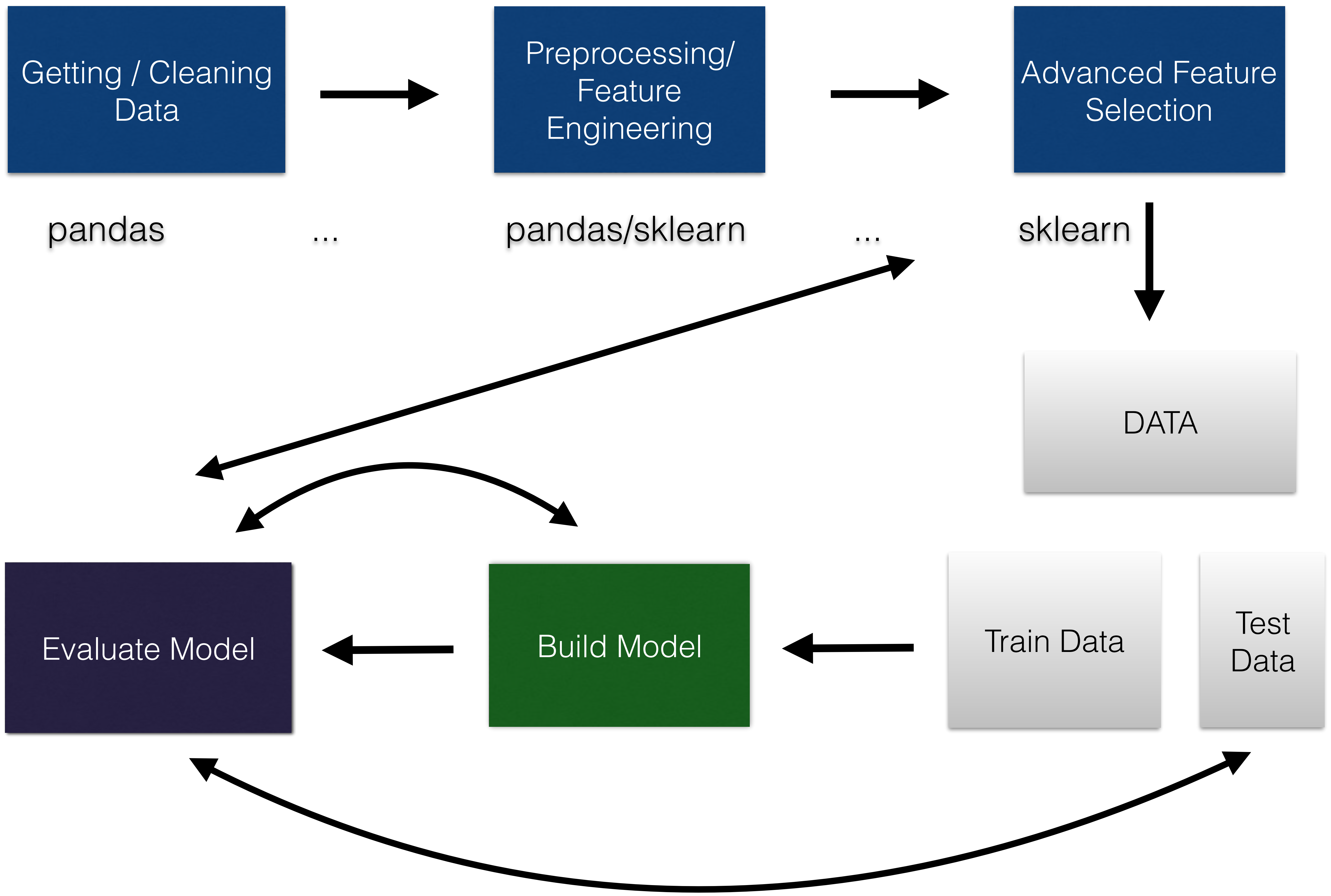
Module 5.1: Machine Learning Part 1
Feature Engineering

From URL strings to “Features”

