

Questions

- Q1. How do you load a CSV file into a Pandas DataFrame?
- Q2. How do you check the data type of a column in a Pandas DataFrame?
- Q3. How do you select rows from a Pandas DataFrame based on a condition?
- Q4. How do you rename columns in a Pandas DataFrame?
- Q5. How do you drop columns in a Pandas DataFrame?
- Q6. How do you find the unique values in a column of a Pandas DataFrame?
- Q7. How do you find the number of missing values in each column of a Pandas DataFrame?
- Q8. How do you fill missing values in a Pandas DataFrame with a specific value?
- Q9. How do you concatenate two Pandas DataFrames?
- Q10. How do you merge two Pandas DataFrames on a specific column?
- Q11. How do you group data in a Pandas DataFrame by a specific column and apply an aggregation function?
- Q12. How do you pivot a Pandas DataFrame?
- Q13. How do you change the data type of a column in a Pandas DataFrame?
- Q14. How do you sort a Pandas DataFrame by a specific column?
- Q15. How do you create a copy of a Pandas DataFrame?
- Q16. How do you filter rows of a Pandas DataFrame by multiple conditions?
- Q17. How do you calculate the mean of a column in a Pandas DataFrame?
- Q18. How do you calculate the standard deviation of a column in a Pandas DataFrame?
- Q19. How do you calculate the correlation between two columns in a Pandas DataFrame?
- Q20. How do you select specific columns in a DataFrame using their labels?
- Q21. How do you select specific rows in a DataFrame using their indexes?
- Q22. How do you sort a DataFrame by a specific column?
- Q23. How do you create a new column in a DataFrame based on the values of another column?
- Q24. How do you remove duplicates from a DataFrame?
- Q25. What is the difference between `.loc` and `.iloc` in Pandas?

Answers

Q1. To load a CSV file into a Pandas DataFrame, you can use the `read_csv()` function. Here's an example:

```
```python
import pandas as pd

df = pd.read_csv('filename.csv')
```
```

Replace `'filename.csv'` with the actual path and filename of your CSV file.

Q2. To check the data type of a column in a Pandas DataFrame, you can use the `dtype` attribute of the DataFrame or the `dtypes` property. Here's an example:

```
```python
Check the data type of a single column
print(df['column_name'].dtype)

Check the data type of all columns
print(df.dtypes)
```
```

Replace `'column_name'` with the actual name of the column you want to check.

Q3. To select rows from a Pandas DataFrame based on a condition, you can use boolean indexing. Here's an example:

```
```python
Select rows where a column's value meets a condition
new_df = df[df['column_name'] > 5]
```
```

Replace `'column_name'` with the actual name of the column you want to use for the condition, and `> 5` with the specific condition you want to apply.

Q4. To rename columns in a Pandas DataFrame, you can use the `rename()` function. Here's an example:

```
```python
Rename a single column
df = df.rename(columns={'old_column_name': 'new_column_name'})

Rename multiple columns
df = df.rename(columns={'old_column1': 'new_column1', 'old_column2': 'new_column2'})
```
```

Replace `'old_column_name'` with the current name of the column you want to rename, and `'new_column_name'` with the new name you want to assign.

Q5. To drop columns in a Pandas DataFrame, you can use the `drop()` function. Here's an example:

```
```python
Drop a single column
df = df.drop('column_name', axis=1)

Drop multiple columns
df = df.drop(['column1', 'column2'], axis=1)
```
```

Replace `'column_name'` with the actual name of the column you want to drop, and `['column1', 'column2']` with a list of column names you want to drop.

Q6. To find the unique values in a column of a Pandas DataFrame, you can use the `unique()` function. Here's an example:

```
```python
unique_values = df['column_name'].unique()
```
```

Replace `'column_name'` with the actual name of the column you want to find the unique values for.

Q7. To find the number of missing values in each column of a Pandas DataFrame, you can use the `isnull()` function combined with the `sum()` function. Here's an example:

```
```python
missing_values = df.isnull().sum()
```
```

This will return a Series where the index represents column names, and the values represent the count of missing values in each column.

Q8. To fill missing values in a Pandas DataFrame with a specific value, you can use the `fillna()` function. Here's an example:

```
```python
df = df.fillna(value)
```
```

Replace `value` with the specific value you want to use for filling the missing values.

Q9. To concatenate two Pandas DataFrames, you can use the `concat()` function. Here's an example:

```
```python
new_df = pd.concat([df1, df2])
```
```

Replace `df1` and `df2` with the DataFrames you want to concatenate.

Q10. To merge two Pandas DataFrames on a specific column, you

can use the `merge()` function. Here's an example:

```
```python
merged_df = pd.merge(df1, df2, on='common_column')
```
```

Replace `df1` and `df2` with the DataFrames you want to merge, and `common_column` with the name of the column on which you want to merge.

Q11. To group data in a Pandas DataFrame by a specific column and apply an aggregation function, you can use the `groupby()` function. Here's an example:

```
```python
grouped_df = df.groupby('column_name')['column_to_aggregate'].agg(['mean', 'sum', 'count'])
```
```

Replace `column_name` with the name of the column you want to group by, and `column_to_aggregate` with the name of the column you want to apply the aggregation function on.

Q12. To pivot a Pandas DataFrame, you can use the `pivot()` function. Here's an example:

```
```python
pivot_df = df.pivot(index='index_column', columns='columns_column', values='values_column')
```
```

Replace `index_column`, `columns_column`, and `values_column` with the actual column names you want to use for the index, columns, and values of the pivoted DataFrame, respectively.

Q13. To change the data type of a column in a Pandas DataFrame, you can use the `astype()` function. Here's an example:

```
```python
df['column_name'] = df['column_name'].astype(new_data_type)
```
```

Replace `column_name` with the actual name of the column you want to change the data type of, and `new_data_type` with the desired data type, such as `int`, `float`, `str`, etc.

Q14. To sort a Pandas DataFrame by a specific column, you can use the `sort_values()` function. Here's an example:

```
```python
```

```
sorted_df = df.sort_values('column_name', ascending=False)
```

```
'''
```

Replace `'column_name'` with the name of the column you want to sort by. Set `ascending=False` if you want to sort in descending order.

Q15. To create a copy of a Pandas DataFrame, you can use the `copy()` function. Here's an example:

```
'''python
```

```
df_copy = df.copy()
```

```
'''
```

This creates a new DataFrame `df_copy` that is a separate copy of the original `df`.

Q16. To filter rows of a Pandas DataFrame by multiple conditions, you can use logical operators such as `&` (and) and `|` (or). Here's an example:

```
'''python
```

```
filtered_df = df[(df['column1'] > 5) & (df['column2'] == 'value')]
```

```
'''
```

Replace `'column1'`, `'column2'`, `> 5`, and `'value'` with your specific conditions.

Q17. To calculate the mean of a column in a Pandas DataFrame, you can use the `mean()` function. Here's an example:

```
'''python
```

```
mean_value = df['column_name'].mean()
```

```
'''
```

Replace `'column_name'` with the name of the column you want to calculate the mean for.

Q18. To calculate the standard deviation of a column in a Pandas DataFrame, you can use the `std()` function. Here's an example:

```
'''python
```

```
std_value = df['column_name'].std()
```

```
'''
```

Replace `'column_name'` with the name of the column you want to calculate the standard deviation for.

Q19. To calculate the correlation between two columns in a Pandas DataFrame, you can use the `'corr()'` function. Here's an example:

```
'''
```

```
python
```

```
correlation = df['column1'].corr(df['column2'])
```

```
'''
```

Replace `'column1'` and `'column2'` with the names of the columns you want to calculate the correlation between.

Q20. To select specific columns in a DataFrame using their labels, you can use the indexing operator `'[]'`. Here's an example:

```
'''python
```

```
selected_columns = df[['column1', 'column2', 'column3']]
```

```
'''
```

Replace `'column1'`, `'column2'`, and `'column3'` with the actual column labels you want to select.

Q21. To select specific rows in a DataFrame using their indexes, you can use the `'loc[]'` or `'iloc[]'` indexer. Here's an example:

```
'''python
```

```
selected_rows = df.loc[[0, 2, 4]]
```

```
'''
```

Replace `'[0, 2, 4]'` with the actual indexes of the rows you want to select.

Q22. To sort a DataFrame by a specific column, you can use the `'sort_values()'` function. Here's an example:

```
```python
sorted_df = df.sort_values('column_name')
```
```

Replace ``column\_name`` with the name of the column you want to sort by.

Q23. To create a new column in a DataFrame based on the values of another column, you can directly assign values to the new column. Here's an example:

```
```python
df['new_column'] = df['existing_column'] * 2
```
```

Replace ``new\_column`` with the name you want to assign to the new column, ``existing\_column`` with the name of the column whose values you want to use for the new column, and ``\* 2`` with the specific calculation or transformation you want to apply.

Q24. To remove duplicates from a DataFrame, you can use the ``drop\_duplicates()`` function. Here's an example:

```
```python
df = df.drop_duplicates()
```
```

This removes all duplicate rows from the DataFrame, keeping only the first occurrence of each unique row.

Q25. The difference between ``.loc`` and ``.iloc`` in Pandas is in the way they reference rows and columns:

- ``.loc`` is label-based and allows you to select rows and columns based on their labels or index values. It accepts label-based indexing for both rows and columns.
- ``.iloc`` is integer-based and allows you to select rows and columns based on their integer positions. It accepts integer-based indexing for both rows and columns.

Here's an example to illustrate the difference:

```
```python
```


Select a single value using .loc

```
value_loc = df.loc[row_label, column_label]
```

Select a single value using .iloc

```
value_iloc = df.iloc[row_index, column_index]
```

Select multiple rows using .loc

```
rows_loc = df.loc[start_label:end_label]
```

Select multiple rows using .iloc

```
rows_iloc = df.iloc[start_index:end_index]
```

Select columns using .loc

```
columns_loc = df.loc[:, ['column1', 'column2']]
```

Select columns using .iloc

```
columns_iloc = df.iloc[:, [0, 1]]
```

...

In the examples above, replace `row_label`, `column_label`, `row_index`, `column_index`, `start_label`, `end_label`, `start_index`, `end_index` with the appropriate labels or indices based on your DataFrame.