CLOUD APPLICATION DEVELOPMENT

E-commerce Application

E-commerce, or electronic commerce, refers to buying and selling goods and services over the internet. It has become a significant part of the global economy, with businesses and consumers engaging in online transactions through websites and mobile apps. E-commerce offers convenience, a wide variety of products, and the ability to reach a global audience. Popular examples include Amazon, eBay, and Alibaba. Types of E-commerce: B2C (Business-to-Consumer): This is the most common form, where businesses sell products or services directly to individual consumers. Examples include Amazon and Walmart. B2B (Business-to-Business): In this type, businesses sell products or services to other businesses. Transactions often involve larger quantities and longer-term contracts. C2C (Consumer-to-Consumer): Individuals sell to other individuals, often through online marketplaces like eBay or Craigslist. C2B (Consumer-to-Business): This is less common but involves consumers selling products or services to businesses. Freelancers offering their services online is an example. Advantages of E-commerce: ~Convenience: Shoppers can browse and make purchases from the comfort of their homes 24/7. ~Global Reach: E-commerce enables businesses to reach a worldwide audience, expanding their market potential. ~Cost Savings: Online businesses often have lower overhead costs compared to brick-and-mortar stores. ~Personalization: E-commerce platforms can use data to tailor recommendations and marketing to individual preferences. ~Analytics: Detailed data collection allows businesses to analyze customer behavior and make data-driven decisions. Challenges of E-commerce: ~Security: Online transactions must be secure to protect customer data and prevent fraud. ~Competition: E-commerce is highly competitive, with numerous businesses vying for customers' attention. ~Logistics: Efficient shipping and delivery are crucial, and managing inventory can be complex. ~Customer Trust: Building trust with online customers is essential for success. ~Mobile Commerce (M-commerce): With the growth of smartphones, mobile shopping and payment apps have become increasingly popular. ~AI and Personalization: AI is used to enhance the shopping experience, from chatbots for customer support to personalized product recommendations. ~AR and VR: Augmented Reality (AR) and Virtual Reality (VR) technologies are being used for immersive shopping experiences. ~E-commerce Regulations: Different regions have various regulations governing online commerce, including tax laws, data privacy regulations, and consumer protection laws. E-commerce continues to evolve with innovations like drone delivery, blockchain for supply chain transparency, and the integration of online and offline shopping experiences (omnichannel retail). E-commerce has transformed the way people shop and do business, and its significance in the global economy is likely to continue growing in the future. Product Review: Rating: ⭐⭐⭐⭐⭐ (5/5 stars) I recently started using E- Daily Things, and it has truly transformed my grocery shopping experience. Here's why I love it: 1. User-Friendly Interface: The app's intuitive design makes it a breeze to navigate. I can quickly find what I need, whether it's fresh produce, pantry staples, or household items. 2. Extensive Product Range: E- Daily Things offers an impressive selection of products. I can easily locate both popular brands and specialty items, ensuring I can

complete my entire shopping list in one place. 3. Time-Saving Features: The app's search function and personalized recommendations are a time-saver. I appreciate how it suggests items based on my previous purchases, making reordering a snap. 4. Flexible Delivery Options: Whether I need groceries ASAP or want to schedule a delivery for a specific time, E- Daily Things offers flexible delivery options that cater to my needs. 5. Freshness Guarantee: I've always received fresh and high-quality products through this app.They have a robust freshness guarantee, and their customer support is excellent if I ever encounter an issue. 6. Savings and Deals: E- Daily Things frequently offers discounts, promotions, and loyalty rewards, helping me save money on my grocery bills. 7. Timely Customer Support: Any questions or concerns are promptly addressed by their customer support team. They are responsive and attentive to customer needs. Wishlist: Key functionalities for wishlist:- 1. User Accounts: Allow users to create accounts to save their wishlists across sessions. 2. Add to Wishlist Button: Include a button on each product page that allows users to add items to their wishlist. 3.View Wishlist: Provide an easily accessible option for users to view and manage their wishlist. 4. Remove Items: Allow users to remove items from their wishlist. 5. Quantity Selection: Let users specify quantities for wishlist items. 6. Share Wishlist: Enable users to share their wishlist with others through email or social media. 7. Notifications: Send notifications or emails when items in the wishlist go on sale or come back in stock. 8. Privacy Settings: Give users the option to set their wishlist as public or private. 9. Recently Viewed Items: Consider adding a section that displays recently viewed items to encourage users to add them to their wishlist. 10. Mobile Responsiveness: Make sure the wishlist is user-friendly on mobile devices. 11. Security: Protect user data and privacy by implementing secure authentication and authorization mechanisms. 12. Feedback Mechanism: Allow users to provide feedback on the wishlist feature for continuous improvements. Personalized recommendations: 1. User Profiling: Create detailed user profiles by collecting data on their browsing history, purchase history, demographics, and preferences. 2. Collaborative Filtering: Use collaborative filtering algorithms to suggest products based on what similar users have purchased or viewed. 3. Content-Based Filtering: Recommend products by analyzing the attributes of items users have interacted with and suggesting similar items. 4. Machine Learning: Implement machine learning models to predict user preferences and behavior over time, refining recommendations as users engage with the platform. 5. Real-Time Updates: Keep recommendations updated in real-time to reflect changing user preferences and trends. 6. Personalized Email Campaigns: Send personalized email recommendations based on user behavior and preferences, including abandoned cart reminders. 7. Cross-Selling and Up-Selling: Suggest related or higher-priced items when users view or purchase products, increasing the average order value. 8. Seasonal and Trend-Based Recommendations: Highlight trending or seasonal products to encourage users to explore new items. 9. User Feedback Integration: Allow users to provide feedback on recommended products to refine future recommendations. 10. Segmentation: Group users into segments based on behavior or demographics and tailor recommendations accordingly. 11. Personalized Homepage: Customize the homepage for each user, showcasing products and categories they are likely to be interested in. 12. Social Integration: Allow users to connect their social media accounts to the app for more personalized recommendations based on their social network's preferences. 1 3. P e rfo r m a n c e O p timiz a tio n: O p timiz e a p p p e rfo r m a n c e t o d eliv e r r e c o m m e n d a tio n s q uic kly a n d s e a mle s sly.

Project for Phase-3
E-COMMERCE APPLICATION:

```
# Importing pandas library
import pandas as pd
# Create a custom dataset with duplicate rows
data = pd.DataFrame({'name': ['John', 'Emily', 'Peter', 'John', 'Emily'],
'age': [20, 25, 30, 20, 25],
'income': [50000, 60000, 70000, 50000, 60000]})
# Check for duplicate rows
duplicates = data[data.duplicated()]
# Drop duplicate rows
data_deduplicated = data.drop_duplicates()
# Print the original and deduplicated datasets
print("Original dataset:")
print(data)
print("\nDuplicate rows:")
print(duplicates)
print("\nDeduplicated dataset:")
print(data_deduplicated)
```

The output of the above code is shown below.

Original dataset
name age income
John 20 50000
Emily 25 60000
Peter 30 70000
John 20 50000
Emily 25 60000
Duplicate rows
name age income
John 20 50000
Emily 25 60000
Deduplicated dataset
name age income
John 20 50000
Emily 25 60000
Peter 30 70000

The duplicate rows are removed from the original dataset based on the deduplicated dataset's name, age,
and income columns.

Handing outliers

In real-world data analysis, we often come across data with outliers. Outliers are very small or huge
values that deviate significantly from other observations in a dataset. Such outliers are first identified,
then removed, and the dataset is transformed at a specific scale. Let's understand with the following
detail.

Identifying outliers

As we've already seen, the first step is to identify the outliers in our dataset. Various statistical
techniques can be used for this, such as the interquartile range (IQR), z-score, or Tukey methods.
We'll mainly look at z-score. It's a common technique for the identification of outliers in the dataset.
The z-score measures how many standard deviations an observation is from the mean of the dataset. The
formula for calculating the z-score of an observation is this:
z = (observation - mean) / standard deviation
The threshold for the z-score method is typically chosen based on the level of significance or the desired
level of confidence in identifying outliers. A commonly used threshold is a z-score of 3, meaning any
observation with a z-score more significant than 3 or less than -3 is considered an outlier.
Removing outliers
Once the outliers are identified, they can be removed from the dataset using various techniques such as
trimming, or removing the observations with extreme values. However, it's important to carefully analyze
the dataset and determine the appropriate technique for handling outliers.
Transforming the data
Alternatively, the data can be transformed using mathematical functions such as logarithmic, square root,
or inverse functions to reduce the impact of outliers on the analysis:

```
# Import pandas and numpy libraries
import pandas as pd
import numpy as np
# Create a custom dataset with outliers
data = pd.DataFrame({'age': [20, 25, 30, 35, 40, 200],
'income': [50000, 60000, 70000, 80000, 90000, 100000]})
# Calculate the mean and standard deviation of the data
mean = data.mean()
std_dev = data.std()
# Identify outliers using the z-score method
threshold = 3
z_scores = ((data - mean) / std_dev).abs()
outliers = data[z_scores > threshold]
# Remove outliers
data_without_outliers = data[z_scores <= threshold]
# Print the original and cleaned datasets
print("Original dataset:")
print(data)
print("\nOutliers:")
print(outliers)
print("\nDataset without outliers:")
print(data_without_outliers)
```

In this example, we've created a custom dataset with outliers in the age column. We then apply the outlier
handling technique to identify and remove outliers from the dataset. We first calculate the mean and
standard deviation of the data, and then identify the outliers using the z-score method. The z-score is
calculated for each observation in the dataset, and any observation that has a z-score greater than the
threshold value (in this case, 3) is considered an outlier. Finally, we remove the outliers from the dataset.
The output of the above code in table form is shown below.

Original dataset

age	income
20	50000
25	60000
30	70000
35	80000
40	90000
200	100000

Outliers

age	income
200	100000

Dataset without outliers

age	income
20	50000
25	60000
30	70000
35	80000
40	90000

The outlier (200) in the age column in the dataset without outliers is removed from the original dataset.

Data Transformation

Data transformation is another method in data processing to improve data quality by modifying it. This
transformation process involves converting the raw data into a more suitable format for analysis by
adjusting the data's scale, distribution, or format.

Log transformation is used to reduce outliers' impact and transform skewed (a situation where the
distribution of the target variable or class labels is highly imbalanced) data into a normal distribution. It's
a widely used transformation technique that involves taking the natural logarithm of the data.

Square root transformation is another technique to transform skewed data into a normal distribution. It
involves taking the square root of the data, which can help reduce the impact of outliers and improve the
data distribution.

