Pandas Revision Notes

In this session, we covered several aspects of handling and manipulating data using Pandas DataFrame in Python. Let's go through the concepts discussed in detail:

Overview of Pandas

Pandas is a powerful library in Python for data manipulation and analysis. It provides data structures like Series and DataFrames to handle various data formats. In this session, we concentrated on DataFrames.

Working with Rows and Columns

Adding Rows

1. **Using `concat`** : To add a new row, you can use the `concat` method. First, you create a new dictionary where your DataFrame's columns are the keys, and the new row's values are the corresponding values. Convert this dictionary to a DataFrame and concatenate it to your existing DataFrame .

   ```
   import pandas as pd

   new_row = {'country': 'India', 'year': 2000, 'population': 135C
   df = pd.concat([df, pd.DataFrame([new_row])], ignore_index=True
   ```

   This approach uses `ignore_index=True` to maintain a proper sequence of indices in the DataFrame.

Dropping Rows

- **Using `drop` method**: To remove a row, use the `drop` method and specify the index of the row you wish to delete. The `axis=0` is used for rows .

  ```
  df = df.drop([0, 2], axis=0)
  ```

Accessing Data by Index

- **`.loc` and `.iloc`**: Access specific rows and columns using `.loc` (label-based) and `.iloc` (position-based). These methods behave similarly to list slicing. Remember, `.iloc` uses integer-location based indexing .

```
df.loc[0]  # Access row with label 0
df.iloc[0] # Access the first row
```

Handling Duplicates

Pandas provides convenient ways to identify and manage duplicate data in a DataFrame.

- **Finding Duplicates**: Use `DataFrame.duplicated()` to get a Series marking entries as `True` where duplicates are found .
- **Dropping Duplicates**: Use `DataFrame.drop_duplicates()` to remove duplicate rows. You can also specify a subset of columns to consider while determining duplicates .

```
df_unique = df.drop_duplicates()
```

Merging and Concatenating DataFrames

Concatenation

- **`pd.concat`**: This method appends DataFrames along a particular axis (rows or columns) .

Merging

- **`merge`**: Combines dataframes intelligently using common columns as keys, similar to SQL joins. Different types of joins can be specified such as inner, outer, left, or right .

```
merged_df = df1.merge(df2, on='key_column', how='outer')
```

Best Practices with Pandas

detailed explanations .

- **Efficient Usage**: Utilize methods like `.reset_index()` , `axis` , and `inplace` correctly to manage DataFrame indices and avoid common pitfalls in data duplication and indexing .

In summary, this session imparted foundational skills in manipulating data structures effectively with Pandas, equipping learners to perform data manipulation tasks with confidence. Understanding these fundamental operations lets learners build more complex data processing and analytics workflows.