WAYNE STATE
UNIVERSITY

Data Preprocessing: Cleaning and Organizing

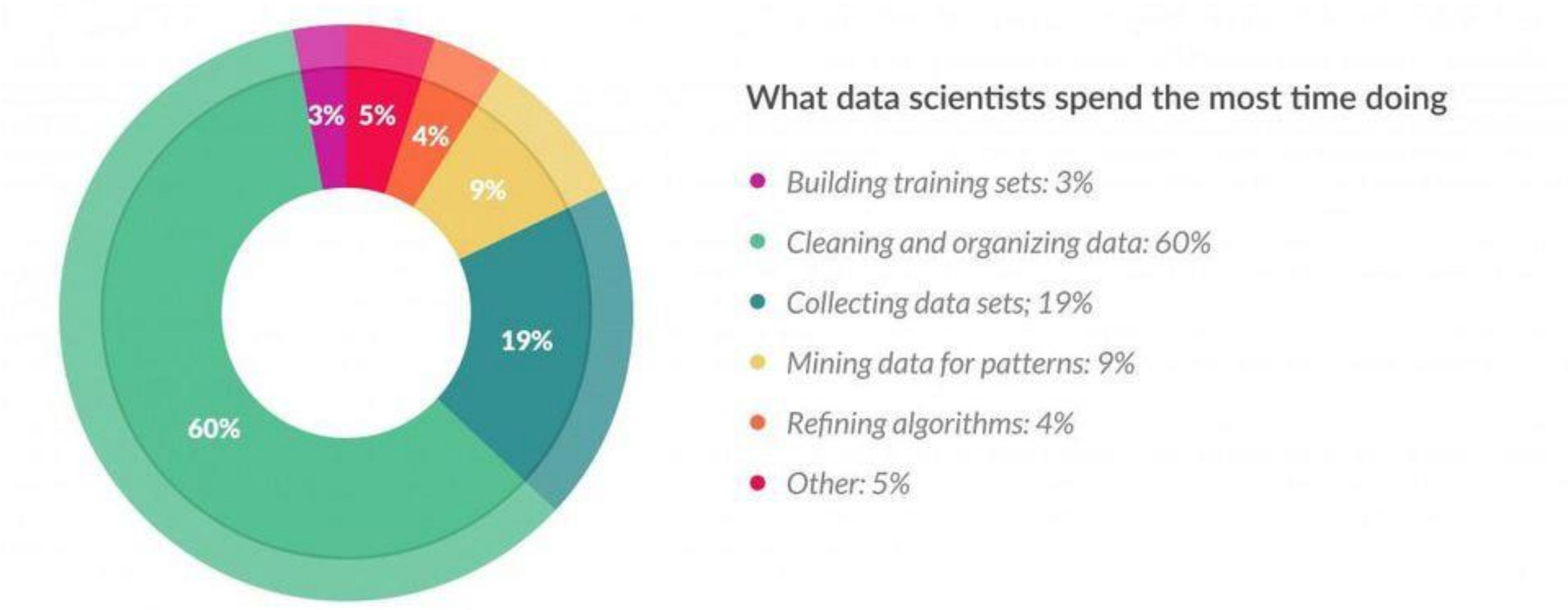
Dr. Ratna Babu Chinnam

Industrial & Systems Engineering

Wayne State University

Source: Emre Rencberoglu | Link

Motivation



According to a survey in Forbes (March 2016), data scientists spend **80%** of their time on **data preparation**. | Link

Hughes, G.F. (January 1968). "On the mean accuracy of statistical pattern recognizers". *IEEE Transactions on Information Theory*. **14** (1): 55–63.doi:10.1109/TIT.1968.1054102

Handling Outliers*

CAUTION: Best way to detect outliers is to explore data visually. Numerical methods are prone to mistakes.

Outlier Detection with Standard Deviation (σ)

- If a value has a distance to the average higher than $k * \sigma$, it can be assumed as an outlier. What should k (factor) be?
  - No trivial solution for k, but usually, a value between (2, 4) seems practical.
- #Dropping outlier rows with σ

```
factor = 3
upper_lim = data['column'].mean () + data['column'].std () * factor
lower_lim = data['column'].mean () - data['column'].std () * factor
data = data[(data['column'] < upper_lim) & (data['column'] > lower_lim)]
```

Outlier Detection with Percentiles

- Another method to detect outliers is to use percentiles.
  - Assume a certain % of the value from the top or the bottom as an outlier.
  - Threshold depends on distribution of data.
  - If your data ranges from **0** to **100**, your top **5%** is not the values between **96** and **100**. Top **5%** means here the values that are out of the **95th** percentile of data.
- #Dropping the outlier rows with Percentiles

```
upper_lim = data['column'].quantile(.95)
lower_lim = data['column'].quantile(.05)
data = data[(data['column'] < upper_lim) & (data['column'] > lower_lim)]
```

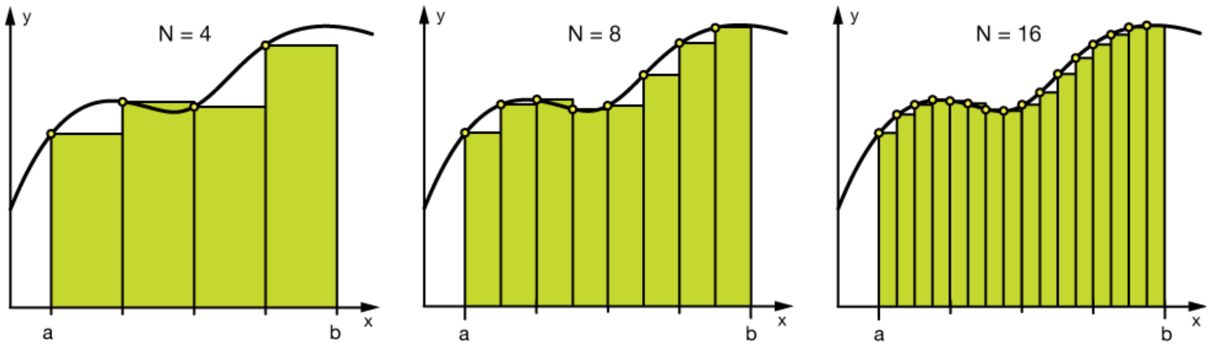
*Python code assumes that you have imported **Pandas** and **Numpy** libraries.

An Outlier Dilemma: Drop or Cap

- If possible, detected outliers should be confirmed with domain experts
- One option for handling outliers is to **cap** them instead of dropping.
    - It might be better for final model performance.
    - Capping can affect the distribution of the data.
- #Capping the outlier rows with Percentiles
    - `upper_lim = data['column'].quantile(.95)`
    - `lower_lim = data['column'].quantile(.05)`
    - `data.loc[(df[column] > upper_lim),column] = upper_lim`
    - `data.loc[(df[column] < lower_lim),column] = lower_lim`

Variable Binning

- Process of converting a numerical variable into a categorical variable
  - Age: [0,5),[5,10),[10,20),[20,60),[60+)
  - Ensure that all bins have a decent frequency
- Binning can help deal with highly non-linear effects
  - May not be useful for ANNs; Might even degrade performance.



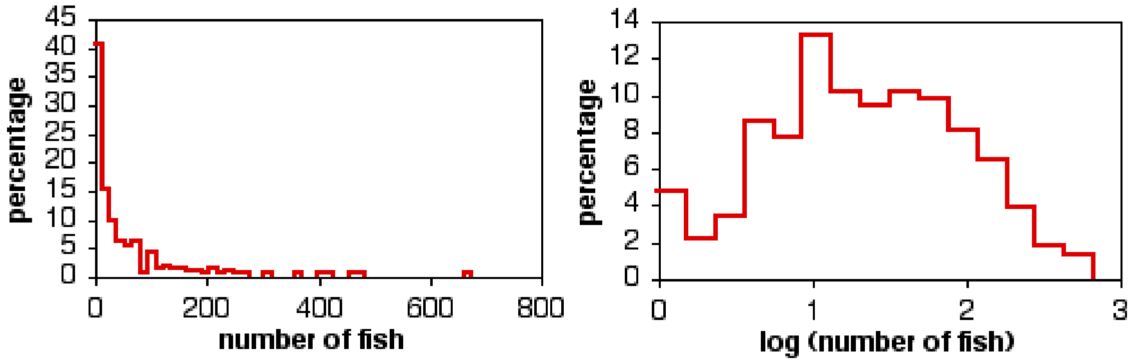
#Numerical Binning Example

- ```
  data['bin'] = pd.cut(data['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
  ```

Logarithm Transformation

- Log transform is commonly used in feature engineering to deal with skewed data.
  - May not be useful for ANNs; Might even degrade performance.
  - It also decreases the effect of the outliers, due to the normalization of magnitude differences and the model become more robust.
- Data must have only + values, otherwise you receive an error.
  - Add a constant to your data before transformation to avoid issues.



Source: McDonald | Link

#Log Transform Example

- `data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})`
- `data['log+1'] = (data['value']+1).transform(np.log)`
- #Negative Values Handling
- #Note that the values are different
- `data['log'] = (data['value']-data['value'].min()+1) .transform(np.log)`

```
  value  log(x+1)  log(x-min(x)+1)
0     2   1.09861          3.25810
1    45   3.82864          4.23411
2   -23       nan          0.00000
3    85   4.45435          4.69135
4    28   3.36730          3.95124
5     2   1.09861          3.25810
6    35   3.58352          4.07754
7   -12       nan          2.48491
```

One-Hot Encoding

- Spreads the values in a column to multiple flag columns and assigns 0 or 1 to them.
  - These binary values express the relationship between grouped and encoded column.
- If you have N distinct values in the column, it is enough to map them to N-1 binary columns, because the missing value can be deducted from other columns.
- `encoded_columns = pd.get_dummies(data['column'])`
- `data = data.join(encoded_columns).drop('column', axis=1)`

Scaling

- In many cases, numerical features of dataset do not have a certain range and they differ from each other.
- Scaling data to a fixed range (e.g., [0,1]) or standardizing it by making the $\mu = 0$ and $\sigma = 1$ can help accelerate learning.
- **Fixed Range:**
- `data = pd.DataFrame({'value':[2,45,-23,85,28,2]})`
- `data['normalized'] = (data['value'] - data['value'].min()) / (data['value'].max() - data['value'].min())`

```
   value  normalized
0      2        0.23
1     45        0.63
2    -23        0.00
3     85        1.00
4     28        0.47
5      2        0.23
```

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

```
   value  standardized
0      2         -0.52
1     45          0.70
2    -23         -1.23
3     85          1.84
4     28          0.22
5      2         -0.52
```

Extracting Date

- Date/time columns might provide valuable information about the model target (e.g., sales on weekdays vs weekends; ER admissions by hour of day)
- Three types of preprocessing for dates:
  - Extracting the parts of the date into different columns: Year, month, day, etc.
  - Extracting the time period between the current date and columns in terms of years, months, days, etc.
  - Extracting some specific features from the date: Name of the weekday, Weekend or not, holiday or not, etc.
- `from datetime import date`
- `data = pd.DataFrame({'date':['01-01-2017','04-12-2008','23-06-1988','25-08-1999','20-02-1993',]})`
- #Transform string to date

```
data['date'] = pd.to_datetime(data.date, format="%d-%m-%Y")
```

- #Extracting Year

```
data['year'] = data['date'].dt.year
```

- #Extracting Month

```
data['month'] = data['date'].dt.month
```

- #Extracting passed years since the date

```
data['passed_years'] = date.today().year - data['date'].dt.year
```

- #Extracting passed months since the date

```
data['passed_months'] = (date.today().year - data['date'].dt.year) * 12 + date.today().month - data['date'].dt.month
```

- #Extracting the weekday name of the date

```
data['day_name']=data['date'].dt.day_name()
```

	date	year	month	passed_years	passed_months	day_name
0	2017-01-01	2017	1	2	26	Sunday
1	2008-12-04	2008	12	11	123	Thursday
2	1988-06-23	1988	6	31	369	Thursday
3	1999-08-25	1999	8	20	235	Wednesday
4	1993-02-20	1993	2	26	313	Saturday

References

- Jacqes Peeters (April 2020) A framework for feature engineering and machine learning pipelines | Link
  - Very good general outline on how to write effective code
- Emre Rençberoğlu (April 2019) Fundamental Techniques of Feature Engineering for Machine Learning | Link
- Ways to Detect and Remove the Outliers | Link
- Understanding Feature Engineering:
  - Continuous Numeric Data | Link
  - Categorical Data | Link
- Log Transformations for Skewed and Wide Distributions | Link
- Tidy data | Link
- About Feature Scaling and Normalization | Link