R language for Data Science

Damian Fiłonowicz
Data Engineering

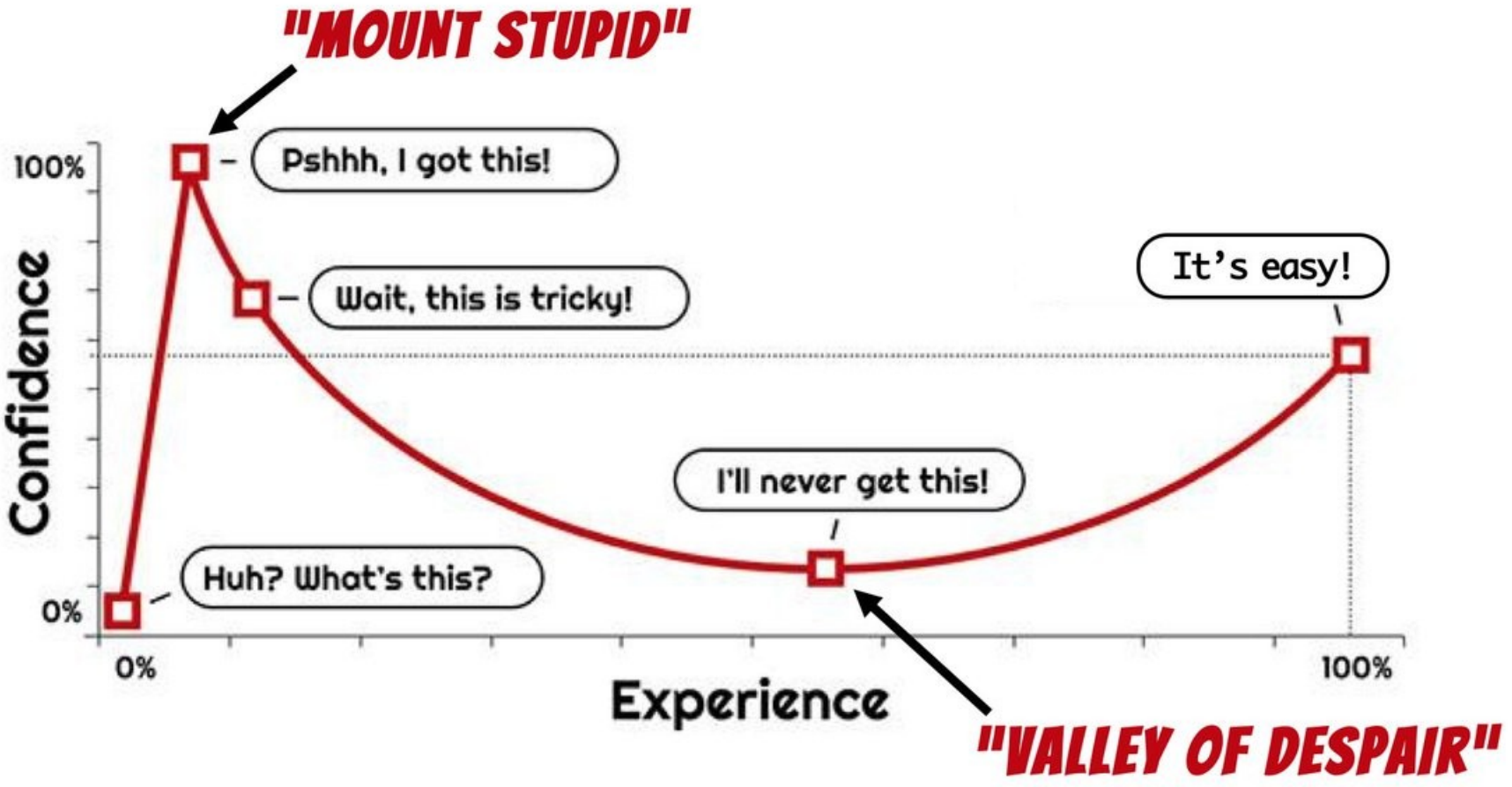
POLITECHNIKA GDAŃSKA

KNZMI QR

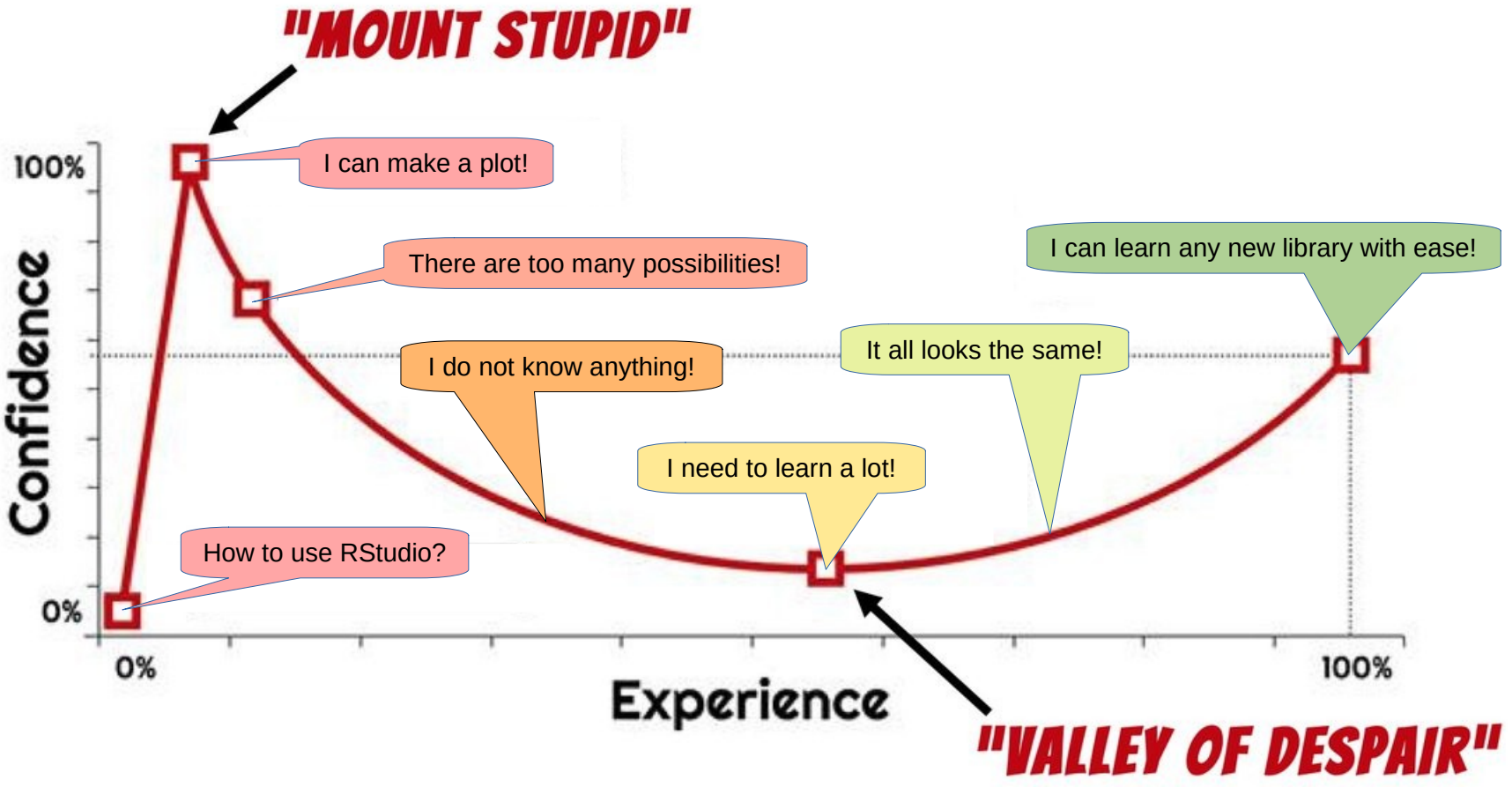
Agenda

1. Cognitive biases in learning.
2. What is and why R?
3. Data Science and R.
4. Exercise – plots and data manipulation.
5. Artificial Intelligence, Machine Learning and AutoML.
6. Ways of using ML in R.
7. Exercise – h2o.
8. Introduction to Shiny.
9. Exercise – Shiny ML application.

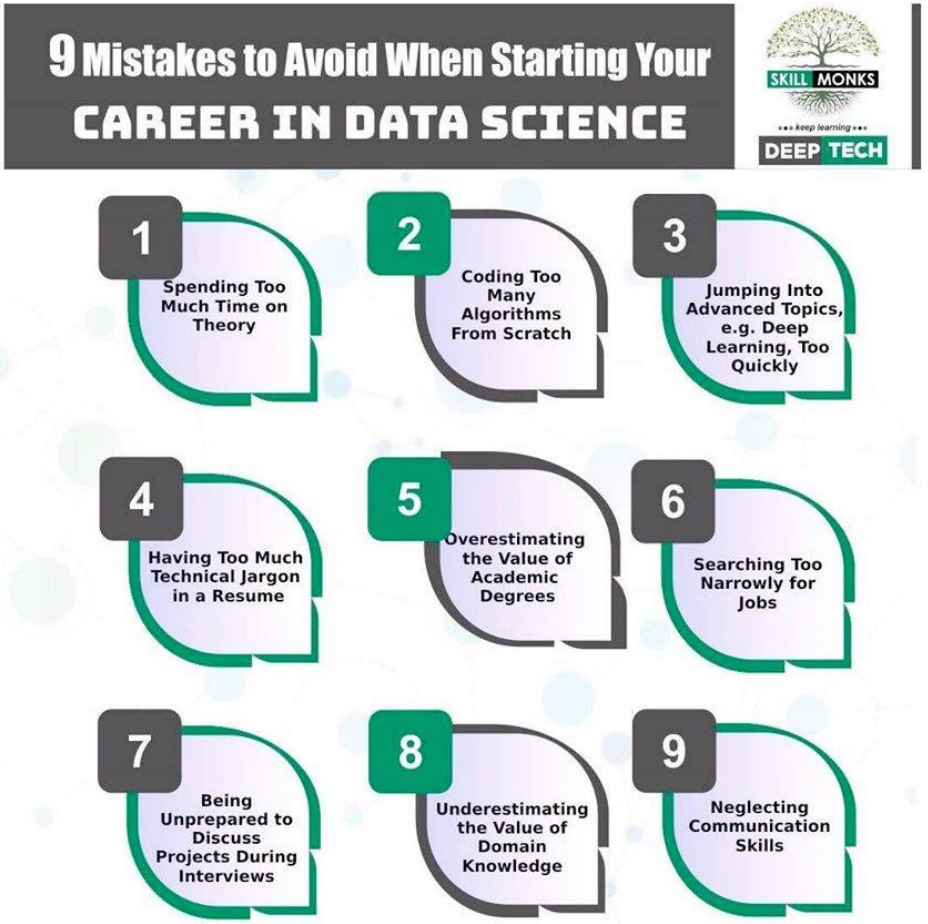
The Dunning–Kruger effect



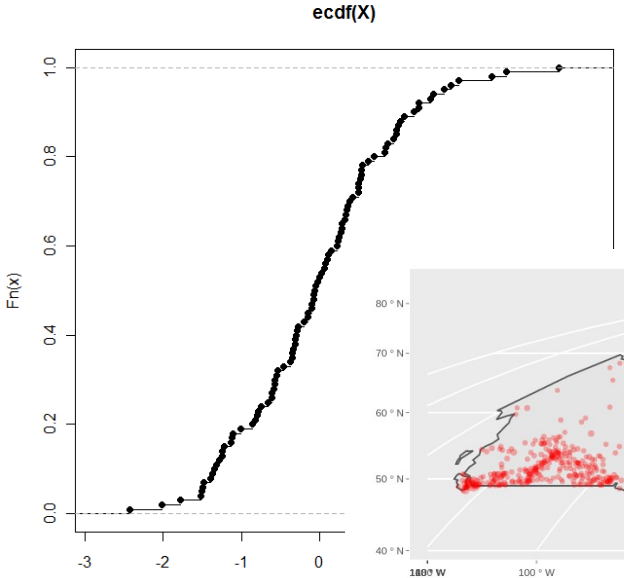
The Dunning–Kruger effect in R

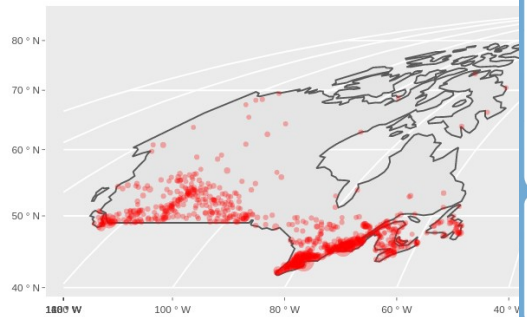


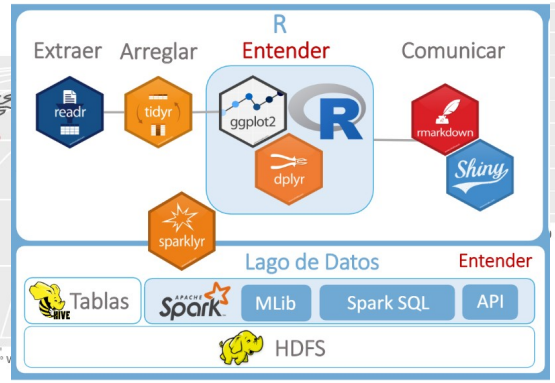
Some useful info

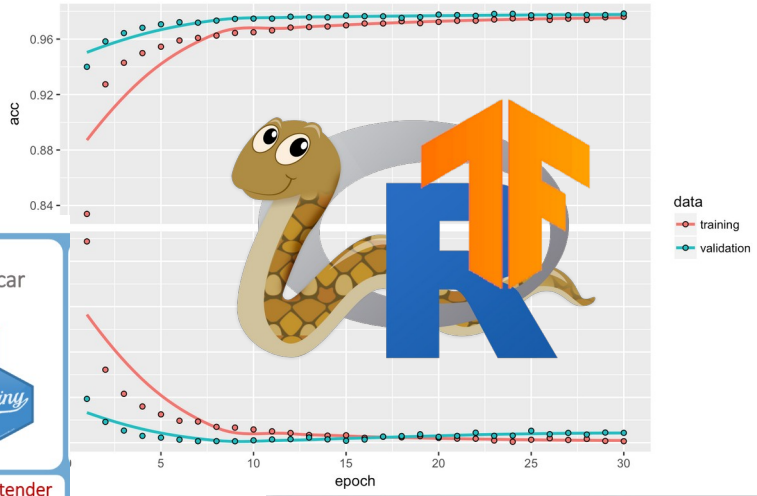


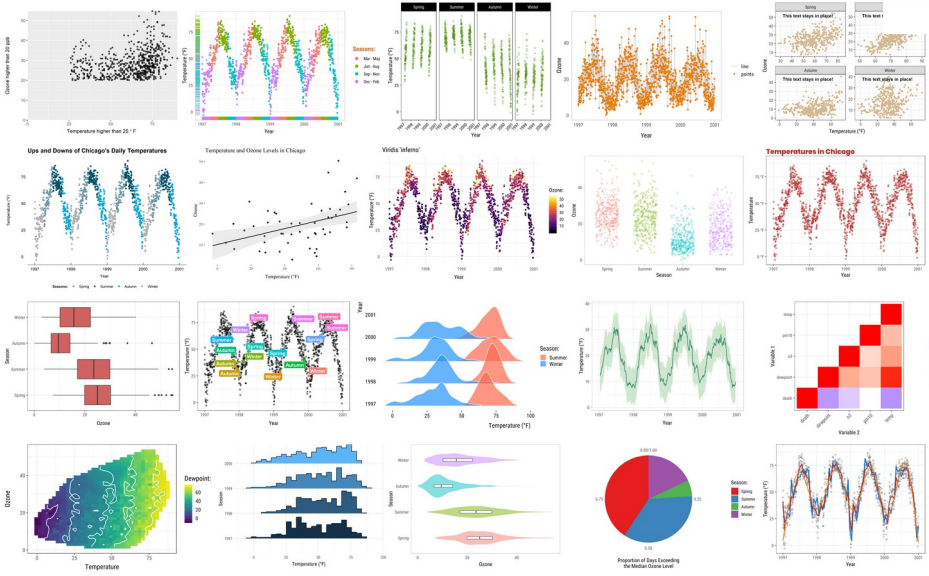
What is R?

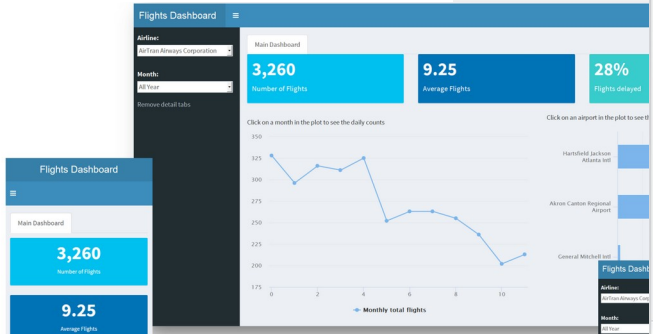




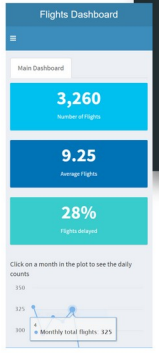




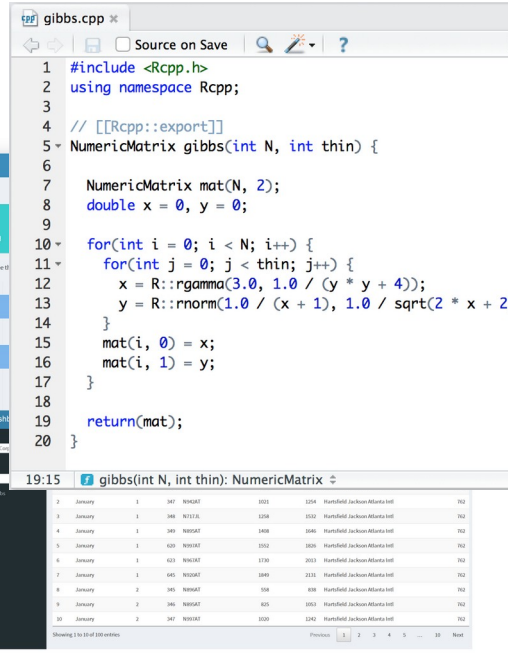




Interactive dashboards

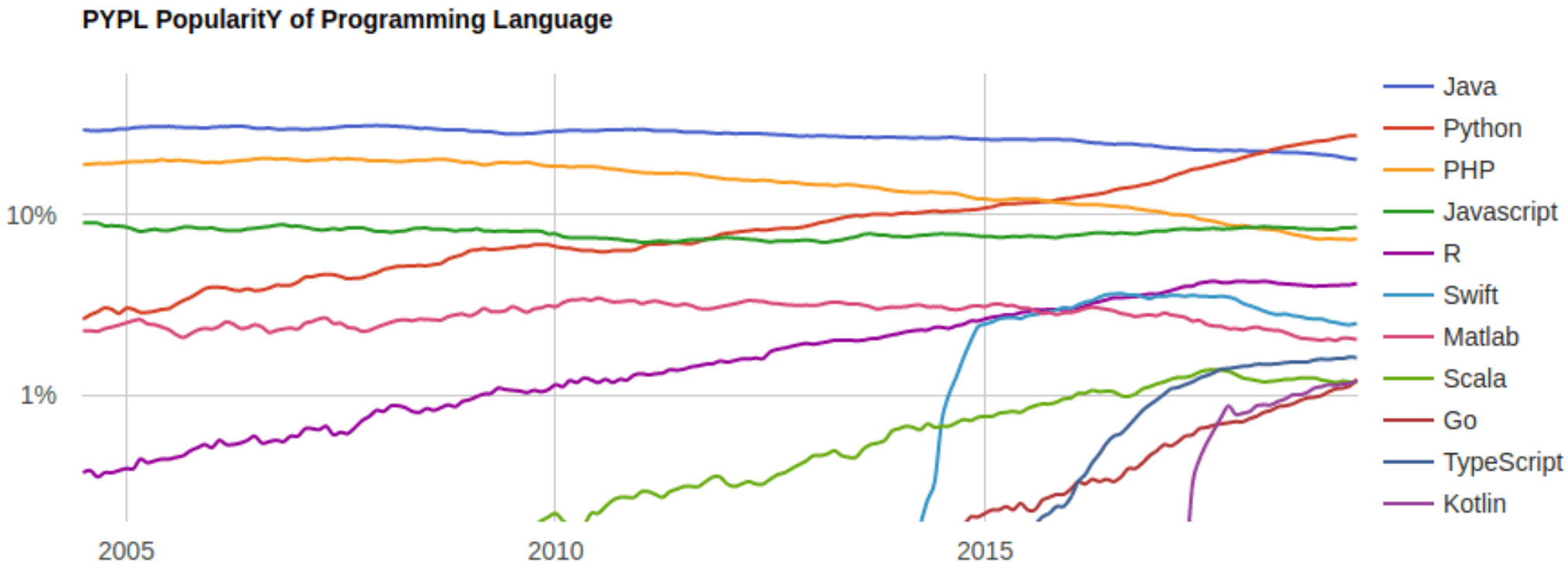


Mobile ready



Drill-down reporting

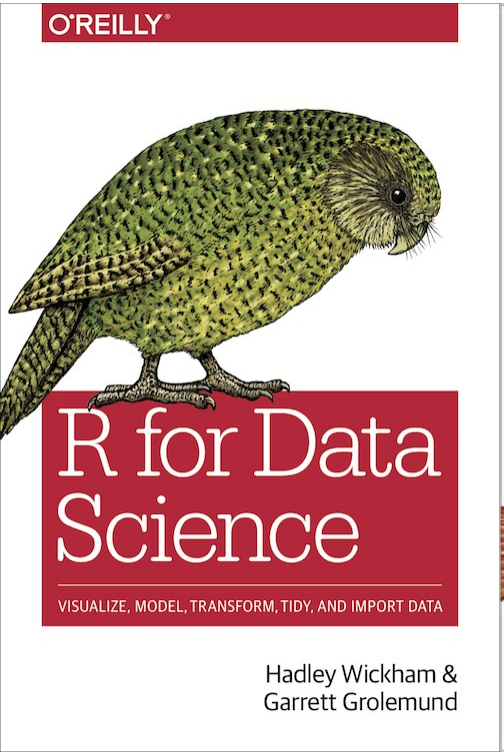
Why R?



