# PROJECT 3 EGR 491 Python Pandas

Althoug NumPy provided wonderful tools for data manipulation there is a better module for larger datasets. This module is known as pandas. The panda import does not provided you with an endangered spieces of bear but provied three data sturfictures which will be explored later.

## 1. Table of Contents

* [1. Table of Contents](#X790031b8412ba72e400d8037ec4a34901313c49)
* [2. Table of Display](#Xc18aed7918c6c742a699bc018b233a2549efdd5)
* [3. Background](#X87094d5cc2003181c30bd3fb8c9e78d4fb0c530)
  + [3.1. DataObjects](#X24d33430d4fe665b7d554ae040752381a459047)
    - [3.1.1. Series](#X8c2fb325868df296f7aaa73f9ae1977fb2ba6da)
    - [3.1.2. DataFrame](#Xa51439f679ce2169b615be656c5ce6a614eded7)
  + [3.2. Data Manipulation](#X34198a2aa3d9d65a5f382e57f992cc134bfbeb0)
    - [3.2.1. uFuncs and Broadcasting](#X00f30b9723f25b681493c1ceed731db5cbadb5f)
    - [3.2.2. Missing Data](#X13e6c0059eec4d1c9df5a97da7647375347df9c)
    - [3.2.3. Advanced Dataframes](#Xf21b0e67b12ca55297d77c7197ab5950ff557b6)
      * [3.2.3.1. Advanced Indexing](#X99127e67e3c1021290d77fc0d54c0f38955e239)
      * [3.2.3.2. Changing Columns](#X3c53c5afd1d2f7052f0c08e8282b48ef6eb59b1)
      * [3.2.3.3. Sorting](#Xfdadac233be1bc9f09e354cd09927ca1fdd607c)
      * [3.2.3.4. Fliping](#X28cb1a07dab29d1409100aeea1efe869e4c6f70)
      * [3.2.3.5. Combining Tables](#Xfe474fa87cf23f91cf423184479601af96d1a6b)
    - [3.2.4. Querying and Evaluating DataFrames](#Xfb907a80b8b45833a00382e4f4719fc83041a54)
    - [3.2.5. Strings](#X8b20c61b718a480eefa112ab4625865793d62ef)
  + [3.3. Files](#Xc89abfdd3d3cf3ce01a25446acf894b9c1aa4e9)
* [4. Code](#Xcb6cb0c72f6c02e38915d8558c711e8104fab95)
  + [4.1. Output](#Xbd406c83490d69e1a9a7a43516ca9e7c327b82f)
  + [4.2. Code](#X0f9131522ceeaa9d542a31395d809401f29d4c6)

## 2. Table of Display

|  |  |
| --- | --- |
| Topic | Method |
| Series | Summary w/ Code |
| DataFrame | Summary w/ Code |
| Index | Summary w/ Code |
| Indexing/Selection of Data | Summary |

## 3. Background

The pandas module is a more complex addition to NumPy and reqiures that module to be installed and automatically comes installed with Jupyter Labs or can be installed by using “pip install pandas” in the command line.

### 3.1. DataObjects

Pandas is designed as a more advaced table interfacting system. To use a pandas object you will either use a Series or the much more powerful dataframe

#### 3.1.1. Series

The simpliest of these data structures is the pandas series which is simply an array. The uniques attribute of a Panda Series is the ability to use unique indexes. As seen below.

*EXAMPLE*

import pandas as pd  
  
pand1 = pd.Series([1,2,3,4,5]) # This is a normal array  
pand2 = pd.Series([1,2,3,4,5], index=['one','two','three','four','five']) # This is an array with special indexes  
  
dic = {'one':1, 'two':2, 'three':3, 'four':4}  
pand3 = pd.Series(dic)  
  
print(pand1)  
print(pand2)  
print(pand3)  
###  
#OUTPUT  
#0 1  
#1 2  
#2 3  
#3 4  
#4 5  
#dtype: int64  
#one 1  
#two 2  
#three 3  
#four 4  
#five 5  
#dtype: int64  
#one 1  
#two 2  
#three 3  
#four 4  
#dtype: int64

#### 3.1.2. DataFrame

The most popular and power data tool Pandas provides is the Dataframe. This object is a more complex version of the numpy structured array, but it is a bit more powerful.

This is because each dimension of a dataframas array is a labeled array similar to how an Excel table works. Below is a simple example of how to make a dataframe.

import pandas as pd  
  
employees = ['Paul', 'Shelby', 'Josh', 'Phil', 'Steve']  
pay = [25.00, 15.75, 13.24, 10.11, 15]  
tenure = [10, 5, 7, 13, 45]  
  
employInfo = pd.DataFrame({'Employee':employees, 'pay':pay, 'Tenure':tenure})  
print(employInfo)  
# Employee pay Tenure  
#0 Paul 25.00 10  
#1 Shelby 15.75 5  
#2 Josh 13.24 7  
#3 Phil 10.11 13  
#4 Steve 15.00 45

You can also access any column by calling out the column name such as employInfo.pay and you can even add columns that cosist of operations between multiple columns. Below is an exmaple

import pandas as pd  
employees = ['Paul', 'Shelby', 'Josh', 'Phil', 'Steve']  
pay = [25.00, 15.75, 13.24, 10.11, 15]  
tenure = [10, 5, 7, 13, 45]  
employInfo = pd.DataFrame({'Employee':employees, 'Pay':pay, 'Tenure':tenure})  
employInfo['RPY'] = (employInfo['Pay']-7.25)/employInfo['Tenure']  
print(employInfo.T) # This will print the transposed table  
#######################################################  
# 0 1 2 3 4  
#Employee Paul Shelby Josh Phil Steve  
#Pay 25.0 15.75 13.24 10.11 15.0  
#Tenure 10 5 7 13 45  
#RPY 1.775 1.7 0.855714 0.22 0.172222

There are three ways to access its data these are

* iloc: This uses numbers to index each column such as employInfo.iloc[:2, :2]
* loc: This uses the Index Names the such as employInfo.loc[]:"Josh", :"Pay"].
* ix: Alows the use of both indexes and the column titles. employInfo.ix[:'Josh', :2]
* Mask: These check each row agaisnts a boolan expression employInfo[employInfo.Tenure >= 10]

### 3.2. Data Manipulation

#### 3.2.1. uFuncs and Broadcasting

Panda objects do support the same universal functions as a numpy array. These operations will keep the same row/column information that the series has. This also means that pandas will atempts to align all indexies as provided.

To display this we will consider the following objects.

import pandas as pd;  
import numpy as np;  
  
myIndex = pd.Index(['a','b','c','d','e'])  
myIndex2 = pd.Index(['a','c','d'])  
ds1 = pd.Series(np.random.random(5), index=myIndex)  
ds2 = pd.Series([1,2,3], index=myIndex2)  
ds3 = pd.Series([0,1,2,3,4], index=myIndex)  
df = pd.DataFrame({'rand':ds1,'set':ds3}, index=myIndex)

If we were to print ds1+100 we would get

a 100.163441  
b 100.578039  
c 100.096041  
d 100.963564  
e 100.549283  
dtype: float64

If we were to pritn df+10 we would get

rand set  
a 10.163441 10  
b 10.578039 11  
c 10.096041 12  
d 10.963564 13  
e 10.549283 14

But something different happens when we add two series together. Consder adding ds1 and ds2. This will ouput.

a 1.163441  
b NaN  
c 2.096041  
d 3.963564  
e NaN  
dtype: float64

Notice that the indicies were added together properly, but for the indecies which ds2 did not have (b and e) the result becomes NaN, but this is just the default behavious. If you were to use ds1.add(ds2) you can set the output with the function parameter fill\_value. If you were to use a *fill\_value* of 0 your output would be

a 1.973342  
b 0.408203  
c 2.147969  
d 3.458576  
e 0.430023  
dtype: float64

#### 3.2.2. Missing Data

When working on large datasets some functions on multiple DataFrames or Series will result in tables which will have NaN or null values. Therefore pandas proviedes a few functions to asasist in changing null values or to simply ignore them. These are:

* isnull(): Returns a mask the location of each null value as **TRUE**
* notnull(): Returns a mask the location of each null value as **FALSE**
* dropna(): Returns the Array with all Null values removed, in dataframs it drop both rows and columns unless specifiec by the *axis* prameter.
* fillna(): Fills the Null values with a specified value or and use ffill which pulls the previous value or bfill which pulls the next non null value. The *axis* parameter can be used on fillna as well.

#### 3.2.3. Advanced Dataframes

Dataframs can be manipulated in many ways and the pandas documentation includes a wonderful [cheat sheet](https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf) for quick reference.

##### 3.2.3.1. Advanced Indexing

This feature applies both to the Series and Dataframe Object and can be used to add indecies. A simple way to do this is set you index object to a list of tuples such as

import pandas as pd  
 dex = pd.MultiIndex.from\_tuples([('a',0),('a',1),('a',2),('b',2),('b',3)], names=['letter','nummber'])

Notice that there is a extra parameter *names* which will name each index like a column in the main part of the data.

##### 3.2.3.2. Changing Columns

Columns of a dataframe can be editing using .rename(columns = "old\_name":"new\_name"}). Or columns can be delted using .drop("column name"), this function can also use a list of column names for quickly droping multiple columns. Futher editing can be done by using .reset\_index() which will make the index a column within the dataframe and replaces the index with the default numbered index.

##### 3.2.3.3. Sorting

To sort a dataframe use the .sort\_values('column\_name') to sort columns in ascending order (for decending order set the parameter *ascending* to False). To sort a dtaframe by the index use .sort\_index().

##### 3.2.3.4. Fliping

Sometimes it is useful to flip a table such that the rows become columns and the columns become rows. Pandas provides two methods for this. The first is pandas.melt(data\_frame) which changes each columns to a row. The second is data\_frame.pivot(columns, values). The *columns* row values you would like to be columns. The *values* parameter is the is the values that would be placed into the new columns. Below is an example of how these functions work

import pandas as pd;  
import numpy as np;  
idex = pd.Index(['a','b','c','d','e'])  
s1 = pd.Series(np.arange(5), idex)  
s2 = pd.Series([100,200,300,400,500], idex)  
df = pd.DataFrame({'Ones':s1, 'Hundreds':s2})  
print(df)  
print(pd.melt(df))  
print(df.pivot(columns='Hundreds', values='Ones'))

OUTPUT  
---------------  
 Ones Hundreds  
a 0 100  
b 1 200  
c 2 300  
d 3 400  
e 4 500  
 variable value  
0 Ones 0  
1 Ones 1  
2 Ones 2  
3 Ones 3  
4 Ones 4  
5 Hundreds 100  
6 Hundreds 200  
7 Hundreds 300  
8 Hundreds 400  
9 Hundreds 500  
Hundreds 100 200 300 400 500  
a 0.0 NaN NaN NaN NaN  
b NaN 1.0 NaN NaN NaN  
c NaN NaN 2.0 NaN NaN  
d NaN NaN NaN 3.0 NaN  
e NaN NaN NaN NaN 4.0

##### 3.2.3.5. Combining Tables

You may find it neccesary at times to load multiple tables and attach them together. There is where the pandas.concat() function comes in handy. First you will input a list of Dataframes, this will by deafult append rows to the Datatables; but if you use the *axis* parameter you can either append rows (axis = 0) or append columns (axis= 1).

In addition to appending you can also merge rows using the pandas.merge() function. This function takes two dataframes in. By defulat thsil will output the intersection of the tables, but if you set the *how* paramter to “outer” it will output the Union of the dataframes.

#### 3.2.4. Querying and Evaluating DataFrames

Those familiary with databases will love the pandas functions .eval() and .query() which allow you to search or run functions.

The .eval() function will take in a string function that can inslude arithmatic, compaiison, and bitwise functions. These evaluation can also use paramters throug the method df.column and specific index values through df[index]. By comging index and column values you can access specific values via df.column[index]. In addtion to using internal values the @ tag can be used at the begining of local variables to be used in a function. When used on a dataframe .eval() cna use the *inplace* parameter to add columns to its body when the funciton string assigns the value a column name.

The .query() function allows you to query a dataframe similar to how one would query a database with SQL. The same paramters tools you can use iwth eval apply. If you run a query it will return any rows which will output the rows that meets the required values.

import pandas as pd;  
import numpy as np;  
s1 = pd.Series([0,1,2,3,4,5])  
s2 = pd.Series([3.14,7.19,6.4])  
value = 3  
print(s1); print(s2);  
df = pd.DataFrame({'A':s1,'B':s2})  
print(df)  
print("Where is A Greater than ", value)  
print(df.eval('(A < @value)'))  
print("What Data is in those locations")  
print(df.query('A < @value'))

OUTPUT  
0 0  
1 1  
2 2  
3 3  
4 4  
5 5  
dtype: int64  
0 3.14  
1 7.19  
2 6.40  
dtype: float64  
 A B  
0 0 3.14  
1 1 7.19  
2 2 6.40  
3 3 NaN  
4 4 NaN  
5 5 NaN  
Where is A Greater than 3  
0 True  
1 True  
2 True  
3 False  
4 False  
5 False  
dtype: bool  
What Data is in those locations  
 A B  
0 0 3.14  
1 1 7.19  
2 2 6.40

#### 3.2.5. Strings

Strings can be manipulated using **Vertorized** functions to use these simply add .str.<function> after you dataframe or series as seen below.

import pandas as pd  
ds1 = pd.Series(['all', 'these', 'are', 'lower', 'case'])  
print(ds1)  
print(ds1.str.capitalize())  
ds2 = pd.Series(['O|M|C','M|D|G','S|D|D'])  
print(ds2)  
print(ds2.str.get\_dummies())

OUTPUT

0 all  
1 these  
2 are  
3 lower  
4 case  
dtype: object  
0 All  
1 These  
2 Are  
3 Lower  
4 Case  
dtype: object  
0 O|M|C  
1 M|D|G  
2 S|D|D  
dtype: object  
 C D G M O S  
0 1 0 0 1 1 0  
1 0 1 1 1 0 0  
2 0 1 0 0 0 1

### 3.3. Files

Pandas also allows for input and output to common tabular data formats. To use these functions there are two types of functions to read and write. To read from a file you will use pandas.read\_<file type> you can replace the file type with anything from html and csv, to excel and sql. To wrtie to tiles you must use dataframe.to\_<file type> which will write to alsmost all the file types which pandas can read. If you are curous of which files you can use I suggest going to [Pandas Documentation](https://pandas.pydata.org/docs/user_guide/io.html).

## 4. Code

### 4.1. Output

Hello and Welcome to The Income Data Manager  
Checking for a database  
You do not have a database  
Do not worry we will get you started  
How Much are you payed/hr?10  
How many hours did you work?10  
How much did you take home?90  
 Date Pay/Hr Hours Net  
2021-06-12 2021-06-12 10.0 10.0 90.0  
Please Select an option  
1. Make a new Entry  
2. View Statistics  
3. Print Data  
4. Exit  
1  
How Much are you payed/hr?10  
How many hours did you work?20  
How much did you take home?185.33  
 Date Pay/Hr Hours Net  
2021-06-12 2021-06-12 10.0 20.0 185.33  
Please Select an option  
1. Make a new Entry  
2. View Statistics  
3. Print Data  
4. Exit  
2  
avg Pay/Hr 10.00  
 Hours 15.00  
 Deduction% 0.09  
est Annual Income 3561.97  
dtype: float64  
Please Select an option  
1. Make a new Entry  
2. View Statistics  
3. Print Data  
4. Exit  
3  
 Date Pay/Hr Hours Net Gross Deduction Deduction%  
2021-06-12 2021-06-12 10.0 20.0 185.33 200.0 14.67 0.07335  
2021-06-12 2021-06-12 10.0 10.0 90.0 100.0 10.0 0.1  
Please Select an option  
1. Make a new Entry  
2. View Statistics  
3. Print Data  
4. Exit  
4  
--------------------------  
Hello and Welcome to The Income Data Manager  
Checking for a database  
Your database is located loading now  
Your Database has been loaded  
Please Select an option  
1. Make a new Entry  
2. View Statistics  
3. Print Data  
4. Exit  
3  
 Date Pay/Hr Hours Net Gross Deduction Deduction%  
2021-06-12 2021-06-12 10 20 185.33 200 14.67 0.07335  
2021-06-12 2021-06-12 10 10 90.00 100 10.00 0.10000  
Please Select an option  
1. Make a new Entry  
2. View Statistics  
3. Print Data  
4. Exit  
1  
How Much are you payed/hr?15  
How many hours did you work?10  
How much did you take home?120.15  
 Date Pay/Hr Hours Net  
2021-06-12 2021-06-12 15.0 10.0 120.15  
Please Select an option  
1. Make a new Entry  
2. View Statistics  
3. Print Data  
4. Exit  
2  
avg Pay/Hr 11.67  
 Hours 13.33  
 Deduction% 0.12  
est Annual Income 3542.46  
dtype: float64  
Please Select an option  
1. Make a new Entry  
2. View Statistics  
3. Print Data  
4. Exit

### 4.2. Code

#%%  
# This project will keep trake of your income  
# It will estmate the amount you are taxed and give you an estimated annual income  
  
#%%  
# Imports/Constants/Global Vars  
from os import stat  
import pandas as pd  
import numpy as np  
import datetime as dt  
FILE\_NAME= "incomeSheet.xlsx"  
INCOME\_SHEET = 'INCOME'  
STATS\_SHEET= 'STATISTICS'  
stats = pd.Series([None])  
masterDB = pd.DataFrame(columns=['Pay/Hr', 'Hours', 'Net'])  
#%%  
#  
def check4Excel():  
 import os  
 return os.path.isfile(FILE\_NAME)  
# %% Read Excel  
def readExcel(filename, sheet):  
 xlFile = pd.ExcelFile(filename)  
 return pd.read\_excel(xlFile,sheet\_name=sheet, index\_col=0)  
  
# %% Make Data Entry  
def makeEntry():  
 global masterDB  
 date = dt.date.today()  
 hrPay = float(input('How Much are you payed/hr?'))  
 hrs = float(input('How many hours did you work?'))  
 netPay = float(input('How much did you take home?'))  
 data = pd.Series([date,hrPay, hrs, netPay], name=str(date), index=['Date','Pay/Hr', 'Hours', 'Net'])  
 print(data.to\_frame().T)  
 if masterDB.empty:  
 masterDB = data.to\_frame().T.round(2)  
 else:  
 masterDB = pd.concat([data.to\_frame().T, masterDB]).round(2)  
  
  
# %% Run Calcualtions  
  
def masterCalcs():  
 global masterDB  
 masterDB['Gross'] = masterDB['Pay/Hr'] \* masterDB['Hours']  
 masterDB['Deduction'] = masterDB.Gross - masterDB.Net  
 masterDB['Deduction%'] = masterDB['Deduction']/masterDB.Gross  
  
def statCalcs():  
 global stats  
 Idex = pd.MultiIndex.from\_tuples([('avg', 'Pay/Hr'),('avg', 'Hours'), ('avg','Deduction%'),('est','Annual Income')])  
 avgPayPerHr = masterDB['Pay/Hr'].mean()  
 avgHrs = masterDB.Hours.mean()  
 avgDed = masterDB['Deduction%'].mean()  
 estIncome = ((avgPayPerHr\*avgHrs)\*(1-avgDed))\*(52/2)  
 stats = pd.Series([avgPayPerHr, avgHrs, avgDed, estIncome], index=Idex).round(2)  
# %% Menu  
def menuLogic(u\_in):  
 global masterDB  
 if u\_in == '1': # Make New Data Entry  
 makeEntry()  
 elif u\_in =='2': # Make  
 print(stats)  
 elif u\_in == '3':  
 print(masterDB)  
 elif u\_in == '4':  
 print('Goodbye')  
 return  
 else:  
 print('Invalid Input')  
  
 masterCalcs()  
 statCalcs()  
 print\_menu()  
  
def print\_menu():  
 global masterDB  
 print('Please Select an option')  
 print("1. Make a new Entry")  
 print("2. View Statistics")  
 print("3. Print Data")  
 print("4. Exit")  
 menuLogic(input())  
  
# %% Welcome/Start  
# Welcome  
print('Hello and Welcome to The Income Data Manager')  
print('Checking for a database')  
  
if check4Excel():  
 print('Your database is located loading now')  
 xl = pd.ExcelFile(FILE\_NAME)  
 masterDB = readExcel(FILE\_NAME, INCOME\_SHEET)  
 print('Your Database has been loaded')  
 print\_menu()  
else:  
 print('You do not have a database')  
 print('Do not worry we will get you started')  
  
 makeEntry()  
 masterCalcs()  
 statCalcs()  
 print\_menu()  
  
with pd.ExcelWriter(FILE\_NAME) as w:  
 masterDB.to\_excel(w, sheet\_name=INCOME\_SHEET)  
 stats.to\_excel(w, sheet\_name=STATS\_SHEET)  
 print("FILES SAVED")  
# %%