Feature Engineering

Techniques & Use Cases

— *Joy Qi*

References

- Feature Engineering Techniques
- Fundamental Techniques of Feature Engineering for Machine Learning
- `demo.ipynb` in Github Repo

THE THINKING PROCESS..

Techniques

Data Type

- Numerical
- Categorical
- Date/Timestamp
- String

1. Imputation
2. Handling Outliers
3. Binning
4. Grouping
5. Transform
6. Scaling
7. Encoding
8. Extraction

……

Data Problem

- Missing values
- Outliers
- Skewness
- Overfitting*

WHAT HOW WHY

The Menu

Numerical | Categorical | Timestamp

1.Imputation

- Just Drop it!
- Numerical Imputation
- Categorical Imputation

2.Outliers

- Drop it
- Cap it

3.Binning

- Numerical Binning
- Categorical Binning

4.Grouping

- Categorical Groupings
- Aggregated sum, mean etc

5.Transform

- Log, Square Root
- Box-Cox

6.Scaling

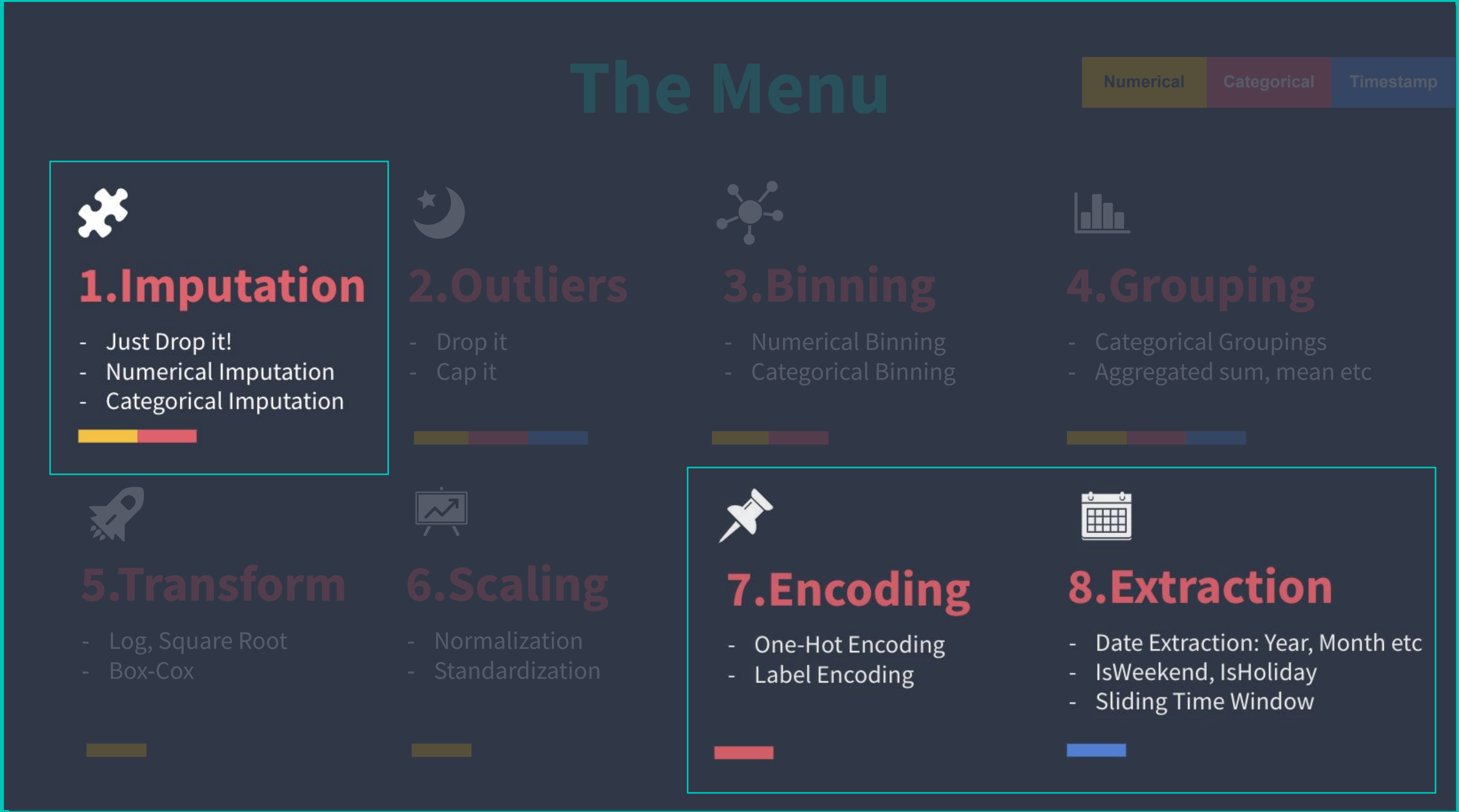
- Normalization
- Standardization

7.Encoding

- One-Hot Encoding
- Label Encoding

8.Extraction

- Date Extraction: Year, Month etc
- IsWeekend, IsHoliday
- Sliding Time Window



The Menu
Numerical
Categorical
Timestamp
1.Imputation
- Just Drop it!
- Numerical Imputation
- Categorical Imputation
2.Outliers
- Drop it
- Cap it
3.Binning
- Numerical Binning
- Categorical Binning
4.Grouping
- Categorical Groupings
- Aggregated sum, mean etc
5.Transform
- Log, Square Root
- Box-Cox
6.Scaling
- Normalization
- Standardization
7.Encoding
- One-Hot Encoding
- Label Encoding
8.Extraction
- Date Extraction: Year, Month etc
- IsWeekend, IsHoliday
- Sliding Time Window

1.Imputation

Use Case: Missing values (NA/NaN/NULL)

Just drop it!

Numerical Imputation

Categorical Imputation

1.Imputation

Use Case: Missing values (NA/NaN/NULL)

Just drop it!

```
#Dropping rows with NA% higher than threshold
row_threshold = 0.01
df = df.loc[df.isnull().mean(axis=1) < row_threshold]

#Dropping column with NA% higher than threshold
col_threshold = 0.99
df = df.loc[df.isnull().mean() < col_threshold]
```

Note: Always evaluate the percentage of missing values(NA%) first

1.Imputation

Use Case: Missing values (NA/NaN/NULL)

Numerical Imputation

```
#Filling all NAs with certain value
val = 0
df_filled = df.fillna(val)

#Filling NA with median/mean of each column
df_median = df.fillna(df.median())
```

1.Imputation

Use Case: Missing values (NA/NaN/NULL)

Categorical Imputation

```
#Replace all the NULLs of gender column with ‘Other’
df[‘gender’].replace(np.NaN, 'Other', inplace=True)
```

2.Outliers

Detection & Handling

Use Visualization to detect

Standard Deviation

Percentile

z-score

Drop it

Cap it

```
#Dropping the outlier rows with Standard Deviation
factor = 3
upper_lim = df['column'].mean() + df['column'].std() * factor
lower_lim = df['column'].mean() - df['column'].std() * factor

df = df[(df['column'] < upper_lim) & (df['column'] > lower_lim)]
```

```
#Dropping the outlier rows with Percentiles
upper_lim = df['column'].quantile(.95)
lower_lim = df['column'].quantile(.05)

df = df[(df['column'] < upper_lim) & (df['column'] > lower_lim)]
```

```
#Capping the outlier rows with Percentiles
upper_lim = df['column'].quantile(.95)
lower_lim = df['column'].quantile(.05)

df.loc[(df[column] > upper_lim),column] = upper_lim
df.loc[(df[column] < lower_lim),column] = lower_lim
```

3.Binning

Numerical Binning

Categorical Binning

Use Case: **Need for segmentation, Problem of Overfitting**

- Continuous to discrete data
- Reduce num. of categories
- Drawback: Sacrifice of information for performance

Numerical Binning

Value		Bin
0-30	—>	Low
31-70	—>	Mid
71-100	—>	High

```
#Numerical Binning Example

df['Bin'] = pd.cut(df['Value'], bins=[0,30,70,100], labels=["Low", "Mid", "High"])

   value   bin
0      2   Low
1     45   Mid
2      7   Low
3     85  High
4     28   Low
```

Categorical Binning

Country		Continent
Spain	—>	Europe
Italy	—>	Europe
Brazil	—>	South America
Singapore	—>	Asia

```
#Categorical Binning Example

conditions = [df['Country'].str.contains('Spain'),
              df['Country'].str.contains('Italy'),
              df['Country'].str.contains('Brazil'),
              df['Country'].str.contains('Singapore')]

choices = ['Europe', 'Europe', 'South America', 'Asia']

df['Continent'] = np.select(conditions, choices, default='Other')
```

4.Grouping

Use Case: Transaction datasets where one instance has multiple records hence needs to group by instance and apply aggregate functions

1. **Categorical Grouping**
    - Count for frequency
    - Pivot Table

	City	Counts
1	London	45678
2	Tokyo	23456
...	...	...

`.value_counts()`

2. **Numerical Grouping**
    - Aggregated sum, mean

	month	Sum
1	August	34000
2	September	56000
...	...	...

`.groupby()`

User	City	Visit_Days
1	Roma	1
2	Madrid	2
1	Madrid	1
3	Paris	1
2	Paris	4
1	Paris	3
1	Roma	3

Pivot Table

(Similar to One-Hot Encoding)

User	Paris	Madrid	Roma
1	3	1	4
2	4	2	0
3	1	0	0

```
#Pivot table Pandas Example
df.pivot_table(index='column_to_group',
               columns='column_to_encode',
               values='aggregation_column',
               aggfunc=np.sum,
               fill_value = 0)
```

5.Transform

Use Case: Data skewness which violates model assumptions

1. **Log Transformation:**
   - **Typical example:** the vast majority of incomes are small and very few are big
   - **Normalize the magnitude difference:** skewed to normal distribution
   - **Decrease outliers effect:** model become more robust

   Note: Can only be applied to positive values, negative value will return NULL

```
#Log Transform Example with Negative values
df = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})

#Negative Values Handling
#Choose constant c where c=1-Xmin so that min(X+c)=1 and (X-Xmin+1) will always >=1 (Log(1)=0)
df['log'] = (df['value']- df['value'].min()+1).transform(np.log)
```

2. **Other Transformations: Square Root, Box-Cox**

6.Scaling

Use Case: Compare columns with different ranges (e.g. Age and income)

1. **Normalization: Scale all column values to same fixed range [0, 1]**

$$X_{norm} = (X - X_{min})/(X_{max} - X_{min})$$

```
df = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})
df['normalized'] = (df['value'] - df['value'].min()) / (df['value'].max() - df['value'].min())
```

2. **Standardization: Take consideration of std to get different ranges**

$$z = (X - X_{mean})/std$$

```
df = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})
df['standardized'] = (df['value'] - df['value'].mean()) / df['value'].std()
```

7.Encoding

1. Label Encoding
    - Use Case: Convert categories in 1 column into numerical values
    - Drawback: Create unnecessary weightage for nominal categories

      ✔ Apply only to ordinal categories or use One-Hot Encoding

2. One-Hot Encoding
    - Use Case: Convert categories in 1 column into multiple binary columns
    - Drawback: Create too many binary features, data becomes sparse

      ✔ Selectively apply to the most frequent categories

Std	Grade
1	A
2	D
3	B
4	A
5	C
6	D

Ordinal Categories

Label Encoding

Std	Grade	Value
1	A	5
2	D	2
3	B	4
4	A	5
5	C	3
6	D	2

Id	Season
1	Spring
2	Winter
3	Summer

Nominal Categories

Id	Season	Value
1	Spring	1
2	Winter	4
3	Summer	3

User	City
1	Tokyo
2	Sydney
3	Tokyo
4	London
5	London
6	Honolulu

One-Hot Encoding

User	Tokyo	Sydney	London	Honolulu
1	1	0	0	0
2	0	1	0	0
3	1	0	0	0
4	0	0	1	0
5	0	0	1	0
6	0	0	0	1

	City	Counts
1	London	45678
2	Tokyo	23456
...	...	...
512	Honolulu	1
513	Budapest	1
514	Bangkok	1

Apply to the most frequent ones

Park under 'Others'

User	London	Tokyo	Others
1	1	0	0
2	0	1	0
...	...	...	...

Label Encoding for the Grade

```
#Create map of category:value
cat_dict = {'A': 5, 'B': 4, 'C': 3, 'D': 2}
df['Value']=df['Grade'].replace(cat_dict, inplace =False)
```

One-Hot Encoding for the first 30 most frequent categories

```
#Create category list (30 + 1 Other)
cat_list = df['col'].value_counts()[:30].keys().tolist()
cat_list.append('other')

#Map original column into a new column with only 31 categories
df['new_col']=pd.Categorical(df['col'], categories=cat_list)
df['new_col'].fillna('other', inplace=True)

#One-Hot Encoding
col_encoding = pd.get_dummies(df['new_col'])

#Join the encoding columns back to original DataFrame
df = pd.concat([df, col_encoding], axis=1)
df.drop(['new_col'], axis=1, inplace= True)
```

8.Extraction on Date

1. **Timestamp to year, month, day-of-week, hour, minute etc.**
   - **Use Case:** For aggregating features (e.g.Sum per month)

2. **IsWeekday, IsWeekend, IsHoliday**
   - **Use Case:** User behaviour driven (e.g. higher # of shopping transactions during Xmas holidays)

3. **Sliding Time Window**
   - **Use Case:** Aggregating under Dynamic/Customized time window

Extract Date Features: weekday, isweekend, isholiday

```
holiday_list = ['2019-01-01', '2019-01-21', '2019-05-27', '2019-07-04',
                '2019-09-02', '2019-11-28', '2019-11-29', '2019-12-25']
```

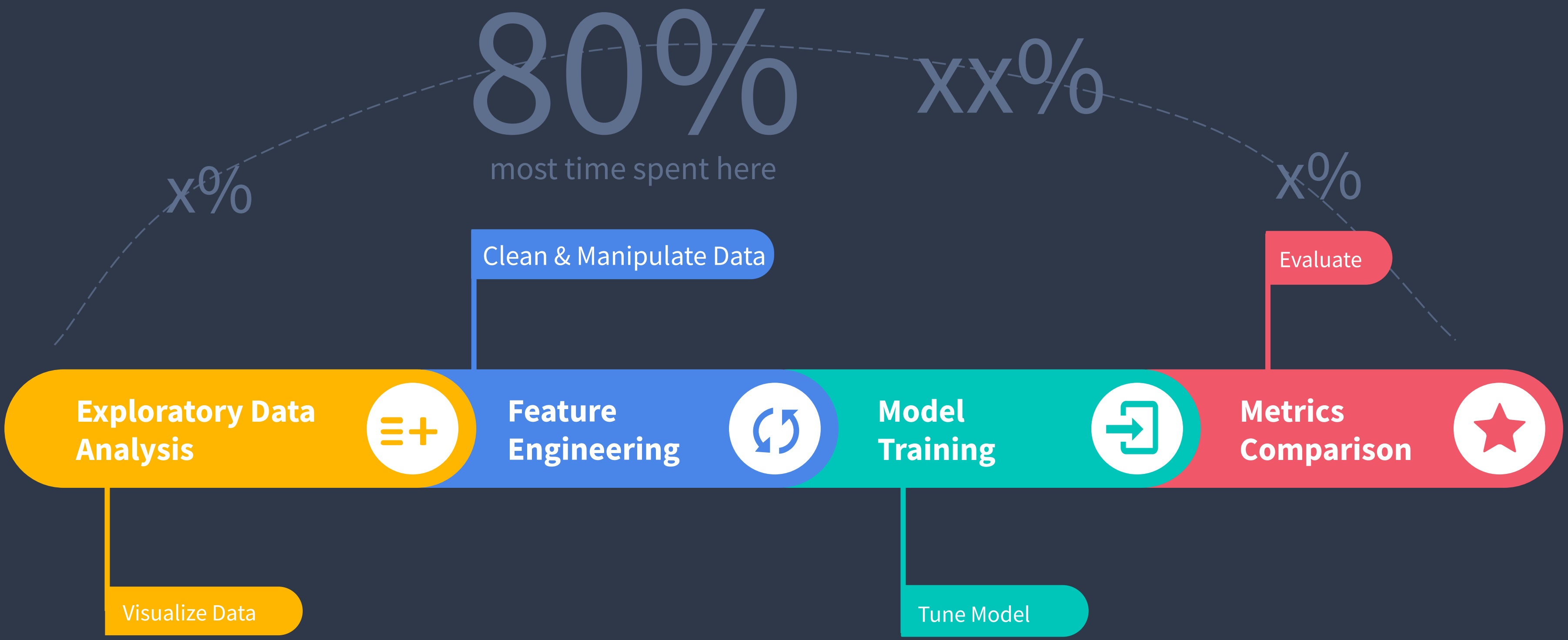
```
def get_date_features(df):
    """
    Given a DataFrame with Date columns
    Create Binary Columns on whether the date:
    - is weekend: 1, otherwise: 0
    - is Federal holiday: 1, otherwise: 0

    Return a DataFrame with features created
    """
    df['departure_weekday'] = pd.to_datetime(df['departure_date']).dt.weekday
    df['departure_isweekend'] = (df['departure_weekday'].isin([6, 7])).astype(int)
    df['departure_isholiday'] = (df['departure_weekday'].isin(holiday_list)).astype(int)

    return df
```

```
df = get_date_features(df)
```

GENERAL DATA SCIENCE STEPS



THE TAKEAWAYS

1.

2.

Feature Engineering comes from understanding the Data

The Specials for next time

NLP

- TF-IDF
- StopWords removal
- Continuous Bag-of-Words
- Word Embeddings

Web Scraping

- BeautifulSoup BS4
- Selenium
- Handling JSON data

Regularization

- L1: Lasso
- L2: Ridge
- Elastic-net (combination of L1 & L2)

PCA

- Principal Component Analysis
- linear dimensionality reduction

K-means

- Elbow Method
- Calculate Distances: Euclidean, Haversin etc.

Imbalanced data

- Up-sampling
- Down-sampling

THANK YOU

Happy Data Science 🙂🙃🙂🙃

Q&A