Dropping Data

FEATURE ENGINEERING WITH PYSPARK

John Hogue
Lead Data Scientist, General Mills

Where can data go bad?

- Recorded wrong
- Unique events
- Formatted incorrectly
- Duplications
- Missing
- Not relevant

Dropping Columns

```
df.select(['NO', 'UNITNUMBER', 'CLASS']).show()
```

```
+----+----------+-----+
|  NO|UNITNUMBER|CLASS|
+----+----------+-----+
|   1|      null|   SF|
| 156|        A8|   SF|
| 157|       207|   SF|
| 158|      701G|   SF|
| 159|        36|   SF|
```

Multiple fields are not needed for our analysis

- `'NO'` auto-generated record number
- `'UNITNUMBER'` irrelevant data
- `'CLASS'` all constant

Dropping Columns

`drop(*cols)`

- `*cols` – a column name to drop or a list of column names to drop.
- Returns a new DataFrame that drops the specified

```
# List of columns to drop
cols_to_drop = ['NO', 'UNITNUMBER', 'CLASS']
# Drop the columns
df = df.drop(*cols_to_drop)
```

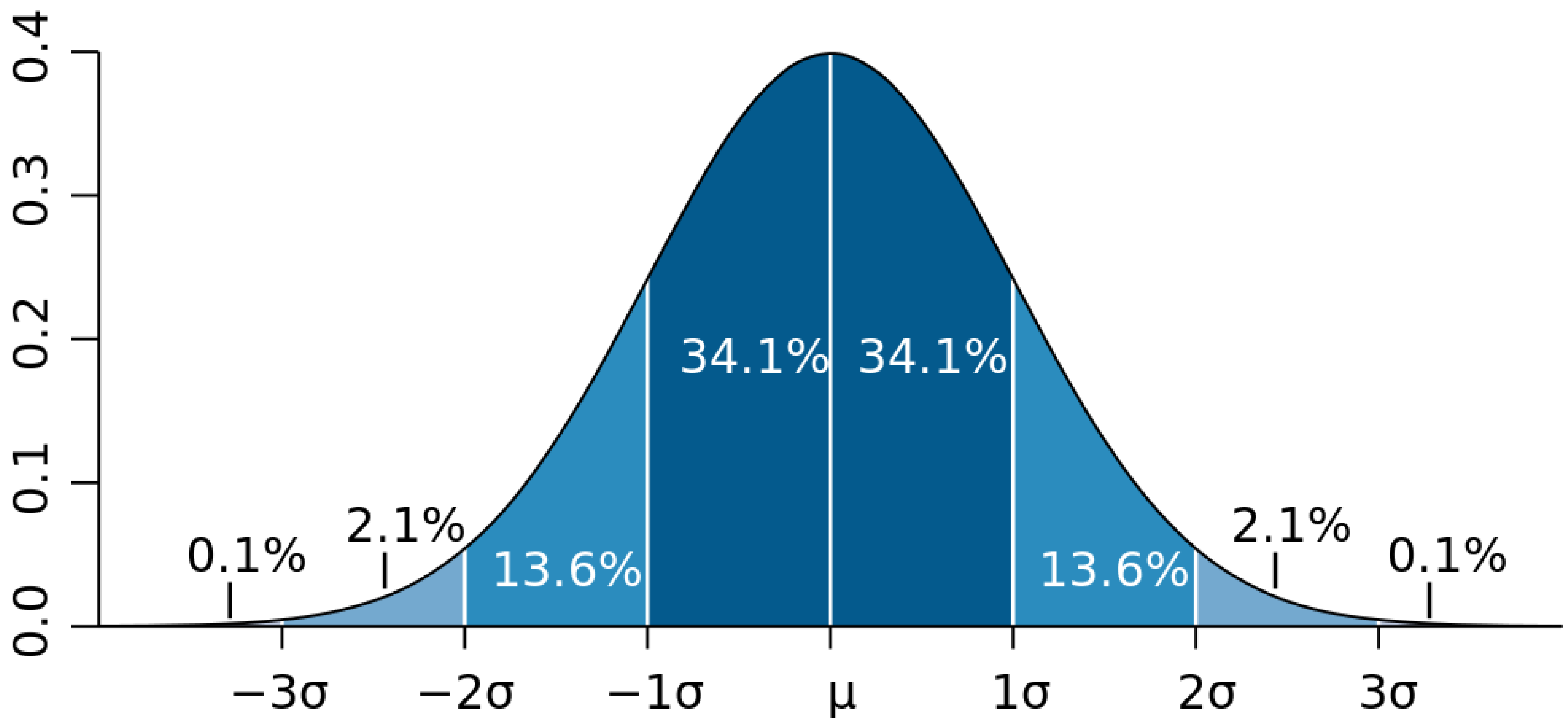
Text Filtering

- `where(condition)`
  - condition – a Column of `types.BooleanType` or a string of SQL expression.
  - Filters dataframe where the condition is true
- `like(other)`
  - other – a SQL LIKE pattern
  - Returns a boolean Column
- `~`
  - The NOT condition

```
df = df.where(~df['POTENTIALSHORTSALE'].like('Not Disclosed'))
```

Outlier Filtering

Filter data to within three standard deviations (3?) of the mean (?)