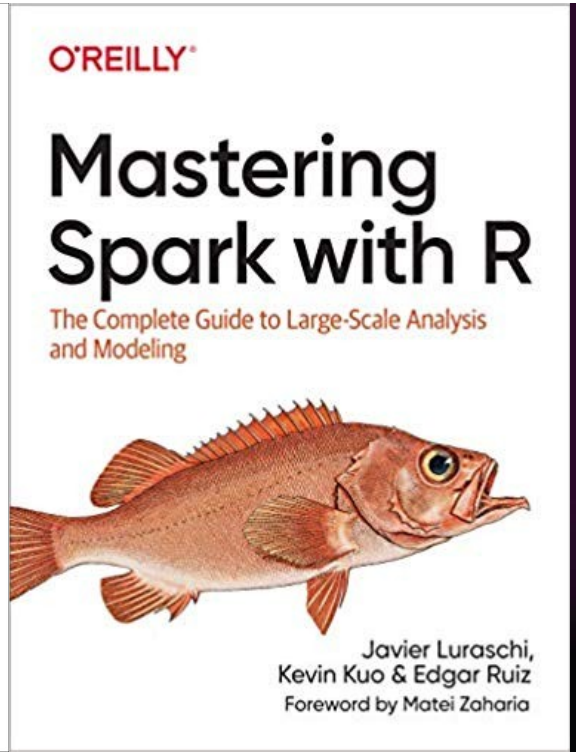
Source: https://www.valuecoders.com/blog/technology-and-apps/programming-trends-2017-it-outsourcing-companies/

Information sources

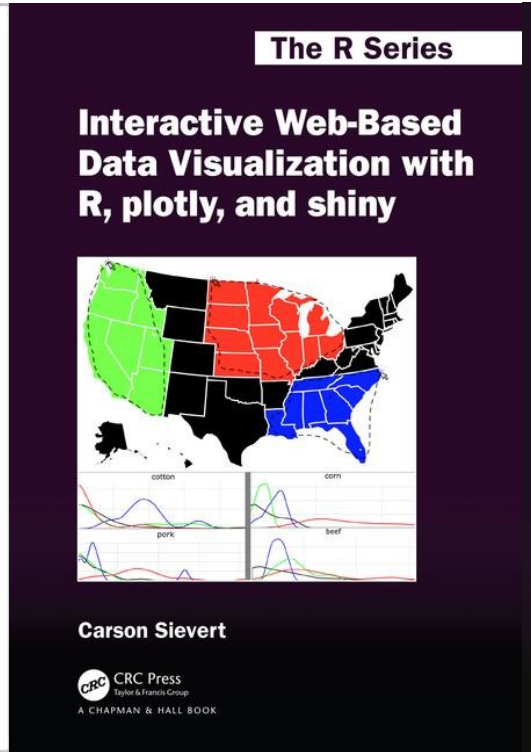
r4ds.had.co.nz



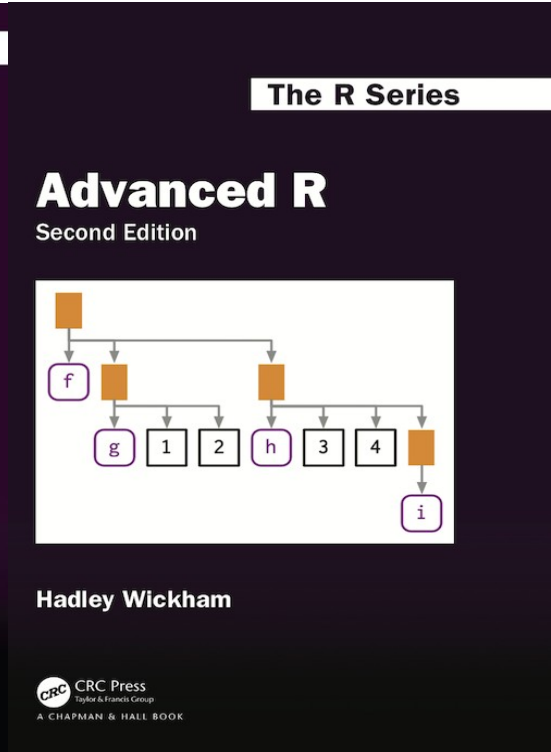
therinspark.com



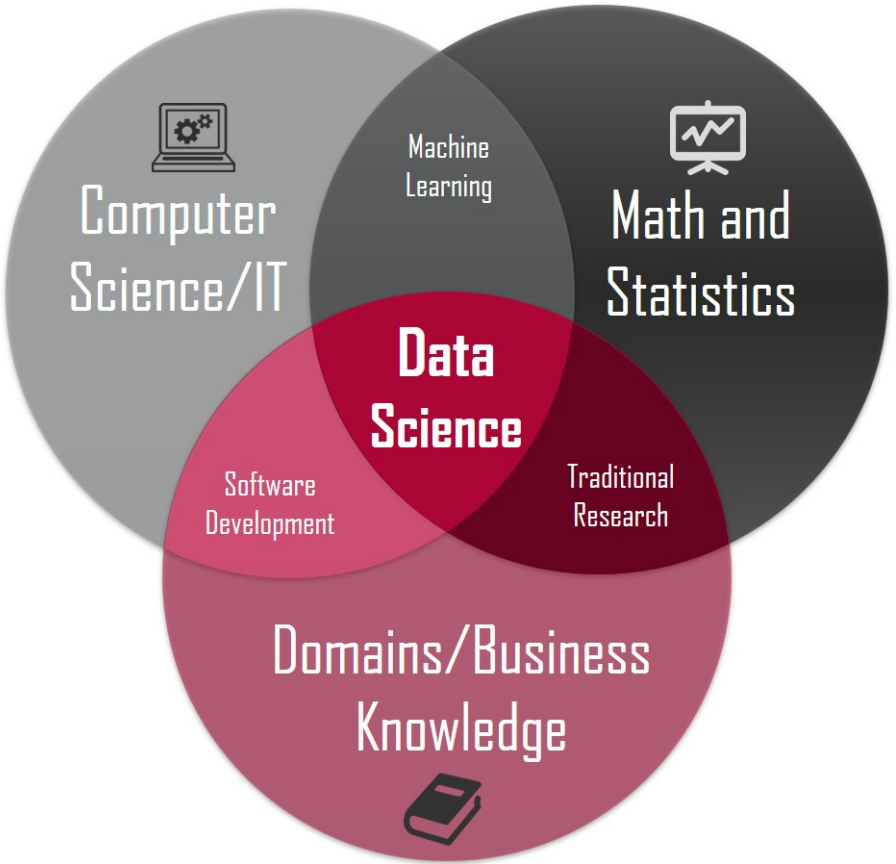
plotly-r.com

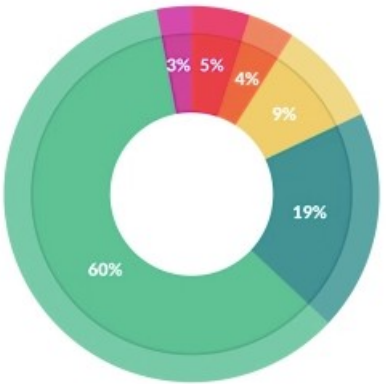


adv-r.hadley.nz



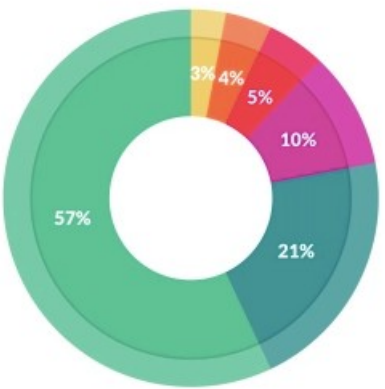
What is Data Science all about?





What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets; 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%



What's the least enjoyable part of data science?

- Building training sets: 10%
- Cleaning and organizing data: 57%
- Collecting data sets: 21%
- Mining data for patterns: 3%
- Refining algorithms: 4%
- Other: 5%

source: https://towardsdatascience.com/q-a-common-questions-in-data-science-7cd7f9d82a8d

source: http://www.vizwiz.com/2016/06/data-scientists-need-alteryx.html

Tips for starters

RStudio®

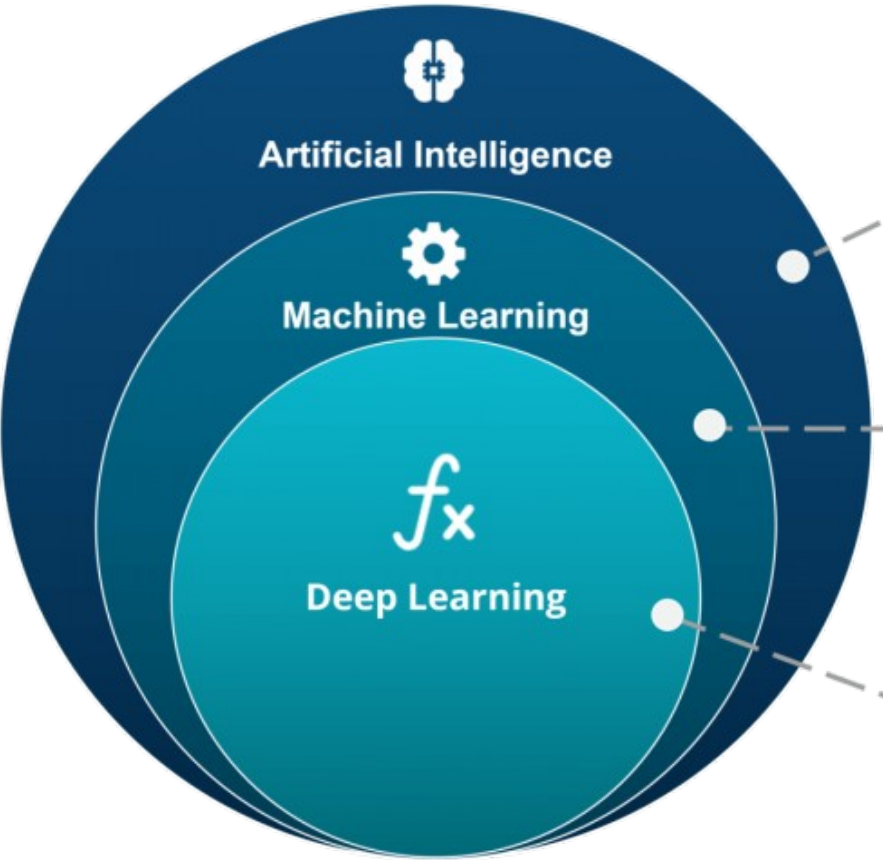
- R != RStudio
- RStudio are monopolists.
- Everything they make is of excellent quality.
- Use their products (tidyverse, shiny, sparklyr, tensorflow etc.).
- Otherwise, use **fast and reputable libraries** rather that slow and ugly functions from r-base.
- (except for statistical packages and other minor exceptions)

Demo and exercise

Using the data taken from Eurostat prepare TWO graphs showing the differences in employment level between males and females among 5 EU countries of your choice in the last 2 years.

1. Quick language and IDE introduction
2. Data loading (readxl)
3. Data processing (dplyr, stringi)
4. Spatial visualizations (sf, leaflet)
5. Graphs creation

AI – ML – DL

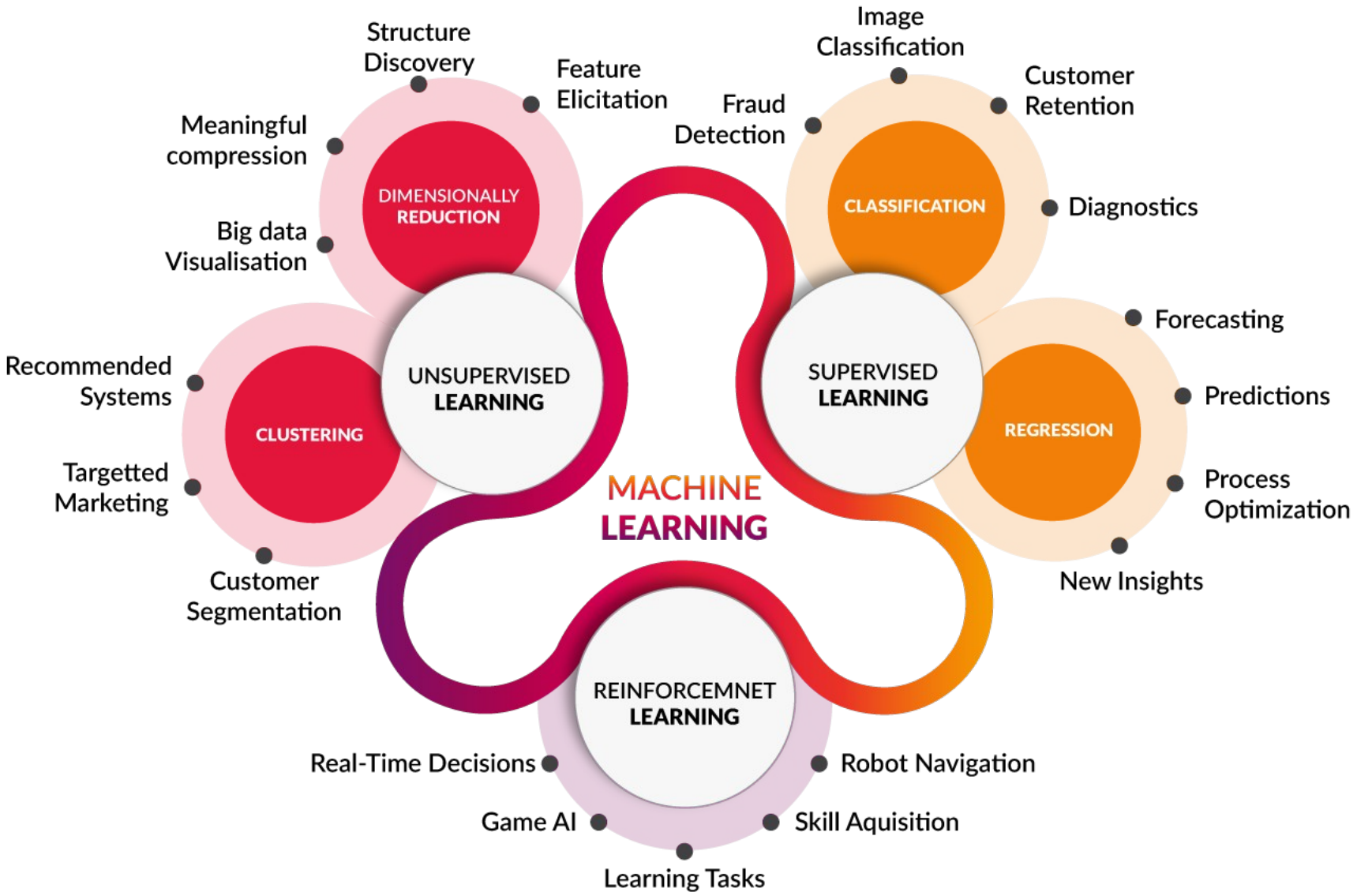


ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

source: https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/



source: https://towardsdatascience.com/coding-deep-learning-for-beginners-types-of-machine-learning-b9e651e1ed9d

The problem of ML in R

There is also inconsistency in how different packages return predictions. *Most* R packages use the `predict()` function to make predictions on new data. If we want to get class probabilities for our logistic regression model, using `predict(obj, newdata, type = "response")` will return a vector of probabilities for the second level of our factor. However, this convention can be wildly inconsistent across R packages. Examples are:

FUNCTION	PACKAGE	CODE
`glm`	`stats`	`predict(obj, type = "response")`
`lda`	`MASS`	`predict(obj)`
`gbm`	`gbm`	`predict(obj, type = "response", n.trees)`
`mda`	`mda`	`predict(obj, type = "posterior")`
`rpart`	`rpart`	`predict(obj, type = "prob")`
`Weka`	`RWeka`	`predict(obj, type = "probability")`
`logitboost`	`LogitBoost`	`predict(obj, type = "raw", nIter)`
`pamr.train`	`pamr`	`pamr.predict(obj, type = "posterior")`

An added complication is that some models can create predictions across multiple *submodels* at once. For example, boosted trees fit using i iterations can produce predictions using less than i iterations (effectively creating a different prediction model). This can lead to further inconsistencies.

These issues, in aggregate, can be grating. Sometimes it might feel like:

> *"Is R working for me or am I working for R?"*

The caret way, part 1

```r
set.seed(825)
rdaFit <- train(Class ~ ., data = training,
                method = "rda",
                trControl = fitControl,
                tuneLength = 4,
                metric = "ROC")
rdaFit
```

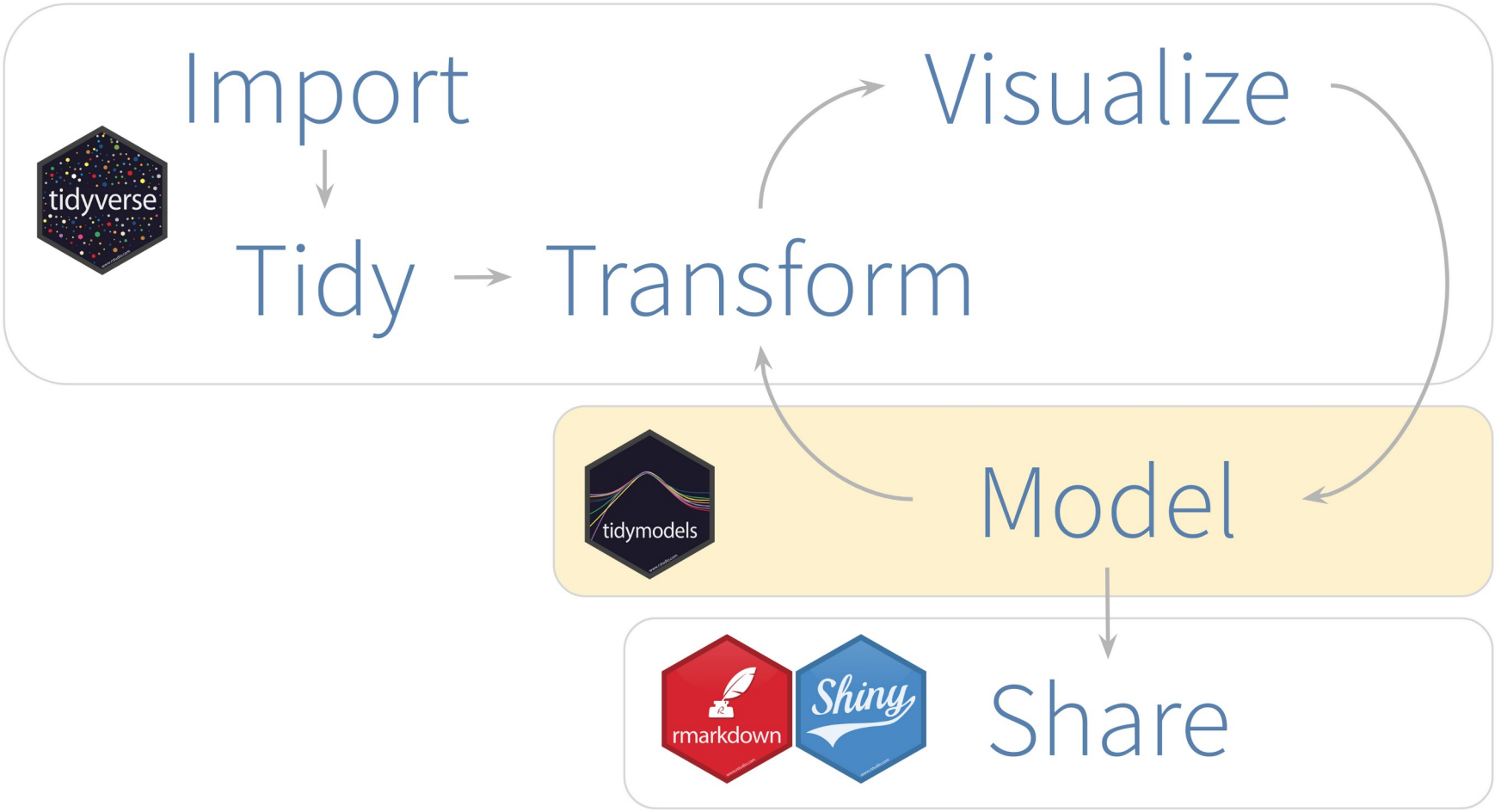
```
## Regularized Discriminant Analysis
##
## 157 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 141, 142, 141, 142, 141, 142, ...
## Resampling results across tuning parameters:
##
##   gamma      lambda     ROC        Sens       Spec
##   0.0000000  0.0000000  0.6426029  0.9311111  0.3364286
##   0.0000000  0.3333333  0.8543564  0.8076389  0.7585714
##   0.0000000  0.6666667  0.8596577  0.8083333  0.7766071
##   0.0000000  1.0000000  0.7950670  0.7677778  0.6925000
##   0.3333333  0.0000000  0.8509276  0.8502778  0.6914286
##   0.3333333  0.3333333  0.8650372  0.8676389  0.6866071
##   0.3333333  0.6666667  0.8698115  0.8604167  0.6941071
##   0.3333333  1.0000000  0.8336930  0.7597222  0.7542857
##   0.6666667  0.0000000  0.8600868  0.8756944  0.6482143
##   0.6666667  0.3333333  0.8692981  0.8794444  0.6446429
##   0.6666667  0.6666667  0.8678547  0.8355556  0.6892857
##   0.6666667  1.0000000  0.8277133  0.7445833  0.7448214
##   1.0000000  0.0000000  0.7059797  0.6888889  0.6032143
##   1.0000000  0.3333333  0.7098313  0.6830556  0.6101786
##   1.0000000  0.6666667  0.7129489  0.6672222  0.6173214
##   1.0000000  1.0000000  0.7193031  0.6626389  0.6296429
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were gamma = 0.3333333 and lambda
##  = 0.6666667.
```

The caret way, part 2

```r
set.seed(825)
svmFit <- train(Class ~ ., data = training,
                method = "svmRadial",
                trControl = fitControl,
                preProc = c("center", "scale"),
                tuneLength = 8,
                metric = "ROC")
svmFit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 157 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## Pre-processing: centered (60), scaled (60)
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 141, 142, 141, 142, 141, 142, ...
## Resampling results across tuning parameters:
##
##   C      ROC        Sens       Spec
##    0.25  0.8438318  0.7373611  0.7230357
##    0.50  0.8714459  0.8083333  0.7316071
##    1.00  0.8921354  0.8031944  0.7653571
##    2.00  0.9116171  0.8358333  0.7925000
##    4.00  0.9298934  0.8525000  0.8201786
##    8.00  0.9318899  0.8684722  0.8217857
##   16.00  0.9339658  0.8730556  0.8205357
##   32.00  0.9339658  0.8776389  0.8276786
##
## Tuning parameter 'sigma' was held constant at a value of 0.01181293
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01181293 and C = 16.
```

The tidymodels way, part 1



The tidymodels way, part 2

engine	mode	model	class	conf_int	numeric	pred_int	prob	quan
C5.0	classification	boost_tree()	✔	×	×	×	✔	×
	classification	decision_tree()	✔	×	×	×	✔	×
earth	classification	mars()	✔	×	×	×	✔	×
	regression	mars()	×	×	✔	×	×	×
flexsurv	regression	surv_reg()	×	×	✔	×	×	✔
glm	classification	logistic_reg()	✔	✔	×	×	✔	×
glmnet	classification	logistic_reg()	✔	×	×	×	✔	×
	classification	multinom_reg()	✔	×	×	×	✔	×
	regression	linear_reg()	×	×	✔	×	×	×
keras	classification	logistic_reg()	✔	×	×	×	✔	×
	classification	mlp()	✔	×	×	×	✔	×
	classification	multinom_reg()	✔	×	×	×	✔	×
	regression	linear_reg()	×	×	✔	×	×	×
	regression	mlp()	×	×	✔	×	×	×
kernlab	classification	svm_poly()	✔	×	×	×	✔	×
	classification	svm_rbf()	✔	×	×	×	✔	×
	regression	svm_poly()	×	×	✔	×	×	×
	regression	svm_rbf()	×	×	✔	×	×	×
kknn	classification	nearest_neighbor()	✔	×	×	×	✔	×
	regression	nearest_neighbor()	×	×	✔	×	×	×
lm	regression	linear_reg()	×	✔	✔	✔	×	×
nnet	classification	mlp()	✔	×	×	×	✔	×
	regression	mlp()	×	×	✔	×	×	×
parsnip	classification	null_model()	✔	×	×	×	✔	×
	regression	null_model()	×	×	✔	×	×	×
randomForest	classification	rand_forest()	✔	×	×	×	✔	×
	regression	rand_forest()	×	×	✔	×	×	×
ranger	classification	rand_forest()	✔	✔	×	×	✔	×
	regression	rand_forest()	×	✔	✔	×	×	×
rpart	classification	decision_tree()	✔	×	×	×	✔	×
	regression	decision_tree()	×	×	✔	×	×	×
spark	classification	boost_tree()	✔	×	×	×	✔	×
	classification	decision_tree()	✔	×	×	×	✔	×
	classification	logistic_reg()	✔	×	×	×	✔	×
	classification	multinom_reg()	✔	×	×	×	✔	×
	classification	rand_forest()	✔	×	×	×	✔	×
	regression	boost_tree()	×	×	✔	×	×	×
	regression	decision_tree()	×	×	✔	×	×	×
	regression	linear_reg()	×	×	✔	×	×	×
	regression	rand_forest()	×	×	✔	×	×	×
stan	classification	logistic_reg()	✔	✔	×	✔	✔	×
	regression	linear_reg()	×	✔	✔	✔	×	×
survival	regression	surv_reg()	×	×	✔	×	×	✔
xgboost	classification	boost_tree()	✔	×	×	×	✔	×
	regression	boost_tree()	×	×	✔	×	×	×

What is H2O?

```
library(h2o)
h2o.init()

H2O is not running yet, starting it now...

java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)

Starting H2O JVM and connecting: ... Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         2 seconds 124 milliseconds
    H2O cluster version:        3.20.0.8
    H2O cluster total nodes:    1
    H2O cluster total memory:   3.56 GB
    H2O cluster total cores:    8
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
    R Version:                  R version 3.5.1 (2018-07-02)
```

Model training with H2O

- Gradient Boosted models with `h2o.gbm()` & `h2o.xgboost()`
- Generalized linear models with `h2o.glm()`
- Random Forest models with `h2o.randomForest()`
- Neural Networks with `h2o.deeplearning()`

```
gbm_model <- h2o.gbm(x = x, y = y,
                     training_frame = train, validation_frame = valid)

Model Details:
==============

H2OMultinomialModel: gbm
Model ID:  GBM_model_R_1540736041817_1
Model Summary:
number_of_trees number_of_internal_trees model_size_in_bytes min_depth
             50                      150               24877         2
max_depth mean_depth min_leaves max_leaves mean_leaves
        5    4.72000          3         10     8.26667
```

Evaluate model performance with H2O

- Model **performance**

```
perf <- h2o.performance(gbm_model, test)

h2o.confusionMatrix(perf)

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
       1 2 3  Error     Rate
1      7 0 1 0.1250 =  1 / 8
2      0 8 0 0.0000 =  0 / 8
3      0 0 5 0.0000 =  0 / 5
Totals 7 8 6 0.0476 = 1 / 21

h2o.logloss(perf)

[1] 0.2351779
```

- **Predict** new data

```
h2o.predict(gbm_model, test)
```

Hyperparameters in H2O models

- Hyperparameters for **Gradient Boosting**:

```
?h2o.gbm
```

- `ntrees`: Number of trees. Defaults to 50.
- `max_depth`: Maximum tree depth. Defaults to 5.
- `min_rows`: Fewest allowed (weighted) observations in a leaf. Defaults to 10.
- `learn_rate`: Learning rate (from 0.0 to 1.0) Defaults to 0.1.
- `learn_rate_annealing`: Scale the learning rate by this factor after each tree (e.g., 0.99 or 0.999) Defaults to 1.

Automatic Machine Learning (AutoML)

- **Automatic tuning of algorithms**, in addition to hyperparameters
- AutoML makes model tuning and optimization much **faster and easier**
- AutoML only needs a **dataset**, a **target** variable and a **time or model number limit** for training

AutoML in H2O

AutoML compares

- Generalized **Linear Model** (GLM)
- (Distributed) **Random Forest** (DRF)
- Extremely **Randomized Trees** (XRT)
- Extreme **Gradient Boosting** (XGBoost)
- **Gradient Boosting** Machines (GBM)
- **Deep Learning** (fully-connected multi-layer artificial neural network)
- Stacked **Ensembles** (of all models & of best of family)

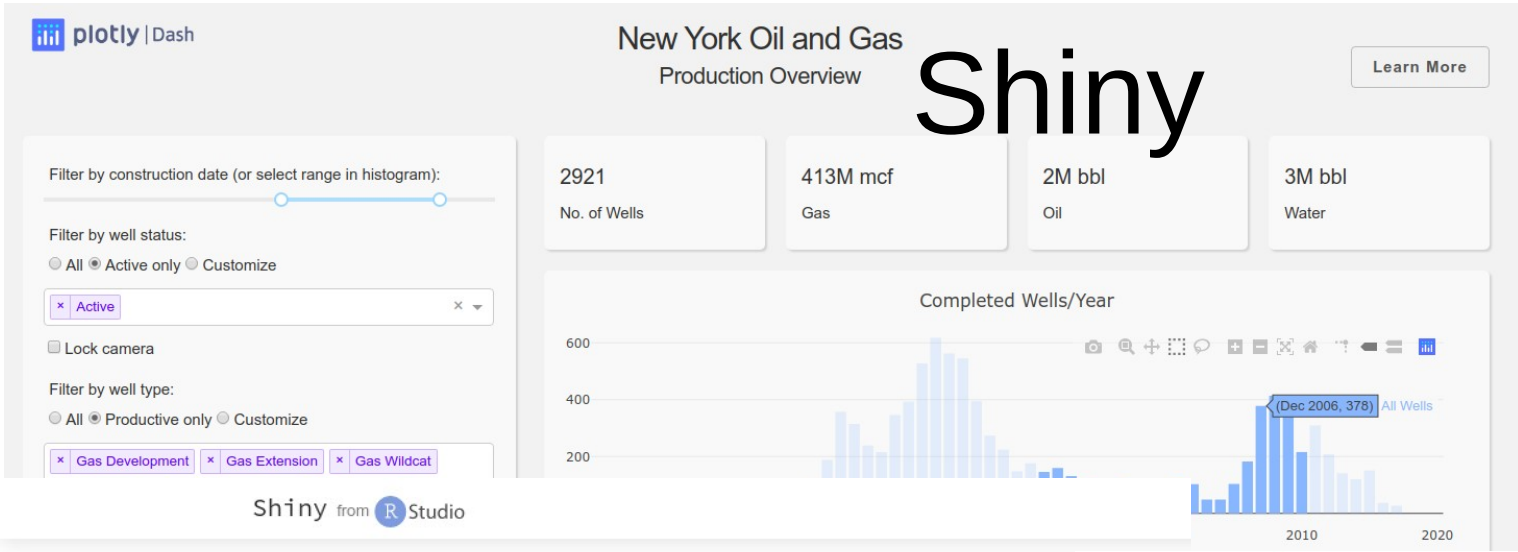
Using AutoML with H2O

- `h2o.automl` function

```
automl_model <- h2o.automl(x = x,
                           y = y,
                           training_frame = train,
                           validation_frame = valid,
                           max_runtime_secs = 60,
                           sort_metric = "logloss",
                           seed = 42)
```

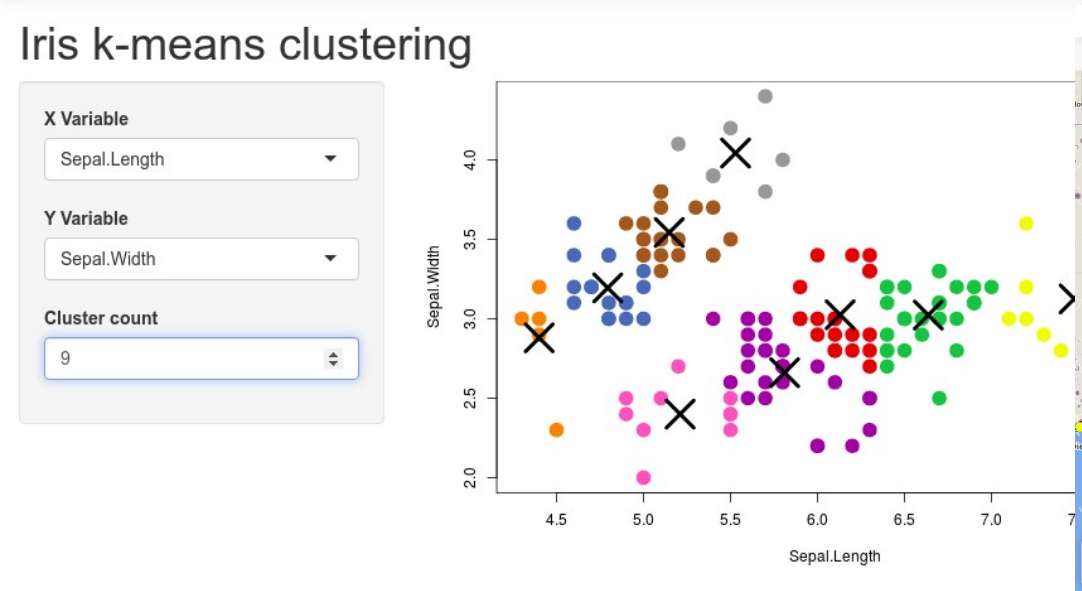
- returns a **leaderboard** of all models, **ranked** by the chosen metric (here "logloss")

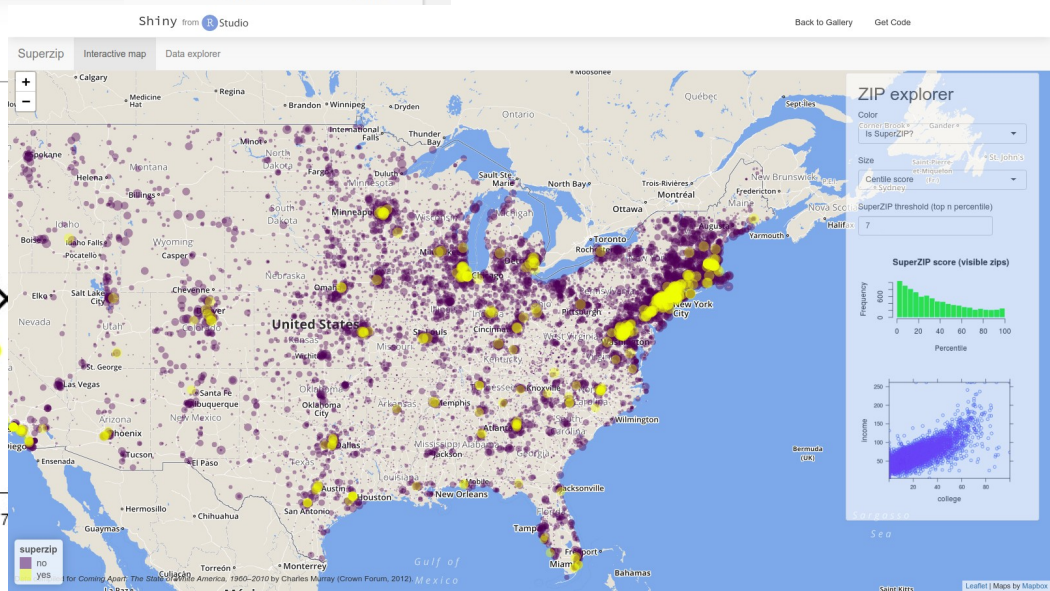
```
Slot "leader":
Model Details:
==============

H2OMultinomialModel: gbm
Model Summary:
 number_of_trees number_of_internal_trees model_size_in_bytes min_depth
             189                       567                65728         1
 max_depth mean_depth min_leaves max_leaves mean_leaves
         5    2.96649          2          6     4.20988
```


plotly | Dash
New York Oil and Gas
Production Overview
Learn More
Filter by construction date (or select range in histogram):
Filter by well status:
All Active only Customize
Active
Lock camera
Filter by well type:
All Productive only Customize
Gas Development
Gas Extension
Gas Wildcat
2921
No. of Wells
413M mcf
Gas
2M bbl
Oil
3M bbl
Water
Completed Wells/Year
600
400
200
(Dec 2006, 378)
All Wells
2010
2020
Shiny from RStudio

Shiny


Shiny
www.rstudio.com


Iris k-means clustering
X Variable
Sepal.Length
Y Variable
Sepal.Width
Cluster count
9
Sepal.Width
Sepal.Length
4.5
5.0
5.5
6.0
6.5
7.0
2.0
2.5
3.0
3.5
4.0


Shiny from RStudio
Back to Gallery
Get Code
Superzip
Interactive map
Data explorer
ZIP explorer
Color
Is SuperZIP?
Size
Centile score
SuperZIP threshold (top n percentile)
SuperZIP score (visible zips)
Percentile
Frequency
Income
college
United States
superzip
no
yes
Gulf of Mexico
Sargasso Sea

The End

Thank you for listening!

All the materials (this presentation and sample code) will be added to the GitHub repository.