Springboard DSC Group Mentorship Session: Data Wrangling

AJ Sanchez–DSC Mentor // V2: August 25, 2023

Disclaimer

The information presented in these slides should not be construed as representing or coming from Springboard.

Such information is being shared for educational and informational purposes only and should not be construed as professional advice.

For professional Data Science and Software Engineering advice and services, please get in touch with the author of these slides by sending email to ajs@ExodusSoftServices.com

AJ Sanchez: Brief Profile

- DSC Mentor for 7 years
- Mentored 100’s of professionals, many of whom have been hired for data-related positions
- 20+ Years of experience as a **Software Engineering** Consultant
- 11 Years of experience as a **Data Science** Consultant
- Currently offers consulting expertise in both fields
- PhD and MSc in Computer Science from Rensselaer Polytechnic Institute (RPI)

Intended Audience

- Aspiring Data Scientists who are currently in one of the DSC programs
- Recent graduates from one of the DSC programs
- Preferably with some (even small) exposure to Python via–for instance–Anaconda, Jupyter Notebooks (e.g., via VS Code), and Pandas

Topic:
Data Wrangling

Format:

- **<u>This is a second Experimental Session</u>**
- The focus is on professional practices used in real-world projects
- Collaborative
- Hands-on: it would be great to do some live coding and sharing from your laptop!
- ***<u>Fast-paced … we only have 1 hour</u> …***

Collaborating Today

Rules of Engagement:

- Please send your questions to all to avoid repeated questions. We might not be able to answer all the questions during the session, but we will answer the rest off-line. **Thank you Dhara for your help!**
- When I pose a coding question, I will pause for X minutes (TBD; typically, 1-2) and then I will ask for a volunteer to either show their code, or explain how the person would code it. Unfortunately we will not have more time than that

Sample Context–But there can be others …

CLIENT

- **Has specific needs**
- **Wants effective results**
- **Wants efficient processes**
- **Has limited resources (time and money–for instance)**

INFRASTRUCTURE

- Data and Processes are in the Cloud–for instance: Redshift Data Warehouse, and AWS process-oriented tools (e.g., Lambda, etc.)
- Multiple data sources
- Required properties: security, availability, scalability, resilience, etc. Also known as "ilities"

DELIVERY

- New data source acquisition points
- New processes that run unattended, or on-demand
- Local apps and/or dashboards for end-users
- Preservation of properties: security, resilience, etc.

Before a single line of code is written …

Project Defined → **Project Architecture** → **Data Sources**

Contract in Place

- Legally binding document
- Defines goals, scope, deliverables, schedule of delivery, budget, etc
- Also covers other not-so-happy scenarios

Blueprint in Place

- Major components to be developed and their relationships
- Integration with existing systems
- Expected end user deliverables characterized: apps, reports, dashboards, processes, etc

Data Sources Identified

- Rarely: bunch of CSV files
- Access via APIs (multiple sources)
- Pieces already existing in the Data Warehouse
- Need to be extracted from existing systems

