Code 1 Explanation:

1. Importing Libraries
At the beginning, the code is loading some tools (libraries):
- pandas: Used to handle data in table format (like spreadsheets).
- random: Allows the program to choose random elements, like randomly picking a number or person.
- geopy.distance: Used to calculate distances between two locations based on their latitude and longitude.
- matplotlib.pyplot: A tool for creating charts and graphs.
- numpy: Helps to handle numbers and large sets of data.

2. Loading Data
The code then loads two datasets:
- One contains information about orders (like the destination and the ID of each order).
- The other has data about delivery riders (like their ID and their location).

3. Assigning Data
The order data is stored in a variable called `data`, which makes it easier to work with throughout the code.

4. Selecting a Smaller Sample
Instead of using the entire dataset, the code selects orders from a specific day (October 17, 2022) and only looks at the first 100 orders. This is a small sample to make things quicker and simpler to process.

5. Random Assignment of Orders (Inefficient)
The code contains a function that randomly matches delivery riders with orders. This is done in a way that doesn't really care about efficiency, meaning the rider could be far away from the customer.

Here's the process:
- The program starts with an empty list to save the order-rider pairs.
- It creates a list of all rider IDs.
- For each order:
  - It picks a random rider from the list.
  - It gets the delivery address for that order (where the package is going).
  - Then it chooses a random starting location for the rider (to simulate randomness).
  - It calculates the distance between the rider's random location and the customer's delivery address.
  - Finally, it saves this match (which rider, which order, and how far the rider has to travel).
  - At the end, it returns a table with all the random matches.

6. Saving the Results
Once the random assignment is done, the results are saved in a new file. This lets you keep a record of which riders were assigned to which orders, along with the distances they have to travel.

7. Summary of Data

Next, the program generates a summary that includes:
- Basic statistics (like average distance, maximum, minimum, etc.).
- A count of how many orders were assigned to each rider (to see if some riders got more orders than others).

8. Scatter Plot of Rider Assignments

The code then creates a scatter plot, which is a type of chart that shows dots for each rider. The dots represent how many orders each rider was assigned.

To make sure the dots don't overlap, the program adds a tiny bit of randomness (called jitter) so each dot is spread out. This makes it easier to see the distribution of orders across different riders.

9. Histogram of Delivery Distances

Next, the program creates a histogram, which is like a bar chart. This shows how often different delivery distances occurred. For example, it will show if most deliveries were short or if there were some really long ones.

10. Final Summary

Lastly, the program prints a summary that includes:
- The total number of orders processed.
- How many were successfully assigned to riders.
- The average delivery distance that riders had to travel.

This summary helps to evaluate how well (or poorly) the random assignment process worked.

Code 2 Explanation:

1. Importing Libraries

The code first loads some tools:
- pandas: Helps in handling and processing data in table format.
- geopy.distance: Calculates the distance between two locations using their latitude and longitude.
- BallTree from sklearn: This helps in efficiently finding the closest points (like riders near a restaurant) based on their location.
- numpy: A tool for working with numbers, especially when dealing with large arrays of data.
- matplotlib.pyplot: Used to create visual graphs and charts.

2. Loading Data

The program loads several datasets:
- waybill_info: Contains information about the orders, like where the order is coming from (restaurant) and where it's going (customer).
- courier_info: Information about delivery riders (where they are located, their IDs, etc.).
- rider_info: Another dataset about riders' details.
- dispatch_info: Information about how the orders were dispatched.

3. Filtering the Orders

To make things simpler, the code only looks at the orders from one specific day (October 17, 2022). It also only selects the first 100 orders to work with, so it doesn't process too much data at once.

4. Assigning Orders to Riders Using BallTree

The code then uses a function to match orders with riders based on how close the riders are to the restaurant:

- First, the riders' locations (latitude and longitude) are converted into a format that the `BallTree` can use to find the closest rider.
- The BallTree is like a tool that helps quickly find which rider is closest to a particular location (in this case, the restaurant).
- For each order, the code checks the location of the restaurant and then finds the 5 closest riders.
- Out of the 5 closest riders, it picks the rider who is closest to the restaurant and assigns that rider to the order.
- The program then calculates the total distance the rider will have to travel from the restaurant to the customer's location.
- If the distance is less than or equal to 5 kilometers, the order is assigned to the rider, and the match is saved.