Agenda

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

Machine Learning Process



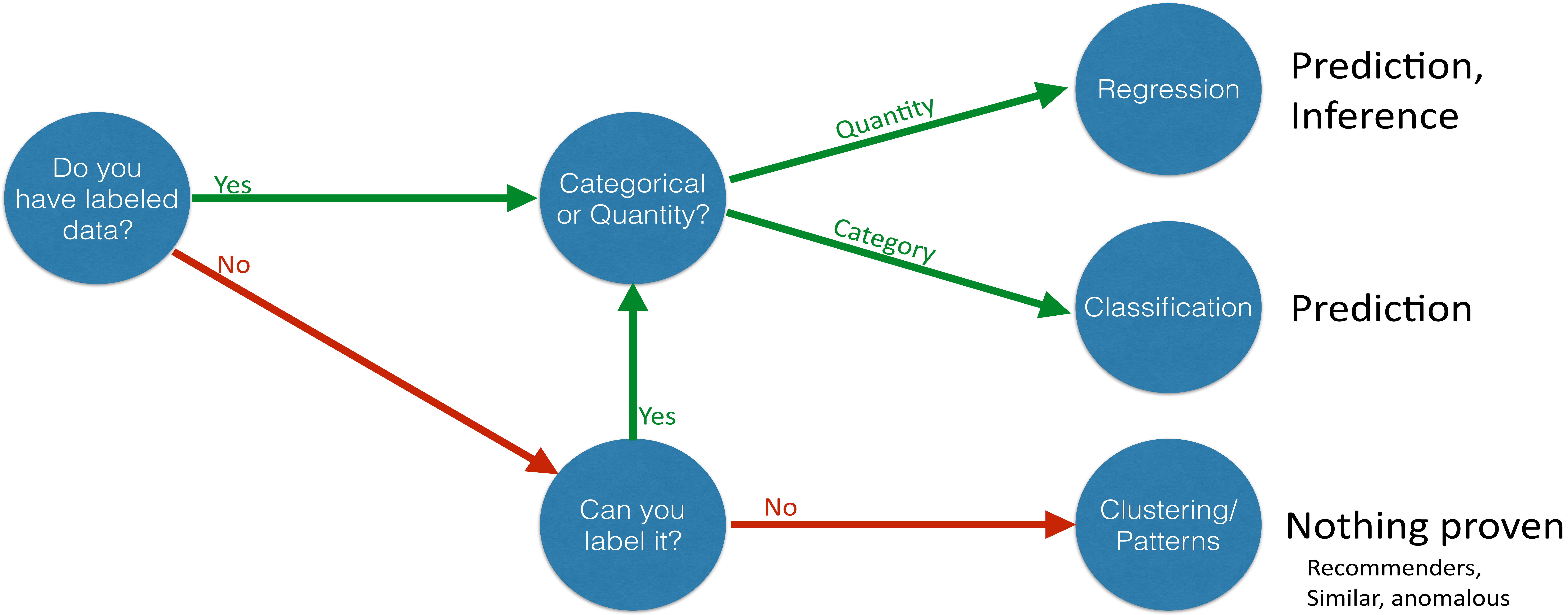
Machine Learning Terms

- **Features:** The mathematical representation of the original data. The features are the columns in your data set. Since the features will be a matrix, the 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.

GET ON THE FAST TRACK



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)

URL Definition

https://www.google.com/search?q=URL&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjcl6ut-lDUAhVEPCYKHdJGDsYQ_AUIDCgD&biw=1215&bih=652

