



# Module 4: Machine Learning

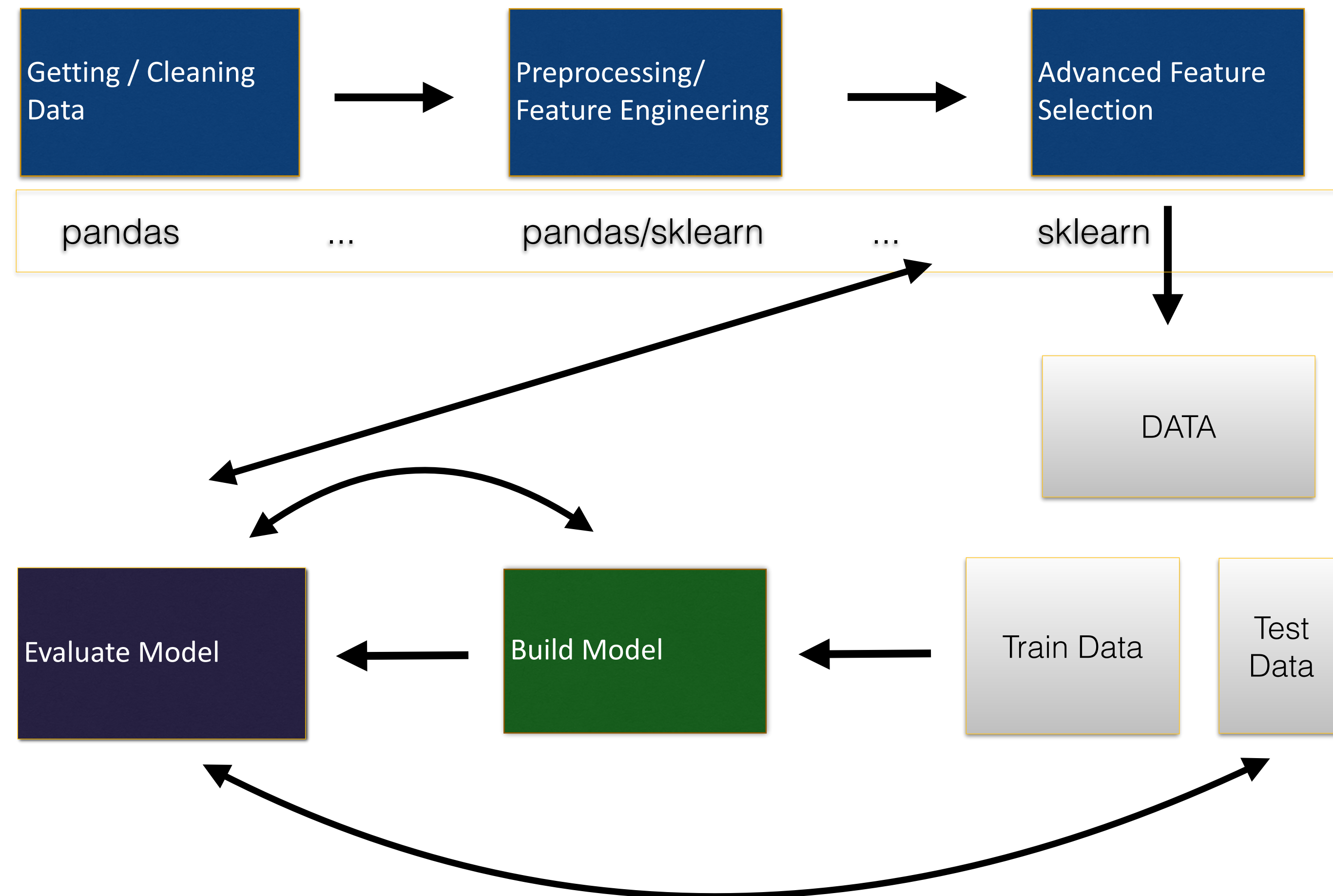
## Part 1

### Feature Engineering

# Agenda

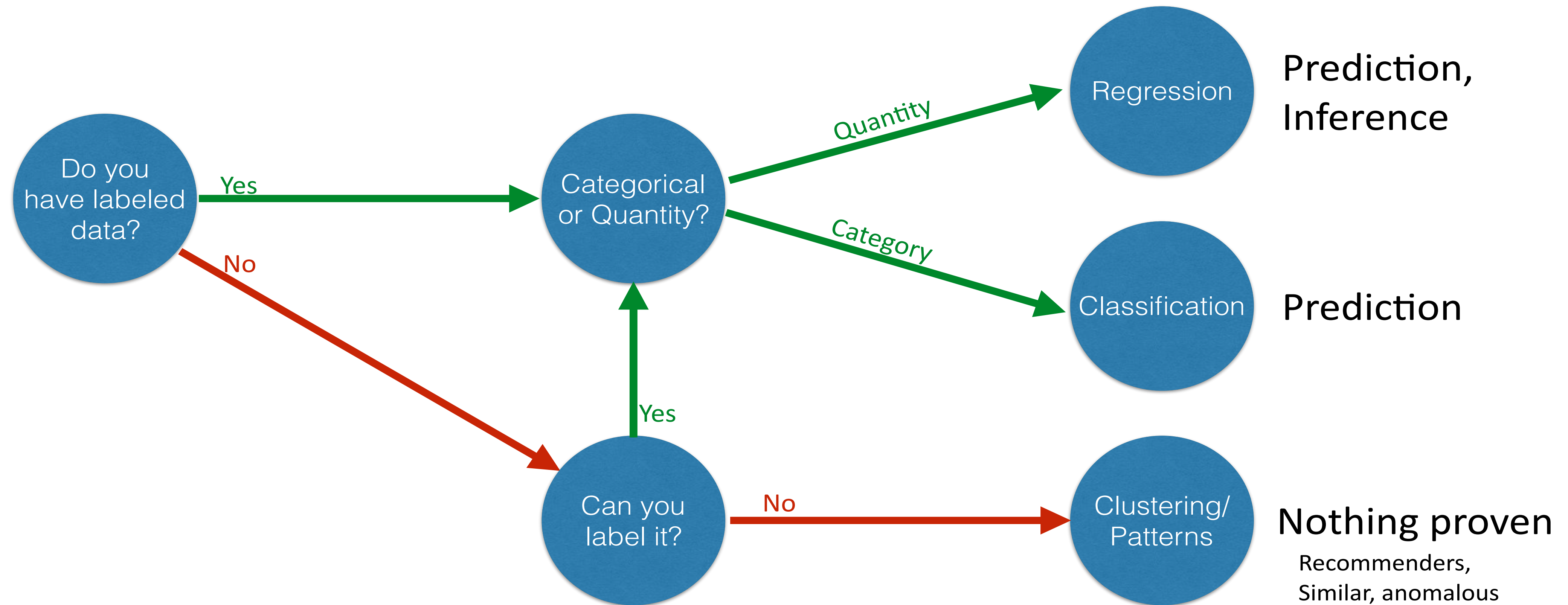
- Feature Selection & Engineering
- Math free overview of classification models
- Evaluating Model Performance
- Improving model performance

# Machine Learning Process



# Machine Learning Terms

- **Features:** The mathematical representation of the original data. The features are the columns in your data set. Since the features will be a matrix, they are often written as  $X$ .
- **Observations:** The rows of your feature set.
- **Target:** The variable that you are trying to predict. Often represented as  $y$ .



