Lecture #2

Data Cleaning and Preprocessing

Dr. Charnon Pattiyanon
Assistant Director of IT, Instructor
Department of Artificial Intelligence and Computer Engineering
CMKL University

Recap from the Previous Session

- We were agreed on the definition of data and information and their differences.
- We learned about types of data, data representation, data quality, and data sensitivity.

Section 1:

An Introduction to Data Cleaning and Preprocessing

1.1 Introduction to Data Cleaning and Preprocessing

- **Why are data cleaning and preprocessing so crucial?**

  In essence, data is messy. Real-world data, the kind that companies and organizations collect every day, is often filled with inaccuracies, inconsistencies, and missing entries.

color	director_name	duration	gross	movie_title	anguage	country	budget	title_year	imdb_score
Color	Martin Scorsese	240	116866727	The Wolf of Wall StreetÂ	English	USA	100000000	2013	8.2
Color	Shane Black	195	408992272	Iron Man 3Â	English	USA	200000000	2013	7.2
color	Quentin Tarantino	187	54116191	The Hateful EightÂ	English	USA	44000000	2015	7.9
Color	Kenneth Lonergan	186	46495	MargaretÂ	English	usa	14000000	2011	6.5
Color	Peter Jackson	186	258355354	The Hobbit: The Desolation of SmaugÂ	English	USA	225000000	2013	7.9
	N/A	183	330249062	Batman v Superman: Dawn of JusticeÂ	English	USA	250000000	202	6.9
Color	Peter Jackson	-50	303001229	The Hobbit: An Unexpected JourneyÂ	English	USA	180000000	2012	7.9
Color	Edward Hall	180		RestlessÂ	English	UK		2012	7.2
Color	Joss Whedon	173	623279547	The AvengersÂ	English	USA	220000000	2012	8.1
Color	Joss Whedon	173	623279547	The AvengersÂ	English	USA	220000000	2012	8.1
	Tom Tykwer	172	27098580	Cloud AtlasA	English	Germany	102000000	2012	-7.5
Color	Null	158	102515793	The Girl with the Dragon TattooÂ	English	USA	90000000	2011	7.8
Color	Christopher Spencer	170	59696176	Son of GodÂ	English	USA	22000000	2014	5.6
Color	Peter Jackson	164	255108370	The Hobbit: The Battle of the Five ArmiesÂ	English	New Zealand	250000000	2014	7.5
Color	Tom Hooper	158	148775460	Les MisÃ©rablesÂ	English	USA	61000000	2012	7.6
Color	Tom Hooper	158	148775460	Les MisÃ©rablesÂ	English	USA	61000000	2012	7.6
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

1.2 Data Cleaning and Preprocessing Workflow

- A typical data cleaning and preprocessing workflow may involve the following steps:

Data Collection

Collect the raw data from various **sources**. The data might come from databases, APIs, web scraping, manual entry, etc.

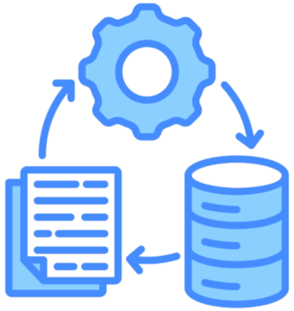
Data Cleaning

Clean the collected data by identifying and correcting **errors**, removing **duplicates** and irrelevant observations, and handling **missing values**.

Data Integration

Integrate data from multiple sources, **resolving** any **inconsistencies**. This might involve aligning columns, dealing with **conflicting entries**, and merging tables or datasets.

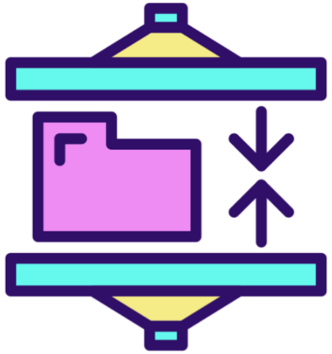
Data Transformation



Transform the data to make it suitable for analysis. This might include **encoding** categorical variables, **normalizing** numerical features, and **creating** derived features.

Data Reduction

Reduce the data dimensionality if necessary. This might include **feature selection** and **extraction** to focus on the most relevant variables



1.3 Python Libraries for Data Cleaning and Preprocessing

- **Python** is a preferred language for many data scientists, mainly because of its ease of use, and extensive, feature-rich libraries dedicated to data tasks
  - **Pandas:** It is a widely-used data manipulation library in Python. It provides data structures and functions needed to manipulate structured data. It includes key features for **filtering**, **sorting**, **aggregating**, **merging**, **reshaping**, **cleaning**, and **data wrangling**.

```
# import the pandas library
import pandas as pd

# read a CSV file into a pandas DataFrame
df = pd.read_csv('filename.csv')

# display the first few rows
df.head()
```

	color	director_name	duration	gross	genres	movie_title	title_year	language
0	Color	Martin Scorsese	240	116866727.0	Biography|Comedy|Crime|Drama	The Wolf of Wall Street	2013	English
1	Color	Shane Black	195	408992272.0	Action|Adventure|Sci-Fi	Iron Man 3	2013	English
2	color	Quentin Tarantino	187	54116191.0	Crime|Drama|Mystery|Thriller|Western	The Hateful Eight	2015	English
3	Color	Kenneth Lonergan	186	46495.0	Drama	Margaret	2011	English
4	Color	Peter Jackson	186	258355354.0	Adventure|Fantasy	The Hobbit: The Desolation of Smaug	2013	English

1.3 Python Libraries for Data Cleaning and Preprocessing (Cont')

- **Python** is a preferred language for many data scientists, mainly because of its ease of use, and extensive, feature-rich libraries dedicated to data tasks
  - **NumPy:** NumPy, short for 'Numerical Python', is another fundamental library for numerical computations in Python. It provides a high-performance, multidimensional array object and tools for working with arrays. Although Pandas is generally more high-level, NumPy is extensively used under the hood in many Pandas operations.

