CSX415
Data Science Principals and Practice
Solution Deployment

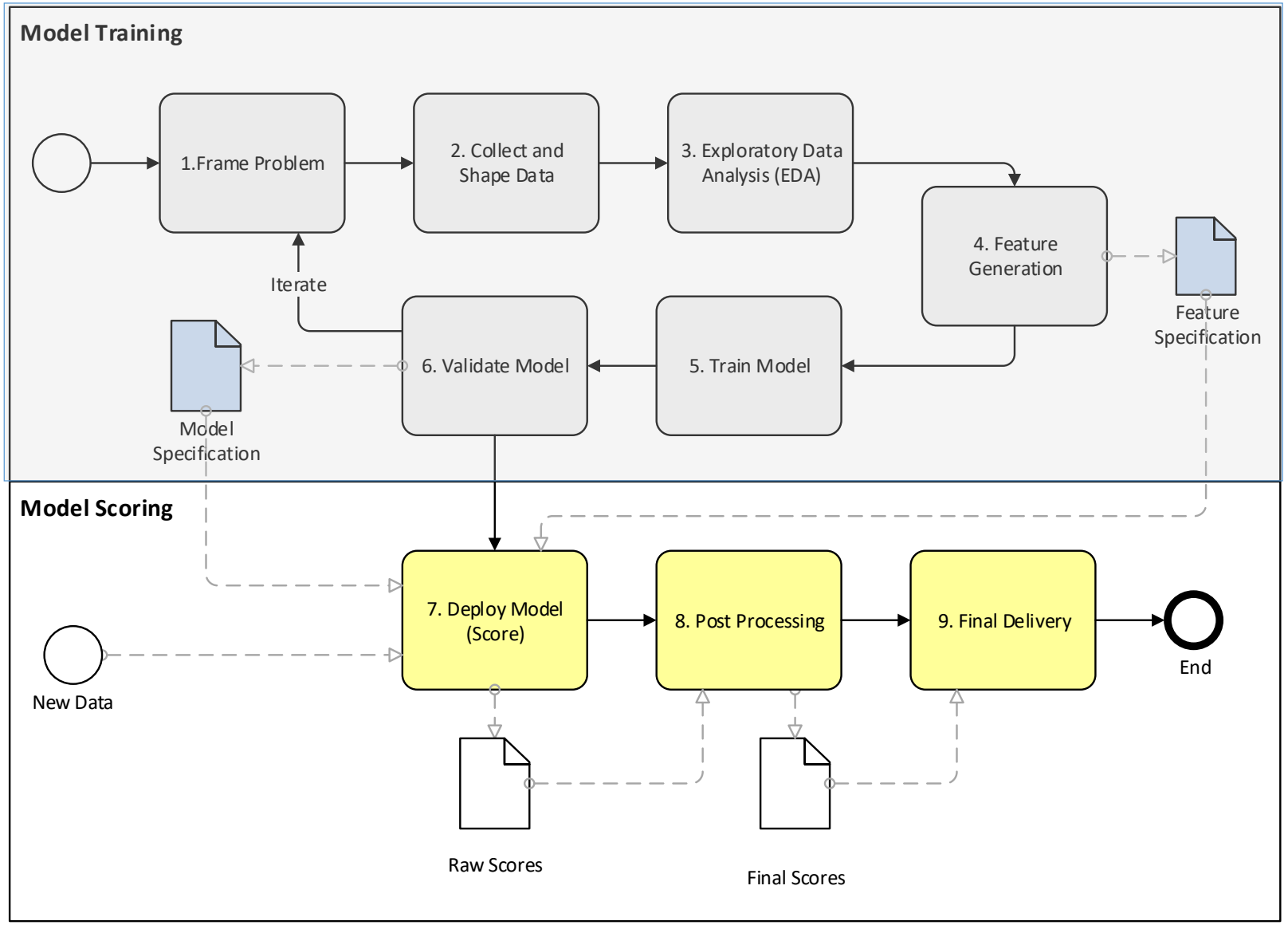
Christopher Brown
U.C. Berkeley / Decision Patterns LLC
Spring 2018

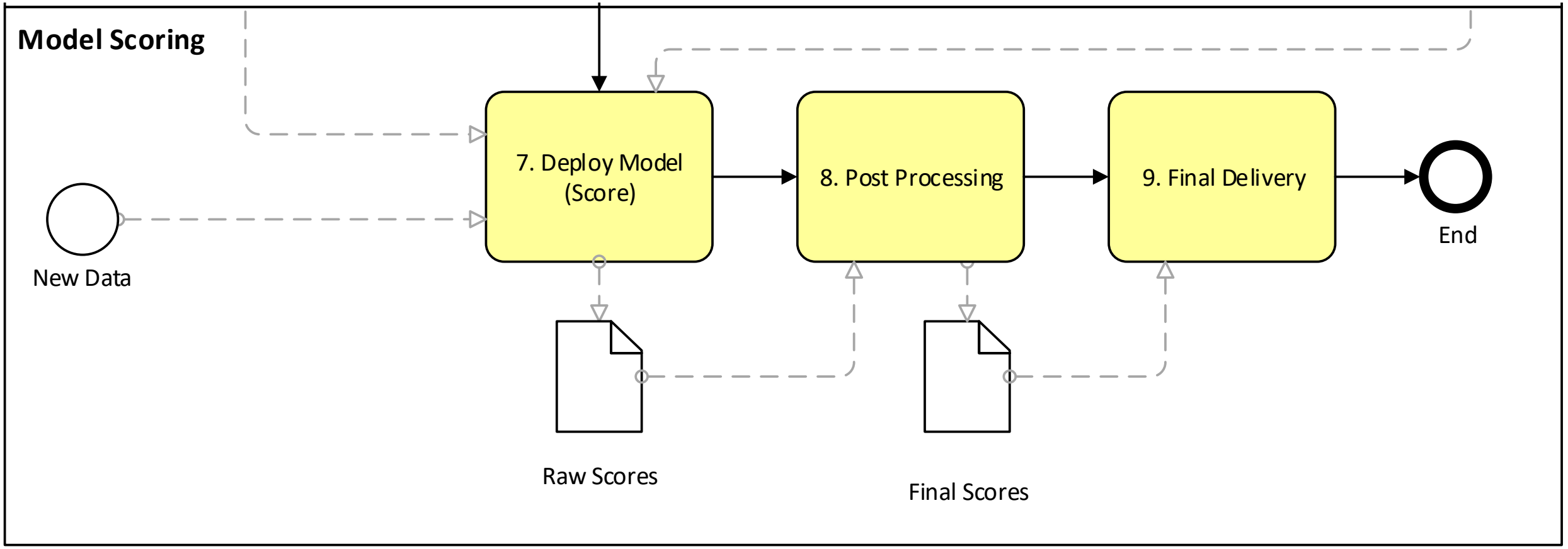
Berkeley
UNIVERSITY OF CALIFORNIA

DEPLOYMENT

Make a model available to a (larger) group of users

Putting it all together




Model Scoring
New Data
7. Deploy Model (Score)
8. Post Processing
9. Final Delivery
End
Raw Scores
Final Scores

ML Models

Produce

One or more scores

(numeric values)

… used to (help) make a decision

… btw, this is just more information.

ML Solutions

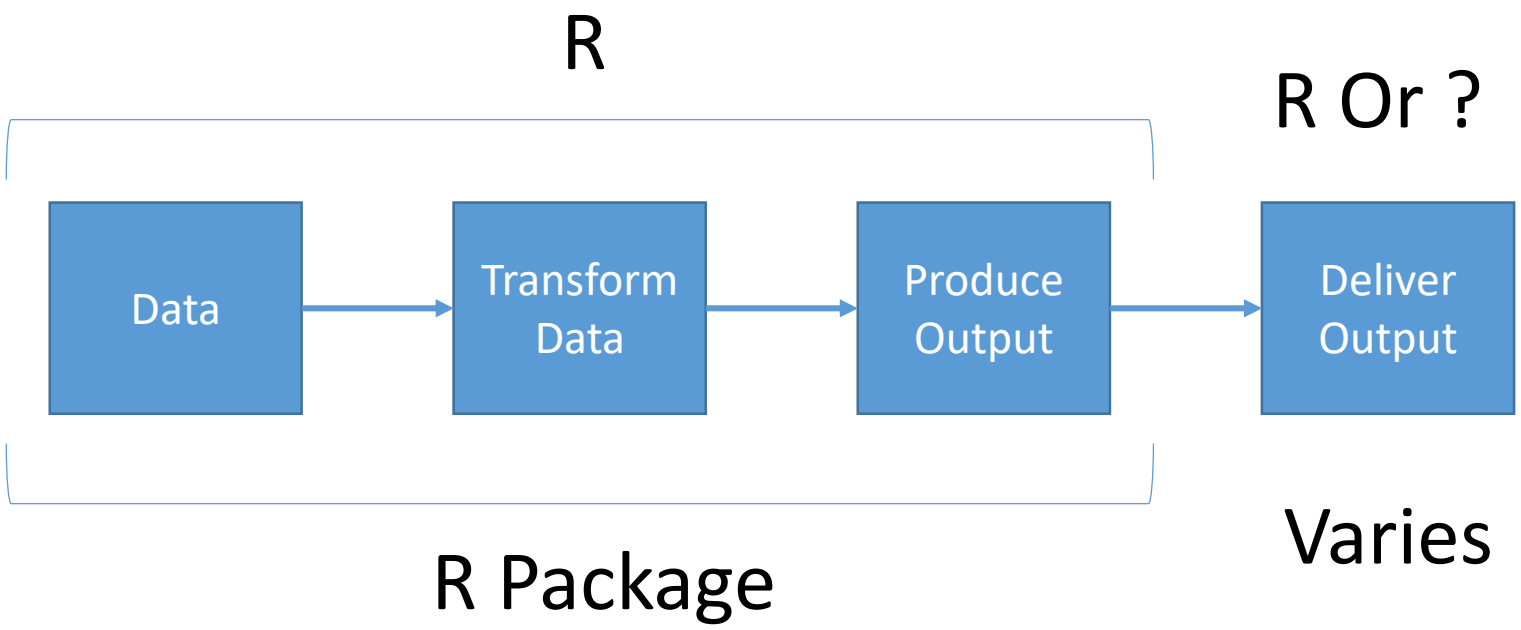
Provide

scores,
calibrations,
inferences,
charts,
estimates,
supporting documentation

… used to (help) make a decision
… and elucidate, explain and describe it

(this may overlap with some of the assets used to evaluate model training)

General Framework



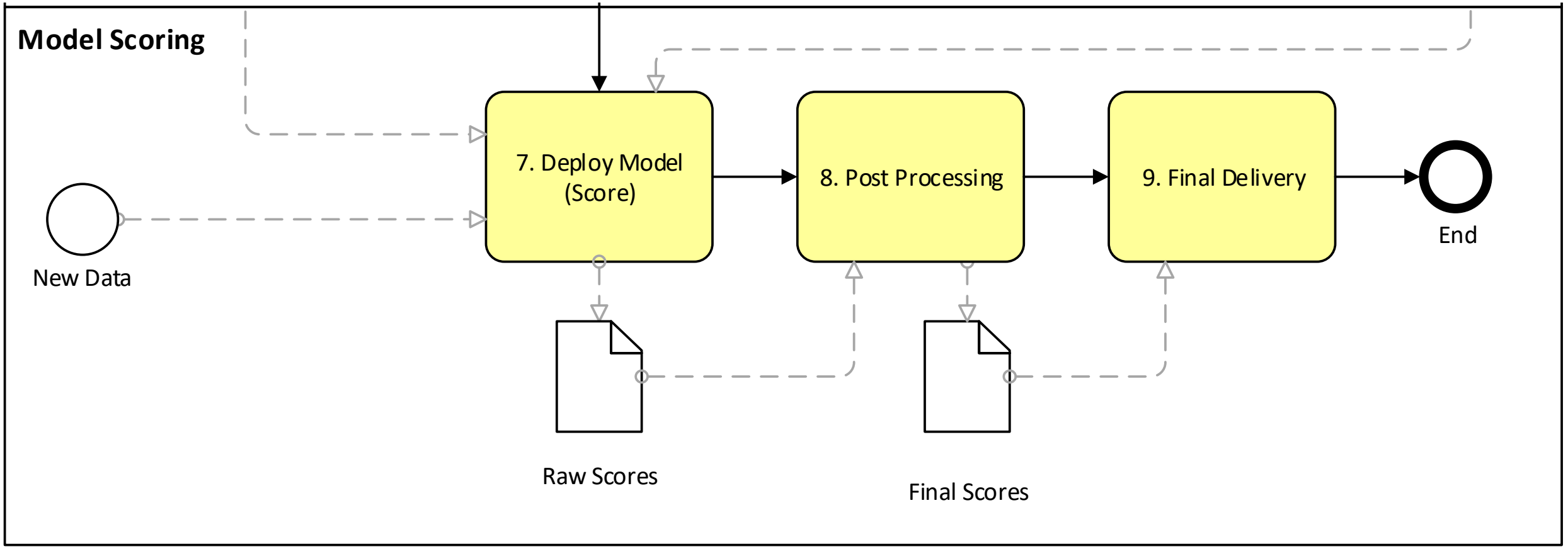
Who is/are the users (consumer)?

Automated Systems

- Require fewer deployment assets
- may be able to consume raw scores.

People

- Require more deployments assets
- Require contextual information
- Generally skeptical
  - Reluctant to give up control
- Curious (why – this)
  - Like to break things
- Have preconceived notions of how things do or should work.


Model Scoring
New Data
7. Deploy Model (Score)
8. Post Processing
9. Final Delivery
End
Raw Scores
Final Scores

Popular Deployment Patterns

Access Point	User	Deployment Technology	Consideration(s)
Web	Client, Non-technical	Shiny	Interface changes with most model updates
Excel	Client, Excel-user	OpenCPU + VB REST libs	Excel quirks, e.g. Auto calculate
RESTful API	Non-R App. Developer	OpenCPU	Marshalling JSON data to from R
Web **RESTful API** **Native R UI** **R / Rstudio**	R Developer	Shiny OpenCPU Rserve libraries / git / packrat	
command line **cron**	Recurrent Process Manual Automated	Rscript + optigrab or	Logging, trapping errors, default parameters, notifications (long processes)
Database or custom application	Proprietary Application • Database • BI Application	SQL Server (SSAS), R Oracle Enterprise, SAP HANA/PAL	Your mileage varies

Deploy to

R (easiest)

Who: User proficient in R

Requirements

- Functions for:
  fetching > shaping > feature generation > scoring > generating outputs
- Additional functions for other assets
- Documentation of use README.md

Usage:

```
R> score( fit, newdata )
R> graph_outliers( fit, newdata )
```

Deploy to

R (easiest)

What to build

- Instructions (README.md) on installing the packages
- R package(s)

```
|- README.md
|- pkgs/
|   |- mymodel/
|   |   |- …
```

Considerations: Ensure the consumer has access to all the packages necessary

Deploy to

Command Line

easy + cronable

Who:

Users or system with a system command prompt

Requirements:

- All requirements for **Deploy to R**
- Functions for creating assets for user
- Command line program(s)

Deploy to

Command Line

easy + cronable

What to build

- Everything from **Deploy to R**
- Docs for ***installation*** and ***usage***
- Command line program **Rscript**

```
|- README.md
|- app
|   |- .packrat/
|   |- myprogram
```

