Chapter 1 (2,3, 4, 7, 9, and 10), Chapter 3 (3,4,7,8, and 9), and Chapter 4 (2,3, 4, and 5).

Chapter 1:

1. Define Machine learning with examples. (Include one more definition in your answer by doing a general internet search and/or from other resources)
    - Machine learning is a subset of artificial intelligence that enables computers to learn and improve from data without explicit programming. Examples include image recognition, recommendation systems, autonomous vehicles, healthcare diagnostics, and fraud detection. It finds applications in a wide range of fields, offering automation and data-driven decision-making.

2. Describe two business use cases of ML. (other than above in slides)
    - **Sentiment Analysis:** Businesses often use machine learning for sentiment analysis, which involves analyzing text data, such as customer reviews or social media comments, to determine public sentiment towards their products or services. This information helps companies gauge customer satisfaction, identify areas for improvement, and adjust their marketing and product development strategies accordingly.
    - **Chatbots for Customer Support:** Businesses often employ chatbots with machine learning capabilities to provide automated customer support. These chatbots can understand and respond to customer inquiries in a conversational manner. Over time, they learn from customer interactions, improving their ability to answer questions and resolve issues efficiently, reducing the need for human intervention in routine support tasks.

3. List types of ML, with one example of each.

    There are three main types of machine learning, each with its own characteristics and applications:

    - **Supervised Learning**:

      Example: Email Classification

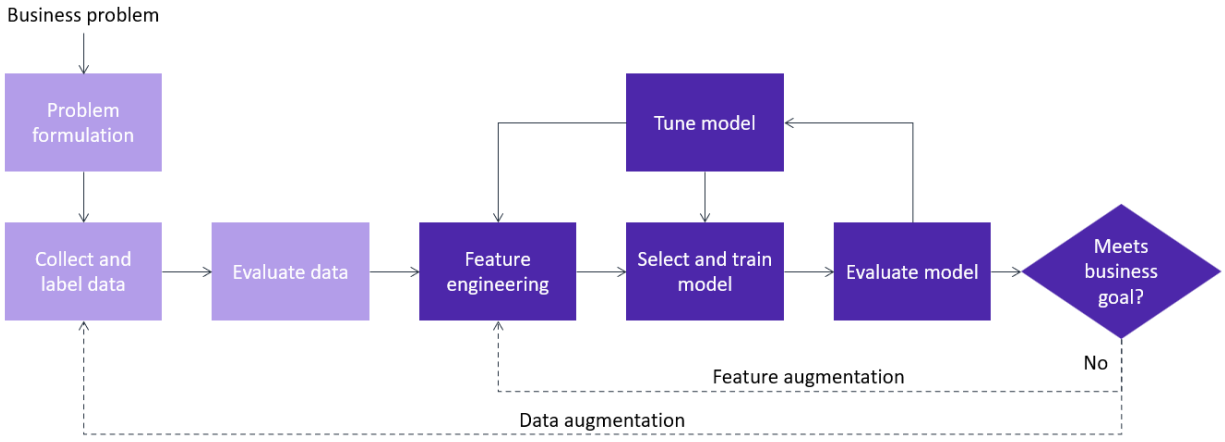
      In supervised learning, the algorithm is trained on a labeled dataset, where the input data is paired with the corresponding correct output. The model learns to make predictions or classifications based on this historical data. For example, in email classification, the algorithm is trained on a dataset of emails labeled as "spam" or "not spam" to learn how to classify new, unlabeled emails.
    - **Unsupervised Learning:**

      Example: Clustering Customer Segments

      Unsupervised learning involves working with unlabeled data to discover patterns, relationships, or structures within the data. For instance, in customer segmentation, unsupervised learning algorithms can group customers based on shared characteristics or behaviors, helping businesses identify target audiences or tailor marketing strategies.

- **Reinforcement Learning:**
  Example: Game Playing
  Reinforcement learning is about training agents to make sequences of decisions in an environment to maximize a cumulative reward. For example, in game playing, a reinforcement learning agent can learn to play games like chess or Go by taking actions and receiving rewards or penalties based on its performance. Over time, it learns to make better decisions to maximize its score or chances of winning.

