = dataset$Salary),  
 colour = 'red') +  
 geom\_line(aes(x = x\_grid, y = predict(regressor, newdata = dataset)),  
 colour = 'blue') +  
 ggtitle('Truth or Bluff (Regression Model)') +  
 xlab('Level') +  
 ylab('Salary')

# Visualizing the non-Continuous Regression Model results (for higher resolution and smoother curve)

library("ggplot2")  
x\_grid = seq(min(dataset$Level), max(dataset$Level), 0.1)

ggplot() +  
 geom\_point(aes(x = dataset$Level, y = dataset$Salary),  
 colour = 'red') +  
 geom\_line(aes(x = x\_grid, y = predict(regressor, newdata = data.frame(Level = x\_grid))),  
 colour = 'blue') +  
 ggtitle('Truth or Bluff (Regression Model)') +  
 xlab('Level') +  
 ylab('Salary')

### Simple Linear Regression

regressor <- lm(formula = Salary ~ YearsExperience,  
 data = training\_set)

summary(regressor) - Important part is the coefficients, 3 stars means high correlation. Usually want p-value to be under 0.05.

--- Caret

modelFit <- train(y ~ x, data=training, method="lm")  
summary(modelFit)

ggplot() +  
 geom\_point(aes(x=trainFaith$waiting, y=trainFaith$eruptions), colour="red")+  
 geom\_line(aes(x=trainFaith$waiting, y=predict(modelFit,newdata = trainFaith)),colour="blue")

predict(modelFit,data.frame(waiting=70))

### Multiple Linear Regression

Dummy variables for categorical variables, always omit one dummy variable.

dataset$State <- factor(dataset$State,  
 levels = c("New York", "California", "Florida"),  
 labels = c(1,2,3))

R library already removes one dummy variable when performing the regression.

regressor <- lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend + State,  
 data = dataset)

Remove the variables with the least statistical significance (highest p-value above your threshold) where the adjusted r-squared increases as you remove.

--- Caret

featurePlot(x=training[,c("age","education","jobclass")],  
 y = training$wage,  
 plot = "pairs")

# Can see two different groups, add colour to see why difference.  
ggplot(data=training,aes(x=age,y=wage)) +  
 geom\_point(aes(colour=jobclass))

# Fit linear model  
modelFit <- train(wage ~ age + jobclass + education,  
 method = "lm", data = training)

finMod <- modelFit$finalModel  
print(modelFit)

# Diagnostics - want chart to be as straight as possible at 0. If you see a lot of variation away from line it means there is some data not explained by model variables. If that’s the case you can then colour points by variables not used in model (residuals are difference between true values and fitted).  
plot(finMod,1,pch=19,cex=0.5,col="grey")  
ggplot() +  
 geom\_point(aes(x=finMod$fitted.values, y=finMod$residuals,colour=training$race))

# Plot by index (skewed with outliers probably variable missing)  
plot(finMod$residuals,pch=19)

# Predicted versus truth in test set (can use this after model has been built)  
pred <- predict(modelFit, testing)  
ggplot() +  
 geom\_point(data=testing, aes(x=wage, y=pred,colour=year))

### Polynomial Regression

Non-linear regression model, turns the line into a curve.

dataset$Level2 <- dataset$Level^2  
dataset$Level3 <- dataset$Level^3  
dataset$Level4 <- dataset$Level^4

poly\_reg <- lm(formula = Salary ~ .,  
 data = dataset)

# Predicting a new result with Polynomial Regression

y\_pred <- predict(poly\_reg, data.frame(Level = 6.5,  
 Level2 = 6.5^2,  
 Level3 = 6.5^3,  
 Level4 = 6.5^4))

### SVR Regression

Accounts for outliers & gives a best fitting line.

Penalty parameter C in SVM (the smaller C, the less outliers).

library(e1071)

regressor <- svm(formula = Salary ~ .,  
 data = dataset,  
 type = 'eps-regression')

### Decision Tree Regression

Non-linear and non-continuous.

Can have two independent variables and Y falls somewhere in the middle.

Algorithm splits into terminal leaves, takes averages for leaves and assigns the value to any predictions of where the value falls.

The whole point is to add more information into the system to better predict.



Decision Trees, be it Random Forest or GBM, handle messier data and messier relationships better than regression models. And there is seldom a dataset in the real world where relationships are not messy. No wonder you will seldom see a linear regression model outperforming RF or GBM.

library(rpart)

regressor <- rpart(formula = Salary ~ .,  
 data = dataset,  
 control = rpart.control(minsplit = 1))

y\_pred <- predict(regressor, data.frame(Level = 6.5))

library("ggplot2")

x\_grid = seq(min(dataset$Level), max(dataset$Level), 0.01)

ggplot() +  
 geom\_point(aes(x = dataset$Level, y = dataset$Salary),  
 colour = 'red') +  
 geom\_line(aes(x = x\_grid, y = predict(regressor, newdata = data.frame(Level = x\_grid))),  
 colour = 'blue') +

### Random Forest Regression

A form of ensemble learning – taking the same algorithm multiple times and putting them together. Much more accurate and stable than others as a change in dataset won’t affect a whole forest as much.



Random forests fit data better from the get-go without transforms.

They’re more forgiving in almost every way. You don’t need to scale your data, you don’t need to do any monotonic transformations (log etc). You often don’t even need to remove outliers.

You can throw in categorical features, and it’ll automatically partition the data if it aids the fit.

You don’t have to spend any time generating interaction terms.

And perhaps most important: in most cases, it’ll probably be notably more accurate.

library(randomForest)  
set.seed(1234)

# The [] gives a dataframe subset, whereas $ gives a vector.

regressor <- randomForest(x = dataset[1],  
 y = dataset$Salary,  
 ntree = 500)

y\_pred <- predict(regressor, data.frame(Level = 6.5))

library("ggplot2")

x\_grid = seq(min(dataset$Level), max(dataset$Level), 0.01)

ggplot() +  
 geom\_point(aes(x = dataset$Level, y = dataset$Salary),  
 colour = 'red') +  
 geom\_line(aes(x = x\_grid, y = predict(regressor, newdata = data.frame(Level = x\_grid))),  
 colour = 'blue') +

### Regularized Regression

Penalize or shrink large coefficients.

Pros: can help with bias/variance tradeoff & model selection. Cons: can be hard computationally on large data sets & doesn’t perform as well as random forest and boosting.

Use lasso in Caret for this.

modFit <- train(CompressiveStrength ~ ., data = training, method = "lasso")

plot.enet(modFit$finalModel, xvar = "penalty", use.color = TRUE) # Shows cement is last variable to be set to zero.

### R-Squared



But basically, R-squared tells us how good the best fitting regression line is in comparison to the average. The closer to 1 the better.

In caret, use postResample(pred, obs)

### Adjusted R-Squared

R-Squared would be bias for multiple linear regression, instead adjusted R-Squared has a penalisation factor for each variable added to make it fair.



### Interpreting Coefficients

If the coefficient is positive then your independent value is correlated to the dependent variable. So if you’re increasing then the dependent variable will be increasing. If it’s negative then it’s the opposite, as you increase your independent variable your dependent will decrease.

If you’re interpreting the magnitude you need to remember it’s in units of independent variable (e.g. you would say 1 has a greater impact than another per unit).

You can only interpret coefficients as the additional effect of variables with the others in place.

## Classification

Unlike regression where you predict a continuous number, you use classification to predict a category. Classification models include linear models like Logistic Regression and SVM; and non linear ones like K-NN, Kernel SVM, Naïve Bayes, Decision Tree and Random Forest.

Good way to start is plot variables with the variable you are trying to predict as the colour.

For classification it’s better to do feature scaling.

- Logistic Regression or Naive Bayes when you want to rank your predictions by their probability. For example if you want to rank your customers from the highest probability that they buy a certain product, to the lowest probability. Eventually that allows you to target your marketing campaigns. And of course for this type of business problem, you should use Logistic Regression if your problem is linear, and Naive Bayes if your problem is non-linear.

- SVM when you want to predict to which segment your customers belong to. Segments can be any kind of segments, for example some market segments you identified earlier with clustering.

- Decision Tree when you want to have clear interpretation of your model results,

- Random Forest when you are just looking for high performance with less need for interpretation.

List of caret models: <https://rdrr.io/cran/caret/man/models.html>



Naïve bayes is particularly useful when you have lots of categorical variables.

