Asignacion1

April 13, 2022

1 Asignación

Actividad desarrollada por Marcial Mosqueda

A continuación se mostrará el programa desarrollado paso a paso y, adicionalmente, encontrará las respuestas a las preguntas del apartado 1 de la asignación

[1]:
```
import pandas as pd
import plotly.express as px
from IPython.display import Image
```

1.1 Sección: Comprobación del archivo y tratamiento de la data

Nota: Este apartado pertenece a la segunda parte de la asignación

Debido a que me gusto lo aprendido en clases de crear nuestras propias excepciones, decidí crear 3 especificas para el ejercicio propuesto con la intención de prácticar!!!

[2]:
```
class Errors(Exception):
    pass

class ColumnError(Errors):
    pass

class SepsError(Errors):
    pass

class EmptyError(Errors):
    pass
```

Creamos una función responsable de importar el csv y su validación, en caso de error se genera una excepción

1. Se comprueba el fichero y cantidad de columnas
2. se comprueba que las columnas sean las indicadas por el usuario
3. Se valida que no existan columnas vacias

[3]:
```
def create_df(path, columns, seps=[',',]):
    """Crea un dataframe de un archivo csv
    path:str -> path donde se encuentra ubicado el csv
```

```
    columns:list -> lista con las columnas en el csv
    sep:list -> lista con los posibles separadores usados en el csv """
    #Validación de los argumentos
    assert(type(path)==str),f'path debe ser un string y fue suministrado un␣
↪{type(path)}'
    assert(type(columns)==list),f'columns debe ser de tipo lista y fue␣
↪suministrado un {type(columns)}'
    assert(type(seps)==list),f'seps debe ser de tipo lista y fue suministrado␣
↪un {type(seps)}'
    assert(all(isinstance(x,str) for x in seps)),'elementos en seps deben ser␣
↪del tipo string'
    try:
        df = pd.read_csv(path,sep=seps[0]) #Creación del DataFrame
        check = False

        # Primera validación: Validación del constructor y pruebas con␣
↪separadores
        if len(df.columns) == len(columns):
            check = True
        else:
            for item in seps[1:]:
                df = pd.read_csv(path,sep=item)
                if len(df.columns)==len(columns):
                    check = True
                    break

        # Segunda validación: Columnas en el DataFrame y separador
        if not check:
            if len(df.columns)==1:
                raise SepsError
            msg = 'El Dataframe no posee la cantidad de columnas indicadas'
            raise ColumnError

        for col in columns:
            if col not in df.columns:
                msg = f'Columna {col} no se encuentra en el DataFrame'
                raise ColumnError

        cols_vals = {col:df[col].isna().values.all() for col in df.columns}

        # Tercera validación: Datos en las columnas
        if sum(cols_vals.values()) != 0:
            raise EmptyError

        return df

    except IOError:
```

```
        print(f'validar el path suministrado {path}')
    except SepsError:
        print('Validar los separadores indicados del csv')
    except EmptyError:
        print('Validar DataFrame y/o separadores suministrados\n'+
              'Columnas vacias\n' + f'{cols_vals}')
    except ColumnError:
        print(msg)
```

Procedemos a crear una función que nos ayude a validar los daros del Dataframe. Pasos de la función:

- Al trabajar con data numérica se validará solo columnas que no posean valores numéricos
- Se eliminaran las comillas simples o dobles de las celdas que contengan comillas
- Se convertirán los errores y/o NaN en 0

La función se aplicará al Dataframe a través del método apply()

[4]:
```
def check(col):
    if col.dtypes == 'O':
        col = col.str.replace("'",'')
        col = col.str.replace('"','')
        col = pd.to_numeric(col,errors='coerce').fillna(0)
    return col
```

1.2 Sección: Implementación de la asignación

Declaración de los siguientes parámetros:

- Path del archivo
- Columnas
- Posibles separadores

Nota: El archivo csv debe estar en el mismo fichero del .py

[5]:
```
path = 'finanzas2020[1].csv'

columns = ['Enero','Febrero','Marzo','Abril','Mayo','Junio',
    'Julio','Agosto','Septiembre','Octubre','Noviembre','Diciembre']

seps = [',',';',':','\t']
```

Generamos el Dataframe del archivo Finanzas2020 con ayuda de la función creada en el apartado 1

[6]:
```
data = create_df(path,columns,seps)
data
```

[6]:

	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	\
0	-760	343	265	-624	-390	-796	601	-780	-491	645	

```
1    223      491   -397   -123     6   -115   157    -741       -951     267
2   -872     -913    558    278   544   -223   607    -113        348     576
3    111     -842    730   -761   158   -963  -290    -669        191     130
4    919      111   -688     15   395      9   553     297       -302     695
..   …       …     …      …     …     …     …       …          …
95  -652      233    -65    431  -593    -72  -140    -159       -581     771
96   777     -905   -405    110  -444   -978  -285    -346        207    -363
97  -380      946   -790      9   -59   -743  -296     321       -767     944
98   244      -44   -271    462   -55   -699   674    -203       -792    -848
99   126     -221    290   -960  -303   -384   506     -11       -456     268

   Noviembre  Diciembre
0       -248        714
1         14       -596
2       -977        195
3        170       -274
4        730       -731
..       …          …
95       191        313
96       696       -971
97      -116       -873
98       263       -378
99       630        271

[100 rows x 12 columns]
```

Validemos el tipo de datos de cada columna

[7]:
```
data.dtypes
```

[7]:
```
Enero         object
Febrero        int64
Marzo          int64
Abril          int64
Mayo           int64
Junio          int64
Julio         object
Agosto         int64
Septiembre    object
Octubre       object
Noviembre     object
Diciembre      int64
dtype: object
```

Observamos que tenemos varias columnas con un tipo de Objeto, por lo que alguno de sus items son strings.

Para corregir esta situación, haremos uso de la función “check” definida en el apartado anterior.

Nota: Recordar que se debe aplicar a través del método apply()

[8]:
```
data = data.apply(check)
data.dtypes
```

[8]:
```
Enero           int64
Febrero         int64
Marzo           int64
Abril           int64
Mayo            int64
Junio           int64
Julio           int64
Agosto          int64
Septiembre    float64
Octubre       float64
Noviembre       int64
Diciembre       int64
dtype: object
```

Como podemos observar las 12 columnas poseen un tipo de datos numérico.

Los meses de Septiembre y Octubre poseen un tipo de dato tipo float64 debido a que contenian errores (NaN) luego de la conversión

Procederemos a hacer una revisión general del DataFrame con describe()

[9]:
```
data.describe()
```

[9]:

	Enero	Febrero	Marzo	Abril	Mayo	Junio
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	115.230000	0.390000	-79.690000	-189.330000	103.040000	-14.770000
std	545.948842	573.923743	585.916595	538.862139	507.166817	560.952596
min	-966.000000	-989.000000	-979.000000	-960.000000	-981.000000	-978.000000
25%	-359.000000	-475.000000	-586.500000	-666.500000	-368.000000	-476.750000
50%	169.500000	-0.500000	-159.000000	-212.500000	151.000000	-50.000000
75%	541.750000	442.500000	390.750000	235.750000	539.500000	472.750000
max	990.000000	992.000000	999.000000	897.000000	998.000000	989.000000

	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	76.980000	-87.350000	-109.480000	34.120000	11.570000	-30.440000
std	533.468164	559.822399	545.570288	568.993623	583.819006	568.313424
min	-994.000000	-987.000000	-980.000000	-988.000000	-977.000000	-991.000000
25%	-292.250000	-580.500000	-525.250000	-481.000000	-539.750000	-538.000000
50%	91.500000	-163.000000	-139.500000	71.500000	44.000000	0.500000
75%	528.250000	367.250000	348.000000	499.750000	445.250000	426.250000
max	926.000000	955.000000	978.000000	995.000000	985.000000	980.000000

Para poder continuar con el análisis, procederemos a incluir dos nuevas filas al DataFrame.

- 'Ahorro': será la suma de todos los valores >0 de cada mes
- 'Gasto': será la suma de todos los valores <0 de cada mes

[10]:
```
data.loc['Gasto'] = data[data<0].sum()
data.loc['Ahorro'] = data[data>0].sum()
data.tail(2)
```

[10]:

	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio
Gasto	-18162.0	-24398.0	-29690.0	-34133.0	-17200.0	-24197.0	-18992.0
Ahorro	29685.0	24437.0	21721.0	15200.0	27504.0	22720.0	26690.0

	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Gasto	-29013.0	-29151.0	-22957.0	-24180.0	-25861.0
Ahorro	20278.0	18203.0	26369.0	25337.0	22817.0

Ahora tenemos toda la información ordenada para proceder a responder las preguntas del apratado 1 de la asignación:

1. ¿Qué mes se ha gastado más?

[11]:
```
data[data == data.loc['Gasto'].max()].loc['Gasto'].idxmax()
```

[11]:
```
'Mayo'
```

2. ¿Qué mes se ha ahorrado más?

[12]:
```
data[data == data.loc['Ahorro'].max()].loc['Ahorro'].idxmax()
```

[12]:
```
'Enero'
```

3. ¿Cuál es la media de gastos al año?

[13]:
```
'{:+,.2f}'.format(data.loc['Gasto'].mean())
```

[13]:
```
'-24,827.83'
```

4. ¿Cuál ha sido el gasto total a lo largo del año?

[14]:
```
'{:+,.2f}'.format(data.loc['Gasto'].sum())
```

[14]:
```
'-297,934.00'
```

5. ¿Cuáles han sido los ingresos totales a lo largo del año?

[15]:
```
'{:,.2f}'.format(data.loc['Ahorro'].sum())
```

[15]:
```
'280,961.00'
```

6. Gráfica de los ingresos a lo largo del año

Separo la data de Gastos y Ahorro por mes en un DataFrame independiente data2

[16]:
```
data2 = data.loc[['Ahorro','Gasto']]
data2
```

[16]:

	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Ahorro	29685.0	24437.0	21721.0	15200.0	27504.0	22720.0	26690.0	20278.0	18203.0	26369.0	25337.0	22817.0
Gasto	-18162.0	-24398.0	-29690.0	-34133.0	-17200.0	-24197.0	-18992.0	-29013.0	-29151.0	-22957.0	-24180.0	-25861.0

Creamos la gráfica

[17]:
```
fig = px.bar(data2.transpose(),opacity=0.7,title='Gasto y Ahorro por␣
↪mes',text_auto=True)
fig.layout.xaxis.title.text = 'Mes'
fig.layout.yaxis.title.text = 'Valor'
```

Vamos a pasar la gráfica a una imagen e imprimirla en el notebook

[18]:
```
img = fig.to_image(format='jpeg',width=1200,height=600,scale=2)
Image(img)
```

[18]: