> Variable Assignment

>>> x=5

>>> x

5

Calculations With Variables

>>> x + 2 *#Sum of two variables*

**7**

>>> x - 2 *#Subtraction of two variables*

**3**

>>> x \* 2 *#Multiplication of two variables*

**10**

>>> x \*\* 2 *#Exponentiation of a variable*

**25**

>>> x % 2 *#Remainder of a variable*

**1**

>>> x / float(2) *#Division of a variable*

**2.5**

> Types and Type Conversion

str()

'5', '3.45', 'True' *#Variables to strings “ ” or ‘ ‘*

int()

5, 3, 1 *#Variables to integers*

float()

5.0, 1.0 *#Variables to floats*

bool()

True, True, True *#Variables to booleans Variable*

> Libraries

(pandas) Data analysis (matplotlib ) 2D plotting

(NumPy) Scientific computing (scikit) Machine learning

(datetime) a date and should be represented as a datetime object

**Import Libraries** **Selective import**

>>> import matplotlib.pyplot as plt >>> from math import pi

>>> import numpy as np >>> from datetime import datetime

>>>Import pixiedust >>> from sklearn.cluster import KMeans

>>> from sklearn.cluster import

AgglomerativeClustering

> Strings

>>> my\_string = 'thisStringIsAwesome'

>>> my\_string

**'thisStringIsAwesome'**

String Operations

>>> my\_string \* 2

**'thisStringIsAwesomethisStringIsAwesome'**

>>> my\_string + 'Innit'

**'thisStringIsAwesomeInnit'**

>>>'m' in my\_string

**True**

String indexing

>>> my\_string[3] *# Indexing starts from* ***0*** *and ends at* ***N-1*** *or -****1***

>>> my\_string[4:9] *# This is called* ***Slicing*** *the index from Range \_\_\_ to \_\_\_*

String Methods

>>> my\_string.upper() *#String to uppercase*

>>> my\_string.lower() *#String to lowercase*

>>> my\_string.count('w') *#Count String elements*

>>> my\_string.replace('e' , 'i' ) *#Replace String elements*

>>> my\_string.strip() *#Strip whitespaces*

> NumPy Arrays *#import Numpy as np*

>>> my\_list = [ 1, 2, 3, 4 ] *# Normal List*

>>> my\_array = np.array( my\_list ) *# 1D array List*

>>> my\_2darray = np.array( [[1,2,3] , [4,5,6]] ) *# 2D array List*

Selecting Numpy Array Elements

*# Index starts at 0*

**Subset**

>>> my\_array[ 1 ] *# Select item at index 1*

**2**

**Slice**

>>> my\_array[ 0 : 2 ]  *# Select items at index 0 and 1*

**array([1, 2])**

**Subset 2D Numpy arrays**

>>> my\_2darray[ **:** , 0 ] *# my\_2darray[rows, columns]*

**array([1, 4])** *#* ***:*** *the Symbol means Select\Slice from the*

*start to end*

Numpy Array Operations

>>> my\_array > 3

**array( [ False, False, False, True ] , dtype=bool)**

>>> my\_array \* 2

**array( [ 2, 4, 6, 8 ] )**

>>> my\_array + np.array( [ 5, 6, 7, 8 ] )

**array( [ 6, 8, 10, 12 ] )**

Numpy Array Functions

>>> my\_array.shape *# Get the dimensions of the array*

>>> np.append(other\_array) *# Append items or other arrays to an array*

>>> np.insert(my\_array, 1, 5) *# Insert items in an array*

>>> np.delete(my\_array,[1]) *# Delete items in an array*

>>> np.mean(my\_array)  *# Mean of the array*

>>> np.median(my\_array)  *# Median of the array*

>>> my\_array.corrcoef()  *# Correlation coefficient*

>>> np.std(my\_array) *# Standard deviation*

> Lists

>>> a = ‘is’

>>> b = ‘nice’

>>> my\_list = [ ‘my’, ‘list’, a, b]

>>> my\_list2 = [[4,5,6,7], [3,4,5,6]]

Selecting List Elements

**Subset**

>>> my\_list[1]  *#Select item at index 1*

**‘my’**

>>> my\_list[-3] *#Select 3rd last item*

**‘list’**

**Slice**

>>> my\_list[1:3] *#Select items at index 1 and 2 but not 3*

>>> my\_list[1:] *#Select items after index 0, 1 included*

>>> my\_list[:3] *#Select items before index 3, 3 is not included*

>>> my\_list[:] *#Copy my\_list {Shallow copy\Diff. Adress}*

**Subset Lists of Lists**

>>> my\_list2[1][0] *#my\_list2[list number] [item Of List]*

**4**

>>> my\_list2[1][:2] *#my\_list2[list NO 1] [From 0\start : index 2]*

**[4,5]**

List Operations

>>> my\_list + my\_list

**[ ‘my’ , ‘list’ , ‘is’ , ‘nice’ , ‘my’ , ‘list’ , ‘is’ , ‘nice’ ]**

>>> my\_list \* 2

**[ ‘my’ , ‘list’ , ‘is’ , ‘nice’ , ‘my’ , ‘list’ , ‘is’ , ‘nice’ ]**

>>> my\_list2 > 4

**True**

List Methods

>>> my\_list.index(a) *#Get the index of an item (a)*

>>> my\_list.count(a) *#Count an item appearance (a)*

>>> my\_list.append( ‘X’ ) *#Append an item at a time ( ‘X’* )

>>> my\_list.remove( ‘X’ ) *#Remove an item ( ‘X’ )*

>>> del(my\_list[0:1]) *#Remove items via Slicing*

>>> my\_list.reverse() *#Reverse the list order*

>>> my\_list.extend( ‘X’ ) *#Append an item at the ending of list ( ‘X’ )*

>>> my\_list.pop(-1) *#Remove an item via indexing or -1 default*

>>> my\_list.insert(0, ‘X’ ) *#Insert an item after the said index (NO, ‘X’ )*

>>> my\_list.sort() *#Sort the list*

Import Files to Jupyter as pandas dataframe

>>>Variable\_name = pd.read\_csv ("File Name ON PC.csv" , sep = ' | ' )

*#note: we can define a url instead of a local file then input the defined url into the*

>>>pd.read\_csv(“url”)

*# sep = ‘ | ’ makes more space between the columns instead of the default, change it to ‘ , ’ if it was separated by comas instead of columns.*

*# read csv assume there is a header in line 1, so if there isn’t a header define the header as none as shown in the following*

>>>Variable\_name = pd.read\_csv ("File Name ON PC.csv" , header = None )

>>> Variable\_name.head(*# the num. of rows shown from top*)

>>> Variable\_name.tail(*# the num. of rows shown from bottom*).

>>>Variable\_name = pd.read\_excel ("File Name ON PC.excel")

Import headers to pandas dataframe

1. Define the headers of the columns in a list

>>> headers = [“…”, “..”, …]

1. Then insert the headers into the dataframe

>>> Variable\_name.columns = headers.

Exporting pandas dataframe to CSV file

1. Define a path to you local machine

>>> path = “c:\desktop\.....\ Variable\_name.csv

1. Type following command

>>> Variable\_name.to\_csv(path)

Timeline

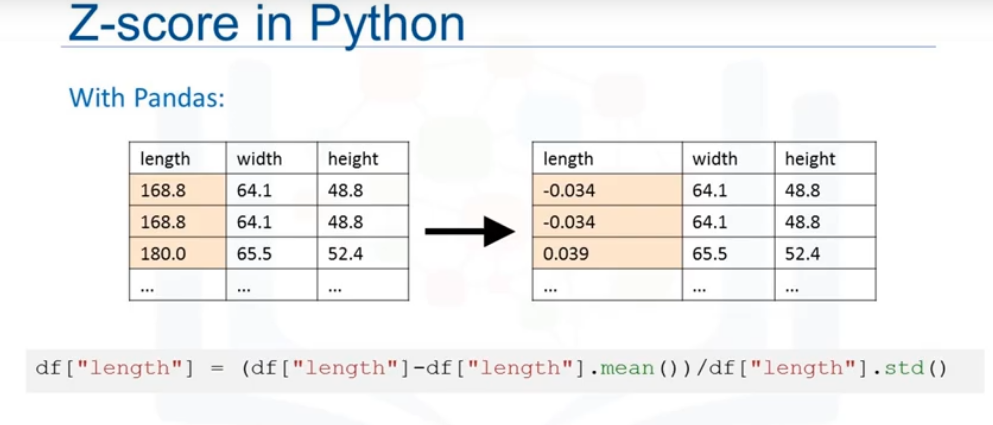
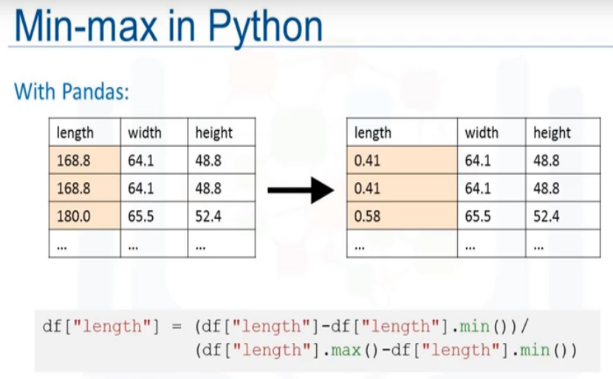
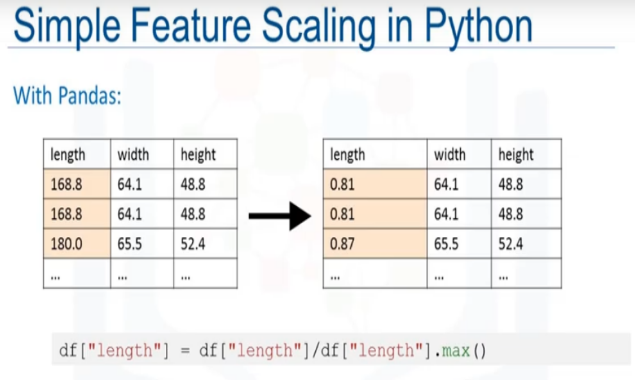
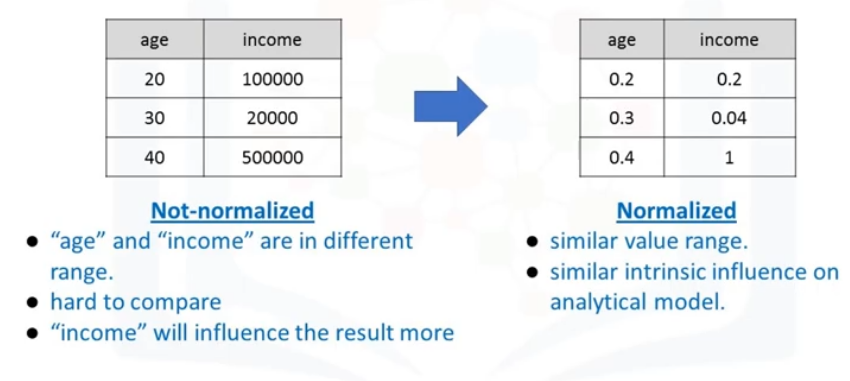
Description automatically generatedpandas dataframe cleaning steps

1. Table

   Description automatically generatedCheck the data type of the columns they should be one of the following

>>>Variable\_name.dtypes()

1. Change the types to their correct format by .astype()



>>>Variable\_name[ “ColumnX” ] = Variable\_name[ “ColumnX” ]. astype( “int or float or object” etc… )

1. See a quick overview of the statistical numerical data by the following code.

>>>Variable\_name.descride()

*# if you want all the columns including object/ non numerical columns use following code.*

>>>Variable\_name.descride(include = “all” )

1. Table

   Description automatically generatedTo see a quick frequency count of categorical data use following:
2. To double check if you missed any miss-types see the first & last 30 rows by

>>>Variable\_name.info()

1. Replace any symbol, 0, ?, N/A or blank cell with NaN

>>>New\_variable\_name = variable\_name.replace('?',np.NaN)

1. Use the dropna() to remove the missing values

>>>Variable\_name = New\_variable\_name.dropna(subset=["columnX"], axis=0)

*#axis = 0 drops entire row, axis = 1 drops entire columns*

1. You can skip 4 and 5 by this one step, if you don’t want a new var and want to edit the old one in one go use this. # ***inplace = True***

>>> Variable\_name.dropna(subset=["columnX"], axis=0 , inplace = True )

1. Normalize the Values in all the data set so no one value has more influence on the dataset then other values. We do that by following.
2. We can use binning to simplify data into bins as shown in the pic.

Table

Description automatically generated

1. *Table

   Description automatically generated with medium confidence*Table

   Description automatically generatedChange any category data “Objects” into Numerical data 0,1 by adding more columns. By using the following code.

Error : invalid syntax

Don’t know why

>>>pd.get\_dummies(Variable\_name[ ‘ColumnX’ ])

Numpy Tabels

>>> Variable\_name.head(‘X’) *# show the table where ‘ X ’ is any Num. of rows*

>>> Variable\_name.shape *# Get the dimensions of the Table [Rows, Columns]*

>>> type(Variable\_name) *# Verify files were read into a …… data frame*

>>> Variable\_name.dtypes *# see the ……. columns data types*

>>> Variable\_name[Column\_Y] = Variable\_name[' Column\_Y '].map(lambda x **:** x.replace( ' ***String or Num or Symbol*** ' , ' ***anything you want*** ' ))

*# Remove the '* ***String or Num or Symbol*** *' from '* ***Column\_Y*** *'by applying a string replace method, this task must be done to each element in the column by using a map function. Where* ***Column\_Y*** *is a name of a column.*

>>> Variable\_name [‘***Column\_Y*** ’].describe()

*# Gather some statistics about the data columns*

>>> Variable\_name ["***Column\_Y*** "].unique()

*# To know what unique values “****Column\_Y****” string column has, run the unique() function on the column, the output is an array of all the unique values in this column. Ex:*

>>> *customer\_data["****MARITAL STATUS****"].unique()*

array(['Single', 'Married', 'Divorced', 'Widow/Widower'], dtype=object)

>>> from datetime import datetime

*#The Enrolment Date column is a date and should be represented as a datetime object. You change the data type of Enrollment date from object to a datetime object, but first you must import the datetime library.*

>>> Variable\_name [**'ENROLLMENT DATE'**]=\

>>> Variable\_name [**'ENROLLMENT DATE'***][ Variable\_name ['ENROLLMENT DATE'].notnull()]*.map(lambda x :datetime.strptime(x, '%d-%m-%Y') )

*# datetime.strptime is a method that returns a datetime that corresponds to date\_string, which is parsed according to a specific format.*

*# Running this command without using this method produces an error because conversion. The reason behind the presence of null values is not a data quality issue, but because some customers are not enrolled / they have no enrolment dates.*

*# To bypass this error, apply only the conversion function to the not-null values by applying a filter on the column and then converting it. Column filters in Pandas can be created by using the following syntax:*

# >>> DataFrame[**column]**[*Column Filter Condition*].map to repeat the replace action on all rows

>>>print( ' null values for Variable\_name ? ', Variable\_name.isnull().values.any())

*# Check whether all the data frames have null values.* ***Answer*** *is [True, False]*

>>> Variable\_name.columns[Variable\_name.isna().any()].tolist()

# Finds where null values are. **Answer** is the Column name containing the null.

*# You now have the total price of each transaction. If you sum up all the values, you get the total revenue for Retailer X. This sum might be a good performance indicator that you can report to Retailer X, but it is better if you can report the revenue per category. To do so, first group by “Category” column, then aggregate by summing the total\_price column. You then sort in descending order based on the revenue. Use the “groupby”, “agg” and “sort\_values” functions for this task*

>>> Table\_File[**‘New Column’**] = Table\_File [**'Column\_X'**] ***\* or / or – or any operator*** Table\_File [**'Column\_Y'**] ***any operator*** (1- Table\_File [**'Column\_Z'**])

>>> Variable\_name = Table\_file.groupby(**‘CATEGORY'**).agg({**'Total\_Price':'sum'**}).sort\_values(**'Total\_Price'**,ascending=False) *# Single Group by*

>>> Variable\_name = Table\_file.groupby ([**‘Column\_X’**, **‘Column\_Y’**]).agg({'Total\_Price':'sum'}) *# Multiple Group by, because when you group by some columns, they change from a column to a multi- or a hierarchical index. To revert them back, use the reset\_index() method.*

>>> Variable\_name.reset\_index().head()

*# .reset\_index() from its name Resets the index and .head() shows the table for us to see. Make it permanent by running the following code.*

>>> Variable\_name = Variable\_name.reset\_index()

>>> Variable\_name = Table\_File.rename(columns={**'Column\_X'**: **'Column\_Y'**})

*# To rename Columns*

Analysis of the distribution of variables using graphs using matplotlib.pyplot

Note: All the plots down are of 1 Variable, in case of Stacked there are 3 Variables 1 one X axis the other 2 are a true or false Column

>>> Variable\_name [**‘** **Column\_Y ’**].value\_counts().plot(kind='bar') *#bar plot*

>>> Variable\_name.plot(kind = 'bar', stacked = True, figsize=(6,6)) *# Stacked bar*

*#You need to identify the* ***Variable\_name*** *Columns Ex:*

>>> Stacked\_plot = pd.crosstab(Table[**‘Column\_X’**], Table[**‘Column\_Y’**]

*# Where* ***Column\_Y’*** *is a Major tab with 2 minor Sub\_tabs or a True and False statement.*

>>> Variable\_name [**‘** **Column\_Y ’**].hist(bins=10) *# histogram*

>>> plt.figure(figsize=(8,8)) *# Box plot, Shows how this*

>>> plt.boxplot(Variable\_name. **Column\_Name**,0,'rs',1) *variable is dispersed*

>>> plt.grid(linestyle='-',linewidth=1)

>>> plt.show()

>>> Variable\_name [‘Variable\_name’].plot(kind='pie',autopct='%1.1f%%',legend = True) *# Pie chart + View*

Note: All the plots down are of Bivariate Analysis (2-variable analysis)

>>> plt.scatter(Table[**‘Column\_X’**], Table[**‘Column\_Y’**])

>>> plt.xticks([0,1,2,3,4,5,6,7,8,9],[1,2,3,4,5,6,7,8,9,10]) # Change the ticks of X

>>> plt.xlabel("Name for X Axis") *# add a name for X axis*

>>> plt.ylabel("Name for Y Axis") *# add a name for Y axis*

>>> plt.show() *#shows the plot*

To convert from Pandas to Numpy

>>> Pandas\_table.head().values

To Merage 2 pandas

>>> Variable\_name = pd.concat([Variable \_input\_1, Variable \_input\_2],axis=1)

To Merage 2 Tables

New\_Variable\_name = Table\_1.merge(Table\_2,how='inner',left\_on = ‘Starting Column name of Table\_1’),right\_on = 'Starting Column name of Table\_2')

To Fill NAN 2 Values with 0

Variable\_name = Variable\_name.fillna(0)

To Import PIXIEDUST *# for advanced no coding Visualization*

>>> Import pixiedust

>>> pixiedust.enableJobMonitor()

>>> pixiedust.Variable\_input()

>>> Variable\_name = pixiedust.Variable\_input()

>>> display(Variable\_Name)

Grouping data in python *#to simplify visualization*

1. Graphical user interface, table

   Description automatically generateddefine what to group by using **.groupby()**
2. Table

   Description automatically generatedFor 2 columns we can use **.pivot()**

Asking For Help

>>> help(str)