DSBA Codebook

greatlearning
Power Ahead

Preface

Data Science is the art and science of solving real world problems and making data driven decisions. It involves an amalgamation of three aspects and a good data scientist has expertise in all three of them. These are:

1) Mathematical/ Statistical understanding
2) Coding/ Technology understanding
3) Domain knowledge

Your lack of expertise should not become an impediment in your journey in Data Science. With consistent effort, you can become fairly proficient in coding skills over a period of time. This Codebook is intended to help you become comfortable with the finer nuances of Python and can be used as a handy reference for anything related to data science codes throughout the program journey and beyond that.

In this document we have followed the following syntax:

- Brief description of the topic
- Followed with a code example.

Please keep in mind there is no one right way to write a code to achieve an intended outcome. There can be multiple ways of doing things in Python. The examples presented in this document use just one of the approaches to perform the analysis. Please explore by yourself different ways to perform the same thing.

Contents

Table of Figures

Table of Equations

Python Basics

Arithmetic Operations

Addition

```
a=4
b=7
a+b
```

```
11
```

Subtraction

```
a-b
```

```
-3
```

Multiplication

```
a*b
```

```
28
```

Division

```
a/b
```

```
0.5714285714285714
```

Square

```
a**2
```

```
16
```

Square Root

```
a*0.5
```

```
2.0
```

Loops

For Loop

When the number of iterations required is known i.e. n, the ‘for’ is used.

```
n=10 #n is the variable with assigned value 10
for i in range(0,n): #for every value in range 0 to n
    print('Hello World')    #print
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

While Loop

When the number of iterations required is unknown i.e. n, the 'while' is used.

```
i=0 # value of variable i is 0
while i<10: #till i is smaller than 10
    print('Hello World') # print
    i=i+1 # increase value by 1 after every print
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Conditional Statements

IF Statement

This statements check the condition provided and if the condition is True, then the program moves ahead with defined steps

```
x=300
if x>200:
    print('Hey, x is greater than 200!')
```

```
Hey, x is greater than 200!
```

IF-Else Statement

This statement is an extension to IF statement. The program moves to Else statement and performs the alternative steps defined in case 'IF' condition is not met.

```
x=100
if x>200:
    print('Hey, x is greater than 200!')
else: print('Hey, x is smaller than 200!')
```

```
Hey, x is smaller than 200!
```

IF-ELIF Statement

This statement is an extension to IF-Else statement. The program moves to ELIF statement and performs the alternative steps defined in case 'IF' condition is not met and if the else if 'ELIF' in python is not met, then the program moves to another ELIF or Else statement.

Syntax along with example In this example else would be executed only when x=y

```
x=100
y=150
if x>y:
    print('Hey, x is greater than y!')
elif x<y:
    print('Hey, y is greater than x!')
else: print('Hey, x is equal to y!')
```

```
Hey, y is greater than x!
```

User Defined Functions

User-defined functions are very helpful in automating repetitive tasks like selecting odd numbers out of a series or converting a series of dates to timestamps

```
#A function is defined using 'def' followed by function name and arguments the function takes in
def addition(x,y):
    return (x+y) #in a function, return is used most preferably
#this is a function which returns addition of two variables passed into it.

#calling a function
a=1
b=2
addition(a,b)
3
```

Importing Libraries/Modules

A module is a file containing Python definitions and statements. Use ‘import’ and aliasing statement ‘as’

Example:

```
import pandas as pd
```

Here, we are importing pandas module with an alias ‘pd’.

Python Error Debugging

Syntax Error

When the syntax used is wrong. In below snapshot, the print statement is missing parenthesis.

```
print "hello"
executed in 5ms, finished 13:58:38 2020-05-06

  File "<ipython-input-133-2a0eaa89f43f>", line 1
    print "hello"
                ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello")?
```

Figure 1: Syntax Error

Missing one or more parenthesis like in below snapshot, the ‘)’ is missing.

```
print('hello'
executed in 5ms, finished 14:52:47 2020-05-06

  File "<ipython-input-148-0b37b907169d>", line 1
    print('hello'
                 ^
SyntaxError: unexpected EOF while parsing
```

Figure 2: Syntax Error- EOF while parsing

Index Error

When the Index Position is out of bounds. In below snapshot, the lst[8] looks for index position 8 which is not present in the given list lst.

```
lst=[2,4,6,7]
lst[8]
executed in 16ms, finished 14:00:08 2020-05-06

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-134-53cb2e8822ae> in <module>
      1 lst=[2,4,6,7]
----> 2 lst[8]

IndexError: list index out of range
```

Figure 3: Index Error

Module Not Found Error

When the module is not installed in your python environment. In below snapshot, the module ‘geopandas’ is not installed

```
import geopandas
executed in 15ms, finished 14:05:18 2020-05-06

---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-136-fc7d1d298f0c> in <module>
----> 1 import geopandas

ModuleNotFoundError: No module named 'geopandas'
```

Figure 4: Module Not Found Error

Import Error

When the specified module function is not found. In below snapshot, ‘feature_importances_’ is not found in module sklearn.tree.

```
from sklearn.tree import feature_importances_
```
executed in 11ms, finished 14:13:11 2020-05-06

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-141-36c5048d86b3> in <module>
----> 1 from sklearn.tree import feature_importances_

ImportError: cannot import name 'feature_importances_' from 'sklearn.tree' (C:\Users\IT\anaconda3\lib\site-packages\sklearn\tre
e\__init__.py)
```

Figure 5: Import Error

Key Error

When the dictionary's key is not found in the given dictionary. In below snapshot, key 'd' does not exist for the given dictionary.

```
some_dictionary= {'a': 'one','b': 'two', 'c':'three'}

some_dictionary['d']
```
executed in 54ms, finished 14:08:37 2020-05-06

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-137-4b75c014a098> in <module>
      1 some_dictionary= {'a': 'one','b': 'two', 'c':'three'}
      2
----> 3 some_dictionary['d']

KeyError: 'd'
```

Figure 6: Key Error

Value Error

When an inappropriate value is passed into a function. In below snapshot, 'hello' is inappropriately passed in for typecasting to float.

```
float('hello')
```
executed in 14ms, finished 14:21:05 2020-05-06

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-142-879c4bea9d62> in <module>
----> 1 float('hello')

ValueError: could not convert string to float: 'hello'
```

Figure 7: Value Error

Type Error

When an unsupported operation is performed like below snapshot the subtraction of '10' with integer 10.

```
'10' - 10
```
executed in 15ms, finished 14:37:13 2020-05-06

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-144-2d41cd77c43a> in <module>
----> 1 '10' - 10

TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Figure 8: Type Error

Name Error

When an undefined variable/object is used ike in below snapshot.

```
my_variable
```

executed in 13ms, finished 14:46:03 2020-05-06

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-145-75dc0442333b> in <module>
----> 1 my_variable

NameError: name 'my_variable' is not defined
```

Figure 9: Name Error

Indentation Error

When there is an incorrect indentation like in below example of 'for' loop

```
for i in range(0,2):
print(i)
```

executed in 5ms, finished 14:49:41 2020-05-06

```
  File "<ipython-input-146-bd1b2a5b77c1>", line 2
    print(i)
        ^
IndentationError: expected an indented block
```

Figure 10: Indentation Error

Zero Division Error

When there is division by 0 for which you cannot find an answer

```
for i in range(0,2):
    print(i/0)
```

executed in 19ms, finished 14:56:08 2020-05-06

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-153-72d9f9a18197> in <module>
      1 for i in range(0,2):
----> 2     print(i/0)

ZeroDivisionError: division by zero
```

Figure 11: Zero Division Error

Numpy – Numerical Python

Import Numerical Python Package (numpy) in Python with alias np

```
import numpy as np
```

NumPy brings the computational power of languages like C and Fortran to Python, a language much easier to learn and use.

To install numpy

```
pip install numpy
```

Array

```
# simple array

np.array([1,2,3,4])
array([1, 2, 3, 4])
```

Array Operations

```
a= np.array([1,2,3,4])

b = np.array([2,4,6,8])
```

Array Multiplication

```
# array multiplication
a*b
```

```
array([ 2,  8, 18, 32])
```

Array Square

```
np.square(a) # Square
```

```
array([ 1,  4,  9, 16], dtype=int32)
```

Taking Array's Absolute

```
np.abs([-1,-2,-3,-4]) # absolute
```

```
array([1, 2, 3, 4])
```

Array Square Root

```
np.sqrt(a) #square root
```

```
array([1.        , 1.41421356, 1.73205081, 2.        ])
```

Array Mean

```
np.mean(a) # mean
```

```
2.5
```

Array Median

```
np.median(a) # median
```

```
2.5
```

Array Standard Deviation

```
np.std(a) #standard Deviation
```

```
1.118033988749895
```

Array Correlation Coefficient

```
np.corrcoef(a,b)
```

```
array([[1., 1.],
       [1., 1.]])
```

Array Variance

```
np.var(a)
```

```
1.25
```

Array Covariance

```
np.cov(a,b)
```

```
array([[1.66666667, 3.33333333],
       [3.33333333, 6.66666667]])
```

Array Sum

```
np.sum(a)
```
```
10
```

Array Addition

```
a+b
```
```
array([ 3,  6,  9, 12])
```

Searching Array using np.where()

```
np.where(a < 2, 0,a*10) #if smaller than 2 then 0 else multiply by 10
```
```
array([ 0, 20, 30, 40])
```

Pandas

Import Pandas Package (pandas) in Python with alias pd

```
import pandas as pd
```

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

To install pandas:

```
pip install pandas
```

DataFrame

```
a=range(1,20) #creates a range of numbers between the 1 and 20(excluded)
df=pd.DataFrame(a,columns=['A']) #creates a dataframe with column name A
```

DataFrame Head

```
df.head() #gives top 5 rows
```

	A
0	1
1	2
2	3
3	4
4	5

```
df.head(6)  #gives top 6 rows
```

	A
0	1
1	2
2	3
3	4
4	5
5	6

DataFrame Tail

```
df.tail(3)# gives last three rows
```

	A
16	17
17	18
18	19

DataFrame Mean

```
df.mean() #gives mean column wise
```

```
A    10.0
dtype: float64
```

DataFrame Standard Deviation

```
df.std()
```

```
A    5.627314
dtype: float64
```

DataFrame Median

```
df.median()
```

```
A    10.0
dtype: float64
```

DataFrame Summary Statistics

```
df.describe() #gives summary statistics
```

	A
count	19.000000
mean	10.000000
std	5.627314
min	1.000000
25%	5.500000
50%	10.000000
75%	14.500000
max	19.000000

DataFrame Information

```
df.info() # gives information about the data on columns, rows, data type etc
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19 entries, 0 to 18
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       19 non-null     int64
dtypes: int64(1)
memory usage: 280.0 bytes
```

Add a Column to a DataFrame

```
df['B'] = range(20,39) # Adds a column to the df
df.head(3)
```

	A	B
0	1	20
1	2	21
2	3	22

DataFrame Transpose

```
df.describe().T #using .T transposes the dataframe (rows to columns and vice versa)
```

	count	mean	std	min	25%	50%	75%	max
A	19.0	10.0	5.627314	1.0	5.5	10.0	14.5	19.0
B	19.0	29.0	5.627314	20.0	24.5	29.0	33.5	38.0

Null Values Check

```
df.isnull().sum() #gives you column wise null values
```

```
A    0
B    0
dtype: int64
```

```
df.isnull().sum().sum() #gives you total null values
```

```
0
```

Applying a Function to a DataFrame

```
def func(x):# define a function
    if x>28:
        return 'Good'
    else: return 'Bad'
df['C']= df[['B']].applymap(func) #apply the function
```

```
df.groupby('C').sum() # groups the data ,
```

C	A	B
Bad	45	216
Good	145	335

Sorting Values in a DataFrame

```
df.sort_values('B',ascending=False) #ascending = False sorts by Max to Min
```

	A	B	C
18	19	38	Good
17	18	37	Good
16	17	36	Good
15	16	35	Good
14	15	34	Good
13	14	33	Good
12	13	32	Good
11	12	31	Good
10	11	30	Good
9	10	29	Good
8	9	28	Bad
7	8	27	Bad
6	7	26	Bad
5	6	25	Bad
4	5	24	Bad
3	4	23	Bad
2	3	22	Bad
1	2	21	Bad
0	1	20	Bad

Dropping Columns from a DataFrame

```
df.drop('A',axis=1, inplace=True) #drops the column from the original dataframe if inplace = True
```

Pre-processing

Null Value Check

To check Null values, use below:

```
df.isnull().sum()
```

To impute null values, use Simple Imputer: Imputation transformer for completing missing values. For numeric data, if there are no outliers, use ‘median’ and for categorical data, use ‘most_frequent’

Below is an example of imputation of null values in an “object” type column, hence in SimpleImputer strategy used in “most frequent”.

```
objects=df[cols].select_dtypes(include='object').columns
non_objects=df[cols].select_dtypes(exclude='object').columns
```

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent',verbose=0)
imputer= imputer.fit(df[objects])
df[objects]=imputer.transform(df[objects])
```

You do this manually as well

Outlier Check

Use this custom function to check and treat outliers. This is also known as “95% - 5%” capping technique. Based on the count of outliers, decision could be made to use a different capping percentage like 99%-1%.

```
def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
lower_range,upper_range=remove_outlier(df[column])

df[column]=np.where(df[column]>upper_range,upper_range,df[column])
df[column]=np.where(df[column]<lower_range,lower_range,df[column])
```

Train Test Split

Splitting arrays or matrices into random train and test subsets. Model will be fitted on train set and predictions will be made on the test set.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33,random_state=42)
```

Scaling

Standard Scaler

It is a scaling technique that scales down the data with mean equal to zero and standard deviation equal to 1.

mean~ 0 and standard deviation ~ 1

```
from sklearn.preprocessing import StandardScaler
ss= StandardScaler()
#for training data, use fit_transform and for test data use transform
x_train_scaled= ss.fit_transform(x_train)
x_test_scaled=ss.transform(x_test)
```

Min-Max Scaler

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

Source: scikit-learn

```
from sklearn.preprocessing import MinMaxScaler
mm= MinMaxScaler()
#for training data, use fit_transform and for test data use transform
x_train_scaled= mm.fit_transform(x_train)
x_test_scaled=mm.transform(x_test)
```

Statistics

Packages Required: Usually the packages required for this module are numpy, pandas, statsmodels, scipy.stats.

Descriptive Statistics

Descriptive Statistics is a collective term for the summary statistics of a data set. It is comprised of mean, median, mode, standard deviation, Inter Quartile Range (IQR) etc. and is studied through tables, graphs and charts.

Population refers to the entire set of observations of a data set. **Sample** refers to a subset of the population on which most studies about the population are based. Sample is drawn randomly to make inferences about the population parameters.

Measures of Central Tendency (Mean, Median, Mode)

Mean is the sum of all observations divided by number of observations.

```
import numpy as np
a = 100,56,29,90,102,134,809
np.mean(a)
```
```
188.57142857142858
```

Alternatively, print(df.mean()) would give you the mean of all columns of the data 'df'

Please note that you would need to import numpy package for this function.

Median represents the middle value.

Syntax with example:

```
import numpy as np
a = 100,56,29,90,102,134,809
np.median(a)
Output: 100.0
```

Alternatively, print(df.median()) would give you the medin of all columns of a data set 'df'.

Mode represents the most frequently occurring value.

print(df.mode()) gives mode of each column of data frame df. If no value appears more than once it displays NaN as output.

Please note that you can generate the five-point summary which will display most of the measures at one place. It can be generated using df.describe() or df.describe().T

Measures of Dispersion

Measures of Dispersion (Spread) are statistics that describe how data varies. Measure of dispersion gives us the sense of how much the data tends to diverge from the central tendency.

Range: It shows the spread of the values contained in the variable. It is the difference between the maximum and minimum values.

```
Range = df.max() - df.min() #df is the name of the data frame
```

Interquartile Range (IQR)

IQR gives a much better idea about the spread of the data because it doesn't take into account the effect of outliers. IQR is more popular than Range.

```
import numpy as np
Q1,Q3=np.percentile(col,[25,75])      #col is the column name
IQR=Q3-Q1
IQR
```

Variance and Standard deviation

Variance tells us the average degree to which the data is deviating from the mean. Standard deviation is the square root of the variance.

```
df['column'].var()     # Variance #column is the column name
df['column'].std()     # Standard Deviation #column is the column name
```

Correlation

Correlation measures how strongly two variables are related to each other.

```
df.corr()
```

This gives the correlation table showing all variables against each other.

Correlation Plot

```
import seaborn as sns
sns.heatmap(corr, annot=True)
```

This presents a pictorial representation of the correlation data frame and is much easier to make sense of, for smaller data frames.

For bigger data frames with a lot of variables, a correlation table would be preferable.

Skewness

Skewness shows the asymmetry in the data. It shows where most of the data points lie.

```
import pandas as pd
import scipy.stats as stats
Skewness = pd.DataFrame({'Skewness' : [stats.skew(df.col1),stats.skew(df.col2),stats.skew(df.col3)]},  index=[col1,'col2','col3'])
```

To check the Skewness of Col1, Col2, Col3 of the data frame 'df'

Skewness (where is the df … either we should mention that these values mentioned below are representative)

```
Col1    0.283729
Col2    0.055610
Col3    1.514180
```

Visualisations

Histogram:

```
df.hist()                                # Histogram of all continuous variables of the data frame 'df'
df['column'].hist()      # Histogram of a particular column of the data frame 'df'
```

Pairplot:

It is a powerful plot which is used to know the distributions and correlations of all variables of the data frame 'df'.

```
import seaborn as sns
sns.pairplot(df)
plt.show()
```

Boxplot:

```
df.boxplot(figsize=(15,4))
```

Probability Distributions

It is a statistical method that describes all the possible likelihoods for a random variable within a given range.

Please note that loc=0 and scale =1 are default values in the codes below.

Normal Distribution

```
stats.norm.ppf(q= 0.7, loc = 0, scale= 1)  (#q= random variable)
#For cumulative,
stats.norm.cdf(k= 1, loc = 0, scale= 1)   (#k=random variable)
```

Binomial Distribution

```
import scipy.stats as stats

n,p = 10,0.22
k=0
stats.binom.pmf(k,n,p)
0.083357758312362 #output
```

Please note that, n = Number of trials,

p = Probability of success

k = random variable

For Cumulative:

```
stats.binom.cdf(k,n,p)
```

If k=0, then the outcome would be same as the above output.

Poisson Distribution

```
import scipy.stats as stats
stats.poisson.pmf(k,lambda)        #k = random variable
```

For Cumulative:

```
stats.poisson.cdf(k,lambda)
```

Inferential Statistics

Inferential Statistics has a key, i.e., inference. So, in Inferential statistics, we draw out a sample from the population, put the sample through various tests to make inferences about the population.

Hypothesis Testing

Five step process to be followed:

Step 1: Formulate Null and Alternative Hypothesis

It can be mentioned as a comment or in a markdown. Having it in the sheet itself would be helpful.

The Null Hypothesis is often denoted by H_0 or H_{Null} and the Alternate Hypothesis is denoted by H_1or H_A

Example: $H_0 \leq population\ mean\ \mu$; $H_1 > population\ mean\ \mu$

Step 2: Finalise significance level

Decide a significance level (alpha) and state it in the sheet itself in comments or markdown.

(Generally significance level is 5%, however it can be increased or decreased as per the situation at hand)

Step 3: Identify the test to be undertaken and find the critical value

Decide on which test is to be used, for example, t-test, z-test, etc.

Step 4: Calculate the test statistic and p-value

Compute these values and compare it with alpha.

**** Please note that if Alpha is 5% (0.05), then for one tailed test the value of Alpha should be 0.05, but for two tailed test, it should be 0.025)**

Step 5: Decide whether to accept or reject the null hypothesis.

Take a decision to accept or reject the H0 based on the Critical Value, p-value approach and reach conclusions.

Golden Rule: If p-value is low (means p-value is lower than Alpha), then Null Hypothesis should be rejected (reject H0). Accept Alternative Hypothesis.

If p-value is more than Alpha, then we fail to reject Null Hypothesis (Reject Alternate Hypothesis)

Formulae/Codes for Step 3:

Checking Critical value:

z-Distribution

```
import scipy.stats as stats
cv = stats.norm.ppf(Alpha, 0, 1)  #loc=0, scale=1
```

t-Distribution

```
import scipy.stats as stats
cv=stats.t.ppf(0.05, df) #df is the degree of freedom n-1
```

f- Distribution

```
import scipy.stats as stats
stats.f.ppf(0.95,dfn=4,dfd=26)  #q=0.95
2.7425941372218587
```

Chi Square

Sample Code:

```
import scipy.stats as stats
stats.chi2.ppf(0.95, df=7)   #q = 0.95
14.067140449340169
```

Formulae/Codes for Step 4:

Z-test

When Standard Deviation of the population is known,

$$Z_{statistic} = \frac{(\bar{X} - \mu)}{(\sigma/\sqrt{n})}$$

Equation 1: Z-Statistic

Where $\bar{X}$ is sample mean, μ is population mean, σ is population standard deviation, and n is sample size.

Sample Code:

```
z = ((Xbar-Mu)/(SD/np.sqrt(n)))    # manual calculation
#Xbar = Sample Mean, Mu = Population Mean, SD= Population Standard Deviation
```

For array, the following code can be used to calculate the z score

```
scipy.stats.zscore(a, axis=0, ddof=0, nan_policy='propagate')
```

t-test (one sample)

When population standard deviation is not known,

$$t_{statistic} = \frac{(\bar{X} - \mu)}{(s/\sqrt{n})}$$

Equation 2: T-Statistic

Where $\bar{X}$ is sample mean, μ is population mean, s is sample standard deviation, and n is sample size.

t-test (two sample)

$$t_{statistic} = \frac{(X_1 - X_2)}{\sqrt{\frac{(s_1^2)}{n_1} + \frac{(s_2^2)}{n_2}}}$$

Equation 3: T-Statistic for 2 samples

f-test

$$F_{Statistic} = \frac{sum\ of\ squares\ between\ groups\ /(n-1)}{sum\ of\ squares\ within\ groups/(n-k)} = \frac{SSB/(n-1)}{SSW/(n-k)} = \frac{MSB}{MSW}$$

Equation 4: F-Statistic