

Data Exploration using Pandas



CHEATSHEET

1. Reading and Writing Data

- Reading a CSV file**
`>>>df=pd.read_csv('AnalyticsVidhya.csv')`
- Writing content of data frame to CSV file**
`>>>df.to_csv('AV.csv')`
- Reading an Excel file**
`>>>df=pd.read_excel('AV.xlsx','sheet1')`
- Writing content of data frame to Excel file**
`>>>df.to_excel('AV2.xlsx',sheet_name='sheet2')`



2. Getting Preview of Dataframe

- Looking at top n records**
`>>>df.head(5)`
- Looking at bottom n records**
`>>>df.tail(5)`
- View columns name**
`>>>df.columns`

3. Rename Columns of Data Frame

- Rename method helps to rename column of data frame.**
`>>>df2=df.rename(columns={'old_columnname':'new_columnname'})`
This statement will create a new data frame with new column name.
- To rename the column of existing data frame, set inplace=True.**
`>>>df.rename(columns={'old_columnname':'new_columnname'}, inplace=True)`

4. Selecting Columns or Rows

- Accessing sub data frames**
`>>>df[['column1','column2']]`
- Filtering Records**
`>>>df[ df['column1']>10]`
`>>>df[ (df['column1']>10) & df['column2']==30]`
`>>>df[ (df['column1']>10) | df['column2']==30]`



5. Handling Missing Values

This is an inevitable part of dealing with data . To overcome this hurdle, use dropna or fillna function.

- dropna:** It is used to drop rows or columns having missing data
`>>>df1.dropna()`
- fillna:** It is used to fill missing values
`>>>df2.fillna(value=5)` #It replaces all missing values with 5
`>>>mean=df2['column1'].mean()`
`>>>df2['column1'].fillna(mean)` #It replaces all missing values of column1 with mean of available values

6. Creating New Columns

New column is a function of existing columns

```
>>>df['NewColumn1']=df['column2'] #Create a copy of existing column2
```

```
>>>df['NewColumn2']=df['column2']+10 #Add 10 to existing column2 then create a new one
```

```
>>>df['NewColumn3']= df['column1'] + df['column2'] #Add elements of column1 and column2 then create new column
```

7. Aggregate

- Groupby:** Groupby helps to perform three operations
 - Splitting the data into groups
 - Applying a function to each group individually
 - Combining the result into a data structure

```
>>>df.groupby('column1').sum()
>>>df.groupby(['column1','column2']).count()
```



- Pivot Table:** It helps to generate data structure. It has three components index, columns and values (similar to excel)

```
>>>pd.pivot_table(df, values='column1', index=['column2','column3'], columns=['column4'])
```

By default, it shows the sum of values column but you can change it using argument aggfunc

```
>>>pd.pivot_table(df, values='column1', index=['column2','column3'], columns=['column4'], aggfunc=len)
```

#it shows count

- Cross Tab:** Cross Tab computes the simple cross tabulation of two factors.

```
>>>pd.crosstab(df.column1, df.column2)
```

8. Merging/ Concatenating DataFrames

It performs similar operation like we do in SQL.

- Concatenating:** It concatenate two or more data frames based on their columns.

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

- Merging:** We can perform left, right and inner join also.

```
>>>pd.merge(df1, df2, on='column1', how='inner')
>>>pd.merge(df1, df2, on='column1', how='left')
>>>pd.merge(df1, df2, on='column1', how='right')
>>>pd.merge(df1, df2, on='column1', how='outer')
```

9. Applying function to element, column or dataframe

- Map:** It iterates over each element of a series.
`>>>df['column1'].map(lambda x: 10+x)` #this will add 10 to each element of column1
`>>>df['column2'].map(lambda x: 'AV'+x)` #this will concatenate "AV" at the beginning of each element of column2 (column format is string)
- Apply:** As the name suggests, applies a function along any axis of the DataFrame.
`>>>df[['column1','column2']].apply(sum)` #it will returns the sum of all the values of column1 and column2.
- ApplyMap:** This helps to apply a function to each element of dataframe.
`>>>func = lambda x: x+2`
`>>>df.applymap(func)` #it will add 2 to each element of dataframe (all columns of dataframe must be numeric type)

10. Identify unique values

Function unique helps to return unique values of a column.

```
>>>df['Column1'].unique()
```

11. Basic Stats

Pandas helps to understand the data using basic statistical methods.

- describe:** This returns the quick stats (count, mean, std, min, first quartile, median, third quartile, max) on suitable columns

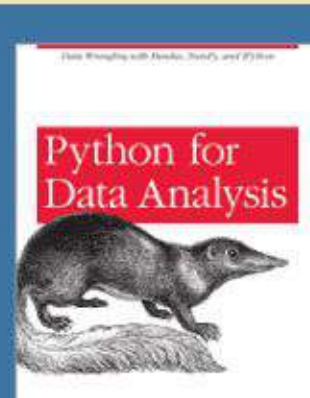
```
>>>df.describe()
```

- covariance:** It returns the co-variance between suitable columns.

```
>>>df.cov()
```

- correlation:** It returns the co-variance between suitable columns.

```
>>>df.corr()
```



To learn more, we recommend
Wes McKinney's **Python for Data Analysis**
Book for Learning Pandas