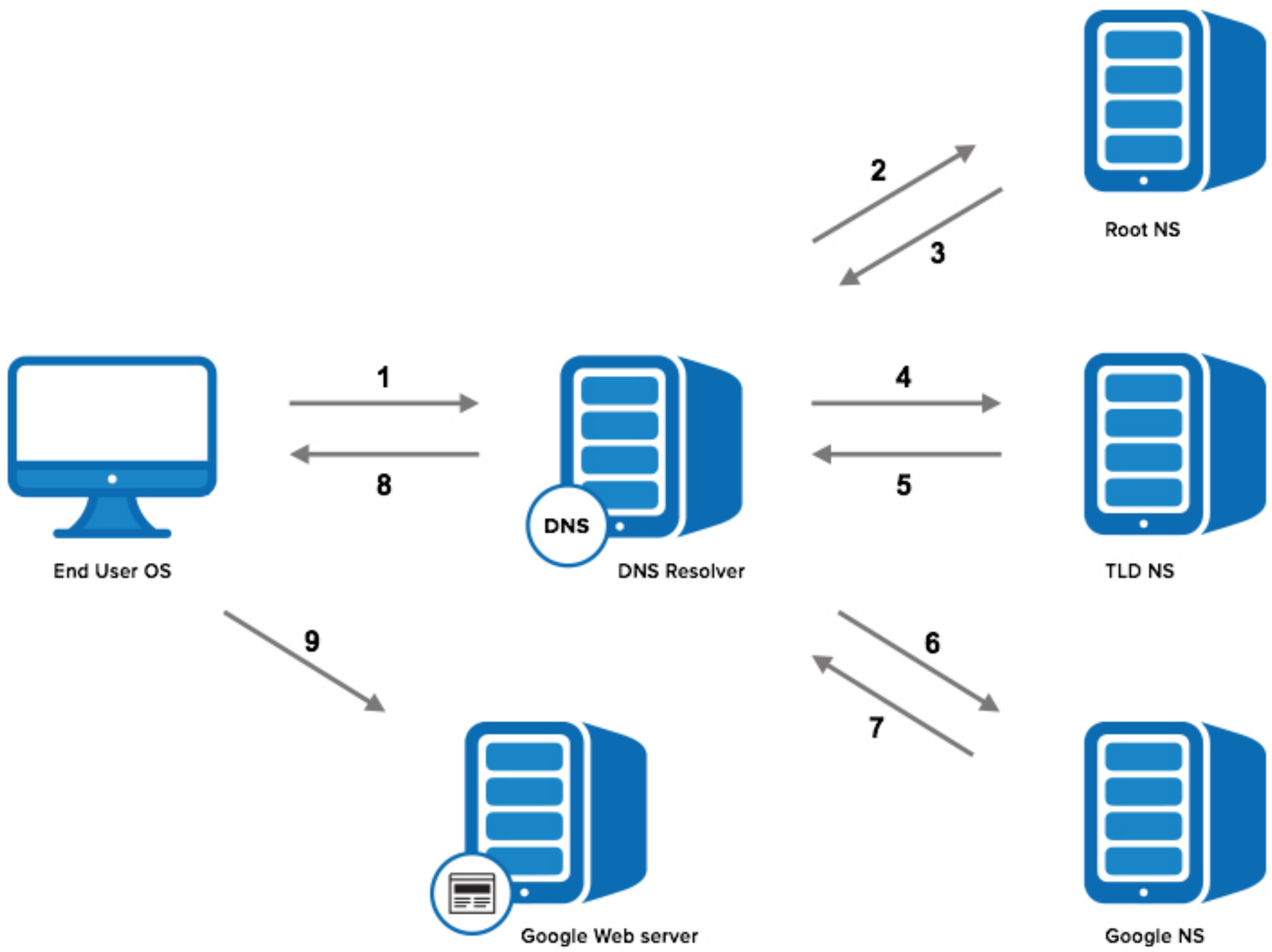
https://	protocol
www	subdomain
google.com	zone apex
google	domain
.com	top-level-domain (tld)
/search?q=URL…	path

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, tldextract, ipaddress`

DNS Flow



Root NS
2
3
1
8
4
5
DNS
End User OS
DNS Resolver
TLD NS
9
6
7
Google Web server
Google NS

What makes them different?

URL BlackList

amazon-sicherheit.kunden-ueberpruefung.xyz

eclipsehotels.com/language/en-GB/eng.exe

bohicacapital.com/page

summerweb.net

ad.getfond.info

vdula.czystykod.pl/rxdjna2.html

svision-online.de/mgfi/administrator/components/com_babackup/classes/fx29id1.txt

URL WhiteList

gurufocus.com/stock/PNC

dvdtalk.ru/review

333cn.com/zx/zhxw.html

made-in-china.com/special/led-lighting

google.com/u/0/112261544981697332354/posts

youtube.com/watch?v=Qp8MQ4shN6U

unesco.org/themes/education-sustainable-developm

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.

