Resumir datos y estadística descriptiva

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaño

Sociedad Ecuatoriana de Estadística

Octubre-2020


SOCIEDAD
ECUATORIANA
ESTADÍSTICA

Nota:
Con *Alt* + *F* o *Option* + *F* puede hacer que estas dapositivas ocupen todo el navegador (es decir que se ignore el aspecto de diapositiva que tiene por default la presentación)

Caso a desarrollar

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaño

Ejemplo: Data de transacciones bancarias

El Banco del Pacífico requiere mejorar los tiempos de atención al cliente en ventanilla, para ello ha recolectado esta información anónimamente para cada cajero y transacción realizada.

Le suministran un excel con dos hojas:

1. Tiene los datos de las transacciones, columnas: Sucursal, Cajero, ID_Transaccion, Transaccion, Tiempo_Servicio_seg, Nivel de satisfacción, Monto de la transaccion.
2. Otra hoja que indica si en la sucursal se ha puesto o no el nuevo sistema.

Ejemplo - Caso Banco: Preeliminares

- Abrir Jupyter Notebook y crear un nuevo notebook dentro de la carpeta scripts dentro de la carpeta del proyecto creado anteriormente
- Importar paquetes necesarios
- Cambiar el directorio de trabajo al directorio del proyecto (recuerden que para importar requerimos que la carpeta Data se pueda acceder directamente)

```
import os
import math as mt
import numpy as np
import pandas as pd
import scipy.stats
# os.getcwd() # obtener el directorio de trabajo
# os.chdir('ruta/al/proyecto/') # Definir el directorio de trabajo
```

Ejemplo - Caso Banco: Importar

Importar las hojas de excel que estamos utilizando

```
data_banco_xlsx = pd.read_excel('Data//Data_Banco.xlsx', sheet_name = 'Data')
data_banco_xlsx.head(5)
```

```
##    Sucursal  Cajero  ID_Transaccion  ... Tiempo_Servicio_seg  Satisfaccion    Monto
## 0        62    4820               2  ...               311.0     Muy Bueno   2889,3
## 1        62    4820               2  ...               156.0          Malo  1670,69
## 2        62    4820               2  ...               248.0       Regular  3172,49
## 3        62    4820               2  ...                99.0       Regular  1764.92
## 4        62    4820               2  ...               123.0     Muy Bueno  1835.69
##
## [5 rows x 7 columns]
```

Ejemplo - Caso Banco: Importar

Importar las hojas de excel que estamos utilizando

```
data_sucursal = pd.read_excel('Data//Data_Banco.xlsx', sheet_name = 'Data_Sucursal')
data_sucursal.head(5)
```

```
##    ID_Sucursal       Sucursal Nuevo_Sistema
## 0           62  Riocentro Sur            No
## 1           85         Centro            Si
## 2          267       Alborada            Si
## 3          443   Mall del Sol            Si
## 4          586      Via Daule            No
```

Entender los datos - Ejemplo

Entender los datos y modificar los que tienen mal el tipo de columna.

```
# Ver la estructura del data.frame
data_banco_xlsx.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 24299 entries, 0 to 24298
## Data columns (total 7 columns):
## Sucursal                24299 non-null int64
## Cajero                  24299 non-null int64
## ID_Transaccion          24299 non-null int64
## Transaccion             24299 non-null object
## Tiempo_Servicio_seg     24299 non-null float64
## Satisfaccion            24299 non-null object
## Monto                   24299 non-null object
## dtypes: float64(1), int64(3), object(3)
## memory usage: 1.3+ MB
```

Entender los datos - Ejemplo

Entender los datos y modificar los que tienen mal el tipo de columna.

```
# Ver la estructura del data.frame
data_sucursal.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 5 entries, 0 to 4
## Data columns (total 3 columns):
## ID_Sucursal      5 non-null int64
## Sucursal         5 non-null object
## Nuevo_Sistema    5 non-null object
## dtypes: int64(1), object(2)
## memory usage: 248.0+ bytes
```

Ejemplo - Manipulacion de datos

Lo primero que necesitamos es corregir los tipos de datos, nótese que

- **Monto** tiene una mezcla de "," y "."
- **Sucursal** y **Cajero** deberían ser de tipo character
- **Satisfaccion** debe ser factor ordenado

```
data_banco_xlsx['Monto'].head(4)
```

```
## 0     2889,3
## 1    1670,69
## 2    3172,49
## 3    1764.92
## Name: Monto, dtype: object
```

```
# Modificar la coma por punto en Monto
data_banco_xlsx['Monto']= data_banco_xlsx['Monto'].replace(',','.', regex=True)
data_banco_xlsx["Monto"] = pd.to_numeric(data_banco_xlsx.Monto, errors='coerce')
data_banco_xlsx['Monto'].head(4)
```

```
## 0    2889.30
## 1    1670.69
## 2    3172.49
## 3    1764.92
## Name: Monto, dtype: float64
```

