Why generate features?

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Feature Engineering

House A is a **two** bedroomed house **2000** sq. ft brownstone.

House B is **1500** sq. ft with **one** bedroom.

House	Bedrooms	sq. ft
A	2	2000
B	1	1500
...	...	...

Different types of data

- Continuous: either integers (or whole numbers) or floats (decimals)
- Categorical: one of a limited set of values, e.g. gender, country of birth
- Ordinal: ranked values, often with no detail of distance between them
- Boolean: True/False values
- Datetime: dates and times

Course structure

- Chapter 1: Feature creation and extraction
- Chapter 2: Engineering messy data
- Chapter 3: Feature normalization
- Chapter 4: Working with text features

Pandas

```
import pandas as pd
df = pd.read_csv(path_to_csv_file)
print(df.head())
```

Dataset

```
              SurveyDate  \
0    2018-02-28 20:20:00
1    2018-06-28 13:26:00
2    2018-06-06 03:37:00
3    2018-05-09 01:06:00
4    2018-04-12 22:41:00


                                   FormalEducation
0    Bachelor's degree (BA. BS. B.Eng.. etc.)
1    Bachelor's degree (BA. BS. B.Eng.. etc.)
2    Bachelor's degree (BA. BS. B.Eng.. etc.)
3    Some college/university study  ...
4    Bachelor's degree (BA. BS. B.Eng.. etc.)
```

Column names

```
print(df.columns)
```

```
Index(['SurveyDate', 'FormalEducation',
       'ConvertedSalary', 'Hobby', 'Country',
       'StackOverflowJobsRecommend', 'VersionControl',
       'Age', 'Years Experience', 'Gender',
       'RawSalary'], dtype='object')
```

Column types

```
print(df.dtypes)
```

```
SurveyDate                                     object
FormalEducation                                object
ConvertedSalary                               float64
...
Years Experience                                int64
Gender                                         object
RawSalary                                      object
dtype: object
```

Selecting specific data types

```
only_ints = df.select_dtypes(include=['int'])
print(only_ints.columns)
```

```
Index(['Age', 'Years Experience'], dtype='object')
```

Lets get going!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Dealing with Categorical Variables

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Encoding categorical features

Index	Country
1	‘India’
2	’USA’
3	‘UK’
4	‘UK’
5	‘France’
…	…

Encoding categorical features

Index	Country
1	‘India’
2	’USA’
3	‘UK’
4	‘UK’
5	‘France’
…	…

Index	C_India	C_USA	C_UK	C_France
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	1	0
5	0	0	0	1
…	…	…	…	…

Encoding categorical features

- One-hot encoding
- Dummy encoding

One-hot encoding

```
pd.get_dummies(df, columns=['Country'],
               prefix='C')
```

```
   C_France  C_India  C_UK  C_USA
0         0        1     0      0
1         0        0     0      1
2         0        0     1      0
3         0        0     1      0
4         1        0     0      0
```

Dummy encoding

```
pd.get_dummies(df, columns=['Country'],
               drop_first=True, prefix='C')
```

```
   C_India  C_UK  C_USA
0        1     0      0
1        0     0      1
2        0     1      0
3        0     1      0
4        0     0      0
```

One-hot vs. dummies

- **One-hot encoding**: Explainable features
- **Dummy encoding**: Necessary information without duplication

Index	Sex
0	Male
1	Female
2	Male

Index	Male	Female
0	1	0
1	0	1
2	1	0

Index	Male
0	1
1	0
2	1

Limiting your columns

```
counts = df['Country'].value_counts()
print(counts)
```

```
'USA'       8
'UK'        6
'India'     2
'France'    1
Name: Country, dtype: object
```

Limiting your columns

```
mask = df['Country'].isin(counts[counts < 5].index)
df['Country'][mask] = 'Other'
print(pd.value_counts(colors))
```

```
'USA'      8
'UK'       6
'Other'    3
Name: Country, dtype: object
```

Now you deal with categorical variables

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Numeric variables

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Types of numeric features

- Age
- Price
- Counts
- Geospatial data

Does size matter?

	Resturant_ID	Number_of_Violations
0	RS_1	0
1	RS_2	0
2	RS_3	2
3	RS_4	1
4	RS_5	0
5	RS_6	0
6	RS_7	4
7	RS_8	4
8	RS_9	1
9	RS_10	0

Binarizing numeric variables

```
df['Binary_Violation'] = 0
df.loc[df['Number_of_Violations'] > 0,
       'Binary_Violation'] = 1
```

Binarizing numeric variables

	Resturant_ID	Number_of_Violations	Binary_Violation
0	RS_1	0	0
1	RS_2	0	0
2	RS_3	2	1
3	RS_4	1	1
4	RS_5	0	0
5	RS_6	0	0
6	RS_7	4	1
7	RS_8	4	1
8	RS_9	1	1
9	RS_10	0	0

Binning numeric variables

```
import numpy as np
df['Binned_Group'] = pd.cut(
    df['Number_of_Violations'],
    bins=[-np.inf, 0, 2, np.inf],
    labels=[1, 2, 3]
)
```

Binning numeric variables

	Resturant_ID	Number_of_Violations	Binned_Group
0	RS_1	0	1
1	RS_2	0	1
2	RS_3	2	2
3	RS_4	1	2
4	RS_5	0	1
5	RS_6	0	1
6	RS_7	4	3
7	RS_8	4	3
8	RS_9	1	2
9	RS_10	0	1

Lets start practicing!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Why do missing values exist?

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

datacamp

How gaps in data occur

- Data not being collected properly
- Collection and management errors
- Data intentionally being omitted
- Could be created due to transformations of the data

Why we care?

- Some models cannot work with missing data (Nulls/NaNs)
- Missing data may be a sign of a wider data issue
- Missing data can be a useful feature

Missing value discovery

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 12 columns):
SurveyDate                    999 non-null object
...
StackOverflowJobsRecommend    487 non-null float64
VersionControl                999 non-null object
Gender                        693 non-null object
RawSalary                     665 non-null object
dtypes: float64(2), int64(2), object(8)
memory usage: 93.7+ KB
```

Finding missing values

```
print(df.isnull())
```

```
   StackOverflowJobsRecommend  VersionControl  ... \
0                        True           False  ...
1                       False           False  ...
2                       False           False  ...
3                        True           False  ...
4                       False           False  ...

   Gender  RawSalary
0   False       True
1   False      False
2    True       True
3   False      False
4   False      False
```

Finding missing values

```
print(df['StackOverflowJobsRecommend'].isnull().sum())
```

```
512
```

Finding non-missing values

```
print(df.notnull())
```

```
   StackOverflowJobsRecommend  VersionControl  ... \
0                       False            True  ...
1                        True            True  ...
2                        True            True  ...
3                       False            True  ...
4                        True            True  ...


   Gender  RawSalary
0    True      False
1    True       True
2   False      False
3    True       True
4    True       True
```

Go ahead and find missing values!

Dealing with missing values (I)

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

datacamp

Listwise deletion

```
      SurveyDate      ConvertedSalary     Hobby ... \
0  2/28/18 20:20                  NaN       Yes ...
1  6/28/18 13:26              70841.0       Yes ...
2    6/6/18 3:37                  NaN        No ...
3    5/9/18 1:06              21426.0       Yes ...
4  4/12/18 22:41              41671.0       Yes ...
```

Listwise deletion in Python

```
# Drop all rows with at least one missing values
df.dropna(how='any')
```

Listwise deletion in Python

```
# Drop rows with missing values in a specific column
df.dropna(subset=['VersionControl'])
```

Issues with deletion

- It deletes valid data points
- Relies on randomness
- Reduces information

Replacing with strings

```
# Replace missing values in a specific column
# with a given string
df['VersionControl'].fillna(
    value='None Given', inplace=True
)
```

Recording missing values

```
# Record where the values are not missing
df['SalaryGiven'] = df['ConvertedSalary'].notnull()
```

```
# Drop a specific column
df.drop(columns=['ConvertedSalary'])
```

Practice time

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Fill continuous missing values

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

datacamp

Deleting missing values

- Can't delete rows with missing values in the test set

What else can you do?

- **Categorical columns:** Replace missing values with the most common occurring value or with a string that flags missing values such as 'None'
- **Numeric columns:** Replace missing values with a suitable value

Measures of central tendency

- Mean
- Median

Calculating the measures of central tendency

```
print(df['ConvertedSalary'].mean())
print(df['ConvertedSalary'].median())
```

