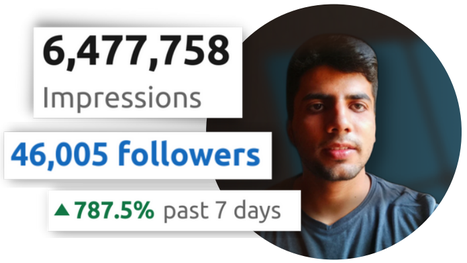
Save for Later

+ Follow

50+ Pandas Tips with code



@arjun-panwar

Use the .groupby() function to group data by a specific column and perform aggregate calculations. For example, to group a dataframe by the 'species' column and calculate the mean of the 'age' column for each group, you can use the following code:

```
df.groupby('species')['age'].mean()
```

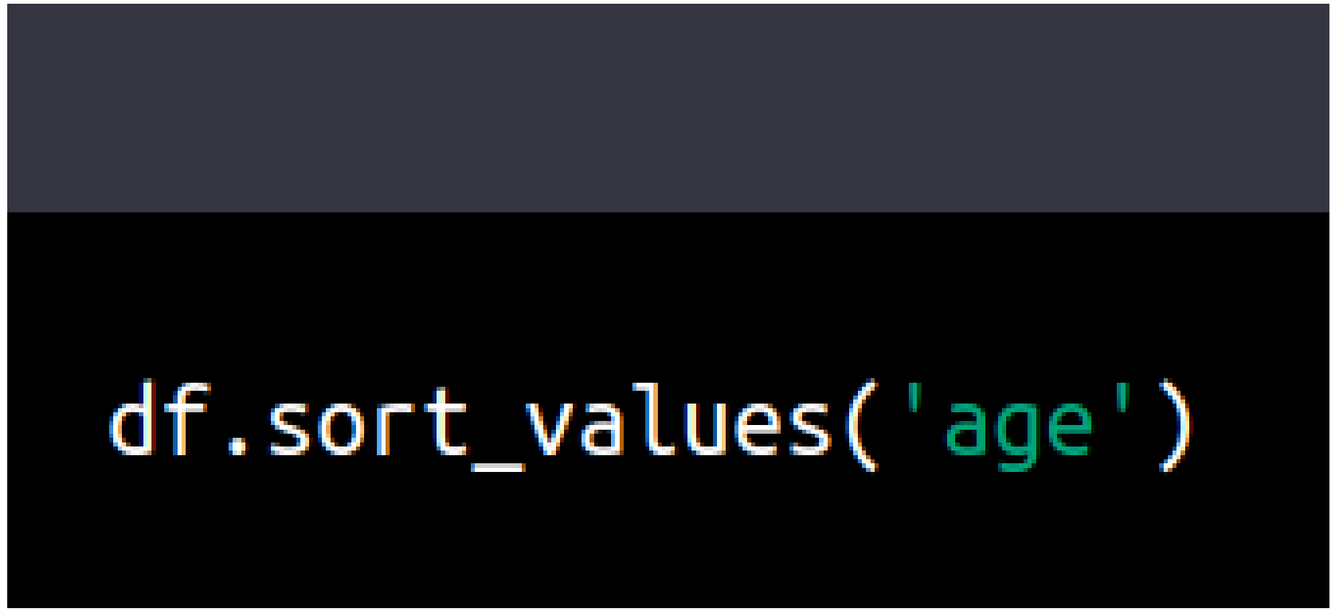
Use the .loc[] indexer to select specific rows and columns from a dataframe based on their labels. For example, to select all rows where the 'species' column is 'giant panda' and the 'age' column is greater than 5, you can use the following code:

```
df.loc[(df['species'] == 'giant panda') & (df['age'] > 5)]
```

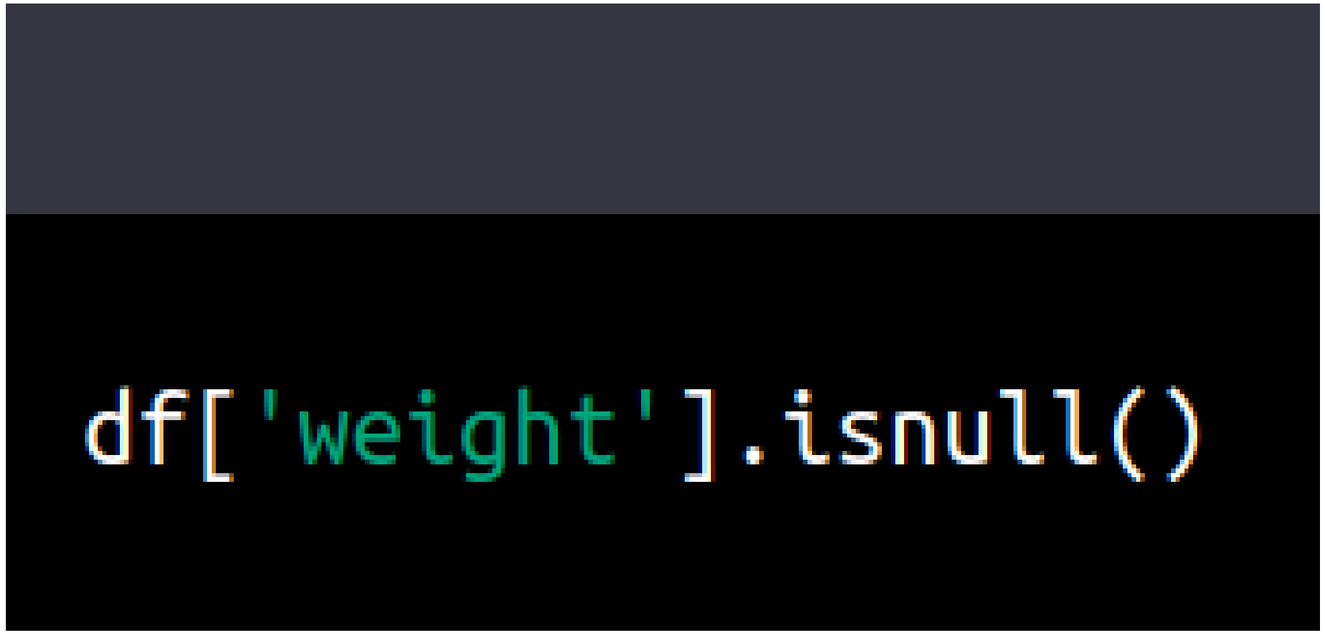
Use the .apply() function to apply a custom function to every element of a dataframe or series. For example, to convert all values in the 'weight' column to kilograms, you can use the following code:

```
df['weight'] = df['weight'].apply(lambda x: x * 0.453592)
```

Use the .sort_values() function to sort a dataframe by a specific column. For example, to sort a dataframe by the 'age' column in ascending order, you can use the following code:



Use the .isnull() function to check for missing values in a dataframe or series. For example, to check for missing values in the 'weight' column, you can use the following code:



Use the .pivot_table() function to create a pivot table from a dataframe. For example, to create a pivot table that shows the mean 'weight' for each 'species' and 'age' group, you can use the following code:

```
df.pivot_table(values='weight', index='species', columns='age', aggfunc='mean')
```

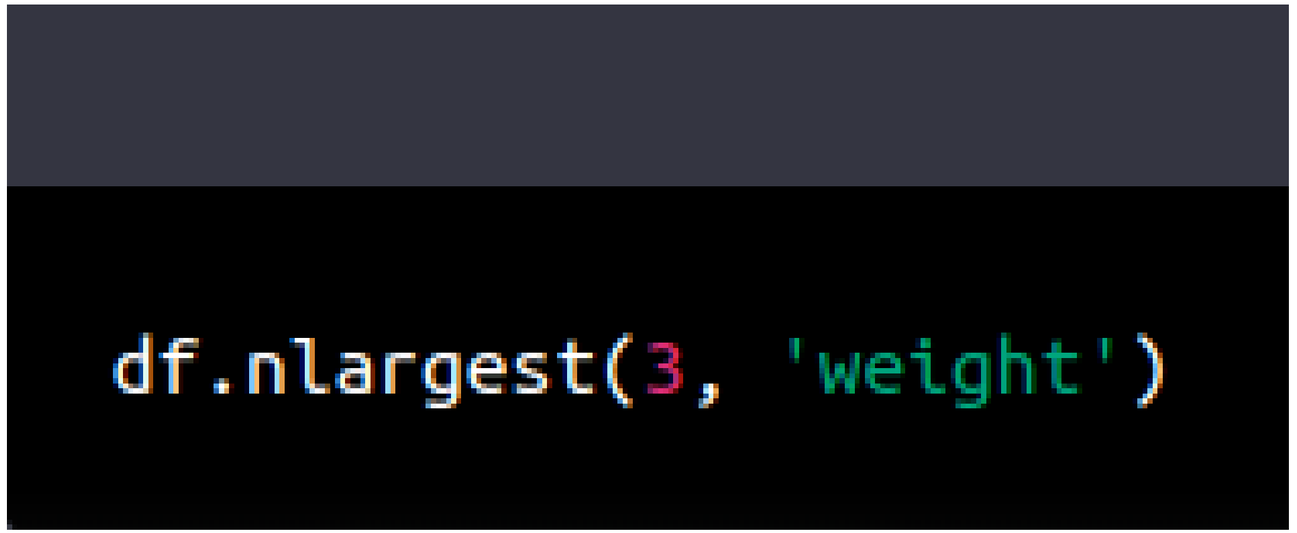
Use the .corr() function to calculate the correlation between columns in a dataframe. For example, to calculate the correlation between the 'weight' and 'height' columns, you can use the following code:

```
df[['weight', 'height']].corr()
```

Use the .rolling() function to apply a rolling window calculation to a column. For example, to calculate the rolling mean of the 'weight' column with a window of 3, you can use the following code:

```
df['weight'].rolling(3).mean()
```

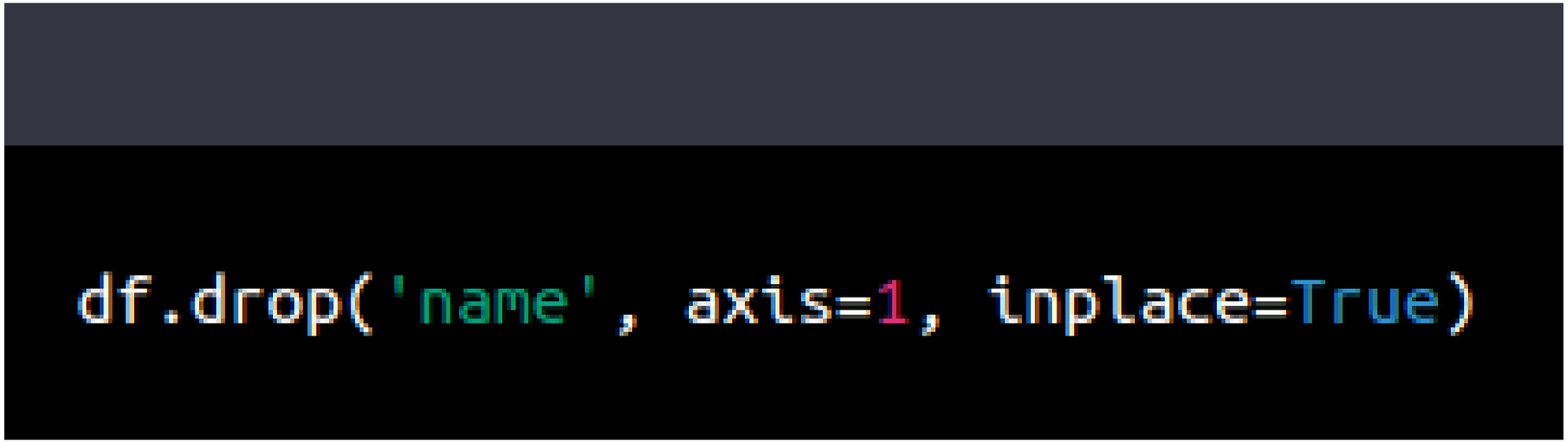
Use the .nlargest() function to select the top n rows of a dataframe based on a specific column. For example, to select the top 3 rows based on the 'weight' column, you can use the following code:



Use the .nsmallest() function to select the bottom n rows of a dataframe based on a specific column. For example, to select the bottom 3 rows based on the 'age' column, you can use the following code:

```
df.nsmallest(3, 'age')
```

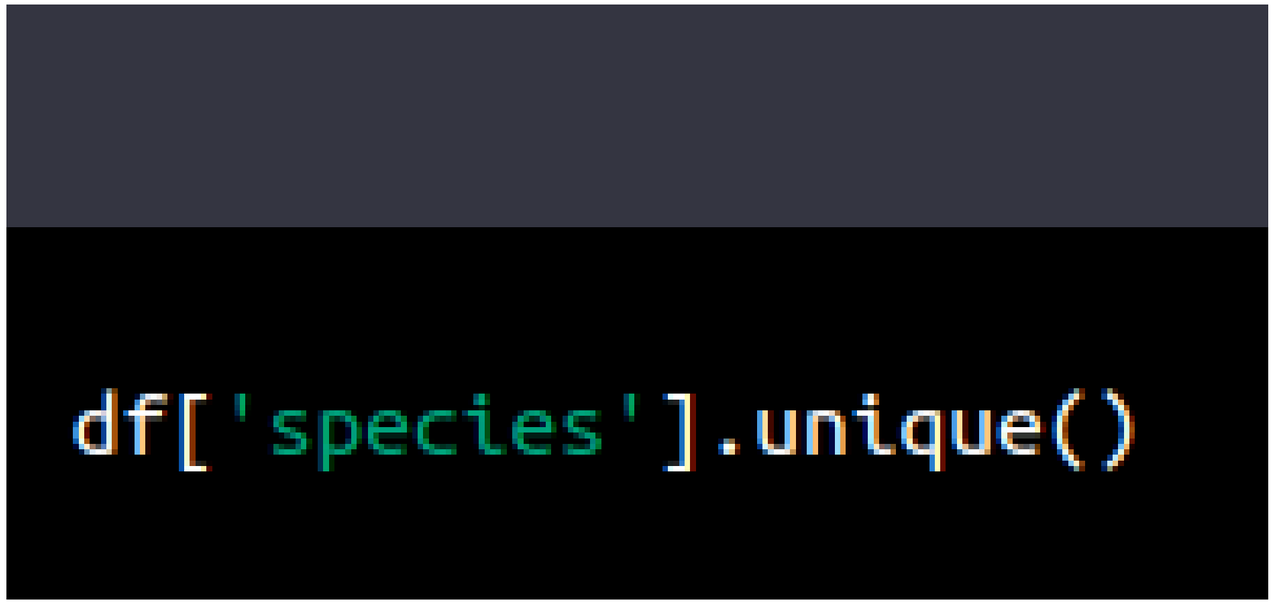
1. Use the .drop() function to drop specific rows or columns from a dataframe. For example, to drop the 'name' column from a dataframe, you can use the following code:



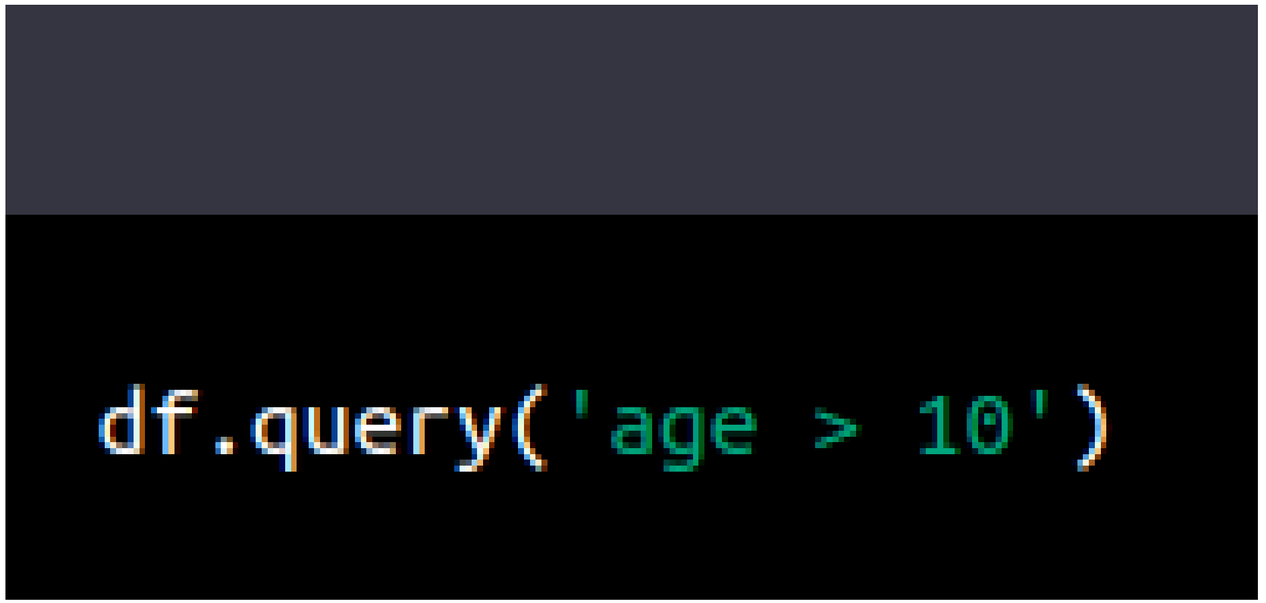
Use the .rename() function to rename columns in a dataframe. For example, to rename the 'species' column to 'type', you can use the following code:

```
df.rename(columns={'species': 'type'}, inplace=True)
```

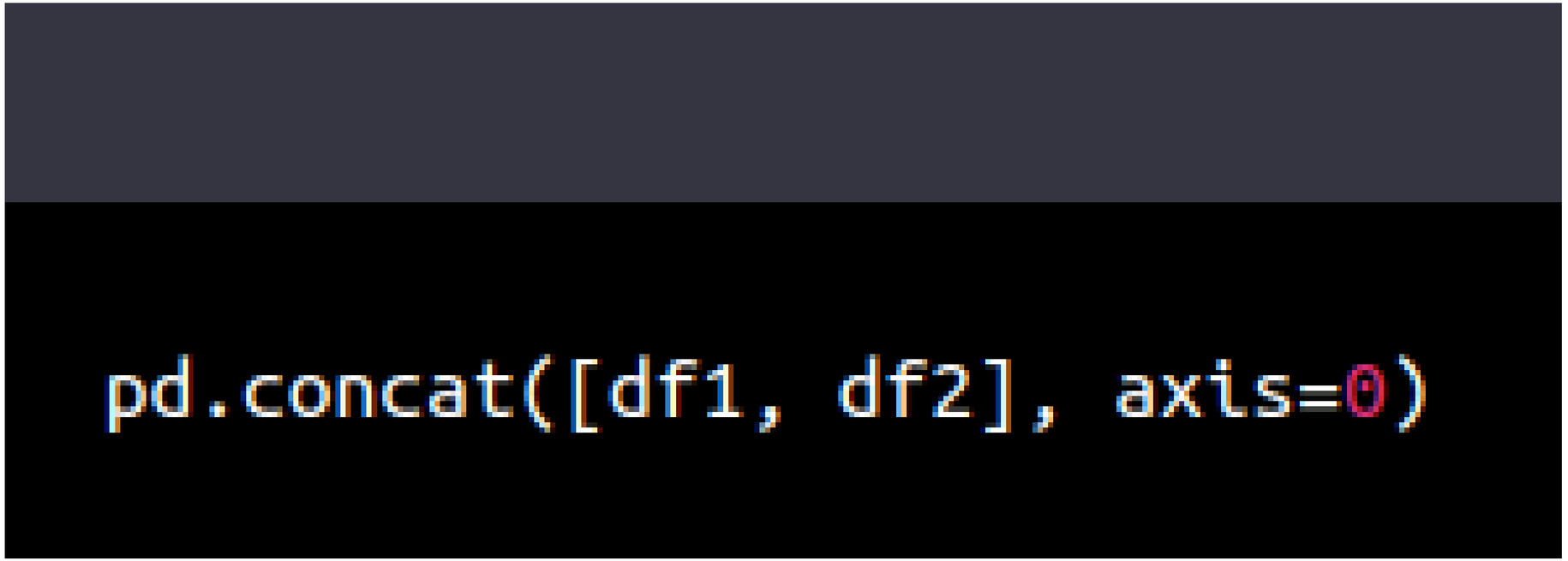
Use the .unique() function to find unique values in a column. For example, to find all unique values in the 'species' column, you can use the following code:



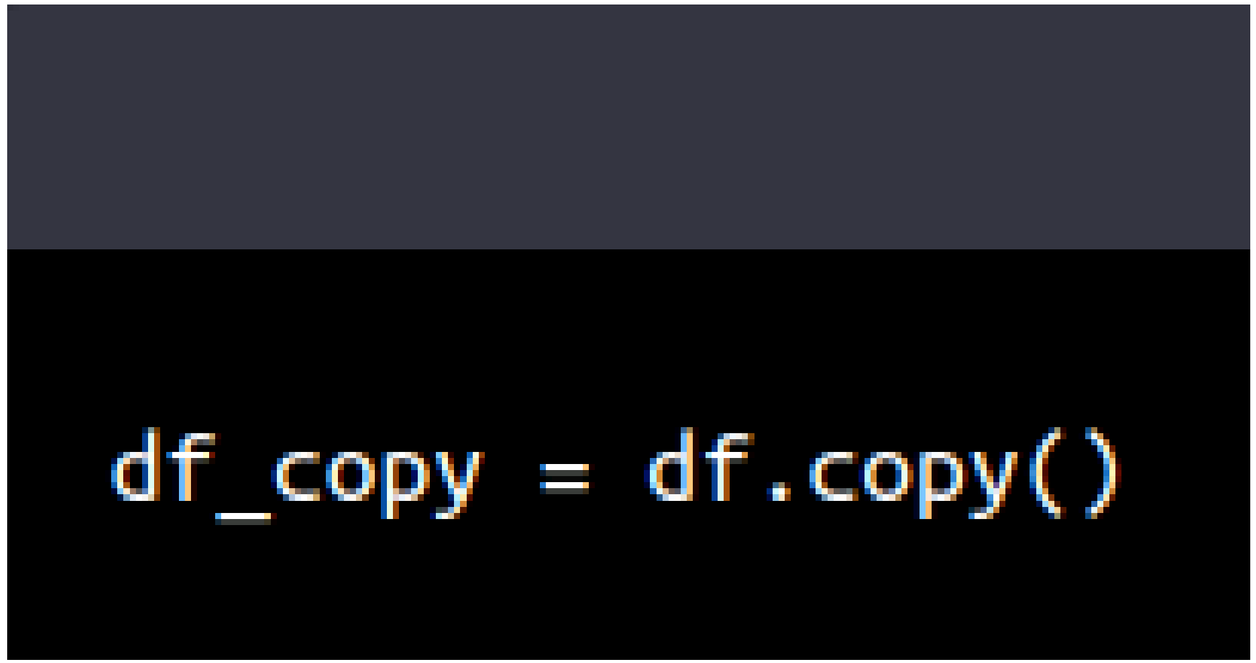
Use the .query() function to filter data based on conditions. For example, to select all rows where the 'age' column is greater than 10, you can use the following code:



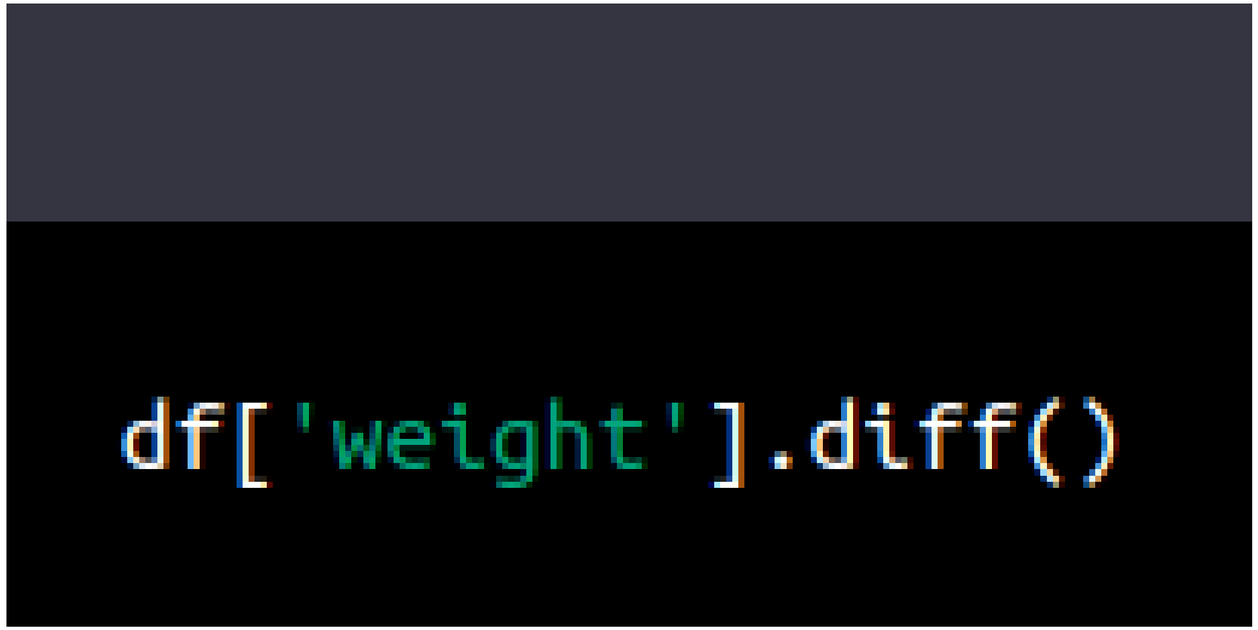
Use the .concat() function to concatenate multiple dataframes together. For example, to concatenate two dataframes df1 and df2 along the rows, you can use the following code:



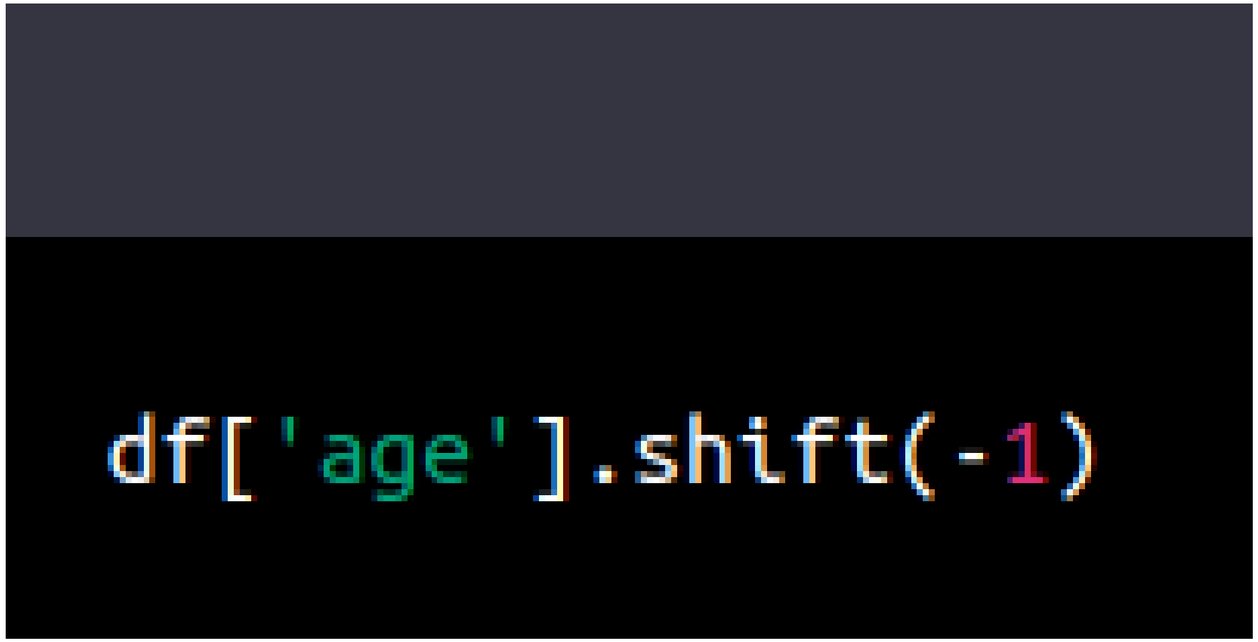
Use the .copy() function to create a copy of a dataframe, this will prevent any accidental modification in the original dataframe. For example, to create a copy of the dataframe 'df', you can use the following code:



Use the .diff() function to calculate the difference between consecutive rows in a column. For example, to calculate the difference in 'weight' between consecutive rows, you can use the following code:



Use the .shift() function to shift the values in a column by a certain number of rows. For example, to shift the values in the 'age' column up by 1 row, you can use the following code:



Use the .replace() function to replace specific values in a column. For example, to replace all occurrences of 'giant panda' in the 'species' column with 'Giant Panda', you can use the following code:

```
df['species'].replace('giant panda', 'Giant Panda', inplace=True)
```

Use the .applymap() function to apply a function to every element in a dataframe. For example, to convert all values in the dataframe to their absolute values, you can use the following code:

```
df.applymap(lambda x: abs(x))
```

Use the .get_dummies() function to create dummy variables from categorical columns. For example, to create dummy variables for the 'species' column, you can use the following code:

```
df = pd.get_dummies(df, columns=['species'])
```

Use the .agg() function to perform multiple aggregate calculations on a dataframe or series. For example, to calculate the mean, median, and standard deviation of the 'weight' column, you can use the following code:

```
df['weight'].agg(['mean', 'median', 'std'])
```

Use the .between() function to select rows where a column's values fall within a specific range. For example, to select all rows where the 'age' column is between 5 and 10, you can use the following code:

```
df[df['age'].between(5, 10)]
```

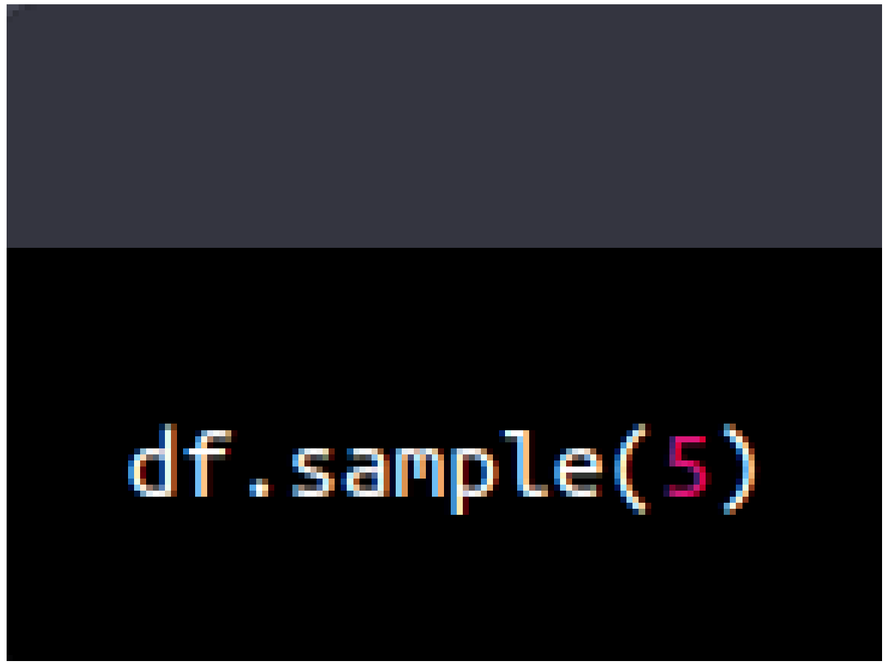
Use the .str.contains() function to filter rows based on a string match in a specific column. For example, to select all rows where the 'name' column contains the string 'panda', you can use the following code:

```
df[df['name'].str.contains('panda')]
```

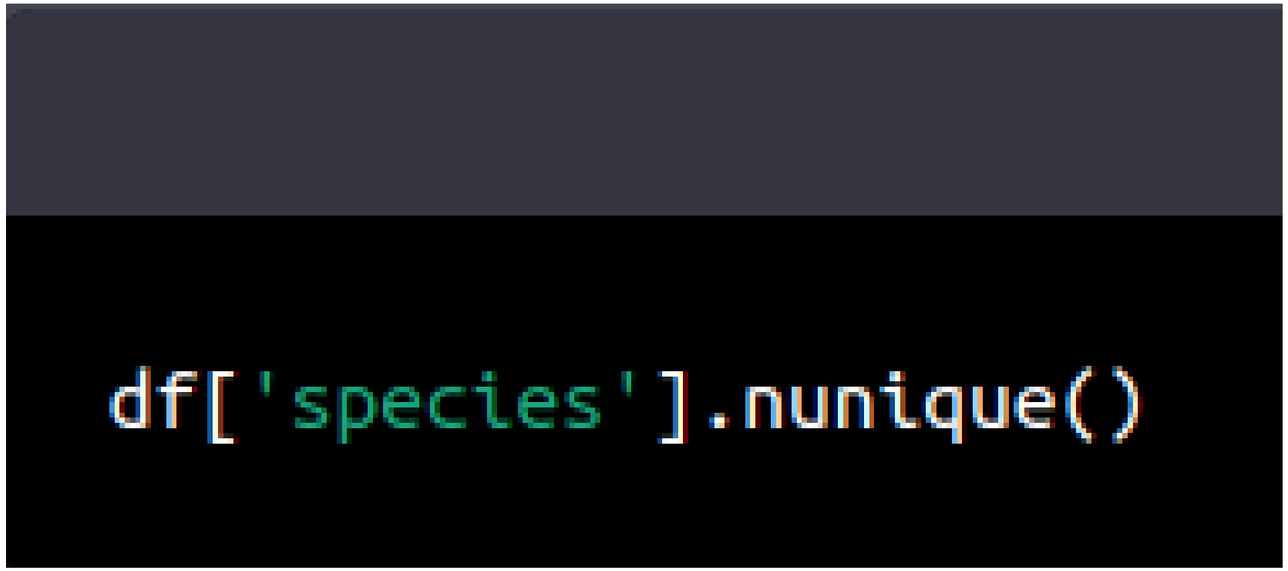
Use the .update() function to update the values in a dataframe or series. For example, to update the values in the 'weight' column by adding 10, you can use the following code:

```
df['weight'].update(df['weight'] + 10)
```

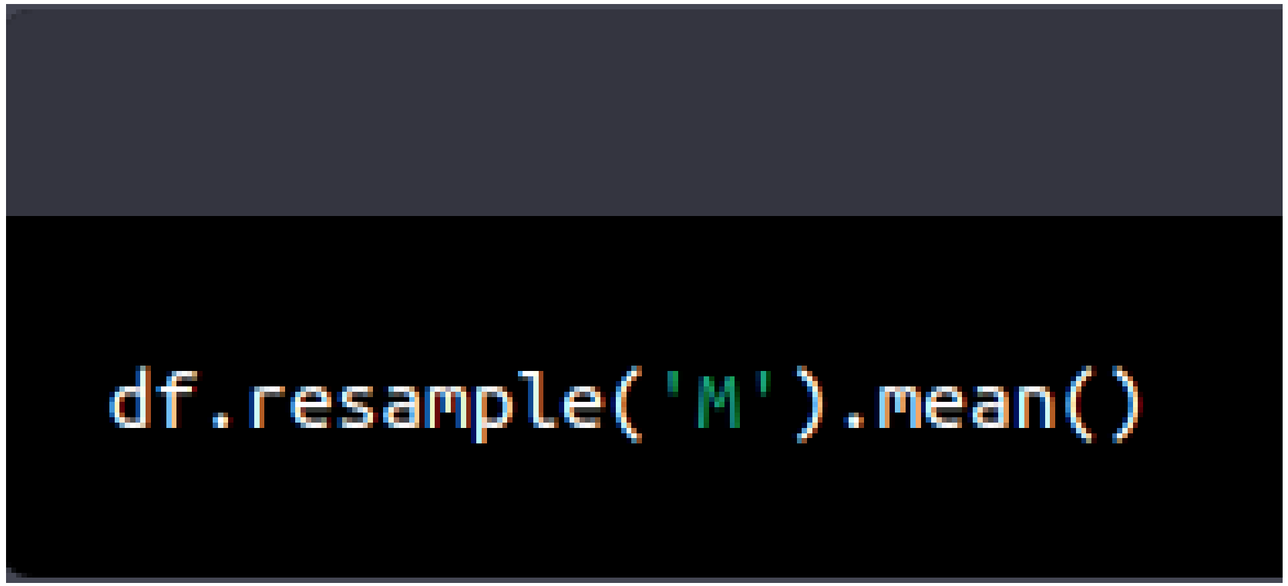
Use the .sample() function to randomly select rows from a dataframe. For example, to randomly select 5 rows from a dataframe, you can use the following code:



Use the .nunique() function to find the number of unique values in a column. For example, to find the number of unique values in the 'species' column, you can use the following code:



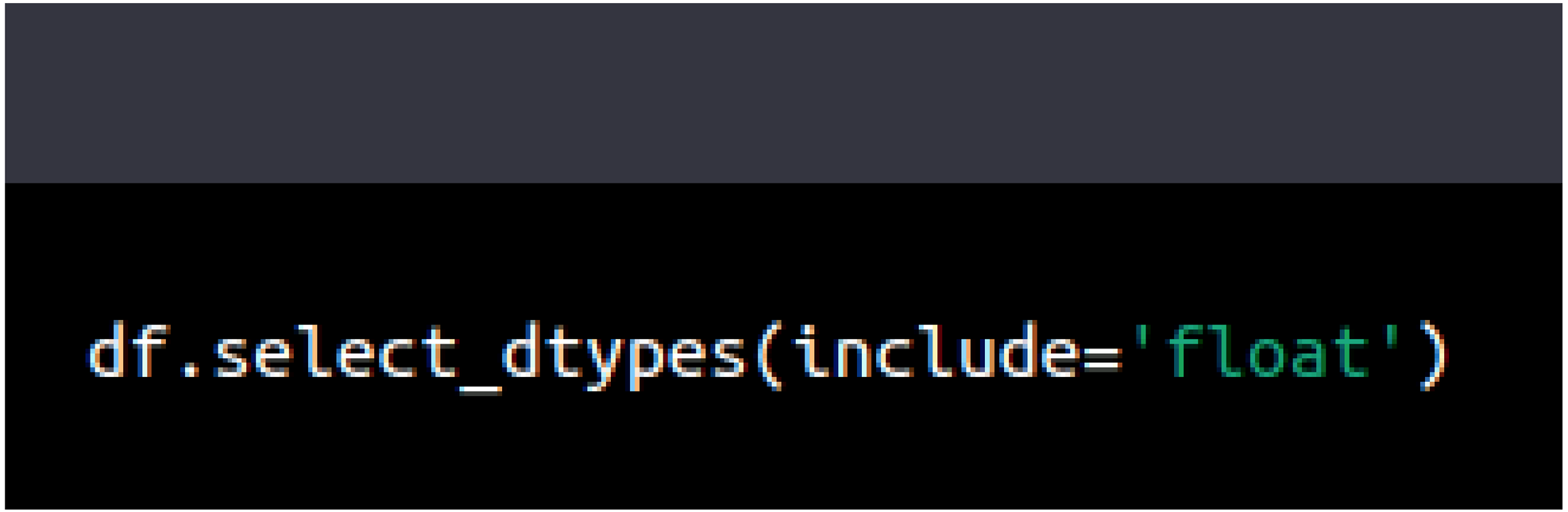
Use the .resample() function to resample time-series data at a different frequency. For example, to resample data at a monthly frequency and calculate the mean for each month, you can use the following code:



Use the .cut() function to bin values in a column into specific ranges. For example, to bin the 'age' column into ranges of 5 years, you can use the following code:

```
pd.cut(df['age'], bins=range(0,101,5))
```

Use the .select_dtypes() function to select columns of a specific data type. For example, to select all float columns in a dataframe, you can use the following code:



se the .fillna() function to fill missing values in a dataframe. For example, to fill missing values in the 'weight' column with the mean of the column, you can use the following code:

```
df['weight'].fillna(df['weight'].mean(), inplace=True)
```

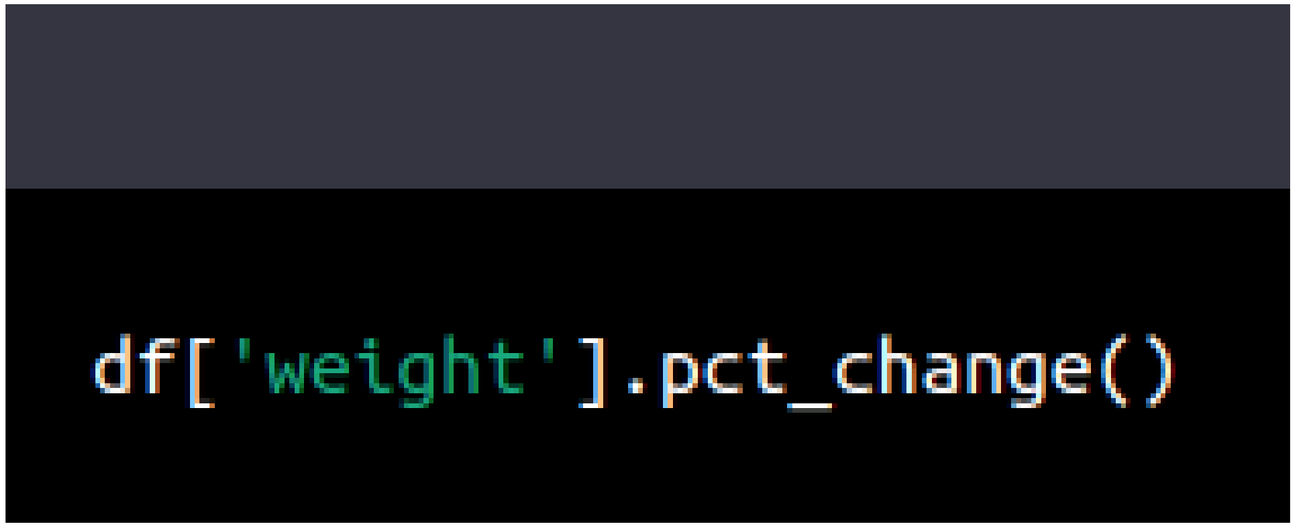
Use the .map() function to apply a function to each element of a series. For example, to convert all values in the 'species' column to lowercase, you can use the following code:

```
df['species'] = df['species'].map(lambda x: x.lower())
```

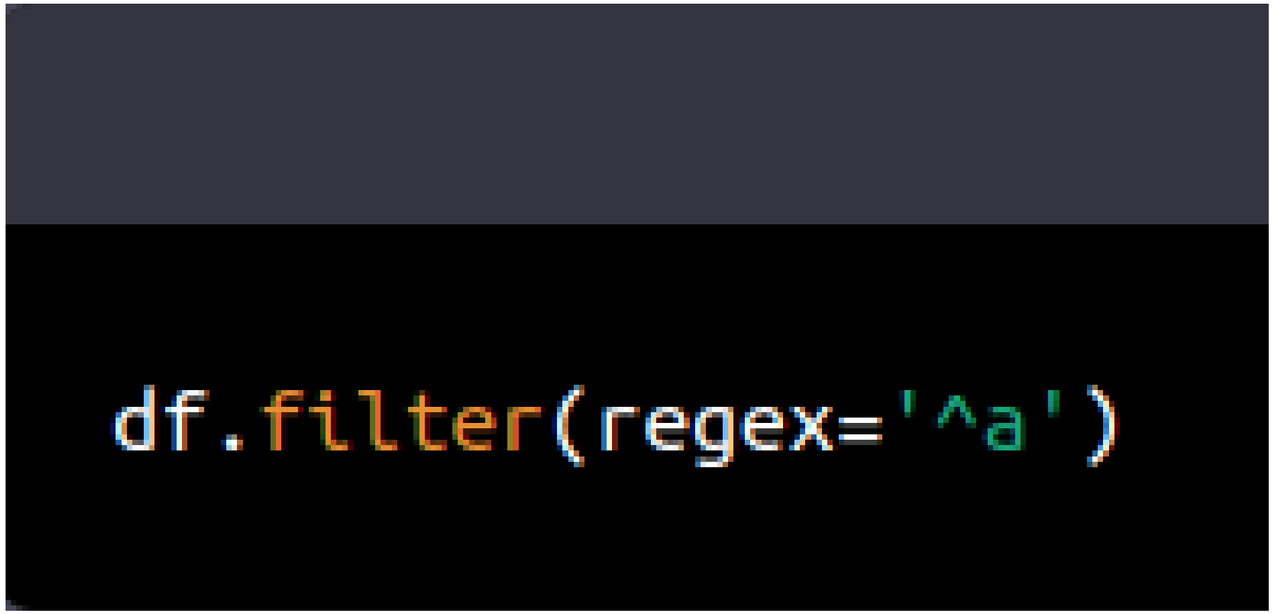
Use the .to_csv() function to export a dataframe to a csv file. For example, to export a dataframe to a csv file named 'pandas_data.csv', you can use the following code:

```
df.to_csv('pandas_data.csv', index=False)
```

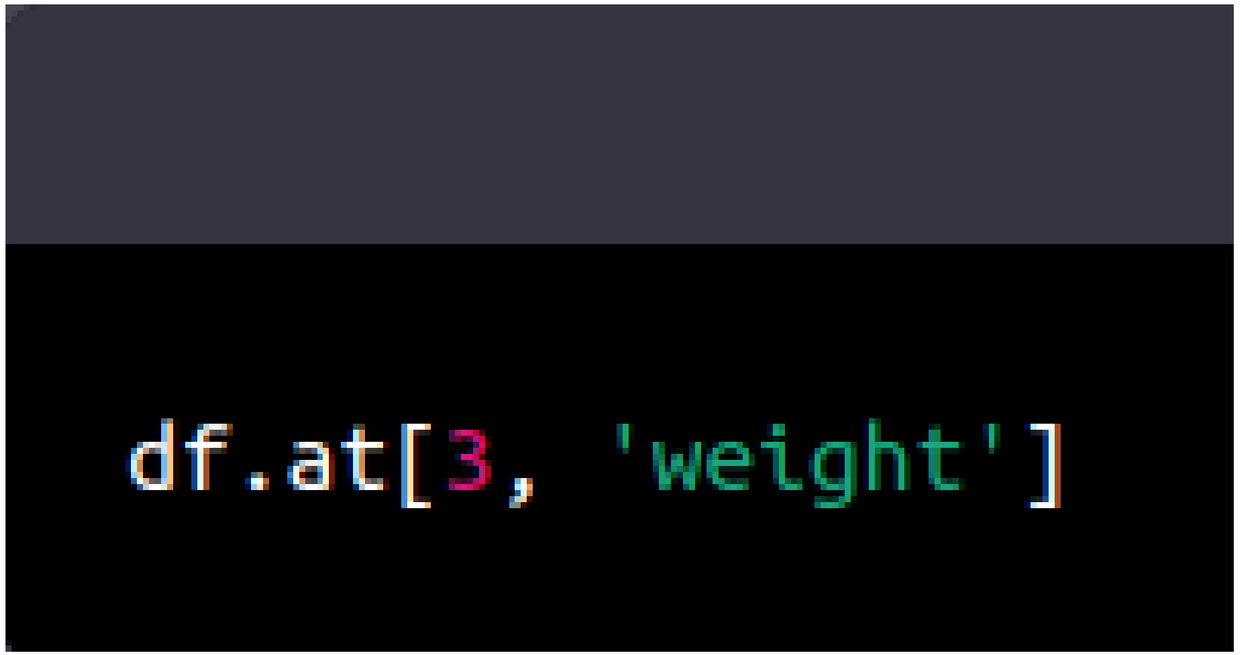
Use the .pct_change() function to calculate the percentage change between consecutive rows in a column. For example, to calculate the percentage change in the 'weight' column, you can use the following code:



Use the .filter() function to select columns from a dataframe based on a condition. For example, to select all columns where the column name starts with 'a', you can use the following code:



Use the .at() function to access a specific value in a dataframe based on its index and column label. For example, to access the value in the 'weight' column at index 3, you can use the following code:



Use the .where() function to filter rows based on a condition. For example, to select all rows where the 'species' column is equal to 'giant panda', you can use the following code:

```
df.where(df['species'] == 'giant panda')
```

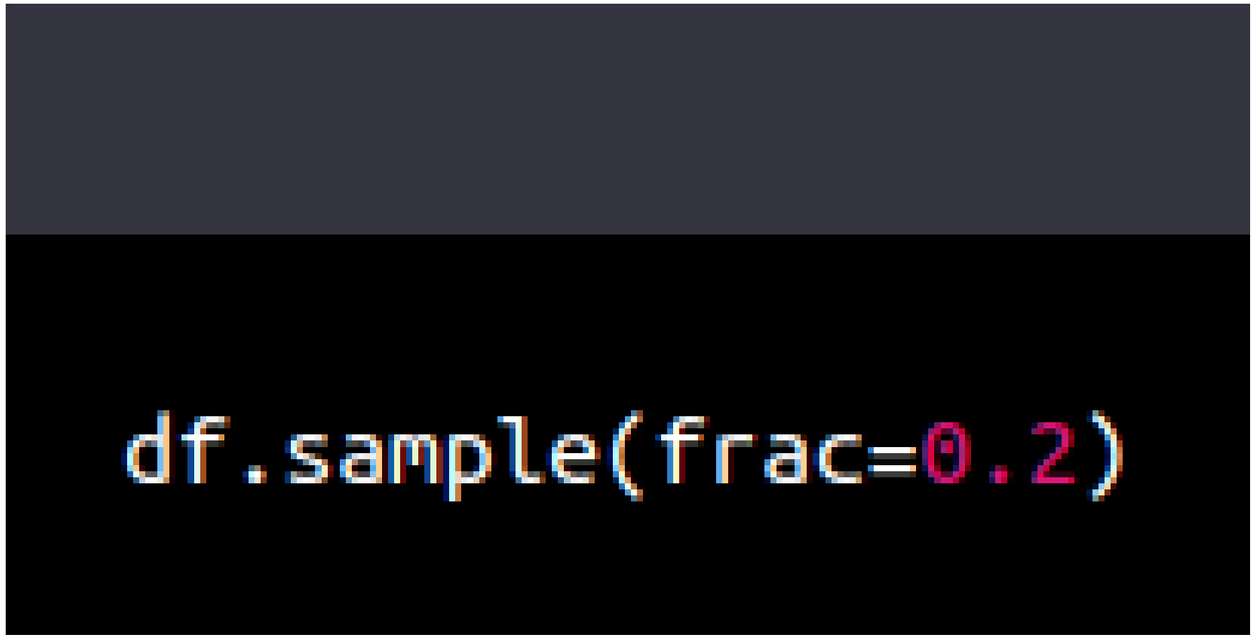
Use the .set_index() function to set a column as the index of a dataframe. For example, to set the 'name' column as the index, you can use the following code:

```
df.set_index('name', inplace=True)
```

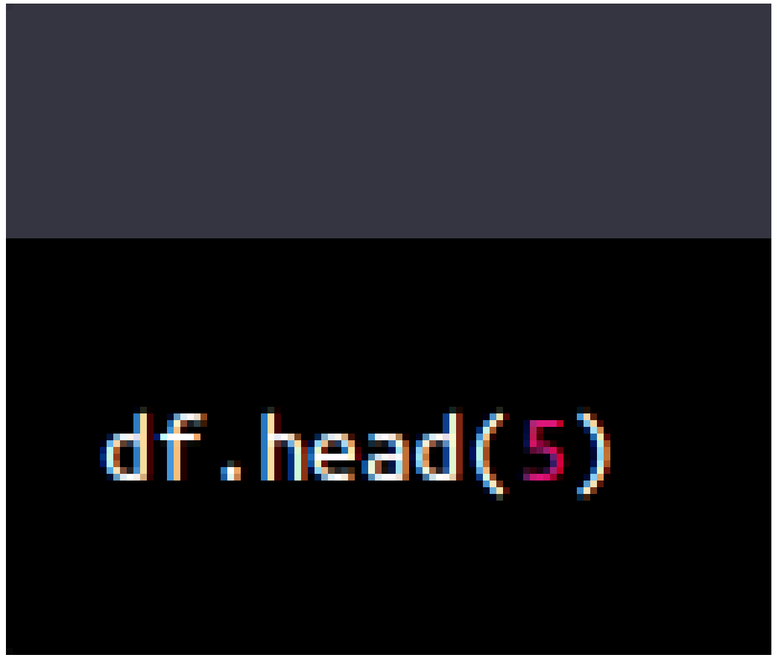
Use the .drop_duplicates() function to remove duplicate rows from a dataframe. For example, to remove duplicate rows based on the 'name' and 'species' columns, you can use the following code:

```
df.drop_duplicates(subset=['name', 'species'], inplace=True)
```

Use the .sample() function with the frac parameter to randomly sample a percentage of rows from a dataframe. For example, to randomly sample 20% of the rows from a dataframe, you can use the following code:



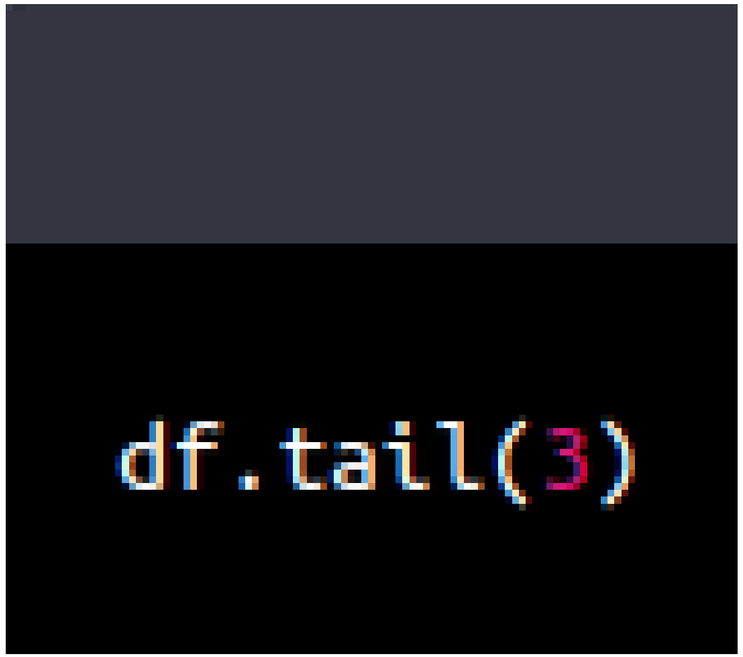
Use the .head() function to select the first n rows of a dataframe. For example, to select the first 5 rows of a dataframe, you can use the following code:



Use the .sort_values() function with the ascending parameter to sort a dataframe or series in descending order. For example, to sort the 'weight' column in descending order, you can use the following code:

```
df.sort_values('weight', ascending=False)
```

Use the .tail() function to select the last n rows of a dataframe. For example, to select the last 3 rows of a dataframe, you can use the following code:



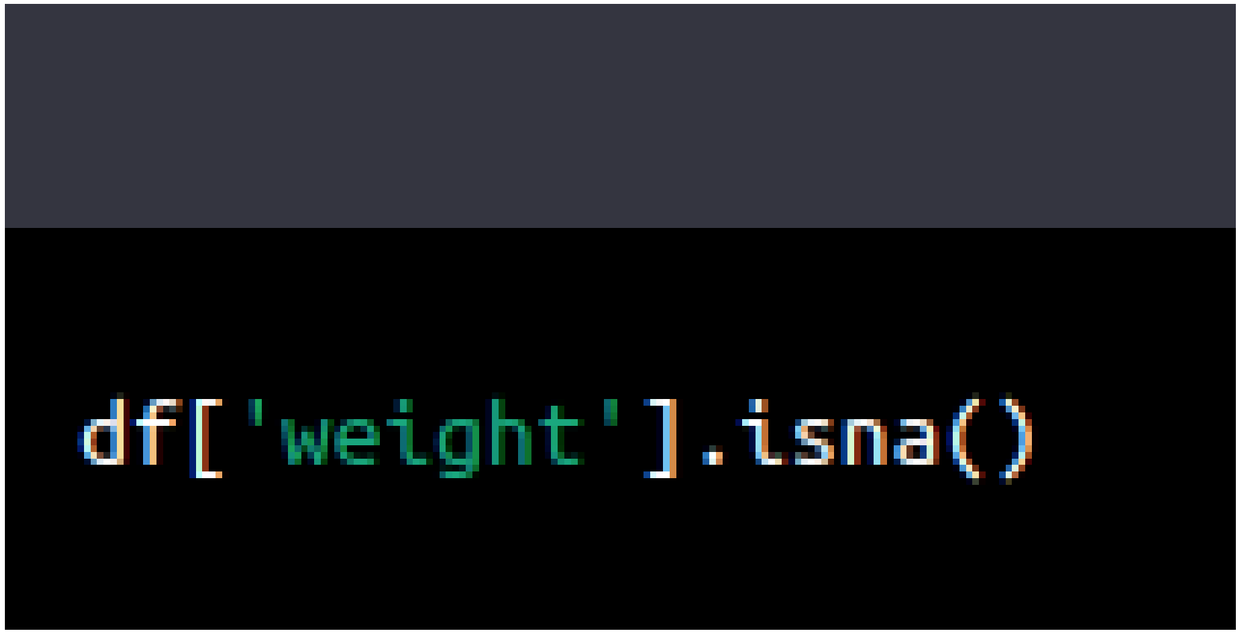
Use the .nsmallest() function to select the bottom n rows of a dataframe based on a specific column. For example, to select the bottom 3 rows based on the 'weight' column, you can use the following code:

```
df.nsmallest(3, 'weight')
```

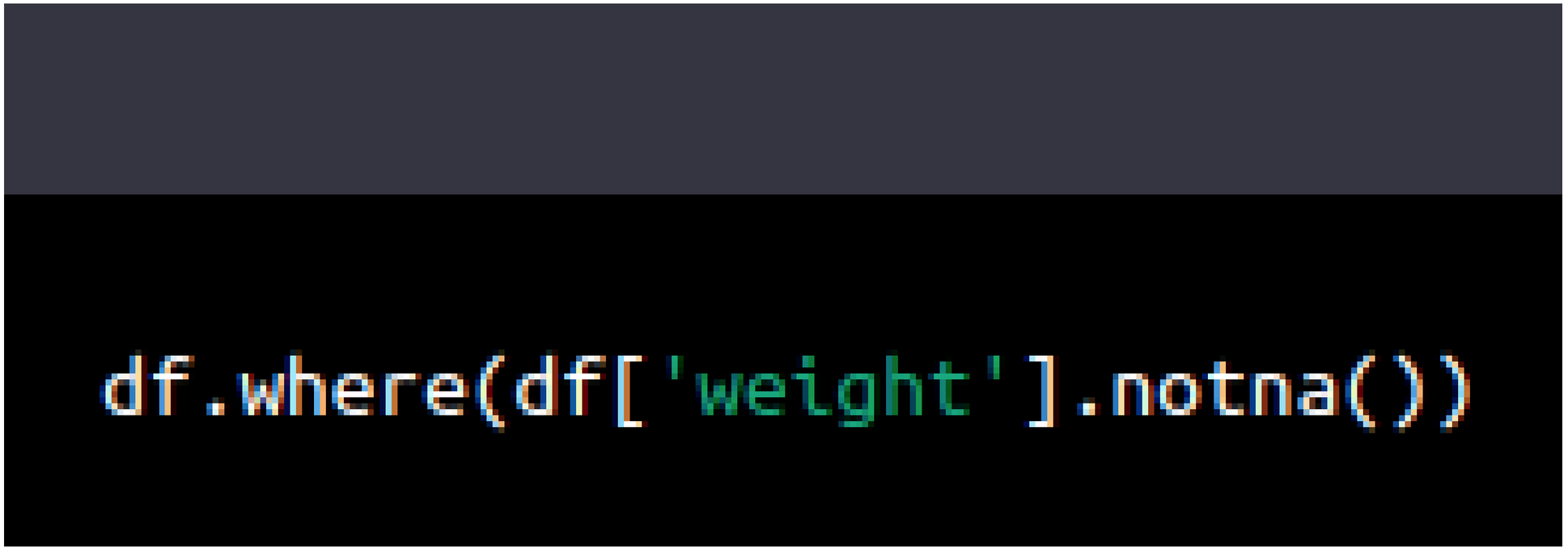
Use the .cov() function to calculate the covariance between columns in a dataframe. For example, to calculate the covariance between the 'weight' and 'height' columns, you can use the following code:

```
df[['weight', 'height']].cov()
```

Use the .isna() function to check for missing values in a dataframe or series and return a boolean mask. For example, to check for missing values in the 'weight' column, you can use the following code:



Use the .where() function along with the .notna() function to filter out rows with missing values. For example, to select all rows where the 'weight' column is not missing, you can use the following code:



Dask:

Dask is a parallel computing library that can handle large data sets and perform complex computations using pandas dataframes and series. It allows you to easily scale up your pandas code to handle large data sets by breaking them down into smaller chunks.

```
import dask.dataframe as dd


# Convert pandas dataframe to dask dataframe
df = dd.from_pandas(df, npartitions=8)


# Perform operations on dask dataframe
result = df.groupby('species').mean()


# Convert dask dataframe back to pandas dataframe
result = result.compute()
```

Vaex:

Vaex is a library that allows you to perform fast data exploration and visualization on large data sets by using lazy evaluation and column-based computation. It is built on top of pandas and Dask.

```
import vaex

df = vaex.from_pandas(df)

df_filtered = df[df.column_name > 5]
```

Modin:

Modin is a library that uses parallel processing to speed up pandas operations. It is designed to be a drop-in replacement for pandas and can be used with the same syntax as pandas.

```
import modin.pandas as mpd

df = mpd.read_csv('large_file.csv')
```

pandas-profiling:

pandas-profiling is a library that generates a detailed report of a pandas dataframe including information such as variable types, missing values, and basic statistics. It can be used to quickly get an overview of a large dataset and identify potential issues. For example, to use pandas-profiling to generate a report for a pandas dataframe, you can use the following code:

```
import pandas_profiling

pandas_profiling.ProfileReport(df)
```

Pyarrow:

Pyarrow is a library that allows for faster data serialization and deserialization of pandas dataframes. It can be used to speed up the process of reading and writing large datasets to disk. For example, to use Pyarrow to write a pandas dataframe to a parquet file, you can use the following code:

```
import pyarrow as pa
import pyarrow.parquet as pq
pq.write_table(pa.Table.from_pandas(df), "data.parquet")
```

These libraries can help speed up and optimize the performance of pandas dataframe operations, especially when working with large datasets.

Looking to master Linux, Python, SQL, Machine Learning, Deep Learning, and Statistics?

Send me a message to customize your learning journey

@arjun-panwar