Ejemplo - Manipulacion de datos

Lo primero que necesitamos es corregir los tipos de datos, nótese que

- **Monto** tiene una mezcla de "," y "."
- **Sucursal** y **Cajero** deberían ser de tipo character
- **Satisfaccion** debe ser factor ordenado

```
# Modificar a String
data_banco_xlsx['Sucursal'] = data_banco_xlsx['Sucursal'].astype(str)
data_banco_xlsx['Cajero'] = data_banco_xlsx['Cajero'].astype(str)
data_banco_xlsx['ID_Transaccion'] = data_banco_xlsx['ID_Transaccion'].astype(str)
```

Ejemplo - Manipulacion de datos

Lo primero que necesitamos es corregir los tipos de datos, nótese que

- **Monto** tiene una mezcla de "," y "."
- **Sucursal** y **Cajero** deberían ser de tipo character
- **Satisfaccion** debe ser factor ordenado

```
## Dato Categorical
data_banco_xlsx['Satisfaccion'] = pd.Categorical(
    data_banco_xlsx['Satisfaccion'],
    categories= ['Muy Malo', 'Malo', 'Regular', 'Bueno', 'Muy Bueno'],
    ordered=True)
data_banco_xlsx.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 24299 entries, 0 to 24298
## Data columns (total 7 columns):
## Sucursal               24299 non-null object
## Cajero                 24299 non-null object
## ID_Transaccion         24299 non-null object
## Transaccion            24299 non-null object
## Tiempo_Servicio_seg    24299 non-null float64
## Satisfaccion           24299 non-null category
## Monto                  24299 non-null float64
## dtypes: category(1), float64(2), object(4)
## memory usage: 1.1+ MB
```

Ejemplo - Explorar los datos

Con los datos corregidos, podemos empezar a explorar; para ello podemos seleccionar columnas o filtrar filas pero esto nos deja aún con muchas filas como para poder obtener información, de ahí la utilidad de la estadística descriptiva como herramienta para resumir los datos y empezar a obtener información inicial *"insights"*. Entonces ahora vamos a **aprender a aplicar estadística descriptiva que nos permitirá entender nuestros datos y obtener insights**.

Estadística descriptiva - Estadisticos|Medidas

Introducción al análisis de datos

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaño

Estadística descriptiva - Estadisticos|Medidas

Pandas tiene ya desarrollado algunos de las medidas estadísticas más usadas, pero ademas en python tenemos varios paquetes adicionales como por ejemplo:

- import statistics
- from scipy import stats o import scipy.stats

Medidas de Tendencia Central

Media.- Promedio de los valores

- Se la puede entender como el punto de equilibrio
- Muy sensible a valores aberrantes
- dataframe.mean(x, na.rm= TRUE)

Media Acotada.- Promedio de los valores, pero quitando un porcentaje de valores extremos.

- Es menos sensible a valores aberrantes
- Se puede perder información importante
- scipy.stats.trim_mean(, trim es porcentaje a quitar a cada lado)
- En pandas se puede combinar .clip con .mean pero es trabajoso

Medidas de Tendencia Central

```
# Media del tiempo de servicio
data_banco_xlsx['Tiempo_Servicio_seg'].mean()
```

```
## 155.579993233514
```

```
# Media acotada del Monto
scipy.stats.trim_mean(data_banco_xlsx.Monto , 0.05)
```

```
## 1982.6435558502126
```

Medidas de Tendencia Central

Mediana.- Punto medio de los valores una vez que se han ordenado de menor a mayor o de mayor a menor.

- Valor importante pero poco usado
- No es sensible a valores aberrantes
- En pandas: DataFrame.median()

Media Ponderada.- Promedio de los valores, pero asignando un peso diferente a cada valor.

- Normalmente se utiliza cuando se tiene datos agrupados
- Es también sensible a valores aberrantes
- En numpy: np.average(array=, weights=)

Medidas de Tendencia Central

```
# Mediana del tiempo de servicio
data_banco_xlsx['Tiempo_Servicio_seg'].median()
# Media ponderada
```

```
## 122.45229035353
```

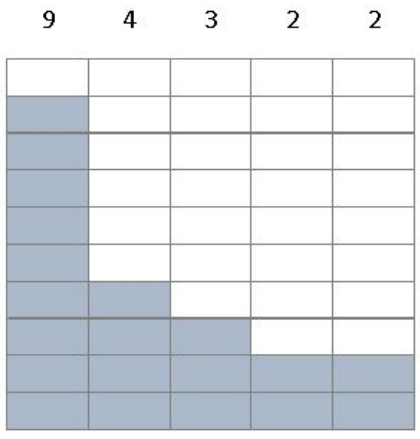
```
x= [10, 11, 10.5, 5] # Ej: Ingreso promedio por provincia
p= [0.7, 0.6, 0.8, 0.79] #Ej: Porcentaje de habitantes por provincia
np.average( x, weights= p)
```

```
## 8.97923875432526
```