# Features

<http://www.google.com>



# Features

<http://www.google.com>



domain_length	vowel_count	digit_count
6	3	0



# Representation of URL Knowledge

- Come up with a **representation/set of knowledge** that has **enough complexity** to accurately describe the problem for the computer
- Knowledge here does not mean hard-coded knowledge or formal set of rules
- **The computer rather uses the knowledge we provide to extract patterns and acquire own knowledge**
- We should provide knowledge about reality that has **high variance about the problem** it describes (e.g. a feature that is high when it rains and low when it's sunshine)



[https://www.google.com/search?  
q=URL&source=Inms&tbm=isch&sa=X&ved=0ahUKEwjcl6ut-  
IDUAhVEPCYKHdJGDsYQ\\_AUIDCgD&biw=1215&bih=652](https://www.google.com/search?q=URL&source=Inms&tbm=isch&sa=X&ved=0ahUKEwjcl6ut-IDUAhVEPCYKHdJGDsYQ_AUIDCgD&biw=1215&bih=652)

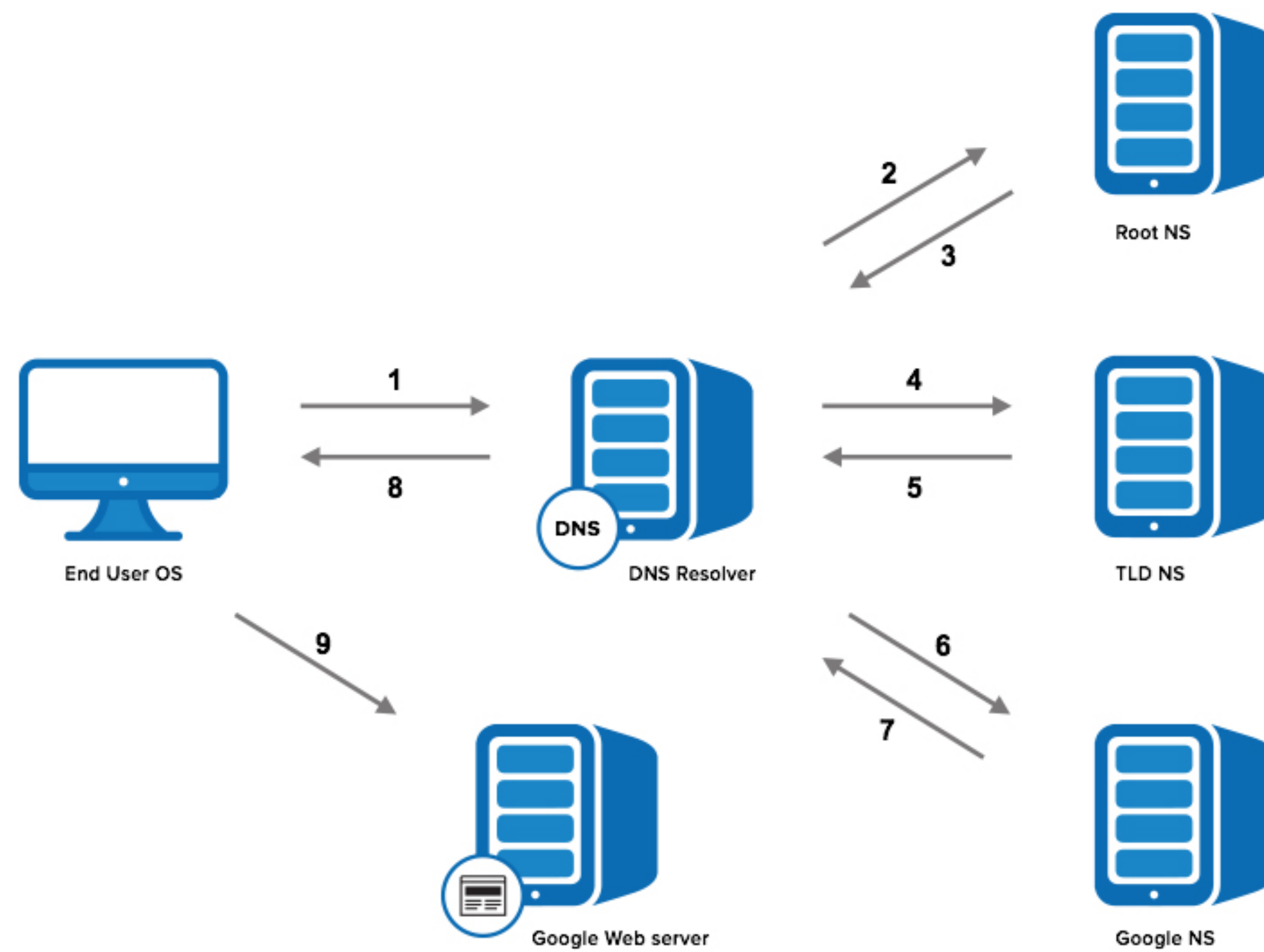
https://	protocol
www	subdomain
google.com	zone apex
google	domain
.com	top-level-domain (tld)
/search?q=URL...	path

# DNS 101

- **Domain Name Service** (DNS) resolves domain names to IP addresses (like a phone book)
- **Domain Registrars**: authority that signs unique domain names (GoDaddy, BlueGtaor)
- **State of Authority** (SOA): Contains for example name of server for zone, administrator of zone, default time-to-live (ttl = time a DNS record is cached), seconds of secondary name server should wait before checking for updates
- **Root Zone** controlled by Internet Assigned Numbers Authority (IANA)
- **Name Servers** (NS Records): used by tld servers to direct traffic to DNS server (which contains authoritative DNS records)
- **A records** (part of DNS record): “A” stands for IP Address
- **CNAME** (part of DNS record): resolves one domain name to another
- **Autonomous System** (AS) and Border gateway Protocol (BGP) info

Python libraries: `python-whois`, `dnspython`, `tlsextract`, `ipaddress`

# DNS Flow



# What makes them different?

## URL BlockList

[amazon-sicherheit.kunden-ueberpruefung.xyz](https://amazon-sicherheit.kunden-ueberpruefung.xyz)

[eclipsehotels.com/language/en-GB/eng.exe](https://eclipsehotels.com/language/en-GB/eng.exe)

[bohicacapital.com/page](https://bohicacapital.com/page)

[summerweb.net](https://summerweb.net)

[ad.getfond.info](https://ad.getfond.info)

[vdula.czystykod.pl/rxdjna2.html](https://vdula.czystykod.pl/rxdjna2.html)

[svision-online.de/mgfi/administrator/  
components/com\\_babackup/classes/fx29id1.txt](https://svision-online.de/mgfi/administrator/components/com_babackup/classes/fx29id1.txt)

## URL AllowList

[gurufocus.com/stock/PNC](https://gurufocus.com/stock/PNC)

[dvdtalk.ru/review](https://dvdtalk.ru/review)

[333cn.com/zx/zhxw.html](https://333cn.com/zx/zhxw.html)

[made-in-china.com/special/led-lighting](https://made-in-china.com/special/led-lighting)

[google.com/u/0/112261544981697332354/posts](https://google.com/u/0/112261544981697332354/posts)

[youtube.com/watch?v=Qp8MQ4shN6U](https://youtube.com/watch?v=Qp8MQ4shN6U)

[unesco.org/themes/education-sustainable-developm](https://unesco.org/themes/education-sustainable-developm)

[thisisthefirst.com/page/5](https://thisisthefirst.com/page/5)