Considerations

- Logging, default parameters, trapping errors, notification

Resources

packrat

SO:parsing-command-line-arguments-in-r-scripts

Deploy to

Command-Line write back to database

very common

Who:

Automated system with a trigger (usually)

Requirements:

- All for **Deploy to Command Line**
- Package functions for writing to database, usually bulk load
- Command line program(s)

Deploy to

Command-Line write back to database

very common

What to build

- All from **Deploy to Command Line**
- DB write methods in R package to
  - write directly to DB
  - write to FS and do BULK load.

```
|- README.md
|- pkgs/
|   |- mymodel/
|   |   |- R/write-scores.R
```

Considerations

- Datatype mappings

Resources

- https://db.rstudio.com/
- Packages: DBI, odbc, ROBC

Deploy to

Web API

universal

Who:

Users/Systems via web integration

Requirements

- All Requirements for **Deploy to R**
- Server reachable by users
- Functions for endpoints

Deploy to

Web API

universal

What to build

- Instructions (README.md) on running server
- Web API functions
- Command line runtime

```
|- README.md
|- pkgs/
|   |- mywebapi/
|   |   |- R/run.R
|- app/start-api.r
```

Resources (packages):

- Plummer (recommended)
- OpenCPU
- Rserve (older)

Deploy to

Web/mobile application

universal

Who:

Users via web/mobile intereface

Requirements

- All Requirements for **Deploy to R**
- Server reachable by users
- ui and server (optional) components

Deploy to

Web/mobile application

universal

SHINY

What to build

- Instructions (README.md)
- Shiny Application

```
|- README.md
|- app
|   |- server.R
    |- ui.R
    |- ...
```

Resources (packages):

- Shiny

Deploy to

Web/mobile application Via WebAPI

universal

Your Lang of Choice.

What to build

- Instructions (README.md)
- All of **Deploy to WebAPI**

```
|- README.md
|- app
|   |- index.html
    |- ...
```

Resources (packages):

Popular Deployment Patterns (summary)

Access Point	User	Deployment Technology	Consideration(s)
Web	Client, Non-technical	Shiny	Interface changes with most model updates
Excel	Client, Excel-user	OpenCPU + VB REST libs	Excel quirks, e.g. Auto calculate
RESTful API	Non-R App. Developer	OpenCPU	Marshalling JSON data to from R
Web **RESTful API** **Native R UI** **R / Rstudio**	R Developer	Shiny OpenCPU Rserve libraries / git / packrat	
command line **cron**	Recurrent Process Manual Automated	Rscript + optigrab or	Logging, trapping errors, default parameters, notifications (long processes)
Database or custom application	Proprietary Application • Database • BI Application	SQL Server (SSAS), R Oracle Enterprise, SAP HANA/PAL	Your mileage varies

Step 1
Define / Propose Use Case and User Stories

(though the later are often obvious)

User: Person, System or Both

Trigger(s):

- interactive (online)
- requests (a|synchronous)
- scheduled

Inputs

- source of all data, e.g. user|database|file
- how much data
- timeliness RT(online) vs cached(offline)

Output(s) …

Step 2
Stand-up “delivery”

Ensure that you can deliver each type of asset/output.

Build delivery code using fake data if necessary.

You don’t want to find out you can’t do something at the end.

Step 3
Build Plumbing

Build R functions for building outputs…

Step 4
Build Installers and document

Build code to install the solution and document this as well as all pre-requisites.

Deal with Bundls.

Step 5
Build tests

Test for outputs, use testthat

Test for delivery, varies

General Considerations

Model maintainence (i.e. updates) may/generally cause cascade of changes to the code

Clearly understand what model parameters are provide at time of scoring (online) and which can be pre-generated or cached (offline)

Generally need one R function for creating each output/asset and one for delivering it.

Do delivery first (with synthetic/fake data) before plumbing.