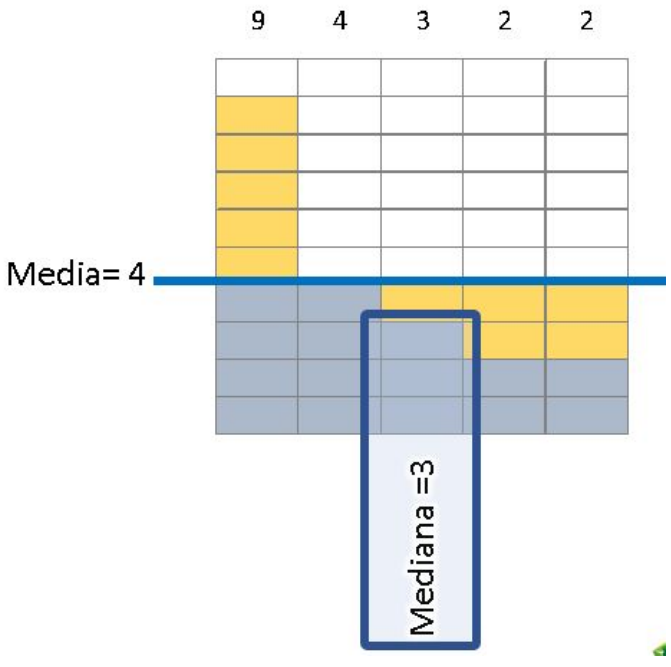
Medidas de Tendencia Central

Entendiendo media vs mediana

- Observaciones

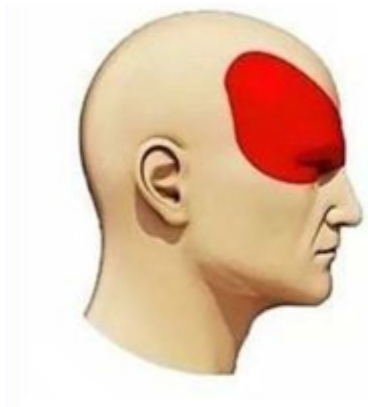


- Media y mediana

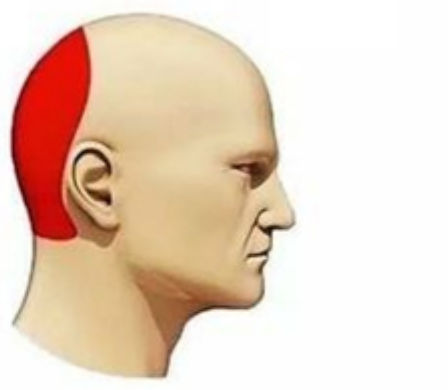


Los dolores de cabeza de un estadístico o científico de datos

Migraña

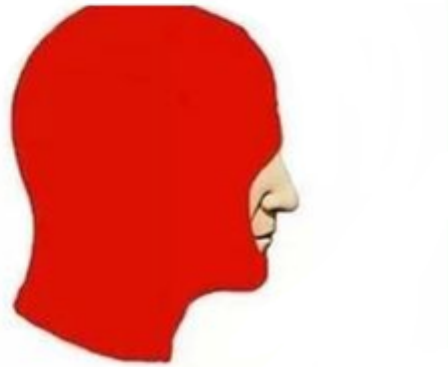


Hipertensión



Estrés

Ver a un "data scientist" obteniendo promedio de una variable categórica



SOCIEDAD ECUATORIANA DE ESTADISTICA

Medidas de Posición

- Min y Max
  - En pandas `dataframe.min .max`
- Cuartiles.- Dividen al conjunto de observaciones en **4** partes iguales
  - El segundo cuartil es la mediana
  - En pandas `dataframe.quantile([0.25, 0.50, 0.75])`
- Deciles.- Dividen al conjunto de observaciones en **10** partes iguales
  - El quinto decil sería igual a la mediana
  - En pandas `dataframe.quantile( np.arange(0, 1.1, 0.1) )`
- Centiles.- Dividen al conjunto de observaciones en **100** partes iguales
  - El quincuagésimo centil es la mediana
  - En pandas `dataframe.quantile([prob])`

Medidas de Posición

Entendiendo los cuartiles

Posición	1	2	3	4	5	6	7	8	9	10	11
Observación	11	25	38	41	57	62	71	79	84	91	99
			Cuartil 1			Cuartil 2			Cuartil 3		

Medidas de Posición

Calcular las medidas de Posición para el tiempo de servicio data de Banco

```
# Minimo y Máximo
data_banco_xlsx.Tiempo_Servicio_seg.min()
```

```
## 18.1317703726497
```

```
data_banco_xlsx.Tiempo_Servicio_seg.max()

# Cuartiles
```

```
## 1602.69831855495
```

```
data_banco_xlsx.Tiempo_Servicio_seg.quantile( [0.25, 0.50, 0.75] )
```

```
## 0.25     75.691187
## 0.50    122.452290
## 0.75    197.730457
## Name: Tiempo_Servicio_seg, dtype: float64
```

Medidas de Posición

Calcular las medidas de Posición para todo el dataframe

```
# Deciles
data_banco_xlsx.quantile( np.arange(0, 1.1, 0.1) )
```

```
##      Tiempo_Servicio_seg     Monto
## 0.0            18.131770    53.820
## 0.1            49.622996   858.604
## 0.2            67.000000  1247.588
## 0.3            84.000000  1576.056
## 0.4           102.000000  1903.782
## 0.5           122.452290  2087.430
## 0.6           146.990131  2225.728
## 0.7           178.834834  2384.100
## 0.8           220.246940  2596.178
## 0.9           298.782570  2946.810
## 1.0          1602.698319  6278.020
```