# Malicious URL Detection Features (Literature)

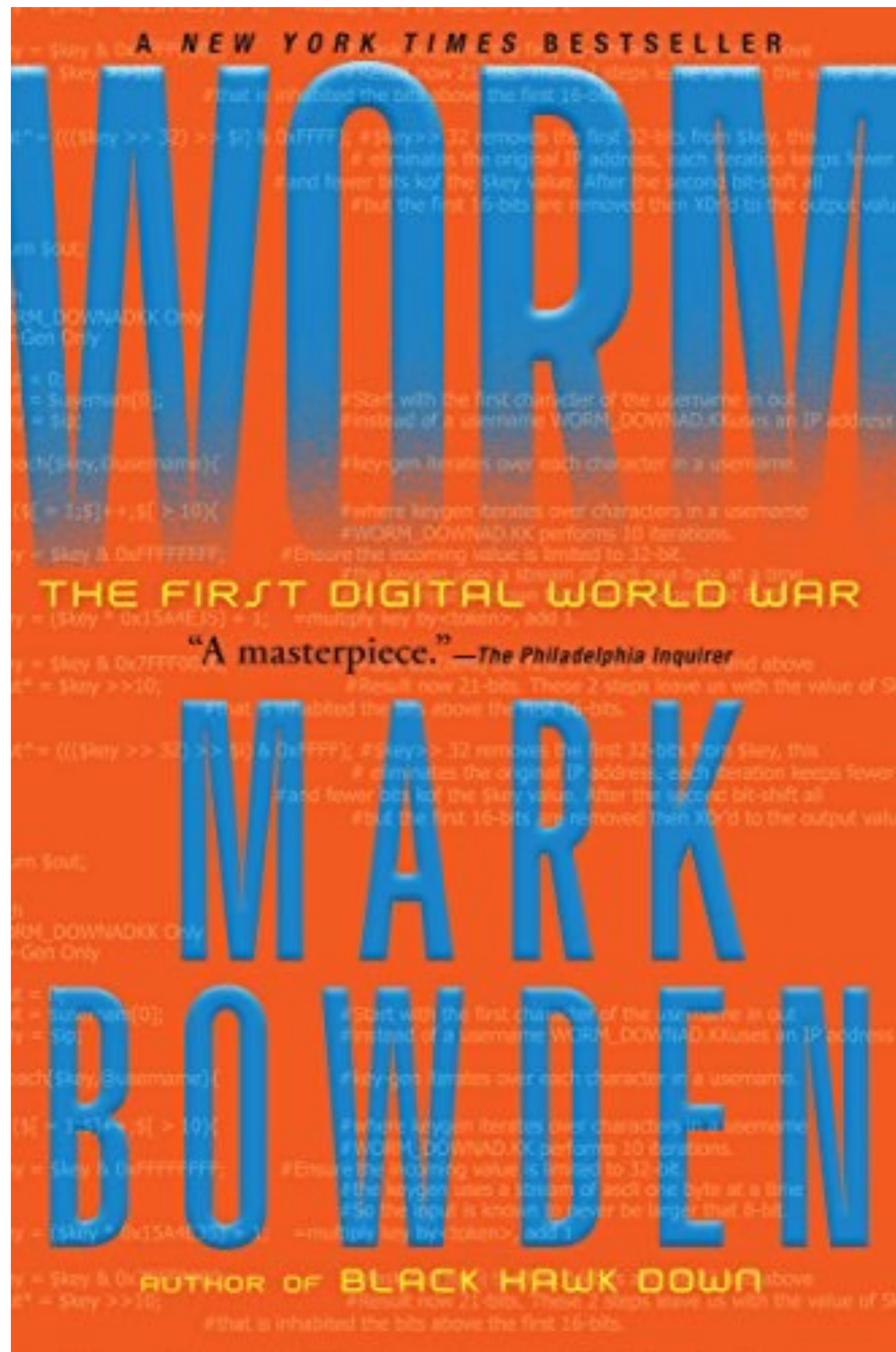
1. **BlackList Features:** BlackLists suffer from a high false negative rate, but can still be useful as machine learning feature.
2. **Lexical Features:** Capture the property that malicious URLs tend to "look different" from benign URLs. **Contextual information** such as the length of the URL, number of digits, lengths of different parts, entropy of domain name.
3. **Host-based Features:** Properties of web site host. **"Where"** the site is hosted, **"who" owns it** and **"how" it is managed**. API queries are needed (WHOIS, DNS records). Examples: Date of registration, the geolocations, autonomous system (AS) number, connection speed or time-to-live (TTL).
4. **Content-based Features:** Less commonly used feature as it requires **execution of web-page**. Can be not only be not safe, but also increases the computational cost. Examples: HTML or JavaScript based.

# Domain Generating Algorithm (DGA) Detection

# What is DGA?

- Domain Generating Algorithms (DGA) are algorithms which generate pseudo-random domains that are used for infected machines to communicate with the controller.
- The seeds on both the victim and command server are synchronized
- First seen with the Conficker worm.
- "Hello world" of cyber machine learning.





# Preparation In Class Exercise

## ML Feature Engineering

### Lexical Features

1. Length of URL
2. Length of domain
3. Count of digits
4. Entropy of domain
5. Position (or index) of the first digit
6. **Bag-of-words** for tld, domain and path parts of the URL

### Host-based Features

1. Time delta between today's date and creation date
2. Check if it is an IP address

# Data Set (Features and Target)

	url	isMalicious	domain	created
56675	jeita.biz/w/google/drive/document.html?ssl=yes	1	jeita.biz	2012-04-11 17:08:19
73229	sosnovskoe.info/layouts/plugins/mailbox	1	sosnovskoe.info	2011-09-19 09:53:07
60112	teothemes.com/html/mp3pl/blue-preview.jpg	1	teothemes.com	2011-09-08 21:43:00
66946	kfj.cc:162/17852q	1	kfj.cc	2013-08-18 05:52:47
81906	verapdpf.info/db/6d1b281b5c4bbcf3b99228680c232fa	1	verapdpf.info	2016-08-18 07:09:03

Features or X

	isMalicious	isIP	Length	LengthDomain	DigitsCount	EntropyDomain	FirstDigitIndex	com	org	net	...	w	waset
73320	1	0	27	21	0	3.558519	0	0	1	0	...	0	0
30785	0	0	77	11	14	3.095795	22	1	0	0	...	0	0
60789	1	0	141	11	5	3.459432	103	0	1	0	...	0	0
19495	0	0	59	13	20	3.546594	31	1	0	0	...	0	0
45022	1	0	23	11	7	3.277613	13	0	0	0	...	0	0

Target or y



# Bag-of-Words

- Bag-of-words model: (Frequency of) occurrence of each word is used as a feature
- Sklearn's `CountVectorizer`: Convert a collection of text documents to a matrix of token counts

Simple Example: Imposing a vocabulary of `top_tlds=[ '.com', '.de', '.uk' ]`

```
CountVectorizer_tlds = CountVectorizer(analyzer='word', vocabulary=top_tlds)
CountVectorizer_tlds = CountVectorizer_tlds.fit(tlds)
matrix_tlds = CountVectorizer_tlds.transform(tlds)
```

URL string
...google.ru...
...facebook.com...
...google.de...

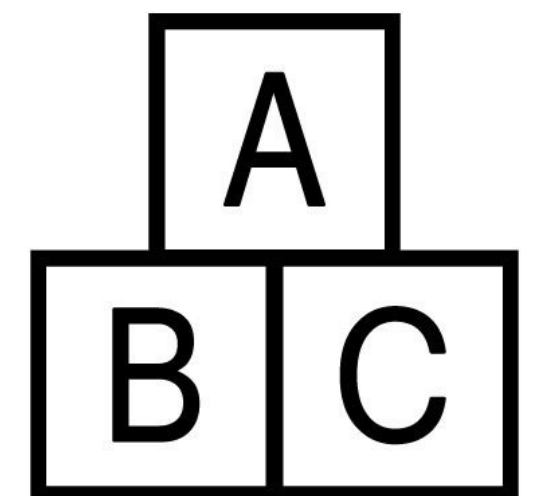
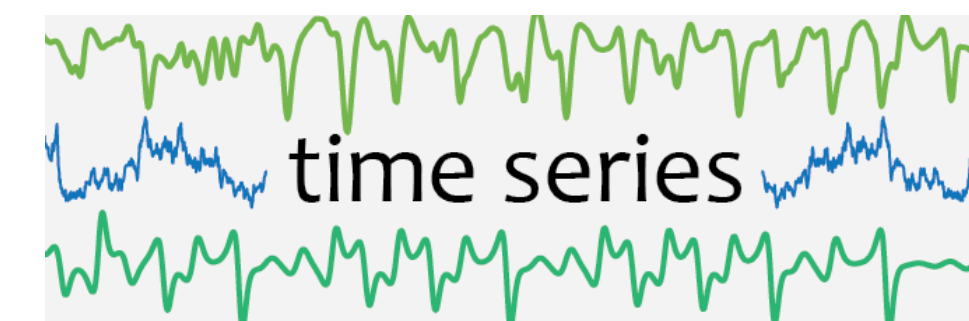
Bag-of-words model fitting

'.com'	'.de'	'.uk'
0	0	0
1	0	0
0	1	0

# Preprocessing - an Art Work!

- Imputing missing values
- Scaling/Normalization
- One-Hot Encoding (Encoding categorical features)
- Embedding (e.g. word2vec)
- Binarizing (e.g. needed for Deep Learning multi-class target vector encoding)
- Encoding strings as int
- Dimensionality Reduction (e.g. PCA)
- Augmentation (e.g. tild/zoom images)
- Feature selection based on classifier
- Variance threshold

Data Types



01

# Imputing Missing Values

```
# using the most_frequent value  
df['src_bytes'] = df['src_bytes'].fillna  
(df['src_bytes'].value_counts().index[0])
```

```
# using the mean value  
df['dst_bytes'] = df['dst_bytes'].fillna(df['dst_bytes'].mean())
```

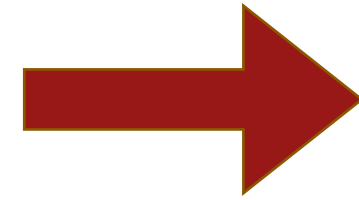
# One Hot Encoding

Color
Red
Red
Blue
Green
Yellow
Red



# One Hot Encoding

4 Categories



4 Columns with 1 when Category is True and delete original column!

Color
Red
Red
Blue
Green
Yellow
Red

Color_Red	Color_Blue	Color_Yellow	Color_Green
1	0	0	0
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
1	0	0	0

# One Hot Encoding

```
colors = [ 'Red', 'Red', 'Blue', 'Green', 'Yellow', 'Red' ]  
series_data = pd.Series( colors )  
pd.get_dummies( series_data )  
  
# df scenario  
df=pd.get_dummies(df, prefix=None, prefix_sep='_',  
dummy_na=False, columns=[ 'protocol_type', 'flag' ],  
sparse=False)
```

# Better way: Feature Engine

- Feature Engine is a module which helps automate the complexities of feature engineering.
  - Missing Value Imputation
  - One Hot Encoding
  - Outlier Capping
  - More...
- Docs here: <https://feature-engine.readthedocs.io/en/latest/>
- Blog post: <https://thedataist.com/when-categorical-data-goes-wrong/>

# Feature Engine

```
from feature_engine import categorical_encoders as ce
import pandas as pd

# set up the encoder
encoder = ce.OneHotCategoricalEncoder(
    top_categories=3,
    drop_last=False)

# fit the encoder
encoder.fit(df)
encoder.transform(df)
```

# Encoding Strings as Integers

```
PROBE = ['portsweep.', 'satan.', 'nmap.', 'ipsweep.']  
df = df.replace(to_replace = PROBE, value=1)
```

# Feature Scaling

When you're creating a scaling object, you should first "fit" it to the **training data**, then **transform** both the **training and testing data** using the "fit" scaler.

If you try to fit the training and testing data separately, you will



# Standard Scaling

$$zscore(x_i) = \frac{x_i - \mu}{\sigma}$$

```
scaled_feature = (feature - column_mean) / standard_deviation
```

# Standard Scaling

$$zscore(x_i) = \frac{x_i - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
scaler.fit(features)
```

```
features_scaled = scaler.transform(features)
```

or

```
features_scaled = scaler.fit_transform(features)
```

# Standard Scaling

original values:

```
[[ 0.9  0.1 40. ]  
 [ 0.3  0.2 50. ]  
 [ 0.6  0.8 60. ]]
```

Mean of each column:

```
[ 0.6  0.3667 50.]
```

SD of each column:

```
[ 0.2449  0.3091 8.165 ]
```

scaled values:

```
[[ 1.2247 -0.8627 -1.2247]  
 [-1.2247 -0.5392  0.     ]  
 [ 0.      1.4018  1.2247]]
```

**Means of scaled data, per column:**

```
[ 0. -0.  0.]
```

**SD's of scaled data, per column:**

```
[ 1.  1.  1.]
```

Notice that the mean of standard scaled data is zero and the StdDev is 1.

# Min/Max Scaling

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
normed_feature = (feature - col_min) / (col_max - col_min)
```

# Min/Max Scaling

```
from sklearn.preprocessing import MinMaxScaler  
minmax = MinMaxScaler()
```

```
minmax.fit(features)  
features_scaled_minmax = minmax.transform(features)
```

or

```
features_scaled_minmax = minmax.fit_transform(features)
```

# Min/Max Scaling

original values:

```
[[ 0.9  0.1 40. ]  
 [ 0.3  0.2 50. ]  
 [ 0.6  0.8 60. ]]
```

Mean of each column:

```
[ 0.6  0.3667 50.]
```

SD of each column:

```
[ 0.2449  0.3091 8.165 ]
```

scaled values:

```
[[ 1.  0.  0. ]  
 [ 0.  0.1429 0.5 ]  
 [ 0.5  1.  1. ]]
```

Means of scaled data, per column:

```
[ 0.5  0.381 0.5 ]
```

SD's of scaled data, per column:

```
[ 0.4082  0.4416 0.4082]
```

# Dealing with Imbalanced Classes



# Dealing with Imbalanced Classes

- A lot of real-world security data will have very imbalanced targets.
- Fortunately, there is a library called imbalanced-learn which can assist.
- Imbalanced-learn provides a series of options to resample the data so that you have more balanced classes
- Docs available here: <http://imbalanced-learn.org/>

# Dealing with Imbalanced Classes

```
from imblearn.over_sampling import  
RandomOverSampler  
ros = RandomOverSampler(random_state=0)  
X_resampled, y_resampled = ros.fit_resample(X, y)
```

# Selecting Features

**Should we use all of them?**

**How do we know which features  
to use and which to discard?**

# Before using automation, you should remove:

- Unnecessary features that are uninformative or repetitive
- Irrelevant features
- Duplicate observations

# Selects k features according to the highest score

```
best_features = SelectKBest(score_func=chi2,k=3).fit_transform(features,target)
```

Selects all features above a given threshold in the scoring function

```
best_features =  
SelectPercentile(score_func=chi2,percentile=3).fit_transform(features,target)
```

Available Scoring Functions:

- **For regression:** `f_regression`, [`mutual\_info\_regression`](#)
- **For classification:** `chi2`, `f_classif`, `mutual_info_classif`

References:

[http://scikit-learn.org/stable/modules/generated/sklearn.feature\\_selection.SelectKBest.html](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html)

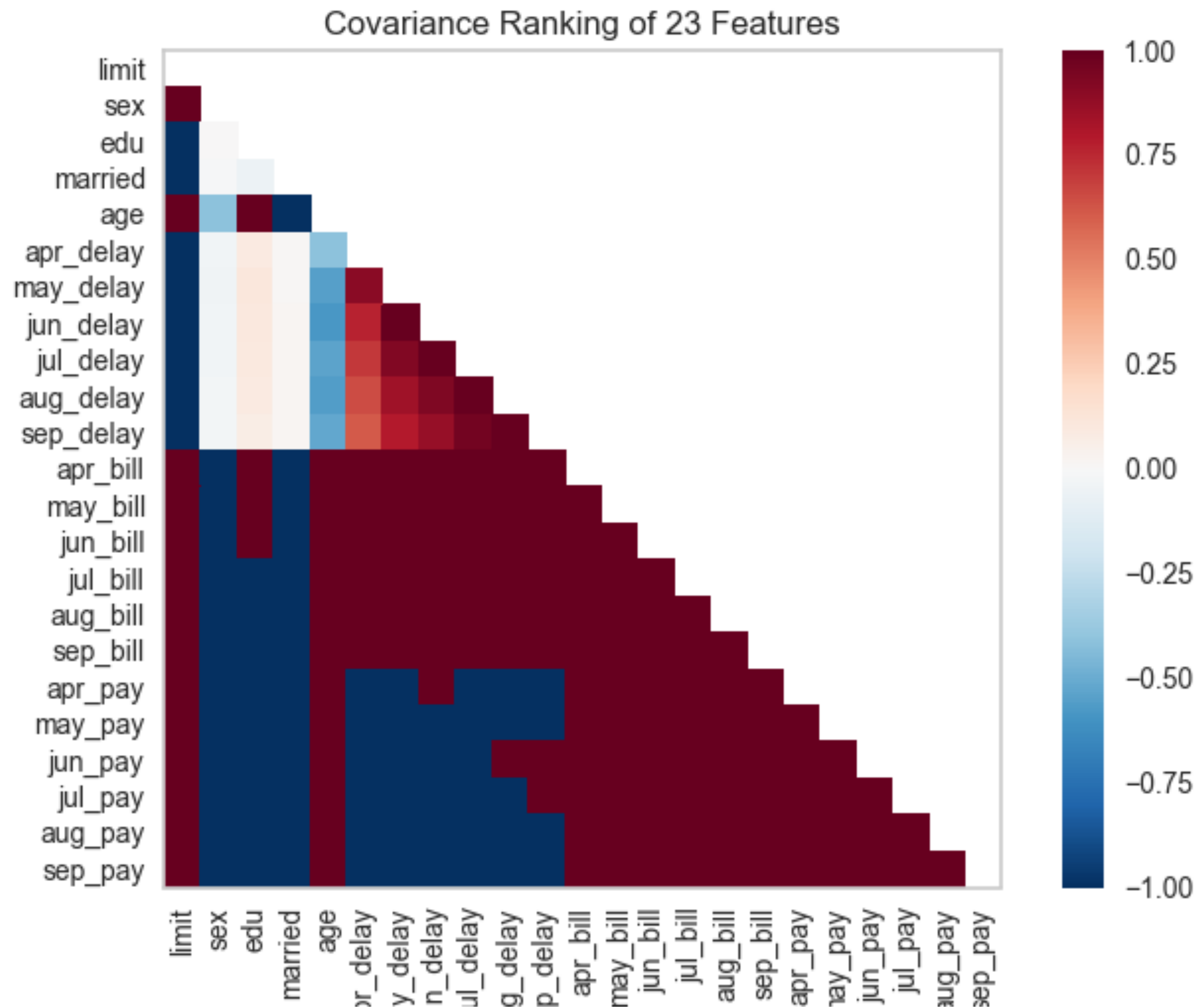
[http://scikit-learn.org/stable/modules/feature\\_selection.html#univariate-feature-selection](http://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection)

**How do we know which features  
to use and which to discard?**

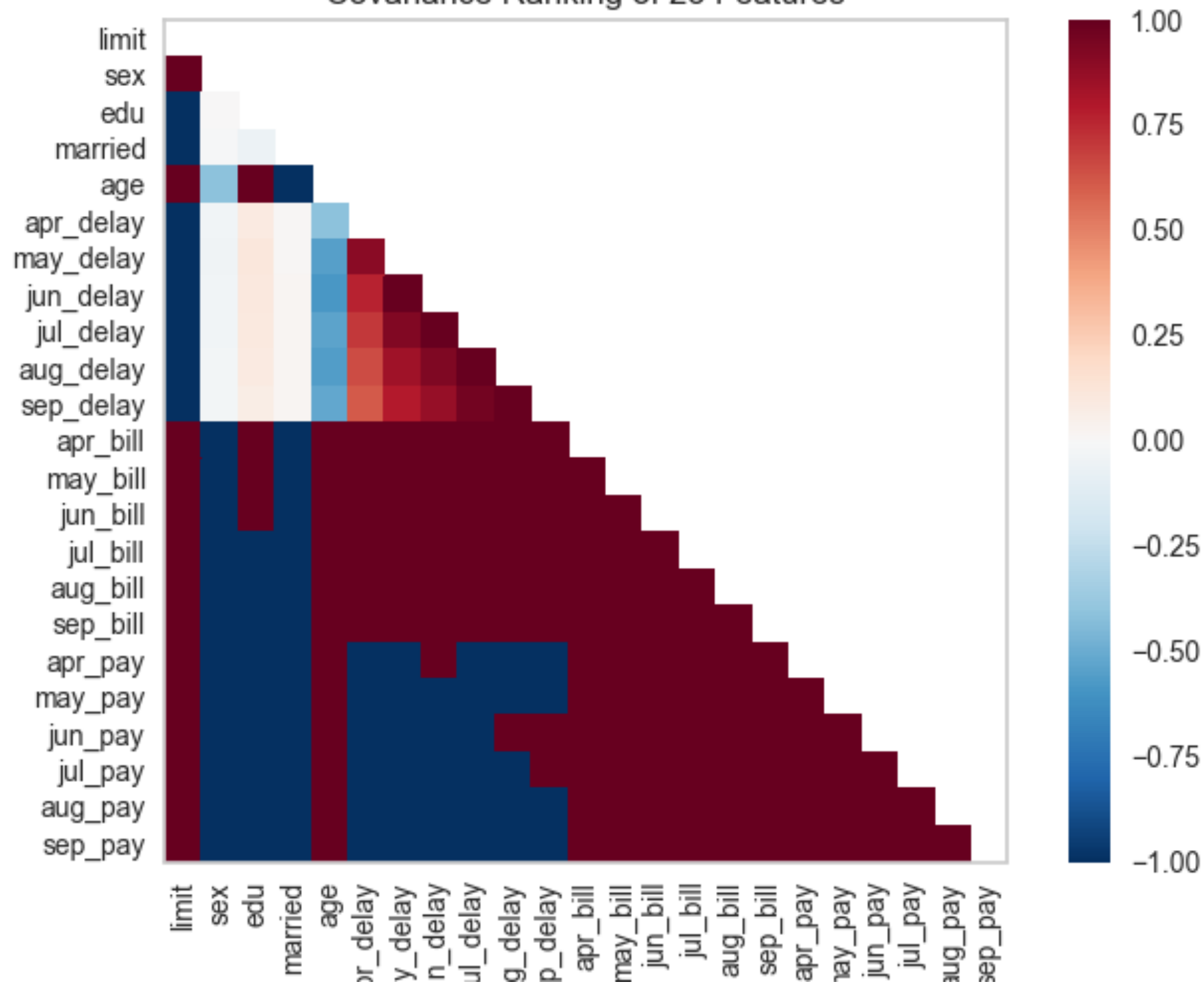
**Visualize Them!!**



# Introducing Yellowbrick and scikit-plot



Covariance Ranking of 23 Features

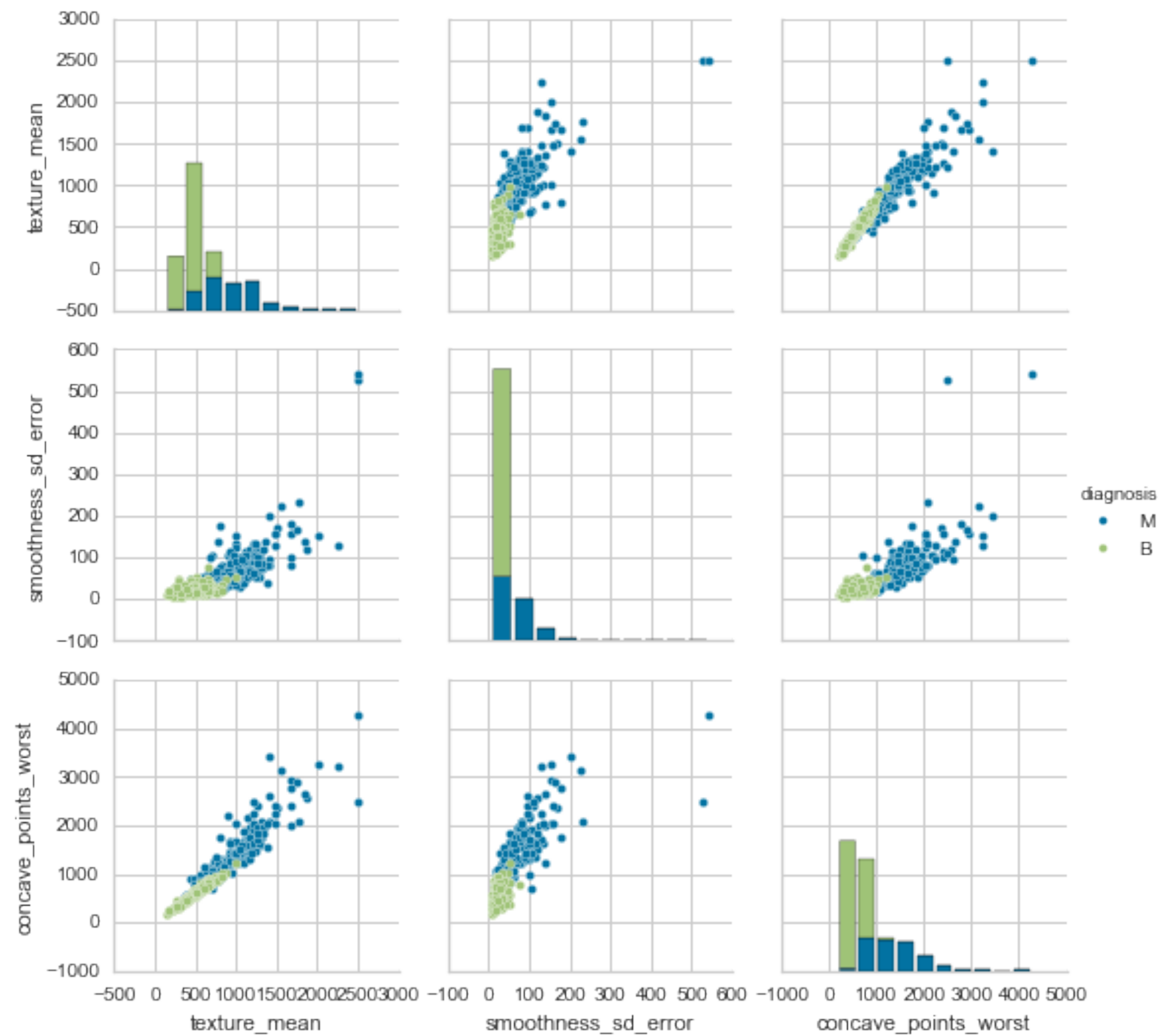


```
import pandas as pd
import seaborn as sns
from yellowbrick.features.rankd import Rank2D

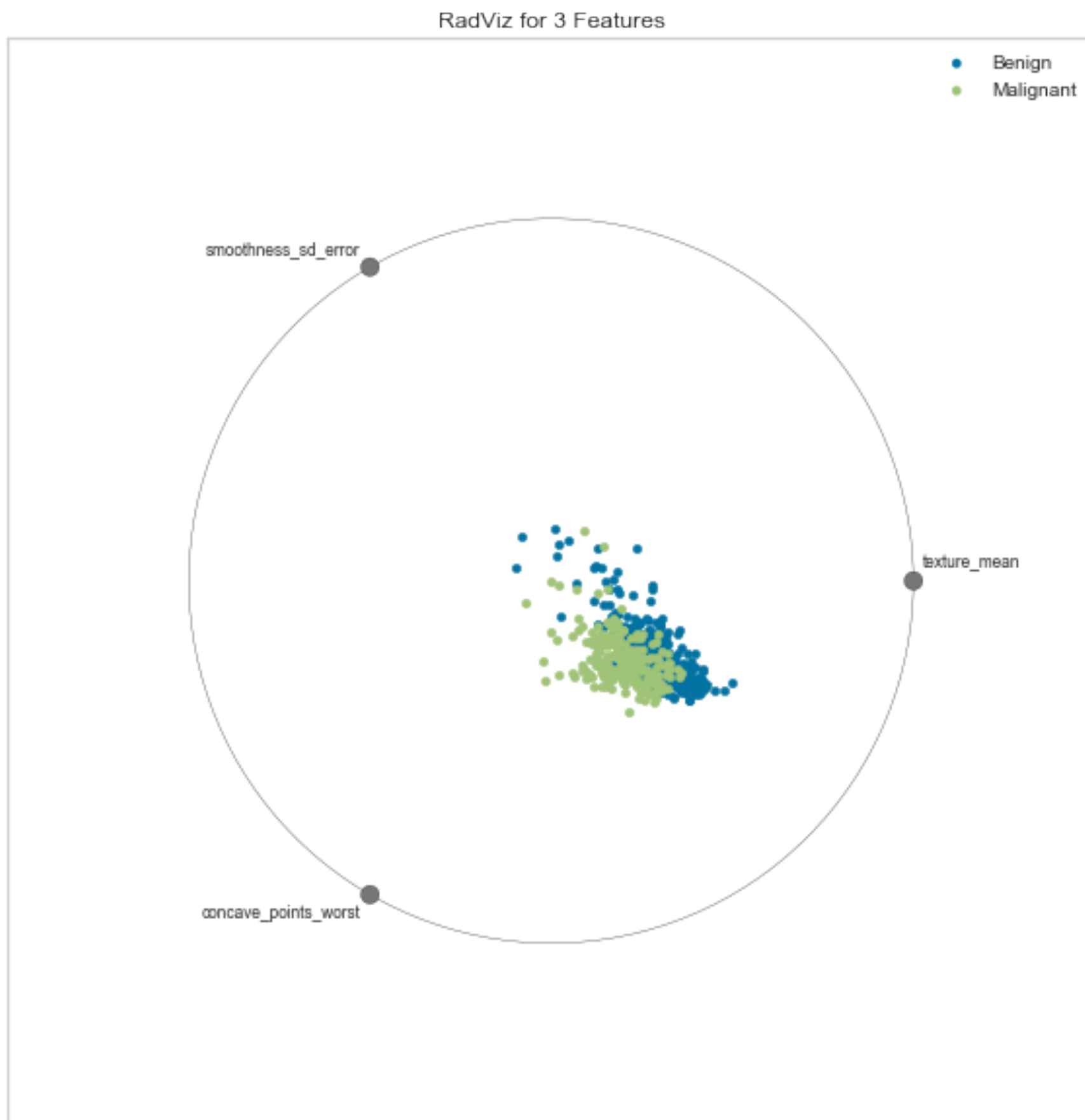
# Extract the numpy arrays from the data frame
features = df[features]
target = df['diagnosis']

# Instantiate the visualizer with the Covariance ranking algorithm
visualizer = Rank2D(features=features, algorithm='covariance')

visualizer.fit(features, target)
visualizer.transform(features)
visualizer.poof()
```



```
import seaborn as sns
sns.pairplot(<features>, hue='<target>' )
```



```
from yellowbrick.features.radviz import RadViz
...
visualizer = RadViz(classes=<target classes>, features = <features>)
visualizer.fit(features, target)
visualizer.transform(features)
visualizer.poof()
```

# Do it all at once!

- YData-Profiling (Formerly Pandas-Profiling) is a module which can generate very thorough summaries of your datasets.
- Simple to use, but computationally intense...

```
from ydata_profiling import ProfileReport
profile = ProfileReport(df, title='DGA Feature Profiling Report')
profile
```

# YData Profiling

## Overview

Overview

Reproduction

Warnings 4

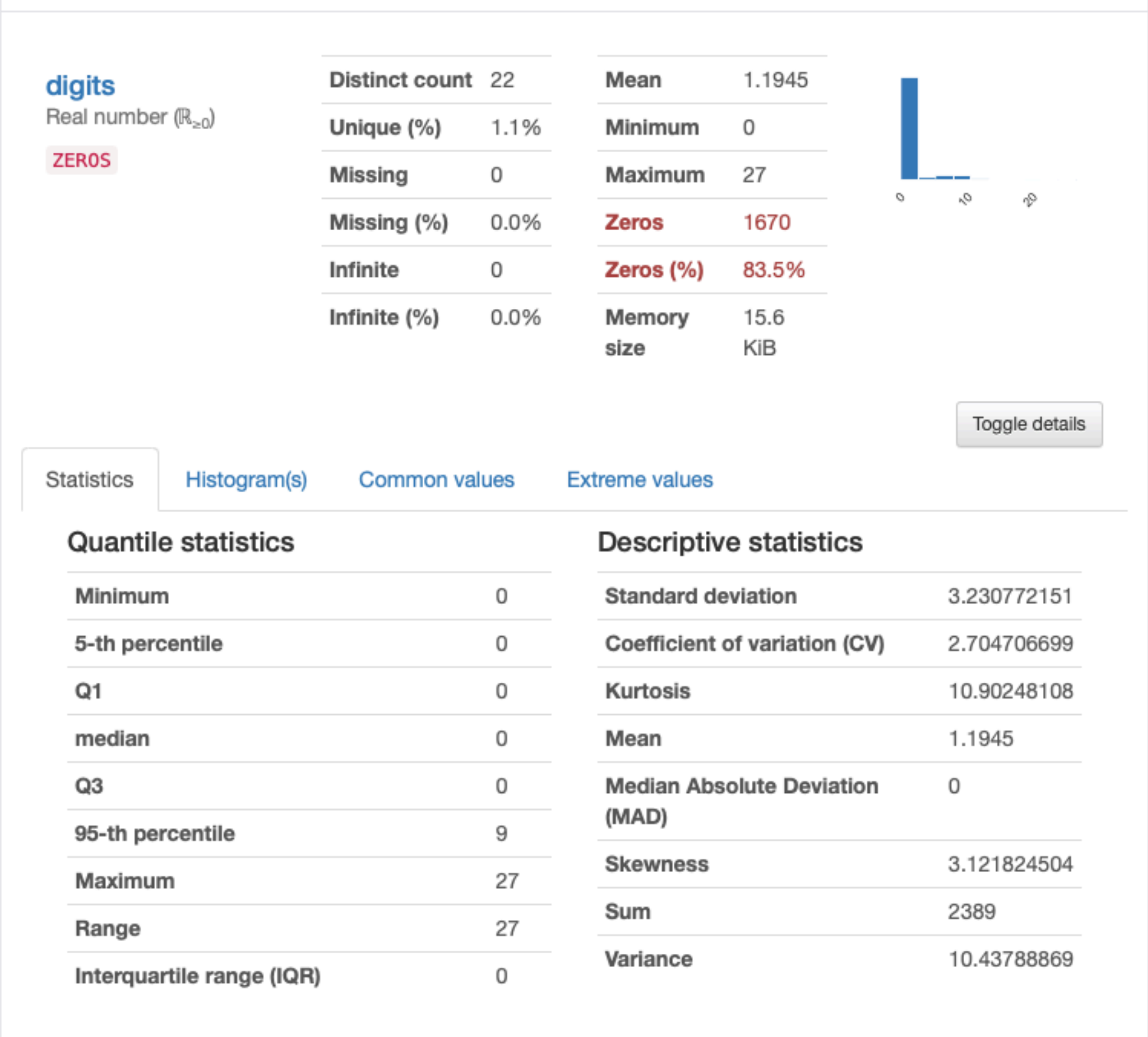
### Dataset statistics

Number of variables	7
Number of observations	2000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	2
Duplicate rows (%)	0.1%
Total size in memory	109.5 KiB
Average record size in memory	56.1 B

### Variable types

NUM	6
BOOL	1

# YData Profiling





# YData Profiling

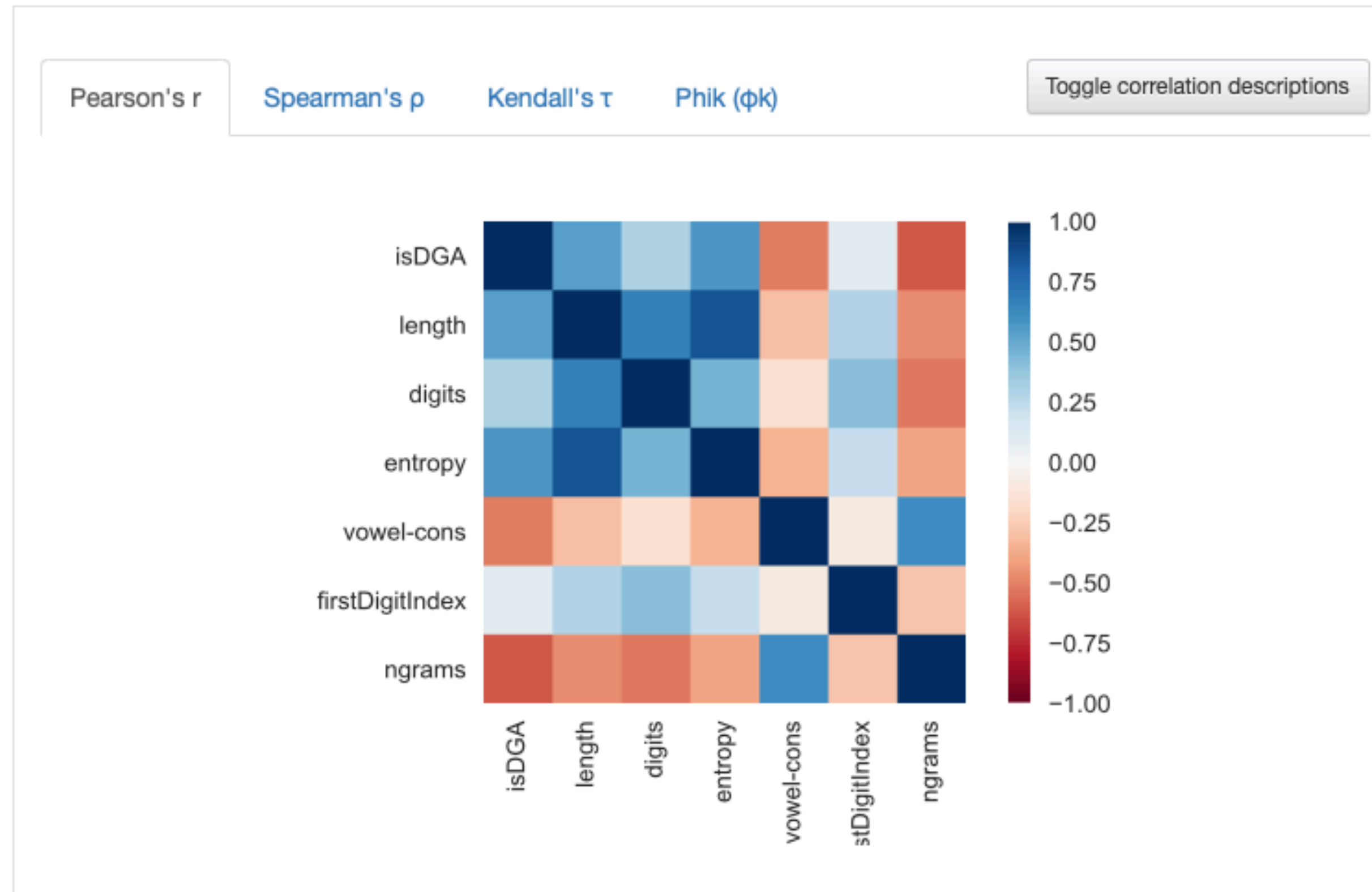
[Overview](#) [Reproduction](#) [Warnings 4](#)

## Warnings

Dataset has 2 (0.1%) duplicate rows	Duplicates
<code>digits</code> has 1670 (83.5%) zeros	Zeros
<code>vowel-cons</code> has 87 (4.3%) zeros	Zeros
<code>firstDigitIndex</code> has 1670 (83.5%) zeros	Zeros

# YData Profiling

## Correlations



**WHAT IF I TOLD YOU...**

**FEATURE ENGINEERING  
CAN BE AUTOMATED**





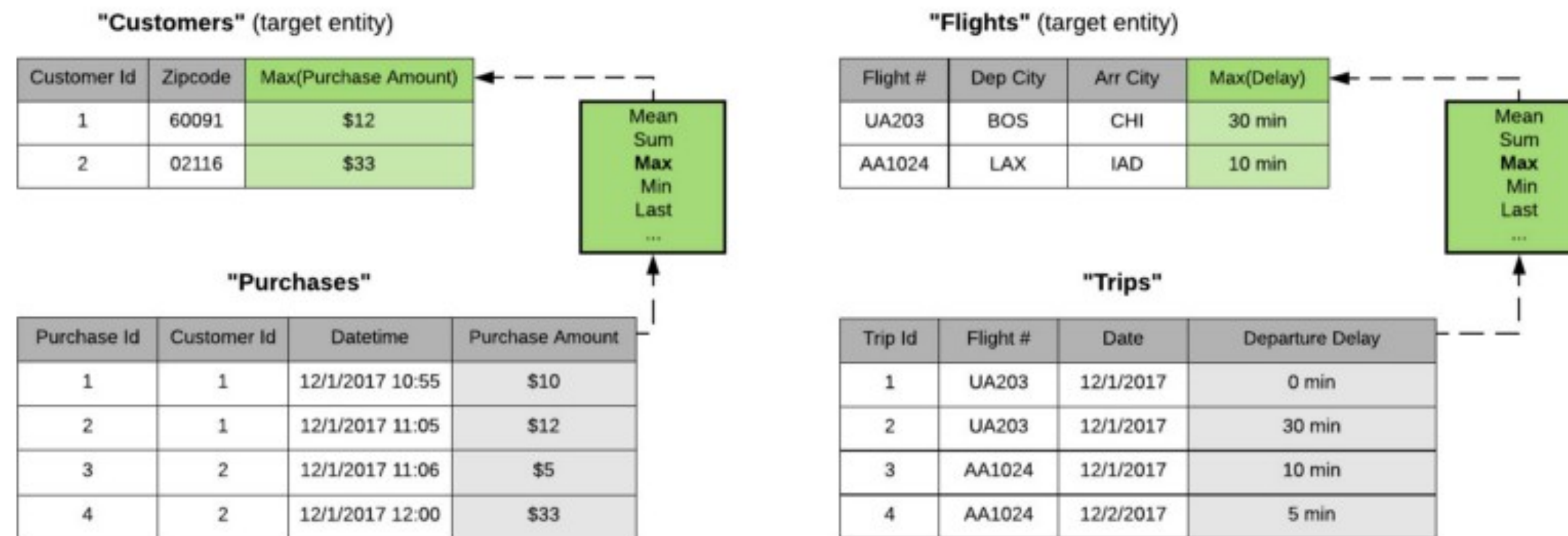
# Feature Tools

- Open Source
- Automated Feature Engineering



# Feature Tools

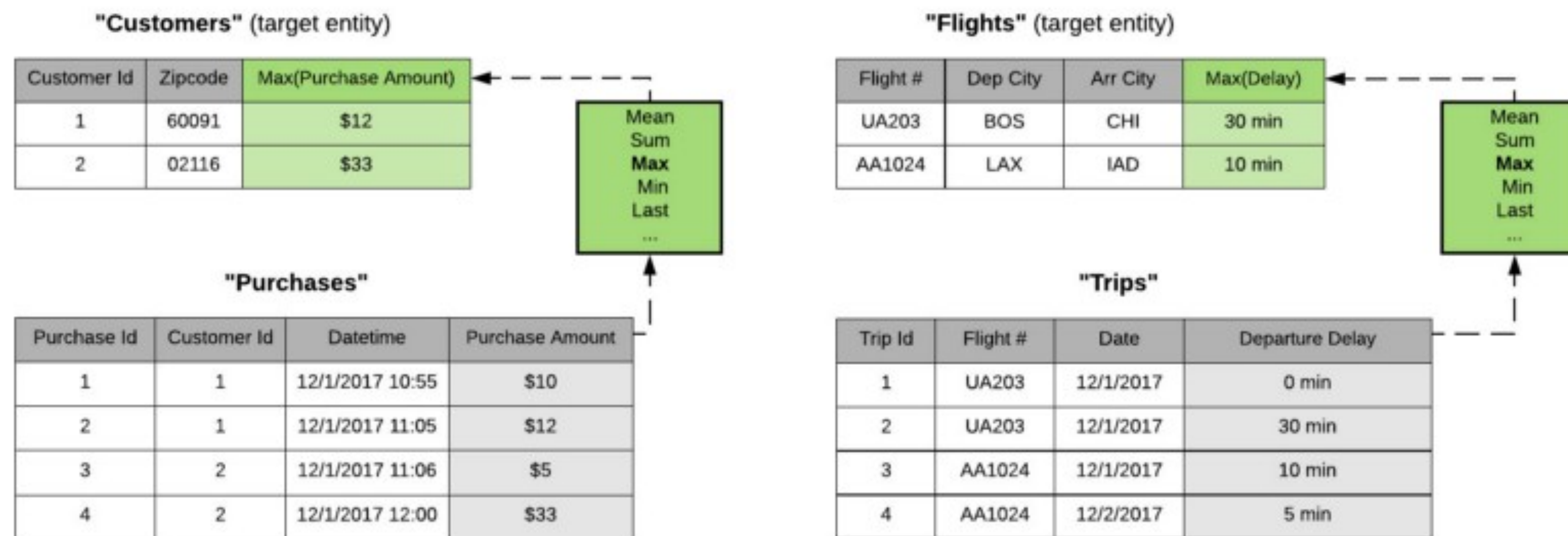
1. Features are derived from relationships between the data points in a dataset.



To calculate a customer's most expensive purchase, we apply the **Max** primitive to the purchase amount field in all related purchases. When we perform the same steps to a dataset of airplane flights, we calculate "the longest flight delay".

# Feature Tools

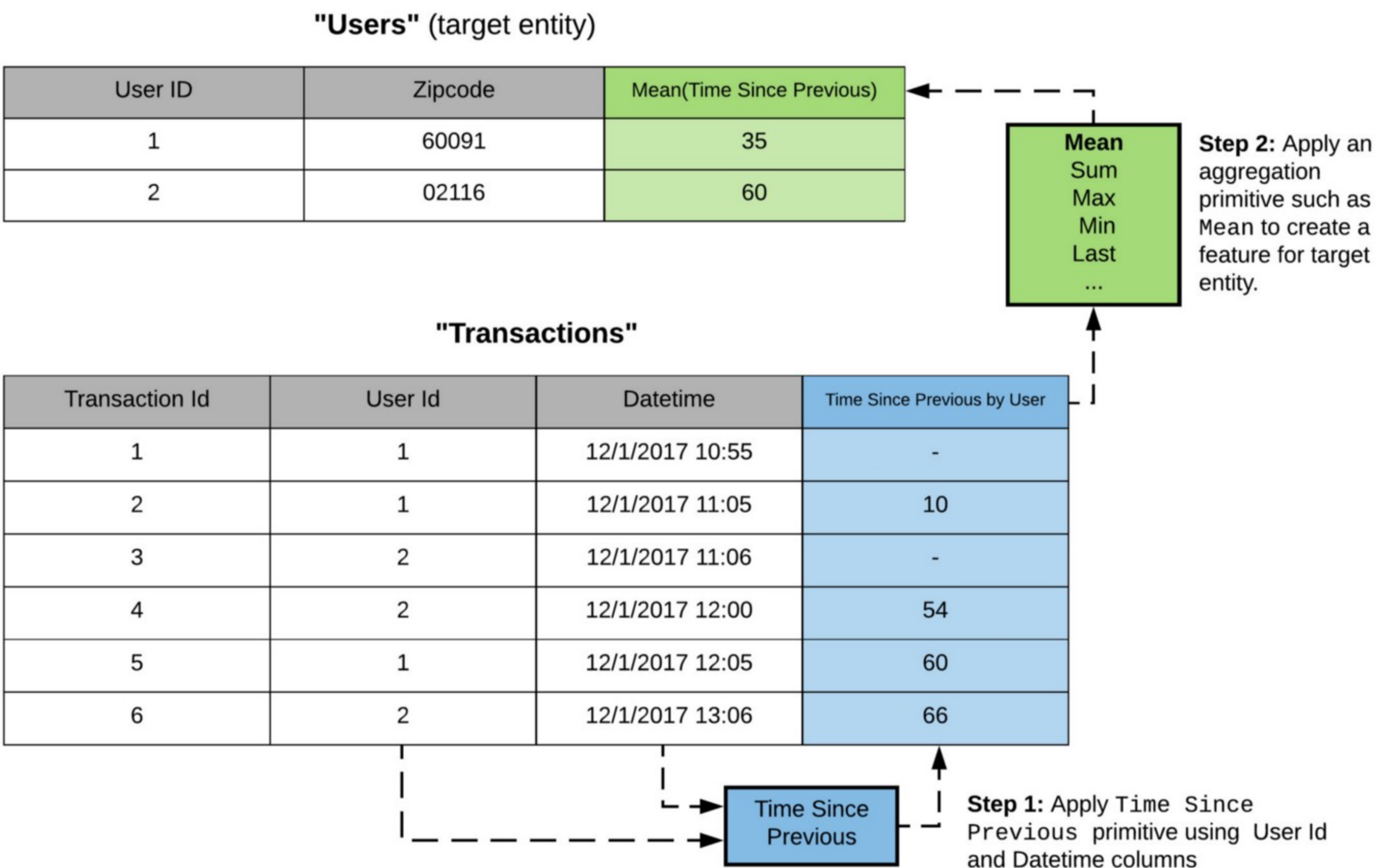
2. Across datasets, many features are derived by using similar mathematical operations.



To calculate a customer's most expensive purchase, we apply the **Max** primitive to the purchase amount field in all related purchases. When we perform the same steps to a dataset of airplane flights, we calculate "the longest flight delay".

# Feature Tools

3. New features are often composed from utilizing previously derived features.



# In Class Exercise

Please take 45 minutes and complete

**Worksheet 4 - Feature Engineering**



# Questions?