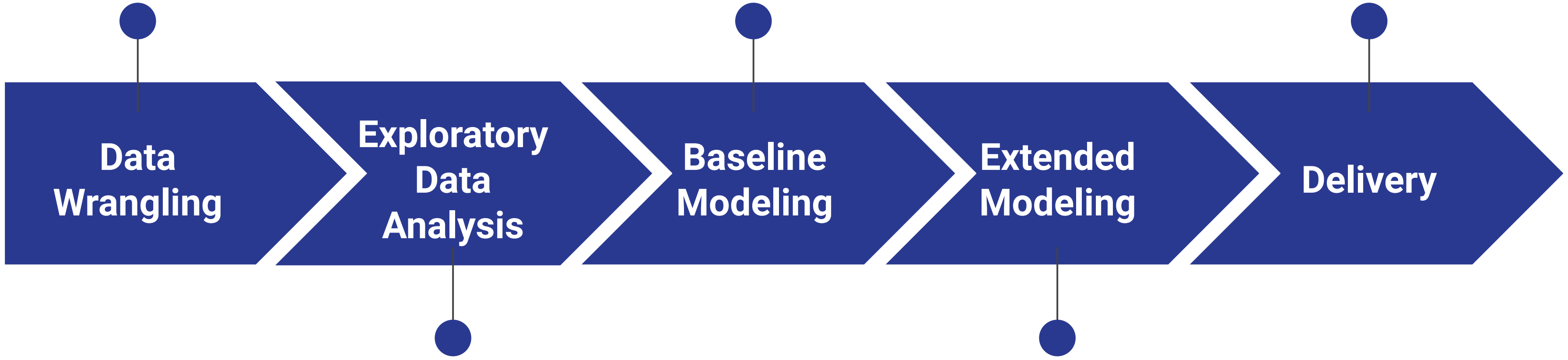
Caveats

Other scenarios: *typically fast-paced and "fractal" (chaotic)*

- "Green" Projects
- Minimally Viable Products (MVP)
- Internal Projects
- Trial Projects/Proof of Concept
- Many more …

Classical Data Science Process*

* a.k.a. “Data Science Method”, and akin to “Waterfall” Software Development Process



Acquire and prepare the data for further analysis

Build the simplest models that produce useful results

Interpretability, Causality, Conformity, Observability, End-User Delivery Development, and Integration

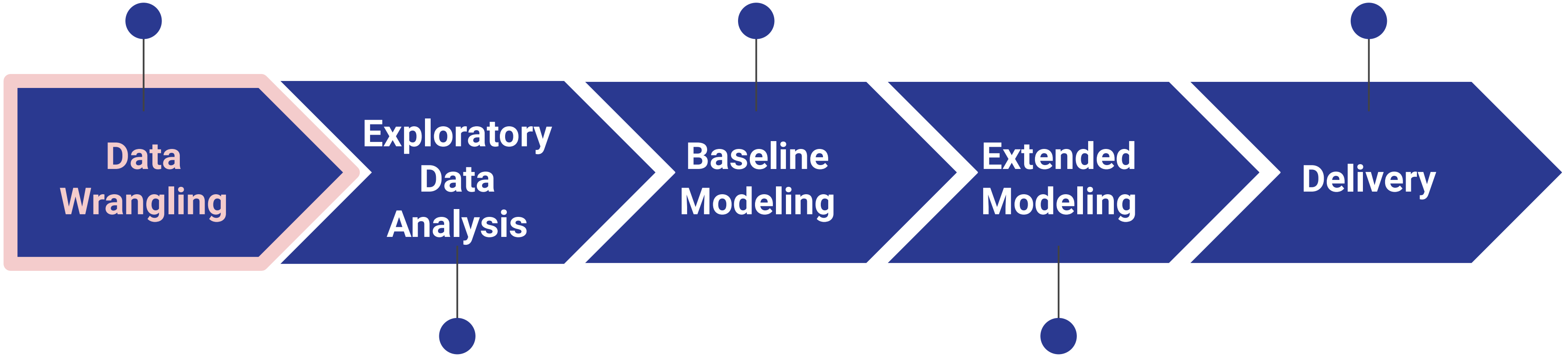
Become more familiar with the data, problem, problem domain, and gather information for downstream phases

Iterative development, ranking, and combination of models, hopefully via Auto-ML

Acquire and prepare the data for further analysis

Build the simplest models that produce useful results

Interpretability, Causality, Conformity, Observability, End-User Delivery Development, and Integration



Become more familiar with the data, problem, problem domain, and gather information for downstream phases

Iterative development, ranking, and combination of models, hopefully via Auto-ML

Caveats

*I did not consider **scalability** in the previous process!*

- The process mentioned before can be used at a smaller scale to identify the models to be later implemented at a larger scale
- The scaling up–a **major undertake**–can be placed before "Delivery" in the previous pipeline
- Also, in spite of the linear illustration, the process is cyclical

Important Principle

All Data Science projects are Software Development projects, but some Software Development projects are NOT Data Science Projects!

Implications of this Principle

*That have a direct implication in the tools and skills needed. Also **Cloud-Based** is an **umbrella requirement!***

- Some **tools** in order of importance (non-exhaustive list):
  - SQL, No-SQL, and other data stores (e.g., Graph, and "Vector" DBs)
  - API access
  - Python + SQL + Pandas
  - Viz tools (e.g,. Tableau, Pandas Profiling)
  - Large and Distributed data stores
- Some **software development skills** (non-exhaustive list):
  - Modularity (e.g., classes, functions, pipelines, frameworks, config files, etc)
  - Reusability, and Efficiency
  - Parallel and Distributed Processing

<u>Minimal</u> Data Wrangling

*"Acquire and prepare the data **<u>for further analyses</u>**"*–In Other Words …
*"**<u>Quickly</u>** get the data **<u>ready for wider and deeper analyses</u>**"*

Get to EDA <u>ASAP</u>

… but not too fast!
*(Emphasis here is on **<u>tabular data</u>**)*

- Repeat **for all data sources**:
  - Get data source
  - Row + Column count
  - Type + Semantics (when possible) per column
  - Categorical, non Categorical characterization
  - Percent count of missing values, and gaps per column (e.g., time series)
  - Data quality checks per column (needs information about semantics of column)
  - Identify duplicate rows
- Gather and stage prepared data sources on data infrastructure to use downstream

Common Data Wrangling Pitfalls

Data Wrangling Golden Rule: ***ALWAYS*** *consult with the client before making decisions about what to do with the data (when in doubt), and about data intended semantics!*

Avoid …

List is not exhaustive

- Using the wrong tool for the job:
  - E.g., complex joins with Pandas. Use Python+SQL instead!
- Making decisions too early:
  - Deleting data (beware!), type transformations inside of DataFrame
- Tasks that do not belong:
  - EDA, Feature Engineering, “Outlier Detection and Elimination”, Modeling
- Confussions:
  - Categorical vs Non-Categorical
  - What to do with duplicates
  - What to do with “outliers”
- Only relying on “eyeballing”
  - Check invariants instead
- Making decisions w/o consulting with the client
  - Avoid “I-feel” decisions

First Exercise: X minutes (TBD)

Do just what is needed to “Acquire and prepare the data ***for further analysis****” for the dataset I shared with you. If you started to work on the wrangling of this dataset, compare your approach with the minimal approach discussed here, and consider sharing your thoughts and work with the audience*

First Exercise: Discussion

Do we have one volunteer to show their work?

About Types

- There is a difference between Python Types and Pandas dtypes. The attached notebook discusses this in more detail
- Many Python container-like data structures (e.g,. Lists, Dictionaries, DataFrame, etc.) are able to store values of different types. During Data Wrangling it is important to investigate this and appropriately act on the findings

About Categorical VS Non Categorical [1]

- **Categorical:** values **<u>represent classes</u>**–i.e., they are labels of classes. For instance: MALE, FEMALE

- **Non Categorical:** values **<u>represent a measurement of something</u>**. For instance: the temperature of a room

- **Confusion comes from trying to infer whether a value is categorical or not based only the type of the value**

About Categorical VS Non Categorical [2]

- Typically people think that a numeric value must always be interpreted as a measurement of something; and therefore conclude that numeric values imply non categorical, and that non numeric values imply categorical
- This reasoning is incorrect since what matters is what the value represents, not its type
- Thus numeric values can represent measurements OR categories; and non numeric values can also represent measurements OR categories

About Categorical VS Non Categorical [3]

- In conclusion, ask
  - Do values represent classes? YES →CATEGORICAL
  - Do values represent measurements? YES → NON CATEGORICAL
- DO NOT base your analysis ONLY on the (Python) TYPE of the values, but on what THEY REPRESENT

About Categorical VS Non Categorical [4]

- CAVEATS: there are cases where the values can represent EITHER a measurement OR a class
- Consider reviews and the number of stars associated with it, say 1-5 stars
- We can consider that number as the label of a class (the class of reviews with one star, two stars, etc.)
- We can ALSO consider that number as a percent with respect to the maximum number of stars, and thus measuring how close they are to 5 (100%, 80%, 60%, etc.)

About Duplicates

- <u>**ONE NEEDS TO FIRST DEFINE WHAT CONSTITUTE DUPLICATES**</u>
- **Some examples:**
  - **Component-wise duplicates**
  - **Duplicates that must be there by design (e.g., bootstrapping, patients with identical health profiles, etc.)**
- **When dealing with classification problems one must make sure that valid duplicates do not fall in the training set and test set simultaneously. So they either go together to the training set or to the test set, but not both**

About “Outliers”

- It is OK to deal with outliers in the context of conducting semantic checks. For instance: “age must be less than 90”
- It is also OK to identify statistical outliers and pass this information to EDA–meaning: the nature of these outliers is investigated then
- It is NOT OK to–for instance–mechanically “*identify and delete outliers*” during Data Wrangling. Thus, I personally prefer to deal with outliers during EDA

About Semantic Checking

- To perform semantic checking one must first have semantic rules. For instance: phone numbers, zip codes, and social security numbers must follow certain format. Zip codes must be valid, etc
- These rules must come from and validated by the client, as well as what to do with data that are not valid with respect to these rules

Second Exercise: X minutes (TBD)

How would you wrangle the columns in the file to be provided to you, assuming that they are part of a larger dataset? Be ready to share your code and/or ideas to code the wrangling process

Second Exercise: Discussion

Do we have one volunteer to show their work?

About Semantic Checking [1]

- From our “Minimal Data Wrangling” checklist, perhaps the point that requires more attention is semantic checking. I mention some examples below
- WIth respect to Specialties, Industries, and Ownership there might be standards to be followed
- With respect to Description, one might check if the language is English

About Semantic Checking [2]

- Data Wrangling would NOT be the right place to apply NLP-like transformations such as Dummies, Vectorization, and even Embeddings
- Depending on the goal of the application, these interventions can be spread through EDA, Feature Extraction in Baseline Modeling, and Extended Modeling

Third Exercise: X minutes (TBD)

How would you code a Python class that you can use to implement GET API calls in a wide variety of projects that require the use of REST APIs to acquire datasets?

Third Exercise: Discussion

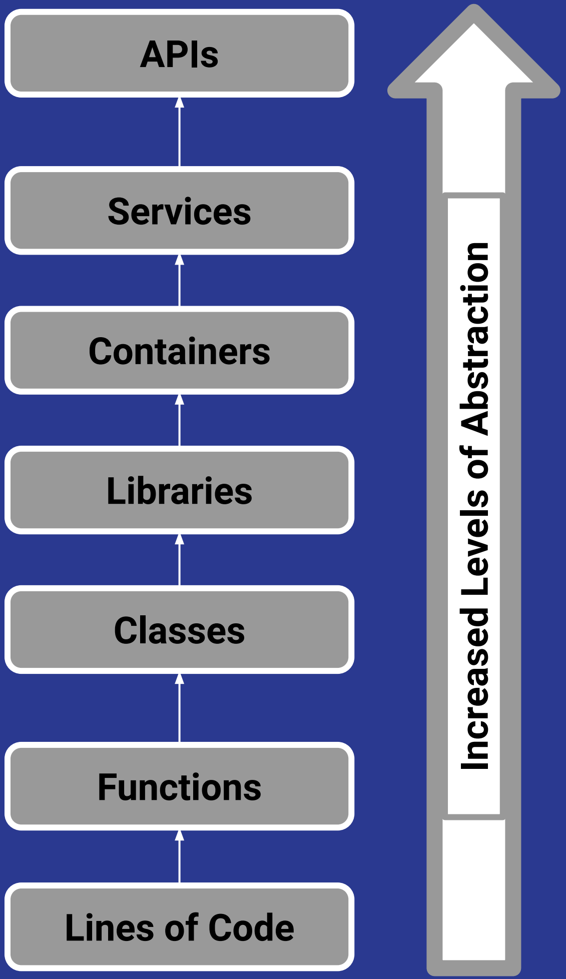
Do we have one volunteer to show their work?

Keep in mind ...

- The code associated with this Exercise in the notebook that I am sharing with you should be used as a template that will hopefully guide you when building your own classes
- For instance, try to create your own class to handle “GET” API calls in connection with the “API Mini-Project” in the curriculum

One Fundamental Software Engineering Goal

Cross-Cutting: Modularization, Reusability, and Efficiency



Thank you for Attending!

Additional Thanks and Credit to:

- **Rod Aronas**–Director of Course Talent Operations
- **Dhara Damiana**–Operation Manager
- **All my mentees over the years!**
- Background image on the title slide by Mika Baumeister–Unsplash