```
92565.16992481203
55562.0
```

Fill the missing values

```
df['ConvertedSalary'] = df['ConvertedSalary'].fillna(
    df['ConvertedSalary'].mean()
)
```

```
df['ConvertedSalary'] = df['ConvertedSalary']\
                             .astype('int64')
```

Rounding values

```
df['ConvertedSalary'] = df['ConvertedSalary'].fillna(
    round(df['ConvertedSalary'].mean())
)
```

Let's Practice!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Dealing with other data issues

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Bad characters

```
print(df['RawSalary'].dtype)
```

```
dtype('O')
```

Bad characters

```
print(df['RawSalary'].head())
```

```
0          NaN
1    70,841.00
2          NaN
3    21,426.00
4    41,671.00
Name: RawSalary, dtype: object
```

Dealing with bad characters

```
df['RawSalary'] = df['RawSalary'].str.replace(',', '')
```

```
df['RawSalary'] = df['RawSalary'].astype('float')
```

Finding other stray characters

```
coerced_vals = pd.to_numeric(df['RawSalary'],
                             errors='coerce')
```

Finding other stray characters

```
print(df[coerced_vals.isna()].head())
```

```
0           NaN
2           NaN
4     $51408.00
Name: RawSalary, dtype: object
```

Chaining methods

```
df['column_name'] = df['column_name'].method1()
df['column_name'] = df['column_name'].method2()
df['column_name'] = df['column_name'].method3()
```

Same as:

```
df['column_name'] = df['column_name']\
                    .method1().method2().method3()
```

Go ahead and fix bad characters!

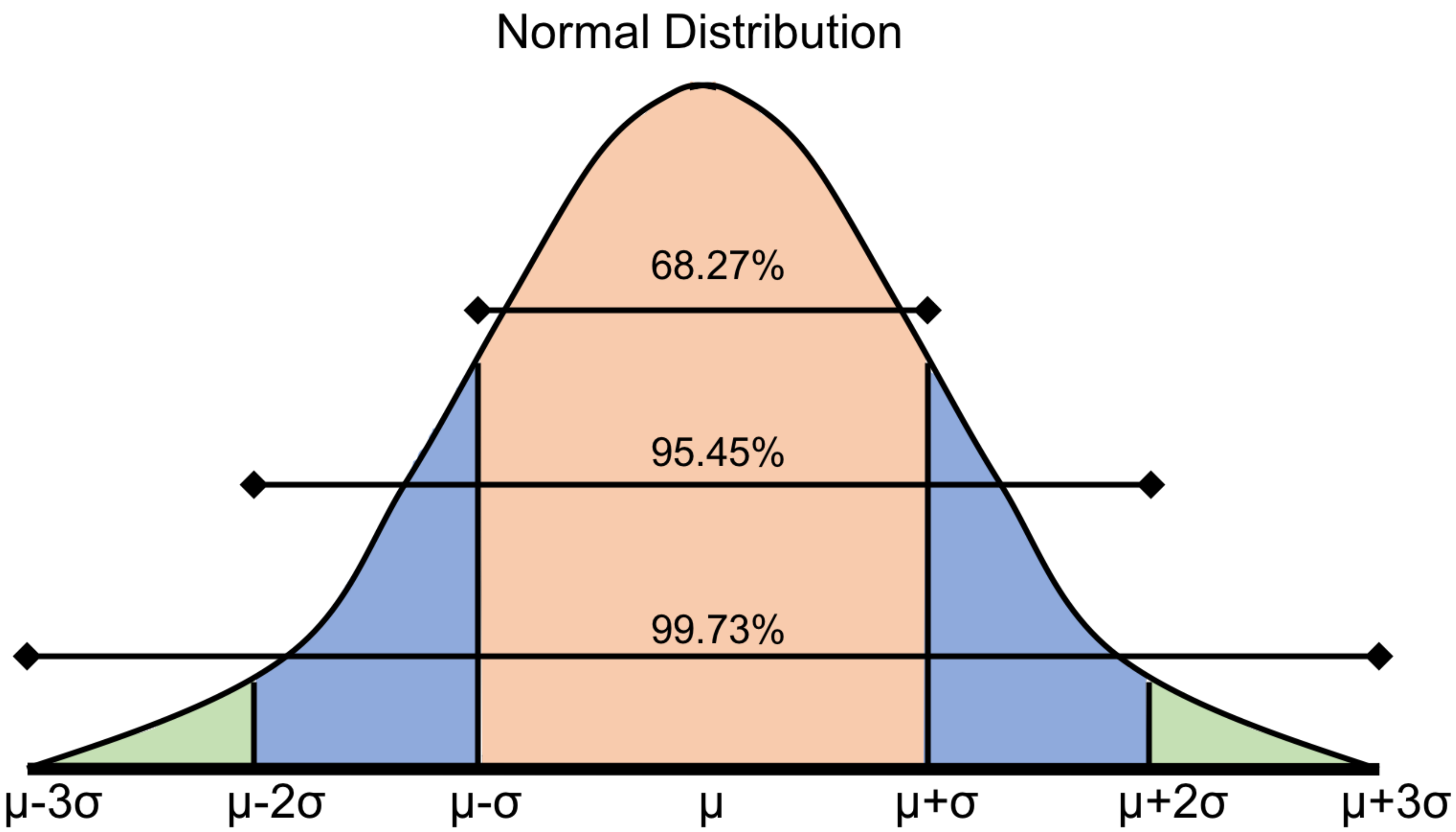
FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Data distributions

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

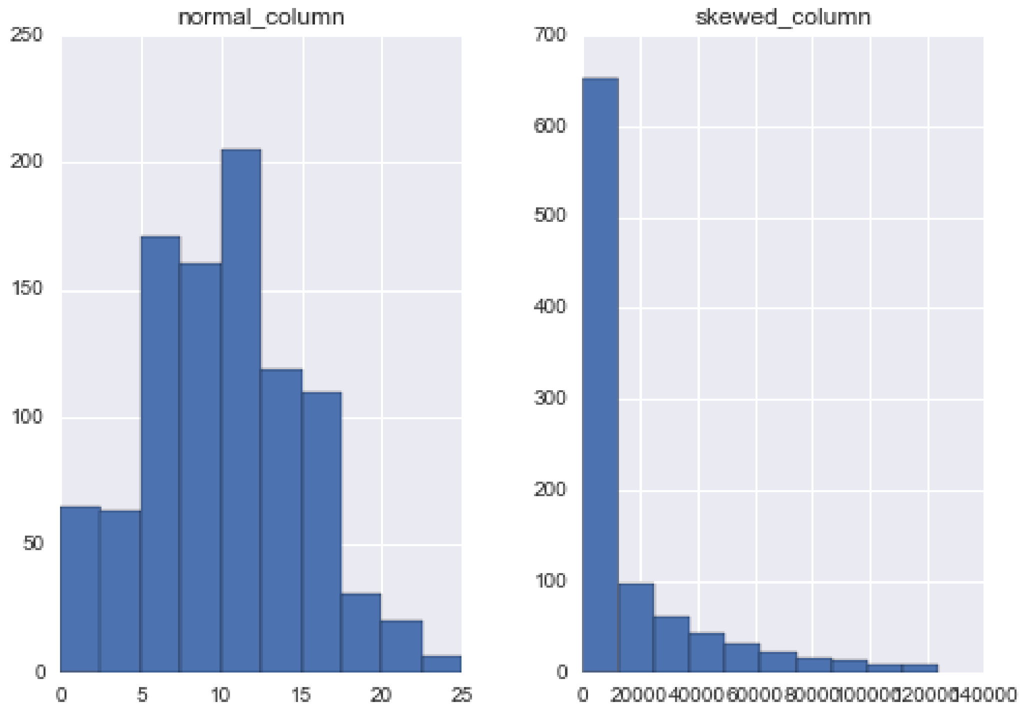
Distribution assumptions



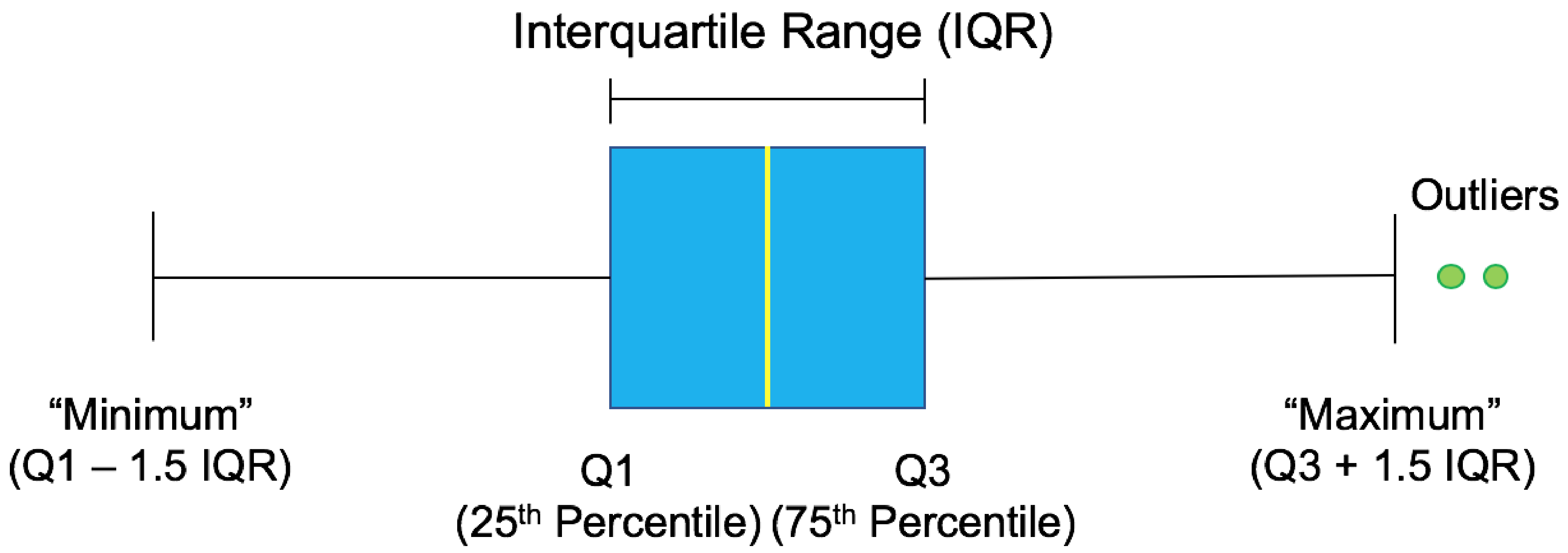
Observing your data

