Code Sheet - Pandas

Sunday, September 22, 2019   3:08 PM

CREATING AND IMPORTING DATA

```
df = pd.read_csv(''E:/Skills training/Market Basket Analysis/file_name.csv')
df.to_csv('E:/Skills training/Market Basket Analysis/NewFileName.csv')
df_name.to_excel(' ….. ')
help(fun name)
<img src="image.png">  # Adding img to jupyter notebk, if img in Notebook dir. If in sub dir then need to specify path
                            #can drag img and drop into jupytr as well but would make file heavier
```

PANDAS -- Series and Dataframes

```
import pandas as pd
```

SERIES - 1d ndArray with Axis labels

```
pd.Series( ['milk','eggs','brush','soap'] )        #A 1D Pandas Series. And can have diff Datatypes
                                                    a 1d numpy array has no labels.
pd.Series( [[1,2,3,4,5] , [22,33,44,55]] )      # This also a 1d series (single col)
pd.Series(['milk','eggs','brush',22],index=['item1','item2','item3','item4'])      # Pandas Series with Index names
```

DATAFRAME - 2d tabular structure with Index (rows) and Col names

```
pd.DataFrame( [ [1,2,3,4],[22,33,'sha',55] ] )  #Creates a Dataframe. EACH LIST ADDED AS ROWS
pd.DataFrame([[1,2,3,4],[22,33,'sha',55]], index=['row1','row2'])    #Creates DF with Index names
pd.DataFrame([[1,2,3,4],[22,33,'sha',55]], index=['row1','row2'],columns=['col1','col2','col3','col4'])      #DF with
                                                                                                   Index and Column names

df2 = pd.DataFrame({'name':['lampard','greard','rooney'],'club':['chel','pool','utd'],'year':['2006','2012','2009']},
          index=['player1','player2','player3'])        #Pandas Dataframe using a dictionary
#Use this TO COMBINE 2 NPARRAYS INTO A DATAFRAME with Labels
```

#Dataframe to Numpy Array

```
new_arr = df.values
new_nparray = df.loc[:,'col2'].values
```

#Numpy Array to Dataframe

```
df_1 = pd.DataFrame(Array_name, index = ['pl1','pl2','pl3','pl4'], columns=['club1','name','titles'])
```

PANDAS General Commands

```
df.head()
df.tail()
df.shape
df.info()          # can check if any nulls in any column
df.describe().transpose()          #also df.T
df[['col1','col2','col3']].describe
df.columns
df.index       #to refer to the index as a column of a dataframe

df['col_x'].replace(23,33,inplace= True)          #a ctrl+H. Find and replace.
df['col_x'].str.replace("IV","Four",inplace= True)      #A Control H on colx but for string

df['col_name'].astype('float')       #Changing data type of an entry from

data.sort_values(by='Year', ascending = False)
data.sort_values(by=['Year','Country name'])

df.set_index('colName', inplace=True)    #to set index of a dataframe to a certain column or an Array of the correct length
df.reset_index(inplace=True)     #To reset index to default

df.rename(columns = {'curr_col_name':'new_col_name' , '2nd_colname' : 'new2nd'})        # Renaming a column using
```

★ **ALL OTHER DATAFRAME FUNCTIONS**

Dictionary substitution

OPERATIONS ON DATAFRAMES

```
demodata[['ColName1','ColName2']][26:31]    #Slicing a Dataframe
fifa_df[fifa_df['ShortPassing']>89][['Name','Overall']]          #Filtering a dataframe. Filter means essentially Filtering
                                                                  out rows
```

#loc and iloc

#loc [supply label]   iloc [supply index]

#Format supplied is *[rows,[columns]]*

```
fifa_df.loc[2:10,['Name','Age','Club']]      #using loc[]
defenders_analysis_df.loc[defenders_analysis_df['StandingTackle']>=88,'Name':'Potential']   #Filtering data

top_tacklers.iloc[0:5,0:3]       #using iloc[]
defenders_analysis_df[defenders_analysis_df['ShortPassing']>89].iloc[:,[0,2,5]]

#When multiple filter condition, result can be acomplished using ( ) as seen below
top_oceania_wines = reviews[(reviews.loc[:,'points']>=95) & \
((reviews['country']== 'Australia') | (reviews.loc[:,'country']=='Italy'))][:]
#Also here  \  used to split command into 2 lines
```

#LAMBDA FUNCTIONS (PENDING)

~~*df_name.apply(lambda x : min(dataset) + max(dataset))*~~

#Also apply()

OTHER COMMANDS

#In Pandas ROW is AXIS 0 and COLUMNS is AXIS 1

```
fifa_df['Nationality'].unique()       #List of unique entries

nunique()          #Count(number) of unique entries

fifa_df['Position'].value_counts()         #Unique entries in a column and its frequency
```

# GROUP BY

```
df.groupby('field_to_grpby)['colname_to_agg'].sum()   #Group By
df.groupby('row_name').mean()        #Group By
```

# PIVOT TABLE link

```
basket = df.pivot_table(index='InvoiceNo',columns='Description',values='Quantity',aggfunc=np.sum,fill_value=0)
```

# Add Column with name New_Scores

```
pos_list = ["ST","GK","CB","LS","LB","CB", ..............'CM']
df['New_Col_name'] = pos_list                 #Can create a list or a nd Array and assign to a new col
OR
df['New_Scores'] = [10,20, 45, 33, 22, 11]           #Can directly assign values to a new col

dfObj['Percentage'] =  (dfObj['Marks'] / dfObj['Total'] ) * 100         #Creating new col based of existing cols

demodata.columns= ['col1','col2','col3','col4','col5']     #Renaming Columns (All col names must be specified)
```

DATA CLEANING

#FOR All **DROP Commands, By default original dataframe is not changed**, but **n** dataframe itself.
#Either that of assign to a new Dataframe.

```
df[col_name].isna()      #Checks NAs. Returns a Boolean Series. Can use as filter to assign values to entries with missing data
df.isna().any()     #Will list all cols. Cols with 'True' have nulls

df.dropna(inplace= True)            #Drops fields (entire Row) with missing values Nan (can change to axis 1)
df.fillna(0, inplace = True)          #Dont want to drop the rows so fill Nas with 0
df[col_name].fillna(method='bfill')      #Dont want to fill with 0, but a value
df.dropna(axis=0, subset=['Col_name'])        #Drop row only if entry in specified col has NA
df.dropna(axis=1)      #to drop the columns with NA


#To delete a column
new_df = df.drop('Col_Name',axis=1)          #Need to specify Axis. By default drop() drops rows
df.drop(['col_1','col_2','col_3'],axis = 1, inplace = True)         #To drop multiple cols in the same df

del df['col_name']         #Another approach to del a col

#To copy a dataframe
df_1 = df
func1(df_1)
#Here df_1 is a view of df and not a copy. So func1 will modify df through df_1

df2 = df.copy()       #Will create a copy
```

JOINS / COMBINE

```
df1.append(df2)                        #Rows of df1 and df2 are combined, duplicates remain.

df_union_all= pd.concat([df1, df2])                #Rows added to the end
df_union= pd.concat([df1, df2]).drop_duplicates()        #Rows added to the end and no duplicates

pd.concat([df1, df2],axis=1)      #add the columns in df1 to the end of df2 (rows should be identical)

df1.join(df2,on='col1',how='inner')            #This takes into account the Index column as well.
```

.merge() works same as join.

```
joined_df = pd.merge(left_df, right_df, how='inner', left_on= 'col_1name', right_on='col_2name)
#basically left_df inner join right_ df on left_on 'col' = right_on 'col'
```

ANALYTICAL FUNCTIONS

#Calculated for along a column/columns. Can change to rows

```
df['col_name'].min()
df['col_name'].max()
sum
mean/median/quantile
idxmin/idxmax         #will return the index of the row where the first minimum/maximum is found.
```

SQL Within PYTHON

-- Reading from SQl

CORRELATIONS

```
corr_matrix =  df.corr()                   #Creates a correlation matrix of every attribute against every other attribute
```

Code Sheet - Numpy

Tuesday, December 31, 2019 1:54 PM

***NUMPY** -- Arrays (1d arrays or nd arrays)*
all elements of same type

```
import numpy as np
```

Creating Numpy Arrays

```
List1 = [11,22,54,17]                #List
np.array(list_name)        #1d np Array. Remember simpliest form of array is list. so keyword 'array' + listname
                                                                                              OR a list of lists
```

```
np.arange(4,13,step=2)        #returns evenly spaced values between [4incl & 13excl). Step optional (default 1)
np.linspace(3,7,num=9)        # returns 9 samples (num=9) between 3 and 7 (both included)
np.random.randint(2,7,size=12)          #returns 12 random integers between 2 incl and 7excl
np.random.random(size=6)          # 4 random floating point numbers
np.random.random(size=6).reshape(2,3)          # a (2,3) array of random floating point numbers
```

```
Using above 3 can create 2 dim arrays --
np.arange(4,19).reshape (3,5)        # 2 dim array with 3 rows 5 cols.
np.random.randint(4,9,size = 15).reshape(5,3)              #2 dim array
```

```
arr_name.shape()              #To get shape of an array
arr_name.ndim            #To get dimension of an array 1,2 etc.
```

> *A **shape** of **4,3** is a 2 dimensional array*
> *4 indicates its 4 - 1d arrays*
> *3 elements each (or 4 rows 3 cols)*
> *Ill'ly*
> *5,4,3 is a 3 dimensional array.*
> *5 indicates its 5 - (4x3 arrays) 2 dimensional arrays*
> *4 incdicates 4 groups of 1d arrays in each matrice*
> *3 indicates no of elements in each array*

Therefore

```
Arr1 = np.array([List1,List2,List3])       #Creating a 2D Numpy Array List1,2,3,4 each have 4 elements
```

Arr1 has 3 - 1d Arrays.
Each array has 4 elements
Therefore -- 2 Dimensional array; Shape 3,4

```
arr_t2 = np.array( [ [ l1,l2] , [l3,l4] ] )
```

A 3 Dimensional array.
has shape 2,2,4 (So 2 - (2x4) arrays)
each group has 2 - 1d arrays
each array has 4 elements.

```
#Can also have 4 dimensional array
```

SLICING

```
#Arr1[rowno,columnno]                    #Slicing a matrix
Arr1[ row x : row y , col p : col q]

test_3D_array[1,3,2] =1111             #To assign a new value to an entry in a 3 dim array

arr3 = np.copy(arr1)        #Will create a deep copy of an array.

arr2 = arr1[2:4,3]                # not a real copy. New reference (name) for same location in memory
                                 #A slice does not create a copy. Changes made to elemens of a slice will effect the original
```

OTHER FUNCTIONS

```
np.zeros((5,4,3))   /    np.ones(4)        #array of 0's or 1's. Can be 1 or n dimensional.


array_name.size    #no of elements or length
arr_name.dtype       #datatype of elements in the array
type(arr_name)      #type

np.unique(test_arr[:,1])        #unique values in an Array

np.inner --
np.dot (mat1,mat2)       #matrix multiplication function of numpy. (as done in math)

np.logical_and            #numpy's logical AND function

np.random.rand(2,6)            #Random values in a given shape. from uniform distribution between [0,1)
```

APPEND

```
arrB=np.append(arrA, [5,6,7,8])      #append 5,6,7,8 to arrA
np.append(arrA,arrC, axis=0)          #Append along axis 0  #Remember Axis can take axis = 0,1,or 2

hstack (horiz stack is same as append)
```

Boolean Masks

Ex - Getting all entries that are divisible by 7

```
my_vector = np.array([-17, -4, 0, 2, 21, 37, 105])
mask_arr = 0 == (my_vector%7)             #gives a array mask_arr with boolean values
my_vector[mask_arr]            #returns entries for True

my_vector[(my_vector%7)==0]       #Can be done directly
```

BROADCASTING

Code sheet - Plotting

Tuesday, December 31, 2019 6:22 PM

PLOTTING

*~~~~SYNTAX TO REMEMBER~~~*

#MATLAB SYNTAX

```
plt.plot(Arr1, Arr2 ...)     #Simple plot of ColX or XvsY as lines or marks. No specific type of plot.
                               # NOTE - df.loc[:,'col'].values  to convert to numpy array

plt.plot(df[col4X] , ...)     #Simple plot of ColX or XvsY as lines or marks. No specific type of plot.
plt.scatter(df[col1],df[col2]..)      #Similarly we have plt.hist
```

#OBJECT ORIENTED SYNTAX

```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2,figsize=(8, 4))    #initializing FIGURE and AXES. & initialize objects
ax1,ax2 for each axes
ax1.scatter(…)
```

Subplots

```
plt.figure(figsize=(4,6)
plt.subplot(221) ; plt.hist(…)       #for multiple Axes/plots/vizzs in same figure.    2 rows 2 columns 1 index
```

PANDAS

```
df.plot( 'col4x' , 'col4y' , kind='…', figsize=(8,12))   <<-- can use this as the go to command
df[['col1','col2','col3']].plot(kind='bar')       #Plot the dataframe i.e. df[[col1,col2,col3]] aginst INDEX
```

SEABORN

```
sns.boxplot(df[colA],df[colB])      #So works directly with dataframes
#OR:
sns.boxplot(x='colname', y = 'col2name', data=df_name)      #works directly with dataframes
```