PYTHON FOR DATA SCIENCE CHEAT SHEET

Python Basics

Datatypes

- Numbers: a=2(Integer), b=2.0(Float), c=1+2j(Complex)
- List: a=[1,2,3,'Word']
- Tuple: a= (1,2,4)
- String: a="New String"
- Sets: a= {2,3,4,5}
- Dictionary: x= {'a': [1,2],'b': [4,6]}

Operators

Numeric Operator: (Say, a holds 5, b holds 10)

- a + b = 15
- a – b = -5
- a * b = 50
- 7.0//2.0 = 3.0, -11//3 = -4
- b/a = 2
- b % a = 0
- a**b =9765625

Comparison Operator:

- (a == b): not true
- (a!= b): true
- (a > b): not true
- (a > b): not true
- (a >= b): not true
- (a <= b) is true

Boolean Operator:

- a and b
- a or b
- not a

Operations

List Operations

- List=[]: Defines an empty list
- list[i]=a: Stores a at the ith position
- list[i]: Retrieves the character at the ith position
- list[i:j]: Retrieves characters in the range i to j
- list.append(val): Adds item at the end
- list.pop([i]): Removes and returns item at index i

String Operations

- String[i]: Retrieves the character at the ith position
- String[i:j]: Retrieves characters in the range i to j

Dictionary Operations

- dict={} : Defines an empty dictionary
- dict[i]=a: stores "a" to the key "i"
- dict[i]: Retrieves the item with the key "i"
- dict.key: Gives all the key items
- dict.values: Gives all the values

OOPS

Inheritance:

A process of using details from a new class without modifying existing class.

Polymorphism:

A concept of using common operation in different ways for different data input.

Encapsulation:

Hiding the private details of a class from other objects.

Class/object

Class:

```
class Pen:
    pass
```

Object:

```
obj=Pen()
```

Flow Control Method

- **if-else (Conditional Statement)**

```
if price>=700:
        print("Buy.")
 else:
        print("Don't buy.")
```

- **For loop (Iterative Loop Statement)**

```
a="New Text"
count=0
for i in a:
    if i=='e':
        count=count+1
print(count)
```

- **While loop (Conditional Loop Statement)**

```
a=0
i=1
while i <10:
        a=a*2
        i=i+1
print(a)
```

- **Loop Control: Break, Pass and continue**

Functions

```
def new_function():
    print("Hello World")

new_function()
```

Lambda Function

lambda a,b: a+b

lambda a,b: a*b

Comments

 # Single Line Comment

"""

Multi-line comment

"""

Generic Operations

- range(5): 0,1,2,3,4
- S=input("Enter:")
- Len(a): Gives item count in a
- min(a): Gives minimum value in a
- max(a): Gives minimum value in a
- sum(a): Adds up items of an iterable and returns sum
- sorted(a): Sorted list copy of a
- importing modules: import random

File Operations

f= open("File Name","opening mode")

(Opening modes: r: read, w: write, a: append, r+: both read and write)

Try & Except Block

```
try:
        [Statement body block]
        raise Exception()
except Exception as e:
        [Error processing block]
```

IntelliPaat

FURTHERMORE:

Python for Data Science Certification Training Course

PYTHON FOR DATA SCIENCE CHEAT SHEET

Python NumPy

What is NumPy?

A library consisting of multidimensional array objects and a collection of routines for processing those arrays.

Why NumPy?

Mathematical and logical operations on arrays can be performed. Also provides high performance.

Import Convention

import numpy as np – Import numpy

ND Array

Space efficient multi-dimensional array, which provides vectorized arithmetic operations.

Creating Array

- a=np.array([1,2,3])
- b=np.array([(1,2,3,4),(7,8,9,10)],dtype=int)

Initial Placeholders

- **np.zeros(3)** - 1D array of length 3 all zeros

```
array([0., 0., 0.])
```

- **np.zeros((2,3))** - 2D array of all zeros

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

- **np.zeros((3,2,4))** - 3D array of all zeros

```
array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.]]])
```

- **np.full((3,4),2)** - 3x4 array with all values 2
- **np.random.rand(3,5)** - 3x5 array of random floats between 0-1
- **np.ones((3,4))** - 3x4 array with all values 1
- **np.eye(4)** - 4x4 array of 0 with 1 on diagonal

Saving and Loading

On disk:

- np.save("new_array",x)
- np.load("new_array.npy")

Text/CSV files:

- np.loadtxt('New_file.txt') - From a text file
- np.genfromtxt('New_file.csv',delimiter=',') - **From a CSV file**
- np.savetxt('New_file.txt',arr,delimiter=' ') - **Writes to a text file**
- np.savetxt('New_file.csv',arr,delimiter=',') - **Writes to a CSV file**

Properties:

- array.size - **Returns number of elements in array**
- array.shape - **Returns dimensions of array(rows, columns)**
- array.dtype - **Returns type of elements in array**

Operations

Copying:

- **np.copy(array)** - Copies array to new memory array.
- **view(dtype)** - Creates view of array elements with type dtype

Sorting:

- **array.sort()** - Sorts array
- **array.sort(axis=0)** - Sorts specific axis of array
- **array.reshape(2,3)** - Reshapes array to 2 rows, 3 columns without changing data.

Adding:

- **np.append(array,values)** - Appends values to end of array
- **np.insert(array,4,values)** - Inserts values into array before index 4

Removing:

- **np.delete(array,2,axis=0)** - Deletes row on index 2 of array
- **np.delete(array,3,axis=1)** - Deletes column on index 3 of array

Combining:

- **np.concatenate((array1,array2),axis=0)** - Adds array2 as rows to the end of array1
- **np.concatenate((array1,array2),axis=1)** - Adds array2 as columns to end of array1

Splitting:

- **np.split(array,3)** - Splits array into 3 sub-arrays

Indexing:

- **a[0]=5** - Assigns array element on index 0 the value 5
- **a[2,3]=1** - Assigns array element on index [2][3] the value 1

Subseting:

- **a[2]** - Returns the element of index 2 in array a.
- **a[3,5]** - Returns the 2D array element on index [3][5]

Slicing:

- **a[0:4]** - Returns the elements at indices 0,1,2,3
- **a[0:4,3]** - Returns the elements on rows 0,1,2,3 at column 3
- **a[:2]** - Returns the elements at indices 0,1
- **a[:,1]** - Returns the elements at index 1 on all rows

Array Mathematics

Arithmetic Operations:

- **Addition:** np.add(a,b)
- **Subtraction:** np.subtract(a,b)
- **Multiplication:** np.multiply(a,b)
- **Division:** np.divide(a,b)
- **Exponentiation:** np.exp(a)
- **Square Root:** np.sqrt(b)

Comparison:

- **Element-wise:** a==b
- **Array-wise:** np.array_equal(a,b)

Functions

- **Array-wise Sum**: a.sum()
- **Array-wise min value:** a.min()
- **Array row max value:** a.max(axis=0)
- **Mean:** a.mean()
- **Median:** a.median()

PYTHON FOR DATA SCIENCE CHEAT SHEET

Python Pandas

What is Pandas?

It is a library that provides easy to use data structure and data analysis tool for Python Programming Language.

Import Convention

import pandas as pd – Import pasdas

Pandas Data Structure

- **Series:**

  s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])

- **Data Frame:**

  data_mobile = {'Mobile': ['iPhone', 'Samsung', 'Redmi'], 'Color': ['Red', 'White', 'Black'], 'Price': [High, Medium,Low]}

  df = pd.DataFrame(data_mobile, columns=['Mobile', 'Color', 'Price'])

Importing Data

- pd.read_csv(filename)
- pd.read_table(filename)
- pd.read_excel(filename)
- pd.read_sql(query, connection_object)
- pd.read_json(json_string)

Exporting Data

- df.to_csv(filename)
- df.to_excel(filename)
- df.to_sql(table_name, connection_object)
- df.to_json(filename)

Create Test/Fake Data

- pd.DataFrame(np.random.rand(4,3)) - 3 columns and 4 rows of random floats
- pd.Series(new_series) - Creates a series from an iterable new_series

Plotting

- **Histogram:** df.plot.hist()
- **Scatter Plot:** df.plot.scatter(x='column1',y='column2')

Operations

View DataFrame Contents:

- df.head(n) - look at first n rows of the DataFrame.
- df.tail(n) – look at last n rows of the DataFrame.
- df.shape() - Gives the number of rows and columns.
- df.info() - Information of Index, Datatype and Memory.
- df.describe() -Summary statistics for numerical columns.

Selection:

- iloc
  - df.iloc[0] - Select first row of data frame
  - df.iloc[1] - Select second row of data frame
  - df.iloc[-1] - Select last row of data frame
  - df.iloc[:,0] - Select first column of data frame
  - df.iloc[:,1] - Select second column of data frame
- loc
  - df.loc([0], [column labels])- Select single value by row position & column labels
  - df.loc['row1':'row3', 'column1':'column3']- Select and slicing on labels

Sort:

- df.sort_index() - Sorts by labels along an axis
- df.sort_values(by='Column label') - Sorts by the values along an axis
- df.sort_values(column1) - Sorts values by column1 in ascending order
- df.sort_values(column2,ascending=False) - Sorts values by column2 in descending order

Operations - GroupBy

from one column

- df.groupby([column1,column2]) - Returns a groupby object values from multiple columns
- df.groupby(column1)[column2].mean() - Returns the mean of the values in column2, grouped by the values in column1
- df.groupby(column1)[column2].median() - Returns the mean of the values in column2, grouped by the values in column1

Functions

Mean:

- df.mean() - mean of all columns

Median

- df.median() - median of each column

Standard Deviation

- df.std() - standard deviation of each column

Max

- df.max() - highest value in each column

Min

- df.min() - lowest value in each column

Count

- df.count() - number of non-null values in each DataFrame column

Describe

- df.describe() - Summary statistics for numerical columns

FURTHERMORE:
Python for Data Science Certification Training Course

IntelliPaat

PYTHON FOR DATA SCIENCE CHEAT SHEET

Python Scikit-Learn

Introduction

Scikit-learn:"sklearn" is a machine learning library for the Python programming language. Simple and efficient tool for data mining, Data analysis and Machine Learning.

Importing Convention - import sklearn

Preprocessing

Data Loading

- **Using NumPy:**

```
>>>import numpy as np
>>>a=np.array([(1,2,3,4),(7,8,9,10)],dtype=int)
>>>data = np.loadtxt('file_name.csv',
   delimiter=',')
```

- **Using Pandas:**

```
>>>import pandas as pd
>>>df=pd.read_csv('file_name.csv',header=0)
```

Train-Test Data

```
>>>from sklearn.model_selection
import train_test_split

>>> X_train, X_test, y_train, y_test =
train_test_split(X,y,random_state=0)
```

Data Preparation

- **Standardization**

```
>>>from sklearn.preprocessing import
StandardScaler
>>>get_names = df.columns
>>>scaler =
preprocessing.StandardScaler()
>>>scaled_df = scaler.fit_transform(df)
>>>scaled_df =
pd.DataFrame(scaled_df,
columns=get_names)m
```

- **Normalization**

```
>>>from sklearn.preprocessing import
   Normalizer
>>>pd.read_csv("File_name.csv")
 >>>x_array = np.array(df['Column1'])
 #Normalize  Column1
 >>>normalized_X =
preprocessing.normalize([x_array])
```

Working On Model

Model Choosing

Supervised Learning Estimator:

- **Linear Regression:**

```
>>> from sklearn.linear_model import
LinearRegression
>>> new_lr =
LinearRegression(normalize=True)
```

- **Support Vector Machine:**

```
>>> from sklearn.svm import SVC
>>> new_svc = SVC(kernel='linear')
```

- **Naive Bayes:**

```
>>> from sklearn.naive_bayes import
GaussianNB
>>> new_gnb = GaussianNB()
```

- **KNN:**

```
>>> from sklearn import neighbors
>>>
knn=neighbors.KNeighborsClassifier(n_ne
ighbors=1)
```

Unsupervised Learning Estimator:

- **Principal Component Analysis (PCA):**

```
>>> from sklearn.decomposition import
PCA
>>> new_pca= PCA(n_components=0.95)
```

- **K Means:**

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=5,
random_state=0)
```

Train-Test Data

Supervised:

```
>>>new_ lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>>new_svc.fit(X_train, y_train)
```

Unsupervised :

```
>>> k_means.fit(X_train)
>>> pca_model_fit =
new_pca.fit_transform(X_train)
```

Post-Processing

Prediction

Supervised:

```
>>> y_predict =
new_svc.predict(np.random.random((3,5)))
>>> y_predict = new_lr.predict(X_test)
>>> y_predict = knn.predict_proba(X_test)
```

Unsupervised:

```
>>> y_pred = k_means.predict(X_test)
```

Model Tuning

Grid Search:

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3), "metric":
          ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization:

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5), "weights":
["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
param_distributions=params, cv=4, n_iter=8, random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Evaluate Performance

Classification:

1. Confusion Matrix:

```
>>> from sklearn.metrics import
      confusion_matrix
>>> print(confusion_matrix(y_test,
      y_pred))
```

2. Accuracy Score:

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import
      accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Regression:

1. Mean Absolute Error:

```
>>> from sklearn.metrics import mean_absolute_error

>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_predict)
```

2. Mean Squared Error:

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_predict)
```

3. R^2 Score :

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_predict)
```

Clustering:

1. Homogeneity:

```
>>> from sklearn.metrics import
homogeneity_score
>>> homogeneity_score(y_true,
y_predict)
```

2. V-measure:

```
>>> from sklearn.metrics import
v_measure_score
>>> metrics.v_measure_score(y_true,
y_predict)
```

Cross-validation:

```
>>> from
sklearn.cross_validation
import cross_val_score
>>>
print(cross_val_score(knn,
X_train, y_train, cv=4))
>>>
print(cross_val_score(new_
lr, X, y, cv=2))
```

FURTHERMORE:

Python for Data Science Certification Training Course