Medidas de Posición - Boxplot

Boxplot.- Muestra gráficamente las medidas de posición, se puede usar varios paquetes para realizar gráficos tanto estáticos como dinámicos en python, esto no se verá en el presente curso, sin embargo se muestra un ejemplo:

```
import seaborn as sns
# Un primer Boxplot
sns.boxplot(x= "Tiempo_Servicio_seg", data= data_banco_xlsx)
```

Medidas de Dispersión

- Varianza.- Media aritmética de las desviaciones de la media elevadas al cuadrado
  - $s^2 = \frac{\sum(x-\bar{x})^2}{n-1}$
  - En pandas: DataFrame.var()
- Desviación Estándar.- Raíz cuadrada de la varianza.
  - Esta medida se utiliza frecuentemente para realizar comparaciones entre dos conjuntos de datos
  - $s = \sqrt{\frac{\sum(x-\bar{x})^2}{n-1}}$
  - En pandas: DataFrame.std()
- Mediana de las desviaciones absolutas.- Estimador Robusto
  - $mad = mediana[x - mediana(x)]$
  - En pandas: DataFrame.mad()

Medidas de Dispersión

Calcular las medidas de dispersión

```
# Desviacion estándar del tiempo de servicio en segundo
data_banco_xlsx.Tiempo_Servicio_seg.std()
# Media de las desviaciones absolutas del tiempo de servicio en segundo
```

```
## 120.00945744360449
```

```
data_banco_xlsx.Tiempo_Servicio_seg.mad()
# MAD para todos las columnas
```

```
## 84.76386877314536
```

```
data_banco_xlsx.mad()
```

```
## Tiempo_Servicio_seg     84.763869
## Monto                  639.526455
## dtype: float64
```

Y ahora, todo junto

Con .describe() se obtinen algunas estadísticas descriptivas

```
data_banco_xlsx.describe()
```

```
##        Tiempo_Servicio_seg         Monto
## count         24299.000000  24299.000000
## mean            155.579993   1996.156149
## std             120.009457    816.146998
## min              18.131770     53.820000
## 25%              75.691187   1417.730000
## 50%             122.452290   2087.430000
## 75%             197.730457   2482.090000
## max            1602.698319   6278.020000
```

Y ahora, todo junto

Con .describe() se obtinen algunas estadísticas descriptivas

```
data_banco_xlsx.describe( percentiles= [0.1, 0.9] )
```

```
##        Tiempo_Servicio_seg         Monto
## count         24299.000000  24299.000000
## mean            155.579993   1996.156149
## std             120.009457    816.146998
## min              18.131770     53.820000
## 10%              49.622996    858.604000
## 50%             122.452290   2087.430000
## 90%             298.782570   2946.810000
## max            1602.698319   6278.020000
```

Pandas - Descriptivas

- Comando describe() presenta varias estadísticas descriptivas, como summary en R.
- Existen funciones como min(), max(), mode(), median(), mean(), std(), corr(), count(), rank()
- Existen funciones como mean() que con el parámetro 1 permite obtiener media por filas
- Más información en https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#computations-descriptive-stats

Resumir datos agrupando por una o más variables

type: sub-section

Cálculos con agrupamiento

Pandas permite obtener resúmenes o cálculos agrupando por los valores de una variable del dataFrame, como summarise + group_by en R o un Select, from, group by en SQL.

```
## Obtener las descriptivas del Monto por Transaccion
data_banco_xlsx[['Tiempo_Servicio_seg', 'Transaccion']].groupby('Transaccion').describe()
```

```
##                                Tiempo_Servicio_seg  ...
##                                              count  ...          max
## Transaccion                                         ...
## Cobrar cheque (Cta del Bco)                 5407.0  ...   913.116263
## Cobro/Pago (Cta externa)                    3005.0  ...  1602.698319
## Deposito                                   15887.0  ...   594.796607
##
## [3 rows x 8 columns]
```

Cálculos con agrupamiento

Pandas permite obtener resúmenes o cálculos agrupando por los valores de una variable del dataFrame, como summarise + group_by en R o un Select, from, group by en SQL.

```
## Obtiene las descriptivas por Transaccion
data_banco_xlsx.groupby('Transaccion').describe()
```

```
##                                Tiempo_Servicio_seg                ...      Monto
##                                              count         mean  ...        75%       max
## Transaccion                                                       ...
## Cobrar cheque (Cta del Bco)                 5407.0   185.865204  ...   2557.935   6229.91
## Cobro/Pago (Cta externa)                    3005.0   301.428249  ...   2945.040   6278.02
## Deposito                                   15887.0   117.685731  ...   2376.730   5849.85
##
## [3 rows x 16 columns]
```

Cálculos con agrupamiento

Pandas permite obtener resúmenes o cálculos agrupando por los valores de una variable del dataFrame, como summarise + group_by en R o un Select, from, group by en SQL.

```
## Obtiene las Media del Tiempo y Monto por Transaccion
data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto', 'Transaccion']].groupby('Transaccion').mean()
```

```
##                                Tiempo_Servicio_seg        Monto
## Transaccion
## Cobrar cheque (Cta del Bco)             185.865204  2079.749432
## Cobro/Pago (Cta externa)                301.428249  2448.715328
## Deposito                                117.685731  1882.105087
```

Cálculos con agrupamiento

Tambien soporta agrupar por varias columnas así como varios cálculos

```
## Obtiene las Media del Tiempo y Monto por Sucursal y Nivel de Satisfaccion
data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto', 'Sucursal',
  'Satisfaccion']].groupby(['Sucursal', 'Satisfaccion']).describe()
```

```
##                       Tiempo_Servicio_seg                ...      Monto
##                                     count        mean    ...        75%      max
## Sucursal Satisfaccion                                    ...
## 267      Muy Malo                   539.0  178.288403    ...  2531.0250  5701.04
##          Malo                       848.0  176.860435    ...  2528.1000  5299.05
##          Regular                    642.0  175.813945    ...  2472.5250  5308.24
##          Bueno                      707.0  182.631086    ...  2588.9950  5401.83
##          Muy Bueno                  593.0  202.188738    ...  2614.7200  5259.18
## 443      Muy Malo                   461.0  150.047746    ...  2462.9100  4323.11
##          Malo                       673.0  162.763379    ...  2521.6600  5591.55
##          Regular                    823.0  192.666638    ...  2615.5600  6044.42
##          Bueno                     1044.0  185.257154    ...  2573.7175  6278.02
##          Muy Bueno                 1189.0  191.550564    ...  2596.3600  6082.55
## 586      Muy Malo                   335.0   80.352239    ...  2278.8750  3804.37
##          Malo                       381.0   77.351706    ...  2185.3300  4649.41
##          Regular                    345.0   80.347826    ...  2293.0000  3433.63
```

Cálculos con agrupamiento

Tambien soporta agrupar por varias columnas así como varios cálculos

```
## Obtiene las Media del Tiempo y Monto por Sucursal y Nivel de Satisfaccion
data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto', 'Sucursal',
  'Satisfaccion']].groupby(['Sucursal', 'Satisfaccion']).mean()
```

```
##                          Tiempo_Servicio_seg        Monto
## Sucursal Satisfaccion
## 267      Muy Malo                 178.288403  2070.285083
##          Malo                     176.860435  2058.706568
##          Regular                  175.813945  1994.600732
##          Bueno                    182.631086  2077.691598
##          Muy Bueno                202.188738  2122.173794
## 443      Muy Malo                 150.047746  1969.573623
##          Malo                     162.763379  2026.256820
##          Regular                  192.666638  2121.600182
##          Bueno                    185.257154  2102.173927
##          Muy Bueno                191.550564  2101.123532
## 586      Muy Malo                  80.352239  1734.234866
##          Malo                      77.351706  1670.242178
##          Regular                   80.347826  1734.118435
##          Bueno                     82.795337  1705.907176
```

Cálculos con agrupamiento

Tambien soporta agrupar por varias columnas así como varios cálculos

```
## Obtiene las Media y Mediana del
## Tiempo y Monto por Sucursal y Nivel de Satisfaccion
## Se usa .agg()
data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto', 'Sucursal',
  'Satisfaccion']].groupby(['Sucursal', 'Satisfaccion']).agg(["mean", "median"])
```

```
##                       Tiempo_Servicio_seg                    Monto
##                                      mean      median         mean    median
## Sucursal Satisfaccion
## 267      Muy Malo                178.288403  149.466417  2070.285083  2108.840
##          Malo                    176.860435  148.075930  2058.706568  2143.855
##          Regular                 175.813945  142.179657  1994.600732  2095.580
##          Bueno                   182.631086  145.554547  2077.691598  2163.910
##          Muy Bueno               202.188738  159.871332  2122.173794  2194.420
## 443      Muy Malo                150.047746  129.687657  1969.573623  2073.580
##          Malo                    162.763379  132.723704  2026.256820  2083.710
##          Regular                 192.666638  156.922623  2121.600182  2179.240
##          Bueno                   185.257154  145.259379  2102.173927  2170.020
##          Muy Bueno               191.550564  150.487781  2101.123532  2170.550
## 586      Muy Malo                 80.352239   72.000000  1734.234866  1854.810
```

Cálculos con agrupamiento

Si queremos que el resultado siga siendo un dataframe usamos .reset_index()

```
## Obtiene las Media y Mediana del
## Tiempo y Monto por Sucursal y Nivel de Satisfaccion
## Se usa .agg()
data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto', 'Sucursal',
  'Satisfaccion']].groupby(['Sucursal', 'Satisfaccion']).agg(["mean",
  "median"]).reset_index()
```

```
##    Sucursal Satisfaccion Tiempo_Servicio_seg                    Monto
##                                        mean      median          mean    median
## 0       267     Muy Malo          178.288403  149.466417  2070.285083  2108.840
## 1       267         Malo          176.860435  148.075930  2058.706568  2143.855
## 2       267      Regular          175.813945  142.179657  1994.600732  2095.580
## 3       267        Bueno          182.631086  145.554547  2077.691598  2163.910
## 4       267    Muy Bueno          202.188738  159.871332  2122.173794  2194.420
## 5       443     Muy Malo          150.047746  129.687657  1969.573623  2073.580
## 6       443         Malo          162.763379  132.723704  2026.256820  2083.710
## 7       443      Regular          192.666638  156.922623  2121.600182  2179.240
## 8       443        Bueno          185.257154  145.259379  2102.173927  2170.020
## 9       443    Muy Bueno          191.550564  150.487781  2101.123532  2170.550
## 10      586     Muy Malo           80.352239   72.000000  1734.234866  1854.810
```

