Pandas Cheat Sheet

Import convention

```
>>> import pandas as pd
```

Creating data

Creating Series

```
>>> s = pd.Series([3, -5, 7, 4],
      index=['a', 'b', 'c', 'd'])
```

Creating Dataframe

```
>>> df = pd.DataFrame(
  {"name" : ["Ram","Rahul","Ravi"],
   "age" : [51, 28, 19],
   "weight" : [69.3, 44.6, 36.9]})
```

Loading Data

```
>>> df = pd.read_csv('data.csv')
```

- Loading the data from a csv file into python.

Updating Rows/Columns

```
>>> df.rename(columns={'age':'Age'})
```

- Renames the column names

```
>>> df.loc[2, ['age', 'weight']] = [35, 89.1]
```

- Updates row values at given columns.

```
>>> df["age"].apply(lambda x: x + 5)
```

- Updates the column value as per the lambda function.

```
>>> df.apply(max)
```

- Applies the given function on dataframe.

```
>>> p_df = pd.DataFrame(
      {"name" : ["Jack Smith", 'Jane Lodge'],
      "place" : ["HYD", "DEL"]})
```

```
>>> p_df.applymap(str.lower)
```

- Applies the function to every element.

```
>>> df['name'].map({'Rahul':'Raghu'})
```

- Map values of the Series according to input correspondence.

```
>>> df.replace(to_replace=[51, 69.3], value = 58)
```

- Replaces values '51, 69.3' to 58 in the whole dataframe'.

```
>>> df['age'].replace({51:58})
```

- Replace value '51' in age column to '58'.

```
>>> p_df[['first', 'last']] =
p_df['name'].str.split(' ', expand=True)
```

- Splits columns.

```
>>> age_df = pd.DataFrame(
      {"age": [35, 17]})
>>> pd.concat([p_df, age_df], axis=1)
```

- Concatenates pandas objects along axis.

```
>>> p_df.drop(labels='last', axis='columns')
```

- Removes rows or columns by specifying label names and corresponding axis.

```
>>> p_df.append(
      {'name':'Jim lake',
      'first':'Jim',
      'last':'lake'}, ignore_index=True)
```

- Appends rows to p_df and returns a new object.

```
>>> df2 = pd.DataFrame(
      {"place" : ["HYD","DEL"],
      "state" : ["TEL", "UP"]})
>>> p_df.merge(df2, on="place")
```

- Merge two dataframes based on given column.

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

- Sort by the values of the given column.

```
>>> df['age'].nlargest(2)
```

- Orders first 2 rows based on given column in descending order.

```
>>> df['age'].nsmallest(2)
```

- Orders first 2 rows based on given column in ascending order.

Properties of Dataframe

`>>> df.head(n)`	First n rows
`>>> df.tail(n)`	Last n rows
`>>> df.shape`	Shape of df
`>>> df.columns`	Column labels
`>>> df.dtypes`	Datatypes of columns
`>>>df.describe`	Summary statistics

Accessing Data

`>>> df.loc[0]`	Row by label
`>>> df.loc[[0, 2], ['age', 'weight']]`	Group of rows and columns by label(s)
`>>> df.iloc[[0, 1]]`	Group of rows and columns by indices.
`>>>df.at[1,'weight']`	Single value for a row-column label pair.

Filtering Based on Criteria

`>>> df [df['age'] > 50]`	Extracts rows that meet logical criteria.
`>>> df.query('age < weight & age==11')`	DataFrame resulting from the provided query expression.
`>>> filter = df['name'].str.contains('Rah')` `>>> df[filter]`	Series resulting from the provided string query expression.

Display Options

```
>>> pd.set_option('display.max_rows', n)
```
- Sets the max visible rows for dataframe.

```
>>> pd.reset_option('display')
```
- Resets all the display options.

Changing the Index

```
>>> df.set_index('name')
```
- Set the index to become the ‘name’ column.

```
>>> df.reset_index()
```
- Reset the index of df and use the default one.

```
>>> df.sort_index(axis=0)
```
- Sort object by labels (along an axis).

```
>>> pd.read_csv('data.csv', index_col =
'column_name')
```
- Setting the index while reading the csv file.

Grouping and Aggregation

```
>>> df.groupby(by=['age', 'name'])
```
- Returns Groupby object grouped by values in given columns.

```
>>> df['name'].value_counts()
```
- Counts the number of times each value is repeated.

```
>>> df.groupby('name')['age'].mean()
```
- Splits into groups based on 'age' and aggregation done on 'name'.

```
>>> df.count()
```
- Counts non-NA cells for each column or row.

```
>>> df['age'].min()
```
- Returns minimum of the values.

```
>>> df.aggregate(['sum', 'min', 'mean'])
```
- Aggregates the data using the functions: 'sum', 'min', mean'.

```
>>> df['age'].cumsum()
```
- Returns the cumulative sum of a Series or DataFrame.

Cleaning Data

Handling Missing Values

```
>>> nan_df = pd.DataFrame({ "A" :[1.0, -3.0, 1.0],
                            "B" : [1.0, np.nan, 1.0],
                            "C" : [3.0, -2.0, 3.0],
                            "D" :[1.0, -3.0, 1.0])}
```

```
>>> nan_df.isna()
```
- Returns a boolean same-sized object indicating if the values are NA.

```
>>> nan_df.fillna(2)
```
- Fills NA/NaN values with the given value.

```
>>> nan_df.dropna()
```
- Returns a DataFrame with the NaN entries dropped from it.

```
>>> nan_df.replace('NA', np.nan, inplace=True)
```
- Handle other missing values by replacing with given value.

```
>>> df.first_valid_index()
```
- Index of the first non-NA/null value.

Handling Duplicates

```
>>> nan_df.duplicated()
```
- Returns a boolean series for each of the duplicated rows.

```
>>> nan_df.drop_duplicates()
```
- Returns a dataframe with the duplicated rows removed.

Changing Datatypes

```
>>>  df['weight'].astype('int64')
```
- Converts 'weight' column into integer.

```
>>> df.astype('string')
```
- Converts every element in the df to string.

```
>>> data = pd.DataFrame(
      {'year': [2015, 2016],
       'month': [2, 3],'day': [4, 5]})
```

```
>>> datetime_df = pd.to_datetime(data)
```
- Converts into datetime datatype.

```
>>>  datetime_df.dt.month
```
- Returns months in the timestamps.

```
>>>  datetime_df.dt.year
```
- Returns years in the timestamps.

```
>>>  datetime_df.dt.day_name()
```
- Returns weekday in the timestamps.