5. Saving the Assignments

Once the orders are matched with riders, the results are saved into a new file. This keeps track of which rider was assigned to which order, and the distance they have to travel.

6. Showing the First Few Matches

The code prints out the first few rows of the matched data (rider + order + distance) to check that everything worked correctly.

7. Summary of Assignments

The program then creates a summary of the order-rider assignments. It provides basic statistics, such as:

- The average distance that riders have to travel to deliver the orders.

8. Creating a Scatter Plot

The code generates a scatter plot to show how many orders each rider was assigned. It adds a bit of randomness (jitter) to the points so they don't overlap and are easier to see on the graph. The plot shows:

- Rider IDs on the x-axis (who the rider is).
- Number of orders on the y-axis (how many orders they were assigned).

9. Creating a Histogram

The code also creates a histogram to show the distribution of delivery distances. This means it will show how often certain distances occurred:

- For example, it might show that most deliveries were 2-3 km, with fewer deliveries over 4-5 km.

10. Summarizing the Results

Finally, the program prints a summary of how well the assignments worked:
- The total number of orders processed.
- The number of orders that were successfully assigned to riders.
- The number of unassigned orders (if any).
- The average delivery distance the riders had to travel.

Code 3 Explanation:

1. Importing Libraries

The program starts by loading three tools (libraries):
- numpy: Helps with working on numbers and mathematical operations.
- pandas: Allows for handling and analyzing data in tables (like spreadsheets).
- matplotlib.pyplot: A tool used to create different types of graphs and charts.

2. Loading Data

The code loads two datasets:
- optimized_assignments: This contains order assignments that were made using an optimized approach (probably assigning riders in a way that minimizes travel distance).
- inefficient_random_order_assignments: This contains order assignments made using a random and inefficient approach (riders were assigned orders randomly, without considering travel distance).

3. Adding Jitter to Avoid Overlapping Points

A jitter function is used to add a small random adjustment to the values when plotting them on a graph. This helps prevent the points from overlapping, so they are easier to see.

4. Creating a Plot for Comparison

The code then sets up four different plots (two scatter plots and two histograms) to compare the optimized and inefficient assignments.

a. Scatter Plot for Optimized Assignments

- The first scatter plot shows the number of orders assigned to each rider in the optimized approach.
- The x-axis represents the rider ID (who the rider is), and the y-axis represents the number of orders each rider got.
- The points on the scatter plot are spread out using jitter to avoid overlap.

b. Scatter Plot for Inefficient Random Assignments

- The second scatter plot shows the number of orders assigned per rider in the inefficient random approach.
- Like the first scatter plot, the x-axis is the rider ID, and the y-axis is the number of orders.
- Again, jitter is used to spread out the points.

c. Histogram for Delivery Distances (Optimized)

- The third plot is a histogram that shows the distribution of delivery distances for the optimized approach.

- A histogram groups distances into “bins” (ranges of values) and shows how often those distances occurred.
- The x-axis is the distance in kilometers, and the y-axis shows how many orders had that distance.

d. Histogram for Delivery Distances (Random Inefficient)

- The fourth plot is another histogram, this time showing the delivery distances for the random inefficient assignments.
- Like the optimized histogram, it shows how often different delivery distances occurred, but for the inefficient random assignments.

5. Displaying the Plots

After creating these four graphs, the code adjusts the layout to make sure everything is displayed neatly, and then it displays the plots so you can visually compare the optimized vs. random inefficient assignments.

6. Final Comparison Summary

The code calculates the average delivery distance for both the optimized and inefficient random assignments:
- It calculates how far riders had to travel on average in the optimized approach.
- It does the same for the random inefficient approach.

Finally, the program prints out a comparison summary, which shows:
- The average delivery distance for the optimized approach.
- The average delivery distance for the random inefficient approach.

This helps you easily see which method worked better in terms of minimizing the distance that riders had to travel.

Code 4 Explanation:

1. Loading the Optimized Assignment Results

- The program first loads the optimized assignment results from a file. This file, called `'order_assignments1.csv'`, contains data about orders and the riders assigned to them using an optimized method.

2. Setting Up Plots for Comparison

- The code then sets up two plots side-by-side (one for the optimized assignment and one for the inefficient random assignment) to compare the delivery distances for both methods.

3. Creating a Histogram for Optimized Delivery Distances

- In the first plot, the code creates a histogram to show the distribution of delivery distances for the optimized method.
- A histogram breaks the distances into "bins" (or ranges) and shows how many orders fell into each distance range.
- The x-axis shows the distance in kilometers, and the y-axis shows the number of orders that had that delivery distance.

4. Creating a Histogram for Random Inefficient Delivery Distances

- In the second plot, another histogram is created, this time for the inefficient random assignment method.
- Like the optimized histogram, it shows how many orders had different delivery distances but for the random, inefficient assignments.
- Again, the x-axis shows the distance in kilometers, and the y-axis shows the number of orders.

5. Displaying the Plots

- After creating the two histograms, the code adjusts the layout to make sure the plots look neat and don't overlap. Then, it displays both plots side by side so you can compare the delivery distances for the optimized and random inefficient methods.

6. Calculating and Comparing Average Delivery Distances

- The code then calculates the average delivery distance for both:
  - It finds out how far, on average, riders had to travel in the optimized assignment.
  - It also calculates the average distance for the random inefficient assignment.

7. Counting the Total Orders Assigned

- The program counts the total number of orders assigned in both methods:
  - The total number of orders in the optimized assignment.
  - The total number of orders in the random inefficient assignment.

8. Creating a Summary of the Comparison

- The final step is to create a summary that compares the two methods.
- The summary includes:
  - The average delivery distance for the optimized method.
  - The average delivery distance for the random inefficient method.
  - The total number of orders assigned in both methods.
- The program prints out this summary so you can easily see which method resulted in shorter delivery distances and how many orders were assigned in each method.

Code 5 Explanation:

1. Importing Libraries

The code starts by loading some tools (libraries) that are needed for different tasks:

- pandas: Helps in handling and analyzing data that is organized in tables.
- geopy.distance: Used to calculate the distance between two locations on Earth.
- matplotlib.pyplot: A tool for creating visual graphs and charts.
- numpy: A tool for working with numerical data and performing mathematical operations.

2. Loading Data

The code loads several datasets into the program:

- `optimized_assignments`: This is a file that contains information about orders and their assigned riders using an optimized method. It's loaded from `'order_assignments1.csv'`.
- `inefficient_random_assignments`: This file contains information about orders and their assigned riders using a random, inefficient method. It's loaded from `'inefficient_random_order_assignments.csv'`.

3. Loading Additional Information

The code also loads additional information needed for distance calculations:
- `rider_info`: This file contains details about the riders, such as their location (latitude and longitude). It's loaded from `'cleaned_dispatch_rider_meituan.csv'`.
- `waybill_info`: This file contains details about the orders, such as where they are coming from (the sender's location) and where they are going (the recipient's location). It's loaded from `'cleaned_all_waybill_info_meituan.csv'`.

In summary:
- The code starts by importing the necessary tools.
- It then loads data about orders and their assignments (both optimized and random) and additional information about riders and orders to perform further analysis or calculations.

Code 6 Explanation:

1. Defining a Function to Calculate Distances

The code starts by defining a function called `rider_to_restaurant_distances`. This function calculates how far each rider has to travel to pick up orders from the restaurant.

Inside the Function:
- `assignments`: This is a list of orders and which riders are assigned to them.
- `rider_info`: Contains details about the riders, including their locations.
- `waybill_info`: Contains details about the orders, including the restaurant locations.

Steps the Function Takes:

1. Create an Empty List: A list called `distances` is created to store the distances.

2. Loop Through Each Assignment: For each order and rider assignment:
  - Find the Rider: Gets the rider's location by looking up their ID in the `rider_info` data.
  - Find the Order: Gets the restaurant's location by looking up the order ID in the `waybill_info` data.
  - Calculate the Distance: Uses the `geodesic` tool to calculate how far the rider is from the restaurant, in kilometers.
  - Store the Distance: Adds this distance to the `distances` list.

3. Return the List: The function returns the list of distances.

2. Calculating Distances for Both Assignment Methods
- `optimized_rider_restaurant_distances`: Uses the function to calculate distances for orders assigned using the optimized method.
- `random_rider_restaurant_distances`: Uses the same function to calculate distances for orders assigned randomly.

3. Creating a Comparison Plot

The code then makes a graph to compare the distances:
- Create a Histogram:
  - A histogram is a type of graph that shows how many times different distances occurred.

- Two histograms are plotted on the same graph:
  - One for distances using the optimized method.
  - One for distances using the random method.
- Customize the Plot:
  - Title: Gives the graph a title - “Rider to Restaurant Distance Comparison”.
  - Labels: The x-axis shows the distance in kilometers, and the y-axis shows the number of orders at each distance.
  - Legend: A small box that helps you understand which color represents which method.

4. Display the Graph

- The graph is displayed so you can visually compare the distances traveled by riders in the optimized versus random methods.

In summary:

- The function calculates the distance each rider traveled to pick up an order.
- The code then calculates these distances for both optimized and random methods.
- Finally, it creates and shows a graph comparing these distances.

Code 7 Explanation:

1. Defining a Function to Count Distance Violations

The code starts by defining a function called `count_distance_violations`. This function checks how many orders have a delivery distance that is greater than 5 kilometers, which is considered a violation of the rule.

Inside the Function:

- `assignments`: This is a list of orders with the distances they cover.

Steps the Function Takes:

1. Find Violations:

  - Filter Violations: The function looks at all the orders where the `total_distance` is more than 5 kilometers.
  - Count Violations: It counts how many of these orders there are.

2. Total Orders:

  - Count Total Orders: It counts the total number of orders in the list.

3. Return Results:

  - The function returns two numbers:
    - Number of Violations: Orders where the distance is greater than 5 kilometers.
    - Total Number of Orders: All orders in the list.

2. Counting Violations for Both Methods

- Optimized Method: The function is used to count how many orders violated the 5 km rule in the optimized assignment list.
  - `optimized_violations`: Number of violations.
  - `total_orders_optimized`: Total number of orders in the optimized method.
  - Random Inefficient Method: The function is also used for the random inefficient assignment list.
  - `random_violations`: Number of violations.
  - `total_orders_random`: Total number of orders in the random method.

3. Creating a Bar Chart for Comparison

The code then makes a bar chart to compare the percentage of orders that violated the 5 km rule for both methods:

- Labels: Two labels are used for the chart:
  - 'Optimized'
  - 'Random Inefficient'
- Calculate Percentages:
  - Optimized Percentage: The number of violations in the optimized method divided by the total number of orders in that method.
  - Random Inefficient Percentage: The number of violations in the random method divided by the total number of orders in that method.
- Create the Bar Chart:
  - Figure Size: The chart is sized to be 6x6 inches.
  - Bars: One bar for each method showing the percentage of orders violating the rule.
  - Colors: Bars are colored green for the optimized method and red for the random inefficient method.
  - Title and Labels:
    - The title of the chart is “Percentage of Orders Violating the 5 km Rule”.
    - The y-axis is labeled “Violation Percentage”.
- Display the Chart: Finally, the bar chart is shown so you can easily compare the percentages of violations between the two methods.

In summary:

- The function counts how many orders exceed the 5 km distance rule and calculates the total number of orders.
- This count is done for both optimized and random methods.
- A bar chart is then created to compare the percentage of violations between these two methods.