Cálculos con agrupamiento

Si queremos que el resultado siga siendo un dataframe usamos .reset_index()

```
## Obtiene las Media y Mediana del
## Tiempo y Monto por Sucursal y Nivel de Satisfaccion
## Se usa .agg()
data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto', 'Sucursal',
  'Satisfaccion']].groupby(['Sucursal', 'Satisfaccion']).agg(["mean",
  "median"]).reset_index().info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 25 entries, 0 to 24
## Data columns (total 6 columns):
## (Sucursal, )                    25 non-null object
## (Satisfaccion, )                25 non-null category
## (Tiempo_Servicio_seg, mean)     25 non-null float64
## (Tiempo_Servicio_seg, median)   25 non-null float64
## (Monto, mean)                   25 non-null float64
## (Monto, median)                 25 non-null float64
## dtypes: category(1), float64(4), object(1)
## memory usage: 1.3+ KB
```

Cálculos con agrupamiento

Si queremos que el resultado siga siendo un dataframe usamos .reset_index()

```
## Obtiene las Media y Mediana del Tiempo y Monto por Sucursal y Satisfaccion
## Dejar las Sucursales+satifaccion cuyo promedio sea mayor a 100 segundos
res_bySucSatis= data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto',
  'Sucursal', 'Satisfaccion']].groupby(['Sucursal', 'Satisfaccion']).agg(["mean",
  "median"]).reset_index()

res_bySucSatis.loc[ res_bySucSatis[('Tiempo_Servicio_seg', 'mean')] > 100, ]
```

```
##    Sucursal Satisfaccion Tiempo_Servicio_seg                  Monto
##                                         mean      median         mean    median
## 0       267     Muy Malo           178.288403  149.466417  2070.285083  2108.840
## 1       267         Malo           176.860435  148.075930  2058.706568  2143.855
## 2       267      Regular           175.813945  142.179657  1994.600732  2095.580
## 3       267        Bueno           182.631086  145.554547  2077.691598  2163.910
## 4       267    Muy Bueno           202.188738  159.871332  2122.173794  2194.420
## 5       443     Muy Malo           150.047746  129.687657  1969.573623  2073.580
## 6       443         Malo           162.763379  132.723704  2026.256820  2083.710
## 7       443      Regular           192.666638  156.922623  2121.600182  2179.240
## 8       443        Bueno           185.257154  145.259379  2102.173927  2170.020
## 9       443    Muy Bueno           191.550564  150.487781  2101.123532  2170.550
```

Cálculos con agrupamiento

Finalmente, se puede usar .apply() para los cáclulos a realizar.

```
## Funcion que controla los calculos
def DESCR(x):
    return pd.Series({
            'Monto_Media': x.Monto.mean(),
            'Monto_Mediana': x.Monto.median(),
            'Tiempo_Media': x.Tiempo_Servicio_seg.mean(),
            'Tiempo_Mediana': x.Tiempo_Servicio_seg.median()})
## Probar la función
DESCR(data_banco_xlsx)
```

```
## Monto_Media       1996.156149
## Monto_Mediana     2087.430000
## Tiempo_Media       155.579993
## Tiempo_Mediana     122.452290
## dtype: float64
```

Cálculos con agrupamiento

Finalmente, se puede usar .apply() para los cáclulos a realizar.

```
## Ya teniendo la funcion DESCR creada
## Usamos la función en .apply
data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto', 'Sucursal']].groupby(
  'Sucursal').apply(DESCR)
```

```
##           Monto_Media  Monto_Mediana  Tiempo_Media  Tiempo_Mediana
## Sucursal
## 267       2063.555879        2144.200    182.627140      148.095875
## 443       2078.908518        2139.390    181.011574      144.276091
## 586       1719.796059        1815.595     82.334563       71.000000
## 62        1758.289824        1851.145     89.392530       76.000000
## 85        2048.338975        2121.110    166.395467      135.590069
```

Cálculos con agrupamiento

El mismo resultado se puede obtener con los llamados: pd.NamedAgg

```
## Una tupla que controla el nombre: (Columna, Funcion_a_aplicar)
aggregation = {
    'Monto_Media': ('Monto','mean'),
    'Monto_Mediana': ('Monto','median'),
    'Tiempo_Media': ('Tiempo_Servicio_seg','mean'),
    'Tiempo_Mediana' : ('Tiempo_Servicio_seg', 'median')
}

## Usando .agg
data_banco_xlsx[['Tiempo_Servicio_seg', 'Monto', 'Sucursal']].groupby(
  'Sucursal').agg(**aggregation)
```

```
##           Monto_Media  Monto_Mediana  Tiempo_Media  Tiempo_Mediana
## Sucursal
## 267       2063.555879        2144.200    182.627140      148.095875
## 443       2078.908518        2139.390    181.011574      144.276091
## 586       1719.796059        1815.595     82.334563       71.000000
## 62        1758.289824        1851.145     89.392530       76.000000
## 85        2048.338975        2121.110    166.395467      135.590069
```