```python
import matplotlib as plt

df.hist()
plt.show()
```

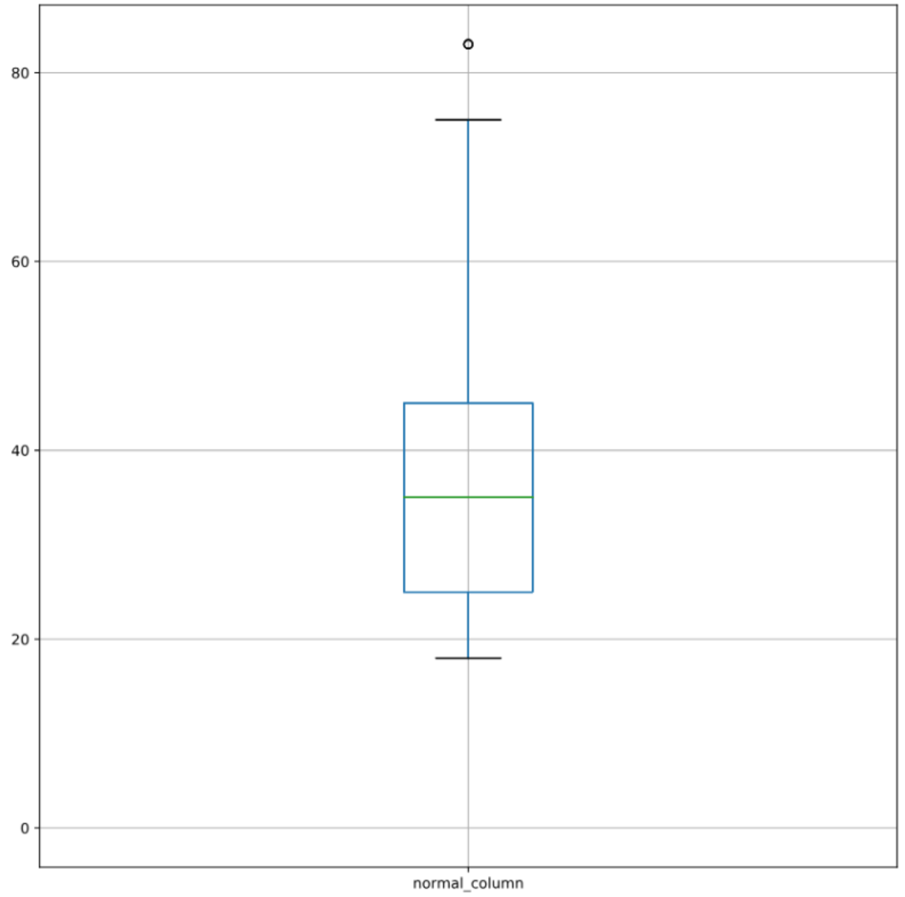


Delving deeper with box plots



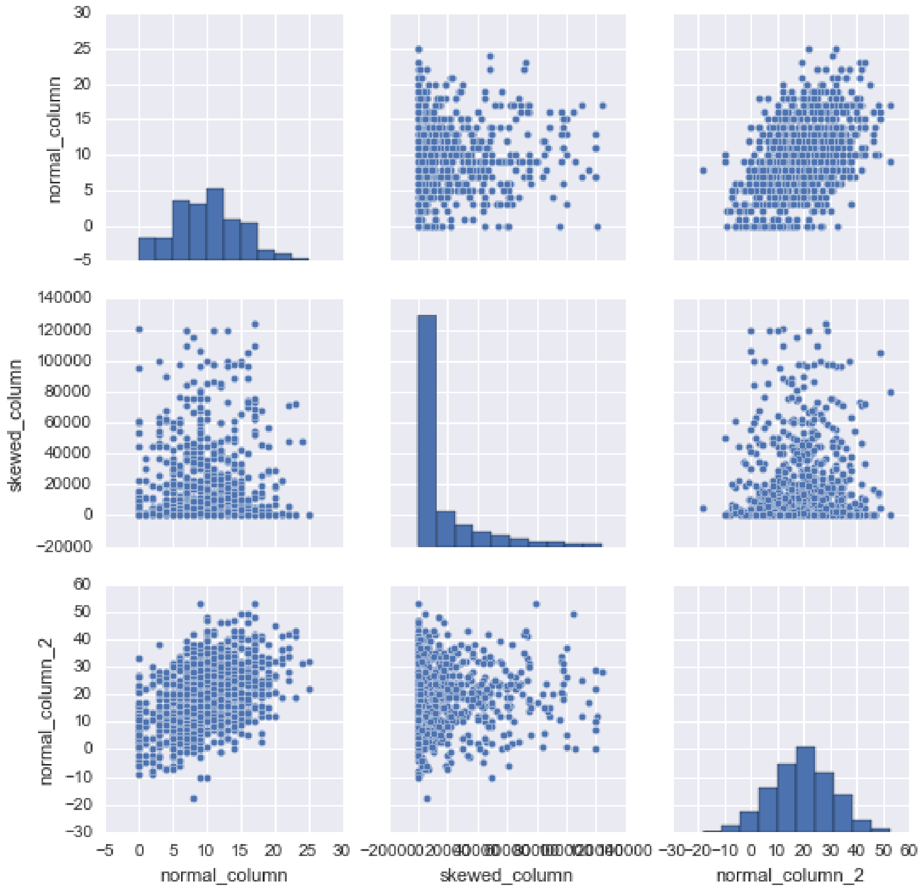
Box plots in pandas

```
df[['column_1']].boxplot()
plt.show()
```



Paring distributions

```python
import seaborn as sns
sns.pairplot(df)
```



Further details on your distributions

