PRACTICAL DATA SCIENCE

Agenda

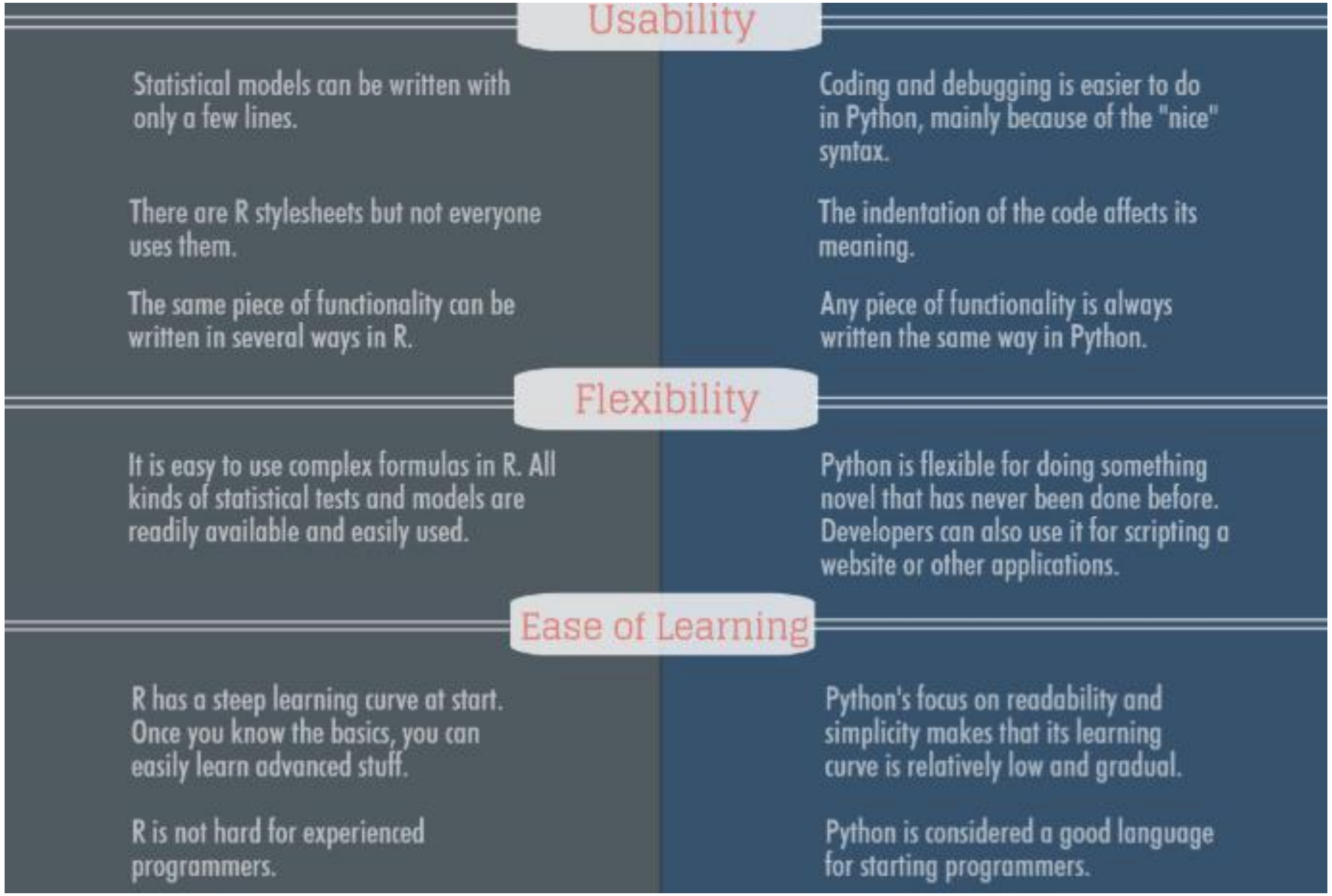
1. R for Data science
2. Data Manipulation & Visualization
3. Data Modeling with 'caret'
4. Advanced Machine Learning models

Long Nguyen

Sep 2019

WHY R?

- R is the most preferred programming tool for statisticians, data scientists, data analysts and data architects
- R has over **10,000 packages** (a lot of available algorithms) from multiple repositories.



- The choice between R and Python really depends on your level of knowledge and objective.
- Day-to-day users and data scientists are getting best of both worlds

ESSENTIALS OF R PROGRAMMING

- Basic computations
- Five basic classes of objects
  - Character
  - Numeric (Real Numbers)
  - Integer (Whole Numbers)
  - Complex
  - Logical (True / False)
- Data types in R
  - Vector: a vector contains object of same class
  - List: a special type of vector which contain elements of different data types
  - Matrix: A matrix is represented by set of rows and columns.
  - Data frame: Every column of a data frame acts like a list

```
2+3

sqrt(121)

myvector<- c("Time", 24, "October", TRUE, 3.33) #convert to chars

my_list <- list(22, "ab", TRUE, 1 + 2i)

my_list[[1]]

my_matrix <- matrix(1:6, nrow=3, ncol=2)

df <- data.frame(name = c("ash","jane","paul","mark"), score =
c(67,56,87,91))

 df

      name score

   1 ash NA

   2 jane NA

   3 paul 87

   4 mark 91
```

ESSENTIALS OF R PROGRAMMING

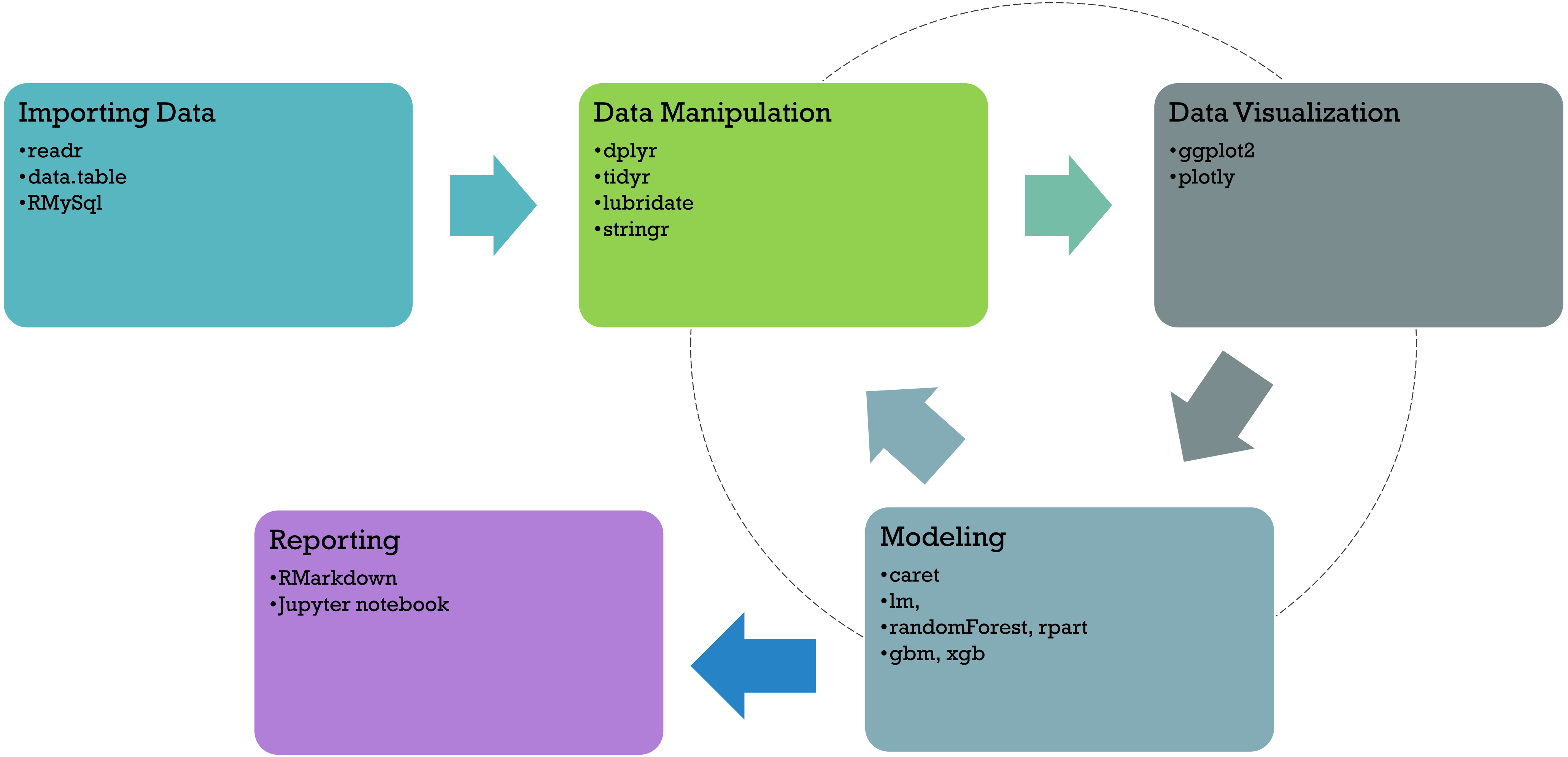
- Control structures
  - If (condition){

        Do something
    }else{
        Do something else
    }
- Loop
  - For loop
  - While loop
- Function
  - *function.name* <- *function(arguments*) { computations on the arguments some other code }

```
x <- runif(1, 0, 10)
if(x > 3) {
        y <- 10
} else {
        y <- 0
}
for(i in 1:10) {
        print(i)
}

mySquaredFunc<-function(n){
        # Compute the square of integer `n`
        n*n
}
mySquaredVal(5)
```

USEFUL R PACKAGES

- Install packages: install.packages('readr', 'ggplot2', 'dplyr', 'caret')
- Load packages: library(package_name)



IMPORTING DATA

- CSV file

```
mydata <- read.csv("mydata.csv")  # read csv file
library(readr)
mydata <- read_csv("mydata.csv")  # 10x faster
```

- Tab-delimited text file

```
mydata <- read.table("mydata.txt")  # read text file
mydata <- read_table("mydata.txt")
```

- Excel file:

```
library(XLConnect)
wk <- loadWorkbook("mydata.xls")
df <- readWorksheet(wk, sheet="Sheet1")
```

- SAS file

```
library(sas7bdat)

mySASData <- read.sas7bdat("example.sas7bdat")
```

- Other files:
  - Minitab, SPSS(foreign),
  - MySQL (RMySQL)

```
Col1,Col2,Col3
100,a1,b1
200,a2,b2
300,a3,b3
```

```
100   a1   b1
200   a2   b2
300   a3   b3
400   a4   b4
```

DATA MANIPULATION WITH 'DPLYR'

- Some of the key "verbs":
  - **select**: return a subset of the columns of a data frame, using a flexible notation
  - **filter**: extract a subset of rows from a data frame based on logical conditions
  - **arrange**: reorder rows of a data frame
  - **rename**: rename variables in a data frame

```
library(nycflights13)

flights

select(flights, year, month, day)

select(flights, year:day)

select(flights, -(year:day))


jan1 <- filter(flights, month == 1, day == 1)

nov_dec <- filter(flights, month %in% c(11, 12))

filter(flights, !(arr_delay > 120 | dep_delay > 120))

filter(flights, arr_delay <= 120, dep_delay <= 120)


arrange(flights, year, month, day)

arrange(flights, desc(arr_delay))

rename(flights, tail_num = tailnum)
```

DATA MANIPULATION WITH 'DPLYR'

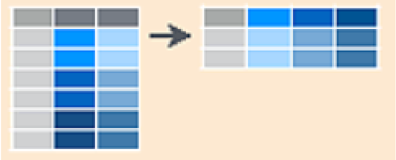
- Some of the key "verbs":
  - **mutate**: add new variables/columns or transform existing variables
  - **summarize**: generate summary statistics of different variables in the data frame
  - **%>%**: the "pipe" operator is used to connect multiple verb actions together into a pipeline

```
flights_sml <- select(flights,   year:day, ends_with("delay"),
                    distance,   air_time )
mutate(flights_sml,   gain = arr_delay - dep_delay,
                    speed = distance / air_time * 60 )
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delays, count > 20, dest != "HNL")
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```

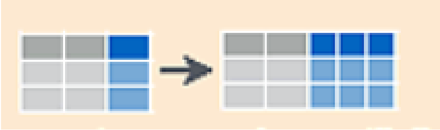
DATA MANIPULATION WITH ‘TIDYR’

- Some of the key “verbs”:
  - **gather**: takes multiple columns, and gathers them into key-value pairs

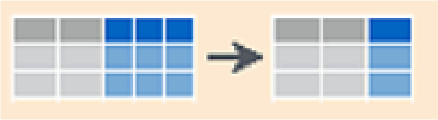
  - **spread**: takes two columns (key & value) and spreads in to multiple columns



  - **separate**: splits a single column into multiple columns

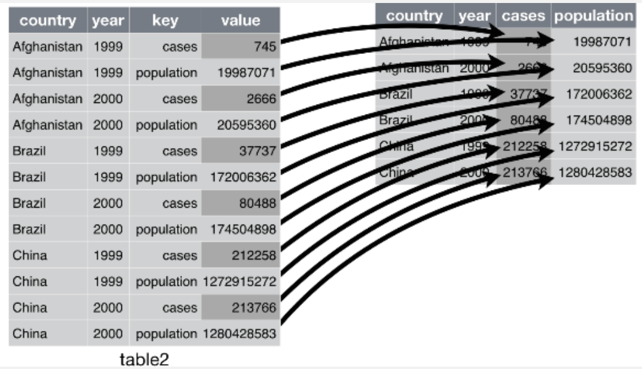


  - **unite:** combines multiple columns into a single column



```
library(tidyr)
tidy4a <- table4a %>%
  gather(`1999`, `2000`, key = "year", value = "cases")
tidy4b <- table4b %>%
  gather(`1999`, `2000`, key = "year", value = "population")
left_join(tidy4a, tidy4b)
```

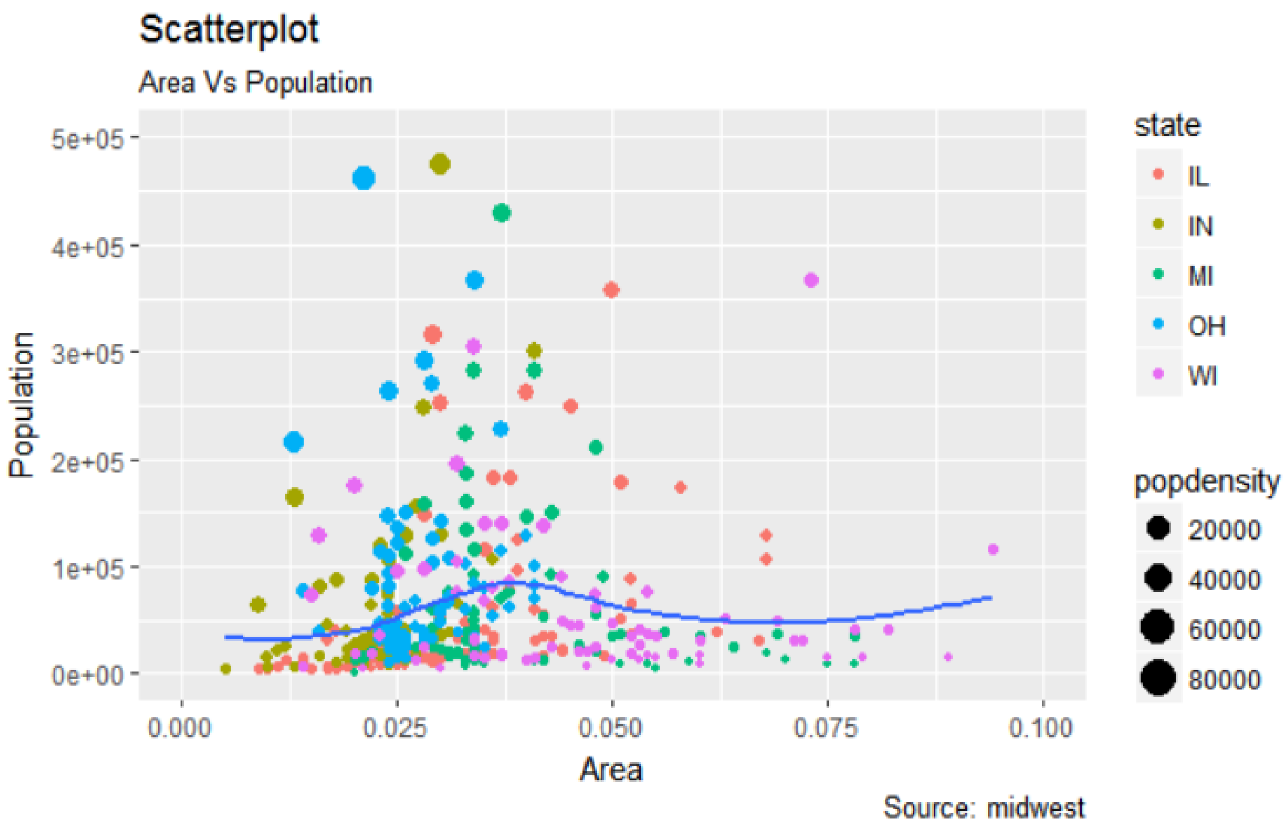

```
spread(table2, key = type, value = count)
```




```
separate(table3, year, into = c("century", "year"), sep = 2)
separate(table3, rate, into = c("cases", "population"))
unite(table5, "new", century, year, sep = "")
```

DATA VISUALIZATION WITH 'GGPLOT2'

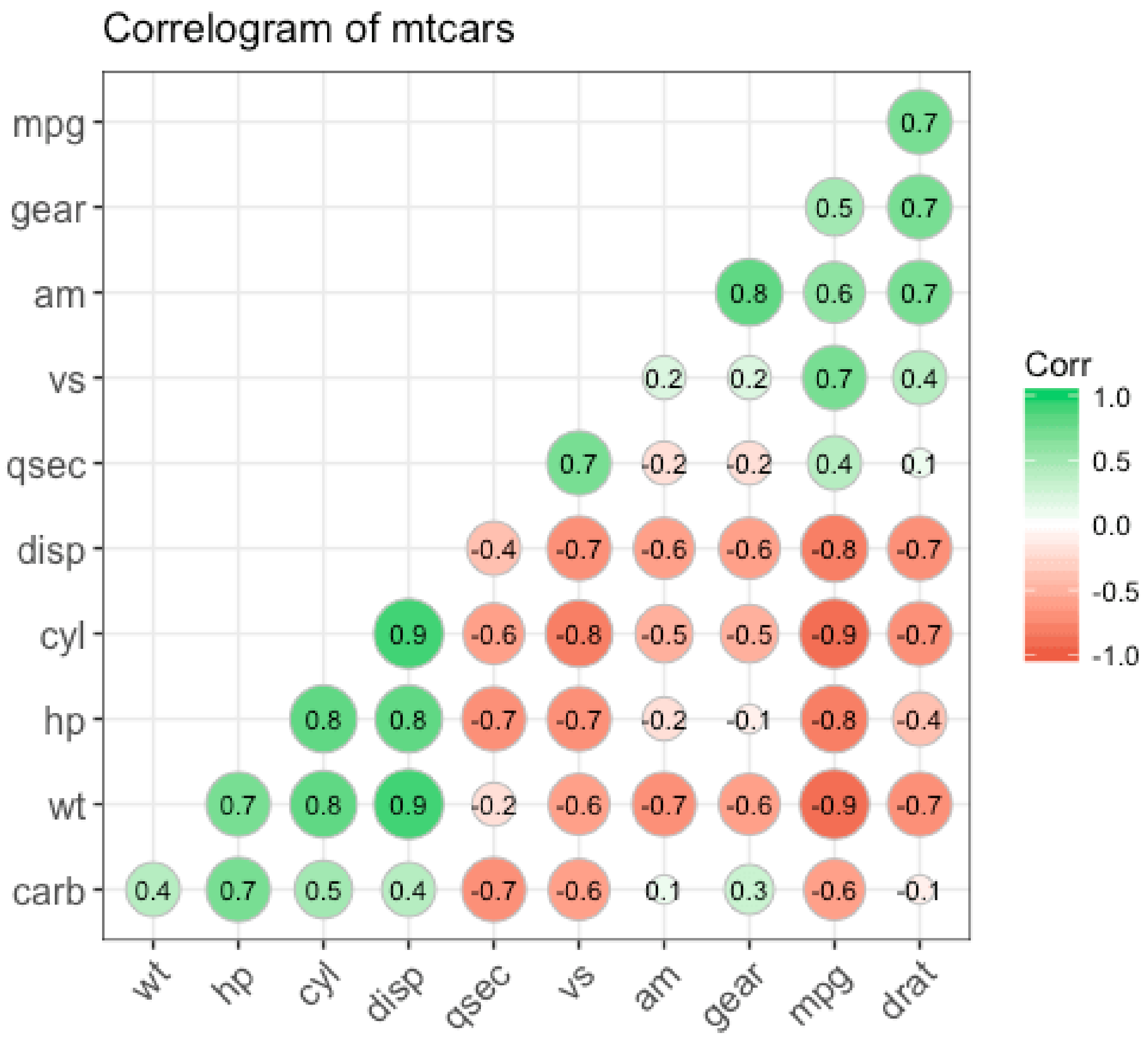
- Scatter plot




```
library(ggplot2)

ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() +
geom_smooth(method="lm")


ggplot(midwest, aes(x=area, y=poptotal)) +
 geom_point(aes(col=state, size=popdensity)) +
 geom_smooth(method="loess", se=F) +
 xlim(c(0, 0.1)) +
 ylim(c(0, 500000)) +
 labs(subtitle="Area Vs Population",
     y="Population",
     x="Area",
     title="Scatterplot",
     caption = "Source: midwest")
```

DATA VISUALIZATION WITH 'GGPLOT2'

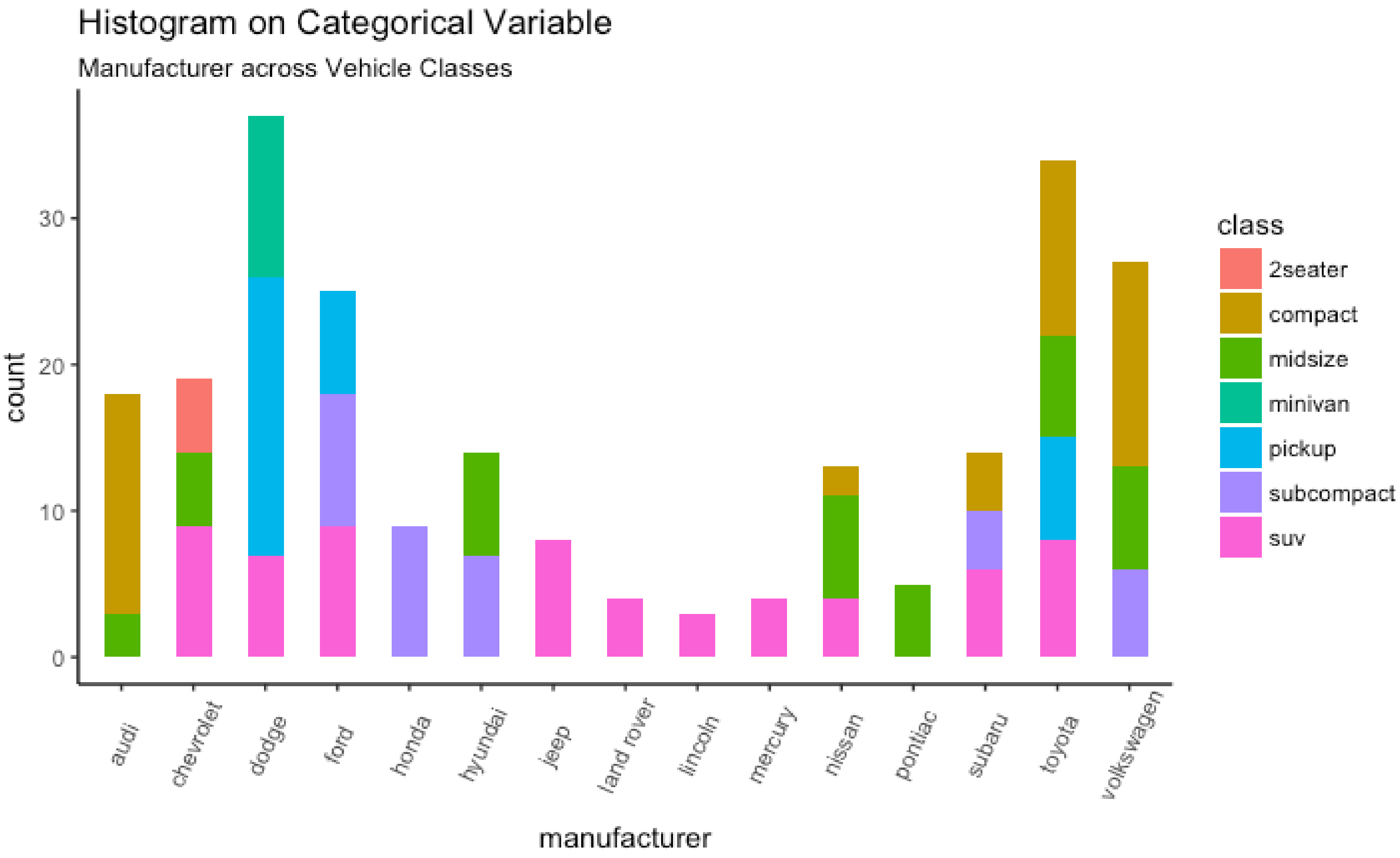
- Correlogram




```
library(ggplot2)

library(ggcorrplot)

# Correlation matrix

data(mtcars)

corr <- round(cor(mtcars), 1)

# Plot

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)
```

DATA VISUALIZATION WITH 'GGPLOT2'

- Histogram on Categorical Variables

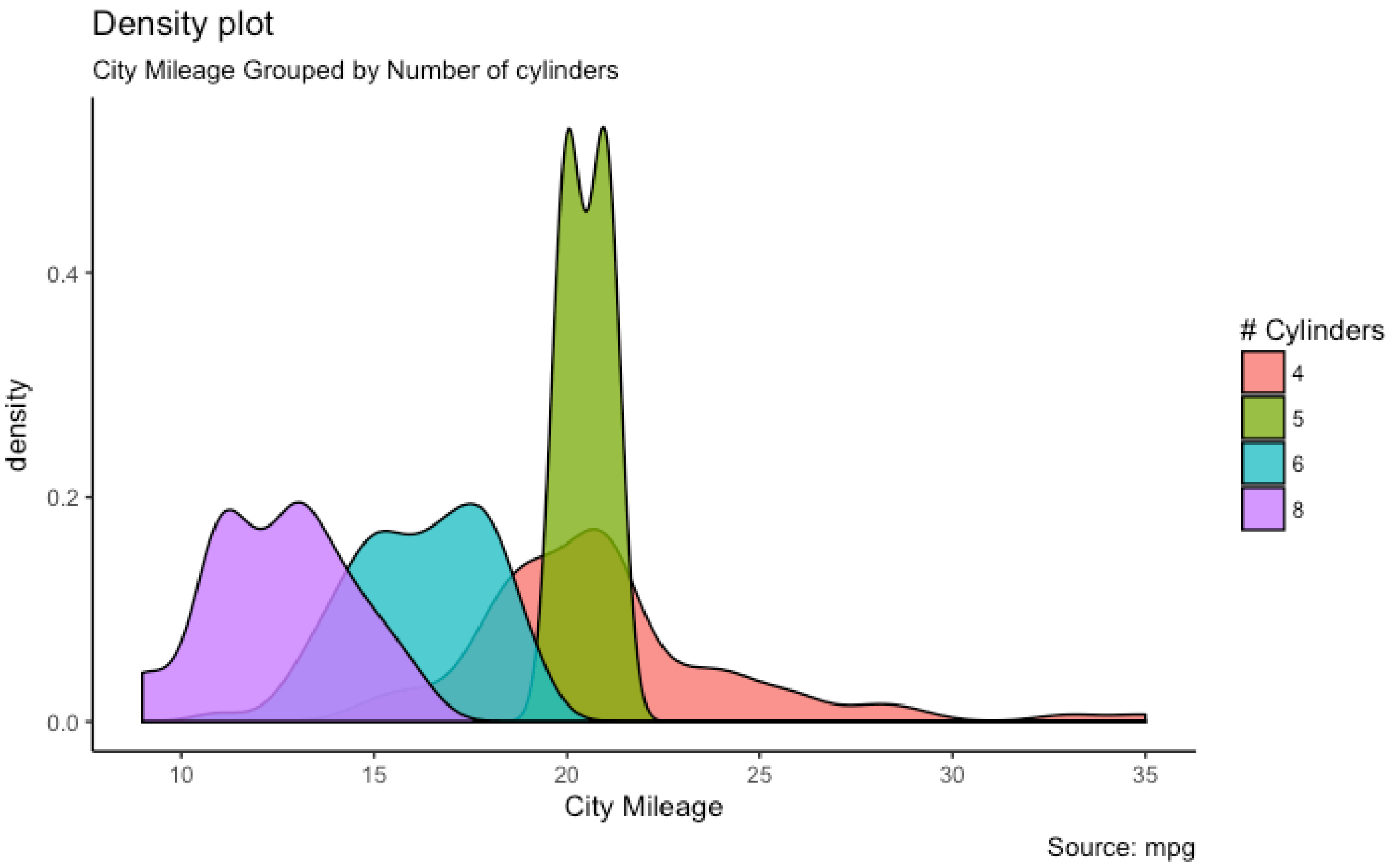



```
ggplot(mpg, aes(manufacturer)) +
geom_bar(aes(fill=class), width = 0.5) +

theme(axis.text.x = element_text(angle=65,
          vjust=0.6)) +

labs(title="Histogram on Categorical Variable",
          subtitle= "Manufacturer across Vehicle
          Classes")
```

DATA VISUALIZATION WITH 'GGPLOT2'

- Density plot

```
ggplot(mpg, aes(cty)) +
geom_density(aes(fill=factor(cyl)), alpha=0.8) +

 labs(title="Density plot", subtitle="City Mileage
            Grouped by Number of cylinders",
            caption="Source: mpg", x="City Mileage",
            fill="# Cylinders")
```

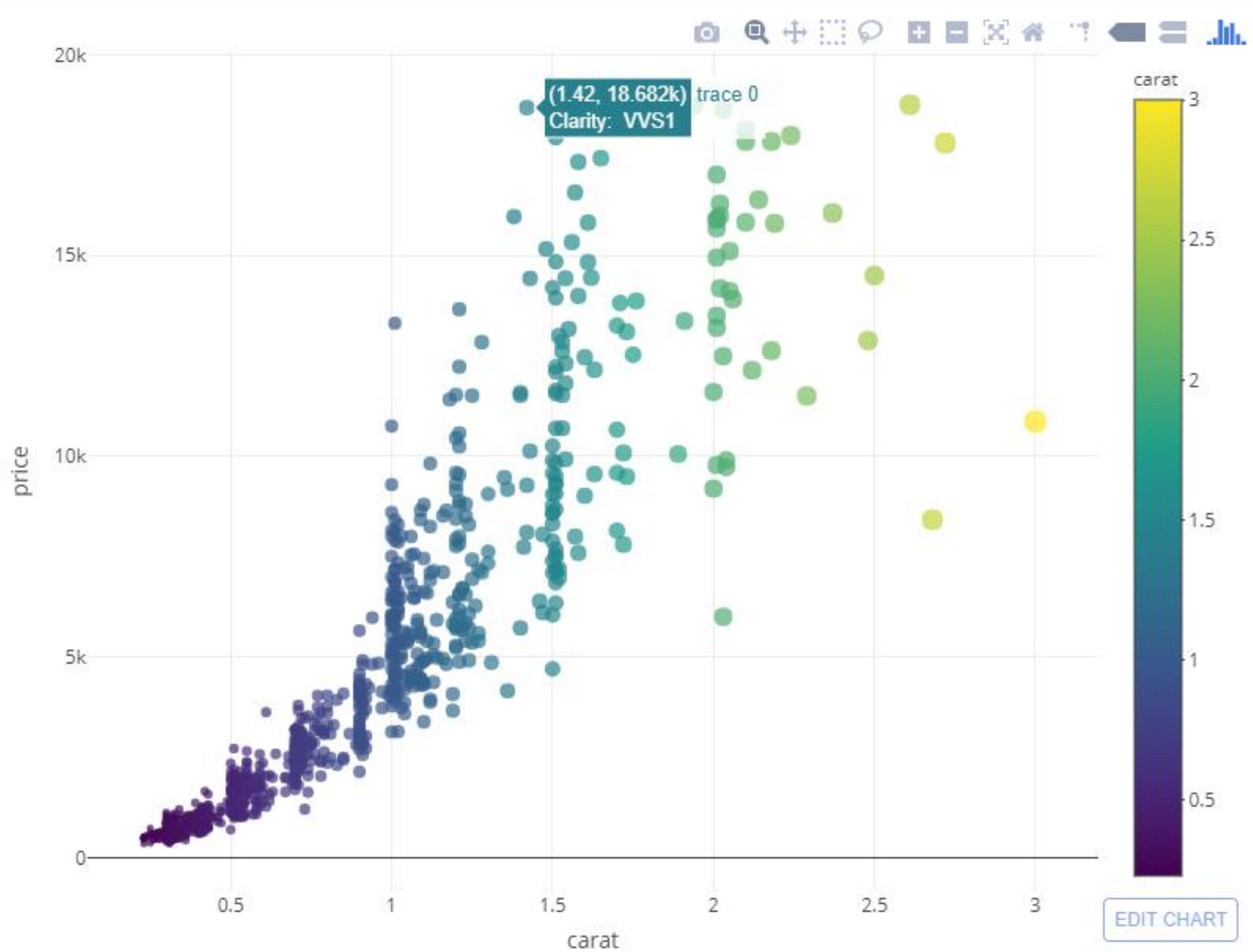


Other plots:

- Box plot
- Pie chart
- Time-series plot

INTERACTIVE VISUALIZATION WITH 'PLOTLY'

Plotly library makes interactive, publication-quality graphs online. It supports line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heat maps, subplots, multiple-axes, and 3D charts.




```
library(plotly)

d <- diamonds[sample(nrow(diamonds), 1000), ]

plot_ly(d, x = ~carat, y = ~price, color = ~carat,
size = ~carat, text = ~paste("Clarity: ", clarity))
```

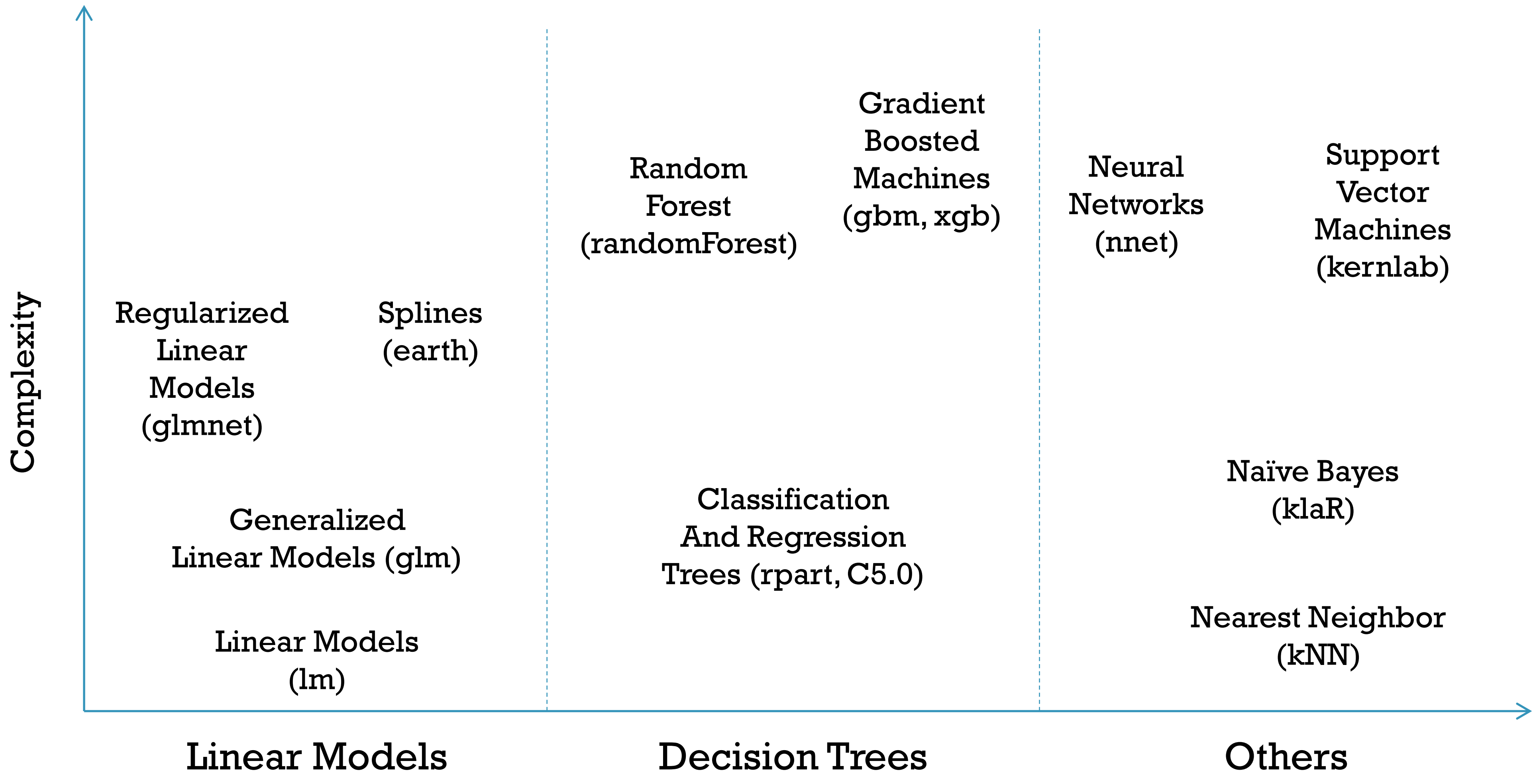
R WITH JUPYTER NOTEBOOK

Install R in Anaconda (link)
Markdown syntax
Text formatting
italic or _italic_
bold   __bold__
`code`
superscript^2^ and subscript~2~
Headings
1st Level Header
2nd Level Header
3rd Level Header
Lists
*   Bulleted list item 1
*   Item 2
  * Item 2a
  * Item 2b
1.   Numbered list item 1
2.   Item 2. The numbers are incremented automatically in the output.
Links and images
`<http://example.com>[linked phrase](http://example.com)![optional caption text](path/to/img.png)`

PREDICTIVE MODELING LANDSCAPE

General Purpose Algorithms*

* for illustrative purposes only, not to scale, precise, or comprehensive



More Comprehensive List: http://caret.r-forge.r-project.org/modelList.html

DATA MODELING WITH 'CARET'

- Loan prediction problem

Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
Male	No	0	Graduate	No	5849	0	NA	360	1	Urban	Y
Male	Yes	1	Graduate	No	4583	1508	128	360	1	Rural	N
Male	Yes	0	Graduate	Yes	3000	0	66	360	1	Urban	Y
Male	Yes	0	Not Graduate	No	2583	2358	120	360	1	Urban	Y
Male	No	0	Graduate	No	6000	0	141	360	1	Urban	Y

- Data standardization and imputing missing values using kNN

```
preProcValues <- preProcess(train, method = c("knnImpute","center","scale"))
library('RANN')
train_processed <- predict(preProcValues, train)
```

- One-hot encoding for categorical variables

```
dmy <- dummyVars(" ~ .", data = train_processed,fullRank = T)
train_transformed <- data.frame(predict(dmy, newdata = train_processed))
```

- Prepare training and testing set

```
index <- createDataPartition(train_transformed$Loan_Status, p=0.75, list=FALSE)
trainSet <- train_transformed[ index,]
testSet <- train_transformed[-index,]
```

- Feature selection using rfe

```
predictors<-names(trainSet)[!names(trainSet) %in% outcomeName]
Loan_Pred_Profile <- rfe(trainSet[,predictors], trainSet[,outcomeName], rfeControl = control)
```

DATA MODELING WITH 'CARET'

- **Take top 5 variables**

```
predictors<-c("Credit_History", "LoanAmount", "Loan_Amount_Term", "ApplicantIncome", "CoapplicantIncome")
```

- **Train different models**

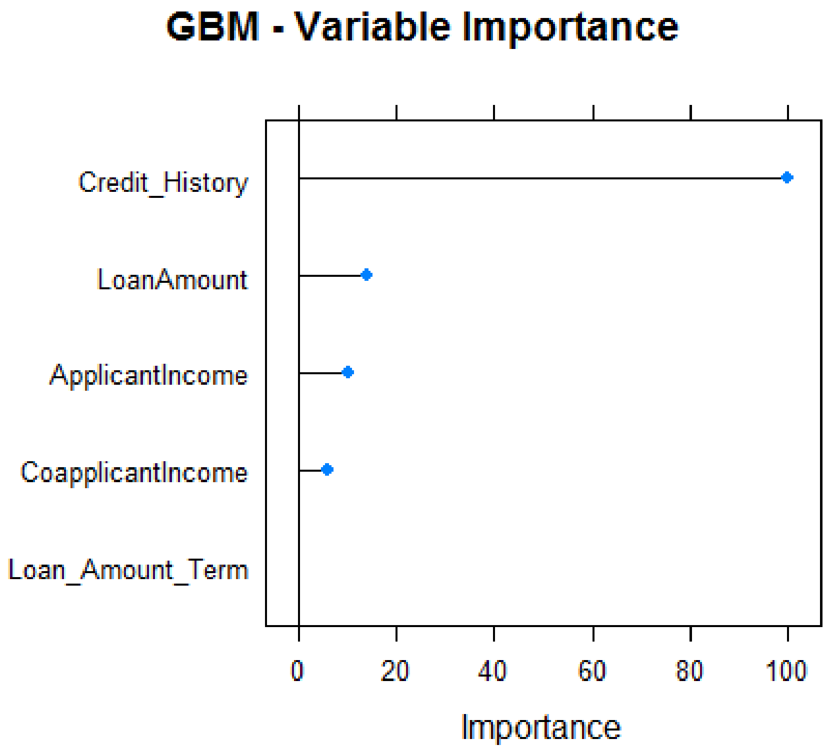
```
model_gbm<-train(trainSet[,predictors],trainSet[,outcomeName],method='gbm')
model_rf<-train(trainSet[,predictors],trainSet[,outcomeName],method='rf')
model_nnet<-train(trainSet[,predictors],trainSet[,outcomeName],method='nnet')
model_glm<-train(trainSet[,predictors],trainSet[,outcomeName],method='glm')
```

- **Variable important**

```
plot(varImp(object=model_gbm),main="GBM - Variable Importance")
plot(varImp(object=model_rf),main="RF - Variable Importance")
plot(varImp(object=model_nnet),main="NNET - Variable Importance")
plot(varImp(object=model_glm),main="GLM - Variable Importance")
```

- **Prediction**

```
predictions<-predict.train(object=model_gbm,testSet[,predictors],type="raw")
confusionMatrix(predictions,testSet[,outcomeName])
    #Confusion Matrix and Statistics
    #Prediction   0   1
    #    0  25   3
    #    1  23 102
    #Accuracy : 0.8301
```



ADVANCED ML MODEL - XGBOOST

XGBoost is short for eXtreme Gradient Boosting. It is:

- An open-sourced tool
  - Computation in C++
  - R/python/Julia interface provided
- A variant of the gradient boosting machine
  - Tree-based model
- The winning model for several Kaggle competitions
- XGBoost is currently host on Github.

Why should we use XGBoost?

- Easy to use
  - Easy to install.
  - Highly developed R/python interface
- Efficiency
  - Automatic parallel computation on a single machine.
  - Can be run on a cluster
- Accuracy
  - Good result for most data sets
- Feasibility
  - Customized objective and evaluation Tunable parameters

ADVANCED ML MODEL - XGBOOST

Basic parameters

- Input features
  - XGBoost allows dense, sparse matrix or its own class xgb.DMatrix as the inputs.
- Target variable
  - A numeric vector. Use integers starting from 0 for classification, or real values for regression.
- Objective
  - For regression use 'reg:linear'
  - For binary classification use 'binary:logistic'.
  - For multiclass classification use 'multi:softmax'
- Number of iteration
  - The number of trees added to the model.

Advance parameters

- *nthread*: number of parallel threads.
- *booster*: gbtree (tree-based model), gblinear (linear function).
- *eta*: step size shrinkage used in update to prevents overfitting. Range in [0,1], default 0.3.
- *max_depth*: Maximum depth of a tree. Range [1,], default 6.
- *min_child_weight*: Minimum sum of instance weight needed in a child. Range [0,], default 1.
- *subsample*: Subsample ratio of the training instance. Range (0, 1], default 1.
- *Colsample_bytree*: Subsample ratio of columns when constructing each tree. Range (0, 1], default 1.

TUNING PARAMETERS

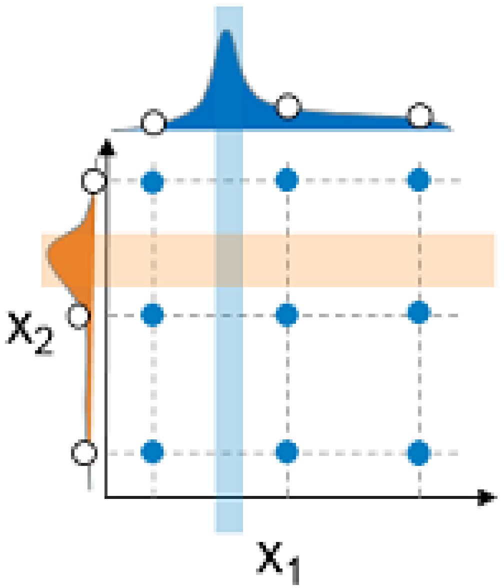
It is nearly *impossible* to give a set of universal optimal parameters, or a global algorithm achieving it.

The key points of parameter tuning are:

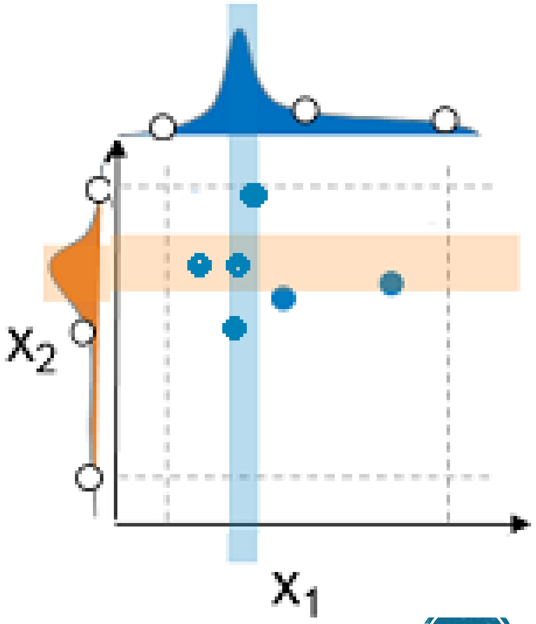
- Control Overfitting: "Bias-Variance Tradeoff"
  - *max_depth, min_child_weight*
- Robust to noise
  - *subsample, colsample_bytree*
- Deal with Imbalanced data
  - Balance the positive and negative weights, by *scale_pos_weight*
  - Use "*auc*" as the evaluation metric
- Trust the cross validation
  - Use early.stop.round to detect continuously being worse on test set.
  - If overfitting observed, reduce stepsize *eta* and increase *nround* at the same time.

TUNING PARAMETERS

- **Non-expert approach**: apply grid search on all parameter space
  - Zero effort and no supervision
  - Enormous parameters' space
  - Very time consuming



- **Expert approach** = experience+ intuition + one-by-one approach
  - Set learning rate (*eta*) parameter 0.1-0.2 (based on dataset size and available resources); all other parameters at default
  - Test maximum tree depth parameter, rule: 6-8-10-12-14; pick best performing on CV
  - Tune min leaf node size (min_child_weight), rule: 1-5-10-20-50;
  - Tune randomness of each iteration (column/row sampling); Usually 0.7/0.7
  - Decrease *eta* to value which you are comfortable with your hardware; 0.025 is typically a good choice;



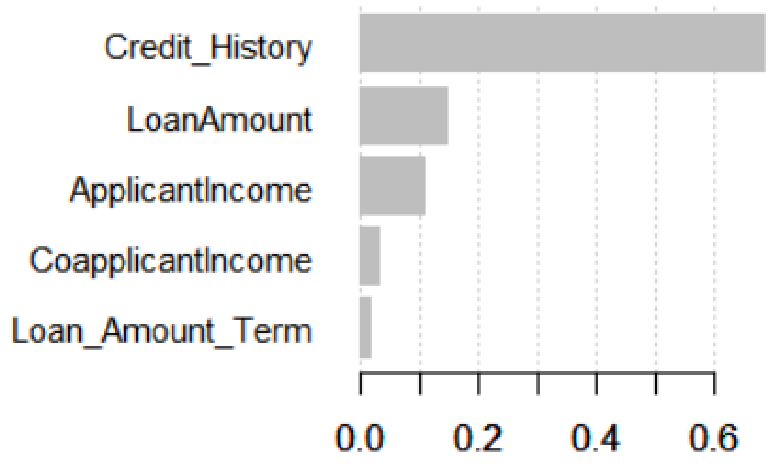
ADVANCED ML MODEL - XGBOOST

Code

```
param <- list(objective = "binary:logistic", base_score = 0.5)

xgboost.cv = xgb.cv(

  param = param,

  data = xgb.train.data,

  folds = cv,

  nrounds = 1500,

  early_stopping_rounds = 100,

  metrics = 'auc'

)

best_iteration = xgboost.cv$best_iteration

xgb.model <-

  xgboost(param = param, data = xgb.train.data, nrounds =
best_iteration)
```

Results

Model	AUC
LR	0.726
RF	0.724
GBM	0.724
XGBoost	0.784



ENSEMBLE MODELS

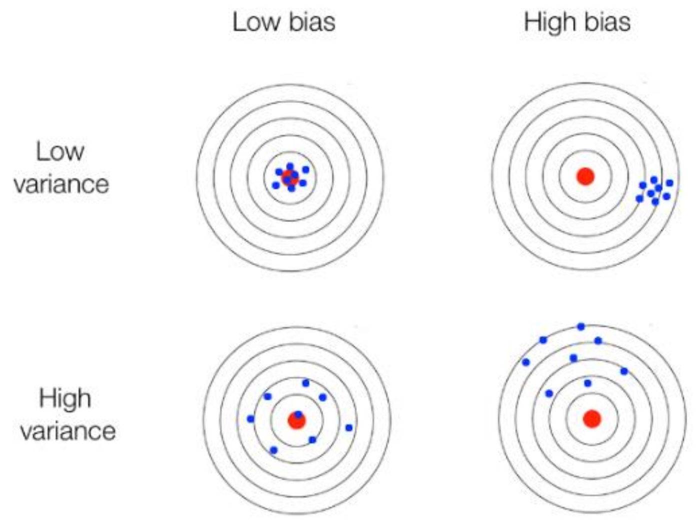
- Basic concepts
  - Average
  - Majority vote
  - Weighted average

Model1	Model2	Model3	AveragePrediction
45	40	65	50

Model1	Model2	Model3	VotingPrediction
1	0	1	1

	Model1	Model2	Model3	WeightAveragePrediction
Weight	0.4	0.3	0.3	
Prediction	45	40	60	48

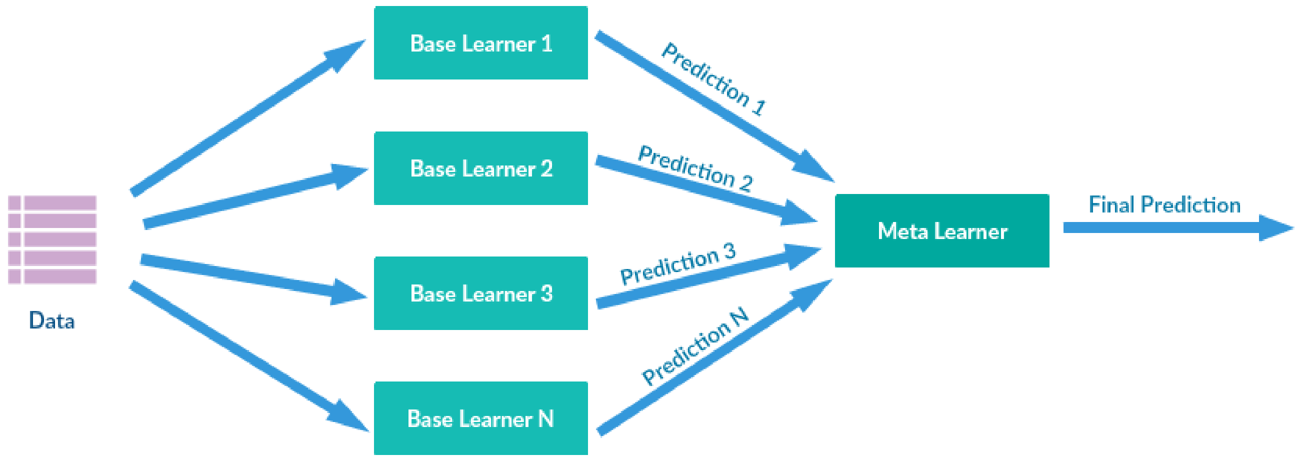
- Bagging
  - Create multiple bootstrapped samples and use the majority vote or averaging concepts to get the final prediction.
  - Mostly used to reduce the variance in a model, e.g., Random Forest algorithm.
- Boosting
  - Give higher weight to those observations that were poorly predicted by the previous model.
  - Reduce bias in a model, e.g., Ada-Boost, XGBoost, Gradient Boosted Decision Trees etc.



Data		Bootstraped Sample
Row 1		Row 2
Row 2		Row 1
Row 3		Row 1

STACKING

- **Different** weak learners are fitted independently from each others and a meta-model is trained on top of that to predict outputs based on the outputs returned by the base models



- Stacking Tips:
  - Choose a con
  - Use a simple meta learner to avoid overfitting

THANK YOU

Reference:

1. Practical Data Science with R, Second Edition, Nina Zumel and John Mount
2. https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/
3. https://mlwave.com/kaggle-ensembling-guide/
4. https://machinelearningmastery.com/machine-learning-ensembles-with-r/
5. https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/