Python Cheat Sheet for Data Science & Machine Learning

Pandas, Numpy, and Scikit-Learn are among the most popular libraries for data science and analysis with Python.

Numpy is used for lower level scientific computation. Pandas is built on top of Numpy and designed for practical data analysis in Python. Scikit-Learn comes with many machine learning models that you can use out of the box.

In this cheat sheet, we'll summarize some of the most common and useful functionality from these libraries. Let's jump straight in!

Setting Working Directory

```
import os
cwd = os.getcwd()
cwd
os.chdir('C:\\Users\\\OneDrive\\Documents\\Data Science\\Datasets')
os.getcwd()
```

Importing Data

Any kind of data analysis starts with getting hold of some data. Pandas gives you plenty of options for getting data into your Python workbook:

```
import pandas as pd

df = pd.read_csv('SalaryData.csv')

df

df.shape
```

```
pd.read_csv(filename) # From a CSV file
pd.read_table(filename) # From a delimited text file (like TSV)
pd.read_excel(filename) # From an Excel file
pd.read_sql(query, connection_object) # Reads from a SQL table/database
pd.read_json(json_string) # Reads from a JSON formatted string, URL or file.
pd.read_html(url) # Parses an html URL, string or file and extracts tables to a list of dataframes
pd.read_clipboard() # Takes the contents of your clipboard and passes it to read_table()
pd.DataFrame(dict) # From a dict, keys for columns names, values for data as lists
```

Exploring Data

Once you have imported your data into a Pandas dataframe, you can use these methods to get a sense of what the data looks like:

```
df.shape() # Prints number of rows and columns in dataframe
df.head(n) # Prints first n rows of the DataFrame
df.tail(n) # Prints last n rows of the DataFrame
df.info() # Index, Datatype and Memory information
df.describe() # Summary statistics for numerical columns
s.value_counts(dropna=False) # Views unique values and counts
df.apply(pd.Series.value_counts) # Unique values and counts for all columns
df.describe() # Summary statistics for numerical columns
df.mean() # Returns the mean of all columns
df.corr() # Returns the correlation between columns in a DataFrame
df.count() # Returns the number of non-null values in each DataFrame column
df.max() # Returns the highest value in each column
df.min() # Returns the lowest value in each column
df.median() # Returns the median of each column
df.std() # Returns the standard deviation of each column
```

Selecting

Often, you might need to select a single element or a certain subset of the data to inspect it or perform further analysis. These methods will come in handy:

```
df[col] # Returns column with label col as Series
df[[col1, col2]] # Returns Columns as a new DataFrame
s.iloc[0] # Selection by position (selects first element)
s.loc[0] # Selection by index (selects element at index 0)
df.iloc[0,:] # First row
df.iloc[0,0] # First element of first column
```

Data Cleaning

If you’re working with real world data, chances are you’ll need to clean it up. These are some helpful methods:

```

df.columns = ['a','b','c'] # Renames columns
pd.isnull() # Checks for null Values, Returns Boolean Array
pd.notnull() # Opposite of s.isnull()
df.dropna() # Drops all rows that contain null values
df.dropna(axis=1) # Drops all columns that contain null values
df.dropna(axis=1,thresh=n) # Drops all rows have have less than n non null values
df.fillna(x) # Replaces all null values with x
s.fillna(s.mean()) # Replaces all null values with the mean (mean can be replaced with almost any function from the statistics
section)
s.astype(float) # Converts the datatype of the series to float
s.replace(1,'one') # Replaces all values equal to 1 with 'one'
s.replace([1,3],['one','three']) # Replaces all 1 with 'one' and 3 with 'three'
df.rename(columns=lambda x: x + 1) # Mass renaming of columns
df.rename(columns={'old_name': 'new_ name'}) # Selective renaming
df.set_index('column_one') # Changes the index
df.rename(index=lambda x: x + 1) # Mass renaming of index
```

Filter, Sort and Group By

Methods for filtering, sorting and grouping your data:

```
df[df[col] > 0.5] # Rows where the col column is greater than 0.5
df[(df[col] > 0.5) & (df[col] < 0.7)] # Rows where 0.5 < col < 0.7
df.sort_values(col1) # Sorts values by col1 in ascending order
df.sort_values(col2,ascending=False) # Sorts values by col2 in descending order
df.sort_values([col1,col2], ascending=[True,False]) # Sorts values by col1 in ascending order then col2 in descending order
df.groupby(col) # Returns a groupby object for values from one column
df.groupby([col1,col2]) # Returns a groupby object values from multiple columns
df.groupby(col1)[col2].mean() # Returns the mean of the values in col2, grouped by the values in col1 (mean can be replaced with
almost any function from the statistics section)
df.pivot_table(index=col1, values= col2,col3], aggfunc=mean) # Creates a pivot table that groups by col1 and calculates the mean
of col2 and col3
df.groupby(col1).agg(np.mean) # Finds the average across all columns for every unique column 1 group
df.apply(np.mean) # Applies a function across each column
df.apply(np.max, axis=1) # Applies a function across each row
```

Joining and Combining

Methods for combining two dataframes:

```
df1.append(df2) # Adds the rows in df1 to the end of df2 (columns should be identical)
pd.concat([df1, df2],axis=1) # Adds the columns in df1 to the end of df2 (rows should be identical)
df1.join(df2,on=col1,how='inner') # SQL-style joins the columns in df1 with the columns on df2 where the rows for col have
identical values. how can be one of 'left', 'right', 'outer', 'inner'<strong> </strong>
```

Writing Data

And finally, when you have produced results with your analysis, there are several ways you can export your data:

```
df.to_csv(filename) # Writes to a CSV file
df.to_excel(filename) # Writes to an Excel file
df.to_sql(table_name, connection_object) # Writes to a SQL table
df.to_json(filename) # Writes to a file in JSON format
df.to_html(filename) # Saves as an HTML table
df.to_clipboard() # Writes to the clipboard
```

Machine Learning

The Scikit-Learn library contains useful methods for training and applying machine learning models. For a complete list of the Supervised Learning, Unsupervised Learning, and Dataset Transformation, and Model Evaluation modules in Scikit-Learn, please refer to its user guide.

```
# Import libraries and modules
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.externals import joblib

# Load red wine data.
dataset_url = 'http://mlr.cs.umass.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'
data = pd.read_csv(dataset_url, sep=';')

# Split data into training and test sets
y = data.quality
X = data.drop('quality', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=123,
                                                    stratify=y)

# Declare data preprocessing steps
pipeline = make_pipeline(preprocessing.StandardScaler(),
                         RandomForestRegressor(n_estimators=100))

# Declare hyperparameters to tune
hyperparameters = { 'randomforestregressor__max_features' : ['auto', 'sqrt', 'log2'],
                  'randomforestregressor__max_depth': [None, 5, 3, 1]}

# Tune model using cross-validation pipeline
clf = GridSearchCV(pipeline, hyperparameters, cv=10)

clf.fit(X_train, y_train)

# Refit on the entire training set
# No additional code needed if clf.refit == True (default is True)

# Evaluate model pipeline on test data
pred = clf.predict(X_test)
print r2_score(y_test, pred)
print mean_squared_error(y_test, pred)

# Save model for future use
joblib.dump(clf, 'rf_regressor.pkl')
# To load: clf2 = joblib.load('rf_regressor.pkl')
```