```
df.describe()
```

	Col1	Col2	Col3	Col4
count	100.000000	100.000000	100.000000	100.000000
mean	-0.163779	-0.014801	-0.087965	-0.045790
std	1.046370	0.920881	0.936678	0.916474
min	-2.781872	-2.156124	-2.647595	-1.957858
25%	-0.849232	-0.655239	-0.602699	-0.736089
50%	-0.179495	0.032115	-0.051863	0.066803
75%	0.663515	0.615688	0.417917	0.689591
max	2.466219	2.353921	2.059511	1.838561

Let's practice!

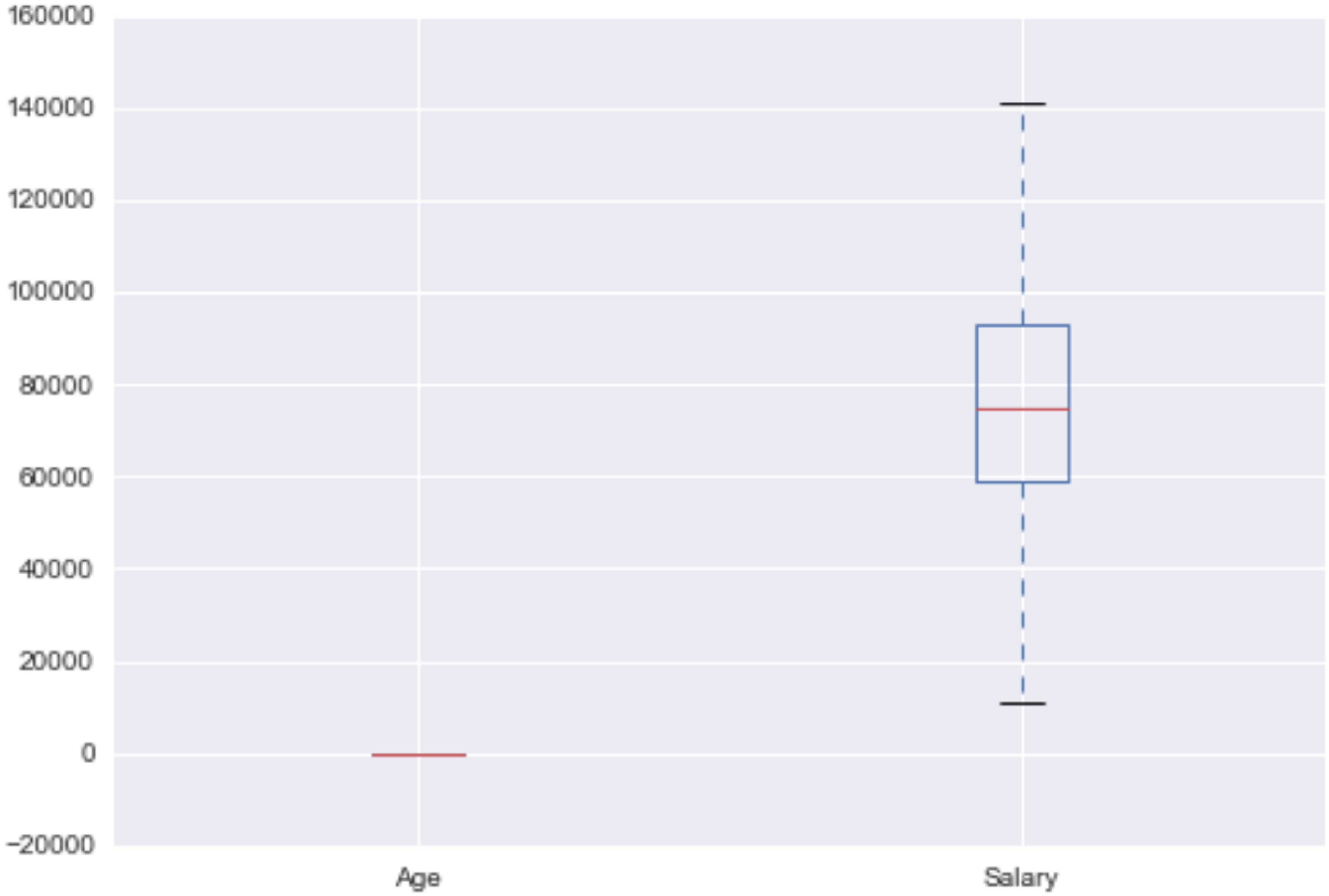
FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Scaling and transformations

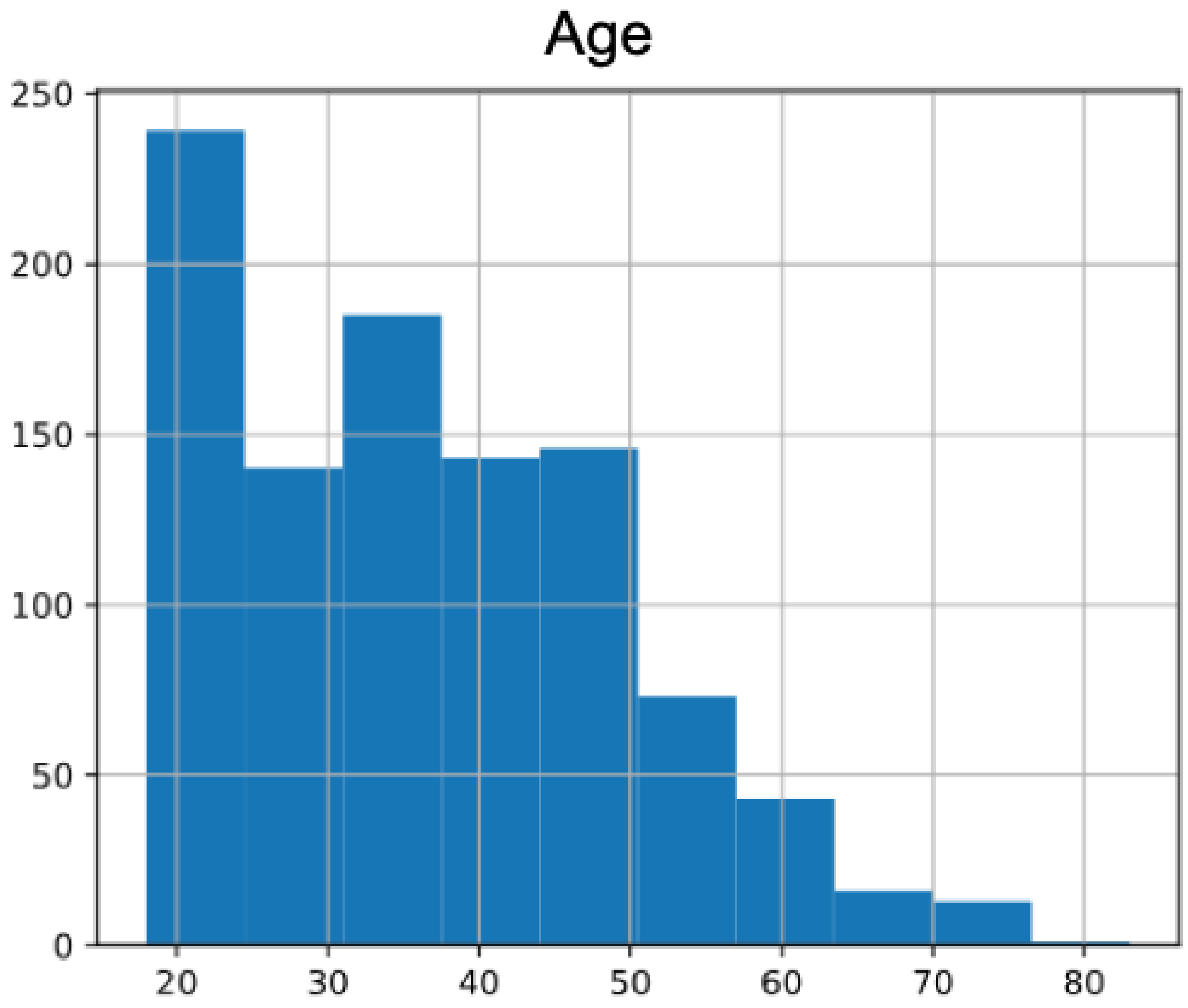
FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Data Scientist

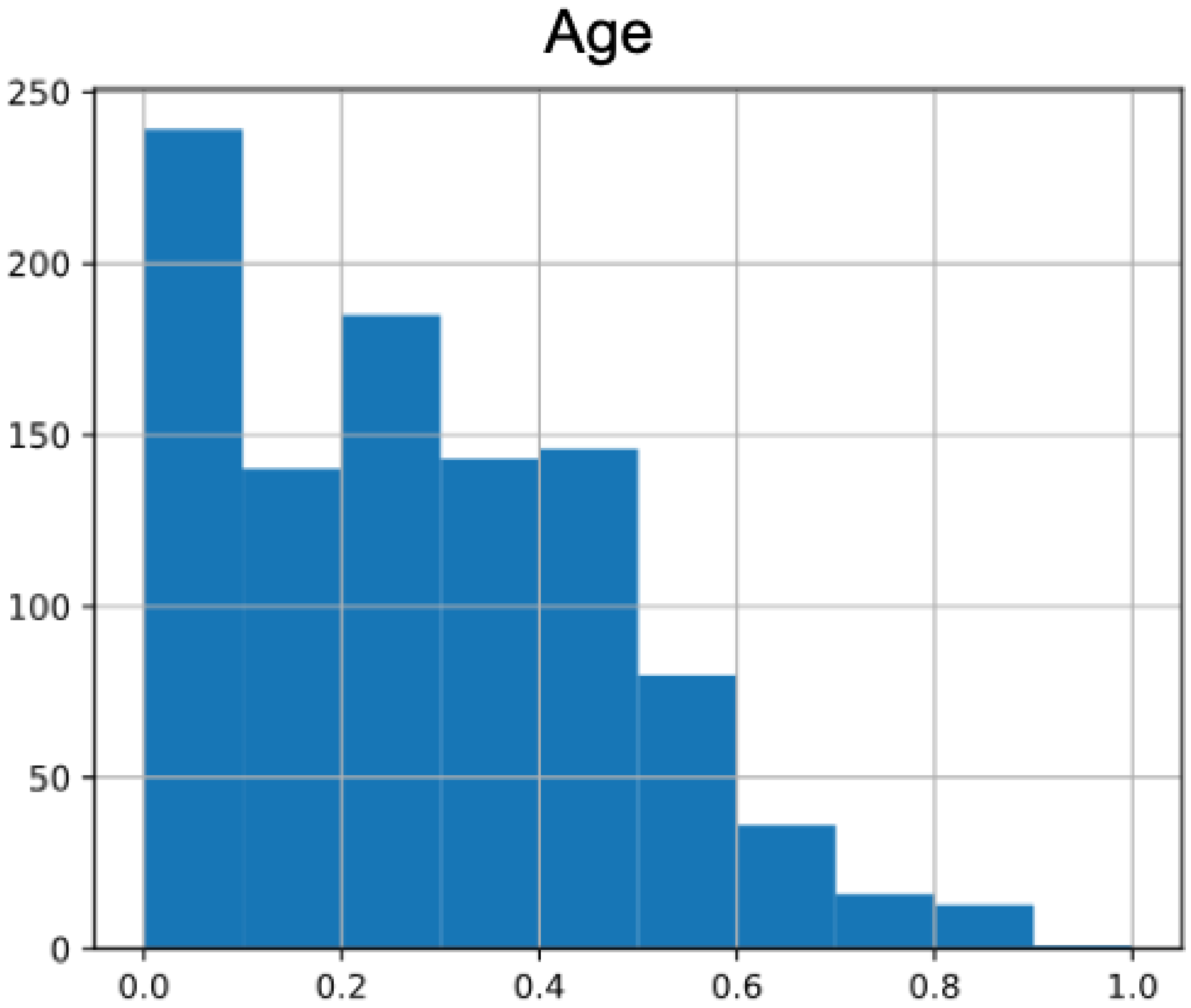
Scaling data



Min-Max scaling



Min-Max scaling



Min-Max scaling in Python

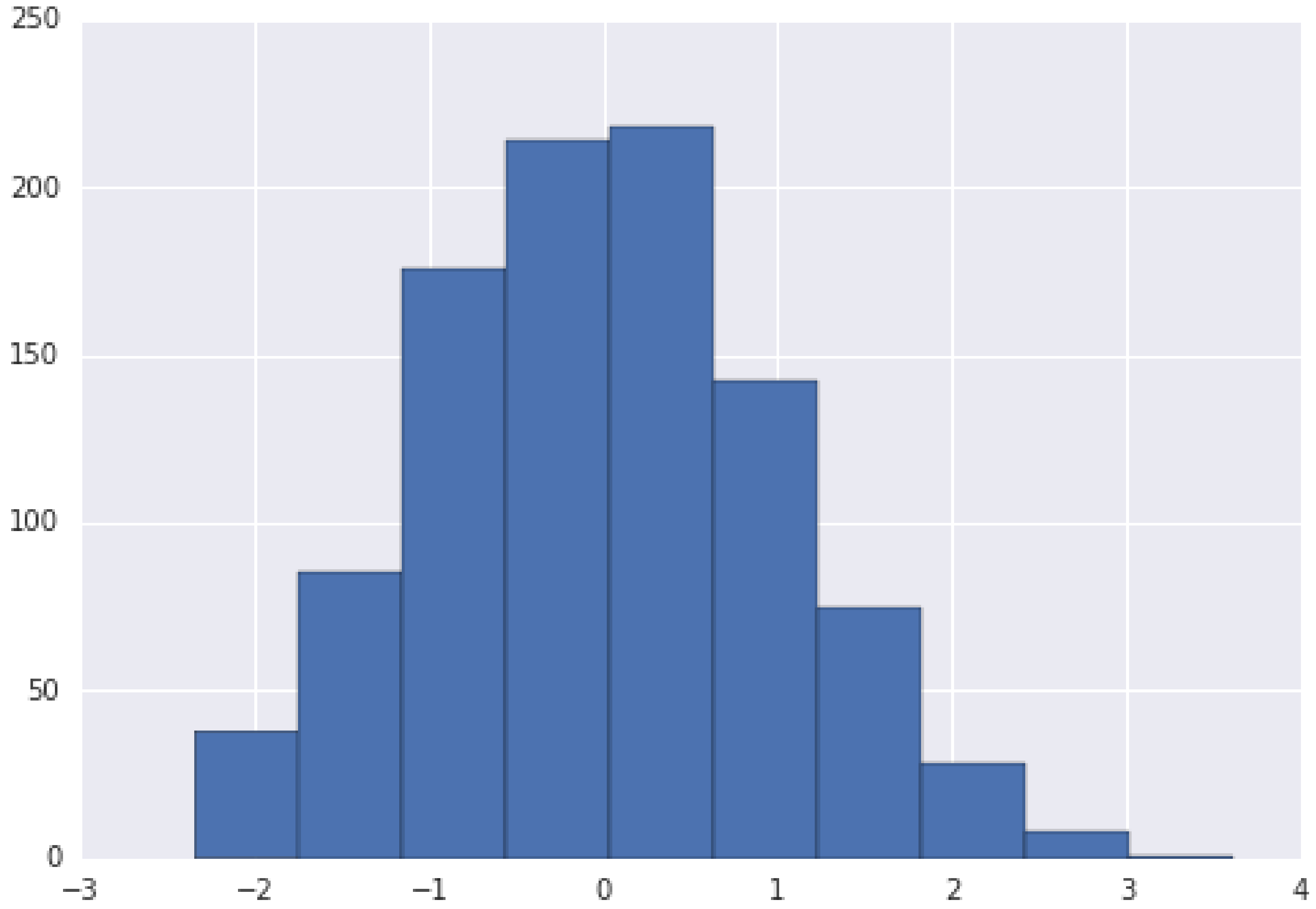
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(df[['Age']])

df['normalized_age'] = scaler.transform(df[['Age']])
```

Standardization



Standardization in Python

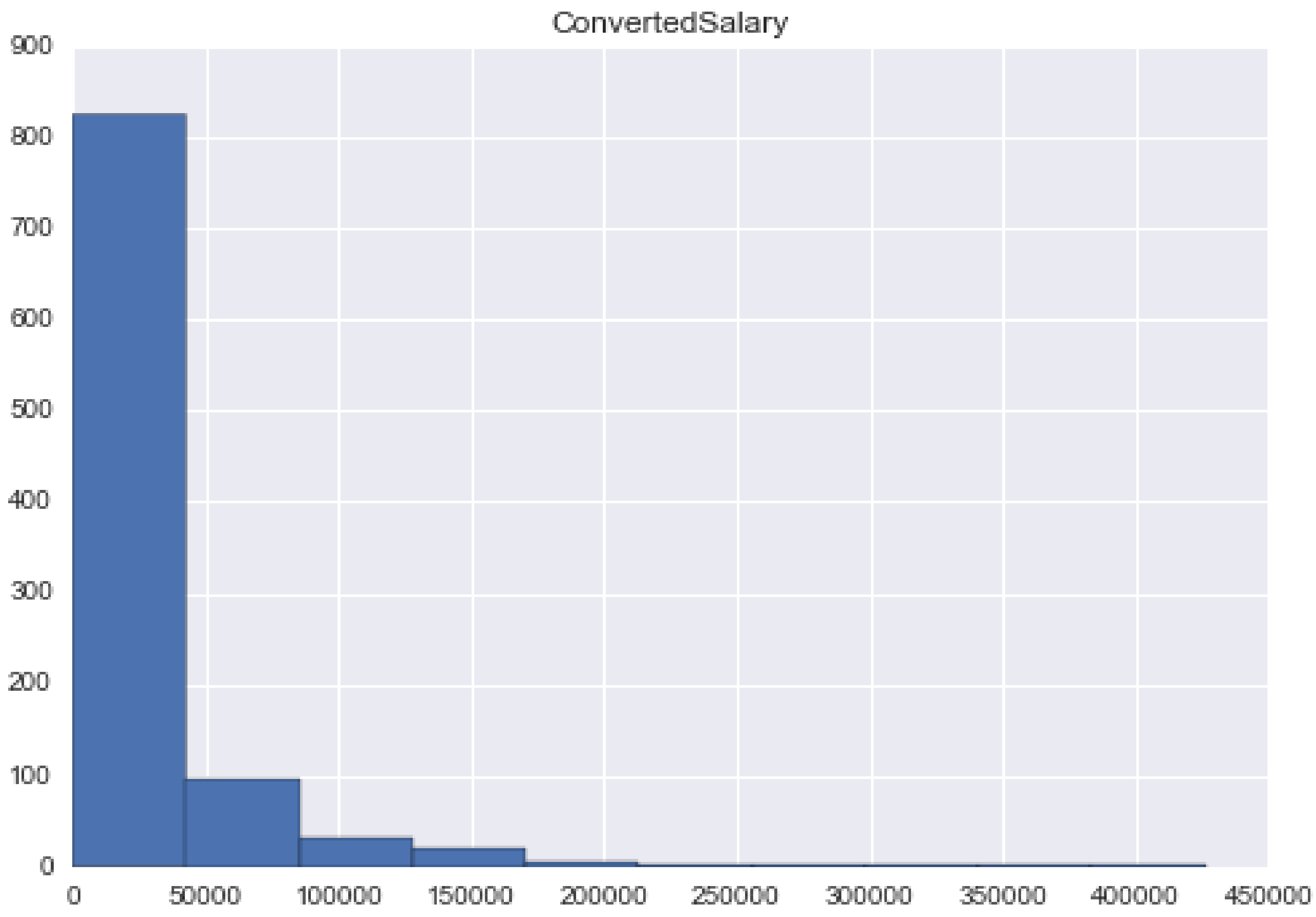
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(df[['Age']])

df['standardized_col'] = scaler\
                            .transform(df[['Age']])
```

Log Transformation



Log transformation in Python

```
from sklearn.preprocessing import PowerTransformer

log = PowerTransformer()

log.fit(df[['ConvertedSalary']])