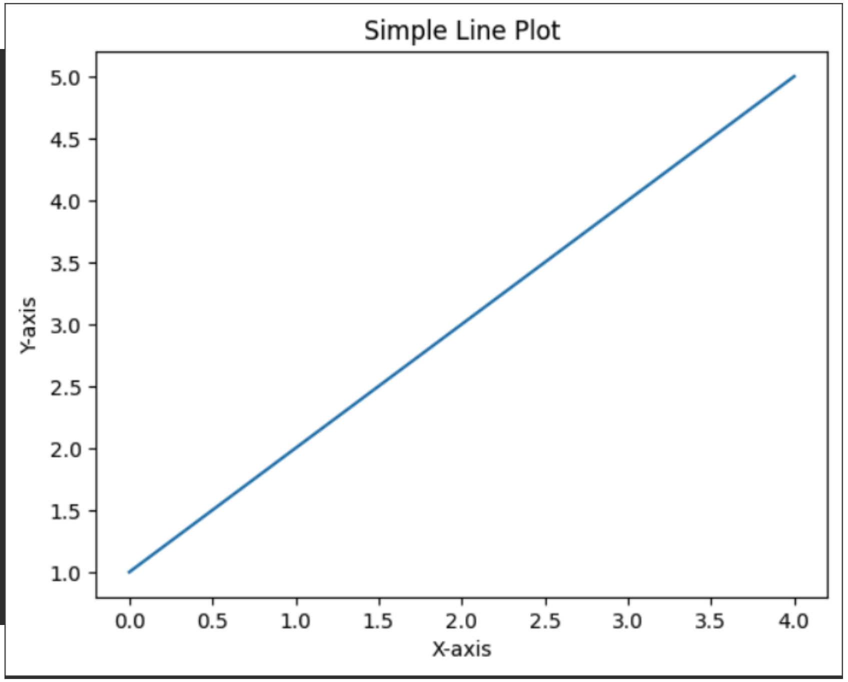
```python
# import the NumPy library
import numpy as np

# create a NumPy array
arr = np.array([1, 2, 3, 4, 5])

# perform element-wise operations
arr2 = arr * 2
```

```
[ 2  4  6  8 10]
```

1.3 Python Libraries for Data Cleaning and Preprocessing (Cont')

- **Python** is a preferred language for many data scientists, mainly because of its ease of use, and extensive, feature-rich libraries dedicated to data tasks
  - **MATPLOTLIB:** Matplotlib is a Python plotting library that can create a variety of **different plots**, such as line, bar, scatter, and others. It's a foundational library for data visualization in Python and is often used to generate plots for exploratory data analysis (EDA) and to diagnose data quality issues.

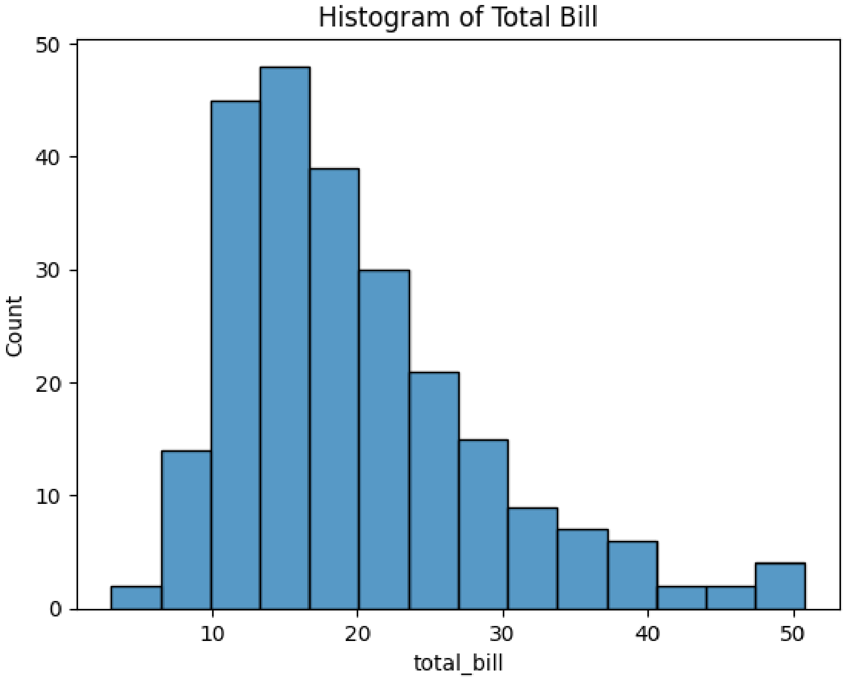
```python
# import the matplotlib library
import matplotlib.pyplot as plt

# create a simple line plot
plt.plot([1, 2, 3, 4, 5])
plt.title("Simple Line Plot")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()
```



1.3 Python Libraries for Data Cleaning and Preprocessing (Cont')

- **Python** is a preferred language for many data scientists, mainly because of its ease of use, and extensive, feature-rich libraries dedicated to data tasks
  - **Seaborn:** Seaborn is a statistical data visualization library built **on top of Matplotlib**. It provides a high-level interface for drawing attractive and informative statistical graphics. With Seaborn, you can create beautiful, rich visualizations with just a few lines of code.

```python
# import the seaborn library
import seaborn as sns

# load an example dataset from seaborn
df = sns.load_dataset('tips')

# create a histogram
sns.histplot(df['total_bill'])
plt.title("Histogram of Total Bill")
plt.show()
```



1.3 Python Libraries for Data Cleaning and Preprocessing (Cont')

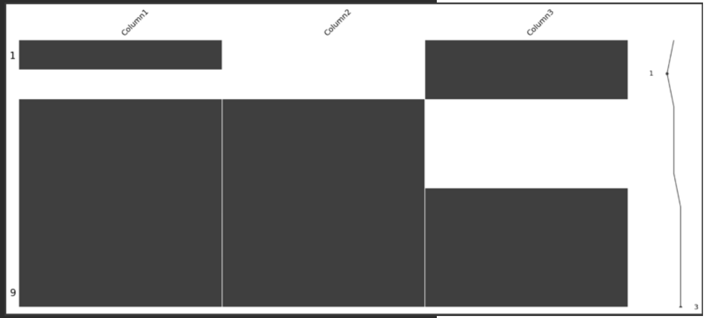
- **Python** is a preferred language for many data scientists, mainly because of its ease of use, and extensive, feature-rich libraries dedicated to data tasks
  - **Scikit-learn:** Scikit-learn is a powerful Python library for machine learning. It provides a range of supervised and unsupervised learning algorithms. Additionally, it includes various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.

```python
# Import scikit-learn libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()

# Create feature and target arrays
X = iris.data
y = iris.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a RandomForestClassifier and fit the model
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Check the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

1.3 Python Libraries for Data Cleaning and Preprocessing (Cont')

- **Python** is a preferred language for many data scientists, mainly because of its ease of use, and extensive, feature-rich libraries dedicated to data tasks
  - **Missingno:** Missingno is a library in Python that provides the ability to visualize the distribution of missing values. This can be particularly useful during the data cleaning process.

```
# import missingno library
import missingno as msno

# create a sample dataframe with missing values
df = pd.DataFrame({'Column1': [1, np.nan, 3, 4, 5],
                   'Column2': [np.nan, np.nan, 7, 8, 9],
                   'Column3': [10, 11, np.nan, np.nan, np.nan]})

# visualize missing values
msno.matrix(df)
plt.show()
```

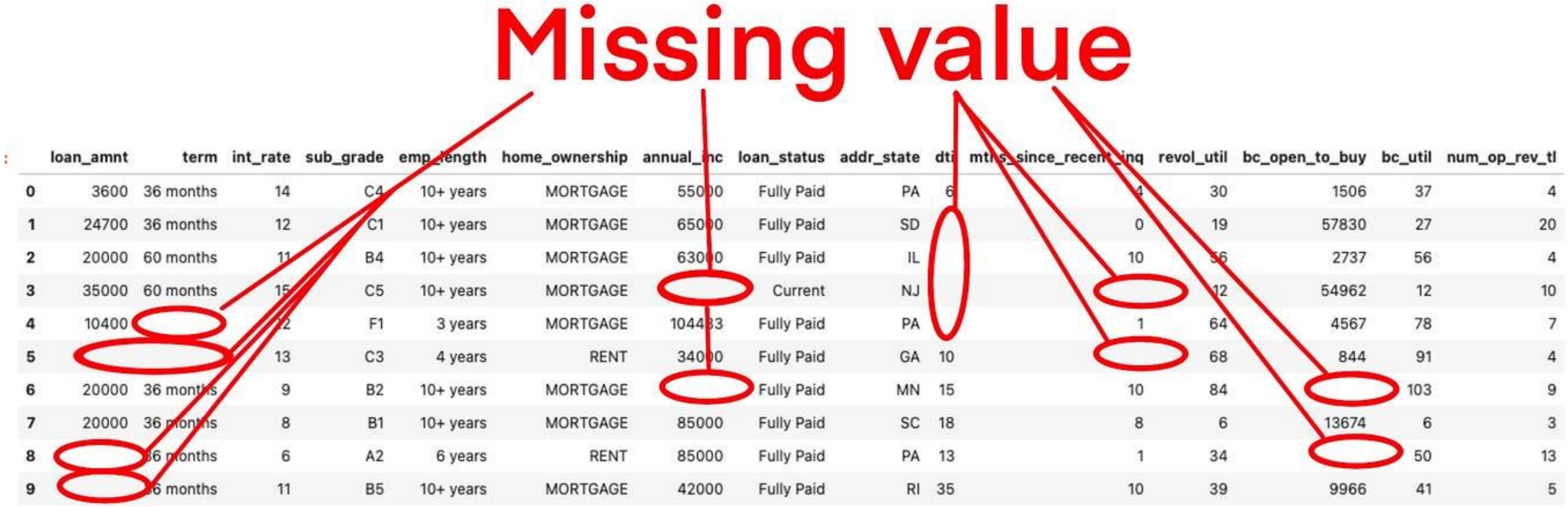


Section 2:

Understanding Data Quality Issues

2.1 Missing Values

- Missing values are the **ghosts** of data science — there, but not there.
- It is when our dataset contains a blank or values that indicate emptiness of the data attribute.
- These arise due to a variety of reasons such as human error during data entry, issues with data collection processes, or instances where certain data fields are deemed not applicable.
- Missing values can lead to **skewed analyses** or introduce bias in your models.



	loan_amnt	term	int_rate	sub_grade	emp_length	home_ownership	annual_inc	loan_status	addr_state	dti	mths_since_recent_inq	revol_util	bc_open_to_buy	bc_util	num_op_rev_tl
0	3600	36 months	14	C4	10+ years	MORTGAGE	55000	Fully Paid	PA	6	4	30	1506	37	4
1	24700	36 months	12	C1	10+ years	MORTGAGE	65000	Fully Paid	SD		0	19	57830	27	20
2	20000	60 months	11	B4	10+ years	MORTGAGE	63000	Fully Paid	IL		10	56	2737	56	4
3	35000	60 months	15	C5	10+ years	MORTGAGE		Current	NJ			12	54962	12	10
4	10400		[illegible]	F1	3 years	MORTGAGE	104433	Fully Paid	PA		1	64	4567	78	7
5			13	C3	4 years	RENT	34000	Fully Paid	GA	10		68	844	91	4
6	20000	36 months	9	B2	10+ years	MORTGAGE		Fully Paid	MN	15	10	84		103	9
7	20000	36 months	8	B1	10+ years	MORTGAGE	85000	Fully Paid	SC	18	8	6	13674	6	3
8		36 months	6	A2	6 years	RENT	85000	Fully Paid	PA	13	1	34		50	13
9		36 months	11	B5	10+ years	MORTGAGE	42000	Fully Paid	RI	35	10	39	9966	41	5

2.1 Missing Values (Cont')

- Missing values are the **ghosts** of data science — there, but not there.
- It is when our dataset contains a blank or values that indicate emptiness of the data attribute.
- These arise due to a variety of reasons such as human error during data entry, issues with data collection processes, or instances where certain data fields are deemed not applicable.
- Missing values can lead to **skewed analyses** or introduce bias in your models.

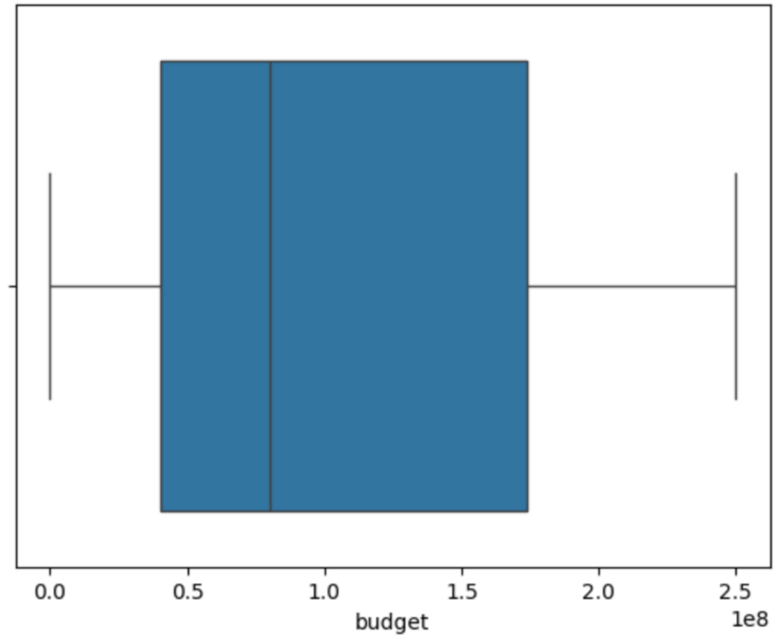
```python
# Importing necessary library
import pandas as pd

# Loading your dataset
df = pd.read_csv('your_file.csv')  # Replace 'your_file.csv' with your filename

# Checking for missing values in each column
missing_values = df.isnull().sum()
print(missing_values)
```

```
color                 11
director_name         11
duration               0
gross                  8
genres                 1
movie_title            0
title_year             0
language               0
country                0
budget                 4
imdb_score             0
actors                 0
movie_facebook_likes   0
dtype: int64
```

2.2 Outliers

- Outlier is like the **black sheep** in your data.
- Outliers are data points that differ significantly from other observations in your dataset.
- They can occur due to measurement errors, data entry errors, or they could be valid but extreme observation.

```
# Importing necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Visualizing outliers using a box plot
sns.boxplot(x=df['your_column'])  # Replace 'your_column' with your column of interest
plt.show()
```



2.3 Inconsistent Formatting

- In an ideal world, all data would follow a consistent format, making a data scientist's life much easier. Unfortunately, that is rarely the case.
- Inconsistent data formatting is a common quality issue that arises due to human errors, system changes, or merging data from multiple sources.

```
# Example: Converting a column with numeric values stored as strings to numeric format
df['numeric_column'] = pd.to_numeric(df['numeric_column'], errors='coerce')
```

```
0     100000000.0
1     200000000.0
2      44000000.0
3      14000000.0
4     225000000.0
          ...
94     20000000.0
95            NaN
96     55000000.0
97     68000000.0
98     40000000.0
Name: budget, Length: 99, dtype: float64
```

- The command converts the value in **`numeric_column`** to a numeric format, converting non-numeric values to **`NaN`**.

2.4 How to Assess Data Quality and Integrity

- Overall data quality should be assessed after the identification of issues.
  - **High-quality data** is complete, accurate, and consistently formatted.
  - **Low-quality data** is rife with errors, missing values, and inconsistencies.
- Data quality assessing is **a must-do preliminary step** before any analysis and preprocessing.

```
# Using pandas to describe the dataset, giving us a sense of data quality
df.describe()
```

- One simple yet powerful tool for data quality assessment is using **descriptive statistics**. The `.describe()` method in Python provides *count*, *mean*, *standard deviation*, *minimum*, *25th percentile*, *median*, *75th percentile*, and *maximum* of the columns

2.4 How to Assess Data Quality and Integrity (Cont')

```
# Using pandas to describe the dataset, giving us a sense of data quality
df.describe()
```

	duration	gross	title_year	budget	imdb_score	movie_facebook_likes
count	99.000000	9.100000e+01	99.000000	9.500000e+01	99.000000	99.000000
mean	155.494949	1.541914e+08	1976.444444	1.048570e+08	6.892929	66045.707071
std	72.797927	1.399503e+08	255.880601	7.703169e+07	1.925514	58108.860365
min	-50.000000	4.122900e+04	202.000000	1.735000e+04	-7.500000	0.000000
25%	138.500000	4.720632e+07	2012.000000	4.000000e+07	6.550000	25000.000000
50%	143.000000	1.156040e+08	2013.000000	8.000000e+07	7.200000	54000.000000
75%	155.000000	2.374894e+08	2014.000000	1.740000e+08	7.850000	85500.000000
max	650.000000	6.232795e+08	2016.000000	2.500000e+08	8.800000	349000.000000

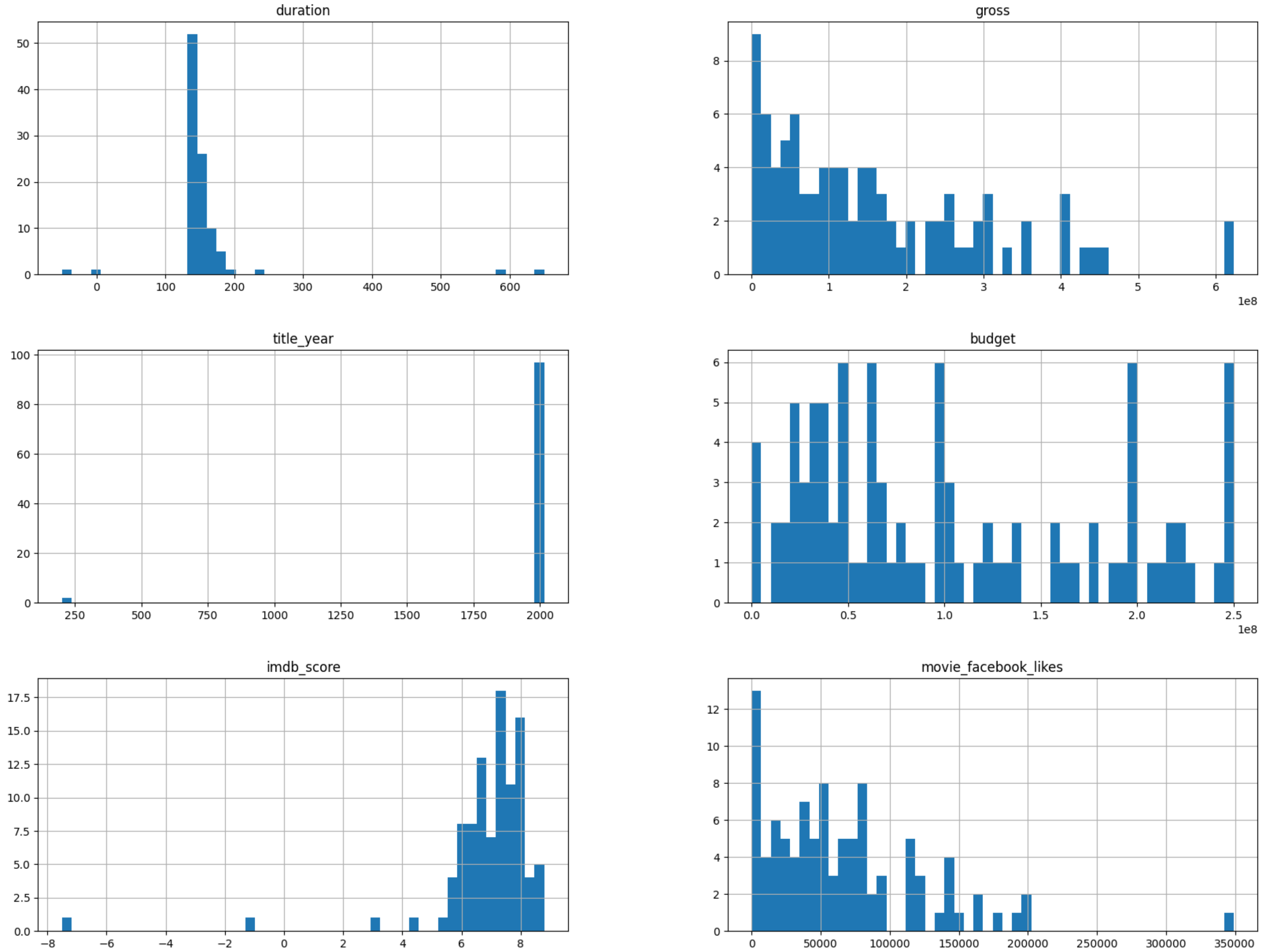
2.5 EDA: Exploratory Data Analysis

- **Exploratory Data Analysis (EDA)** is an approach to analyzing datasets to summarize their main characteristics, often using statistical graphics and other data visualization methods.
- It enables you to understand the data, derive insights, and generate hypotheses.
- One of the significant aspects of EDA is visual exploration. Visualizing your data can provide insights that might not be evident from just looking at tables of data.