### Classification Template - SDS

setwd("C:/Users/rtutt/Desktop/R/Machine Learning/Machine Learning A-Z Template Folder/Part 3 - Classification/Section 14 - Logistic Regression")  
dataset <- read.csv("Social\_Network\_Ads.csv")  
dataset <- dataset[,3:5]

# Encoding the target feature as factor

dataset$Purchased <- factor(dataset$Purchased, levels = c(0,1))

# Split dataset

library('caTools')  
set.seed(123)  
split <- sample.split(dataset$Purchased, SplitRatio = 0.75)  
training\_set <- subset(dataset, split == TRUE)  
test\_set <- subset(dataset, split == FALSE)

# Feature Scaling (Obviously not on the categorical / dependent variable.)

training\_set[,1:2] = scale(training\_set[,1:2])  
test\_set[,1:2] = scale(test\_set[,1:2])

# Fitting Classifier to Training Set

… Create your classifier here

# Predicting the Test Set results

y\_pred <- predict(classifier, newdata = test\_set[,-3])

OR (where predictions are probabilities)

prob\_pred <- predict(classifier,   
 type = 'response',  
 newdata = test\_set[,1:2]) #can write as test\_set[-3]

y\_pred = ifelse(prob\_pred > 0.5, 1, 0)

# Evaluate predictions in confusion matrix

cm = table(test\_set[,3], y\_pred)

# Visualizing the Training Set results

library(ElemStatLearn)  
set = training\_set / test\_set

# Taking range of min / max + 1 so points are not squeezed. X1 = Age, X2 = Salary.  
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)  
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)  
grid\_set = expand.grid(X1, X2)  
colnames(grid\_set) = c('Age', 'EstimatedSalary')

# Plot the predictions.  
y\_grid = predict(classifier, type = 'response', newdata = grid\_set)  
OR  
prob\_set = predict(classifier, type = 'response', newdata = grid\_set)  
y\_grid = ifelse(prob\_set > 0.5, 1, 0)

plot(set[, -3],  
 main = 'Classification (Training set)',  
 xlab = 'Age', ylab = 'Estimated Salary',  
 xlim = range(X1), ylim = range(X2))  
contour(X1, X2, matrix(as.numeric(y\_grid), length(X1), length(X2)), add = TRUE)  
points(grid\_set, pch = '.', col = ifelse(y\_grid == 1, 'springgreen3', 'tomato'))  
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

### Logistic Regression

Binary result– predicting probability (p-hat) with a linear classifier, which you can also use as a score.

In logistic regression, you can understand the effect of the coefficients and compare which have a greater effect, but can’t quantify the effect of variables on dependent variables.

classifier <- glm(formula = Purchased ~ .,  
 family = binomial,  
 data = training\_set)

prob\_pred <- predict(classifier, type = 'response',newdata = test\_set[,1:2]) #can write as test\_set[-3]

y\_pred = ifelse(prob\_pred > 0.5, 1, 0)



--- Caret

modFit <- train(chd ~ age + alcohol + obesity + tobacco + typea + ldl, method = "glm", family = "binomial", data = trainSA)

missClass = function(values,prediction){sum(((prediction > 0.5)\*1) != values)/length(values)}

missClass(trainSA$chd, predict(modFit, newdata = trainSA))  
missClass(testSA$chd, predict(modFit, newdata = testSA))

### K-Nearest Neighbours

Non-linear classifier.

Step 1: Choose the number of K neighbours (usually 5). The value for k is generally chosen as the square root of the number of observations.  
Step 2: Take the K nearest neighbours of the new data point, according to the Euclidean distance.  
Step 3: Among these K neighbours, count the number of new data points in each category.  
Step 4: Assign then new data point to the category where you counted the most neighbours.  
Model is ready.

All variables including the dependent variable must be numeric, so must convert factors using is.numeric()



Fit the classifier & make predictions all in one:

library(class)  
y\_pred <- knn(train = training\_set[, -3], #independent variables  
 test = test\_set[, -3], #independent variables  
 cl = training\_set[, 3], #dependent variable  
 k = 5) #5 neighbours

# Have to change the y\_grid for visualizing KNN, should work for the others.

y\_grid = knn(train = training\_set[, -3],   
 test = grid\_set,  
 cl = training\_set[, 3],  
 k = 5)



### SVM

Linear classification.

Unlike the other algorithms, instead of looking at the cases furthest away from the boundary; it looks at the most extreme cases closest to the boundary to construct the analysis. E.g. values on each side that look most like the other.

SVM is like a sharp knife, it works on smaller datasets but is quite intensive on bigger datasets.

Works well in cases where the number of features is greater than the number of samples.



library(e1071)

classifier <- svm(formula = Purchased ~ .,  
 data = training\_set,  
 type = 'C-classification',  
 kernel = 'linear')

y\_pred <- predict(classifier, newdata = test\_set[,-3])

### Kernel SVM

For those that are not-linearly separable – circumference boundaries using the kernel trick which keeps the data in a 2D space.





classifier <- svm(formula = Purchased ~ .,  
 data = training\_set,  
 type = 'C-classification',  
 kernel = 'radial') #Gaussian kernel



### Naïve Bayes

Probabilistic based classifier.









Naïve Bayes assumes independence between features for model building.

Fit Gaussian distributions to the data and use those to draw decision boundary lines to assign probabilities.



Particularly useful when you have a large number of categorical variables.

modlda <- train(Species ~ ., data = training, method = "lda") #linear discriminant  
modnb <- train(Species ~ ., data = training, method = "nb") # naive bayes  
plda <- predict(modlda, testing)  
pnb <- predict(modnb, testing)  
table(plda,pnb) # results very similar

# Comparison of results  
equalPredictions <- (plda == pnb)  
ggplot() +  
 geom\_point(aes(x = Petal.Width, y = Sepal.Width, colour = equalPredictions), data = testing )

<https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>

### Decision Tree

You don’t have to take the tree all the way down to the terminal leaves, you can stop at any point and predict the probabilities from there.

Old method which aren’t too powerful on their own, now more popular with random forest and gradient boosting etc.

--- Caret

modFit <- train(Species ~ ., method = "rpart", data = training)  
print(modFit$finalModel) # can read model splits

plot(modFit$finalModel, uniform = TRUE,  
 main = "Classification Tree")  
text(modFit$finalModel, use.n = TRUE, all = TRUE, cex = 8)  
  
library(rattle)  
fancyRpartPlot(modFit$finalModel)

predict(modFit, newdata=testing)

# SDS Method

library(rpart)

classifier <- rpart(formula = Purchased ~ .,  
 data = training\_set)

y\_pred <- predict(classifier, newdata = test\_set[-3], type = "class")

# Visualizing (change y-grid but otherwise leave the rest the same)

y\_grid = predict(classifier, type = 'class', newdata = grid\_set)

# Plot decision tree  
plot(classifier)  
text(classifier)

### Random Forest

Ensemble method, runs decision tree method multiple times and then chooses the classification having the most votes.

Beware of overfitting, e.g. some lines jutting out in plot.

Very important to use cross validation to avoid overfitting, can use rfcv function to assess.

Can also be useful in data exploration to visualize which features are most important.





"Global" variable importance is the mean decrease of accuracy over all out-of-bag cross validated predictions, when a given variable is permuted after training, but before prediction. "Global" is implicit. Local variable importance is the mean decrease of accuracy by each individual out-of-bag cross validated prediction. Global variable importance is the most popular, as it is a single number per variable, easier to understand, and more robust as it is averaged over all predictions.

GINI importance measures the average gain of purity by splits of a given variable. If the variable is useful, it tends to split mixed labeled nodes into pure single class nodes.

library(randomForest)  
rfmodFit <- randomForest(mpg ~ ., data=mtcars, ntree=1000, keep.forest=FALSE,  
 importance=TRUE)  
# can also put: type = "prob" or “class”  
summary(rfmodFit)  
plot(rfmodFit, log="y")  
varImpPlot(rfmodFit)

rfpred <- predict(rfModFit, testing, type = “class”)  
rfpred <- predict(rfModFit, testing, type = “prob”) #gives you probabilities of each class, also good for multinomial classification.

--- Caret

fitControl <- trainControl(method = "repeatedcv", number = 4, repeats = 4)  
modFit <- train(Species ~ ., data = training, method = "rf",trControl = fitControl, prox=TRUE)  
modFit

plot(modFit$finalModel)  
text(modFit$finalModel)

# Look at single tree  
getTree(modFit$finalModel,k=2)

# Class "centers", then plot the centers against training data  
classC <- classCenter(training[,c(3,4)], training$Species, modFit$finalModel$prox)  
classC <- as.data.frame(classC)  
classC$Species <- rownames(classC)

p <- ggplot(aes(x=Petal.Width, y=Petal.Length, colour=Species), data = training) +  
 geom\_point()  
p + geom\_point(aes(x=Petal.Width, y=Petal.Length,colour=Species),size=5, shape=4, data=classC)

# Predicting new values and results table  
pred <- predict(modFit, testing)  
testing$predRight <- pred == testing$Species  
table(pred,testing$Species)

# Can also make predictions like this

predictRF <- predict(modelRF, testing, type = "class")

# Visualize errors  
ggplot()+  
 geom\_point(aes(x=Petal.Width, y=Petal.Length, colour=predRight),data=testing)

<https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/>

### Conditional Inference Tree

library(party)

ct <- ctree(Disabled ~ ., data = trainingrf)

plot(ct, type = "simple")  
plot(ct, main = "Conditional Inference Tree")

tr.pred = predict(ct, newdata=raw, type="prob")

<https://www.r-bloggers.com/a-brief-tour-of-the-trees-and-forests/>

### Evaluating Classification Models

False positive (type I error) – predicted result that didn’t occur  
False negative (type II error) – predict something won’t happen and it did. This is more dangerous.

Confusion matrix – top right is type I and bottom left is type II. Can deduce the accuracy/error rate from this. You shouldn’t just use the accuracy rate due to the accuracy paradox.

confusionMatrix(preds, obs)

E.g.   
confusionMatrix(pred1, vowel.test$y)$overall[1]  
confusionMatrix(pred2, vowel.test$y)  
predDf <- data.frame(pred1, pred2, y = vowel.test$y)  
# Get prediction agreement rate.  
dim(predDf)  
dim(predDf[predDf$pred1 == predDf$pred2,])

The default performance function used by train is postResample(pred, obs), which generates the accuracy and Kappa statistics.

