Python Pandas Cheat sheet

DECODING DATA SCIENCE

This covers some of the most commonly used functions and operations in Pandas:

Importing and Exporting Data

Here is a quick Python Pandas cheatsheet that covers some of the most common functions and operations you will use when working with Pandas:

Importing Pandas

To use Pandas, you will first need to import the library:

```
import pandas as pd
```

Reading a CSV file

You can read a CSV file into a Pandas DataFrame using the read_csv function:

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

Displaying the DataFrame

To view the data in a DataFrame, you can use the head function to display the first few rows:

```
df.head()
```

You can also use the tail function to display the last few rows:

```
df.tail()
```

To display the entire DataFrame, you can simply print it:

```
print(df)
```

Selecting Columns

You can select a single column of a DataFrame by using the [] operator and the column name:

```
df['column_name']
```

You can also select multiple columns by passing a list of column names:

```
df[['column_1', 'column_2']]
```

Filtering Rows

You can filter the rows of a DataFrame using a boolean expression. For example, to select all rows where the value in the 'age' column is greater than 30:

```
df[df['age'] > 30]
```

Sorting Data

You can sort the rows of a DataFrame by one or more columns using the sort_values function. For example, to sort the DataFrame by the 'age' column in ascending order:

```
df.sort_values(by='age')
```

To sort in descending order, set the ascending parameter to False:

```
df.sort_values(by='age', ascending=False)
```

Grouping Data

You can group a DataFrame by one or more columns and apply a function to each group using the groupby function. For example, to group the DataFrame by the 'gender' column and compute the mean of each group:

```
df.groupby('gender').mean()
```

Joining DataFrames

You can join two DataFrames using the merge function. For example, to join two DataFrames on the 'user_id' column:

```
df1.merge(df2, on='user_id')
```

Pivot Tables

You can create a pivot table from a DataFrame using the pivot_table function. For example, to create a pivot table with the 'gender' column as the rows, the 'country' column as the columns, and the 'age' column as the values:

```
df.pivot_table(index='gender', columns='country', values='age')
```

Handling Missing Values

Pandas includes functions for handling missing values. To drop rows with missing values:

```
df.dropna()
```

To fill missing values with a specific value, you can use the fillna function:

```
df.fillna(value=0)
```

You can also fill missing values with the mean of the column using the fillna function and the mean function:

```
df.fillna(df.mean())
```

Converting Data Types

You can convert the data type of a column using the astype function. For example, to convert the 'age' column to a string:

```
df['age'] = df['age'].astype(str)
```

Applying Functions

You can apply a function to each element of a column using the apply function. For example, to apply the len function to the 'name' column:

```
df['name'].apply(len)
```

You can also apply a custom function by defining it and passing it to the apply function. For example:

```
def reverse_name(name):
  return name[::-1]
```

```
df['name'].apply(reverse_name)
```

Exporting Data

You can export a DataFrame to a CSV file using the to_csv function. For example:

```
df.to_csv('output.csv')
```

You can also export to other file formats, such as Excel, by using the to_excel function:

```
df.to_excel('output.xlsx', sheet_name='Sheet1')
```

Summary Statistics

You can compute summary statistics for a DataFrame using the describe function, which returns a new DataFrame with statistical information about the columns:

df.describe()

You can also compute specific summary statistics by using the corresponding function. For example, to compute the mean of the 'age' column:

df['age'].mean()

Other summary statistics functions include min, max, median, and mode.

Visualizing Data

You can use the plot function of a DataFrame to create various types of plots. For example, to create a line plot:

df.plot()

You can specify the type of plot using the kind parameter. For example, to create a bar plot:

df.plot(kind='bar')

You can also use the plot.bar function to create a bar plot:

```
df.plot.bar()
```

To customize the plot, you can use various parameters of the plot function. For example, to specify the x and y axis data and the title:

```
df.plot(x='column_1', y='column_2', title='Title')
```

Indexing and Selection

You can select rows and columns using the [] operator and indices or labels.

For example, to select a single row by its index:

```
df.loc[0]
```

To select a range of rows:

```
df.loc[0:2]
```

To select a single column:

```
df['column_name']
```

To select multiple columns:

df[['column_1', 'column_2']]

You can also use the iloc attribute to select rows and columns by integer position. For example, to select the first row:

df.iloc[0]

To select a range of rows:

df.iloc[0:2]

To select a single column:

df.iloc[:, 0]

To select multiple columns:

df.iloc[:, 0:2]

Adding and Removing Columns

You can add a new column to a DataFrame by assigning a list or array to a new column name. For example:

df['new_column'] = [1, 2, 3]

You can also use an existing column to create a new one. For example:

df['new_column'] = df['column_1'] + df['column_2']

To remove a column, you can use the drop function with the axis parameter set to 1:

df.drop('column_name', axis=1)

Adding and Removing Rows

You can add a new row to a DataFrame by using the append function and passing a Series or a dictionary:

df.append({'column_1': 1, 'column_2': 2}, ignore_index=True)

To remove a row, you can use the drop function with the index parameter:

df.drop(index=0)

Renaming Columns

You can rename the columns of a DataFrame using the rename function and the columns parameter. For example:

df.rename(columns={'old_name': 'new_name'})

You can also use the columns attribute to rename the columns in place:

df.columns = ['new_name_1', 'new_name_2']

Iterating Over a DataFrame

You can use a for loop to iterate over the rows of a DataFrame. For example:

```
for index, row in df.iterrows():
  print(row['column_1'], row['column_2'])
```

You can also use the apply function to apply a function to each row or column:

```
df.apply(lambda row: row['column_1'] + row['column_2'], axis=1)
```

Conditional Selection

To select rows based on a condition, you can use the loc attribute and a boolean expression. For example, to select rows where the value in the 'age' column is greater than 30:

```
df.loc[df['age'] > 30]
```

You can also use the where function to select rows based on a condition. For example:

df.where(df['age'] > 30)

To select columns based on a condition, you can use the select_dtypes function and pass the data type as an argument. For example, to select all columns with numerical data:

df.select_dtypes(include=['int', 'float'])

You can also use the select_dtypes function to exclude columns with a specific data type. For example, to exclude object columns:

df.select_dtypes(exclude=['object'])

Resetting the Index

You can reset the index of a DataFrame using the reset_index function. This will create a new column with the old index as its values and set the index to a default integer index starting from 0. For example:

df.reset_index()

You can also specify a name for the new index column using the index.name attribute:

df.reset_index().index.name = 'new_index_name'

Casting a Column to a Different Data Type

You can cast a column of a DataFrame to a different data type using the astype function. For example, to cast the 'age' column to an integer:

df['age'] = df['age'].astype(int)

You can also specify the data type using a string. For example:

df['age'] = df['age'].astype('int')

Duplicate Rows

To identify duplicate rows in a DataFrame, you can use the duplicated function. This will return a boolean Series indicating whether each row is a duplicate. For example:

df.duplicated()

You can then use this Series to select the duplicate rows:

```
df[df.duplicated()]
```

To drop the duplicate rows, you can use the drop_duplicates function:

```
df.drop_duplicates()
```

You can also specify which columns to consider when determining whether a row is a duplicate using the subset parameter:

```
df.drop_duplicates(subset=['column_1', 'column_2'])
```

Concatenating DataFrames

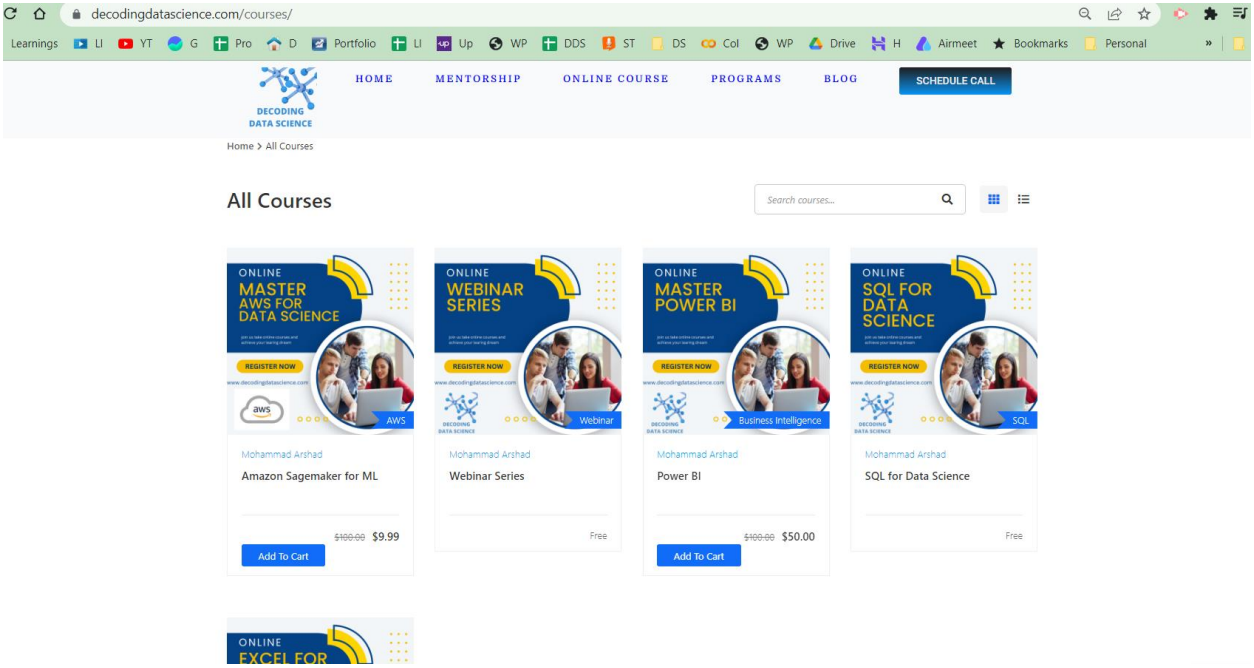
You can concatenate multiple DataFrames using the concat function. For example:

```
pd.concat([df1, df2, df3])
```

You can also specify the axis to concatenate along using the axis parameter. By default, the concat function concatenates along the rows (axis=0). To concatenate along the columns (axis=1), you can set the axis parameter to 1:

Subscribe to community at www.decodingdatascience.com to get more useful documents, ebooks, courses & job tips like this.

Online Courses Available : https://decodingdatascience.com/courses/



Subscribe to the Community:

https://decodingdatascience.com/community/