```
# Plotting histograms for all numerical columns in the dataset
df.hist(bins=50, figsize=(20,15))
plt.show()
```

2.5 EDA: Exploratory Data Analysis (Cont')

```python
# Plotting histograms for all numerical columns
df.hist(bins=50, figsize=(20,15))
plt.show()
```



2.6 Handling Duplicates and Redundant Data

- **Duplicates** are **repeated records** in your data. They can bias your analysis and lead to incorrect conclusions.
- **Redundant data** are data that **do not add any new information**. While not harmful like duplicates, they can slow down your computations and take up unnecessary storage space.

```
# Check for duplicate rows
duplicate_rows = df.duplicated()

# Count of duplicate rows
print(f"Number of duplicate rows: {duplicate_rows.sum()}")

# Drop the duplicates
df = df.drop_duplicates()

# Checking the shape of the data after dropping duplicates
print("Shape of DataFrame After Removing Duplicates: ", df.shape)
```

```
0     False
1     False
2     False
3     False
4     False
      ...
94    False
95    False
96    False
97    False
98    False
Length: 99, dtype: bool
Number of Duplicate Rows: 6
Shape of Data After Dropping Duplicates: (93, 13)
```

Section 3:

Handling Missing Data

3.1 Identifying and Understanding Missing Data

- Identifying missing data might seem **straightforward** — You look for the gaps. But in real-world data, it's rarely so simple.

	Height	Weight	Country	Place	Number of days	Some column
0	12.0	35.0	India	Bengaluru	1.0	NaN
1	NaN	36.0	US	New York	2.0	NaN
2	13.0	32.0	UK	London	NaN	NaN
3	15.0	NaN	France	Paris	4.0	NaN
4	16.0	39.0	US	California	5.0	12.0
5	NaN	NaN	NaN	Mumbai	NaN	NaN
6	NaN	NaN	NaN	NaN	6.0	NaN

10	Louise	Female	8/12/1980	9:01 AM	63241	15.132
11	Julie	Female	10/26/1997	3:19 PM	102508	12.637
12	Brandon	Male	12/1/1980	1:08 AM	112807	17.492
13	Gary	Male	1/27/2008	11:40 PM	109831	5.831
14	Kimberly	Female	1/14/1999	7:13 AM	41426	14.543
15	Lillian	Female	6/5/2016	6:09 AM	59414	1.256
16	Jeremy	Male	9/21/2010	5:56 AM	90370	7.369
17	Shawn	Male	12/7/1986	7:45 PM	111737	6.414
18	Diana	Female	10/23/1981	10:27 AM	132940	19.082
19	Donna	Female	7/22/2010	3:48 AM	81014	1.894
20	Lois	-99	4/22/1995	7:18 PM	64714	4.934
21	Matthew	Male	9/5/1995	2:12 AM	100612	13.645
22	Joshua	-99	3/8/2012	1:58 AM	90816	18.816
23	-99	Male	6/14/2012	4:19 PM	125792	5.042
24	John	Male	7/1/1992	10:08 PM	97950	13.873

VIEWTABLE: Work.Sample

	StudentID	Gender	DOB	Race	Ethnicity	Class	Weight	Height	Enrollment_Date	State_Residency
1	5	1	08/15/1991	2	1	1	226	70	08/15/2012	In state
2	9	1	11/01/1991	3	1	1	144	71	08/15/2012	
3	35	1	10/29/1990	1	.	1	.	.	08/15/2012	Out of state
4	70	2	04/06/1994	1	2	1	175	63	08/15/2012	In state
5	44	1	01/31/1991	1	2	2	170	77	.	In state
6	51	1	.	1	1	2	177	71	08/15/2011	Out of state
7	85	2	09/26/1991	[illegible]	[illegible]	2	141	[illegible]	[illegible]	Out of state
8	19	1	05/25/1991	[illegible]	[illegible]	3	184	[illegible]	[illegible]	In state
9	40	1	10/29/1990	1	2	3	170	67	08/15/2010	In state
10	43	1	02/03/1990	2	2	3	.	.	08/15/2010	Out of state
11	24	1	09/04/1993	1	2	4	167	73	08/15/2007	In state
12	39	1	08/12/1993	3	2	4	150	73	08/15/2006	Out of state
13	45	1	03/09/1994	1	2	4	161	71	08/15/2007	In state
14	79	2	02/16/1992	1	2	4	143	62	08/15/2008	In state
15	89	.	09/11/1993	1	2	4	128	64	08/15/2009	Out of state

Missing numeric values are a period.

Missing character values are blank.

3.1 Identifying and Understanding Missing Data (Cont')

- Missing data can take various forms, from obvious blanks to placeholders like "N/A" or "-999", or even mis-entered data.

```
# Importing necessary library
import pandas as pd

# Loading your dataset
df = pd.read_csv('your_file.csv')  # Replace 'your_file.csv' with your filename

# Checking for missing values in each column
missing_values = df.isnull().sum()
print(missing_values)
```

```
color                   11
director_name           11
duration                 0
gross                    8
genres                   1
movie_title              0
title_year               0
language                 0
country                  0
budget                   4
imdb_score               0
actors                   0
movie_facebook_likes     0
dtype: int64
```

- This Python script is powerful, so it can also detect some common placeholders like "**NaN**" in the dataset.

3.1 Identifying and Understanding Missing Data (Cont')

- Understanding missing data requires recognizing its different types.
- In statistics, missing data are typically categorized into **three main types**:
  - **Missing Completely at Random (MCAR):** The missingness of data is not related to any other variable in the dataset; it occurs purely at random.
    - An example of MCAR is a weighing scale that runs out of batteries. In this case, some data are missing purely due to chance.
  - **Missing at Random (MAR):** The missingness of a variable is related to other variables in the dataset, but not to the variable itself.
    - For example, when placed on a soft surface, a weighing scale may generate more missing values than when placed on a hard surface. Such data are therefore not MCAR.
  - **Missing Not at Random (MNAR):** The missingness of a variable is related to the variable itself.
    - For example, the weighing scale mechanism may deteriorate over time, leading to an increasing amount of missing data as time progresses.

3.2 Techniques for Handling Missing Data

- 1. Deletion
  - This is the simplest method, which involves deleting the records with missing values.
  - However, it's only advisable when the data is MCAR and the missing data is a small fraction of the total dataset.

```
# Drop rows with missing values
df.dropna(inplace=True)
```

- 2. Imputation
  - Imputation is the process of substituting missing data with substituted values.
  - **2.1 Mean/Median/Mode Imputation:** This method involves replacing missing values with the mean (for continuous data), the median (for ordinal data), or the mode (for categorical data). However, it may reduce variance and affect correlations with other variables.

```
# Mean imputation
df.fillna(df.mean(), inplace=True)
```

3.2 Techniques for Handling Missing Data (Cont')

- 2. Imputation
  - Imputation is the process of substituting missing data with substituted values.
  - **2.2 Constant Value Imputation:** This method involves replacing missing values with a constant and is useful when an educated guess about the missing values can be made.

```
# Constant value imputation
df.fillna(0, inplace=True)
```

  - **2.3 Predictive Imputation:** This technique uses statistical models or machine learning algorithms to predict missing values based on other available data. While it is generally more accurate, it is also more complex.

```
# Predictive imputation using linear regression
from sklearn.linear_model import LinearRegression

# Split data into sets with missing values and without
missing = df[df['A'].isnull()]
not_missing = df[df['A'].notnull()]

# Initialize the model
model = LinearRegression()
```

```
# Train the model
model.fit(not_missing.drop('A', axis=1), not_missing['A'])

# Predict missing values
predicted = model.predict(missing.drop('A', axis=1))

# Fill in missing values
df.loc[df['A'].isnull(), 'A'] = predicted
```

3.3 Advanced Missing Data Handling Techniques

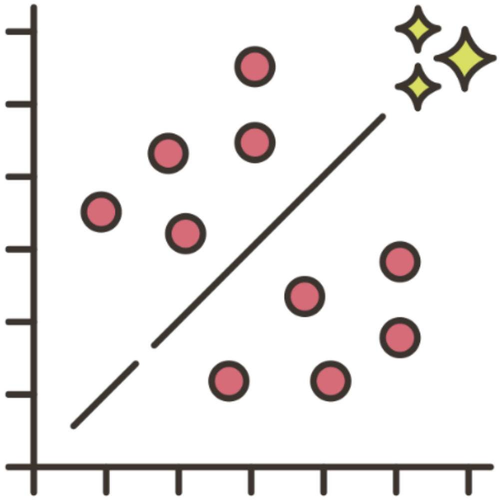
- 1. Multiple Imputation
  - Multiple imputation is a statistical technique for handling missing data, in which each missing value is estimated multiple times to create several complete datasets.
  - This process produces multiple complete datasets, each of which is analyzed, and the results are then pooled to generate a single final outcome.
  - One of the most common methods for multiple imputation is **Multivariate Imputation by Chained Equations (MICE)**, which accounts for the uncertainty surrounding a missing value.

```
# Multiple Imputation by Chained Equations
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Initialize the MICE imputer
mice_imputer = IterativeImputer()

# Apply the imputer
df_imputed = mice_imputer.fit_transform(df)
```

3.3 Advanced Missing Data Handling Techniques (Cont')

- 2. Predictive Imputation
  - While a simple linear regression may be **sufficient** in some cases, more sophisticated methods—like **decision trees, random forests, or even neural networks**—can yield better results depending on the complexity of your data.

```python
# Predictive imputation using random forest
from sklearn.ensemble import RandomForestRegressor