Sahoo et al. 2017: [Malicious URL Detection using Machine Learning: A Survey]

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/6d1b281b5c4bbcfe3b99228680c232fa	1	verapdpf.info	2016-08-18 07:09:03

Features or X

Target or y

	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

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)
```

Bag-of-words model fitting

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

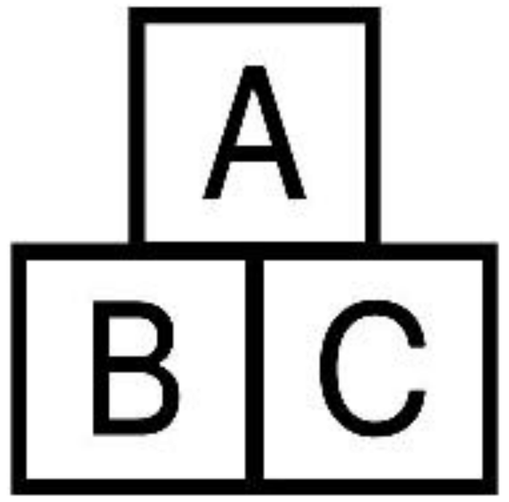
'.com'	'.de'	'.uk'
0	0	0
1	0	0
0	1	0

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

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



Primary Python libraries: `pandas, sklearn, scipy`

http://scikit-learn.org/stable/modules/preprocessing.html

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)
```

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 get inaccurate results**.

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)
```

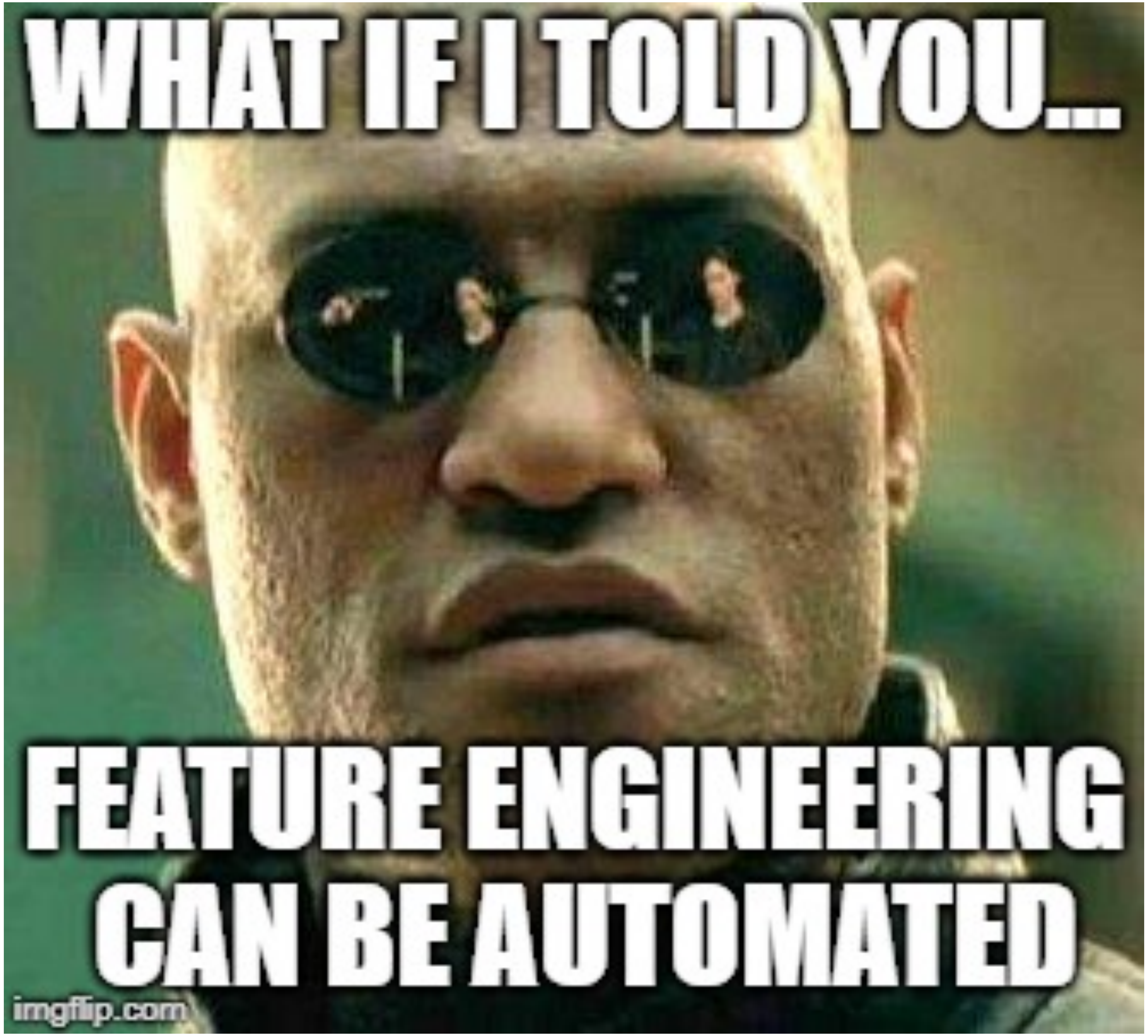
Min/Max Scaling

```
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]
```


WHAT IF I TOLD YOU...
FEATURE ENGINEERING
CAN BE AUTOMATED
imgflip.com

Feature Tools

- Open Source
- Automated Feature Engineering

https://www.featuretools.com/

Feature Tools

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

"Customers" (target entity)

Customer Id	Zipcode	Max(Purchase Amount)
1	60091	$12
2	02116	$33

Mean
Sum
Max
Min
Last
...

"Purchases"

Purchase Id	Customer Id	Datetime	Purchase Amount
1	1	12/1/2017 10:55	$10
2	1	12/1/2017 11:05	$12
3	2	12/1/2017 11:06	$5
4	2	12/1/2017 12:00	$33

"Flights" (target entity)

Flight #	Dep City	Arr City	Max(Delay)
UA203	BOS	CHI	30 min
AA1024	LAX	IAD	10 min

Mean
Sum
Max
Min
Last
...

"Trips"

Trip Id	Flight #	Date	Departure Delay
1	UA203	12/1/2017	0 min
2	UA203	12/1/2017	30 min
3	AA1024	12/1/2017	10 min
4	AA1024	12/2/2017	5 min

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".

https://www.featuretools.com/

Feature Tools

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

"Customers" (target entity)

Customer Id	Zipcode	Max(Purchase Amount)
1	60091	$12
2	02116	$33

Mean
Sum
Max
Min
Last
...

"Purchases"

Purchase Id	Customer Id	Datetime	Purchase Amount
1	1	12/1/2017 10:55	$10
2	1	12/1/2017 11:05	$12
3	2	12/1/2017 11:06	$5
4	2	12/1/2017 12:00	$33

"Flights" (target entity)

Flight #	Dep City	Arr City	Max(Delay)
UA203	BOS	CHI	30 min
AA1024	LAX	IAD	10 min

Mean
Sum
Max
Min
Last
...

"Trips"

Trip Id	Flight #	Date	Departure Delay
1	UA203	12/1/2017	0 min
2	UA203	12/1/2017	30 min
3	AA1024	12/1/2017	10 min
4	AA1024	12/2/2017	5 min

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".

https://www.featuretools.com/

Feature Tools

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

"Users" (target entity)

User ID	Zipcode	Mean(Time Since Previous)
1	60091	35
2	02116	60

Mean
Sum
Max
Min
Last
...

Step 2: Apply an aggregation primitive such as `Mean` to create a feature for target entity.

"Transactions"

Transaction Id	User Id	Datetime	Time Since Previous by User
1	1	12/1/2017 10:55	-
2	1	12/1/2017 11:05	10
3	2	12/1/2017 11:06	-
4	2	12/1/2017 12:00	54
5	1	12/1/2017 12:05	60
6	2	12/1/2017 13:06	66

Time Since Previous

Step 1: Apply `Time Since Previous` primitive using User Id and Datetime columns

https://www.featuretools.com/

Selecting Features

Should we use all of them?

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

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:

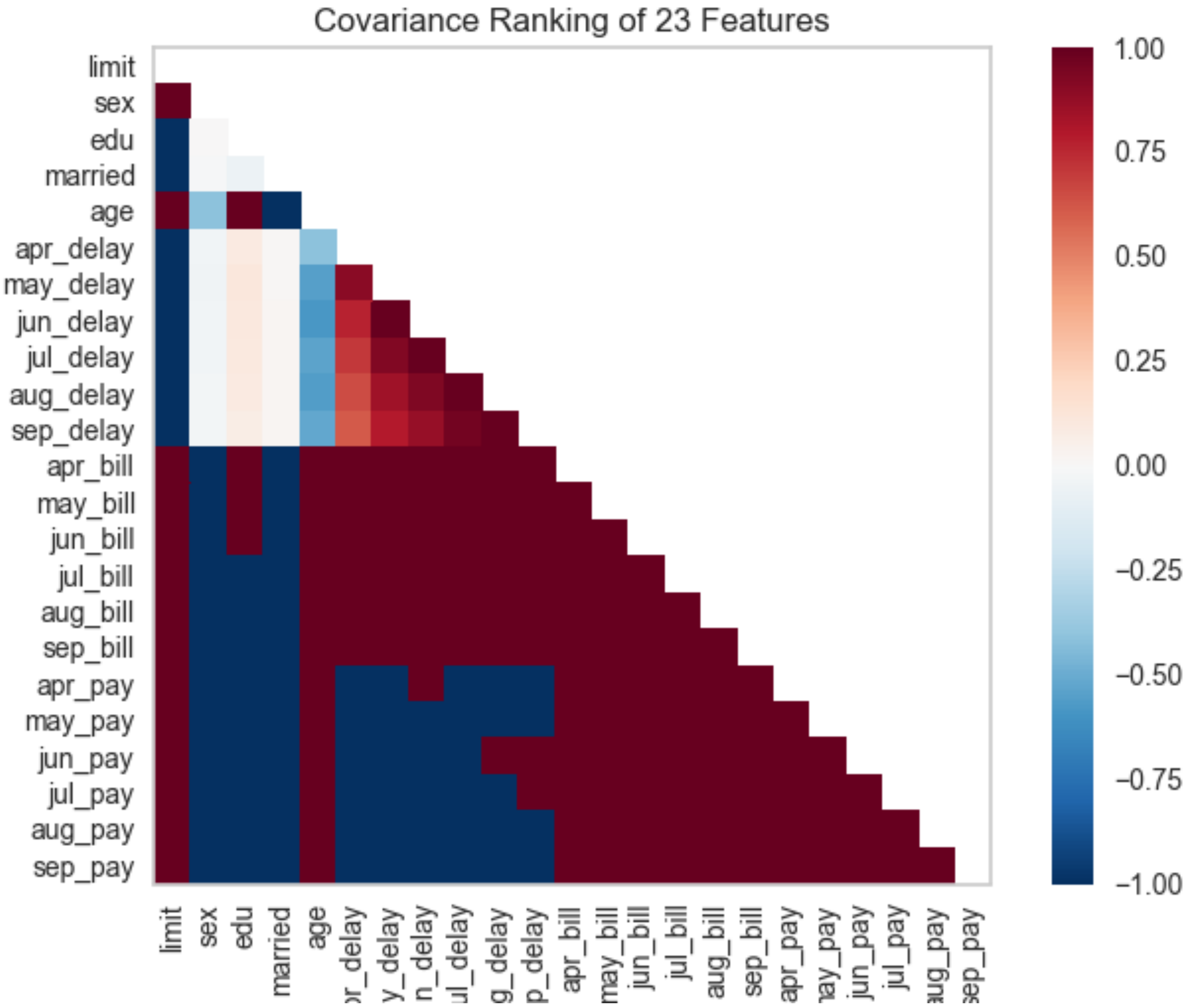