4. List phases of standard Machine Learning pipeline process with diagram.
   1. **Business Problem:**
   - Identify the business problem you want to solve using machine learning. Define the problem, objectives, and expected outcomes.
   2. **Data Collection and Labeling:**
   - Collect relevant data from various sources. Ensure the data is labeled correctly to enable supervised learning.
   3. **Data Evaluation:**
   - Assess the quality and suitability of your collected data. Identify and handle any issues like missing values, outliers, or data imbalances.
   4. **Feature Engineering:**
   - Select or create relevant features from the data that are essential for the machine learning model. This may involve data transformation and normalization.
   5. **Training:**
   - Choose an appropriate machine learning algorithm and train the model using the prepared data. The model learns from the labeled data.
   6. **Evaluation:**
   - Evaluate the model's performance using appropriate evaluation metrics and a validation dataset. This step helps you determine how well the model generalizes to new data.
   7. **Tuning and Deployment:**
   - Fine-tune the model by adjusting hyperparameters to optimize its performance. If the model meets the desired performance criteria, it can be deployed for real-time predictions in a production environment.



5. Describe ML challenges. (There can several other challenges you may find from other resources. Explore it)
    - **Data Quality and Quantity:**
      Challenge: Insufficient, noisy, or biased data can negatively impact model performance.
      Bias and Fairness:
    - Challenge: Models may inherit biases from training data, leading to unfair or discriminatory predictions. Ensuring fairness is essential.
      Interpretability:
    - Challenge: Many machine learning models are complex "black boxes," making it difficult to understand their decision-making processes.
      Scalability:
    - Challenge: Scaling machine learning models to handle large datasets or real-time processing can be resource-intensive.
      Security Risks:
    - Challenge: Machine learning models can be vulnerable to adversarial attacks, impacting their reliability and trustworthiness.

    These challenges can significantly impact the success and ethical use of machine learning in various applications.

6. Which types of roles (jobs) in the area of ML, you may perform in future?
    In the area of machine learning and AI, there are several roles that professionals might perform:
    - Machine Learning Engineer: Develop and implement machine learning models, design and train algorithms, and work on model deployment and optimization.
    - Data Scientist: Analyze large datasets, create predictive models, and extract valuable insights to support data-driven decision-making.
    - AI Research Scientist: Conduct research in the field of artificial intelligence, explore new algorithms and models, and push the boundaries of AI capabilities.
    - Data Engineer: Design, construct, install, and maintain large-scale processing systems for collecting, storing, and analyzing data.
    - AI Ethics and Fairness Specialist: Ensure that AI systems are developed and deployed in an ethical and fair manner, addressing bias, privacy, and transparency issues.
    - AI Product Manager: Oversee the development of AI-driven products, from conception to deployment, working with cross-functional teams.
    - AI Consultant: Provide expertise and guidance to businesses seeking to leverage AI and machine learning for their operations and strategies.
    - AI Research Engineer: Collaborate with academic institutions or industry labs to advance the state of the art in AI through research and development.

Chapter 3:

1. How to collect data from the data source and what type of data source is available, explain with an example.

Collecting data from various sources is a fundamental step in any data-driven project, including machine learning. Data can be sourced from a variety of places, including open-source data, private data, commercial data providers, and data marketplaces like AWS Data Exchange. Let's explore each source type with an example:

- **Open-Source Data:**
  Source: Open-source data is publicly available and can be freely accessed by anyone. It's often provided by government agencies, research institutions, or individuals.
  Example: The U.S. Census Bureau provides open-source data on population demographics, economic indicators, and social trends. Researchers, businesses, and policymakers use this data for various applications, such as market research and urban planning.
- **Private Data:**
  Source: Private data is data owned by an individual or organization and is not publicly accessible. Access to this data typically requires permissions or agreements.
  Example: A retail company collects customer transaction data, which includes purchase history, customer demographics, and browsing behavior on their website. This data is considered private and can be used for personalized marketing and product recommendations.
- **Commercial Data:**
  Source: Commercial data is data provided by specialized data vendors or companies that aggregate, process, and sell datasets for specific purposes.
  Example: A financial institution might purchase credit scoring data from a commercial provider to assess the creditworthiness of loan applicants. Companies like Experian and Equifax provide such data for risk assessment.
- **AWS Data Exchange:**
  Source: AWS Data Exchange is a data marketplace where you can discover, subscribe to, and use data from various providers. It offers a range of datasets, including third-party data and data generated by AWS services.
  Example: A data science team at a marketing agency subscribes to demographic and geographic data from AWS Data Exchange. They use this data to optimize their clients' ad targeting strategies, tailoring advertisements to specific customer segments.

2. Describe the ETL process with any ETL tool/ service of your choice.
   The ETL (Extract, Transform, Load) process is a fundamental part of data integration, where data is collected from various sources, transformed to fit a common data structure, and loaded into a target database or data warehouse. ETL tools and services simplify and automate this process. One commonly used ETL service is Apache Nifi. Here's an overview of the ETL process with Apache Nifi as an example:
   - 1. Extraction (E):
     In this phase, data is extracted from various source systems. Apache Nifi can connect to a wide range of sources, including databases, APIs, files, and streaming data.
     For example, you might extract data from a source like an e-commerce database, a log file, or an external API.
   - 2. Transformation (T):

During the transformation phase, data is cleaned, enriched, and transformed to meet the target schema or format. Apache Nifi offers a set of processors and tools to perform these transformations.
Transformations may include data cleansing, aggregation, data type conversions, and calculations. You can use Nifi processors to filter, join, split, or modify data as needed.
For example, you might clean and format customer names, enrich data with geospatial information, or aggregate sales data by region.

- 3. Loading (L):
  Once data is prepared, it is loaded into a target database or data warehouse. Apache Nifi supports a variety of destination connectors for popular databases, data lakes, and data warehousing platforms.
  You can configure Nifi processors to push the transformed data to the target system with appropriate credentials and settings.
  For instance, you might load the cleaned and enriched data into a data warehouse like Amazon Redshift, Google BigQuery, or an on-premises SQL Server database.
- 4. Orchestration and Scheduling:
  ETL processes may need to be orchestrated and scheduled to run at specific intervals or in response to certain events. ETL tools like Nifi often provide scheduling and workflow capabilities.
  You can schedule Nifi flows to run periodically, trigger ETL jobs based on data arrival, or create complex data workflows.
- 5. Monitoring and Error Handling:
  Monitoring tools and dashboards are essential for tracking the ETL process's health. Apache Nifi offers logging and monitoring features to detect and troubleshoot errors or bottlenecks in the ETL pipeline.
- 6. Scalability and Performance Optimization:
  Depending on data volume and complexity, you may need to scale ETL processes. Nifi can be configured to run on a cluster for scalability and performance optimization.
- 7. Data Validation and Testing:
  It's crucial to validate the data at each stage of ETL to ensure it meets quality and consistency standards. ETL tools often include testing and validation features.
- 8. Security and Compliance:
  Ensure that the ETL process complies with security and data privacy standards. Apache Nifi provides features for securing data in transit and at rest.

3. How pandas describe() function help you in phase-2 and phase-3 of the ML pipeline process.
   The describe() function in Pandas is a critical tool in both Phase 2 (Data Preprocessing) and Phase 3 (Feature Engineering) of the machine learning pipeline.
   In Phase 2:
   - Data Understanding: describe() provides a comprehensive statistical snapshot of numeric data, offering insights into distributions, central tendencies (mean), variability (standard deviation), and the presence of missing values. This is pivotal for understanding the dataset's fundamental characteristics.

- Handling Missing Values: By comparing the count of entries to the dataset's total size through describe(), missing values are quickly identified. This initial observation forms the basis for developing strategies to address these gaps, be it through imputation or removal.
- Detecting Outliers: Outliers can wield considerable influence over model performance. describe() assists in pinpointing potential outliers by exposing extreme values. This guides the decision-making process concerning their management.

  In Phase 3:
- Feature Selection: describe() serves as a compass for choosing relevant features for the machine learning model. For instance, low variance (indicated by a small standard deviation) may prompt considerations for feature removal.
- Normalizing and Scaling: The range and distribution insights from describe() are invaluable for implementing techniques such as min-max scaling or standardization for numeric features.
- Feature Creation: The summary statistics from describe() can spark the creation of novel features. For instance, when a wide range is observed, one might consider binning or discretizing a feature into categories, thereby constructing a categorical feature.

4. What is a histogram? And how it helps us in the data evaluation process of the ML pipeline.

   Histograms are graphical representations of data distributions. They show the frequency of different values or value ranges in a dataset by dividing the data into bins and counting the number of data points in each bin. Histograms play a significant role in the data evaluation process of the ML pipeline for several reasons:
   - Data Understanding : Histograms provide a visual and intuitive way to grasp the data's central tendency, spread, and shape. This aids in understanding the distribution of data, whether it's normally distributed, skewed, or exhibits other patterns.
   - Outlier Detection : Outliers, which are data points significantly different from the rest, can have a substantial impact on machine learning models. Histograms make it easy to identify outliers, as they appear as bars far from the central cluster of bars.
   - Data Quality Assessment : Issues with data quality, such as missing values or data entry errors, are identifiable through histograms. If bins have zero or very few data points, it signifies missing data within that range.
   - Feature Engineering : When considering feature engineering, histograms assist in deciding whether data transformation is needed. For example, if the data is highly skewed, transformations like logarithmic scaling can be applied to make it more suitable for certain algorithms.
   - Data Balancing : Histograms of the target variable in classification tasks reveal class imbalances. This is crucial for selecting the right evaluation metrics and addressing class imbalance through techniques like resampling.

5. Explain the use of correlation matrix and heat map.
   - Correlation Matrix and Heat Map play significant roles in data analysis and are essential tools in the data evaluation process of the machine learning pipeline. Here's how they are used:

Correlation Matrix (3 marks):

- A correlation matrix is a square table that shows the correlation coefficients between many variables. Each cell in the matrix displays the correlation between two variables. The correlation coefficient measures the strength and direction of a linear relationship between two variables.
- The primary use of a correlation matrix is to identify associations or relationships between variables in a dataset. It helps in understanding how variables are related to one another. A positive correlation indicates that as one variable increases, the other also tends to increase, while a negative correlation indicates that as one variable increases, the other tends to decrease.
- Correlation matrices are particularly valuable when dealing with feature selection, where you want to identify which variables are strongly correlated and might be candidates for removal to reduce multicollinearity in predictive models.

Heat Map (2 marks):

- A heat map is a graphical representation of the correlation matrix, where each cell's color corresponds to the correlation coefficient value between two variables. Darker colors often represent stronger correlations, while lighter colors indicate weaker or no correlations.
- Heat maps provide a more intuitive and visual representation of the correlation matrix, making it easier to identify patterns and relationships in the data. It's especially useful when you have a large number of variables.
- Heat maps are often used in exploratory data analysis to quickly identify important correlations and inform decisions related to feature selection, model building, or identifying potential issues like multicollinearity.

Correlation matrices and heat maps are essential for understanding variable relationships. They identify strong correlations, aiding data evaluation in the ML pipeline. Heat maps provide a visual way to grasp complex correlations, enhancing insights.

Chapter 4:

1. Describe Encoding ordinal data with an example.
    - Encoding ordinal data involves representing categorical variables with a clear order or hierarchy numerically. Ordinal data has a natural ranking or order among its categories, but the intervals between categories are not well-defined. Here's an example:
    - Example: Education Level (Ordinal)
    - Suppose you have a dataset with an "Education Level" variable, which includes categories like "High School," "Associate's Degree," "Bachelor's Degree," "Master's Degree," and "Ph.D." These categories have a clear order in terms of educational attainment, where a Ph.D. represents the highest level of education and High School the lowest.
    - You can encode this ordinal data by assigning numerical values based on their ranking:
    - High School: 1

- Associate's Degree: 2
- Bachelor's Degree: 3
- Master's Degree: 4
- Ph.D.: 5

  This encoding preserves the ordinal nature of the data, as higher numerical values represent higher education levels. Machine learning algorithms can then work with this encoded data, preserving the inherent order for analysis and predictions while respecting the ordinal characteristics of the variable.

2. What is the “Cleaning data” concept? Explain with an example.
   - Cleaning data is a fundamental step in data preprocessing where you identify and rectify errors, inconsistencies, and inaccuracies in a dataset to ensure its quality and reliability for analysis. This process involves handling missing values, removing duplicates, correcting errors, and addressing outliers. Here's an example to illustrate the concept of cleaning data:
   - Example: Sales Data
   - Suppose you work for a retail company, and you're given a dataset of daily sales transactions for a chain of stores. The dataset contains information about each sale, such as date, store location, product sold, price, and customer information.
   - During the data cleaning process, you might encounter the following issues:
   - Missing Values:
   - Some sales transactions have missing data, such as incomplete customer addresses, missing product prices, or blank dates.
   - You need to handle these missing values by either imputing reasonable values (e.g., using the mean price for missing product prices) or removing incomplete records.
   - Duplicates:
   - The dataset contains duplicate entries, where the same sale transaction appears multiple times.
   - Duplicates need to be identified and removed to avoid double-counting sales.
   - Inconsistent Data Formats:
   - In the "Date" column, dates are inconsistently formatted. For example, some entries are in MM/DD/YYYY format, while others are in DD-MM-YYYY.
   - You need to standardize the date format to ensure consistency throughout the dataset.
   - Outliers:
   - Some transactions may contain unusually high or low sales amounts, possibly due to data entry errors or exceptional circumstances.
   - Outliers need to be detected and evaluated to determine whether they should be removed or retained based on domain knowledge.
   - Data Validation:
   - The dataset may include entries with invalid or inconsistent data, such as negative product prices, nonexistent store locations, or unrealistic customer information.
   - These issues should be addressed by validating data against predefined rules and domain knowledge.

- By cleaning the data in this example, you ensure that it is free from errors and inconsistencies, making it reliable and suitable for meaningful analysis. Clean data forms the foundation for accurate statistical analysis, machine learning, and informed decision-making in various fields, including business, healthcare, and research.

3. Explain the “drop or impute” missing value with the scenario of each.
    - The decision to "drop or impute" missing values in a dataset depends on the context and the impact of those missing values on the analysis. Let's explore scenarios for each approach:

    - 1. Dropping Missing Values:

    - Scenario: You have a large dataset with a small proportion of missing values in a specific column, and those missing values do not carry critical information.

    - Example: Imagine you're analyzing customer feedback data for an e-commerce platform. One of the columns is "Optional Feedback," where customers can provide additional comments about their shopping experience. In a few cases, customers chose not to leave feedback, resulting in missing values in the "Optional Feedback" column. Since this column is not a primary focus of your analysis, and the missing values do not significantly affect the overall dataset, you decide to drop rows with missing "Optional Feedback" entries.

    - 2. Imputing Missing Values:

    - Scenario: The missing values are essential, and you want to retain as much data as possible to maintain statistical power and the integrity of your analysis.

    - Example: Consider a medical study dataset that includes patient records. The dataset contains a column for "Blood Pressure" measurements, but due to various reasons, some patients have missing values for this vital health indicator. In this case, you understand the importance of "Blood Pressure" in your analysis, and dropping those records would lead to a significant loss of critical data. Therefore, you decide to impute the missing "Blood Pressure" values. You may choose to impute them using the mean, median, or a predictive model based on other relevant variables in the dataset.

    - In summary, the decision to drop or impute missing values depends on the specific circumstances and the significance of the data in your analysis. Dropping is appropriate when missing values are relatively few and non-essential. Imputing is suitable when the missing values are important for your analysis, and you want to retain as much data as possible while maintaining data integrity. The choice should align with your research goals and the nature of the dataset.

4. What are outliners and how you can deal with them?
    - Outliers, in the context of data analysis and statistics, are data points that significantly deviate from the majority of data points in a dataset. They are often much larger or smaller than most other values and can distort the results of statistical analysis and machine learning models. Dealing with outliers is crucial to ensure the integrity and accuracy of your analysis. Here's how you can deal with outliers:
    - Identify Outliers:
    - The first step is to identify outliers in your dataset. Common techniques include visual inspection using box plots or scatter plots, or statistical methods like the Z-score or the IQR (Interquartile Range) method.
    - Understand the Context:
    - Before deciding how to deal with outliers, it's important to understand why they exist. Outliers could be a result of data entry errors, natural variability, or represent real but rare events. Understanding the context can guide your approach.
    - Handling Outliers:
    - a. Transformation:
    - Apply data transformations like log transformation or square root transformation to normalize the data distribution. This can reduce the impact of outliers in your analysis.
    - b. Imputation:
    - Replace extreme outlier values with more reasonable values based on domain knowledge or other statistical methods, such as imputing them with the mean or median of the dataset.
    - c. Truncation or Winsorization:
    - Cap the extreme values by setting a threshold beyond which any value is set to the threshold value. This is known as truncation or winsorization.
    - d. Remove Outliers:
    - In some cases, it may be appropriate to remove outliers from the dataset, especially if they are the result of data entry errors or have an undue impact on your analysis. Be cautious with this approach, as it can lead to data loss and potentially biased results.
    - e. Model-Based Approaches:
    - Use robust statistical models or machine learning algorithms that are less sensitive to outliers. Techniques like robust regression or decision trees can handle outliers better than traditional linear regression.
    - Documentation:
    - Regardless of how you deal with outliers, it's essential to document your approach. Explain why outliers were treated in a certain way and the impact on the analysis. Transparency in handling outliers is important for the reproducibility of your work.
    - The approach to handling outliers should be context-specific. The choice of method should depend on the nature of the data, the domain, and the goals of the analysis. It's important to carefully consider the consequences of your chosen approach and ensure that it aligns with the research objectives.

Chapter 2:

1. Create an array containing the values 1–15, reshape it into a 3-by-5 array, then use indexing and slicing techniques to perform each of the following operations:

    - Select row 2.
    - Select column 5.
    - Select rows 0 and 1.
    - Select columns 2–4.
    - Select the element that is in row 1 and column 4.
    - Select all elements from rows 1 and 2 that are in columns 0, 2, and 4.

```
import numpy as np

# Create an array containing values 1 to 15

original_array = np.arange(1, 16)

# Reshape it into a 3x5 array

reshaped_array = original_array.reshape(3, 5)

# Select row 2

row_2 = reshaped_array[1]

# Select column 5

column_5 = reshaped_array[:, 4]

# Select rows 0 and 1

rows_0_1 = reshaped_array[0:2]

# Select columns 2 to 4

columns_2_4 = reshaped_array[:, 2:5]

# Select the element in row 1 and column 4

element_1_4 = reshaped_array[1, 4]

# Select all elements from rows 1 and 2 that are in columns 0, 2, and 4

rows_1_2_columns_0_2_4 = reshaped_array[1:3, [0, 2, 4]]

# Print the results

print("Row 2:", row_2)

print("Column 5:", column_5)
```

```
print("Rows 0 and 1:", rows_0_1)

print("Columns 2 to 4:", columns_2_4)

print("Element in row 1 and column 4:", element_1_4)

print("Elements from rows 1 and 2 in columns 0, 2, and 4:", rows_1_2_columns_0_2_4)
```

2. Perform the following tasks with pandas DataFrames:

- **2.1**) Create a DataFrame named temperatures from a dictionary of three temperature readings each for 'Maxine', 'James' and 'Amanda'.
- **2.2**) Recreate the DataFrame temperatures in Part (a) with custom indices using the index keyword argument and a list containing 'Morning', 'Afternoon' and 'Evening'.
- **2.3**) Select from temperatures the column of temperature readings for 'Maxine'.
- **2.4**) Select from temperatures the row of 'Morning' temperature readings.
- **2.5**) Select from temperatures the rows for 'Morning' and 'Evening' temperature readings.
- **2.6**) Select from temperatures the columns of temperature readings for 'Amanda' and 'Maxine'.
- **2.7**) Select from temperatures the elements for 'Amanda' and 'Maxine' in the 'Morning' and 'Afternoon'.
- **2.8**) Use the describe method to produce temperatures' descriptive statistics.
- **2.9**) Transpose of temperatures.
- **2.10**) Sort temperatures so that its column names are in alphabetical order

```
import pandas as pd

# 2.1) Create a DataFrame named temperatures from a dictionary

data = { 'Maxine': [72, 76, 68],'James': [68, 74, 82],'Amanda': [75, 71, 79]}

temperatures = pd.DataFrame(data)

# 2.2) Recreate the DataFrame with custom indices

temperatures.index = ['Morning', 'Afternoon', 'Evening']

# 2.3) Select the column of temperature readings for 'Maxine'

maxine_temperatures = temperatures['Maxine']

# 2.4) Select the row of 'Morning' temperature readings

morning_readings = temperatures.loc['Morning']

# 2.5) Select the rows for 'Morning' and 'Evening' temperature readings

morning_evening_readings = temperatures.loc[['Morning', 'Evening']]

# 2.6) Select the columns of temperature readings for 'Amanda' and 'Maxine'
```

```
amanda_maxine_temperatures = temperatures[['Amanda', 'Maxine']]
# 2.7) Select the elements for 'Amanda' and 'Maxine' in the 'Morning' and 'Afternoon'
amanda_maxine_morning_afternoon = temperatures.loc[['Morning', 'Afternoon'], ['Amanda', 'Maxine']]
# 2.8) Use the describe method to produce descriptive statistics
statistics = temperatures.describe()
# 2.9) Transpose temperatures
transposed_temperatures = temperatures.T
# 2.10) Sort temperatures so that its column names are in alphabetical order
sorted_temperatures = temperatures.sort_index(axis=1)
# Printing the results
print("Original DataFrame:")
print(temperatures)
print("\nCustom Indexed DataFrame:")
print(temperatures)
print("\nTemperature Readings for Maxine:")
print(maxine_temperatures)
print("\nMorning Readings:")
print(morning_readings)
print("\nMorning and Evening Readings:")
print(morning_evening_readings)
print("\nAmanda and Maxine's Temperature Readings:")
print(amanda_maxine_temperatures)
print("\nAmanda and Maxine's Morning and Afternoon Readings:")
print(amanda_maxine_morning_afternoon)
print("\nDescriptive Statistics:")
print(statistics)
print("\nTransposed DataFrame:")
print(transposed_temperatures)
print("\nSorted Columns Alphabetically:")
```

```
print(sorted_temperatures)
```

Q3:

- 3.1) Class Average: Writing Grades to a Plain Text File
- In an Ipython session, write code that enables you to store any number of grades into a grades.txt plain text file.

```
# Sample list of grades
grades = [85, 92, 78, 89, 95, 88]
# Write the grades to a plain text file (grades.txt)
with open('grades.txt', 'w') as file:
    for grade in grades:
        file.write(str(grade) + '\n')
```

- 3.2) Class Average: Reading Grades from a Plain Text File
- In an Ipython session, write code that reads the grades from the grades.txt file you created in the previous exercise. Display the individual grades and their total, count and average.

```
# Read grades from the grades.txt file
with open('grades.txt', 'r') as file:
    grades = [int(line.strip()) for line in file]
# Display individual grades, total, count, and average
print("Individual Grades:", grades)
print("Total:", sum(grades))
print("Count:", len(grades))
print("Average:", sum(grades) / len(grades))
```

- 3.3) Class Average: Writing Student Records to a CSV File

An instructor teaches a class in which each student takes three exams. The instructor would like to store this information in a file named grades.csv for later use. Write code that enables an instructor to enter each student's first name and last name as strings and the student's three exam grades as integers. Use the csv module to write each record into the grades.csv file. Each record should be a single line of text in the following CSV format:

  - firstname,lastname,exam1grade,exam2grade,exam3grade

```
import csv

# Sample student records

students = [['John', 'Doe', 85, 92, 78],['Jane', 'Smith', 89, 95, 88],]

# Write student records to a CSV file (grades.csv)

with open('grades.csv', 'w', newline='') as file:
```

```
    writer = csv.writer(file)
    writer.writerow(['firstname', 'lastname', 'exam1grade', 'exam2grade', 'exam3grade'])
    for student in students:
        writer.writerow(student)
```

- 3.4) Class Average: Reading Student Records from a CSV File
- Use the csv module to read the grades.csv file from the previous exercise. Display the data in tabular format.

```
# Read student records from the grades.csv file
with open('grades.csv', 'r', newline='') as file:
    reader = csv.reader(file)
    header = next(reader)  # Skip the header row
    data = list(reader)
# Display data in tabular format
print("First Name  | Last Name  | Exam 1 | Exam 2 | Exam 3")
for row in data:
    print(f"{row[0]:<12} | {row[1]:<11} | {row[2]:<7} | {row[3]:<7} | {row[4]:<7}")
```

- 3.5) Class Average: Creating a Grade Report from a CSV File
- Modify your solution to the preceding exercise to create a grade report that displays each student's average to the right of that student's row and the class average for each exam below that exam's column.

```
# Calculate class averages for each exam
exam_averages = [sum(int(row[i]) for row in data) / len(data) for i in range(2, 5)]
# Display the grade report
print("First Name  | Last Name  | Exam 1 | Exam 2 | Exam 3 | Average")
for row in data:
    average = sum(int(grade) for grade in row[2:]) / 3
    print(f"{row[0]:<12} | {row[1]:<11} | {row[2]:<7} | {row[3]:<7} | {row[4]:<7} | {average:.2f}")
print("Class Averages:", exam_averages)
```