Q1. How do you load a CSV file into a Pandas DataFrame?

To load a CSV file into a Pandas DataFrame, you can use the read\_csv() function from Pandas. Here's how you do it:

pythonCopy code

import pandas as pd # Replace 'file\_path' with the actual path to your CSV file df = pd.read\_csv('file\_path')

Q2. How do you check the data type of a column in a Pandas DataFrame?

To check the data type of a column in a Pandas DataFrame, you can use the dtypes attribute or the dtype method on the specific column. Here's how you do it:

pythonCopy code

# Using dtypes attribute to check data types of all columns print(df.dtypes) # Using dtype method to check the data type of a specific column (replace 'column\_name' with the actual column name) print(df['column\_name'].dtype)

Q3. How do you select rows from a Pandas DataFrame based on a condition?

You can use boolean indexing to select rows from a Pandas DataFrame based on a condition. Here's an example:

pythonCopy code

# Replace 'condition' with your actual condition filtered\_df = df[df['column\_name'] > condition]

Q4. How do you rename columns in a Pandas DataFrame?

To rename columns in a Pandas DataFrame, you can use the rename() method. Here's an example:

pythonCopy code

# Replace 'old\_column\_name' and 'new\_column\_name' with the actual column names df.rename(columns={'old\_column\_name': 'new\_column\_name'}, inplace=True)

Q5. How do you drop columns in a Pandas DataFrame?

To drop columns from a Pandas DataFrame, you can use the drop() method. Here's an example:

pythonCopy code

# Replace 'column\_name' with the actual column name you want to drop df.drop(columns=['column\_name'], inplace=True)

Q6. How do you find the unique values in a column of a Pandas DataFrame?

You can use the unique() method on a specific column to find the unique values. Here's an example:

pythonCopy code

# Replace 'column\_name' with the actual column name unique\_values = df['column\_name'].unique() print(unique\_values)

Q7. How do you find the number of missing values in each column of a Pandas DataFrame?

You can use the isnull() method to check for missing values and then use the sum() method to count them. Here's how you do it:

pythonCopy code

# Count the number of missing values in each column missing\_values\_count = df.isnull().sum() print(missing\_values\_count)

Q8. How do you fill missing values in a Pandas DataFrame with a specific value?

You can use the fillna() method to fill missing values in a Pandas DataFrame with a specific value. Here's an example:

pythonCopy code

# Replace 'value' with the specific value you want to use to fill missing values df.fillna(value, inplace=True)

Q9. How do you concatenate two Pandas DataFrames?

To concatenate two Pandas DataFrames, you can use the concat() function from Pandas. Here's an example:

pythonCopy code

# Replace df1 and df2 with the DataFrames you want to concatenate result\_df = pd.concat([df1, df2])

Q10. How do you merge two Pandas DataFrames on a specific column?

To merge two Pandas DataFrames on a specific column, you can use the merge() function. Here's an example:

pythonCopy code

# Replace 'on\_column' with the column name on which you want to merge merged\_df = df1.merge(df2, on='on\_column', how='inner')

Q11. How do you group data in a Pandas DataFrame by a specific column and apply an aggregation function?

You can use the groupby() method along with an aggregation function to group data in a Pandas DataFrame and apply the function. Here's an example:

pythonCopy code

# Replace 'group\_column' with the column name you want to group by, and 'agg\_function' with the aggregation function you want to apply grouped\_data = df.groupby('group\_column').agg(agg\_function)

Q12. How do you pivot a Pandas DataFrame?

To pivot a Pandas DataFrame, you can use the pivot() method. Here's an example:

pythonCopy code

# Replace 'index\_column', 'columns\_to\_pivot', and 'values\_column' with the actual column names pivot\_table = df.pivot(index='index\_column', columns='columns\_to\_pivot', values='values\_column')

Q13. How do you change the data type of a column in a Pandas DataFrame?

You can use the astype() method to change the data type of a column in a Pandas DataFrame. Here's an example:

pythonCopy code

# Replace 'column\_name' with the actual column name and 'new\_data\_type' with the desired data type df['column\_name'] = df['column\_name'].astype(new\_data\_type)

Q14. How do you sort a Pandas DataFrame by a specific column?

You can use the sort\_values() method to sort a Pandas DataFrame by a specific column. Here's an example:

pythonCopy code

# Replace 'column\_name' with the actual column name you want to sort by sorted\_df = df.sort\_values(by='column\_name', ascending=False)

Q15. How do you create a copy of a Pandas DataFrame?

You can use the copy() method to create a copy of a Pandas DataFrame. Here's an example:

pythonCopy code

df\_copy = df.copy()

Q16. How do you filter rows of a Pandas DataFrame by multiple conditions?

You can use the & operator for AND conditions and the | operator for OR conditions to filter rows based on multiple conditions. Here's an example:

pythonCopy code

# Replace 'condition1' and 'condition2' with your actual conditions filtered\_df = df[(df['column1'] > condition1) & (df['column2'] < condition2)]

Q17. How do you calculate the mean of a column in a Pandas DataFrame?

You can use the mean() method to calculate the mean of a column in a Pandas DataFrame. Here's an example:

pythonCopy code

# Replace 'column\_name' with the actual column name column\_mean = df['column\_name'].mean() print(column\_mean)

Q18. How do you calculate the standard deviation of a column in a Pandas DataFrame?

You can use the std() method to calculate the standard deviation of a column in a Pandas DataFrame. Here's an example:

pythonCopy code

# Replace 'column\_name' with the actual column name column\_std = df['column\_name'].std() print(column\_std)

Q19. How do you calculate the correlation between two columns in a Pandas DataFrame?

You can use the corr() method to calculate the correlation between two columns in a Pandas DataFrame. Here's an example:

pythonCopy code

# Replace 'column1' and 'column2' with the actual column names correlation = df['column1'].corr(df['column2']) print(correlation)

Q20. How do you select specific columns in a DataFrame using their labels?

You can use the column labels to select specific columns in a Pandas DataFrame. Here's an example:

pythonCopy code

# Replace 'column\_labels' with a list of the actual column labels you want to select selected\_columns = df[column\_labels]

Q21. How do you select specific rows in a DataFrame using their indexes?

You can use the loc accessor to select specific rows in a Pandas DataFrame using their indexes. Here's an example:

pythonCopy code

# Replace 'row\_indexes' with a list of the actual row indexes you want to select selected\_rows = df.loc[row\_indexes]

Q22. How do you sort a DataFrame by a specific column?

I already answered this in Q14. Here's the example again:

pythonCopy code

# Replace 'column\_name' with the actual column name you want to sort by sorted\_df = df.sort\_values(by='column\_name', ascending=False)

Q23. How do you create a new column in a DataFrame based on the values of another column?

You can create a new column in a Pandas DataFrame based on the values of another column by using the following syntax:

pythonCopy code

# Replace 'new\_column\_name' with the desired name for the new column and 'column\_name' with the source column df['new\_column\_name'] = df['column\_name'] \* 2 # You can replace the expression with any other operation you want

Q24. How do you remove duplicates from a DataFrame?

You can use the drop\_duplicates() method to remove duplicates from a Pandas DataFrame. Here's an example:

pythonCopy code

df.drop\_duplicates(inplace=True)

Q25. What is the difference between .loc and .iloc in Pandas?

.loc is used for label-based indexing, meaning you use the row and column labels to access data. On the other hand, .iloc is used for integer-based indexing, meaning you use integer positions to access data.

Here's an example to illustrate the difference:

pythonCopy code

# Using .loc to access data by label value\_at\_label = df.loc[2, 'column\_name'] # Using .iloc to access data by integer position value\_at\_position = df.iloc[2, 0]

In this example, df.loc[2, 'column\_name'] would return the value in the row with label '2' and the column with the label 'column\_name', whereas df.iloc[2, 0] would return the value in the third row and the first column (position 2, 0-based indexing).