- **For regression:** f_regression, mutual_info_regression
- **For classification:** chi2, f_classif, mutual_info_classif

References:
http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
http://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection

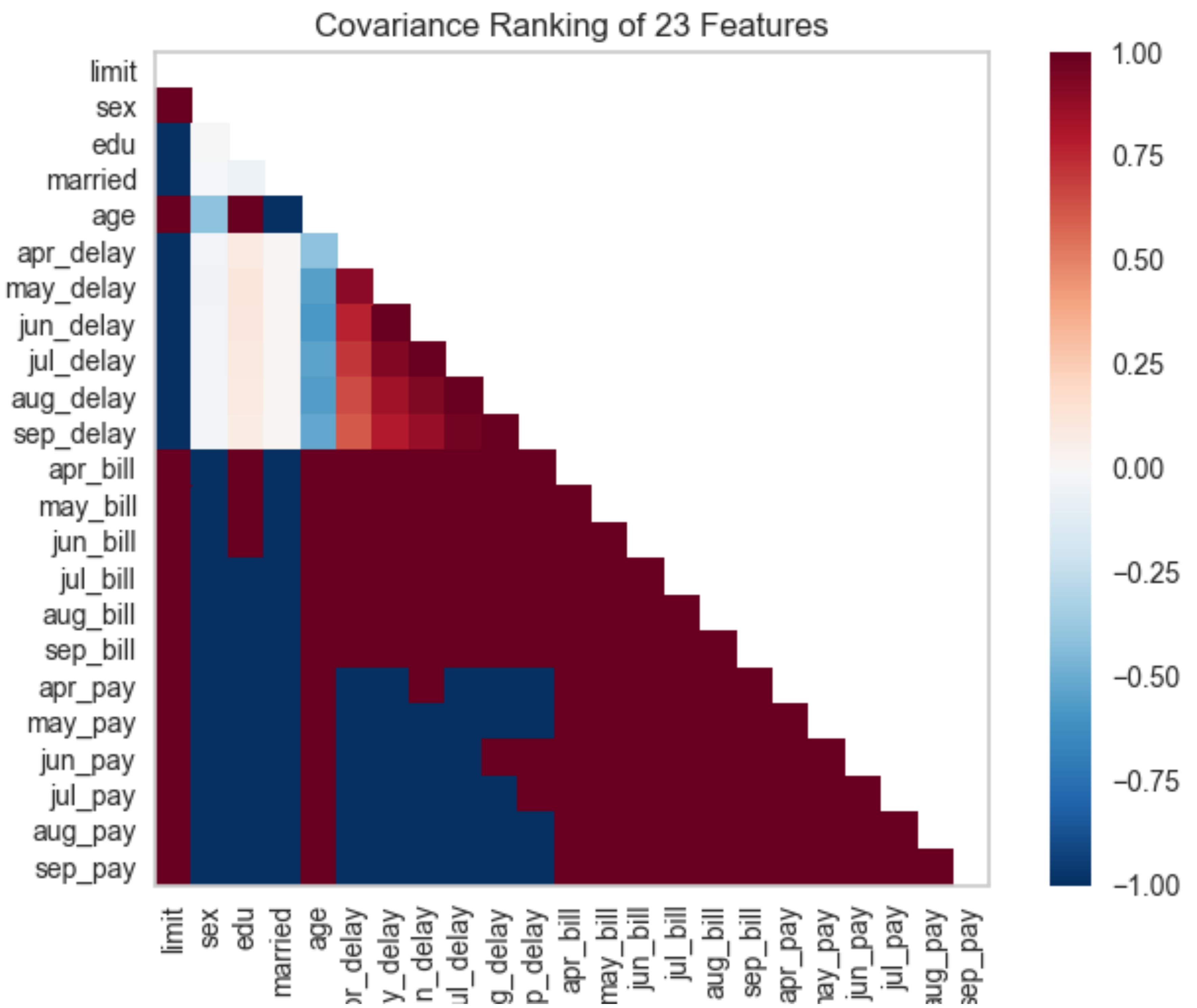
How do we know which features to use and which to discard?

Visualize them!!

Introducing Yellowbrick and scikit-plot

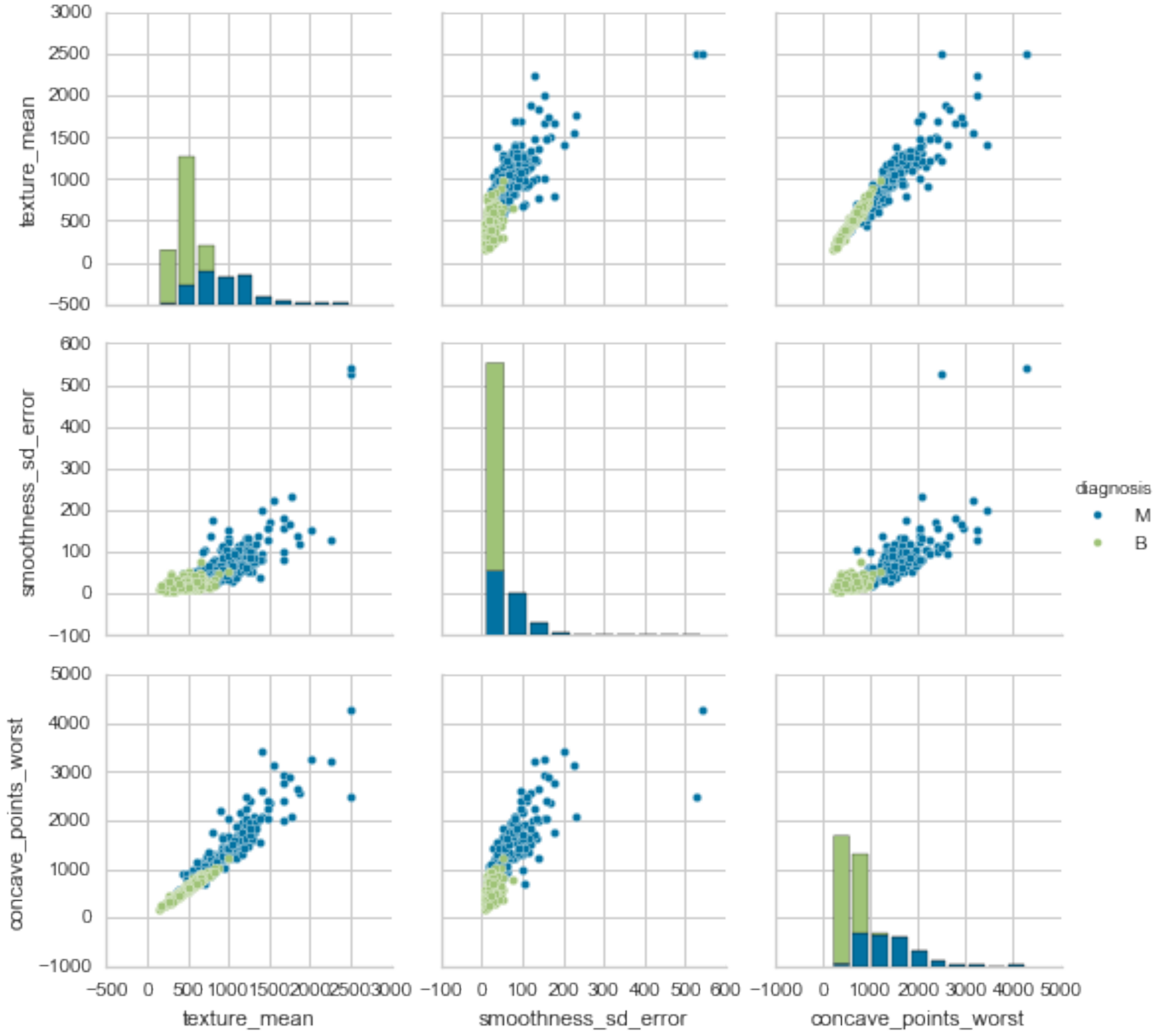


http://www.scikit-yb.org/en/latest/api/features/rankd.html




```python
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()
```


http://www.scikit-yb.org/en/latest/api/features/rankd.html




```python
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()
```

In Class Exercise

Please take 45 minutes and complete

Worksheet 5.1 - Feature Engineering