df['log_ConvertedSalary'] =
     log.transform(df[['ConvertedSalary']])
```

Final Slide

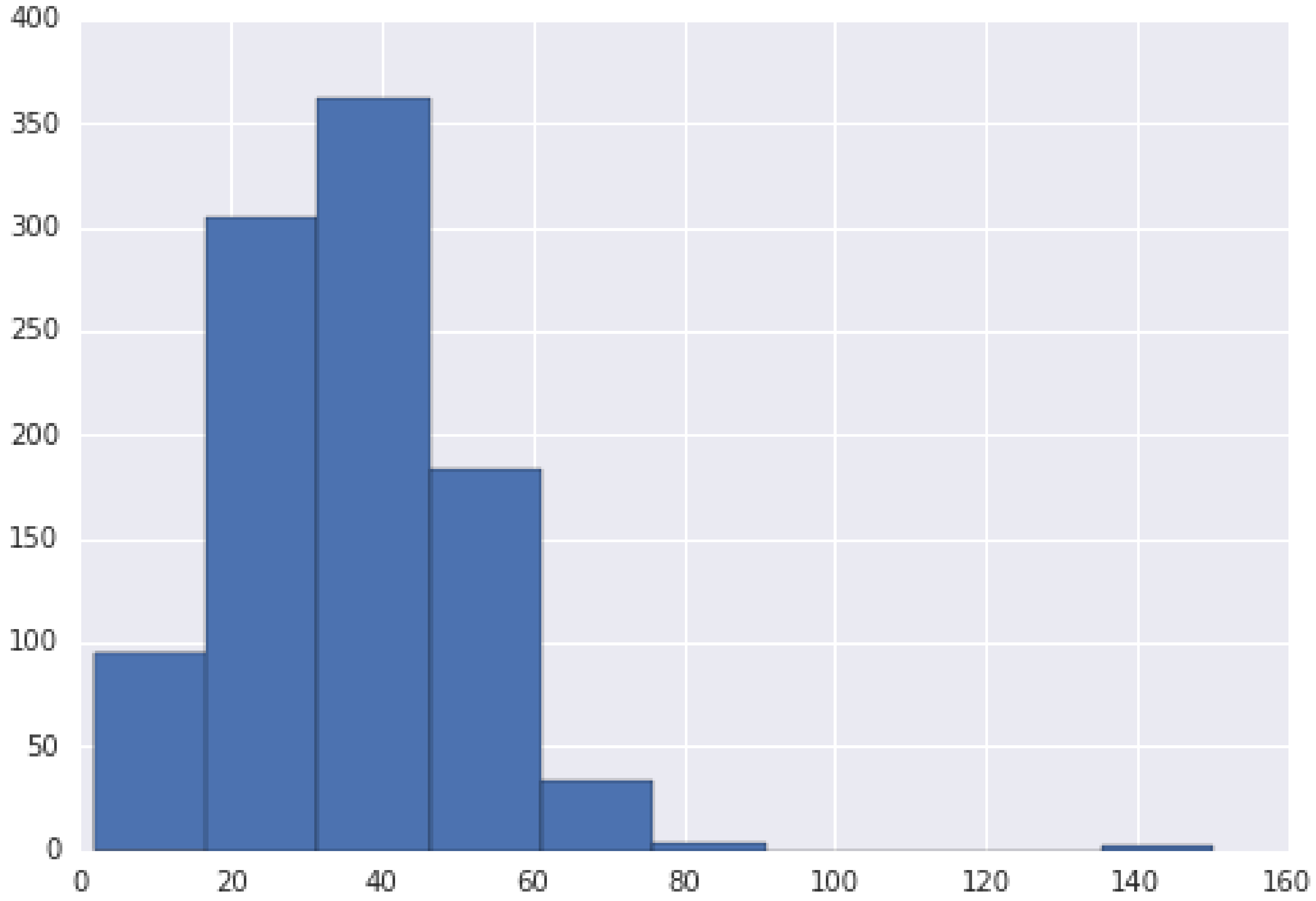
Removing outliers

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

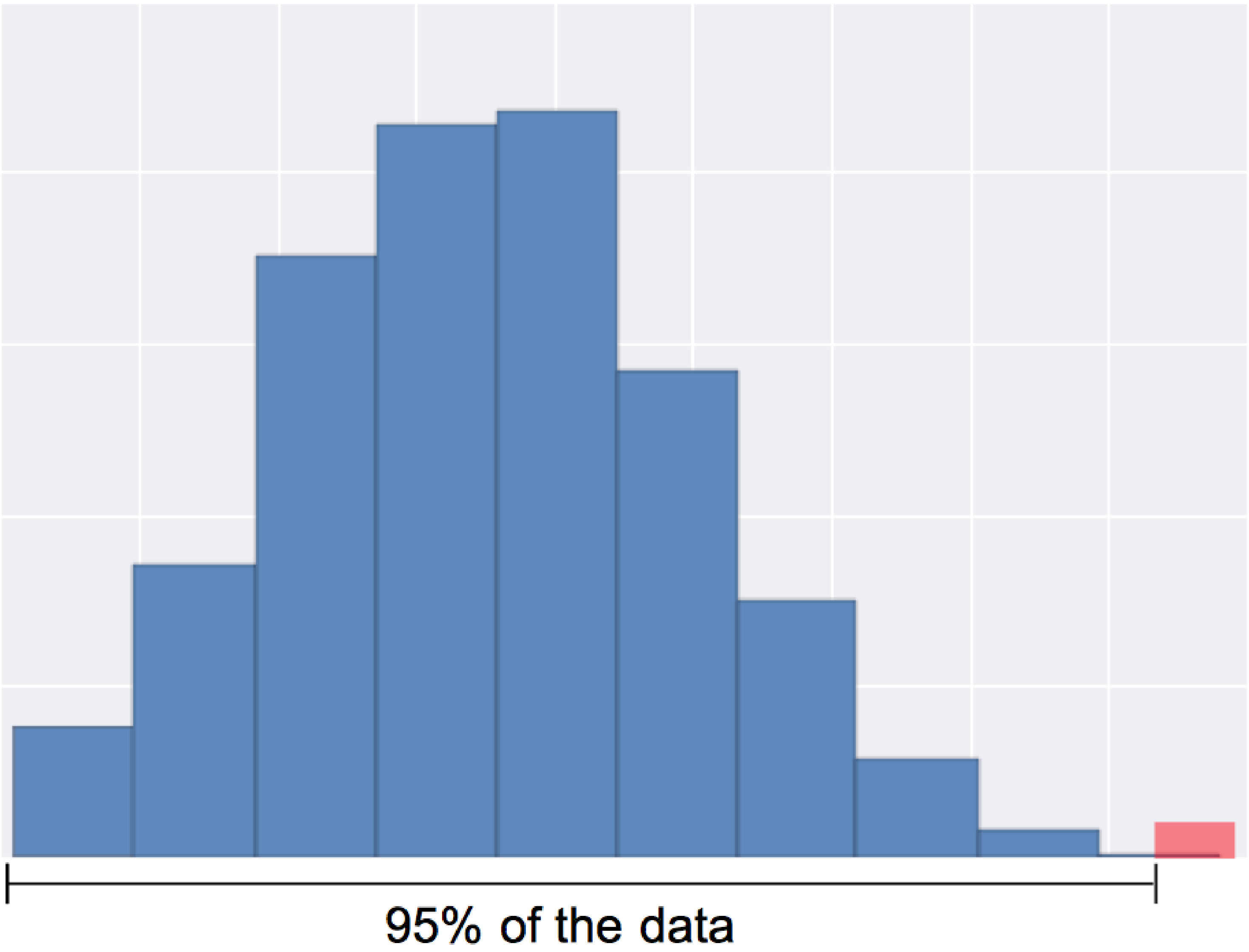
Robert O'Callaghan
Director of Data Science, Ordergroove

datacamp

What are outliers?



Quantile based detection



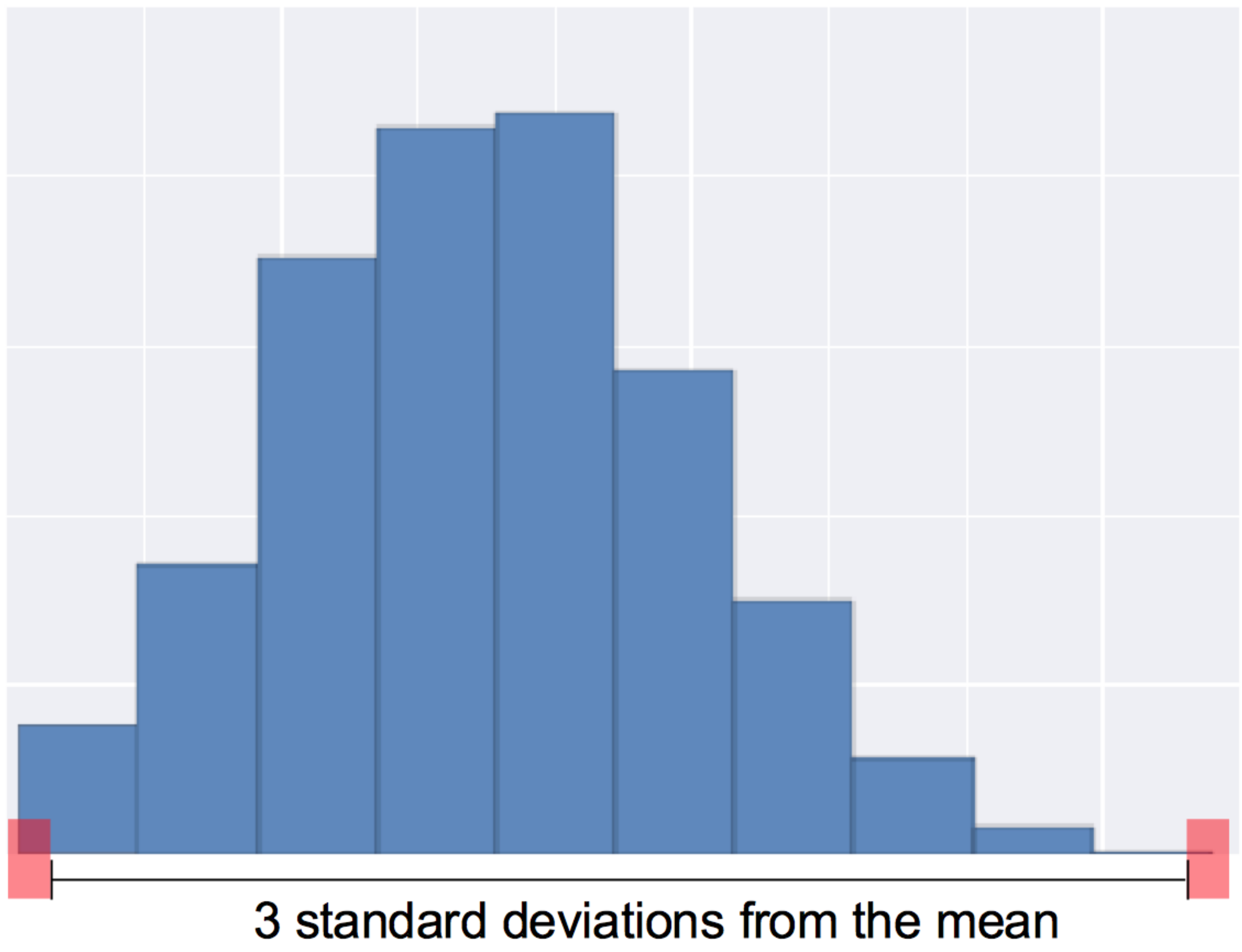
Quantiles in Python

```
q_cutoff = df['col_name'].quantile(0.95)

mask = df['col_name'] < q_cutoff

trimmed_df = df[mask]
```

Standard deviation based detection



Standard deviation detection in Python

```
mean = df['col_name'].mean()
std = df['col_name'].std()
cut_off = std * 3

lower, upper = mean - cut_off, mean + cut_off
new_df = df[(df['col_name'] < upper) &
                 (df['col_name'] > lower)]
```

Let's practice!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Scaling and transforming new data

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robet O'Callaghan
Director of Data Science, Ordergroove

Reuse training scalers

```
scaler = StandardScaler()

scaler.fit(train[['col']])

train['scaled_col'] = scaler.transform(train[['col']])

# FIT SOME MODEL
# ....

test = pd.read_csv('test_csv')

test['scaled_col'] = scaler.transform(test[['col']])
```

Training transformations for reuse

```
train_mean = train[['col']].mean()
train_std = train[['col']].std()

cut_off = train_std * 3
train_lower = train_mean - cut_off
train_upper = train_mean + cut_off

# Subset train data

test = pd.read_csv('test_csv')

# Subset test data
test = test[(test[['col']] < train_upper) &
              (test[['col']] > train_lower)]
```

Why only use training data?

Data leakage: Using data that you won't have access to when assessing the performance of your model

Avoid data leakage!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

datacamp

Introduction to Text Encoding

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

datacamp

Standardizing your text

Example of free text:

> Fellow-Citizens of the Senate and of the House of Representatives: AMONG the vicissitudes incident to life no event could have filled me with greater anxieties than that of which the notification was transmitted by your order, and received on the th day of the present month.

Dataset

```
print(speech_df.head())
```

```
                Name          Inaugural Address    \
0    George Washington     First Inaugural Address
1    George Washington    Second Inaugural Address
2    John Adams                  Inaugural Address
3    Thomas Jefferson      First Inaugural Address
4    Thomas Jefferson     Second Inaugural Address

                      Date                                  text
0    Thursday, April 30, 1789    Fellow-Citizens of the Sena...
1       Monday, March 4, 1793    Fellow Citizens: I AM again...
2     Saturday, March 4, 1797    WHEN it was first perceived...
3    Wednesday, March 4, 1801    Friends and Fellow-Citizens...
4       Monday, March 4, 1805    PROCEEDING, fellow-citizens...
```

Removing unwanted characters

- `[a-zA-Z]` : All letter characters
- `[^a-zA-Z]` : All non letter characters