# Prepare data
missing = df[df['A'].isnull()]
not_missing = df[df['A'].notnull()]

# Initialize the model
model = RandomForestRegressor(n_estimators=100, random_state=0)

# Train the model
model.fit(not_missing.drop('A', axis=1), not_missing['A'])

# Predict missing values
predicted = model.predict(missing.drop('A', axis=1))

# Fill in missing values
df.loc[df['A'].isnull(), 'A'] = predicted
```

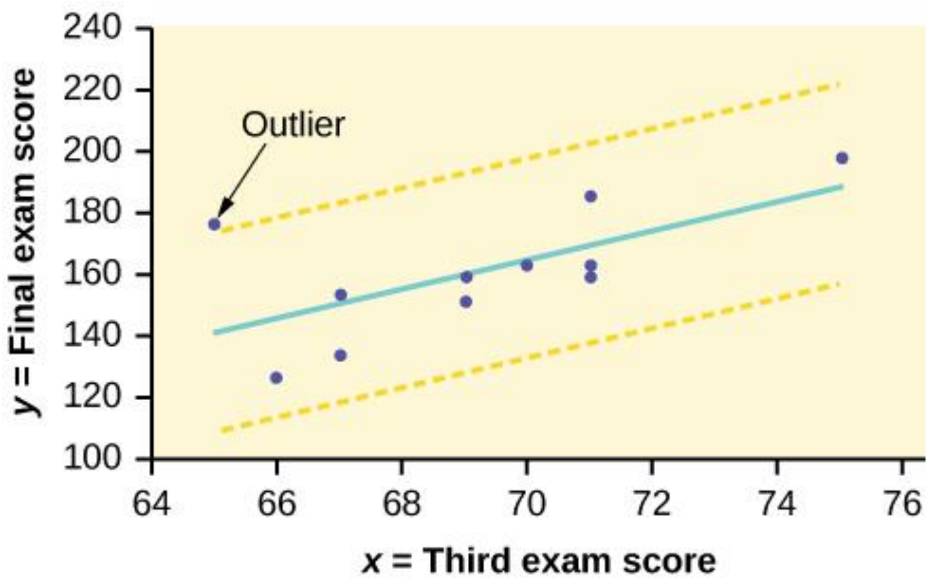


Section 4:

Dealing with Outliers

4.1 Understanding Outliers and Their Impacts

- Outliers are unusual observations that significantly **differ** from the rest of the data.
- While they can sometimes indicate important findings or errors in data collection, they can also **skew** the data and lead to misleading results.
- **Outliers** can arise from various sources, including measurement errors, data processing errors, or true anomalies (e.g., a major event that disrupts the usual process).



Their Impacts:

- **Affect Mean and Standard Deviation:** They can significantly skew your mean and inflate the standard deviation, which distorts the overall data distribution.
- **Impact Model Accuracy:** Many machine learning algorithms are sensitive to the range and distribution of data, and outliers can mislead the training process. This often results in longer training times and less accurate models.

4.1 Understanding Outliers and Their Impacts (Cont')

- Let's demonstrate how outliers can skew the mean using a simple Python example:

```
import numpy as np

# Regular data
regular_data = np.array([10, 20, 30, 40, 50])
print(f'Mean of regular data: {regular_data.mean()}')

# Data with an outlier
outlier_data = np.array([10, 20, 30, 40, 500])  # 500 is an outlier
print(f'Mean of data with an outlier: {outlier_data.mean()}')
```

```
Mean of regular data: 30.0
Standard Deviation of regular data: 14.142135623730951
Mean of outlier data: 1020.0
Standard Deviation of outlier data: 1990.0251254695254
```

4.2 Outlier Detection Techniques

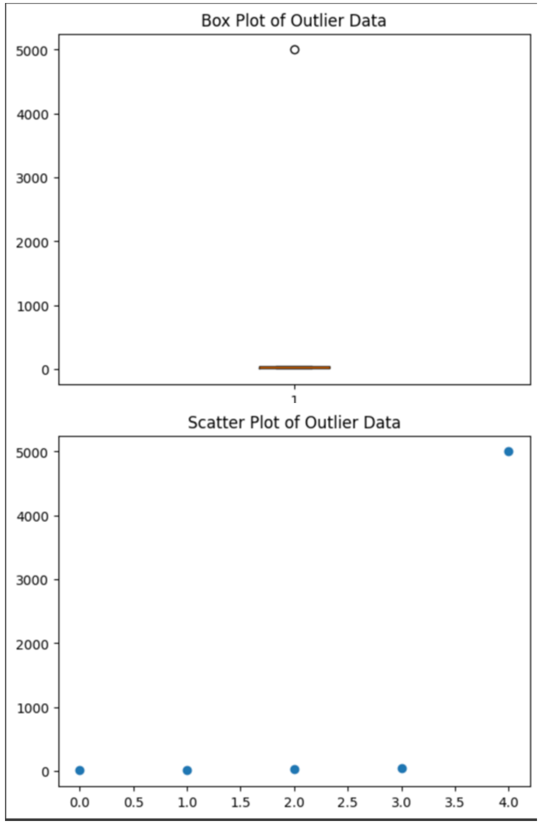
- **1. Statistical Methods**
  - **Z-Score:** The Z-score measures how many standard deviations an observation is from the mean. A common rule of thumb is that a data point with a **Z-score** greater than **3** or less than **-3** is considered an **outlier**.

```
from scipy import stats

z_scores = np.abs(stats.zscore(outlier_data))
outliers = outlier_data[(z_scores > 3)]
```

  - **Interquartile Range (IQR) method:** It identifies data points as **outliers** if they fall below the **first quartile** (Q1) or above the **third quartile** (Q3) by a certain factor of the IQR. A commonly used factor for this method is **1.5**.

```
Q1 = np.percentile(outlier_data, 25)
Q3 = np.percentile(outlier_data, 75)
IQR = Q3 - Q1

outliers = outlier_data[((outlier_data < (Q1 - 1.5 * IQR)) | (outlier_data > (Q3 +
1.5 * IQR)))]
```

4.2 Outlier Detection Techniques (Cont')



- **2. Visualization**
  - Box plots and scatter plots are great tools for visualizing and detecting outliers.

```
import matplotlib.pyplot as plt

# Boxplot
plt.boxplot(outlier_data)
plt.show()

# Scatter plot
plt.scatter(range(len(outlier_data)), outlier_data)
plt.show()
```

- **3. Machine Learning**
  - Certain machine learning algorithms, like DBSCAN and Isolation Forest, are particularly good at detecting outliers.

```
from sklearn.ensemble import IsolationForest

# Initialize the model
clf = IsolationForest(contamination=0.01)

# Fit the model
pred = clf.fit_predict(outlier_data.reshape(-1, 1))

# Outliers are marked with a -1
outliers = outlier_data[pred == -1]
```

4.3 Strategies for Handling Outliers

- **1. Deletion**
  - **Deletion**, or dropping, is the simplest way to handle outliers, but it should be used with **caution**.
  - You should only **drop an outlier** if you are **certain** that it is due to **incorrectly entered or measured data**.
  - **Deleting valuable data points** can lead to a **loss of information** and **biased results**.

```
# Filter out the outliers
filtered_data = outlier_data[(z_scores <= 3)]
```

- **2. Transformation**
  - Transforming variables can also help to minimize the impact of outliers.
  - Transformations such as **log, square root, and inverse** can compress higher values, thereby reducing the **effect** of extreme values.

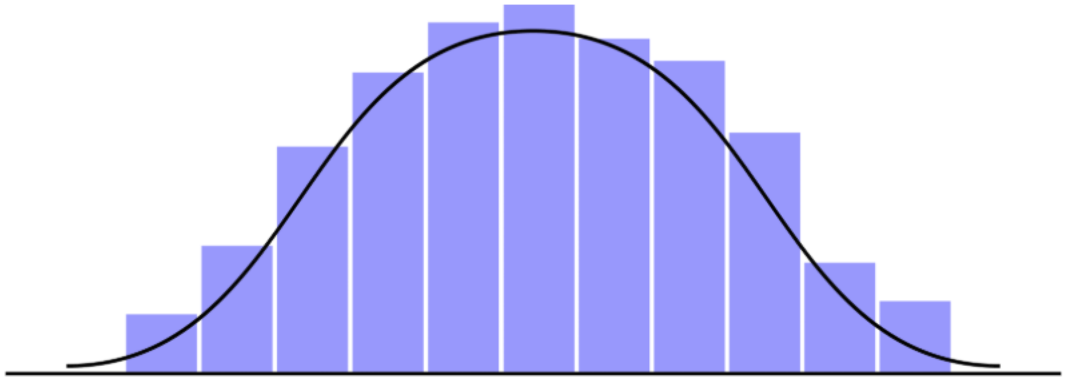
```
# Apply log transformation
log_data = np.log(outlier_data)
```

4.3 Strategies for Handling Outliers (Cont')

- **3. Winsorization**
  - In a **winsorized dataset**, **extreme values** are replaced by specific **percentiles** (typically the **5th and 95th**). Unlike deletion, this technique **maintains the size** of the dataset.

```
from scipy.stats.mstats import winsorize

# Apply winsorization
winsorized_data = winsorize(outlier_data, limits=[0.05, 0.05])
```

- **4. Selecting Machine Learning Models**
  - Some machine learning models, such as **Random Forests** and **SVMs**, are **less sensitive to outliers**.
  - Using these models can therefore be a **viable strategy** for handling them.

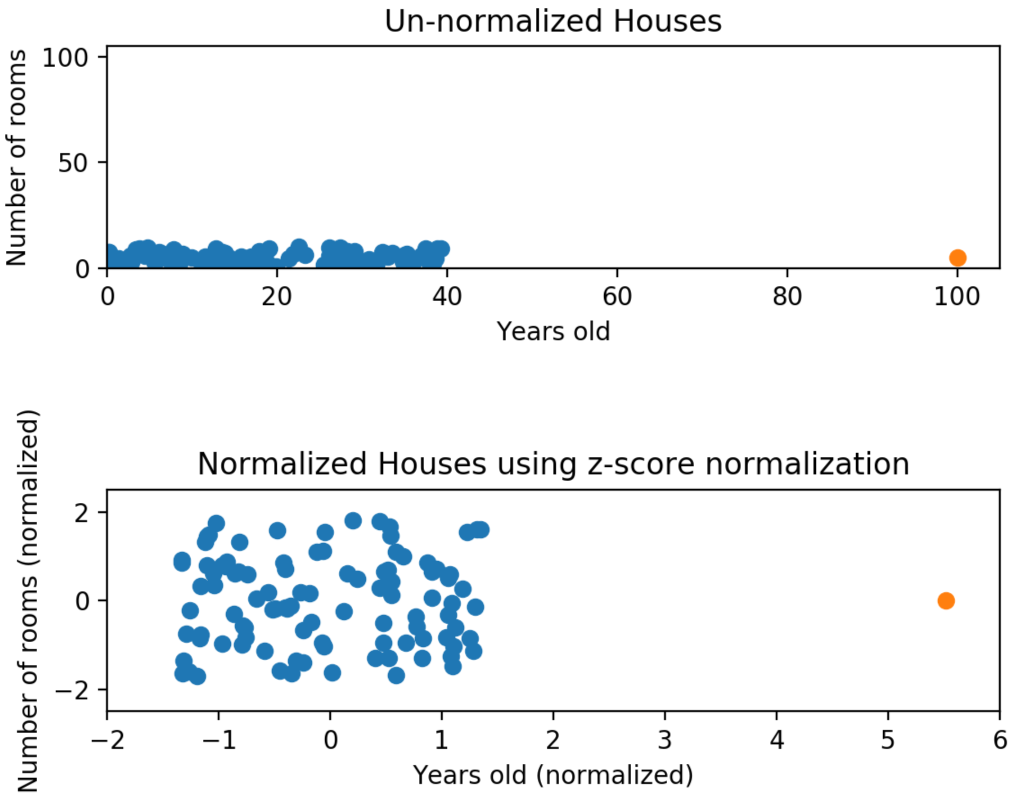


Section 5:

Data Normalization and Scaling

5.1 Understanding the Importance of Data Normalization

- Data normalization and scaling help standardize the range of a dataset's independent variables or features.
- Machine learning algorithms often perform **better** when input **numerical variables** are on a **similar scale**.
- Without normalization or scaling, features with **higher values** may **dominate the model's outcome**. This could lead to **misleading results** and a model that fails to capture the influence of other features.
- Normalization and scaling bring **different features** to the **same scale**, which allows for a **fair comparison** and ensures that no single feature **dominates others**.



5.2 Techniques for Data Normalization

- **1. Min-Max Scaling**
  - **Min-max scaling** is one of the **simplest methods** for data normalization. It individually **scales and translates** each feature so that it is within a **range of 0 to 1**.

```
from sklearn.preprocessing import MinMaxScaler

# Create a simple dataset
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)

# Create a scaler, fit and transform the data
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(data)
```

- **2. Z-Score Normalization (Standardization)**
  - This technique **standardizes** features to have a **mean of 0** and a **standard deviation of 1**.

```
from sklearn.preprocessing import StandardScaler

# Create a scaler, fit and transform the data
scaler = StandardScaler()
standardized_data = scaler.fit_transform(data)
```

5.3 Feature Scaling Techniques

- Feature scaling is an umbrella term for techniques that change the range of a feature.
- **1. Robust Scaling**
  - **Robust scaling** is similar to **min-max scaling**, but it uses the **interquartile range** instead of the min-max range, which makes it more **robust to outliers**.

```
from sklearn.preprocessing import RobustScaler

# Create a scaler, fit and transform the data
scaler = RobustScaler()
robust_scaled_data = scaler.fit_transform(data)
```

🥳

End of the Lecture

Please don’t hesitate to raise your hand and ask questions if you're curious about anything!