Apply CAP curve to assess different models = Cumulative Accuracy Profile.

CAP analysis – draw line at 50% X axis: Aiming for above 70-90% on y axis; anything more is probably overfitting and less isn’t a great model.

library(MLmetrics)  
twoClassSummary(test\_set, lev = levels(test\_set$obs))

--- Example I’ve done, predictions must be 0 or 1:

library(ROCR)  
plot(performance(prediction(xgbpred, validatingdumm$DEPOSITING), measure = 'tpr', x.measure = 'fpr'), col = 2, lwd = 2, main = "ROC Curve")  
abline(a=0, b= 1, lty = 5, col = 4)  
abline(v = 0.5, lty = 3)  
abline(h = 0.7, lty = 3)  
abline(h = 0.9, lty = 3)

<https://www.r-bloggers.com/a-small-introduction-to-the-rocr-package/>

-- Look at actual predictions

full\_performance <- test\_h2o %>%  
 tibble::as\_tibble() %>%  
 select(Attrition) %>%  
 add\_column(EmployeeNumber = as.vector(testing2$EmployeeNumber)) %>%   
 add\_column(Prediction = as.vector(pred\_h2o$predict)) %>%  
 add\_column(No = formatC(as.vector(pred\_h2o$No),digits = 8, format = "f")) %>%  
 add\_column(Yes = formatC(as.vector(pred\_h2o$Yes),digits = 8, format = "f")) %>%  
 mutate\_if(is.character, as.factor) %>%   
 mutate(Result = ifelse(Attrition == Prediction, "Correct","Incorrect"))

full\_performance

write.csv(full\_performance, "fullperformance.csv")

## Model Tuning

trainControl for more control of your model – Resampling:  
Method:  
- boot = bootstrapping  
- cv = cross validation  
- repeatedcv = repeated cross validation  
- LOOCV = leave one out cross validation  
Number:  
- For cross validation  
- Number of subsamples to take  
Repeats:  
- Number of times to repeat subsampling  
- If big this can slow things down

#1  
fitControl <- trainControl(method = “repeatedcv”, number = 10, repeats = 4 #5 folds)

#2  
fitControl <- trainControl(  
 method = "boot",  
 number = 25,  
 savePredictions = "final",  
 #classProbs = TRUE, #gives probabilities with predictions too  
 index = createResample(training$DEPOSITING, 25)  
)

#3  
fitControl <- trainControl(method = "repeatedcv", repeats = 3, classProbs = TRUE, summaryFunction = twoClassSummary)

train(…., trControl = fitControl, metric = "ROC")

<https://cran.r-project.org/web/packages/caret/vignettes/caret.pdf>

## Model Selection

### k-Fold Cross Validation

The larger k = less bias, more variance; smaller k = more bias, less variance.

Splits the training set into 10 fold, trains it on 9 and tests it on 1 fold.

Obviously high variance is less accurate.



library(caret)

set.seed(32323)

folds <- createFolds(y=spam$type,k=10,  
 list=TRUE, returnTrain=TRUE)

sapply(folds, length)

folds[[1]][1:10]

--- OR

folds <- createFolds(training\_set$Purchased, k = 10) # creates list

cv <- lapply(folds, function(x) {  
 training\_fold = training\_set[-x,]  
 test\_fold = training\_set[x,]  
 classifier = svm(formula = Purchased ~ .,  
 data = training\_fold,  
 type = 'C-classification',  
 kernel = 'radial')  
 y\_pred = predict(classifier, newdata = test\_fold[-3])

cm = table(test\_fold[, 3], y\_pred)  
 accuracy = (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])  
 return(accuracy)  
})

accuracy <- mean(as.numeric(cv))

# In this case x is each one of these test folds  
# accuracy = taking accurate predictions / all predictions

## Improving Models

<https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/>

### Boosting

1. Take lots of (possibly) weak predictors.
2. Weight them and add them up
3. Get a stronger predictor

Starts by classifying the original data set and giving equal weights to each observation. If classes are predicted incorrect only then it gives higher weight to the miss classified observation iteratively.

Shows better predictive accuracy than bagging but tends to overfit.

Boosting can be used with any subset of classifiers.

Classification functions:  
> gbm – boosting with trees  
> mboost – model based boosting  
> ada – statistical boosting based on additive logistic regression.  
> gamBoost – for boosting with generalized additive models.

#### Gbm

Gbm tuning parameters:  
1. n.trees - number of iterations  
2. Interaction.depth – determines the complexity of the tree, i.e. total number of splits  
3. Shrinkage – refers to learning rate which determines the impact of each tree on the outcome. Lower values are generally preferred as they make the model robust to specific chareteristics of tree and allow it to generalize well. Lower values require higher number of trees to model all relations and can be computationally expensive.  
4. N.minobsinode – minimum number of training samples required in a node to perform splitting

--- SDS

modFit <- train(wage ~ ., method="gbm",data=training,verbose=FALSE)  
print(modFit)  
pred <- predict(modFit,data=testing)

# Plot (45 degree would be perfect)  
ggplot() +  
 geom\_point(aes(x=pred,y=training$wage))

--- Analytics Vidhya

fitControl <- trainControl(method = “cv”, number = 10, repeats = 4 #5 folds)  
tune\_Grid <- expand.grid(interaction.depth = 2, n.trees = 500, shrinkage = 0.1, n.minobsinnode = 10)

fit <- train(y\_train ~ ., data = train,  
 method = "gbm",  
 trControl = fitControl,  
 verbose = FALSE,  
 tuneGrid = gbmGrid)

predicted= predict(fit,test,type= "prob")[,2]

#### XGBoost

Especially good for large datasets.

Most powerful implementation of gradient boosting in terms of model performance, execution speed and you can keep the interpretation of your model as no feature scaling is required.

Only works on numeric vectors, you can convert categorical variables into numeric vectors using one hot encoding.

library(dummies)  
trainingdumm <- dummy.data.frame(training, names=c("MethodMostUsed","DeviceOperatingSystem","Platform","Country","Hooped","MarketingType","GameTypeMostPlayed"), sep="\_")

xgbModFit <- xgboost(data = data.matrix(trainingdumm[,-1]),  
 label = trainingdumm$DEPOSITING,  
 nrounds = 25)

xgbpred <- predict(xgbModFit, data.matrix(validatingdumm[,-1]))  
xgpred <- ifelse(xgpred > 0.5, 1, 0)

# Need to turn it into number vector for things like ROCR.

xgb.plot.tree(feature\_names = colnames(training), model = xgModFit, n\_*f*irst\_tree = 1, plot\_width = 2000, plot\_height = 2000)

xgb.plot.multi.trees(model = xgModFit, feature\_names = colnames(training), features\_keep = 3)

importancematrix <- xgb.importance(feature\_names = colnames(training[,-1]), model = xgModFit)  
xgb.plot.importance(importancematrix)  
xgb.ggplot.importance(importancematrix, top\_n = 10)  
xgb.ggplot.importance(importancematrix, measure = "Frequency", rel\_to\_first = TRUE)  
# Setting rel\_to\_first = TRUE allows to see the picture from the perspective of "what is feature's importance contribution relative to the most important feature?"

--- SDS

library(xgboost)

classifier <- xgboost(data = as.matrix(training\_set[-11]),  
 label = training\_set$Exited,  
 nrounds = 10)

mod\_xg <- xgboost(data = data.matrix(training[,-17]),  
 label = training$DEPOSITING,  
 nrounds = 10)

# only the features not dependent variable as a matrix.  
# label is the dependent variable as a vector.  
# nrounds is the number of iterations.

# Evaluate the XGBoost model

library(caret)

folds <- createFolds(training\_set$Exited, k = 10) # creates list

cv <- lapply(folds, function(x) {  
 training\_fold = training\_set[-x,]  
 test\_fold = training\_set[x,]  
 classifier <- xgboost(data = as.matrix(training\_set[-11]),  
 label = training\_set$Exited,  
 nrounds = 10)  
 y\_pred = predict(classifier, newdata = as.matrix(test\_fold[-11]))  
 y\_pred = (y\_pred >= 0.5) # not sure if it's ifelse(y\_pred >= 0.5, 1, 0)  
 cm = table(test\_fold[, 11], y\_pred)  
 accuracy = (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])  
 return(accuracy)  
})

<https://www.kaggle.com/nschneider/gbm-vs-xgboost-vs-lightgbm>

<https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/>

<https://cran.r-project.org/web/packages/xgboost/vignettes/discoverYourData.html>

<http://xgboost.readthedocs.io/en/latest/parameter.html>

<https://www.r-bloggers.com/an-introduction-to-xgboost-r-package/>

<https://rdrr.io/cran/xgboost/man/xgb.plot.importance.html>

#### Ada

adaFit <- train(DEPOSITING ~ ., data = training, method = "ada")

### Bagging (Bootstrap Aggregating)

Basic idea is averaging models together for smoother fit by combining the results of multiple classifiers modelled on different sub-samples of the same data set.

1. Create multiple datasets – the new datasets can also have a fraction of the columns as well as rows.
2. Build multiple classifiers – generally the same classifier is modelled on each data set and predictions are made.
3. Combine the predictions of all classifiers with an average or majority vote to make a more robust model.



Similar bias but reduced variance. Most useful for non-linear predictions.

The train function does this for you, using ctreeBag.

### Ensembling/Stacking

Combining predictors improves classifiers by averaging/voting which improves accuracy but reduces interpretability. Boosting, bagging and random forests are variants on this theme.

Approaches for combining classifiers:  
1. Bagging, boosting and random forests.  
2. Combing different classifiers – model stacking and ensembling.

This can be computationally intensive.

library(ISLR)  
library(ggplot2)  
library(caret)

Wage <- subset(Wage, select = - c(logwage)) #get rid of this column.

# Create validation set (30%), then testing (70%) and training (30%) sets from remaining data.

inBuild <- createDataPartition(y = Wage$wage,   
 p = 0.7, list = FALSE)

validation <- Wage[-inBuild,]

buildData <- Wage[inBuild,]

inTrain <- createDataPartition(y = buildData$wage,  
 p = 0.7, list = FALSE)

training <- buildData[inTrain,]  
testing <- buildData[-inTrain,]

# Build two different models

mod1 <- train(wage ~ ., method = "glm", data = training)  
mod2 <- train(wage ~ ., method = "rf", data = training,  
 trControl = trainControl(method = "cv"), number = 3)

# Predicting on testing set

pred1 <- predict(mod1, testing)  
pred2 <- predict(mod2, testing)

qplot(pred1, pred2, colour=wage, data = testing)  
ggplot(data = testing, aes(x = pred1, y = pred2, colour = wage)) +  
 geom\_point()

# Fit a model that combines the predictors. Build dataset of predictions with results. Then build regression model relating wage variable to the two predictions. Then can predict from combined dataset on new models.   
# gam – generalized additive model

predDF <- data.frame(pred1, pred2, wage = testing$wage)  
combModFit <- train(wage ~ ., method = "gam", data = predDF)  
combPred <- predict(combModFit,predDF)

# Testing errors

sqrt(sum((pred1 - testing$wage)^2))  
sqrt(sum((pred2 - testing$wage)^2))  
sqrt(sum((combPred - testing$wage)^2))

# Predict on validation data set

pred1V <- predict(mod1, validation)  
pred2v <- predict(mod2, validation)  
predVDF <- data.frame(pred1 = pred1V, pred2 = pred2v)  
combpredV <- predict(combModFit, predVDF)

sqrt(sum(combpredV - validation$wage)^2)

--- Another example of combining three model results predictions

predDF <- data.frame(pred\_rf, pred\_gbm, pred\_lda, diagnosis = testing$diagnosis)  
combModFit <- train(diagnosis ~ ., method = "rf", data = predDF)  
combPred <- predict(combModFit, predDF)

confusionMatrix(pred\_rf, testing$diagnosis)$overall[1   
confusionMatrix(pred\_gbm, testing$diagnosis)$overall[1]  
confusionMatrix(pred\_lda, testing$diagnosis)$overall[1]  
confusionMatrix(combPred, testing$diagnosis)$overall[1]

<https://mlwave.com/kaggle-ensembling-guide/>

<http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/>

### Caret Ensemble

Used for stacking.

modControl <- trainControl(  
 method = "boot",  
 number = 25,  
 savePredictions = "final",  
 #classProbs = TRUE, #gives probabilities with predictions too  
 index = createResample(training$DEPOSITING, 25)  
)

modList <- caretList(DEPOSITING ~ ., data= training,  
 trControl = modControl,  
 methodList = c("glm", "rpart"))

p <- as.data.frame(predict(modList, newdata = validating))  
print(p)

modelCor(resamples(modList))

confusionMatrix(p$rpart, validating$DEPOSITING)

greedy\_ensemble <- caretEnsemble(  
 modList,   
 metric="ROC",  
 trControl=trainControl(  
 number=2,  
 summaryFunction=twoClassSummary,  
 classProbs=TRUE  
 ))  
summary(greedy\_ensemble)

<https://cran.r-project.org/web/packages/caretEnsemble/vignettes/caretEnsemble-intro.html>

## H2O

All computations are performed (in highly optimized Java code) in the H2O cluster and initiated by REST calls from R.

So models are trained in the H2O cluster in RAM, no limit on cluster size can keep adding nodes. Under the hood the rows are split across machine.

1:08:00 - <https://www.youtube.com/watch?v=9nHXyLR8XEw>

<https://www.stat.berkeley.edu/~ledell/docs/h2o_hpccon_oct2015.pdf>

### Auto\_ml

# Initialize the Java Virtual Machine (JVM) that H2O uses locally and turn off progress bars.

h2o.init()  
h2o.no\_progress()

# Splitting Dataset

data\_h2o <- as.h2o(dataset)  
split\_h2o <- h2o.splitFrame(data\_h20, c(0.7, 0.15), seed = 1234)

train\_h2o <- h2o.assign(split\_h20[[1]], "train") # 70%  
valid\_h2o <- h2o.assign(split\_h20[[2]], "valid") # 15%  
test\_h2o <- h2o.assign(split\_h20[[3]], "test") # 15%

# Modelling (set names & run automated machine learning)

y <- "Purchasing"  
x <- setdiff(names(train\_h2o), y)

automl\_models\_h2o <- h2o.automl(  
 x = x,  
 y = y,  
 training\_frame = train\_h2o,  
 leaderboard\_frame = valid\_h2o,  
 max\_runtime\_secs = 3600 # default 1 hour  
)

automl\_models\_h2o@leaderboard  
automl\_models\_h2o@leader

model <- automl\_models\_h2o@leader

# Saving and loading model

model\_path <- h2o.saveModel(object = model, path = getwd(), force = TRUE)  
print(model\_path)  
model <- h2o.loadModel("T:\\...”)

# Predictions and testing model

pred\_h2o <- h2o.predict(object = model, newdata = test\_h2o)

test\_performance <- test\_h2o %>%  
 tibble::as\_tibble() %>%  
 select(Purchasing) %>%  
 add\_column(pred = as.vector(pred\_h2o$predict)) %>%  
 mutate\_if(is.character, as.factor)

confusion\_matrix <- test\_performance %>%  
 table()   
confusion\_matrix

tn <- confusion\_matrix[1]  
tp <- confusion\_matrix[4]  
fp <- confusion\_matrix[3]  
fn <- confusion\_matrix[2]

accuracy <- (tp + tn) / (tp + tn + fp + fn)  
misclassification\_rate <- 1 - accuracy  
recall <- tp / (tp + fn)  
precision <- tp / (tp + fp)  
null\_error\_rate <- tn / (tp + tn + fp + fn)

tibble(  
 accuracy,  
 misclassification\_rate,  
 recall,  
 precision,  
 null\_error\_rate  
) %>%   
 transpose()

# H2o Performance

perf <- h2o.performance(model, test\_h2o)  
h2o.confusionMatrix(perf)  
h2o.accuracy(perf)  
h2o.tpr(perf)

# Creating table with predictions and probabilities.

full\_performance <- test\_h2o %>%  
 tibble::as\_tibble() %>%  
 select(Attrition) %>%  
 add\_column(EmployeeNumber = as.vector(testing2$EmployeeNumber)) %>%   
 add\_column(Prediction = as.vector(pred\_h2o$predict)) %>%  
 add\_column(No = formatC(as.vector(pred\_h2o$No),digits = 8, format = "f")) %>%  
 add\_column(Yes = formatC(as.vector(pred\_h2o$Yes),digits = 8, format = "f")) %>%  
 mutate\_if(is.character, as.factor) %>%   
 mutate(Result = ifelse(Attrition == Prediction, "Correct","Incorrect"))

write.csv(full\_performance, "fullperformance.csv")

library(ROCR)

h2opred <- ifelse(as.data.frame(pred\_h2o$predict) == 'Yes',1,0)  
h2otest <- ifelse(as.data.frame(test\_h2o$Attrition) == 'Yes',1,0)

plot(performance(prediction(h2opred, h2otest), measure = 'tpr', x.measure = 'fpr'), col = "red", lwd = 2, main = "H2O ROC")  
abline(a=0, b= 1, lty = 5, col = 4)  
abline(v = 0.5, lty = 3)  
abline(h = 0.7, lty = 3)  
abline(h = 0.9, lty = 3)

<http://www.business-science.io/business/2017/09/18/hr_employee_attrition.html>

<http://s3.amazonaws.com/h2o-release/h2o/master/3874/docs-website/h2o-docs/automl.html>

<https://stackoverflow.com/questions/43054383/r-h2o-how-can-i-get-my-trained-model-predictions-probabilities>

<https://stackoverflow.com/questions/41075416/how-to-interpret-results-of-h2o-predict>

<https://groups.google.com/forum/#!topic/h2ostream/rRPcZiCzUQs>

### Rank Driving Features (Lime)

# H2O

class(model)

# Function tells lime() what model type we are dealing with classification', 'regression', 'survival', 'clustering', 'multilabel', etc’ x is our h2o model

model\_type.H2OBinomialModel <- function(x, ...) {  
 return("classification")  
 }

predict\_model.H2OBinomialModel <- function(x, newdata, type, ...) {  
 pred <- h2o.predict(x, as.h2o(newdata))  
 return(as.data.frame(pred[,-1]))  
}

predict\_model(x = model, newdata = as.data.frame(test\_h2o[,-1]), type = 'raw') %>%  
 tibble::as\_tibble()

# Explainer & Explain ( n labels = 1 is single class)

explainer <- lime::lime(  
 as.data.frame(train\_h2o[,-1]),   
 model = model,   
 bin\_continuous = FALSE)

explanation <- lime::explain(  
 as.data.frame(test\_h2o[1:10,-1]),   
 explainer = explainer,   
 n\_labels = 1,   
 n\_features = 4)

# Feature Importance Plot

plot\_features(explanation) +  
 labs(title = "Customer Attrition Predictive Analytics: LIME Feature Importance Visualization",  
 subtitle = "Hold Out (Test) Set, First 10 Cases Shown")

<http://www.business-science.io/business/2017/09/18/hr_employee_attrition.html>

### Grid Search

Grid tuning in h2o - <https://blog.h2o.ai/2016/06/h2o-gbm-tuning-tutorial-for-r/>

## Model Metrics

In the ROC curve we look at:  
TPR (True Positive Rate) = # True positives / # positives = Recall = TP / (TP+FN)  
FPR (False Positive Rate) = # False Positives / # negatives = FP / (FP+TN)

Precision and recall are:  
Precision =# True positives / # predicted positive = TP/(TP+FP)  
Recall = # True positives / # positives = TP / (TP+FN)

Recall and True Positive Rate (TPR) are exactly the same.

Precision measures the probability of a sample classified as positive to actually be positive.

Precision is more focused in the positive class than in the negative class, it actually measures the probability of correct detection of positive values, while FPR and TPR (ROC metrics) measure the ability to distinguish between the classes.

<https://towardsdatascience.com/what-metrics-should-we-use-on-imbalanced-data-set-precision-recall-roc-e2e79252aeba>

## Association Rule Learning

### Apriori

Rules where when one thing happens often another does; so people who did something also did something else (e.g. watched movie).

Step 1: Set a minimum support and confidence (e.g. maybe you want to limit to 20%).  
Step 2: Take all the subsets of transactions having higher support than minimum support.  
Step 3: Take all the rules of these subsets having higher confidence than minimum confidence  
Step 4: Sort the rules by decreasing lift.

What support and confidence you select depends on the business problem. Support relates to frequency of item and confidence relates to how often the rule occurs in the cases. Can see the support from the frequency plot and it is important because if it’s too low then items with low frequency or revenue will be considered for the rules.  
E.g. we only want products purchased 3x per day: 3x7 = 21/7500(total txns)  
Confidence is arbitrary so make this choice with default and then reduce. Don’t want it too small or you’ll get nonsense rules. Default is 0.8 which is very high as it would mean the rules need to be correct on at least 80% of the transactions.

One of the things to consider is some items will be associated just because of the support (e.g. higher frequency items appearing in multiple rules where it doesn’t necessarily make sense).

Arules library only takes sparse matrix – matrix contains mainly 0s – attribute 1 column for each attribute.  
E.g. customer1 – 0 – 1 – 0 – 1 – 0 – 0 – 0; with the item headings above.

library(arules)

dataset <- read.transactions('Market\_Basket\_Optimisation.csv', sep = ',', rm.duplicates = TRUE)  
 #sep = ',' is auto in read.csv but not read.transactions; rm.duplicates will remove duplicate values

summary(dataset)  
itemFrequencyPlot(dataset, topN = 10) # top 10 sold items.

rules <- apriori(data = dataset, parameter = list(support = 0.004, confidence = 0.2))

inspect(sort(rules, by = 'lift')[1:10])

Often use this algorithm with other association rules and recommendation techniques like collaborative filtering, neighbourhood model and latent factor models.

<https://www.r-bloggers.com/association-rule-learning-and-the-apriori-algorithm/>

## Forecasting

training = dat[year(dat$date) < 2012,]  
testing = dat[(year(dat$date)) > 2011,]

tstrain = ts(training$visitsTumblr)

library(forecast)

mod\_ts <- bats(tstrain)  
fcast <- forecast(mod\_ts, level = 95, h = dim(testing)[1])

sum(fcast$lower < testing$visitsTumblr & testing$visitsTumblr < fcast$upper) / dim(testing)[1]

## Natural Language Processing

<https://www.analyticsvidhya.com/blog/2014/05/build-word-cloud-text-mining-tools/>

<https://www.datasciencecentral.com/profiles/blogs/deep-learning-for-natural-language-processing-tutorials-with?utm_content=buffer9523d&utm_medium=social&utm_source=linkedin.com&utm_campaign=buffer>

<https://www.tidytextmining.com/tidytext.html>

## Web Scraping

str\_split()

library(rvest)

path <- <http://varianceexplained.org/r/mixture-models-baseball/>

html\_code\_text <- read\_html(path) %>%   
 html\_nodes("code") %>%   
 html\_text()

library(purrr) #Scale the scraping to multiple pages on a site

posts\_path <- <http://varianceexplained.org/posts/>

# Extract the post titles  
titles\_vec <- read\_html(posts\_path) %>%  
 html\_node("#main") %>%  
 html\_nodes("article") %>%  
 html\_nodes("a") %>%  
 html\_text(trim = TRUE)

# Extract the post dates  
dates\_vec <- read\_html(posts\_path) %>%  
 html\_node("#main") %>%  
 html\_nodes("article") %>%  
 html\_nodes("p.dateline") %>%  
 html\_text(trim = TRUE) %>%  
 mdy()

# Extract the post hrefs  
hrefs\_vec <- read\_html(posts\_path) %>%  
 html\_node("#main") %>%  
 html\_nodes("article") %>%  
 html\_nodes("a") %>%  
 html\_attr("href")

# Bind the data together in a tibble  
variance\_explained\_tbl <- bind\_cols(  
 title = titles\_vec,   
 date = dates\_vec,   
 href = hrefs\_vec)

Function from Matt Dancho to split the text:

parse\_function\_names <- function(text, stop\_words = c("")) {  
 parser <- function(text, stop\_words) {  
 ret <- text %>%  
 str\_c(collapse = " ") %>%  
 str\_split("\\(") %>%  
 set\_names("text") %>%  
 as.tibble() %>%  
 slice(-n()) %>%  
 mutate(str\_split = map(text, str\_split, " ")) %>%  
 select(-text) %>%  
 unnest() %>%  
 mutate(function\_name = map\_chr(str\_split, ~ purrr::pluck(last(.x)))) %>%  
 select(function\_name) %>%  
 separate(function\_name, into = c("discard", "function\_name"),   
 sep = "(:::|::|\n)", fill = "left") %>%  
 select(-discard) %>%  
 mutate(function\_name = str\_replace\_all(function\_name,   
 pattern = "[^[:alnum:]\_\\.]", "")) %>%  
 filter(!(function\_name %in% stop\_words))   
 return(ret)  
 }  
 safe\_parser <- possibly(parser, otherwise = NA)  
 safe\_parser(text, stop\_words)  
}

Good tutorial from Matt Dancho:  
<http://www.business-science.io/learning-r/2018/03/03/how_to_learn_R_pt1.html>

<http://blog.rstudio.com/2014/11/24/rvest-easy-web-scraping-with-r/>

<https://www.analyticsvidhya.com/blog/2017/03/beginners-guide-on-web-scraping-in-r-using-rvest-with-hands-on-knowledge/>

<https://towardsdatascience.com/web-scraping-tutorial-in-r-5e71fd107f32>

<https://www.datacamp.com/community/tutorials/r-web-scraping-rvest>

## Sources

<https://www.analyticsvidhya.com/blog/2015/09/perfect-build-predictive-model-10-minutes/>

<https://www.analyticsvidhya.com/blog/2015/09/complete-guide-boosting-methods/>

# **R: Basics (In-Depth)**

## Basics

Variables: my\_variable -< … or c(…,…,…)

Class() gives you the data type of the variable.

Names(vector) <- c(“…”,”…”)

Rnorm() gives random even numbers  
#--- -2 --- -1 --- 0 --- 1 --- 2

**Vectors** (one dimensional array): can hold numeric, character or logical values. The elements in a vector all have the same data type.

**Matrices** (two dimensional array): can hold numeric, character or logical values. The elements in a matrix all have the same data type.

**Data frames** (two-dimensional objects): can hold numeric, character or logical values. Within a column all elements have the same data type, but different columns can be of different data type.

## Variables & Data Types

#Integer - use this when you know you're not going to be performing arithmetic   
# (e.g. just using for 1st, 2nd, 3rd)  
x <- 2L  
typeof(x)

#Double  
y <- 2.5  
typeof(y)

#Complex  
z <- 3 + 2i  
typeof(z)

#Character  
a <- "h"  
typeof(a)

#Logical  
q1 <- T  
typeof(q1)  
q2 <- F  
typeof(q2)

## Functions

gsub() and sub() – they look for a pattern and replace it. Sub is only first instance and gsub is all instances. E.g. fin$Expenses <- gsub(" Dollars", "", fin$Expenses)  
To replace special character like $ you must use “\\$”

c() #compile

is.numeric() | is.integer() | is.double() | is.character() | typeof()

rnorm() | seq() | rep() | print() | paste()

rnorm(n=5,mean=10,sd=8) #68% would fall between 2 and 18

seq(from=10,to=20,by=3) | seq(from=10,to=20,length.out = 100) | seq(from=10,to=20,along.with=) #alongwith makes length same as vector

round() | mean() | sqrt() | max() | min()

## Loops & Conditional Statements

#### IF Statements

if (condition1) {  
 expr1  
} else if (condition2) {  
 expr2  
} else if (condition3) {  
 expr3  
} else {  
 expr4

}E.g.  
rm(answer)  
x <- rnorm(1) >  
if(x > 1){  
 answer <- "Greater than 1"  
} else if(x >= -1) {  
 answer <- "Between -1 and 1"  
 } else {  
 answer <- "Less than -1"  
}  
answer

#### Loops

counter <- 1  
while(counter<12)  
{  
 print(counter)  
 counter <- counter + 1  
}  
for(i in 1:5){  
 print("Hello R")  
}

#R-specific programming loop  
# Loop iterating over vector values.  
for(i in x){  
 print(i)  
}

#Conventional programming loop  
for(j in 1:5){  
 print(x[j])  
}

# Vectorized approach (alot easier)  
c <- a \* b

#De-vectorized approach  
d <- rep(NA,N) #100 empty spaces  
for(i in 1:N){  
 d[i] <- a[i] \* b[i]  
}

## Vectors

Vectors are one-dimension arrays that can hold numeric data, character data, or logical data. In other words, a vector is a simple tool to store data.  
In R, you create a vector with the combine function c(). You place the vector elements separated by a comma between the parentheses. For example:  
numeric\_vector <- c(1, 2, 3)  
character\_vector <- c("a", "b", "c")

You can give a name to the elements of a vector with the names() function. Have a look at this example:  
some\_vector <- c("John Doe", "poker player")  
names(some\_vector) <- c("Name", "Profession")  
Can also uses vectors for days of the week and assign them to the days. E.g.   
# Roulette winnings from Monday to Friday  
roulette\_vector <- c(-24, -50, 100, -350, 10)  
# The variable days\_vector  
days\_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")  
# Assign the names of the day to roulette\_vector and poker\_vector  
names(roulette\_vector) <- c(days\_vector)

sum(roulette\_vector,poker\_vector)

poker\_vector[3] – chooses column 3 of the records; poker\_vector[c(1, 5)] – chooses the first and fifth; poker\_vector[2:5]; poker\_vector[c("Monday","Tuesday")]

## Logical Comparison Operators

The (logical) comparison operators known to R are:  
< for less than  
> for greater than  
<= for less than or equal to  
>= for greater than or equal to  
== for equal to each other  
!= not equal to each other

# Which days did you make money on poker?  
selection\_vector <- poker\_vector > 0  
# Select from poker\_vector these days  
poker\_winning\_days <- poker\_vector[selection\_vector]

#Logical  
#True T  
#False F  
# ==  
# !=  
# <  
# >  
# <=  
# >=  
# !  
# |  
# &  
# isTRUE(x)

result <- 4 < 5  
result2 <- !(5 > 1)  
# Or (so at least one has to be true)  
result | result2  
result & result2  
isTRUE(result)

TRUE is seen as greater than FALSE.

You can compare all elements in two vectors.

Can use multiple operators - &, I(or), !

## Matrices

In R, a matrix is a collection of elements of the same data type (numeric, character, or logical) arranged into a fixed number of rows and columns. Since you are only working with rows and columns, a matrix is called two-dimensional.

You can construct a matrix in R with the matrix() function.

E.g. matrix(1:9, byrow = TRUE, nrow = 3)

E.g. Matrix(data = NA, nrow=1,ncol=1,byrow=FALSE,dimensions =NULL)

# Box office Star Wars (in millions!)  
new\_hope <- c(460.998, 314.4)  
empire\_strikes <- c(290.475, 247.900)  
return\_jedi <- c(309.306, 165.8)  
# Construct matrix  
star\_wars\_matrix <- matrix(c(new\_hope, empire\_strikes, return\_jedi), nrow = 3, byrow = TRUE)  
# Vectors region and titles, used for naming  
region <- c("US", "non-US")  
titles <- c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")  
# Name the columns with region  
colnames(star\_wars\_matrix) <- region  
# Name the rows with titles  
rownames(star\_wars\_matrix) <- titles  
# Print out star\_wars\_matrix  
star\_wars\_matrix  
# Calculate sums  
worldwide\_vector <- **rowSums**(star\_wars\_matrix)  
worldwide\_vector <- **colSums**(star\_wars\_matrix)

#### Building Matrices

**Cbind()** Merges matrixes together by column, e.g. adds a new column after doing rowSums  
**Rbind()** Merges matrixes together by row, e.g. adds a new row after doing colSums

Matrix(),Rbind(), cbind(), rownames() <- , colnames() <-

Matplot()

Games[1,drop=F] #When just subsetting one row, need to specify if you want to return matrix instead of vector

E.g.  
MinutesPlayed <- rbind(KobeBryant\_MP, JoeJohnson\_MP, LeBronJames\_MP, CarmeloAnthony\_MP)  
rm(KobeBryant\_MP, JoeJohnson\_MP, LeBronJames\_MP, CarmeloAnthony\_MP) #Removes vectors  
colnames(MinutesPlayed) <- Seasons  
rownames(MinutesPlayed) <- Players

#### Subsetting

Similar to vectors, you can use the square brackets [ ] to select one or multiple elements from a matrix. Whereas vectors have one dimension, matrices have two dimensions. You should therefore use a comma to separate that what to select from the rows from that what you want to select from the columns. For example:  
my\_matrix[1,2] selects the element at the first row and second column.  
my\_matrix[1:3,2:4] results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3, 4.  
my\_matrix[c(1,3),] shows 2nd and 3rd row, all columns.

#### Naming Dimensions

temp.vec <- rep(c("a","b","Zz"), each=3)  
temp.vec  
Bravo <- matrix(temp.vec,3,3)  
Bravo  
rownames(Bravo) <- c("How","Are","You")  
colnames(Bravo) <- c("X","y","z")  
Bravo  
Bravo["Are","y"] <- 0  
Bravo  
rownames(Bravo) <- NULL

#### Visualizing With Matplot

# Visualizing Subsets  
Data <- MinutesPlayed[1:3,]  
matplot(t(Data), type="b", pch=15:18, col=c(1:4,6))  
legend("bottomleft", inset=0.01, legend=Players[1:3], col=c(1:4,6), pch=15:18, horiz=F)

#### *Create Function*

(Passing in Parameters)  
myplot <- function(data, rows=1:10){   
Data <- data[rows,]  
matplot(t(Data), type="b", pch=15:18, col=c(1:4,6))  
legend("bottomleft", inset=0.01, legend=Players[rows], col=c(1:4,6), pch=15:18, horiz=F)

}

## Data Frames

Although matrices are all the same data types, data frames allow data sets of different data types, basically tables made of different data types.

Working with large data sets is not uncommon in data analysis. When you work with (extremely) large data sets and data frames, your first task as a data analyst is to develop a clear understanding of its structure and main elements. Therefore, it is often useful to show only a small part of the entire data set.

#### Constructing Data Frame

You construct a data frame with the data.frame() function. As arguments, you pass the vectors from before: they will become the different columns of your data frame. Because every column has the same length, the vectors you pass should also have the same length. But don't forget that it is possible (and likely) that they contain different types of data.  
E.g. data.frame(name,type,diameter,rotation,rings)

# Constructing DF  
mydf<- data.frame(Countries\_2012\_Dataset, Codes\_2012\_Dataset, Regions\_2012\_Dataset)  
colnames (mydf) <- c("Country","Code","Region")

#-- Can pass names right at the start  
mydf <- data.frame(Country=Countries\_2012\_Dataset, Code=Codes\_2012\_Dataset, Region=Regions\_2012\_Dataset)

#### Exploring DataSet

Nrow() number of rows.

Ncol() number of columns.

Head(…, n=10) first 10 rows, defaults to 6.

Tail() bottom 6 rows.

Str():   
The total number of observations (e.g. 32 car types)  
The total number of variables (e.g. 11 car features)  
A full list of the variables names (e.g. mpg, cyl ... )  
The data type of each variable (e.g. num)  
The first observations

Applying the str() function will often be the first thing that you do when receiving a new data set or data frame. It is a great way to get more insight in your data set before diving into the real analysis.

Summary()

Levels(Dataframe$column)

#### Using $ sign

Columns have names but rows are seen as numbers. E.g. can do stats[3, “Birth Rate”]

Can use $ sign to extract whole column or column with specific row, e.g. stats$Birth.Rate[3]

#### Basic Operations

**Subsetting**  
stats[1:10,]  
stats[c(4,100),]

Extracting one row in data frame will maintain DF rather than a vector. However, if you want one column you need to specify to keep DF – e.g. stats[,1,drop=F]

**Add columns**: stats$MyCalc <- stats$Birth.rate \* stats$Internet.users

**Remove columns:** stats$MyCalc <- NULL

You should see the subset() function as a short-cut to do exactly the same as what you did in the previous exercises.  
subset(my\_df, subset = some\_condition)

In data analysis you can sort your data according to a certain variable in the data set. In R, this is done with the help of the function order().  
a <- c(100,10,1000)  
a[order(100,10,1000)] = 2,1,3  
a[order(a)] = 10 100 1000  
---  
Example, ordered by diameter column:  
positions <- order(planets\_df$diameter)  
planets\_df[positions, ]  
OR  
planets\_df[order(planets\_df$diameter),]

#### Filtering

stats$Internet.users < 2 # Gives True/False  
filter <- stats$Internet.users < 2 # put it into vector  
stats[filter,]

OR can just put straight into subset

stats[stats$Internet.users < 2,]  
stats[stats$Birth.rate > 40 & stats$Internet.users < 2,]  
stats[stats$Income.Group == "High income",]

#### Merging Data Frames

merged <- merge(stats, mydf, by.x = "Country.Code", by.y = "Code")

To delete a column: merged$Country <- NULL

#### Introducing qplot

library(ggplot2)  
?qplot

qplot(data=stats, x=Internet.users)  
qplot(data=stats, x=Income.Group, y = Birth.rate) # categorical variable on x axis = income group, numeric variable on y axis = Birth.rate   
qplot(data=stats, x=Income.Group, y = Birth.rate, size = I(3),colour=I("blue"),geom="boxplot")

#Visualizing what we need

qplot(data=stats, x=Internet.users, y=Birth.rate,size = I(4),colour=I("red"))  
qplot(data=stats, x=Internet.users, y=Birth.rate,size = I(4),colour=Income.Group) # Can map a column to a colour!

#Data, colour, shapes, transparency, title.

qplot(data=merged,   
 x=Internet.users,   
 y=Birth.rate,   
 colour=Region,  
 shape=I(19),  
 alpha=I(0.6),   
 main="Birth rate vs Internet users")

## Factors

Statistical data type storing categorical variables.

Important to define what data types you want – need to convert non factors to factors and vice versa.

E.g. convert non-factor to factor - movies$Year <- factor(movies$Year).

dataset$Country <- factor(dataset$Country,  
 levels = c('France', 'Spain', 'Germany'),  
 labels = c(1,2,3))

Factor Variable Trap (converting factors to non-factors) – R deals with the factor levels as factorized numbers. So when converting back, you must convert it to a character first.  
E.g. as.numeric(as.character(z))

The difference between a categorical variable and a continuous variable is that a categorical variable can belong to a limited number of categories. A continuous variable, on the other hand, can correspond to an infinite number of values.

There are two types of categorical variables: a nominal categorical variable and an ordinal categorical variable. A nominal variable is a categorical variable without an implied order (e.g. different animals “Elephant”, “Giraffe”, “Donkey”. In contrast, ordinal variables do have a natural ordering (e.g. temperatures “Low”, “Medium” and “High”).

When you first get a data set, you will often notice that it contains factors with specific factor levels. However, sometimes you will want to change the names of these levels for clarity or other reasons. R allows you to do this with the function levels(): levels(factor\_vector) <- c("name1", "name2",...).

Flow: Vector – Factor to give categories of results – Levels of factor and re-naming.

# Create factor\_speed\_vector  
speed\_vector <- c("fast", "slow", "slow", "fast", "insane")  
factor\_speed\_vector <- factor(speed\_vector, ordered = TRUE, levels = c("slow", "fast", "insane"))

## Lists

A list in R allows you to gather a variety of objects under one name (that is, the name of the list) in an ordered way. These objects can be matrices, vectors, data frames, even other lists, etc. It is not even required that these objects are related to each other in any way.

You could say that a list is some kind super data type: you can store practically any piece of information in it!

E.g. my\_list <- list(my\_vector,my\_matrix,my\_df)

Naming Lists:  
my\_list <- list(name1 = your\_comp1,   
 name2 = your\_comp2)  
OR  
my\_list <- list(your\_comp1, your\_comp2)  
names(my\_list) <- c("name1", "name2")

Single [] will return a list.

Selecting elements from a list with [[ ]] or with the $ sign. Both will select the data frame representing the reviews:   
shining\_list[["reviews"]]  
shining\_list$reviews

Besides selecting components, you often need to select specific elements out of these components. For example, with shining\_list[[2]][1] you select from the second component, actors (shining\_list[[2]]), the first element ([1]).

To conveniently add elements to lists you can use the c() function, that you also used to build vectors:  
ext\_list <- c(my\_list, my\_name = my\_val)

# **Tableau**

## Basics

Dimensions – independent variables; measures – dependent variables.

Right click under Measures – Create Calculated Field

Colour by factors – drag the column on the left onto the colour field or you can drag the top column onto it while holding Ctrl. Can assign qualitive or quantitive values onto colour.

Can drag items onto label, then right click and change format e.g. from number to currency.

Export: Worksheet – Copy

When looking at the data source you can work off an extract rather than the live data. This way if the original data changes it won't affect the file. Usually in top right or a right click. Benefit - live extracts can be slow.

## Time Series

Double click measure and then dimension - will put in line chart.

If you click on the column period, you can select how to aggregate and can also choose the granularity.

# **My Career**

## Skills/Description

- Analyst coming from more of a business background. Traditionally there’s been a gap between data analysts and end business users, where I see my big strength is bridging this gap between the 2 worlds. Ie. Being a business user who has strong data skills rather than coming from an IT background. So my focus is always on how is this going to add value to the company and how my findings are going to drive change or improvement. So I believe I understand business strategy and business problems, so I can interprate what people want, identify problems or areas of improvements, then translate the insights and their impact concisely.

- My roles have been diverse and I’ve been given a lot of freedom. One of the pioneers of the new data movement at my company. In recent years, my focus has been on innovation and improving the way we do things, many of these fundamentally changed the way departments operated:   
- Risk (scoring, profile comparison to previous, PGBI, gambling problem predictive model, returns predictive model)  
- Customer Insights (customer centricity with bonus offers, new segmentation, churn modelling, high value closure, insights)

- Translating the technical output to give meaningful, concise and easy-to-digest insights.

- Advanced Excel, SQL & R; PowerPoint/Access, analytics skills, management & project management skills, basic Tableau but would like to learn (taken some online courses).

## Noteworthy

- Very broad role which has encompassed many things – went from customer service to fraud officer, fraud analyst, senior fraud analyst, customer insights and now BA with more of a focus on marketing and customer centricity.

### Business/Customer Insights/Marketing

- Customer segmentation of all past month customers by daily purchase behaviour. Then offered a tailored upsell, this increased deposits by 5-20%. Also saw interesting demographic trends around age.

- Report optimization – power pivot to pull data from SQL proc and dashboard with filters.

- New dashboards in SSRS.

- Marketing campaign ROI over 18 months, based on the initial cost we set break even benchmarks to help measure how successful a campaign is.

- I always had differing views around the way to view a customer – e.g. accounts, MID, linked MIDs

- Pitched idea of bonus redemption based on email/IP address. Gave solid numbers based on time periods prior to rego (to also eliminate to many false positives), which estimate saving of $1mil pa, but within this also analysed good players and how many go on to continue purchasing etc.

- Created a different player segmentation model – different way of looking at new regos and monitoring value of campaigns, saw different groups had different attrition rates then built predictive regression models from month 4 to 12 which were very accurate.

- Customer churn model.

- Insights comparing attritions between different countries & segments.

- Insights on player device – especially looking at age, purchases. Could see certain operating systems / devices depositing more and being better customers. Using R to explore with density charts, scatterplots and trend lines, bar/histograms. Using features like Cut2 to create different discrete variables where you could see things like the fact a specific age group purchased the most.

- Plotting top customers on google map – using geocode API and then the count as the size of the dots to drive marketing strategy. Saw some areas had less good customers but those they did have were condensed in one area, targeted them.

- Looked at changing makeup of deposits, changing acquisition of genuine new customers and the attrition of these customers (rolling retention). Then can see if players reach the third month they have a much higher chance of staying around so built predictive model of this to identify these players.

- User experience improvements with RMM system.

-

### Risk

- Risk profiling and then new scoring system.   
> 8 scoring bands based on risk level; 91% landed in top 3 bands which accounted for 18% of results; in particular 60% landed in my top urgent band which accounts for 4% of results.  
> Cut out the lower two bands from being reviewed which removed 40% of workload. So then rather than hire a new member, this freed up time for staff to look at other things.  
> Cut CBs by about 0.3-0.4% - $75-100k per month, but more to the point is the ratio.

- Predictive model based on 2 weeks of a customer’s behaviour to predict whether they will exclude – random forest 80% accuracy, 69% specificity. This was then widely adopted as a method for reaching out to customers and was endorsed by the UK Gambling Commission.

- Returns prediction model – predict those who are going to return their eCheck deposits (fraud). Problem was return rate of 10-20% ($2.5mil pa), seen as cost of business but I created a model that can predict these fraudulent customers. Focus was on very low false positive rate, so focused on 25% of population where model was extremely confident 95%+ success rate. E.g. saving of $625k pa.

- Lots of changes to procedures e.g. UK compliance came up with framework and flow for players e.g. 2000 EUR, large depositing player screening with big fines involved.

- Removal of DC which led to savings of $500k per year.

- Made lots of reports a lot more efficient, analysing what people do and how can it be quicker. Always trying to cut down results without missing anything, optimisation. Pretty much re-wrote all of the reports and fundamentally changed the way we do things, e.g. monitoring system.

- Designed chargeback/return database.

- Implemented a project management framework - KanBanize. I find it a little one dimensional - project and subtasks. I'd prefer something more agile where you have a project and then multiple deliverables that can be assigned to different people within the one project. KanBan more about monitoring tasks through the different stages in life cycle rather than allowing multiple contributions from different people.

- Interactive roster system for management.

## Portfolio

**HR Analytics: Predicting Employee Attrition**Kaggle IBM HR Dataset.Analysing employee information to frame the business problem and build a model to predict attrition (AUC 0.84).  
Details: Data exploration and clean-up, exploratory analysis, feature analysis and engineering, deep learning algorithm and model evaluation, actionable insights and options for applying the model.  
<http://htmlpreview.github.io/?https://github.com/RTutt/Kaggle-Solutions/blob/master/HR_Employee_Attrition_Final.html>

**E-commerce Analytics: Customer & Sales Insights**  
Kaggle E-commerce Dataset.  
Details: Data exploration, summarising and aggregating data, visualizing sales by country with interactive map, order insights, customer clustering, segmentation and visualization of customer insights.  
<http://htmlpreview.github.io/?https://github.com/RTutt/Kaggle-Solutions/blob/master/E-commerce%20Insights.html>

**Financial Analytics: Credit Card Default Analysis & Predictive Models**UCI Machine Learning Repository: Credit Card Default Dataset.  
Analysing Taiwanese banking credit card clients to predict who will default on their next payment (AUC 0.78).  
Details: Data exploration & clean-up, exploratory analysis, pre-processing and feature engineering, various modelling and evaluation, further exploratory analysis in conjunction with modelling process.  
<http://htmlpreview.github.io/?https://github.com/RTutt/Kaggle-Solutions/blob/master/CC_Default_Analysis.html>

**JHU Practical Machine Learning Assignment**John Hopkins University – Data Science Specialization.  
Dataset comes from reading wearable fitness trackers.  
Details: Data cleaning, feature selection & PCA, random forest classification model (99% accuracy) to predict the class of each observation. This assignment was based on accuracy of the model and subsequent predictions, so this was fairly straight forward and to the point.  
<http://htmlpreview.github.io/?https://github.com/RTutt/Practical-Machine-Learning-Assignment/blob/master/Practical%20Machine%20Learning%20Assignment/Assignment_Write_Up_Final.html>

**JHU Exploratory Data Analysis Assignment**John Hopkins University – Data Science Specialization.  
Dataset comes from the US Environmental Protection Agency relating to air pollutants and the tracking of emissions in the atmosphere.  
Details: Exploration of the National Emissions Inventory database and fine particle matter pollution in the US during 1999-2008. The charts were created according to assignment requirements so fairly to the point but with a few extra features added.  
<https://github.com/RTutt/Exploratory-Data-Analysis-Project-1>

## Biggest Challenges

- Making sure I understand the data, how the tables interact and the flow. Also what are the primary keys and unique values, clustered and non-clustered indexes.

- Think about what tables are in the query and aggregations/grouping, often use apply.

- 70/30 theory; 70% creating strategy on prepping and cleaning data (Removing records if not relevant to analysis; Understanding the fields and meanings of the data so you can interpret it).

- No explanatations or documents, just needed to use syscolumns/objects, look at dependencies or existing views etc.

- False positives / especially negatives.

- Statistical models – overfitting & not having enough data. E.g. marketing and customer base has changed so having to analyse and build model based on a years’ worth of results.

- Always very conscious of reviewing the whole data in the big picture and not letting my own pre-conceived notions manipulate the observations as I’ve seen this done many times. People tend to pick and choose which bits of the data to use for their benefit.

## Questions to Ask Interviewer

What kind of data scientist are you looking for?

How many in the team? How experienced? Who’s the best and what makes them the best? Has anyone in the team not worked out previously, if so why didn’t they work out?

What projects are you working on?

Who’s your biggest competitor?

What’s your biggest challenge?

What would my day-to-day activities be?

What software do you guys use or allow?

Is their training, development or mentoring? If so, is there an allowance or how does it work?

## Job Tips

- Put together portfolio of work, maybe publish things on Tableau public. Building a data science portfolio:

<https://www.dataquest.io/blog/build-a-data-science-portfolio/>

- Skill and freeform education rather than uni

- Makeover Monday - Tableau public

- Be sure you have a really strong Excel foundation to fall back on -- advanced stuff such as what-if analysis, lookup functions, chaining text-processing functions together to clean up data, etc. should all be second nature for you.

<https://www.quora.com/How-beneficial-is-having-Python-R-and-SQL-learned-through-self-study-on-my-resume-as-a-non-technical-grad>

Know the right statistical techniques for different cases.

Must know:   
Linear regression  
Classification regression trees  
Naive bayes  
When to use them and drawbacks and explain them.  
Know key concepts like overfitting and data leakage

Uplift modelling – people who were directly affected by the intervention, who wouldn’t have purchased anyway.

## My Project Ideas

**Customer characterization** - based this off daily behaviour. Took their average purchase amount, looked at actual purchasing days to get average sum/count.

Findings - mid-range was split into high-freq and low-freq who behave very differently. Training customers so AB tests need to be longer, e.g. 1 month.

First phase - AB testing different upsell offers or low balance offers.

Second phase - link these to customer health based on b/p, margin etc.

Third phase - start making churn predictions.

**Analysis of segments**  
Further analyse the different segments - geodemographic, demographic, further behavioural, account age, more lifetime figures. This could also give us an idea of the people making up our deposits.

Behavioural differences - responsiveness to offers, how often they play, days of week?

**Customer health / churn modelling of existing customers**

Need to think about segmenting customers differently first, e.g. by their regularity by week or their avg distance between purchases (take into account st dev of distances). Especially if analysing behaviour, ratios will be skewed and so you need to take their regularity into account, e.g. when comparing current week to previous weekly averages, only look at those who purchased in more than half the weeks.

Performing an almost RFM segmentation on purchases, weeks purchased, count purchases and looking at only the top 25% rank.

**RFM Segmentation**

Recency - date since last purchase. I may also somehow use the ratio of net winning sessions to losses in here.

Frequency - count deposits & count days deposits.

Monetary - total net win amount or total deposits. Used to segment value.  
Health Score – past week activity vs previous activity grouped by weeks for 3 months (time between deposits, regularity how many days purchased, count purchases); last experience – margin, num bets. Ends up giving probability of leaving.  
Everyday Behaviour Attribute – based on cluster.

Other characteristics - count of transactions, avg purchase per day, day of week.

Do this weekly to create a scorecard of each characteristic and then segment these customers so we can treat each group differently. Will be based on only recent purchases in past month.  
Put them into different tiers then compare this to their total deposits historically, total deposits within period, bonuses, response to offers, number of days depositing, date opened, age, gender, count days depositing, country.

Aim of characterizing by lots of things, e.g. segments of high freq low depositors, people who come in hard and leave then come back later, big depositors, regular depositors.

Then build my churn model with probabilities on these.

Different demographic **impacts on being good customer** (e.g. percentage of regos or purchases), age, gender, country, types of cards, game choice optimization, understanding deposit makeup, profitability of different first purchases.  
Create customer values based on these.

Recommendation systems

Password Chatbot

**Outbound marketing** to registrations that don’t purchase: Look at first purchasing days since reg, then see where the big drop off is. E.g. 85% of people who purchase do so in the first 3 days, then email all those who registered and didn’t purchase during this time.

**Analyse customer behaviour to find diff angles** - customers that login but don't deposit (present some sort of offer upon login), customers that only deposit on promo days or have a high comparison of promo ID deposits to non promo ID deposits, customers who displayed good behaviour (perhaps we can offer them more).

**Marketing based on volatility –** look at SD of their purchases and categorise customers based on this. E.g. could indicate which customers purchase on feeling and are easily swayed into impulse buying.

Demographic Analysis of Country

Look at conversions by different metrics, a big one is browser. Especially for recurring payments in e-commerce, breaking out of facebook is a big issue. E.g. into Chrome but if someones default browser is different to Chrome they might never find their way back.

Created multiple linear regression model predicting cost & purchasing players based on current standing. Then predict total purchases based on these (model built on previous data).

Custom marketing & bonus offers for high value credit cards - look by BIN.

User Agents - 51 Degrees, e.g. did a lot of work looking at pixel to purch, CPAs etc of different browsers & devices.  
iOS Platform Version 11.3 is normal browser & 11.3.0 is in-app browser where it would remember favourites.

Simulations for value estimates – Simulations to get median and confidence intervals for value - e.g. saw that purchase value irrespective of age, so get the purchase value of lots of customers. Then based on age, sex, country & no. of players it will select that no. randomly of that cohort and get the total purchase value. This simulation will occur multiple times to get median and confidence intervals.

Focusing on better attribution modelling with Big Query & re-structuring team around frameworks/key focus areas.

MD5 code for VBA - https://www.di-mgt.com.au/crypto.html#MD5

## My Tips

Gartner Model – What is the problem, what is the challenge, the opportunity for disruption, efficiency or improvement.

Leave data as un-aggregated as possible for R as you don't know where the analysis is going to take you.

Soundex only good on Anglo, look at other distance algorithms, also consider accents.

Always dig into attrition & value, interlinked but very telling.

Always need a baseline.

Data strategy - move things to the Command line.

## Dataset Ideas

Sentiment analysis - <https://www.ibm.com/watson/services/personality-insights/>