```
speech_df['text'] = speech_df['text']\
                    .str.replace('[^a-zA-Z]', ' ')
```

Removing unwanted characters

Before:

```
"Fellow-Citizens of the Senate and of the House of
Representatives: AMONG the vicissitudes incident to
life no event could have filled me with greater" ...
```

After:

```
"Fellow Citizens of the Senate and of the House of
Representatives AMONG the vicissitudes incident to
life no event could have filled me with greater" ...
```

Standardize the case

```
speech_df['text'] = speech_df['text'].str.lower()
print(speech_df['text'][0])
```

```
"fellow citizens of the senate and of the house of
representatives among the vicissitudes incident to
life no event could have filled me with greater"...
```

Length of text

```
speech_df['char_cnt'] = speech_df['text'].str.len()
print(speech_df['char_cnt'].head())
```

```
0    1889
1     806
2    2408
3    1495
4    2465
Name: char_cnt, dtype: int64
```

Word counts

```
speech_df['word_cnt'] =
    speech_df['text'].str.split()
speech_df['word_cnt'].head(1)
```

```
['fellow', 'citizens', 'of', 'the', 'senate', 'and',...
```

Word counts

```
speech_df['word_counts'] =
    speech_df['text'].str.split().str.len()
print(speech_df['word_splits'].head())
```

```
0    1432
1     135
2    2323
3    1736
4    2169
Name: word_cnt, dtype: int64
```

Average length of word

```
speech_df['avg_word_len'] =
          speech_df['char_cnt'] / speech_df['word_cnt']
```

Let's practice!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Word Count Representation

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

datacamp

Text to columns

“citizens of the senate and of the house of representatives”

Index	citizens	of	the	senate	and	house	representatives
1	1	3	2	1	1	1	1

Initializing the vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
print(cv)
```

```
CountVectorizer(analyzer=u'word', binary=False,
        decode_error=u'strict',
        dtype=<type 'numpy.int64'>,
        encoding=u'utf-8', input=u'content',
        lowercase=True, max_df=1.0, max_features=None,
        min_df=1,ngram_range=(1, 1), preprocessor=None,
        stop_words=None, strip_accents=None,
        token_pattern=u'(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None
```

Specifying the vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0.1, max_df=0.9)
```

`min_df` : minimum fraction of documents the word must occur in

`max_df` : maximum fraction of documents the word can occur in

Fit the vectorizer

```
cv.fit(speech_df['text_clean'])
```

Transforming your text

```
cv_transformed = cv.transform(speech_df['text_clean'])
print(cv_transformed)
```

```
<58x8839 sparse matrix of type '<type 'numpy.int64'>'
```

Transforming your text

```
cv_transformed.toarray()
```

Getting the features

```
feature_names = cv.get_feature_names()
print(feature_names)
```

```
[u'abandon', u'abandoned', u'abandonment', u'abate',
u'abdicated', u'abeyance', u'abhorring', u'abide',
u'abiding', u'abilities', u'ability', u'abject'...
```

Fitting and transforming

```
cv_transformed = cv.fit_transform(speech_df['text_clean'])
print(cv_transformed)
```

```
<58x8839 sparse matrix of type '<type 'numpy.int64'>'
```

Putting it all together

```
cv_df = pd.DataFrame(cv_transformed.toarray(),
                     columns=cv.get_feature_names())\
                              .add_prefix('Counts_')
print(cv_df.head())
```

```
     Counts_aback    Counts_abandoned    Counts_a...
0               1                   0           ...
1               0                   0           ...
2               0                   1           ...
3               0                   1           ...
4               0                   0           ...
```

[1] ```out Counts_aback Counts_abandon Counts_abandonment 0 1 0 0 1 0 0 1 2 0 1 0 3 0 1 0 4 0 0 0 ```

Updating your DataFrame

```
speech_df = pd.concat([speech_df, cv_df],
                      axis=1, sort=False)
print(speech_df.shape)
```

```
(58, 8845)
```

Let's practice!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Tf-Idf Representation

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Introducing TF-IDF

```
print(speech_df['Counts_the'].head())
```

```
0    21
1    13
2    29
3    22
4    20
```

TF-IDF

$$\text{TF-IDF} = \frac{\dfrac{Count\ of\ word\ occurances}{Total\ words\ in\ document}}{\log\left(\dfrac{Number\ of\ docs\ word\ is\ in}{Total\ number\ of\ docs}\right)}$$

Importing the vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer()
print(tv)
```

```
TfidfVectorizer(analyzer=u'word', binary=False, decode_erro
        dtype=<type 'numpy.float64'>, encoding=u'utf-8', in
        lowercase=True, max_df=1.0, max_features=None, min_
        ngram_range=(1, 1), norm=u'l2', preprocessor=None,
        stop_words=None, strip_accents=None, sublinear_tf=F
        token_pattern=u'(?u)\\b\\w\\w+\\b', tokenizer=None,
        vocabulary=None)
```

Max features and stopwords

```
tv = TfidfVectorizer(max_features=100,
                     stop_words='english')
```

`max_features` : Maximum number of columns created from TF-IDF

`stop_words` : List of common words to omit e.g. "and", "the" etc.

Fitting your text

```
tv.fit(train_speech_df['text'])
train_tv_transformed = tv.transform(train_speech_df['text']
```

Putting it all together

```
train_tv_df = pd.DataFrame(train_tv_transformed.toarray(),
                           columns=tv.get_feature_names())\
                                 .add_prefix('TFIDF_')

train_speech_df = pd.concat([train_speech_df, train_tv_df],
                            axis=1, sort=False)
```

Inspecting your transforms

```
examine_row = train_tv_df.iloc[0]
```

```
print(examine_row.sort_values(ascending=False))
```

```
TFIDF_government    0.367430
TFIDF_public        0.333237
TFIDF_present       0.315182
TFIDF_duty          0.238637
TFIDF_citizens      0.229644
Name: 0, dtype: float64
```

Applying the vectorizer to new data

```
test_tv_transformed = tv.transform(test_df['text_clean'])

test_tv_df = pd.DataFrame(test_tv_transformed.toarray(),
                          columns=tv.get_feature_names())\
                            .add_prefix('TFIDF_')

test_speech_df = pd.concat([test_speech_df, test_tv_df],
                           axis=1, sort=False)
```

Let's practice!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Bag of words and N-grams

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Issues with bag of words

Positive meaning

Single word: *happy*

Negative meaning

Bi-gram : *not happy*

Positive meaning

Trigram : *never not happy*

Using N-grams

```
tv_bi_gram_vec = TfidfVectorizer(ngram_range = (2,2))

# Fit and apply bigram vectorizer
tv_bi_gram = tv_bi_gram_vec\
               .fit_transform(speech_df['text'])

# Print the bigram features
print(tv_bi_gram_vec.get_feature_names())
```

```
[u'american people', u'best ability ',
 u'beloved country', u'best interests' ... ]
```

Finding common words

```
# Create a DataFrame with the Counts features
tv_df = pd.DataFrame(tv_bi_gram.toarray(),
                     columns=tv_bi_gram_vec.get_feature_names())\
                        .add_prefix('Counts_')

tv_sums = tv_df.sum()
print(tv_sums.head())
```

```
Counts_administration government    12
Counts_almighty god                 15
Counts_american people              36
Counts_beloved country               8
Counts_best ability                  8
dtype: int64
```

Finding common words

```
print(tv_sums.sort_values(ascending=False)).head()
```

```
Counts_united states         152
Counts_fellow citizens        97
Counts_american people        36
Counts_federal government     35
Counts_self government        30
dtype: int64
```

Let's practice!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Wrap-up

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

datacamp

Chapter 1

- How to understand your data types
- Efficient encoding or categorical features
- Different ways to work with continuous variables

Chapter 2

- How to locate gaps in your data
- Best practices in dealing with the incomplete rows
- Methods to find and deal with unwanted characters

Chapter 3

- How to observe your data's distribution
- Why and how to modify this distribution
- Best practices of finding outliers and their removal

Chapter 4

- The foundations of word embeddings
- Usage of Term Frequency Inverse Document Frequency (Tf-idf)
- N-grams and its advantages over bag of words

Next steps

- Kaggle competitions
- More DataCamp courses
- Your own project

Thank You!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON