HyperionDev

Data Analysis

August 2024

Data Science Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Data Science Session Housekeeping cont.

- For all **non-academic questions**, please submit a query:
  www.hyperiondev.com/support

- Report a **safeguarding** incident:
  www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: **Feedback on Lectures**

Data Cleaning

- Data cleaning is a crucial step in the data science pipeline
- Ensures **data quality and reliability** for analysis and modeling
- Common data quality issues include **missing data, duplicates, inconsistent formatting, and outliers**

Missing Data

HyperionDev

Handling Missing Data

- Missing data refers to the **absence of values in one or more variables in a dataset.**

- Identifying missing values:
  - Look for **null, NaN, or empty cells in the dataset**.
  - Use functions like **isnull() or isna()** in Pandas

```python
# Identify missing values
df_missing.isnull().sum()
```

✓ 0.0s

```
total_bill    10
tip            0
sex           10
smoker         0
day            0
time           0
size           0
dtype: int64
```

Understand Missing Data Mechanisms

- **MCAR:** Missing Completely at Random (missingness unrelated to any variables)
  - Smoking status is not recorded in a random sample of patients
- **MAR:** Missing at Random (missingness depends on observed variables)
  - Smoking status is not documented in female patients because the doctor was to shy to ask 🙂
- **MNAR:** Missing Not at Random (missingness depends on missing values themselves)
  - Smoking status is not recorded in patients admitted as an emergency, who are also more likely to have worse outcomes from surgery

Techniques for Handling Missing Data

- **Deletion:** Remove records with missing values (only suitable if missing data is minimal and random).
  - Suitable for random missingness
  - Not the first resort, dropping data means losing some important context or skewing the dataset in some cases

```
df.shape
✓ 0.0s
(244, 7)
```

```
# Deletion: Remove records with missing values
df_deleted = df_missing.dropna()
df_deleted.shape
✓ 0.0s
(225, 7)
```

HyperionDev

Techniques for Handling Missing Data

- **Imputation:** Fill in missing values with estimated or calculated values.
  - Simple imputation: Mean, median, or mode imputation

```
# Simple Imputation: Fill missing values with mean for numeric columns and mode for categorical
# columns
df_imputed = df_missing.copy()
df_imputed['total_bill'] = df_imputed['total_bill'].fillna(df_imputed['total_bill'].mean())
df_imputed['sex'] = df_imputed['sex'].fillna(df_imputed['sex'].mode()[0])
```

  - Advanced imputation: K-Nearest Neighbors (KNN), Multiple Imputation by Chained Equations (MICE)
    - We'll get to KNN in another lecture 🤩

HyperionDev

Duplicates

HyperionDev

Dealing with Duplicates

❖ Identify duplicates using functions like duplicated() in Pandas

```
# Show all duplicated rows
df_duplicates[df_duplicates.duplicated(keep=False)]
```

	total_bill	tip	sex	smoker	day	time	size
46	22.23	5.00	Male	No	Sun	Dinner	2
92	5.75	1.00	Female	Yes	Fri	Dinner	2
123	15.95	2.00	Male	No	Thur	Lunch	2
158	13.39	2.61	Female	No	Sun	Dinner	2
198	13.00	2.00	Female	Yes	Thur	Lunch	2
202	13.00	2.00	Female	Yes	Thur	Lunch	2
234	15.53	3.00	Male	Yes	Sat	Dinner	2
244	22.23	5.00	Male	No	Sun	Dinner	2
245	15.53	3.00	Male	Yes	Sat	Dinner	2
246	13.39	2.61	Female	No	Sun	Dinner	2
247	5.75	1.00	Female	Yes	Fri	Dinner	2
248	15.95	2.00	Male	No	Thur	Lunch	2

keep = False just marks all duplicates.

Dealing with Duplicates

- Dropping duplicates is fine and encouraged, it does not cause the data to lost necessary context

```
# Remove duplicate records
df_deduplicated = df_duplicates.drop_duplicates()
```

Formatting and Standardization

HyperionDev

Data Formatting and Standardization

- Consistent data formatting is essential for accurate analysis and compatibility
- Common formatting issues:
  - **Date and time formats:** Ensure consistent representation (e.g., YYYY-MM-DD, HH:MM:SS)
  - **Text case inconsistencies:** Convert text to a consistent case (lowercase or uppercase)
  - **Inconsistent value representations:** Standardize values (e.g., "Yes"/"No" vs. "Y"/"N")

Data Formatting and Standardization

- Techniques for standardizing data:
  - Convert date/time columns using to_datetime()
  - Convert text case using str.lower() or str.upper()
  - Map inconsistent values to standardized representations

```
# Standardize text case for 'sex' and 'smoker' columns
df['sex'] = df['sex'].str.upper()
df['smoker'] = df['smoker'].str.title()

df.head()
```

✓ 0.0s

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	FEMALE	No	Sun	Dinner	2
1	10.34	1.66	MALE	No	Sun	Dinner	3
2	21.01	3.50	MALE	No	Sun	Dinner	3
3	23.68	3.31	MALE	No	Sun	Dinner	2
4	24.59	3.61	FEMALE	No	Sun	Dinner	4

Outliers

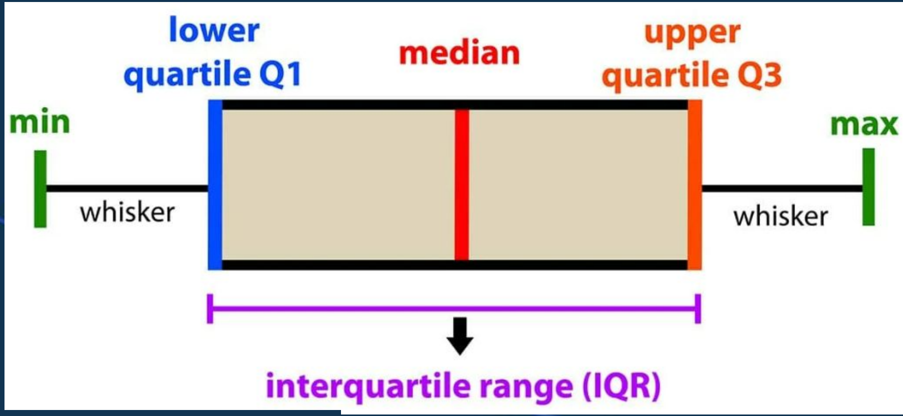
HyperionDev

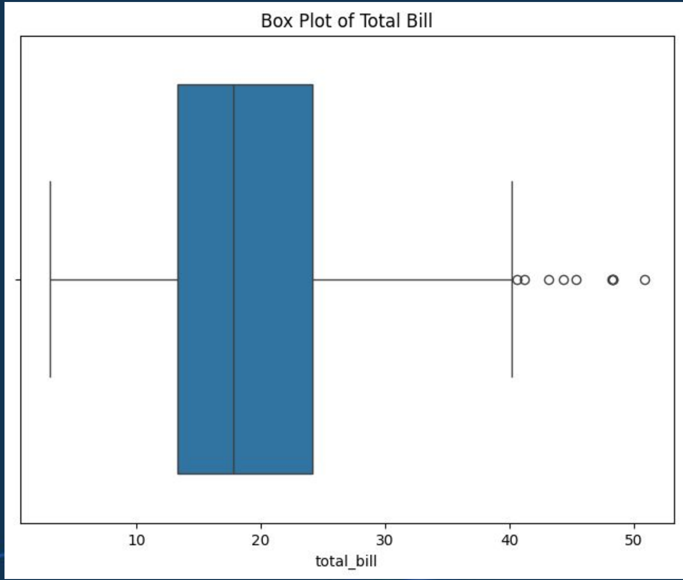
Outliers

❖ Outliers are data points that significantly deviate from the rest of the data distribution

Identifying Outliers

❖ Visual inspection using plots like box plots, scatter plots, or histograms





HyperionDev

Identifying Outliers

- Statistical methods like z-score or interquartile range (IQR)
  - Much less common given how good box plot already show outliers

```python
# Identify outliers using z-score
from scipy import stats

z_scores = np.abs(stats.zscore(df['total_bill']))
threshold = 2.5
outliers_zscore = np.where(z_scores > threshold)

outliers_zscore
```

✓ 0.0s

```
(array([ 59, 102, 156, 170, 182, 197, 212]),)
```

```python
# Identify outliers using Interquartile Range (IQR)
Q1 = df['total_bill'].quantile(0.25)
Q3 = df['total_bill'].quantile(0.75)
IQR = Q3 - Q1

outliers_iqr = df[(df['total_bill'] < (Q1 - 1.5 * IQR)) | (df['total_bill'] > (Q3 + 1.5 * IQR))]
len(outliers_iqr)
```

✓ 0.0s MagicPython

```
9
```

Handling Outliers

- **Removal:** Remove outliers if they are erroneous or irrelevant to the analysis
  - Use when outliers are clearly erroneous or irrelevant to the analysis
  - Be cautious, as removing outliers may result in loss of information

```
# Removal: Remove outliers
df_removed = df[~((df['total_bill'] < (Q1 - 1.5 * IQR)) | (df['total_bill'] > (Q3 + 1.5 * IQR)))]
df_removed.shape
```

✓ 0.0s MagicPython

```
(235, 9)
```

Handling Outliers

- **Transformation:** Apply mathematical transformations (e.g., logarithmic, square root) to reduce the impact of outliers
  - Use when outliers are valid but have a skewed distribution
  - Helps to reduce the impact of outliers while preserving the data

Handling Outliers

- **Winsorization:** Replace extreme values with the nearest non-outlier values
  - Use when outliers are valid but need to be treated to reduce their influence
  - Maintains the overall distribution shape while limiting the extreme values

Iterative Data Cleaning

HyperionDev

Iterating

- Data cleaning is an iterative process that may require multiple rounds
- Continuously assess and refine the cleaned data based on analysis results and feedback
- Integrate data cleaning with data analysis and modeling for optimal results

HyperionDev

Libraries

HyperionDev

fuzzywuzzy

- Provides string matching and similarity scoring functions
- Key features:
  - **Ratio:** Calculates the similarity ratio between two strings
  - **Partial Ratio:** Calculates the similarity ratio considering substrings
  - **Token Set Ratio:** Calculates the similarity ratio considering common tokens

fuzzywuzzy

```
!pip3 install fuzzywuzzy python-Levenshtein
```

```
from fuzzywuzzy import fuzz
```

```
fuzz.ratio("apple", "appel")
✓ 0.0s

80
```

```
fuzz.partial_ratio("apple", "app")
✓ 0.0s

100
```

```
# Gives a 100 if every token in the first string is in the second string
fuzz.token_set_ratio("apple orange", "orange apple")
✓ 0.0s

100
```

HyperionDev

chardet

- Detects the encoding of a given byte string
- Key features:
  - Supports various encodings (e.g., UTF-8, ISO-8859-1, etc.)
  - Provides confidence scores for detected encodings

```
!pip3 install chardet
```

```
import chardet
```

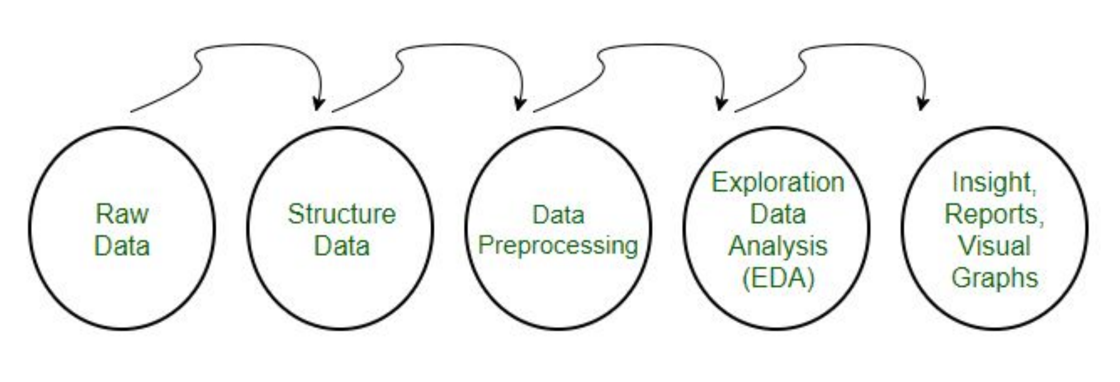
```
byte_string = b"Hello, World!"
encoding = chardet.detect(byte_string)
encoding
```

✓ 0.0s

```
{'encoding': 'ascii', 'confidence': 1.0, 'language': ''}
```

Data Preprocessing

- Data preprocessing is a crucial step in the data science pipeline, going beyond basic cleaning to ensure data quality and suitability for machine learning.



Source: [GeeksForGeeks]

Importance of Data Preprocessing

HyperionDev

Importance

- **Improved data quality:** Addresses complex issues beyond basic cleaning
- **Enhanced model performance:** Optimizes data for learning algorithms
- **Reduced computational complexity:** Reduces dimensionality and creates efficient representations

Feature Scaling

HyperionDev

Feature Scaling

- Purpose: Ensure fair comparison and contribution of features
- Techniques:
  - **Standardization (Z-score normalization):** Transforms features to have zero mean and unit varia

    $$X' = \frac{X - \mu}{\sigma}$$

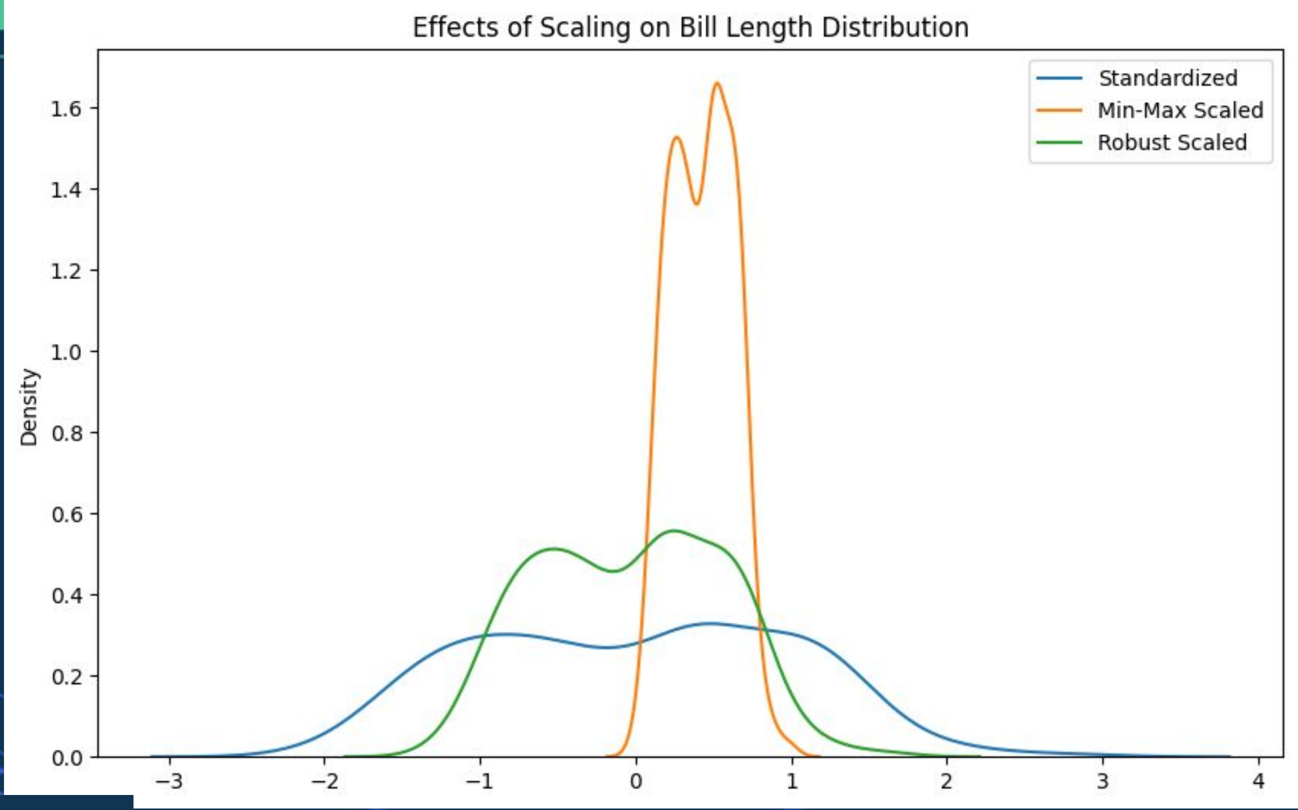
  - **Min-max scaling:** Scales features to a specific range, typically 0 to 1

    $$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

  - **Robust scaling:** Uses robust statistics (median and interquartile range) to scale features
    - (X-median)/IQR

Considerations

- **Standardization:** Good default, assumes Gaussian distribution
- **Min-max scaling:** Suitable for bounded features or non-Gaussian data
- **Robust scaling:** Recommended when outliers are present



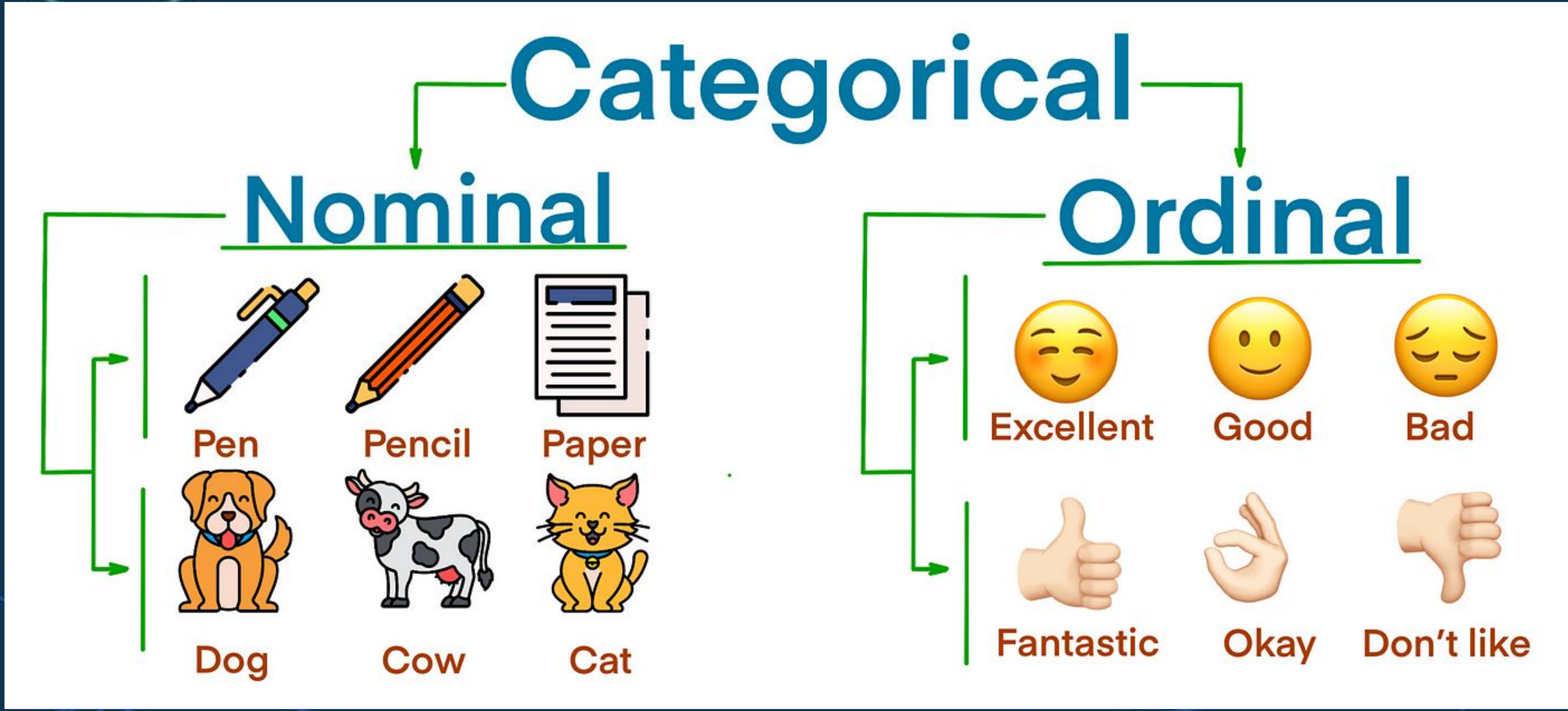
HyperionDev

Encoding Categorical Variables

HyperionDev

Nominal vs. Ordinal

- **Nominal:** Categories without inherent order (e.g., color)
- **Ordinal:** Categories with meaningful order (e.g., size)


Categorical
Nominal
Ordinal
Pen
Pencil
Paper
Dog
Cow
Cat
Excellent
Good
Bad
Fantastic
Okay
Don't like

Encoding Nominal variables

- One-hot encoding: Creates binary dummy variables for each category
  - Increases dimensionality, which may impact model performance

Index	Animal
0	Dog
1	Cat
2	Sheep
3	Horse
4	Lion

One-Hot code →

Index	Dog	Cat	Sheep	Lion	Horse
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	0	1
4	0	0	0	1	0

Source: AnalyticsVidhya

Encoding Nominal variables

- ❖ Binary encoding: Assigns unique binary codes to categories
  - ➢ Useful when the number of categories is large, and one-hot encoding leads to high dimensionality

	City
0	Delhi
1	Mumbai
2	Hyderabad
3	Chennai
4	Bangalore
5	Delhi
6	Hyderabad
7	Mumbai
8	Agra

	City_0	City_1	City_2	City_3
0	0	0	0	1
1	0	0	1	0
2	0	0	1	1
3	0	1	0	0
4	0	1	0	1
5	0	0	0	1
6	0	0	1	1
7	0	0	1	0
8	0	1	1	0

Source: AnalyticsVidhya

HyperionDev

Encoding Ordinal variables

- Label encoding: Assigns numerical labels based on order
  - Maintains ordinal information but implies linear relationships between categories
  - May not be appropriate if the ordinal relationship is not linear

	Degree
0	1
1	4
2	2
3	3
4	3
5	4
6	5
7	1
8	1

Source: AnalyticsVidhya

HyperionDev

Encoding Ordinal variables

- Ordinal encoding: Assigns numerical labels based on order
  - Preserves ordinal information without implying linear relationships
  - Suitable when the ordinal relationship between categories is meaningful

	Degree
0	1
1	4
2	2
3	3
4	3
5	4
6	5
7	1
8	1

Source: AnalyticsVidhya

HyperionDev

Handling High-Cardinality

HyperionDev

Handling High-Cardinality

❖ The Curse of Dimensionality:

As the number of features grows, the amount of data we need to accurately be able to distinguish between these features (in order to give us a prediction) and generalize our model (learned function) grows EXPONENTIALLY.

Source: AnalyticsVidhya

Frequency-based Encoding

- Replaces categories with occurrence count
- Useful when the frequency of categories is informative

Target Encoding

❖ Replaces categories with mean/median of target variable

❖ Captures the relationship between categories and the target variable

	class	Marks
0	A,	50
1	B	30
2	C	70
3	B	80
4	C	45
5	A	97
6	A	80
7	A	68

	class
0	65.000000
1	57.689414
2	59.517061
3	57.689414
4	59.517061
5	79.679951
6	79.679951
7	79.679951

Source: [AnalyticsVidhya]

Hashing

- ❖ Applies hash function to reduce dimensionality
- ❖ Useful when the number of categories is extremely large

	Month
0	January
1	April
2	March
3	April
4	Februay
5	June
6	July
7	June
8	September

	col_0	col_1	col_2	col_3	col_4	col_5
0	0	0	0	0	1	0
1	0	0	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	1	0	0
4	0	0	0	1	0	0
5	0	1	0	0	0	0
6	1	0	0	0	0	0
7	0	1	0	0	0	0
8	0	0	0	0	1	0

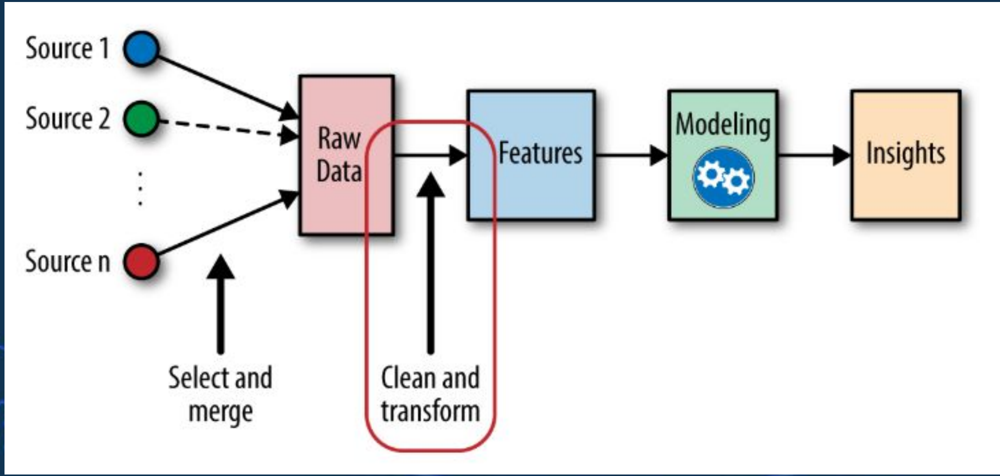
Source: AnalyticsVidhya

Feature Engineering

HyperionDev

Feature Engineering

❖ Create informative features that improve model performance and interpretability



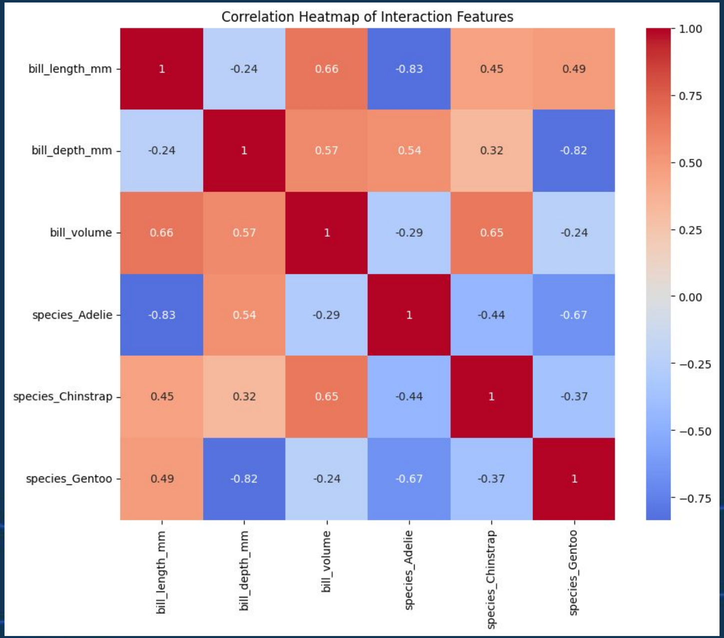
Source: [AnalyticsVidhya]

Techniques

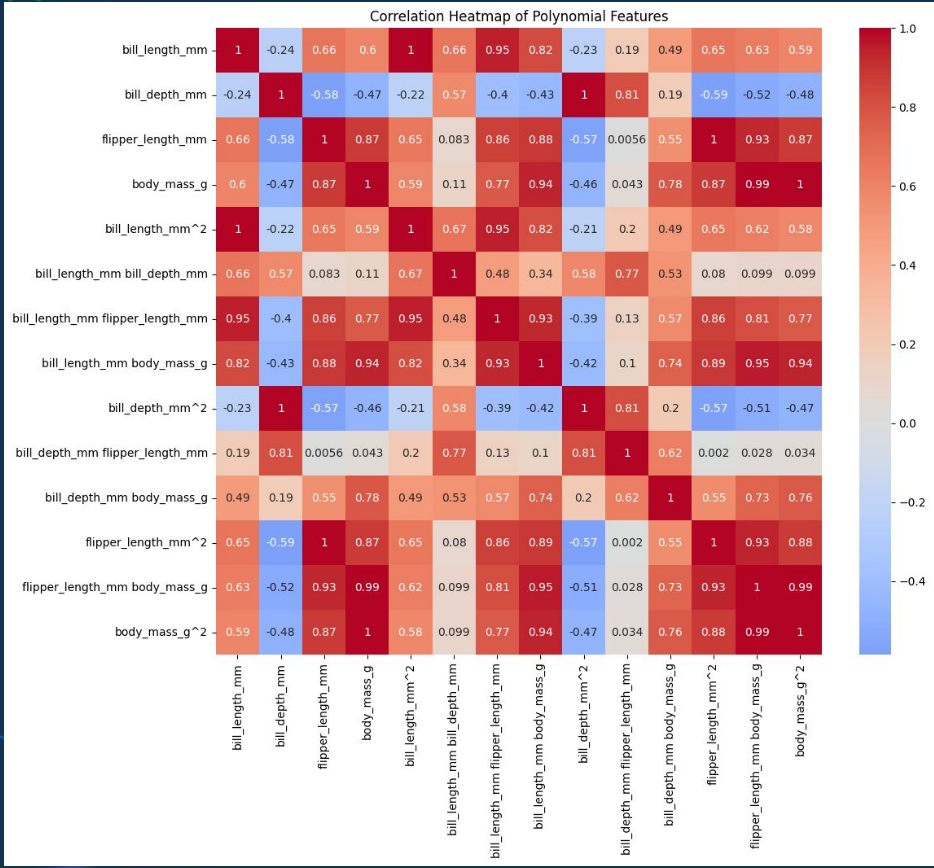
- **Interaction features:** Combine existing features to capture interactions
- **Polynomial features:** Generate higher-order terms to capture non-linear relationships
- **Domain-specific features:** Apply domain knowledge to create meaningful features

Interaction Features

```python
# Interaction features
penguins['bill_volume'] = penguins['bill_length_mm'] * penguins['bill_depth_mm']
```



Polynomial Features



HyperionDev

Handling Imbalanced Data

HyperionDev

Challenge

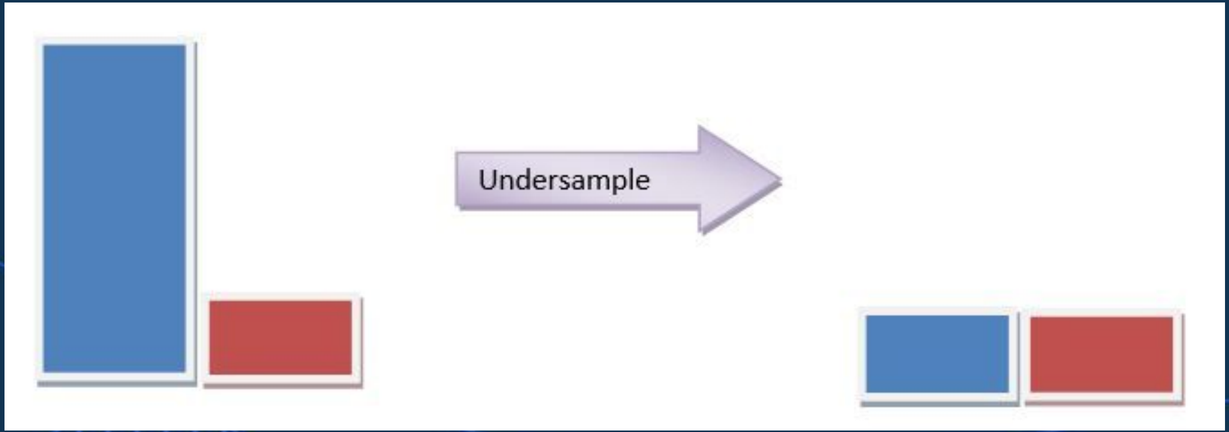
- Skewed class distribution leads to biased models and poor minority class performance

Source: AnalyticsVidhya

HyperionDev

Techniques

- ❖ **Undersampling:** Reduce majority class instances
  - ➢ **Random undersampling:** Remove majority instances



Source: AnalyticsVidhya

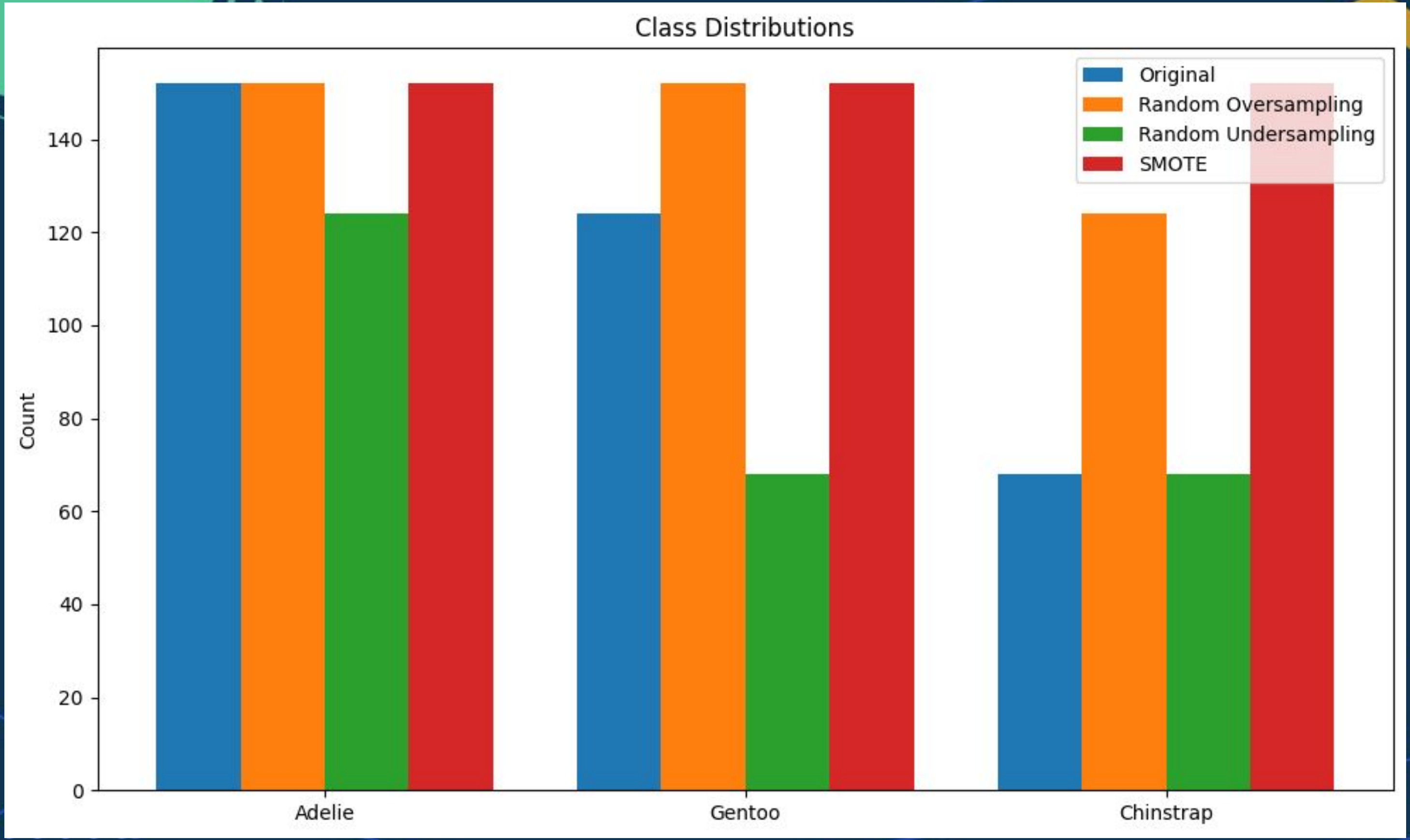
HyperionDev

Techniques

- **Oversampling:** Increase minority class instances
  - **Random oversampling:** Duplicate minority instances
  - **SMOTE:** Generate synthetic minority instances

Source: [AnalyticsVidhya]

HyperionDev


Class Distributions
Original
Random Oversampling
Random Undersampling
SMOTE
Count
0
20
40
60
80
100
120
140
Adelie
Gentoo
Chinstrap

Considerations

- ❖ Oversampling may lead to overfitting, especially with random oversampling
- ❖ Undersampling may discard potentially useful data

Questions and Answers

HyperionDev

Thank you for attending

HyperionDev