Value Filtering Example

```
# Calculate values used for filtering
std_val = df.agg({'SALESCLOSEPRICE': 'stddev'}).collect()[0][0]
mean_val = df.agg({'SALESCLOSEPRICE': 'mean'}).collect()[0][0]
# Create three standard deviation (? ± 3?) upper and lower bounds for data
hi_bound = mean_val + (3 * std_val)
low_bound = mean_val - (3 * std_val)
# Use where() to filter the DataFrame between values
df = df.where((df['LISTPRICE'] < hi_bound) & (df['LISTPRICE'] > low_bound))
```

Dropping NA's or NULLs

`DataFrame.dropna()`

- `how` : ‘any’ or ‘all’. If ‘any’, drop a record if it contains any nulls. If ‘all’, drop a record only if all its values are null.
- `thresh` : int, default None If specified, drop records that have less than thresh non-null values. This overwrites the how parameter.
- `subset` : optional list of column names to consider.

Dropping NA's or NULLs

```
# Drop any records with NULL values
df = df.dropna()
# drop records if both LISTPRICE and SALESCLOSEPRICE are NULL
df = df.dropna(how='all', subset['LISTPRICE', 'SALESCLOSEPRICE '])
# Drop records where at least two columns have NULL values
df = df.dropna(thresh=2)
```

Dropping Duplicates

What is a duplicate?

- Two or more records contains all the same information
- After dropping columns or joining datasets, check for duplicates

`dropDuplicates()`

- Can be run across entire DataFrame or a list of columns
- In PySpark there is no order for which record is removed

```
# Entire DataFrame
df.dropDuplicates()
# Check only a column list
df.dropDuplicates(['streetaddress'])
```

Let's practice!

FEATURE ENGINEERING WITH PYSPARK

Adjusting Data

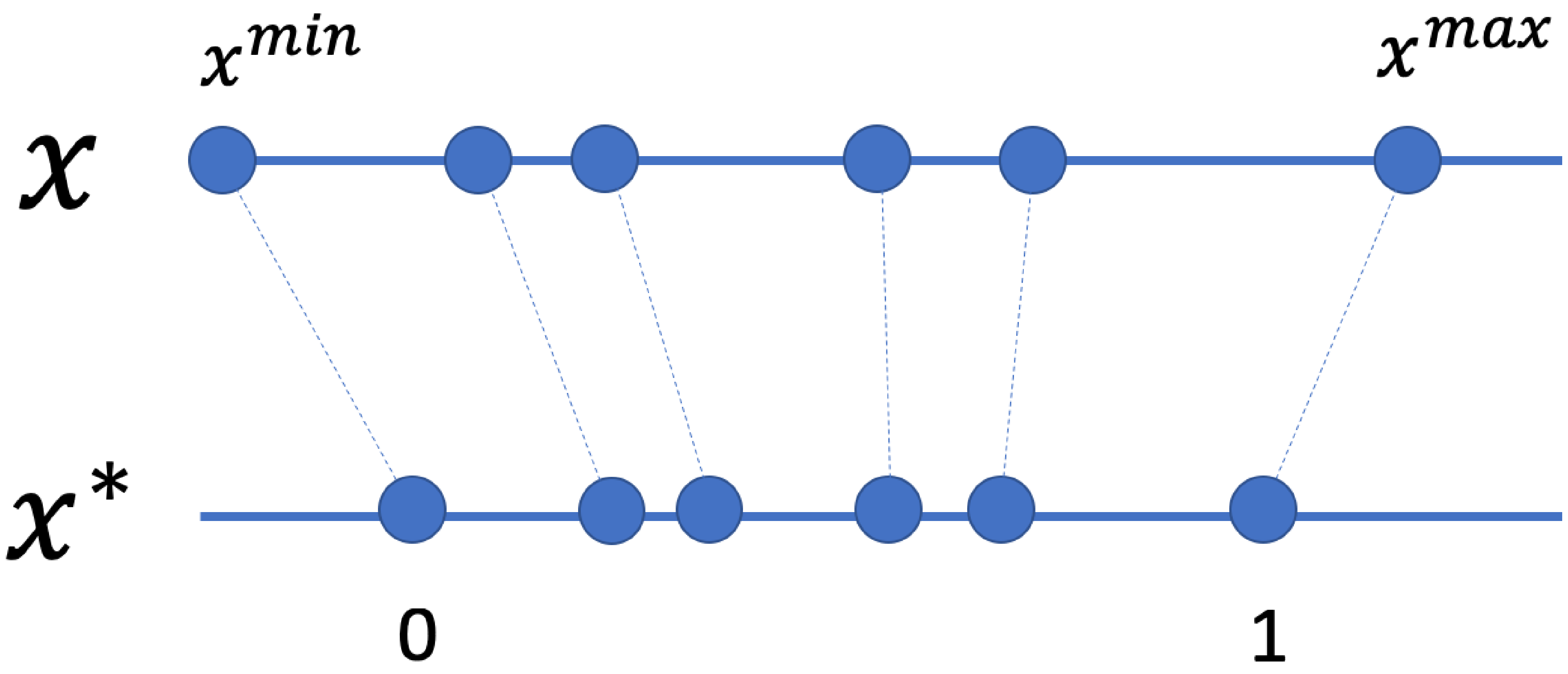
FEATURE ENGINEERING WITH PYSPARK

John Hogue
Lead Data Scientist, General Mills

Why Transform Data?

What is MinMax Scaling

$$x^*_{i,j} = \frac{x_{i,j} - x_j^{min}}{x_j^{max} - x_j^{min}}$$



Minmax Scaling

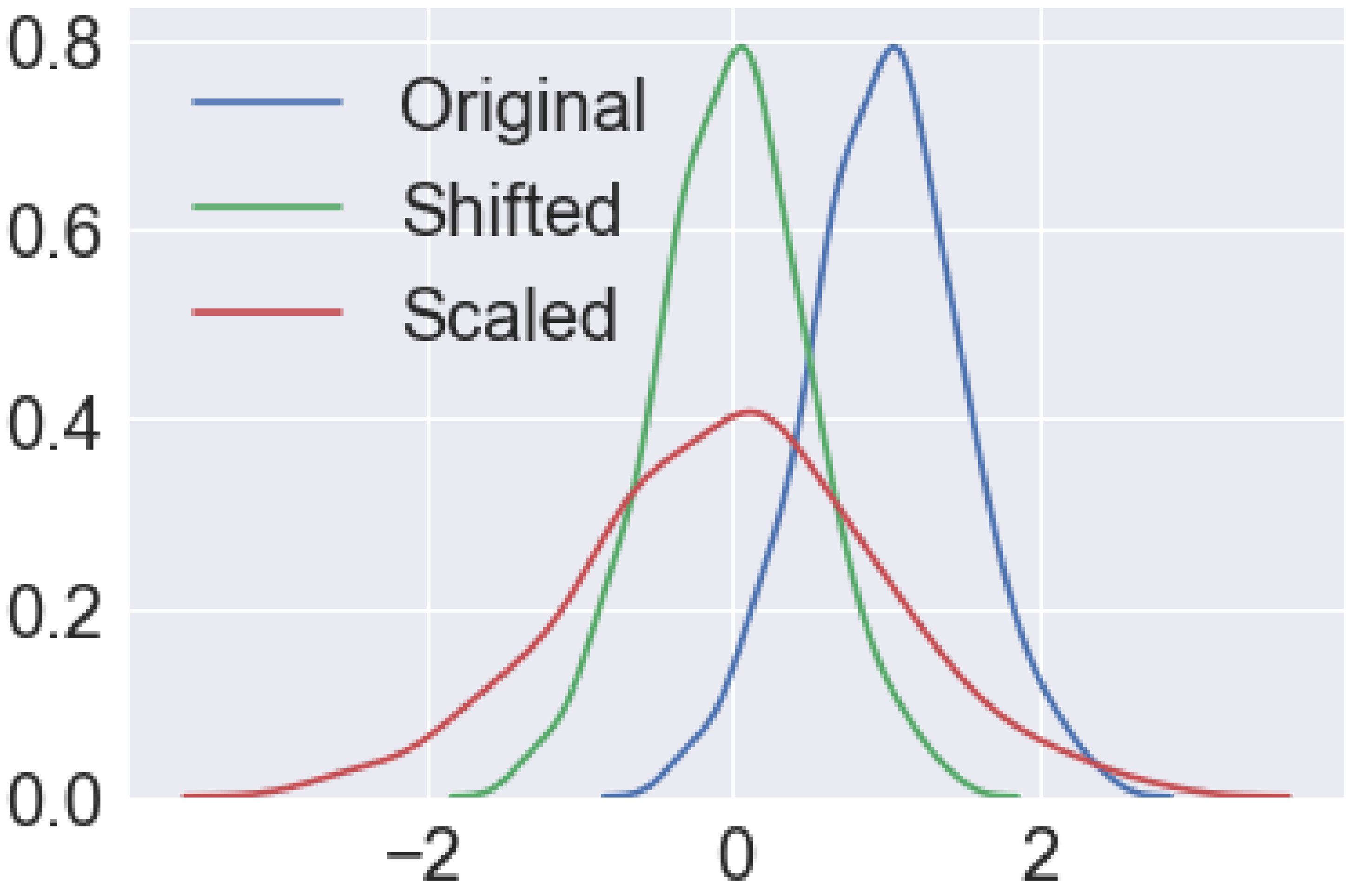
```
# define min and max values and collect them
max_days = df.agg({'DAYSONMARKET': 'max'}).collect()[0][0]
min_days = df.agg({'DAYSONMARKET': 'min'}).collect()[0][0]
# create a new column based off the scaled data
df = df.withColumn("scaled_days",
                   (df['DAYSONMARKET'] - min_days) / (max_days - min_days))
df[['scaled_days']].show(5)
```

```
+--------------------+
|         scaled_days|
+--------------------+
|0.044444444444444446|
|0.017777777777777778|
| 0.12444444444444444|
| 0.08444444444444445|
| 0.09333333333333334|
+--------------------+
only showing top 5 rows
```

What is Standardization?

Transform data to standard normal distribution

- z = (x - ?)/ ?
- Mean, ? of 0
- Standard Deviation, ? of 1



Standardization

```
mean_days = df.agg({'DAYSONMARKET': 'mean'}).collect()[0][0]
stddev_days = df.agg({'DAYSONMARKET': 'stddev'}).collect()[0][0]
# Create a new column with the scaled data
df = df.withColumn("ztrans_days",
                  (df['DAYSONMARKET'] - mean_days) / stddev_days)
df.agg({'ztrans_days': 'mean'}).collect()
```

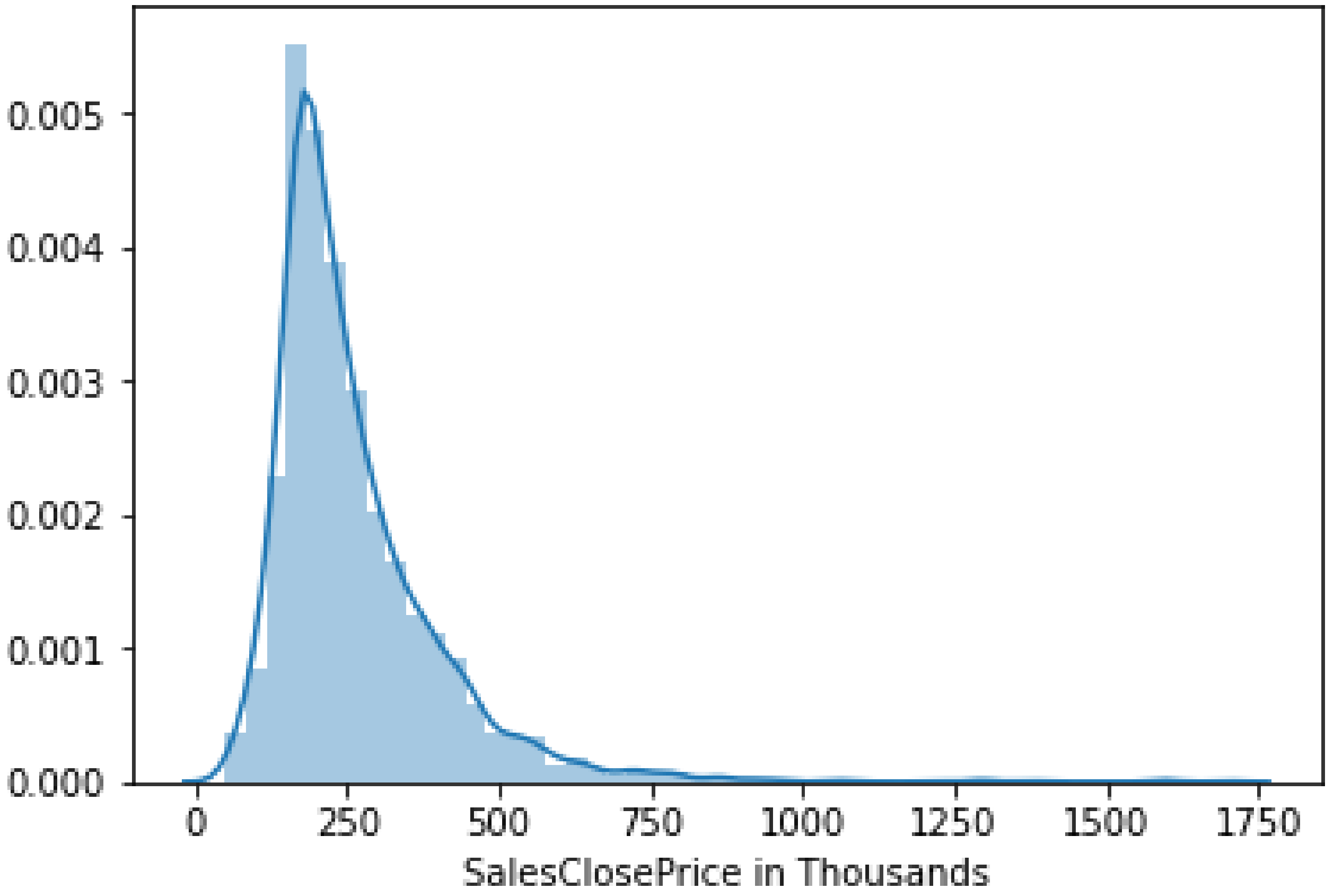
```
[Row(avg(ztrans_days)=-3.6568525985103407e-16)]
```

```
df.agg({'ztrans_days': 'stddev'}).collect()
```

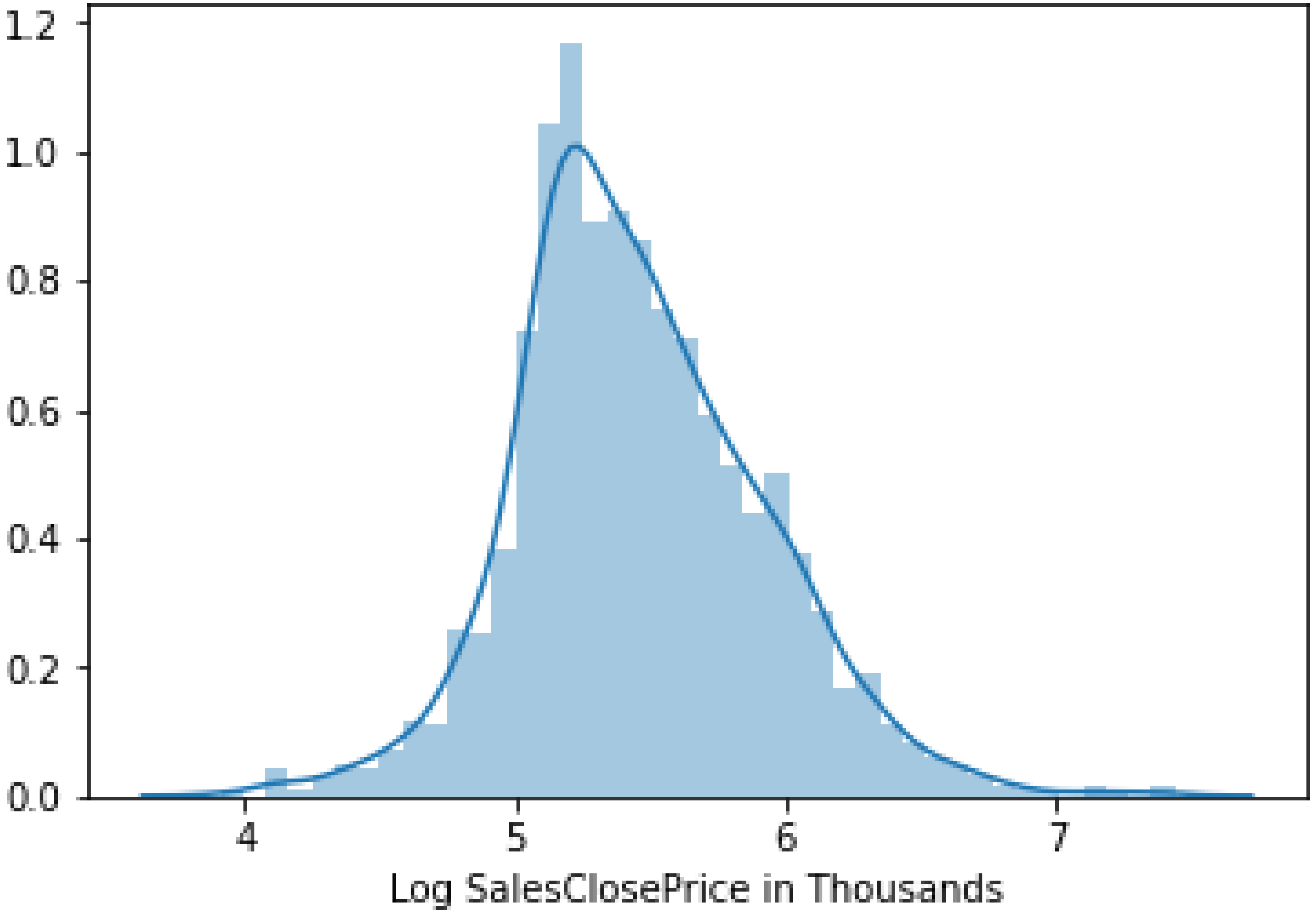
```
[Row(stddev(ztrans_days)=1.0000000000000009)]
```

What is Log Scaling

Unscaled distribution



Log-scaled distribution



Log Scaling

```
# import the log function
from pyspark.sql.functions import log
```

```
# Recalculate log of SALESCLOSEPRICE
df = df.withColumn('log_SalesClosePrice', log(df['SALESCLOSEPRICE']))
```

Let's practice!

FEATURE ENGINEERING WITH PYSPARK

Working with Missing Data

FEATURE ENGINEERING WITH PYSPARK

John Hogue
Lead Data Scientist, General Mills

How does data go missing in the digital age?

Data Collection

- Broken Sensors

Data Storage Rules

- 2017-01-01 vs January 1st, 2017

Joining Disparate Data

- Monthly to Weekly

Intentionally Missing

- Privacy Concerns

Types of Missing

Missing completely at random

- Missing Data is just a completely random subset

Missing at random

- Missing conditionally at random based on another observation

Missing not at random

- Data is missing because of how it is collected

Assessing Missing Values

When to drop rows with missing data?

- Missing values are rare
- Missing Completely at Random

`isNull()`

- True if the current expression is null.

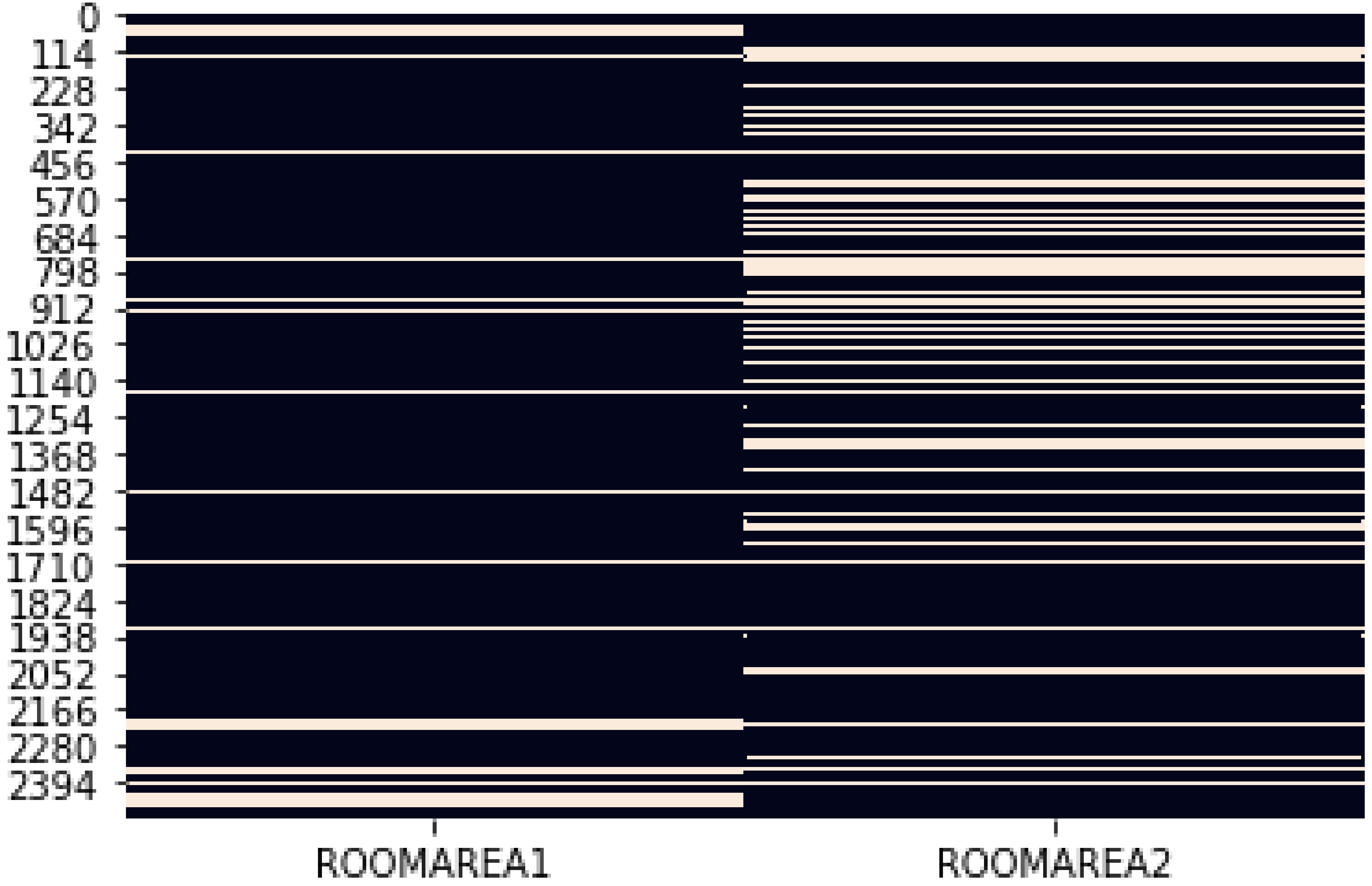
```
df.where(df['ROOF'].isNull()).count()
```

```
765
```

Plotting Missing Values

```
# Import library
import seaborn as sns
# subset the dataframe
sub_df = df.select(['ROOMAREA1'])
# sample the dataframe
sample_df = sub_df.sample(False, .5, 4)
# Convert to Pandas DataFrame
pandas_df = sample_df.toPandas()
# Plot it
sns.heatmap(data=pandas_df.isnull())
```

Missing Values Heatmap



Imputation of Missing Values

Process of replacing missing values

Rule Based

- Value based on business logic

Statistics Based

- Using mean, median, etc

Model Based

- Use model to predict value

Imputation of Missing Values

** `fillna(value, subset=None)`

- `value` the value to replace missings with
- `subset` the list of column names to replace missings

```
# Replacing missing values with zero
df.fillna(0, subset=['DAYSONMARKET'])
```

```
# Replacing with the mean value for that column
col_mean = df.agg({'DAYSONMARKET': 'mean'}).collect()[0][0]
df.fillna(col_mean, subset=['DAYSONMARKET'])
```

Let's practice!

FEATURE ENGINEERING WITH PYSPARK

Getting More Data

FEATURE ENGINEERING WITH PYSPARK

John Hogue
Lead Data Scientist, General Mills

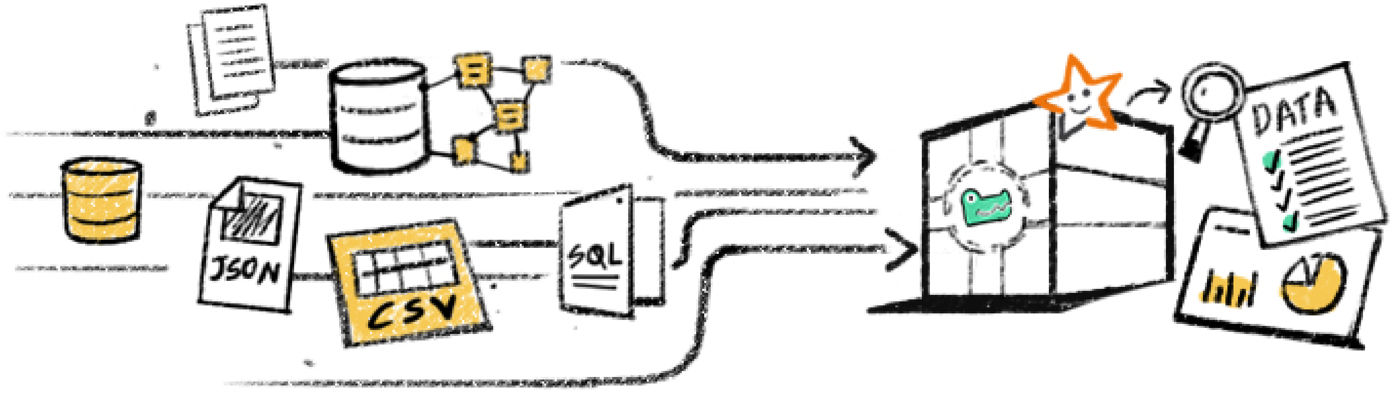
Thoughts on External Data Sets

PROS

- Add important predictors
- Supplement/replace values
- Cheap or easy to obtain

CONS

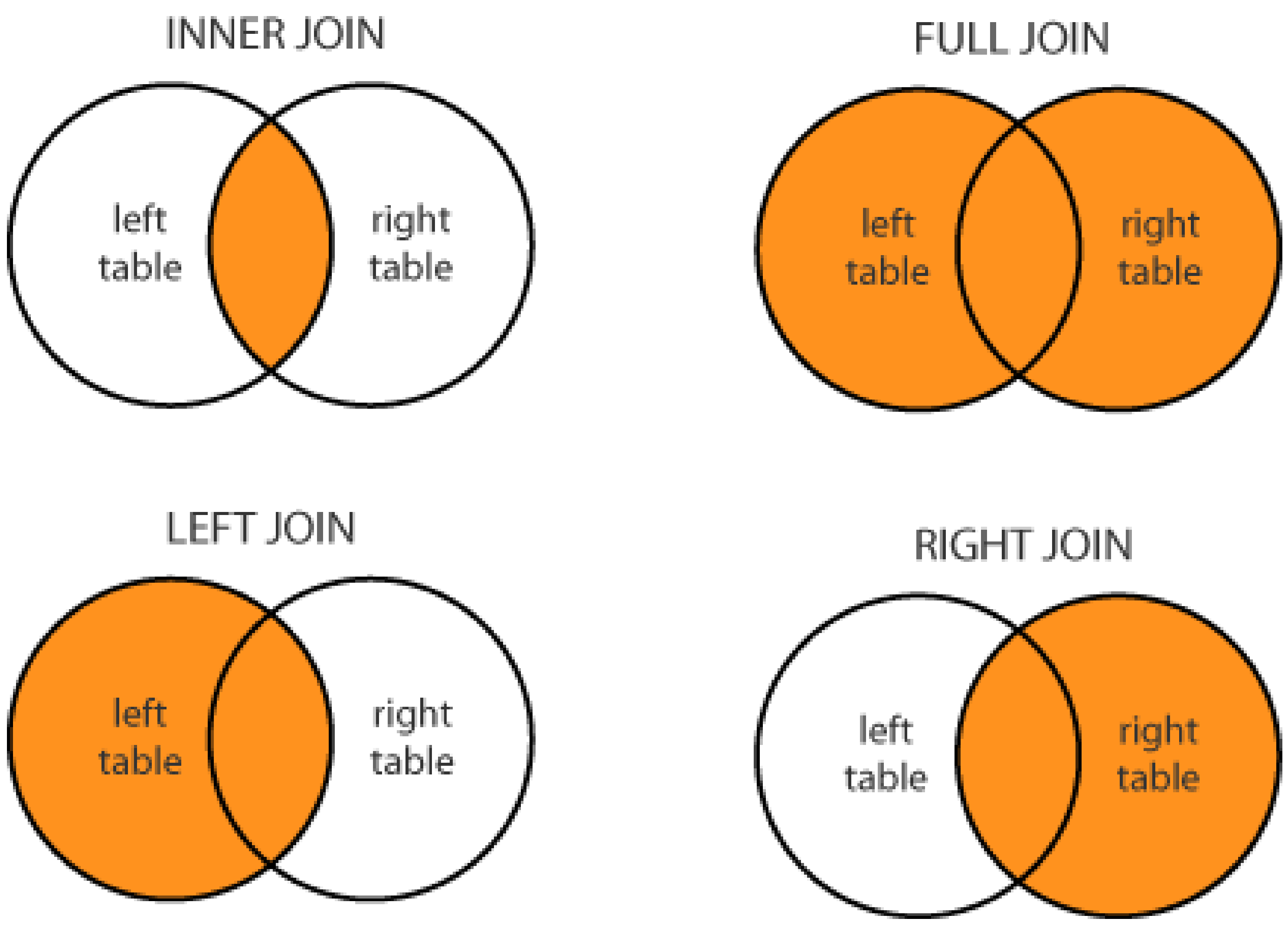
- May 'bog' analysis down
- Easy to induce data leakage
- Become data set subject matter expert



About Joins

Orienting our data directions

- Left; our starting data set
- Right; new data set to incorporate



PySpark DataFrame Joins

```
DataFrame.join(
    other,          # Other DataFrame to merge
    on=None,        # The keys to join on
    how=None)       # Type of join to perform (default is 'inner')
```

PySpark Join Example

```
# Inspect dataframe head
hdf.show(2)
```

```
+----------+--------------------+
|        dt|                  nm|
+----------+--------------------+
|2012-01-02|        New Year Day|
|2012-01-16|Martin Luther Kin...|
+----------+--------------------+
only showing top 2 rows
```

```
# Specify join conditon
cond = [df['OFFMARKETDATE'] == hdf['dt']]
# Join two hdf onto df
df = df.join(hdf, on=cond, 'left')
# How many sales occurred on bank holidays?
df.where(~df['nm'].isNull()).count()
```

```
0
```

SparkSQL Join

- Apply SQL to your dataframe

```
# Register the dataframe as a temp table
df.createOrReplaceTempView("df")
hdf.createOrReplaceTempView("hdf")
```

```
# Write a SQL Statement
sql_df = spark.sql("""
                      SELECT
                        *
                      FROM df
                      LEFT JOIN hdf
                      ON df.OFFMARKETDATE = hdf.dt
                   """)
```

Let's Join Some Data!

FEATURE ENGINEERING WITH PYSPARK