Code 8 Explanation:

1. Defining a Function to Calculate Delivery Time

The code starts by defining a function called `calculate_delivery_time`. This function calculates how long it takes to deliver an order based on its distance.

Inside the Function:

- `assignments`: This is a list of orders with their distances.
- `speed_kmh`: This is the speed at which deliveries are made, given in kilometers per hour (km/h). The default speed is 20 km/h.

Steps the Function Takes:

1. Calculate Delivery Time:

- Formula: The time to deliver an order is calculated using the formula:
- Convert Hours to Minutes: The distance divided by speed gives the time in hours, so we multiply by 60 to convert it into minutes.

2. Return Delivery Time:

- The function returns a list of delivery times for all orders.

2. Calculating Delivery Times for Both Methods

- Optimized Assignments: The function is used to calculate the delivery times for the orders in the optimized method.
  - `optimized_delivery_time`: This is a list of delivery times for the optimized orders.
- Random Inefficient Assignments: The function is also used to calculate delivery times for the random inefficient method.
  - `random_delivery_time`: This is a list of delivery times for the random inefficient orders.

3. Creating a Histogram to Compare Delivery Times

The code then creates a histogram to compare the delivery times for both methods:

- Create the Histogram:
  - Figure Size: The histogram is sized to be 10x5 inches.
  - Histograms: Two histograms are plotted on the same graph:
    - One for delivery times using the optimized method.
    - One for delivery times using the random method.
  - Colors and Transparency: Each histogram has a different color and some transparency to show both sets of data on the same graph.
  - Title and Labels:
    - The title of the graph is “Delivery Time Comparison (Optimized vs Random)”.
    - The x-axis shows the delivery time in minutes.
    - The y-axis shows the number of orders at each delivery time.
  - Legend: A small box that helps you understand which color represents which method.
- Display the Graph: Finally, the histogram is shown so you can easily compare the delivery times between the optimized and random methods.

In summary:
- The function calculates how long it takes to deliver an order based on its distance and a given speed.
- Delivery times are calculated for both optimized and random methods.
- A histogram is then created to compare the delivery times between the two methods.

Code 9 Explanation:

1. Calculating Total Distance Traveled

- For Optimized Method:
  - `total_distance_optimized`: This calculates the total distance traveled by all riders in the optimized method. It adds up all the distances from the optimized assignment list.
- For Random Inefficient Method:
  - `total_distance_random`: This calculates the total distance traveled by all riders in the random inefficient method. It adds up all the distances from the random assignment list.

2. Counting the Number of Orders

- For Optimized Method:
  - `total_orders_optimized`: This counts how many orders were assigned in the optimized method. It's simply the number of rows in the optimized assignment list.
- For Random Inefficient Method:
  - `total_orders_random`: This counts how many orders were assigned in the random inefficient method. It's the number of rows in the random assignment list.

3. Calculating Orders per Kilometer

- For Optimized Method:
  - `orders_per_km_optimized`: This calculates how many orders were assigned per kilometer traveled in the optimized method. It's found by dividing the total number of orders by the total distance traveled.
- For Random Inefficient Method:
  - `orders_per_km_random`: This calculates how many orders were assigned per kilometer traveled in the random inefficient method. It's found by dividing the total number of orders by the total distance traveled.

4. Printing the Results

- The code prints out the number of orders per kilometer for both methods:
  - Optimized Method: Shows how efficient the optimized method is in terms of orders per kilometer.
  - Random Inefficient Method: Shows how efficient the random method is in terms of orders per kilometer.

5. Creating a Bar Chart for Comparison

- *Labels:*
  - Two bars are used in the chart:
    - Optimized: For the optimized method.
    - Random Inefficient: For the random inefficient method.

- *Values:*
  - `orders_per_km_values`: A list that contains the number of orders per kilometer for both methods.
- *Create the Bar Chart:*
  - Figure Size: The chart is sized to be 6x6 inches.
  - Bars: One bar for each method showing how many orders were assigned per kilometer.
  - Colors: Bars are colored green for the optimized method and red for the random inefficient method.
  - *Title and Labels*:
    - The title of the chart is “Orders per Kilometer Comparison”.
    - The y-axis shows the number of orders per kilometer.
- *Display the Chart:* Finally, the bar chart is shown so you can easily compare the efficiency of the two methods in terms of orders per kilometer.

In summary:
- Total distances traveled by riders and the number of orders are calculated for both optimized and random methods.
- The efficiency of each method is measured by how many orders were assigned per kilometer traveled.
- A bar chart is created to visually compare the orders per kilometer between the two methods.

Code 10 Explanation:

1. Set the Average Speed
- `speed_kmh`: This sets the average speed of the riders to 20 kilometers per hour (km/h). This speed will be used to calculate how long it takes to deliver an order.

2. Calculate Delivery Time
- For Optimized Method:
  - `optimized_delivery_time`: This calculates how long it takes to deliver each order in the optimized method.
  - Formula:
    - Divide the distance by the speed to get the time in hours.
    - Multiply by 60 to convert hours into minutes.
- For Random Inefficient Method:
  - `random_delivery_time`: This calculates the delivery time for each order in the random inefficient method using the same formula.

3. Calculate Average Delivery Time
- For Optimized Method:
  - `avg_delivery_time_optimized`: This finds the average delivery time for all orders in the optimized method.
  - `mean()`: Calculates the average of all the delivery times.

- For Random Inefficient Method:
  - `avg_delivery_time_random`: This finds the average delivery time for all orders in the random inefficient method.

4. Print the Results

- The code prints out the average delivery time per order for both methods:
  - Optimized Method: Shows the average time it takes to deliver an order using the optimized method.
  - Random Inefficient Method: Shows the average time it takes to deliver an order using the random inefficient method.

5. Create a Bar Chart for Comparison

- Labels:
  - Two bars are used in the chart:
    - Optimized: For the optimized method.
    - Random Inefficient: For the random inefficient method.
- Values:
  - `avg_delivery_time_values`: A list containing the average delivery times for both methods.
- Create the Bar Chart:
  - Figure Size: The chart is sized to be 6x6 inches.
  - Bars: One bar for each method showing the average delivery time.
  - Colors: Bars are colored blue for the optimized method and orange for the random inefficient method.
  - Title and Labels:
    - The title of the chart is “Average Delivery Time per Order”.
    - The y-axis shows the time in minutes.
- Display the Chart: Finally, the bar chart is shown so you can easily compare the average delivery time between the two methods.

In summary:

- The code calculates how long it takes to deliver an order based on distance and an average speed of 20 km/h.
- It finds the average delivery time for both optimized and random methods.
- A bar chart is created to visually compare the average delivery times between the two methods.

Code 11 Explanation:

1. Set Up Average Speed

- `speed_kmh`: This sets the average speed of the riders to 20 kilometers per hour (km/h). This speed will be used to calculate how long it takes to deliver an order.

2. Calculate Delivery Time

- For Optimized Method:
  - `optimized_delivery_time`: Calculates how long it takes to deliver each order in the optimized method.
    Formula:
    - Divide the distance by the speed to get the time in hours.
    - Multiply by 60 to convert hours into minutes.
- For Random Inefficient Method:
  - `random_delivery_time`: Calculates the delivery time for each order in the random inefficient method using the same formula.

3. Define Customer Satisfaction Function

- Function: `calculate_satisfaction`:
  - Purpose: To calculate a satisfaction score based on delivery time.
  - Threshold: 10 minutes is considered the best time for highest satisfaction.
  - Satisfaction Scores:
    - Best Score: 100 (highest satisfaction) if delivery is 10 minutes or less.
    - Decreasing Score: For delivery times greater than 10 minutes, satisfaction decreases linearly.
    - Minimum Score: The score will not drop below 50, even if the delivery time is very long.

4. Calculate Customer Satisfaction

- For Optimized Method:
  - `optimized_satisfaction`: Uses the `calculate_satisfaction` function to get satisfaction scores for each delivery time in the optimized method.
- For Random Inefficient Method:
  - `random_satisfaction`: Uses the same function to get satisfaction scores for each delivery time in the random inefficient method.

5. Compare Average Satisfaction Scores

- *Average Satisfaction:*
  - For Optimized Method:
    - `avg_satisfaction_optimized`: Calculates the average satisfaction score for the optimized method.
  - For Random Inefficient Method:
    - `avg_satisfaction_random`: Calculates the average satisfaction score for the random inefficient method.
- Print Results:
  - Displays the average customer satisfaction scores for both methods.

6. Plot Customer Satisfaction Distributions

- Two Histograms:
  - For Optimized Method:
    - `plt.subplot(1, 2, 1)`: Creates a histogram showing the distribution of satisfaction scores for the optimized method.
    - Color: Green.
  - For Random Inefficient Method:
    - `plt.subplot(1, 2, 2)`: Creates a histogram showing the distribution of satisfaction scores for the random inefficient method.
    - Color: Red.
  - Both Histograms:
    - Show how often different satisfaction scores occur.
    - Use bins to group scores and display their frequency.

7. Plot Average Satisfaction Comparison

- Bar Chart:
  - Labels: "Optimized" and "Random Inefficient" for the two methods.
  - Values: Average satisfaction scores for both methods.
  - Color: Green for optimized, red for random inefficient.
  - Show: A bar chart comparing average satisfaction scores visually.

In summary:

- The code calculates how long it takes to deliver each order.
- It then calculates customer satisfaction based on delivery times.
- It compares satisfaction scores between optimized and random methods.
- Finally, it creates visual plots to show the distribution and average satisfaction scores.

Code 12 Explanation:

1. Calculate Average Delivery Distance

- For Optimized Method:
  - `avg_distance_optimized`: Calculates the average distance of deliveries in the optimized method.
  - Formula: Computes the mean of all delivery distances listed in `optimized_assignments['total_distance']`.
- For Random Inefficient Method:
  - `avg_distance_random`: Calculates the average distance of deliveries in the random inefficient method.
  - Formula: Computes the mean of all delivery distances listed in `inefficient_random_assignments['total_distance']`.

2. Count Total Orders

- For Optimized Method:
  - `total_orders_optimized`: Counts the total number of orders assigned in the optimized method.
  - Formula: Uses `shape[0]` to get the number of rows (orders) in `optimized_assignments`.
- For Random Inefficient Method:
  - `total_orders_random`: Counts the total number of orders assigned in the random inefficient method.
  - Formula: Uses `shape[0]` to get the number of rows (orders) in `inefficient_random_assignments`.

3. Calculate Violation Percentages

- For Optimized Method:
  - `optimized_violations_percentage`: Calculates the percentage of orders that violated the 5 km rule in the optimized method.
  - Formula:
    - Divides the number of violations by the total number of orders.
    - Multiplies by 100 to convert it into a percentage.
- For Random Inefficient Method:
  - `random_violations_percentage`: Calculates the percentage of orders that violated the 5 km rule in the random inefficient method.
  - Formula: Same as for optimized.

4. Prepare and Print Summary

- `summary`: Creates a formatted string that summarizes key comparisons between the optimized and random inefficient methods.
  - Average Delivery Distance:
    - Displays the average distance for both methods.
  - Total Orders Assigned:
    - Shows the total number of orders assigned for both methods.
  - Violations of 5 km Rule:
    - Lists the number of violations and the percentage of total orders that violated the 5 km rule.
  - Estimated Average Delivery Time:
    - Shows the average delivery time in minutes for both methods (note: average delivery time needs to be calculated before this).
- Print:
  - `print(summary)`: Outputs the summary to the console.

In summary:

- The code calculates and compares average delivery distances and total orders assigned for both optimized and random methods.
- It also computes and compares the percentage of violations against the 5 km rule.
- Finally, it creates a formatted summary with these metrics and prints it.