Tablas de frecuencia

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaño

Frecuencia en variable numérica

En python para hacer una tabla de frecuencias se usa la función .value_counts

```
## Frecuencias del tiempo de servicio en segundos
data_banco_xlsx['Tiempo_Servicio_seg'].value_counts(bins=7).reset_index(
  name='Frecuencia')
```

```
##                    index  Frecuencia
## 0     (16.546, 244.498]       20430
## 1    (244.498, 470.865]        3207
## 2    (470.865, 697.232]         516
## 3    (697.232, 923.598]         122
## 4   (923.598, 1149.965]          17
## 5  (1149.965, 1376.332]           6
## 6  (1376.332, 1602.698]           1
```

Frecuencia en variable numérica

En python para hacer una tabla de frecuencias se usa la función .value_counts
Para completar la tabla podemos usar las funciones que ya sabemos

```
## Frecuencias del tiempo de servicio en segundos
frec_Tiempo= data_banco_xlsx['Tiempo_Servicio_seg'].value_counts(
  bins=7).reset_index(name='Frecuencia')
frec_Tiempo['Frec_Acumulada'] = frec_Tiempo.Frecuencia.cumsum()
frec_Tiempo['Frec_Relativa_Acumulada'] = 100*( frec_Tiempo.Frec_Acumulada /
  frec_Tiempo.Frecuencia.sum() ).round(4)
frec_Tiempo
```

```
##                    index  Frecuencia  Frec_Acumulada  Frec_Relativa_Acumulada
## 0     (16.546, 244.498]       20430           20430                    84.08
## 1    (244.498, 470.865]        3207           23637                    97.28
## 2    (470.865, 697.232]         516           24153                    99.40
## 3    (697.232, 923.598]         122           24275                    99.90
## 4   (923.598, 1149.965]          17           24292                    99.97
## 5  (1149.965, 1376.332]           6           24298                   100.00
## 6  (1376.332, 1602.698]           1           24299                   100.00
```

Frecuencia en variable numérica

Si queremos intervalos personalizados podemos definir los quiebres y hacer la tabla de frecuencia en dos pasos, primero crear una vairable con los intervalos y luego obtener las frecuencias

```
limites = np.arange(0, 600, 60)
limites = np.append(limites, 1610)
data_banco_xlsx['Tiempo_intervalo'] = pd.cut(x=data_banco_xlsx[
  'Tiempo_Servicio_seg'], bins= limites )
data_banco_xlsx['Tiempo_intervalo'].value_counts()
```

```
## (60, 120]      8037
## (120, 180]     5201
## (0, 60]        3861
## (180, 240]     3140
## (240, 300]     1661
## (300, 360]      933
## (360, 420]      550
## (540, 1610]     397
## (420, 480]      296
## (480, 540]      223
## Name: Tiempo_intervalo, dtype: int64
```

Frecuencia en variable numérica

En python para hacer una tabla de frecuencias se usa la función .value_counts
Para completar la tabla podemos usar las funciones que ya sabemos

```
## Frecuencias del tiempo de servicio en segundos
limites = np.arange(0, 600, 60)
limites = np.append(limites, 1610)
data_banco_xlsx['Tiempo_intervalo'] = pd.cut(x=data_banco_xlsx[
  'Tiempo_Servicio_seg'], bins= limites )
frec_Tiempo= data_banco_xlsx['Tiempo_intervalo'].value_counts().reset_index(
  name='Frecuencia')
frec_Tiempo= data_banco_xlsx['Tiempo_Servicio_seg'].value_counts(bins=7). \
  reset_index(name='Frecuencia')
frec_Tiempo['Frec_Acumulada'] = frec_Tiempo.Frecuencia.cumsum()
frec_Tiempo['Frec_Relativa_Acumulada'] = 100*( frec_Tiempo.Frec_Acumulada / \
  frec_Tiempo.Frecuencia.sum() ).round(4)
## Notar que \ al final de la linea le dice a Python que la linea continúa abajo
```

Frecuencia en variable numérica

En python para hacer una tabla de frecuencias se usa la función .value_counts
Para completar la tabla podemos usar las funciones que ya sabemos

```
frec_Tiempo
```

```
##                    index  Frecuencia  Frec_Acumulada  Frec_Relativa_Acumulada
## 0     (16.546, 244.498]       20430           20430                    84.08
## 1    (244.498, 470.865]        3207           23637                    97.28
## 2    (470.865, 697.232]         516           24153                    99.40
## 3    (697.232, 923.598]         122           24275                    99.90
## 4   (923.598, 1149.965]          17           24292                    99.97
## 5  (1149.965, 1376.332]           6           24298                   100.00
## 6  (1376.332, 1602.698]           1           24299                   100.00
```

