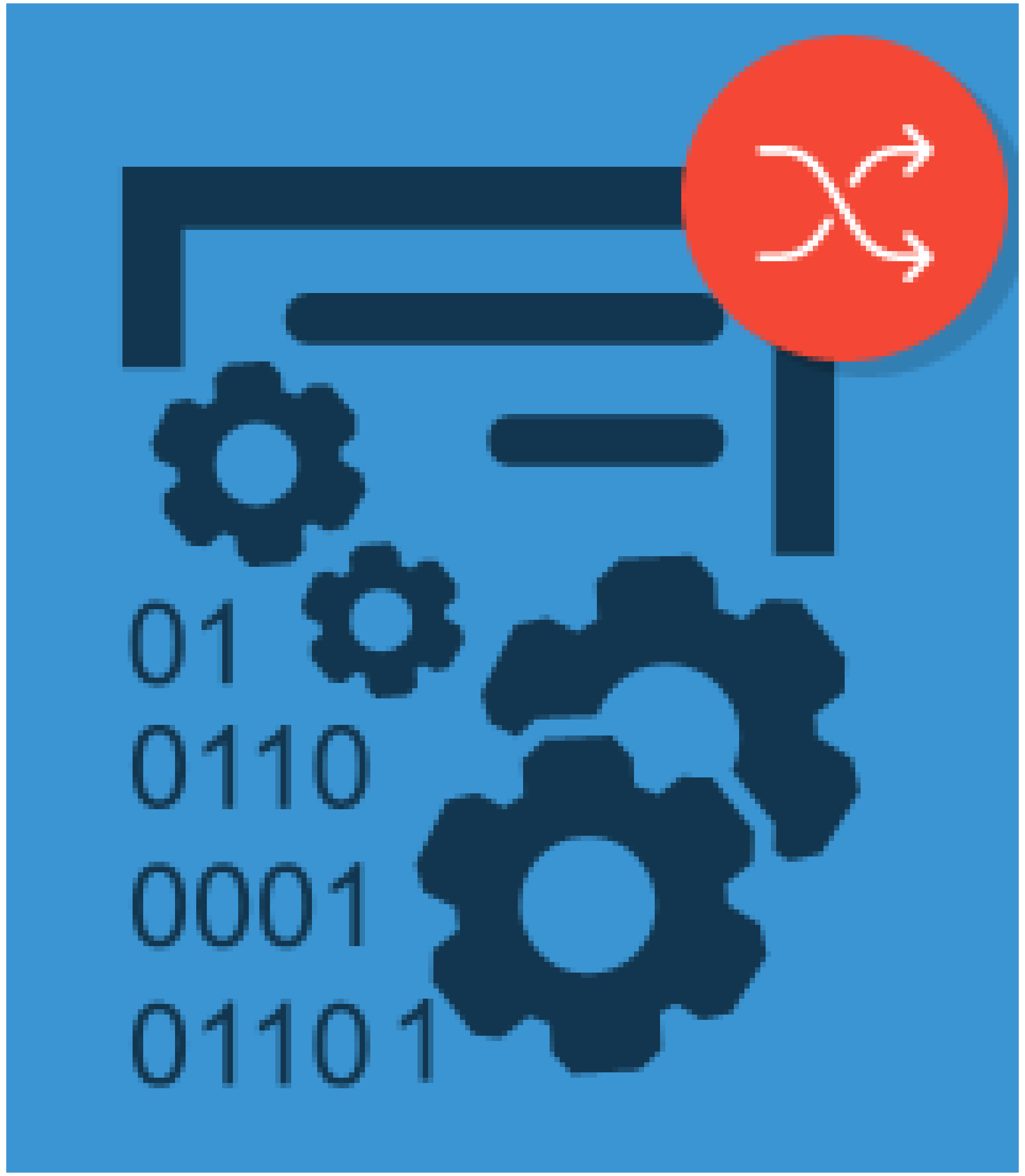
Feature Generation

FEATURE ENGINEERING WITH PYSPARK

John Hogue

Lead Data Scientist

Why generate new features?



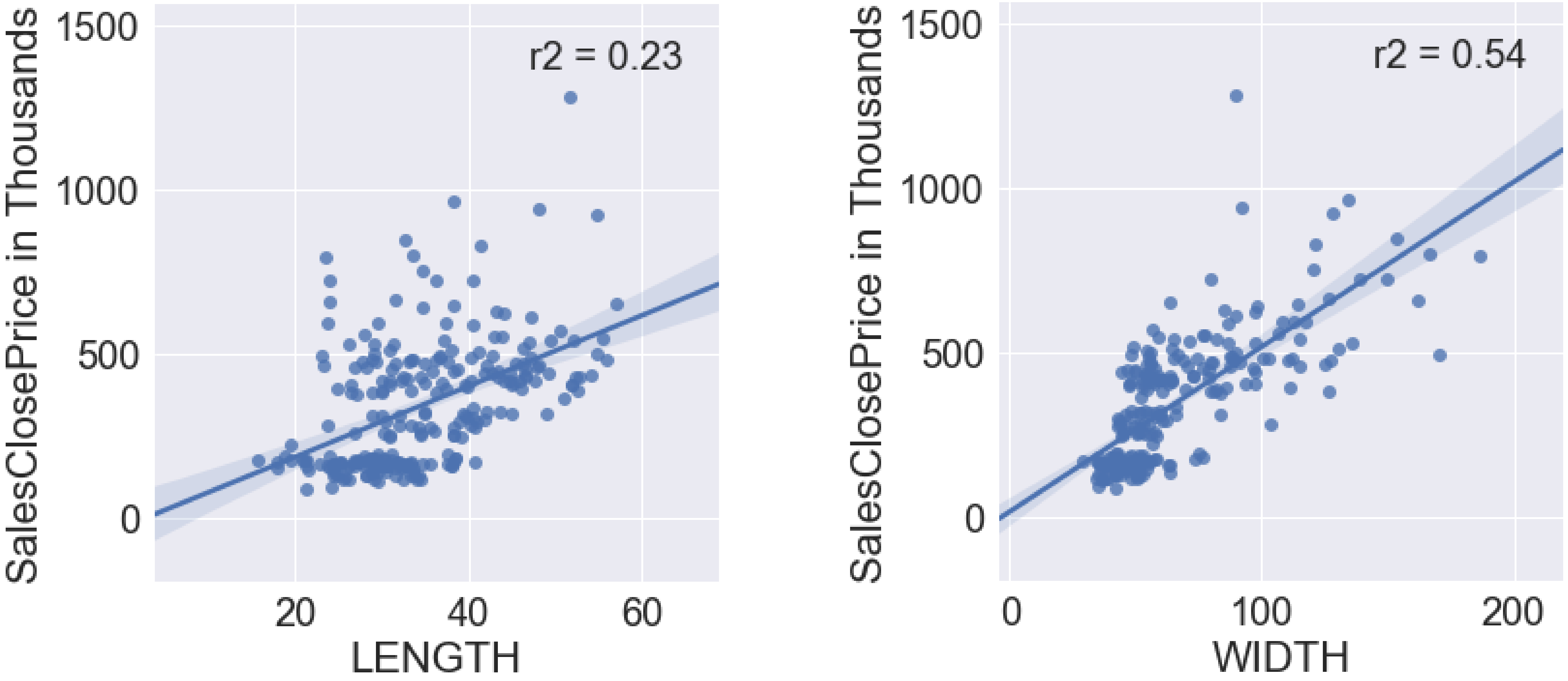
Multiplying

Summing

Differencing

Dividing

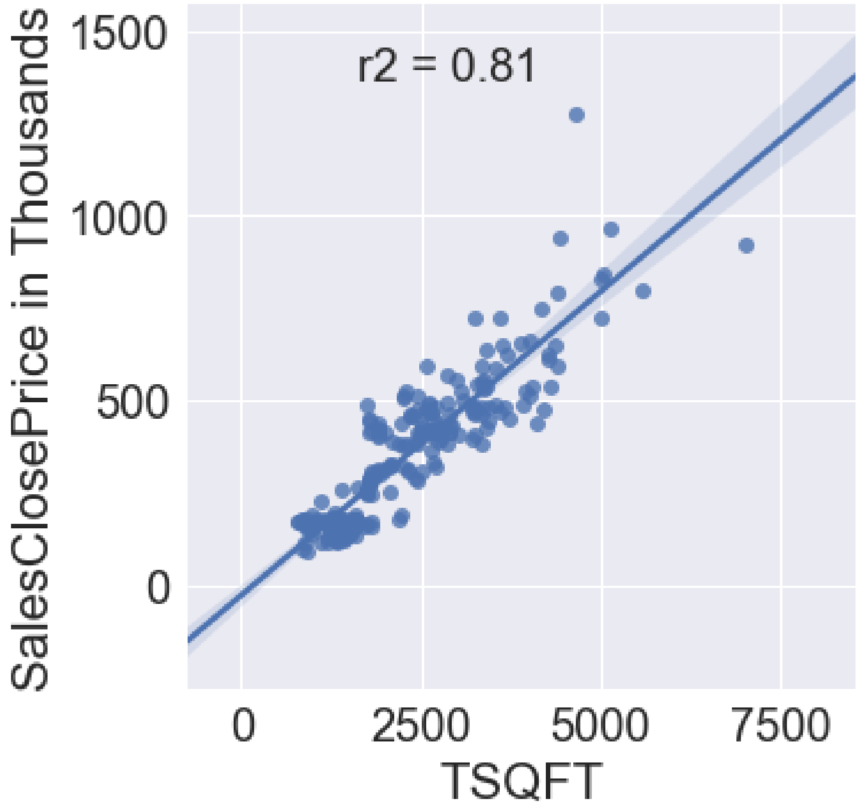
Why generate new features?



Combining Two Features

Multiplication

```
# Creating a new feature, area by multiplying
df = df.withColumn('TSQFT', (df['WIDTH'] * df['LENGTH']))
```



Other Ways to Combine Two Features

```
# Sum two columns
df = df.withColumn('TSQFT', (df['SQFTBELOWGROUND'] + df['SQFTABOVEGROUND']))
```

```
# Divide two columns
df = df.withColumn('PRICEPERTSQFT', (df['LISTPRICE'] / df['TSQFT']))
```

```
# Difference two columns
df = df.withColumn('DAYSONMARKET', datediff('OFFMARKETDATE', 'LISTDATE'))
```

What's the limit?

Automation of Features

- FeatureTools & TSFresh
- Explosion of Features
- Higher Order & Beyond?

Go forth and combine!

FEATURE ENGINEERING WITH PYSPARK

Time Features

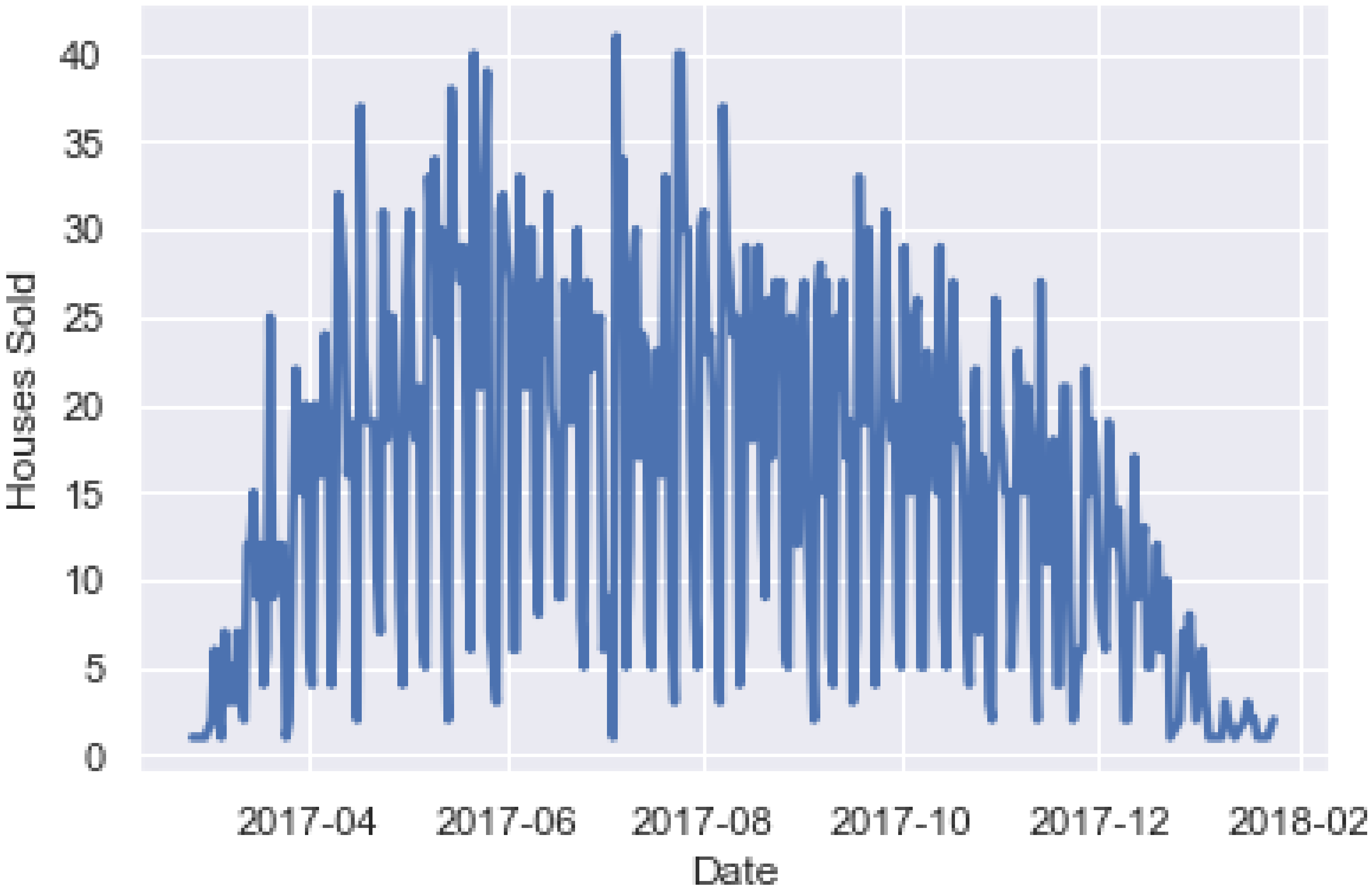
FEATURE ENGINEERING WITH PYSPARK

John Hogue

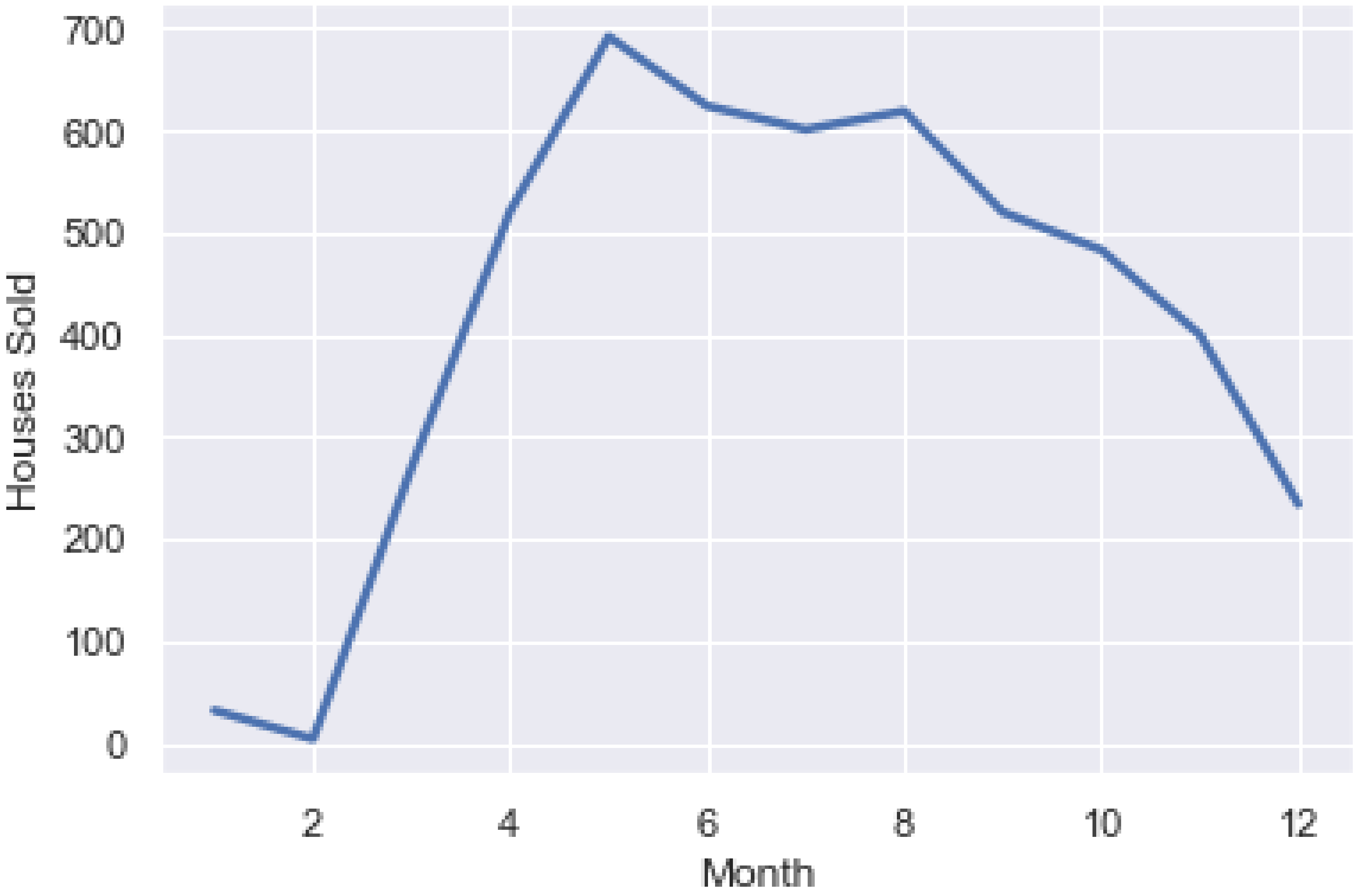
Lead Data Scientist, General Mills

The Cyclical Nature of Things

Choosing the Right Level



Choosing the Right Level



Treating Date Fields as Dates...

```python
from pyspark.sql.functions import to_date

# Cast the data type to Date
df = df.withColumn('LISTDATE', to_date('LISTDATE'))
```

```python
# Inspect the field
df[['LISTDATE']].show(2)
```

```
+----------+
|  LISTDATE|
+----------+
|2017-07-14|
|2017-10-08|
+----------+
only showing top 2 rows
```

Time Components

```
from pyspark.sql.functions import year, month

# Create a new column of year number
df = df.withColumn('LIST_YEAR', year('LISTDATE'))
```

```
# Create a new column of month number
df = df.withColumn('LIST_MONTH', month('LISTDATE'))
```

```
from pyspark.sql.functions import dayofmonth, weekofyear

# Create new columns of the day number within the month
df = df.withColumn('LIST_DAYOFMONTH', dayofmonth('LISTDATE'))
```

```
# Create new columns of the week number within the year
df = df.withColumn('LIST_WEEKOFYEAR', weekofyear('LISTDATE'))
```

Basic Time Based Metrics

```
from pyspark.sql.functions import datediff

# Calculate difference between two date fields
df.withColumn('DAYSONMARKET', datediff('OFFMARKETDATE', 'LISTDATE'))
```

Lagging Features

`window()`

Returns a record based off a group of records

`lag(col, count=1)`

Returns the value that is offset by rows before the current row

Lagging Features, the PySpark Way

```
from pyspark.sql.functions import lag
from pyspark.sql.window import Window
# Create Window
w = Window().orderBy(m_df['DATE'])
# Create lagged column
m_df = m_df.withColumn('MORTGAGE-1wk', lag('MORTGAGE', count=1).over(w))
# Inspect results
m_df.show(3)
```

```
+----------+------------+----------------+
|      DATE|    MORTGAGE|    MORTGAGE-1wk|
+----------+------------+----------------+
|2013-10-10|        4.23|            null|
|2013-10-17|        4.28|            4.23|
|2013-10-24|        4.13|            4.28|
+----------+------------+----------------+
only showing top 3 rows
```

It's TIME to practice!

FEATURE ENGINEERING WITH PYSPARK

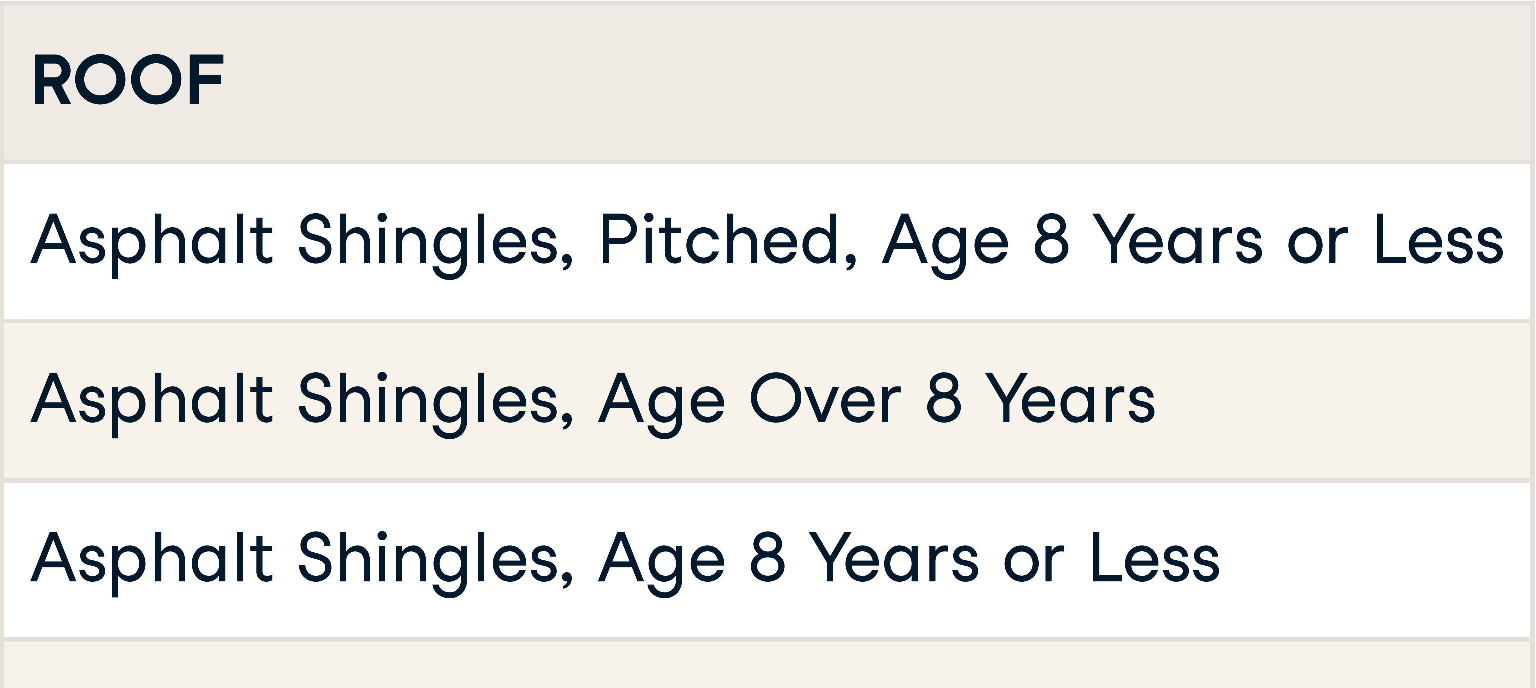
Extracting Features

FEATURE ENGINEERING WITH PYSPARK

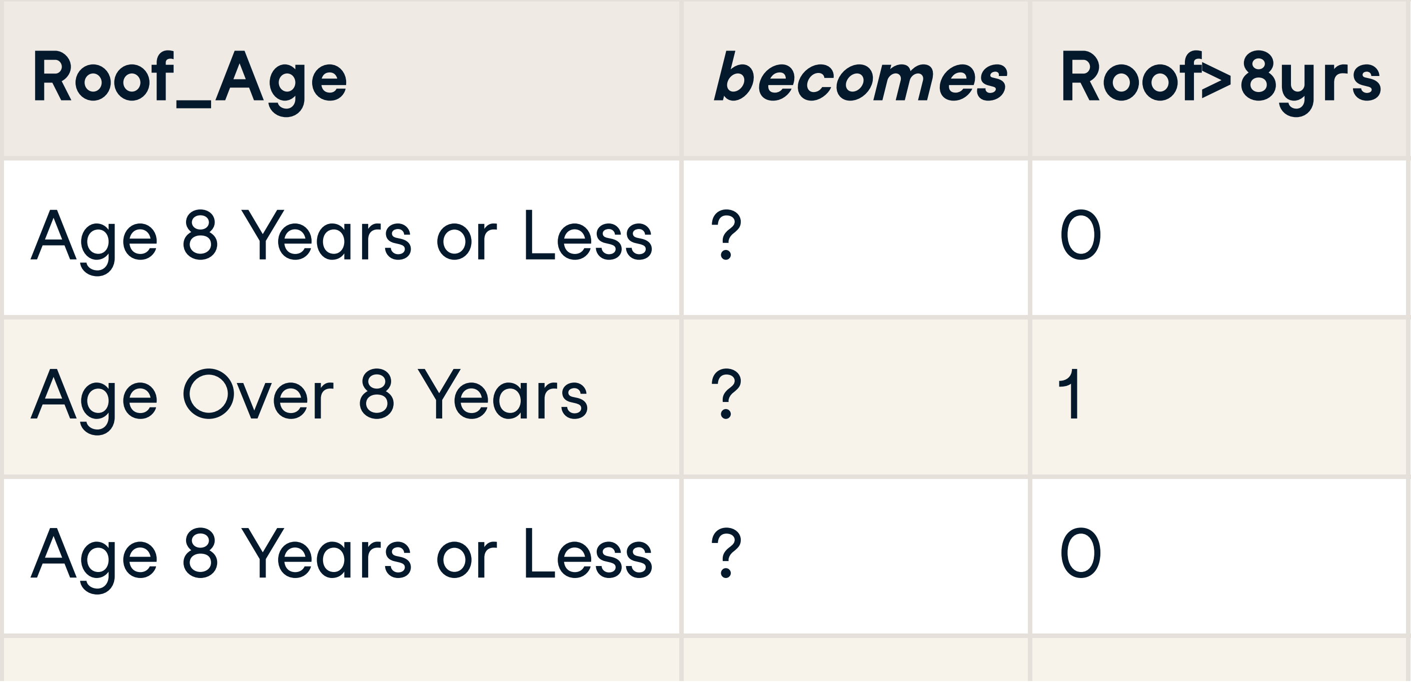
John Hogue

Lead Data Scientist, General Mills

Extracting Age with Text Match



ROOF
Asphalt Shingles, Pitched, Age 8 Years or Less
Asphalt Shingles, Age Over 8 Years
Asphalt Shingles, Age 8 Years or Less



Roof_Age	*becomes*	Roof>8yrs
Age 8 Years or Less	?	0
Age Over 8 Years	?	1
Age 8 Years or Less	?	0

Extracting Age with Text Match

```
from pyspark.sql.functions import when
# Create boolean filters
find_under_8 = df['ROOF'].like('%Age 8 Years or Less%')
find_over_8 = df['ROOF'].like('%Age Over 8 Years%')
# Apply filters using when() and otherwise()
df = df.withColumn('old_roof', (when(find_over_8, 1)
                                .when(find_under_8, 0)
                                .otherwise(None)))
# Inspect results
df[['ROOF', 'old_roof']].show(3, truncate=100)
```

```
+----------------------------------------------+--------+
|                                          ROOF|old_roof|
+----------------------------------------------+--------+
|                                          null|    null|
|Asphalt Shingles, Pitched, Age 8 Years or Less|       0|
|            Asphalt Shingles, Age Over 8 Years|       1|
+----------------------------------------------+--------+
only showing top 3 rows
```

Splitting Columns

ROOF	*becomes*	Roof_Material
Asphalt Shingles, Pitched, Age 8 Years or Less	?	Asphalt Shingles
Null	?	
Asphalt Shingles, Age Over 8 Years	?	Asphalt Shingles
Metal, Age 8 Years or Less	?	Metal
Tile, Age 8 Years or Less	?	Tile
Asphalt Shingles	?	Asphalt Shingles

Splitting Columns

```python
from pyspark.sql.functions import split
# Split the column on commas into a list
split_col = split(df['ROOF'], ',')
# Put the first value of the list into a new column
df = df.withColumn('Roof_Material', split_col.getItem(0))
# Inspect results
df[['ROOF', 'Roof_Material']].show(5, truncate=100)
```

```
+----------------------------------------------+----------------+
|                                          ROOF|   Roof_Material|
+----------------------------------------------+----------------+
|                                          null|            null|
|Asphalt Shingles, Pitched, Age 8 Years or Less|Asphalt Shingles|
|                                          null|            null|
|Asphalt Shingles, Pitched, Age 8 Years or Less|Asphalt Shingles|
|            Asphalt Shingles, Age Over 8 Years|Asphalt Shingles|
+----------------------------------------------+----------------+
only showing top 5 rows
```

Explode!

Starting Record

NO	roof_list
2	[Asphalt Shingles, Pitched, Age 8 Years or Less]

Exploded Record

NO	ex_roof_list
2	Asphalt Shingles
2	Pitched
2	Age 8 Years or Less

Pivot!

Exploded Record

NO	ex_roof_list
2	Asphalt Shingles
2	Pitched
2	Age 8 Years or Less

Pivoted Record

NO	Age 8 Years or Less	Age Over 8 Years	Asphalt Shingles	Flat	Metal	Other	Pitched	...
2	0	1	1	0	0	0	1	...

Explode & Pivot!

```python
from pyspark.sql.functions import split, explode, lit, coalesce, first
```

```python
# Split the column on commas into a list
df = df.withColumn('roof_list', split(df['ROOF'], ', '))
```

```python
# Explode list into new records for each value
ex_df = df.withColumn('ex_roof_list', explode(df['roof_list']))
```

```python
# Create a dummy column of constant value
ex_df = ex_df.withColumn('constant_val', lit(1))
```

```python
# Pivot the values into boolean columns
piv_df = ex_df.groupBy('NO').pivot('ex_roof_list')\
  .agg(coalesce(first('constant_val')))
```

Let's wrangle some features!

FEATURE ENGINEERING WITH PYSPARK

Binarizing, Bucketing & Encoding

FEATURE ENGINEERING WITH PYSPARK

John Hogue
Lead Data Scientist

Binarizing

FIREPLACES	*becomes*	Has_Fireplace
1	?	1
3	?	1
1	?	1
2	?	1
0	?	0

Binarizing

```
from pyspark.ml.feature import Binarizer
# Cast the data type to double
df = df.withColumn('FIREPLACES', df['FIREPLACES'].cast('double'))
# Create binarizing transformer
bin = Binarizer(threshold=0.0, inputCol='FIREPLACES', outputCol='FireplaceT')
# Apply the transformer
df = bin.transform(df)
# Inspect the results
df[['FIREPLACES','FireplaceT']].show(3)
```

```
+----------+-------------+
|FIREPLACES|   FireplaceT|
+----------+-------------+
|       0.0|          0.0|
|       1.0|          1.0|
|       2.0|          1.0|
+----------+-------------+
only showing top 3 rows
```

Bucketing

```python
from pyspark.ml.feature import Bucketizer
# Define how to split data
splits = [0, 1, 2, 3, 4, float('Inf')]
# Create bucketing transformer
buck = Bucketizer(splits=splits, inputCol='BATHSTOTAL', outputCol='baths')
# Apply transformer
df = buck.transform(df)
# Inspect results
df[['BATHSTOTAL', 'baths']].show(4)
```

```
+----------+-----------------+
|BATHSTOTAL|baths            |
+----------+-----------------+
|         2|              2.0|
|         3|              3.0|
|         1|              1.0|
|         5|              4.0|
+----------+-----------------+
only showing top 4 rows
```

One Hot Encoding

CITY	*becomes*	LELM	MAPW	OAKD	STP	WB
LELM - Lake Elmo	?	1	0	0	0	0
MAPW - Maplewood	?	0	1	0	0	0
OAKD - Oakdale	?	0	0	1	0	0
STP - Saint Paul	?	0	0	0	1	0
WB - Woodbury	?	0	0	0	0	1

One Hot Encoding the PySpark Way

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer
```

```
# Create indexer transformer
stringIndexer = StringIndexer(inputCol='CITY', outputCol='City_Index')
```

```
# Fit transformer
model = stringIndexer.fit(df)
# Apply transformer
indexed = model.transform(df)
```

One Hot Encoding the PySpark Way

```
# Create encoder transformer
encoder = OneHotEncoder(inputCol='City_Index', outputCol='City_Vec)
```

```
# Apply the encoder transformer
encoded_df = encoder.transform(indexed)
```

```
# Inspect results
encoded_df[['City_Vec']].show(4)
```

```
+-------------+
|     City_Vec|
+-------------+
|    (4,[],[])|
|    (4,[],[])|
|(4,[2],[1.0])|
|(4,[2],[1.0])|
+-------------+
only showing top 4 rows
```

Get Transforming!

FEATURE ENGINEERING WITH PYSPARK