Let’s look at an example:

```
# Import pandas and numpy libraries
import pandas as pd
import numpy as np
# Create a custom dataset
data = pd.DataFrame({'age': [20, 25, 30, 35, 40, 45],
'income': [50000, 60000, 70000, 80000, 90000, 100000],
'spending': [1, 4, 9, 16, 25, 36]})
# Apply square root transformation
data['sqrt_spending'] = np.sqrt(data['spending'])
# Print the original and transformed datasets
print("Original dataset:")
print(data)
print("\nTransformed dataset:")
print(data[['age', 'income', 'sqrt_spending']])
```

In this example, our custom dataset has a variable called spending. A significant outlier in this variable is causing skewness in the data. We’re controlling this skewness in the spending variable. The square root transformation has transformed the skewed spending variable into a more normal distribution.

Transformed values are stored in a new variable called sqrt_spending. The normal distribution of sqrt_spending is between 1.00000 to 6.00000, making it more suitable for data analysis.

The output of the above code in table form is shown below.

Original dataset

age	income	spending
20	50000	1
25	60000	4
30	70000	9
35	80000	16
40	90000	25
45	100000	36

Transformed dataset

age	income	sqrt_spending
20	50000	1.00000
25	60000	2.00000
30	70000	3.00000
35	80000	4.00000
40	90000	5.00000
45	100000	6.00000

Data Integration

The data integration technique combines data from various sources into a single, unified view. This helps to increase the completeness and diversity of the data, as well as resolve any inconsistencies or conflicts that may exist between the different sources. Data integration is helpful for data mining, enabling data

analysis spread across multiple systems or platforms.
Let’s suppose we have two datasets. One contains customer IDs and their purchases, while the other
dataset contains information on customer IDs and demographics, as given below. We intend to combine
these two datasets for a more comprehensive customer behavior analysis.
Customer Purchase Dataset
Customer ID Purchase Amount
1 $50
2 $100
3 $75
4 $200
Customer Demographics Dataset
Customer ID Age Gender
1 25 Male
2 35 Female
3 30 Male
4 40 Female
To integrate these datasets, we need to map the common variable, the customer ID, and combine the
data. We can use the Pandas library in Python to accomplish this:

```
# Import pandas library
import pandas as pd
# Load customer purchase dataset
purchase_data = pd.DataFrame({'Customer ID': [1, 2, 3, 4],
'Purchase Amount': [50, 100, 75, 200]})
# Load customer demographics dataset
demographics_data = pd.DataFrame({'Customer ID': [1, 2, 3, 4],
'Age': [25, 35, 30, 40],
'Gender': ['Male', 'Female', 'Male', 'Female']})
# Merge the two datasets on customer ID
merged_data = pd.merge(purchase_data, demographics_data, on='Customer ID')
# Print the merged dataset
print(merged_data)
```

The output of the above code in table form is shown below.
Customer ID Purchase Amount Age Gender
1 $50 25 Male
2 $100 35 Female
3 $75 30 Male
4 $200 40 Female
We’ve used the merge() function from the Pandas library. It merges the two datasets based on the
common customer ID variable. It results in a unified dataset containing purchase information and
customer demographics. This integrated dataset can now be used for more comprehensive analysis, such
as analyzing purchasing patterns by age or gender.

Data Reduction

Data reduction is one of the commonly used techniques in the data processing. It's used when we have a lot of data with plenty of irrelevant information. This method reduces data without losing the most critical information.

There are different methods of data reduction, such as those listed below.

Data cube aggregation involves summarizing or aggregating the data along multiple dimensions, such as time, location, product, and so on. This can help reduce the complexity and size of the data, as well as reveal higher-level patterns and trends.

Dimensionality reduction involves reducing the number of attributes or features in the data by selecting a subset of relevant features or transforming the original features into a lower-dimensional space. This can help remove noise and redundancy and improve the efficiency and accuracy of data mining algorithms.

Data compression involves encoding the data in a more minor form, by using techniques such as sampling, clustering, histogram analysis, wavelet analysis, and so on. This can help reduce the data's storage space and transmission cost and speed up data processing.

Numerosity reduction replaces the original data with a more miniature representation, such as a parametric model (for example, regression, log-linear models, and so on) or a non-parametric model (such as histograms, clusters, and so on). This can help simplify the data structure and analysis and reduce the amount of data to be mined.

Data preprocessing is essential, because the quality of the data directly affects the accuracy and reliability of the analysis or model. By properly preprocessing the data, we can improve the performance of the machine learning models and obtain more accurate insights from the data.

Conclusion

Preparing data for machine learning is like getting ready for a big party. Like cleaning and tidying up a room, data preprocessing involves fixing inconsistencies, filling in missing information, and ensuring that all data points are compatible. Using techniques such as data cleaning, data transformation, data integration, and data reduction, we create a well-prepared dataset that allows computers to identify patterns and learn effectively.

It's recommended that we explore data in depth, understand data patterns and find the reasons for

missingness in data before choosing an approach. Validation and test set are also important ways to
evaluate the performance of different techniques.
Importing pandas library

```
import pandas as pd
# Create a custom dataset with duplicate rows
data = pd.DataFrame({'name': ['John', 'Emily', 'Peter', 'John', 'Emily'],
'age': [20, 25, 30, 20, 25],
'income': [50000, 60000, 70000, 50000, 60000]})
# Check for duplicate rows
duplicates = data[data.duplicated()]
# Drop duplicate rows
data_deduplicated = data.drop_duplicates()
# Print the original and deduplicated datasets
print("Original dataset:")
print(data)
print("\nDuplicate rows:")
print(duplicates)
print("\nDeduplicated dataset:")
print(data_deduplicated)
```

The output of the above code is shown below.

```
Original dataset
name age income
John 20 50000
Emily 25 60000
Peter 30 70000
John 20 50000
Emily 25 60000
Duplicate rows
name age income
John 20 50000
Emily 25 60000
Deduplicated dataset
name age income
John 20 50000
Emily 25 60000
Peter 30 70000
```

The duplicate rows are removed from the original dataset based on the deduplicated dataset's name, age,
and income columns.

Handing outliers

In real-world data analysis, we often come across data with outliers. Outliers are very small or huge
values that deviate significantly from other observations in a dataset. Such outliers are first identified,
then removed, and the dataset is transformed at a specific scale. Let's understand with the following
detail.

Identifying outliers

As we've already seen, the first step is to identify the outliers in our dataset. Various statistical techniques can be used for this, such as the interquartile range (IQR), z-score, or Tukey methods.

We'll mainly look at z-score. It's a common technique for the identification of outliers in the dataset.

The z-score measures how many standard deviations an observation is from the mean of the dataset. The formula for calculating the z-score of an observation is this:

z = (observation - mean) / standard deviation

The threshold for the z-score method is typically chosen based on the level of significance or the desired level of confidence in identifying outliers. A commonly used threshold is a z-score of 3, meaning any observation with a z-score more significant than 3 or less than -3 is considered an outlier.

Removing outliers

Once the outliers are identified, they can be removed from the dataset using various techniques such as trimming, or removing the observations with extreme values. However, it's important to carefully analyze the dataset and determine the appropriate technique for handling outliers.

Transforming the data

Alternatively, the data can be transformed using mathematical functions such as logarithmic, square root, or inverse functions to reduce the impact of outliers on the analysis:

```
# Import pandas and numpy libraries
import pandas as pd
import numpy as np
# Create a custom dataset with outliers
data = pd.DataFrame({'age': [20, 25, 30, 35, 40, 200],
'income': [50000, 60000, 70000, 80000, 90000, 100000]})
# Calculate the mean and standard deviation of the data
mean = data.mean()
std_dev = data.std()
# Identify outliers using the z-score method
threshold = 3
z_scores = ((data - mean) / std_dev).abs()
outliers = data[z_scores > threshold]
# Remove outliers
data_without_outliers = data[z_scores <= threshold]
# Print the original and cleaned datasets
print("Original dataset:")
print(data)
print("\nOutliers:")
print(outliers)
print("\nDataset without outliers:")
print(data_without_outliers)
```

In this example, we've created a custom dataset with outliers in the age column. We then apply the outlier
handling technique to identify and remove outliers from the dataset. We first calculate the mean and
standard deviation of the data, and then identify the outliers using the z-score method. The z-score is
calculated for each observation in the dataset, and any observation that has a z-score greater than the
threshold value (in this case, 3) is considered an outlier. Finally, we remove the outliers from the dataset.
The output of the above code in table form is shown below.

Original dataset

age	income
20	50000
25	60000
30	70000
35	80000
40	90000
200	100000

Outliers

age	income
200	100000

Dataset without outliers

age	income
20	50000
25	60000
30	70000
35	80000
40	90000

The outlier (200) in the age column in the dataset without outliers is removed from the original dataset.

Data Transformation

Data transformation is another method in data processing to improve data quality by modifying it. This
transformation process involves converting the raw data into a more suitable format for analysis by
adjusting the data's scale, distribution, or format.

Log transformation is used to reduce outliers' impact and transform skewed (a situation where the
distribution of the target variable or class labels is highly imbalanced) data into a normal distribution. It's
a widely used transformation technique that involves taking the natural logarithm of the data.

Square root transformation is another technique to transform skewed data into a normal distribution. It
involves taking the square root of the data, which can help reduce the impact of outliers and improve the
data distribution.

Let’s look at an example:

```
# Import pandas and numpy libraries
import pandas as pd
import numpy as np
# Create a custom dataset
data = pd.DataFrame({'age': [20, 25, 30, 35, 40, 45],
'income': [50000, 60000, 70000, 80000, 90000, 100000],
'spending': [1, 4, 9, 16, 25, 36]})
# Apply square root transformation
data['sqrt_spending'] = np.sqrt(data['spending'])
# Print the original and transformed datasets
print("Original dataset:")
print(data)
print("\nTransformed dataset:")
print(data[['age', 'income', 'sqrt_spending']])
```

In this example, our custom dataset has a variable called spending. A significant outlier in this variable is causing skewness in the data. We’re controlling this skewness in the spending variable. The square root transformation has transformed the skewed spending variable into a more normal distribution.

Transformed values are stored in a new variable called sqrt_spending. The normal distribution of sqrt_spending is between 1.00000 to 6.00000, making it more suitable for data analysis.

The output of the above code in table form is shown below.

Original dataset

age	income	spending
20	50000	1
25	60000	4
30	70000	9
35	80000	16
40	90000	25
45	100000	36

Transformed dataset

age	income	sqrt_spending
20	50000	1.00000
25	60000	2.00000
30	70000	3.00000
35	80000	4.00000
40	90000	5.00000
45	100000	6.00000

Data Integration

The data integration technique combines data from various sources into a single, unified view. This helps to increase the completeness and diversity of the data, as well as resolve any inconsistencies or conflicts that may exist between the different sources. Data integration is helpful for data mining, enabling data

analysis spread across multiple systems or platforms.
Let’s suppose we have two datasets. One contains customer IDs and their purchases, while the other
dataset contains information on customer IDs and demographics, as given below. We intend to combine
these two datasets for a more comprehensive customer behavior analysis.
Customer Purchase Dataset
Customer ID Purchase Amount
1 $50
2 $100
3 $75
4 $200
Customer Demographics Dataset
Customer ID Age Gender
1 25 Male
2 35 Female
3 30 Male
4 40 Female
To integrate these datasets, we need to map the common variable, the customer ID, and combine the
data. We can use the Pandas library in Python to accomplish this:

```
# Import pandas library
import pandas as pd
# Load customer purchase dataset
purchase_data = pd.DataFrame({'Customer ID': [1, 2, 3, 4],
'Purchase Amount': [50, 100, 75, 200]})
# Load customer demographics dataset
demographics_data = pd.DataFrame({'Customer ID': [1, 2, 3, 4],
'Age': [25, 35, 30, 40],
'Gender': ['Male', 'Female', 'Male', 'Female']})
# Merge the two datasets on customer ID
merged_data = pd.merge(purchase_data, demographics_data, on='Customer ID')
# Print the merged dataset
print(merged_data)
```

The output of the above code in table form is shown below.
Customer ID Purchase Amount Age Gender
1 $50 25 Male
2 $100 35 Female
3 $75 30 Male
4 $200 40 Female
We’ve used the merge() function from the Pandas library. It merges the two datasets based on the common customer ID
variable. It results in a unified dataset containing purchase information and customer demographics. This integrated
dataset can now be used for more comprehensive analysis, such as analyzing purchasing patterns by age or gender.

Data reduction is one of the commonly used techniques in the data processing. It's used when we have a lot of data with
plenty of irrelevant information. This method reduces data without losing the most critical information.
There are different methods of data reduction, such as those listed below.
Data cube aggregation involves summarizing or aggregating the data along multiple dimensions, such as time, location,
product, and so on. This can help reduce the complexity and size of the data, as well as reveal higher-level patterns and
trends.
Dimensionality reduction involves reducing the number of attributes or features in the data by selecting a subset of
relevant features or transforming the original features into a lower-dimensional space. This can help remove noise and
redundancy and improve the efficiency and accuracy of data mining algorithms.
Data compression involves encoding the data in a more minor form, by using techniques such as sampling, clustering,
histogram analysis, wavelet analysis, and so on. This can help reduce the data's storage space and transmission cost
and speed up data processing.
Numerosity reduction replaces the original data with a more miniature representation, such as a parametric model (for
example, regression, log-linear models, and so on) or a non-parametric model (such as histograms, clusters, and so on).
This can help simplify the data structure and analysis and reduce the amount of data to be mined.
Data preprocessing is essential, because the quality of the data directly affects the accuracy and reliability of the
analysis or model. By properly preprocessing the data, we can improve the performance of the machine learning models
and obtain more accurate insights from the data.
Conclusion
Preparing data for machine learning is like getting ready for a big party. Like cleaning and tidying up a room, data
preprocessing involves fixing inconsistencies, filling in missing information, and ensuring that all data points are
compatible. Using techniques such as data cleaning, data transformation, data integration, and data reduction, we create
a well-prepared dataset that allows computers to identify patterns and learn effectively.
It's recommended that we explore data in depth, understand data patterns and find the reasons for missingness in data
before choosing an approach. Validation and test set are also important ways to evaluate the performance of different
techniques.
Share This Article
Rehman Ahmad Chaudhary
Rehman Ahmad Chaudhary

Rehman Ahmad Chaudhary is a blogger, writer, computer vision enthusiast, programming guy, and founder of
StudyEnablers.
data cleaning
pandas
SitePoint Premium
Data cleaning involves identifying and correcting errors, inconsistencies, and inaccuracies in the data. Some standard techniques used in
data cleaning include:
handling missing values
handling duplicates
handling outliers
Let’s discuss each of these data-cleaning techniques in turn.
Handling missing values
Handling missing values is an essential part of data preprocessing. Observations with missing data are dealt with under this technique.
We’ll discuss three standard methods for handling missing values: removing observations (rows) with missing values, imputing missing
values with the statistics tools, and imputing missing values with machine learning algorithms.
We will demonstrate each technique with a custom dataset and explain the output of each method, discussing all of these
techniques of handling missing values individually.
Dropping observations with missing values
The simplest way to deal with missing values is to drop rows with missing ones. This method usually isn’t recommended, as it can
affect our dataset by removing rows containing essential data.
Let’s understand this method with the help of an example. We create a custom dataset with age, income, and education data. We
introduce missing values by setting some values to NaN (not a number). NaN is a special floating-point value that indicates an
invalid or undefined result. The observations with NaN will be dropped with the help of the dropna() function from the Pandas
library:

```
# Importing pandas and numpy libraries
import pandas as pd
import numpy as np
# Create a custom dataset with missing values
data = pd.DataFrame({'age': [20, 25, np.nan, 35, 40, np.nan],
'income': [50000, np.nan, 70000, np.nan, 90000, 100000],
'education': ['Bachelor', np.nan, 'PhD', 'Bachelor', 'Master', np.nan]})
# Drop observations with missing values, axis = 0 means we want to drop rows
data_cleaned = data.dropna(axis=0)
print("Original dataset:")
print(data)
print("\nCleaned dataset:")
print(data_cleaned)
```

Original dataset

age income education
20 50000 Bachelor
25 NaN NaN
NaN 70000 PhD
35 NaN Bachelor
40 90000 Master
NaN 100000 NaN
Cleaned dataset
age income education
20 50000 Bachelor
40 90000 Master
The observations with missing values are removed in the cleaned dataset, so only the observations without missing
values are kept. You'll find that only row 0 and 4 are in the cleaned dataset.
Imputing missing values with statistics tools
This is a more sophisticated way to deal with missing data compared with the previous one. It replaces the missing
values with some statistics, such as the mean, median, mode, or constant value

```
# Importing pandas and numpy libraries
import pandas as pd
import numpy as np
# Create a custom dataset with missing values
data = pd.DataFrame({'age': [20, 25, 30, 35, np.nan, 45],
'income': [50000, np.nan, 70000, np.nan, 90000, 100000],
'gender': ['M', 'F', 'F', 'M', 'M', np.nan],
'marital_status': ['Single', 'Married', np.nan, 'Married', 'Single', 'Single']})
# Fill missing values with median
data_imputed = data.fillna(data.median())
# Print the original and imputed datasets
print("Original dataset:")
print(data)
print("\nImputed dataset:")
print(data_imputed)
```

The output of the above code in table form is shown below.
Original dataset
age income gender marital_status
20 50000 M Single
25 NaN F Married
30 70000 F NaN
35 NaN M Married
NaN 90000 M Single
45 100000 NaN Single
Imputing missing values with machine learning algorithms
Machine-learning algorithms provide a sophisticated way to deal with missing values based on features of our data. For
example, the KNNImputer class from the Scikit-learn library is a powerful way to impute missing values. Let's understand
this with the help of a code example:

```
# Import pandas and numpy libraries
import pandas as pd
import numpy as np
# Create a sample dataset with missing values
df = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
'age': [25, 30, np.nan, 40, 45],
'gender': ['F', 'M', 'M', np.nan, 'F'],
'salary': [5000, 6000, 7000, 8000, np.nan]})
# Print the original dataset
print('Original Dataset')
print(df)
# Import KNNImputer class from Scikit-Learn
from sklearn.impute import KNNImputer
# Create an instance of KNNImputer with default parameters
imputer = KNNImputer()
# Convert the categorical column gender to numeric values
df['gender'] = df['gender'].map({'F': 0, 'M': 1})
# Impute missing values with KNNImputer
df_imputed = imputer.fit_transform(df[['age', 'gender', 'salary']])
# Convert the imputed array back to a dataset
df_imputed = pd.DataFrame(df_imputed, columns=['age', 'gender', 'salary'])
# Add the name column back to the imputed dataset
df_imputed['name'] = df['name']
# Print the dataset after imputing with KNNImputer
print('Dataset after imputing with KNNImputer')
print(df_imputed)
# Importing pandas library
import pandas as pd
# Create a custom dataset with duplicate rows
data = pd.DataFrame({'name': ['John', 'Emily', 'Peter', 'John', 'Emily'],
'age': [20, 25, 30, 20, 25],
'income': [50000, 60000, 70000, 50000, 60000]})
# Check for duplicate rows
duplicates = data[data.duplicated()]
# Drop duplicate rows
data_deduplicated = data.drop_duplicates()
# Print the original and deduplicated datasets
print("Original dataset:")
print(data)
print("\nDuplicate rows:")
print(duplicates)
print("\nDeduplicated dataset:")
print(data_deduplicated)
```

The output of the above code is shown below.

Original dataset

name	age	income
John	20	50000
Emily	25	60000

Peter 30 70000
John 20 50000
Emily 25 60000
Duplicate rows
name age income
John 20 50000
Emily 25 60000
Deduplicated dataset
name age income
John 20 50000
Emily 25 60000
Peter 30 70000
The duplicate rows are removed from the original dataset based on the deduplicated dataset's name, age, and income
columns.
Handing outliers
In real-world data analysis, we often come across data with outliers. Outliers are very small or huge values that deviate
significantly from other observations in a dataset. Such outliers are first identified, then removed, and the dataset is
transformed at a specific scale. Let's understand with the following detail.
Identifying outliers
As we've already seen, the first step is to identify the outliers in our dataset. Various statistical techniques can be used
for this, such as the interquartile range (IQR), z-score, or Tukey methods.
We'll mainly look at z-score. It's a common technique for the identification of outliers in the dataset.
The z-score measures how many standard deviations an observation is from the mean of the dataset. The formula for
calculating the z-score of an observation is this:
z = (observation - mean) / standard deviation
The threshold for the z-score method is typically chosen based on the level of significance or the desired level of
confidence in identifying outliers. A commonly used threshold is a z-score of 3, meaning any observation with a z-score
more significant than 3 or less than -3 is considered an outlier.
Removing outliers
Once the outliers are identified, they can be removed from the dataset using various techniques such as trimming, or
removing the observations with extreme values. However, it's important to carefully analyze the dataset and determine
the appropriate technique for handling outliers.
Transforming the data
Alternatively, the data can be transformed using mathematical functions such as logarithmic, square root, or inverse
functions to reduce the impact of outliers on the analysis:
# Import pandas and numpy libraries
import pandas as pd

```
import numpy as np
# Create a custom dataset with outliers
data = pd.DataFrame({'age': [20, 25, 30, 35, 40, 200],
'income': [50000, 60000, 70000, 80000, 90000, 100000]})
# Calculate the mean and standard deviation of the data
mean = data.mean()
std_dev = data.std()
# Identify outliers using the z-score method
threshold = 3
z_scores = ((data - mean) / std_dev).abs()
outliers = data[z_scores > threshold]
# Remove outliers
data_without_outliers = data[z_scores <= threshold]
# Print the original and cleaned datasets
print("Original dataset:")
print(data)
print("\nOutliers:")
print(outliers)
print("\nDataset without outliers:")
print(data_without_outliers)
```

In this example, we've created a custom dataset with outliers in the age column. We then apply the outlier handling technique to identify and remove outliers from the dataset. We first calculate the mean and standard deviation of the data, and then identify the outliers using the z-score method. The z-score is calculated for each observation in the dataset, and any observation that has a z-score greater than the threshold value (in this case, 3) is considered an outlier.

Finally, we remove the outliers from the dataset.

The output of the above code in table form is shown below.

Original dataset

age	income
20	50000
25	60000
30	70000
35	80000
40	90000
200	100000

Outliers

age	income
200	100000

Dataset without outliers

age	income
20	50000
25	60000
30	70000
35	80000
40	90000

The outlier (200) in the age column in the dataset without outliers is removed from the original dataset.

Learn to Code with JavaScript

Data Transformation

Data transformation is another method in data processing to improve data quality by modifying it. This transformation
process involves converting the raw data into a more suitable format for analysis by adjusting the data's scale,
distribution, or format.

Log transformation is used to reduce outliers' impact and transform skewed (a situation where the distribution of the
target variable or class labels is highly imbalanced) data into a normal distribution. It's a widely used transformation
technique that involves taking the natural logarithm of the data.

Square root transformation is another technique to transform skewed data into a normal distribution. It involves taking
the square root of the data, which can help reduce the impact of outliers and improve the data distribution.

Let's look at an example:

```
# Import pandas and numpy libraries
import pandas as pd
import numpy as np
# Create a custom dataset
data = pd.DataFrame({'age': [20, 25, 30, 35, 40, 45],
'income': [50000, 60000, 70000, 80000, 90000, 100000],
'spending': [1, 4, 9, 16, 25, 36]})
# Apply square root transformation
data['sqrt_spending'] = np.sqrt(data['spending'])
# Print the original and transformed datasets
print("Original dataset:")
print(data)
print("\nTransformed dataset:")
print(data[['age', 'income', 'sqrt_spending']])
```

In this example, our custom dataset has a variable called spending. A significant outlier in this variable is causing
skewness in the data. We're controlling this skewness in the spending variable. The square root transformation has
transformed the skewed spending variable into a more normal distribution. Transformed values are stored in a new
variable called sqrt_spending. The normal distribution of sqrt_spending is between 1.00000 to 6.00000, making it more
suitable for data analysis.

The output of the above code in table form is shown below.

Original dataset

age	income	spending
20	50000	1
25	60000	4
30	70000	9

35 80000 16
40 90000 25
45 100000 36
Transformed dataset
age income sqrt_spending
20 50000 1.00000
25 60000 2.00000
30 70000 3.00000
35 80000 4.00000
40 90000 5.00000
45 100000 6.00000
Data Integration
The data integration technique combines data from various sources into a single, unified view. This helps to increase the
completeness and diversity of the data, as well as resolve any inconsistencies or conflicts that may exist between the
different sources. Data integration is helpful for data mining, enabling data analysis spread across multiple systems or
platforms.
Let’s suppose we have two datasets. One contains customer IDs and their purchases, while the other dataset contains
information on customer IDs and demographics, as given below. We intend to combine these two datasets for a more
comprehensive customer behavior analysis.
Customer Purchase Dataset
Customer ID Purchase Amount
1 $50
2 $100
3 $75
4 $200
Customer Demographics Dataset
Customer ID Age Gender
1 25 Male
2 35 Female
3 30 Male
4 40 Female
To integrate these datasets, we need to map the common variable, the customer ID, and combine the data. We can use
the Pandas library in Python to accomplish this:

```
# Import pandas library
import pandas as pd
# Load customer purchase dataset
purchase_data = pd.DataFrame({'Customer ID': [1, 2, 3, 4],
'Purchase Amount': [50, 100, 75, 200]})
# Load customer demographics dataset
demographics_data = pd.DataFrame({'Customer ID': [1, 2, 3, 4],
'Age': [25, 35, 30, 40],
'Gender': ['Male', 'Female', 'Male', 'Female']})
```

```
# Merge the two datasets on customer ID
merged_data = pd.merge(purchase_data, demographics_data, on='Customer ID')
# Print the merged dataset
print(merged_data)
```

The output of the above code in table form is shown below.

Customer ID	Purchase Amount	Age	Gender
1	$50	25	Male
2	$100	35	Female
3	$75	30	Male
4	$200	40	Female

We've used the merge() function from the Pandas library. It merges the two datasets based on the common customer ID
variable. It results in a unified dataset containing purchase information and customer demographics. This integrated
dataset can now be used for more comprehensive analysis, such as analyzing purchasing patterns by age or gender.

Data Reduction

Data reduction is one of the commonly used techniques in the data processing. It's used when we have a lot of data with
plenty of irrelevant information. This method reduces data without losing the most critical information.

There are different methods of data reduction, such as those listed below.

Data cube aggregation involves summarizing or aggregating the data along multiple dimensions, such as time, location,
product, and so on. This can help reduce the complexity and size of the data, as well as reveal higher-level patterns and
trends.

Dimensionality reduction involves reducing the number of attributes or features in the data by selecting a subset of
relevant features or transforming the original features into a lower-dimensional space. This can help remove noise and
redundancy and improve the efficiency and accuracy of data mining algorithms.

Data compression involves encoding the data in a more minor form, by using techniques such as sampling, clustering,
histogram analysis, wavelet analysis, and so on. This can help reduce the data's storage space and transmission cost
and speed up data processing.

Numerosity reduction replaces the original data with a more miniature representation, such as a parametric model (for
example, regression, log-linear models, and so on) or a non-parametric model (such as histograms, clusters, and so on).
This can help simplify the data structure and analysis and reduce the amount of data to be mined.

Data preprocessing is essential, because the quality of the data directly affects the accuracy and reliability of the
analysis or model. By properly preprocessing the data, we can improve the performance of the machine learning models

and obtain more accurate insights from the data.

Conclusion

Preparing data for machine learning is like getting ready for a big party. Like cleaning and tidying up a room, data preprocessing involves fixing inconsistencies, filling in missing information, and ensuring that all data points are compatible. Using techniques such as data cleaning, data transformation, data integration, and data reduction, we create a well-prepared dataset that allows computers to identify patterns and learn effectively. It’s recommended that we explore data in depth, understand data patterns and find the reasons for missingness in data before choosing an approach. Validation and test set are also important ways to evaluate the performance of different Techniques.

Project for Phase-4

E-commerce Application

When we built a e-commerce app using cloud application development, we'll focus on integrating key features, such as user authentication, product catalog, and shopping cart.

1.User Authentication:

- Implement user registration and login functionality using a cloud-based authentication service like Firebase Authentication, AWS Cognito, or Auth0.
- Ensure secure storage of user data and passwords in the cloud with proper encryption and security measures.

2.Product Catalog Management:

- Create a cloud-based database for storing product information. Services like Firebase Realtime Database, Firestore, or Amazon DynamoDB are good options.
- Develop APIs to manage the product catalog, allowing administrators to add, edit, and remove products.

3.Shopping Cart:

- Design a cloud-based shopping cart system that can store user's cart items across sessions and devices.
- Utilize cloud databases to track cart contents and sync them in real-time for a seamless shopping experience.

4.Order Processing:

- Implement a cloud-based order processing system to handle order placement, payment processing, and order confirmation.
- Use payment gateways like Stripe, PayPal, or cloud-based solutions such as AWS Lambda for order processing.

5.Inventory Management:

- Set up a cloud-based inventory management system that keeps track of product availability and updates it in real-time as users make purchases.

6.Recommendation Engine:

- Integrate a recommendation engine using cloud-based machine learning services to suggest products to users based on their browsing and purchase history.

7.Notifications:

- Utilize cloud-based notification services to send order updates, promotional messages, and

other relevant notifications to users.
8.Scalability and Performance:
- Ensure the application's scalability and performance by using cloud services like AWS Elastic Beanstalk, Google App Engine, or Azure App Service.
- Implement auto-scaling to handle increased traffic during peak times.
9.Security:
- Implement robust security measures to protect user data, payment information, and the application from security threats. Regularly audit and update security protocols.
10.Testing and Quality Assurance:
- Develop a comprehensive testing strategy, including unit testing, integration testing, and user acceptance testing.
- Use cloud-based testing services for load testing and performance testing.
11.Monitoring and Analytics:
- Set up cloud-based monitoring and analytics tools to track application performance, user behavior, and potential issues.
- Use services like AWS CloudWatch, Google Analytics, or similar tools.
12.Compliance and Data Privacy:
- Ensure compliance with data protection regulations (e.g., GDPR, CCPA) by implementing data anonymization and providing users with data management options.
13.Continuous Deployment and DevOps:
- Implement a CI/CD pipeline with cloud-based tools like Jenkins, Travis CI, or GitLab CI/CD for automated deployment and updates.
14.User Support and Feedback:
- Offer customer support through cloud-based services such as chatbots or helpdesk systems.
- Encourage user feedback and use cloud analytics to make data-driven improvements.
15.Cost Optimization:
- Regularly review cloud service costs and optimize resource usage to avoid unnecessary expenses.
Code for developing E-commerce Application:

```
# Import the Firebase SDK
import firebase_admin
from firebase_admin import credentials, db
# Initialize Firebase with your credentials
cred = credentials.Certificate('path/to/your/serviceAccountKey.json')
firebase_admin.initialize_app(cred, {
'databaseURL': 'https://your-app-name.firebaseio.com/'
})
# Reference to your Firebase Realtime Database
ref = db.reference('products')
# Sample function to add a product to the catalog
def add_product(name, price, description):
new_product_ref = ref.push()
new_product_ref.set({
'name': name,
'price': price,
'description': description
})
```

```
# Sample function to retrieve all products
def get_all_products():
return ref.get()
# Sample function to update a product
def update_product(product_id, new_data):
product_ref = ref.child(product_id)
product_ref.update(new_data)
# Sample function to delete a product
def delete_product(product_id):
product_ref = ref.child(product_id)
product_ref.delete()
# Usage example
if __name__ == "__main__":
# Add a product to the catalog
add_product("Sample Product", 19.99, "A sample product description")
# Retrieve and print all products
products = get_all_products()
print("All Products:")
print(products)
# Update a product
update_product("Ice Cream", {'price': 29.99})
# Delete a product
delete_product("Ice Cream")
```