Frecuencia en variable categórica

Para las variables categóricas también se usa la función .value_counts() s

```
## Tabla de Frecuencias del Nivel de satisfaccion
frec_Tiempo= data_banco_xlsx['Satisfaccion'].value_counts().reset_index(
  name='Frecuencia')
frec_Tiempo['Frec_Acumulada'] = frec_Tiempo.Frecuencia.cumsum()
frec_Tiempo['Frec_Relativa'] = 100*( frec_Tiempo.Frecuencia / frec_Tiempo.Frecuencia.
  sum() ).round(4)
frec_Tiempo
```

```
##        index  Frecuencia  Frec_Acumulada  Frec_Relativa
## 0  Muy Bueno        6262            6262          25.77
## 1      Bueno        5915           12177          24.34
## 2    Regular        4639           16816          19.09
## 3       Malo        4474           21290          18.41
## 4   Muy Malo        3009           24299          12.38
```

Frecuencia en variable categórica

Para las variables categóricas también se usa la función .value_counts()

```
## Tabla de Frecuencias del Nivel de satisfaccion
pd.crosstab(index= data_banco_xlsx['Satisfaccion'],
                              columns="Frecuencia")
```

```
## col_0          Frecuencia
## Satisfaccion
## Muy Malo             3009
## Malo                 4474
## Regular              4639
## Bueno                5915
## Muy Bueno            6262
```

Tabla cruzada: 2 Var Categ

Tablas cruzadas entre Sucursales y Nivel de satisfaccion

```
## Tabla cruzada entre Sucursal
satisf_vs_suc = pd.crosstab(index= data_banco_xlsx['Sucursal'],
                            columns= data_banco_xlsx['Satisfaccion'])
satisf_vs_suc
```

```
## Satisfaccion  Muy Malo  Malo  Regular  Bueno  Muy Bueno
## Sucursal
## 267                539   848      642    707        593
## 443                461   673      823   1044       1189
## 586                335   381      345    386        451
## 62                 431   604      520    684        599
## 85                1243  1968     2309   3094       3430
```

Tabla cruzada: 2 Var Categ

Tablas cruzadas entre Sucursales y Nivel de satisfaccion

```
## Tabla cruzada entre Sucursal
satisf_vs_suc = pd.crosstab(index= data_banco_xlsx['Sucursal'],
                            columns= data_banco_xlsx['Satisfaccion'],
                            margins= True)
satisf_vs_suc
```

```
## Satisfaccion  Muy Malo  Malo  Regular  Bueno  Muy Bueno    All
## Sucursal
## 267                539   848      642    707        593   3329
## 443                461   673      823   1044       1189   4190
## 586                335   381      345    386        451   1898
## 62                 431   604      520    684        599   2838
## 85                1243  1968     2309   3094       3430  12044
## All               3009  4474     4639   5915       6262  24299
```

```
nombresCol = np.append( data_banco_xlsx.Satisfaccion.unique(), 'totalSatisf')
nombresFila = np.append( data_banco_xlsx.Sucursal.unique(), 'totalSucur')
satisf_vs_suc.columns = nombresCol
satisf_vs_suc.index= nombresFila
satisf_vs_suc
```

Tabla cruzada: 2 Var Categ

Tablas cruzadas entre Sucursales y Nivel de satisfaccion

```
## Tabla cruzada entre Sucursal
(satisf_vs_suc/satisf_vs_suc.loc["totalSucur", "totalSatisf"]).round(4) * 100
```

```
##              Muy Bueno   Malo  Regular  Bueno  Muy Malo  totalSatisf
## 62              2.22    3.49     2.64   2.91      2.44        13.70
## 85              1.90    2.77     3.39   4.30      4.89        17.24
## 267             1.38    1.57     1.42   1.59      1.86         7.81
## 443             1.77    2.49     2.14   2.81      2.47        11.68
## 586             5.12    8.10     9.50  12.73     14.12        49.57
## totalSucur     12.38   18.41    19.09  24.34     25.77       100.00
```

Tabla cruzada: 2 Var Categ

Tablas cruzadas entre Sucursales y Nivel de satisfaccion

```
## Tabla cruzada entre Sucursal
satisf_vs_suc.div( satisf_vs_suc.loc[:,"totalSatisf"], axis=0).round(4) * 100
```

```
##              Muy Bueno   Malo  Regular  Bueno  Muy Malo  totalSatisf
## 62             16.19  25.47    19.29  21.24     17.81         100.0
## 85             11.00  16.06    19.64  24.92     28.38         100.0
## 267            17.65  20.07    18.18  20.34     23.76         100.0
## 443            15.19  21.28    18.32  24.10     21.11         100.0
## 586            10.32  16.34    19.17  25.69     28.48         100.0
## totalSucur     12.38  18.41    19.09  24.34     25.77         100.0
```

Explorar graficamente dos variables

Una forma de explorar gráficamente

```
## importar los comandos de plotnine
from plotnine import *
(ggplot(data_banco_xlsx, aes(x= 'Transaccion', fill= 'Satisfaccion')) +
  geom_bar( position = "fill" )  +
  labs(title= 'Grafico del % de Nivel de satisfaccion por Transaccion', y= "Porcentaje")
)
```

Mas...

Pandas tiene muchas funciones para hacer estadística descriptiva:

- Comando describe() presenta varias estadísticas descriptivas, como summary en R.
- Existen funciones como min(), max(), mode(), median(), mean(), std(), corr(), count(), rank()
- Existen funciones como mean() que con el parámetro 1 permite obtiener media por filas
- Más información en https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#computations-descriptive-stats

Fin

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaño