CLOUD TECHNOLOGY TRAINING

MANUAL PYTHON ANALISIS DE DATOS

Contenido

1. Introducción a Pandas y Jupyter Notebook

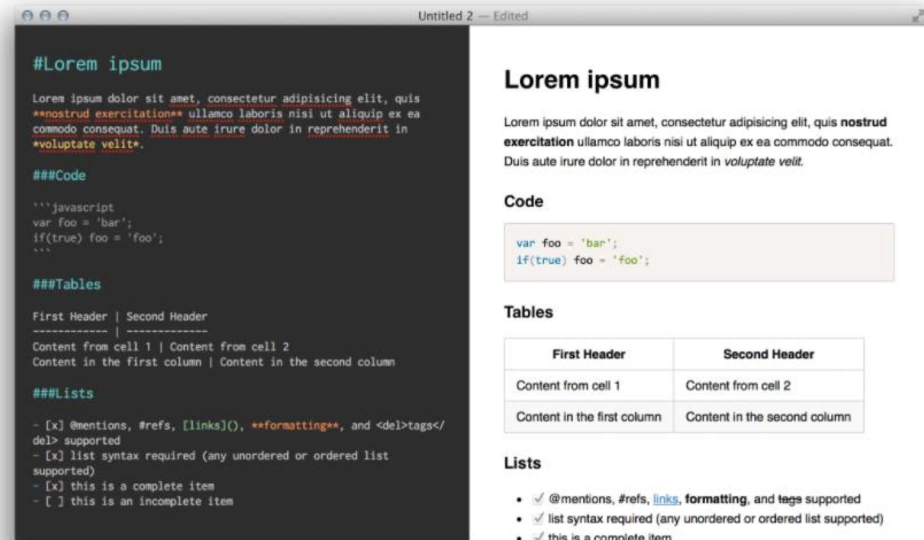
https://www.kaggle.com/faviovaz/telco-churn-prediction-with-h2o-and-datatable

Introducción a Jupyter Notebook

Markdown

Link: https://markdown.es/

Markdown es un lenguaje estándar que permite renderizar documentos. Se utiliza en los README.md para documentación.



Practicar

Titulo

Subtitulo

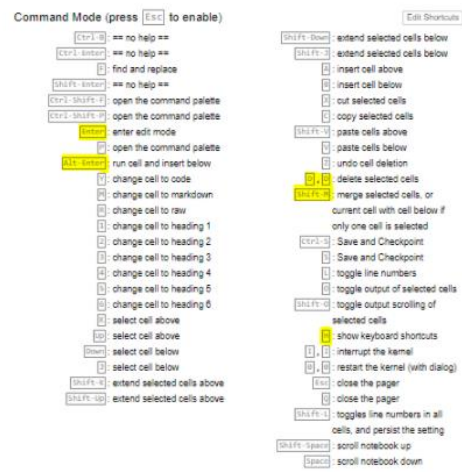
```
variable = "Hola"
print(variable + "mundo")
```

Uno	Dos
1	2

In [ ]:

```

```

Shorcut Keys


Command Mode (press Esc to enable)

Edit Shortcuts

Ctrl-B : == no help ==
Ctrl-Enter : == no help ==
F : find and replace
Shift-Enter : == no help ==
Ctrl-Shift-F : open the command palette
Ctrl-Shift-P : open the command palette
Enter : enter edit mode
P : open the command palette
Alt-Enter : run cell and insert below
Y : change cell to code
M : change cell to markdown
R : change cell to raw
1 : change cell to heading 1
2 : change cell to heading 2
3 : change cell to heading 3
4 : change cell to heading 4
5 : change cell to heading 5
6 : change cell to heading 6
K : select cell above
Up : select cell above
Down : select cell below
J : select cell below
Shift-K : extend selected cells above
Shift-Up : extend selected cells above
Shift-Down : extend selected cells below
Shift-J : extend selected cells below
A : insert cell above
B : insert cell below
X : cut selected cells
C : copy selected cells
Shift-V : paste cells above
V : paste cells below
Z : undo cell deletion
D,D : delete selected cells
Shift-M : merge selected cells, or current cell with cell below if only one cell is selected
Ctrl-S : Save and Checkpoint
S : Save and Checkpoint
L : toggle line numbers
O : toggle output of selected cells
Shift-O : toggle output scrolling of selected cells
H : show keyboard shortcuts
I,I : interrupt the kernel
0,0 : restart the kernel (with dialog)
Esc : close the pager
Q : close the pager
Shift-L : toggles line numbers in all cells, and persist the setting
Shift-Space : scroll notebook up
Space : scroll notebook down

Python

In [1]:

```
2+2
```

Out[1]: 4

In [2]:

```
print("hola mundo")
```

hola mundo

In [3]:

```
import os
import csv
```

In [4]:

```
video = input("Qué pelicula o show estás buscando? ")
```

Qué pelicula o show estás buscando? Breaking Bad

In [5]:

```
csvpath = os.path.join("..", "data", "01netflix.csv")

found = False

```

In [6]:

```
with open(csvpath, newline="") as csvfile:
    csvreader = csv.reader(csvfile, delimiter=",")

    for row in csvreader:
        if row[0] == video:
            print(row[0] + " tiene un rating de " + row[5])

            found = True

            break

    if found is False:
        print("No pude encontrar el show que buscas!")
```

Breaking Bad tiene un rating de 97

Introducción a Pandas

En Computación y Ciencia de datos, pandas es una biblioteca de software escrita como extensión de NumPy para manipulación y análisis de datos para el lenguaje de programación Python. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales. Es un software libre distribuido bajo la licencia BSD versión tres cláusulas.1 El nombre deriva del término "datos de panel", término de econometría que designa datos que combinan una dimensión temporal con otra dimensión transversal.2

In [7]:
```python
import pandas as pd
```

1 Dimensión: Panda Series

https://pandas.pydata.org/docs/reference/api/pandas.Series.html

In [8]:
```python
data_series = pd.Series(["UNAM", "IPN", "UAM","TEC", "ITAM"])
data_series
```

Out[8]:
```
0    UNAM
1     IPN
2     UAM
3     TEC
4    ITAM
dtype: object
```

2 Dimensiones: Data Frame

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

In [9]:
```python
states_dicts = [{"Estado": "Ciudad de Mexico", "Abreviación": "CDMX"},
                {"Estado": "Aguascalientes", "Abreviación": "AGS"}]

states_df = pd.DataFrame(states_dicts)
states_df
```

Out[9]:

	Estado	Abreviación
0	Ciudad de Mexico	CDMX
1	Aguascalientes	AGS

In [10]:
```python
states_df2 = pd.DataFrame(
    {"Estado": ["Ciudad de México", "Aguascalientes"],
     "Abreviación": ["CDMX", "AGS"]
     }
)
states_df2
```

Métodos de los data frames

In [11]:
```
datos = pd.read_csv("../data/01netflix.csv", encoding="UTF-8")
```

In [12]:
```
datos
```

Out[12]:

	title	rating	ratingLevel	ratingDescription	release year	user rating score	user rating size
0	White Chicks	PG-13	crude and sexual humor, language and some drug...	80	2004	82.0	80
1	Lucky Number Slevin	R	strong violence, sexual content and adult lang...	100	2006	NaN	82
2	Grey's Anatomy	TV-14	Parents strongly cautioned. May be unsuitable ...	90	2016	98.0	80
3	Prison Break	TV-14	Parents strongly cautioned. May be unsuitable ...	90	2008	98.0	80
4	How I Met Your Mother	TV-PG	Parental guidance suggested. May not be suitab...	70	2014	94.0	80
...	...	...	...	...	...	...	...
994	Pup Star	G	General Audiences. Suitable for all ages.	35	2016	NaN	82
995	The BFG	PG	for action/peril, some scary moments and brief...	60	2016	97.0	80
996	The Secret Life of Pets	PG	for action and some rude humor	60	2016	NaN	81
997	Precious Puppies	TV-G	Suitable for all ages.	35	2003	NaN	82
998	Beary Tales	TV-G	Suitable for all ages.	35	2013	NaN	82

999 rows × 7 columns

In [13]:
```
datos.describe()
```

Out[13]:

	ratingDescription	release year	user rating score	user rating size
count	999.000000	999.000000	604.000000	999.000000
mean	67.398398	2010.329329	84.100993	80.783784
std	30.783969	8.880562	12.353476	0.973238
min	10.000000	1940.000000	55.000000	80.000000
25%	35.000000	2007.000000	74.750000	80.000000
50%	60.000000	2015.000000	88.000000	80.000000
75%	90.000000	2016.000000	95.000000	82.000000
max	124.000000	2017.000000	99.000000	82.000000

In [14]:
```
datos.head()
```
Out[14]:

	title	rating	ratingLevel	ratingDescription	release year	user rating score	user rating size
0	White Chicks	PG-13	crude and sexual humor, language and some drug...	80	2004	82.0	80
1	Lucky Number Slevin	R	strong violence, sexual content and adult lang...	100	2006	NaN	82
2	Grey's Anatomy	TV-14	Parents strongly cautioned. May be unsuitable ...	90	2016	98.0	80
3	Prison Break	TV-14	Parents strongly cautioned. May be unsuitable ...	90	2008	98.0	80
4	How I Met Your Mother	TV-PG	Parental guidance suggested. May not be suitab...	70	2014	94.0	80

In [15]:
```
datos.tail()
```
Out[15]:

	title	rating	ratingLevel	ratingDescription	release year	user rating score	user rating size
994	Pup Star	G	General Audiences. Suitable for all ages.	35	2016	NaN	82
995	The BFG	PG	for action/peril, some scary moments and brief...	60	2016	97.0	80
996	The Secret Life of Pets	PG	for action and some rude humor	60	2016	NaN	81
997	Precious Puppies	TV-G	Suitable for all ages.	35	2003	NaN	82
998	Beary Tales	TV-G	Suitable for all ages.	35	2013	NaN	82

In [16]:
```
datos["user rating score"]
```
Out[16]:
```
0      82.0
1       NaN
2      98.0
3      98.0
4      94.0
       ...
994     NaN
995    97.0
996     NaN
997     NaN
998     NaN
Name: user rating score, Length: 999, dtype: float64
```

In [17]:
```
datos["user rating score"].mean()
```
Out[17]: `84.10099337748345`

In [18]:
```
datos["user rating score"].sum()
```
Out[18]: `50797.0`

In [19]:
```
datos["user rating score"].head()
```
Out[19]:
```
0    82.0
1     NaN
2    98.0
3    98.0
4    94.0
Name: user rating score, dtype: float64
```

In [20]:
```
datos["user rating score"].tail()
```
Out[20]:
```
994     NaN
995    97.0
996     NaN
997     NaN
998     NaN
Name: user rating score, dtype: float64
```

In [21]:
```
datos["rating"].unique()
```
Out[21]:
```
array(['PG-13', 'R', 'TV-14', 'TV-PG', 'TV-MA', 'TV-Y', 'NR', 'TV-Y7-FV',
       'UR', 'PG', 'TV-G', 'G', 'TV-Y7'], dtype=object)
```

In [22]:
```
datos["rating"].value_counts()
```
Out[22]:
```
TV-14       234
PG          170
TV-MA       148
G           137
TV-Y         68
TV-PG        59
TV-G         52
TV-Y7-FV     44
TV-Y7        38
R            19
PG-13        15
NR           14
UR            1
Name: rating, dtype: int64
```

In [23]:
```
datos["rating redondeado"] = round(datos["user rating score"]/10,0)
```

In [24]:
```
datos["rating redondeado"]
```

Out[24]:
```
0       8.0
1       NaN
2      10.0
3      10.0
4       9.0
       ... 
994     NaN
995    10.0
996     NaN
997     NaN
998     NaN
Name: rating redondeado, Length: 999, dtype: float64
```

In [25]:
```
datos["rating redondeado"].value_counts()
```

Out[25]:
```
9.0     171
10.0    154
8.0     128
7.0      81
6.0      70
Name: rating redondeado, dtype: int64
```

In [26]:
```
datos["rating redondeado"].mean()
```

Out[26]:
```
8.427152317880795
```

Podemos ver que hay un problema con la columna de "Species". vamos a corregirlo de dos formas diferentes: la primera es una solución mas "Python". La segunda es una solución más "Pandas"

In [29]:
```
resultado = datos["Species"].value_counts()
```

In [30]:
```
datos["Species"].unique()
```

Out[30]:
```
array(['Human', 'Half-Human/Half-Giant', 'Werewolf',
       'Human\xa0(Werewolf\xa0traits)', 'Human(goblin ancestry)', 'Ghost',
       nan, 'Centaur', 'Human\xa0', 'Human\xa0(Metamorphmagus)'],
      dtype=object)
```

Primera solucion:

In [31]:
```
resultado = []

for x in datos["Species"]:
    try:
        nuevo = x.replace("\xa0","")
        resultado.append(nuevo)
    except:
        print(x)

resultado = pd.Series(resultado).value_counts()
```

```
nan
nan
nan
nan
nan
nan
```

In [32]:
```
resultado
```

Out[32]:
```
Human                     117
Ghost                       6
Half-Human/Half-Giant       2
Werewolf                    2
Human(goblin ancestry)      1
Human(Metamorphmagus)       1
Human(Werewolftraits)       1
Centaur                     1
dtype: int64
```

Segunda solución:

In [33]:
```
resultado = datos["Species"].str.replace("\xa0","").value_counts()
resultado
```

Out[33]:
```
Human                       117
Ghost                         6
Half-Human/Half-Giant         2
Werewolf                      2
Human(goblin ancestry)        1
Human(Metamorphmagus)         1
Human(Werewolftraits)         1
Centaur                       1
Name: Species, dtype: int64
```

In [34]:
```
pd.DataFrame(resultado)
```

Out[34]:

	Species
Human	117
Ghost	6
Half-Human/Half-Giant	2
Werewolf	2
Human(goblin ancestry)	1
Human(Metamorphmagus)	1
Human(Werewolftraits)	1
Centaur	1

In [35]:
```
resultado = pd.DataFrame(resultado)
```

In [36]:
```
resultado = resultado.reset_index()
```

In [37]:
```
resultado
```

Out[37]:

	index	Species
0	Human	117
1	Ghost	6
2	Half-Human/Half-Giant	2
3	Werewolf	2
4	Human(goblin ancestry)	1
5	Human(Metamorphmagus)	1
6	Human(Werewolftraits)	1

In [38]:
```
resultado = resultado.rename(columns={"index":"Especie","Species":"Conteo"})
```

In [39]:
```
resultado
```

Out[39]:

	Especie	Conteo
0	Human	117
1	Ghost	6
2	Half-Human/Half-Giant	2
3	Werewolf	2
4	Human(goblin ancestry)	1
5	Human(Metamorphmagus)	1
6	Human(Werewolftraits)	1
7	Centaur	1

In [40]:
```
resultado.to_csv("../resultados/01_species.csv", index=False)
```

In [41]:
```
resultado.to_excel("../resultados/01_species.xlsx",index=False)
```

2. Transformación de Datos

Pandas

Operaciones sobre Data Frames

loc & iloc

Dataset: Audi used car listings

In [1]:
```
import pandas as pd
```

In [2]:
```
datos = pd.read_csv("../data/03audi.csv")
```

In [3]:
```
datos.columns
```

Out[3]:
```
Index(['model', 'year', 'price', 'transmission', 'mileage', 'fuelType', 'tax',
       'mpg', 'engineSize'],
      dtype='object')
```

In [4]:
```
datos.dtypes
```

Out[4]:
```
model            object
year              int64
price             int64
transmission     object
mileage           int64
fuelType         object
tax               int64
mpg             float64
engineSize      float64
dtype: object
```

In [5]:
```
datos["year"].dtype
```

Out[5]:
```
dtype('int64')
```

In [6]:
```
datos["year"].astype("str")
```

Out[6]:
```
0        2017
1        2016
2        2016
3        2017
4        2019
         ...
10663    2020
10664    2020
10665    2020
10666    2017
10667    2016
Name: year, Length: 10668, dtype: object
```

In [7]:
```
datos.shape
```

Out[7]:
```
(10668, 9)
```

In [8]:
```
datos.head()
```

Out[8]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0

In [9]:
```
datos.tail()
```

Out[9]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
10663	A3	2020	16999	Manual	4018	Petrol	145	49.6	1.0
10664	A3	2020	16999	Manual	1978	Petrol	150	49.6	1.0
10665	A3	2020	17199	Manual	609	Petrol	150	49.6	1.0
10666	Q3	2017	19499	Automatic	8646	Petrol	150	47.9	1.4
10667	Q3	2016	15999	Manual	11855	Petrol	150	47.9	1.4

In [10]:
```
datos.describe()
```
Out[10]:

	year	price	mileage	tax	mpg	engineSize
count	10668.000000	10668.000000	10668.000000	10668.000000	10668.000000	10668.000000
mean	2017.100675	22896.685039	24827.244001	126.011436	50.770022	1.930709
std	2.167494	11714.841888	23505.257205	67.170294	12.949782	0.602957
min	1997.000000	1490.000000	1.000000	0.000000	18.900000	0.000000
25%	2016.000000	15130.750000	5968.750000	125.000000	40.900000	1.500000
50%	2017.000000	20200.000000	19000.000000	145.000000	49.600000	2.000000
75%	2019.000000	27990.000000	36464.500000	145.000000	58.900000	2.000000
max	2020.000000	145000.000000	323000.000000	580.000000	188.300000	6.300000

In [11]:
```
datos.count()
```
Out[11]:
```
model           10668
year            10668
price           10668
transmission    10668
mileage         10668
fuelType        10668
tax             10668
mpg             10668
engineSize      10668
dtype: int64
```

Loc

Busqueda por nombre de index

In [12]:
```
datos_index = datos.set_index("model")
```

In [13]:
```
datos
```
Out[13]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0
...	...	...	...	...	...	...	...	...	...
10663	A3	2020	16999	Manual	4018	Petrol	145	49.6	1.0
10664	A3	2020	16999	Manual	1978	Petrol	150	49.6	1.0
10665	A3	2020	17199	Manual	609	Petrol	150	49.6	1.0
10666	Q3	2017	19499	Automatic	8646	Petrol	150	47.9	1.4
10667	Q3	2016	15999	Manual	11855	Petrol	150	47.9	1.4

In [14]:
```
datos_index.loc["A3","price"]
```
Out[14]:
```
model
A3     17300
A3     10200
A3     16100
A3     16400
A3     14500
       ...
A3     14995
A3     12695
A3     16999
A3     16999
A3     17199
Name: price, Length: 1929, dtype: int64
```

In [15]:
```
datos_index.loc["A3","price"].mean()
```
Out[15]: 17408.522032141005

In [16]:
```
datos.loc[0:5,["model","price"]]
```

Out[16]:

	model	price
0	A1	12500
1	A6	16500
2	A1	11000
3	A4	16800
4	A3	17300
5	A1	13900

In [ ]:

iloc

Busqueda por posición

In [17]:
```
datos_index.iloc[0:5, 0:3]
```

Out[17]:

model	year	price	transmission
A1	2017	12500	Manual
A6	2016	16500	Automatic
A1	2016	11000	Manual
A4	2017	16800	Automatic
A3	2019	17300	Manual

In [18]:
```
datos.iloc[0:5, 0:4]
```

Out[18]:

	model	year	price	transmission
0	A1	2017	12500	Manual
1	A6	2016	16500	Automatic
2	A1	2016	11000	Manual
3	A4	2017	16800	Automatic
4	A3	2019	17300	Manual

In [19]:
```
datos["model"].unique()
```

Out[19]:
```
array(['A1', 'A6', 'A4', 'A3', 'Q3', 'Q5', 'A5', 'S4', 'Q2', 'A7', 'TT',
       'Q7', 'RS6', 'RS3', 'A8', 'Q8', 'RS4', 'RS5', 'R8', 'SQ5', 'S8',
       'SQ7', 'S3', 'S5', 'A2', 'RS7'], dtype=object)
```

In [ ]:

Filtrar con booleanos

In [21]:
```
filtro = datos["model"]=="A1"
```

In [22]:
```
filtro
```

Out[22]:
```
0         True
1        False
2         True
3        False
4        False
         ...
10663    False
10664    False
10665    False
10666    False
10667    False
Name: model, Length: 10668, dtype: bool
```

In [23]:
```
datos[filtro]
```

Out[23]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
5	A1	2016	13900	Automatic	32260	Petrol	30	58.9	1.4
9	A1	2016	12000	Manual	22451	Petrol	30	55.4	1.4
27	A1	2018	15800	Manual	10793	Petrol	145	56.5	1.4
...	...	...	...	...	...	...	...	...	...
10632	A1	2010	9990	Automatic	38000	Petrol	125	53.3	1.4
10636	A1	2013	9291	Manual	29382	Petrol	125	53.3	1.4
10645	A1	2016	10999	Manual	22150	Diesel	0	76.3	1.6
10646	A1	2016	12380	Manual	40119	Petrol	30	55.4	1.4
10652	A1	2014	9995	Manual	54000	Petrol	30	55.4	1.2

1347 rows × 9 columns

Limpieza de datos

Dataset: Credit Card Transactions

In [24]:
```
banco = pd.read_csv("../data/04credit_card.csv")
```

In [25]:
```
banco.shape
```

Out[25]:
```
(500000, 15)
```

In [26]:
```
banco.count()
```

Out[26]:
```
User              500000
Card              500000
Year              500000
Month             500000
Day               500000
Time              500000
Amount            500000
Use Chip          500000
Merchant Name     500000
Merchant City     500000
Merchant State    444661
Zip               441410
MCC               500000
Errors?             7901
Is Fraud?         500000
dtype: int64
```

In [27]:
```
pd.isna(banco["Zip"])
```

Out[27]:
```
0         False
1          True
2         False
3          True
4         False
          ...
499995     True
499996     True
499997    False
499998    False
499999    False
Name: Zip, Length: 500000, dtype: bool
```

In [28]:
```
pd.isna(banco["Zip"]).value_counts()
```

Out[28]:
```
False    441410
True      58590
Name: Zip, dtype: int64
```

In [29]:
```
filtro = pd.isna(banco["Zip"])
```

In [30]:
```
zip_error = banco[filtro]
```

In [31]:
```
zip_error.shape
```

Out[31]:
```
(58590, 15)
```

In [32]:
```
zip_error.to_excel("../resultados/02errores_zip.xlsx")
```

Actividad

Crea el subconjunto de todos los registros que tienen un error (utiliza la columna "Error")

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Retener casos completos

In [33]:
```
banco_casos_completos = banco.dropna(how='any')
```

In [34]:
```
banco_casos_completos.count()
```

Out[34]:
```
User              6628
Card              6628
Year              6628
Month             6628
Day               6628
Time              6628
Amount            6628
Use Chip          6628
Merchant Name     6628
Merchant City     6628
Merchant State    6628
Zip               6628
MCC               6628
Errors?           6628
Is Fraud?         6628
dtype: int64
```

In [35]:
```
banco_casos_completos["Errors?"].value_counts()
```

Out[35]:
```
Insufficient Balance                        4534
Bad PIN                                     1185
Technical Glitch                             856
Bad Zipcode                                   32
Bad PIN,Insufficient Balance                  11
Insufficient Balance,Technical Glitch          9
Bad Zipcode,Technical Glitch                   1
Name: Errors?, dtype: int64
```

In [36]:
```
banco_casos_completos["Errors?"] = banco_casos_completos["Errors?"].replace({"Technical Glitch": "IT"})
banco_casos_completos["Errors?"].value_counts()
```

```
<ipython-input-36-6930d8b22564>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  banco_casos_completos["Errors?"] = banco_casos_completos["Errors?"].replace({"Technical Glitch": "IT"})
```

Out[36]:
```
Insufficient Balance                        4534
Bad PIN                                     1185
IT                                           856
Bad Zipcode                                   32
Bad PIN,Insufficient Balance                  11
Insufficient Balance,Technical Glitch          9
Bad Zipcode,Technical Glitch                   1
Name: Errors?, dtype: int64
```

In [37]:
```
banco_casos_completos.loc[:,"Errors?"] = banco_casos_completos["Errors?"].replace({"Technical Glitch": "IT"})
```

```
C:\Users\manyv\anaconda3\lib\site-packages\pandas\core\indexing.py:1676: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

In [38]:
```
banco_casos_completos["Errors?"].value_counts()
```

Out[38]:
```
Insufficient Balance                        4534
Bad PIN                                     1185
IT                                           856
Bad Zipcode                                   32
Bad PIN,Insufficient Balance                  11
Insufficient Balance,Technical Glitch          9
Bad Zipcode,Technical Glitch                   1
Name: Errors?, dtype: int64
```

Group By

In [39]:
```
datos = pd.read_csv("../data/03audi.csv")
```

In [40]:
```
datos.head()
```

Out[40]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0

In [41]:
```
datos.groupby(["model"])
```

Out[41]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000021BBAE2D2E0>

In [42]:
```
datos.groupby(["model"]).sum()
```

Out[42]:

	year	price	mileage	tax	mpg	engineSize
model						
A1	2716396	19299480	32999194	103950	79034.2	1837.2
A2	2003	2490	100000	30	65.7	1.4
A3	3890240	33581039	55730907	183995	109866.4	3166.4
A4	2785473	27972777	41003257	152330	75071.7	2780.7
A5	1779271	20795015	20740491	122975	44448.6	1830.5
A6	1508480	16976148	26184182	86140	40647.5	1661.0
A7	246047	3521593	3352675	19350	5742.8	360.0
A8	238029	4127858	2154311	20430	5186.0	364.9
Q2	1659144	18508954	8875805	117060	39327.1	1251.7
Q3	2858557	32589954	30033132	205380	65325.7	2522.9
Q5	1769279	26700869	19573112	150755	37039.1	1927.0

In [43]:
```
datos.groupby(["model"])[["price","mileage"]].mean().head()
```

Out[43]:

	price	mileage
model		
A1	14327.750557	24498.288048
A2	2490.000000	100000.000000
A3	17408.522032	28891.087092
A4	20255.450398	29690.989862
A5	23577.114512	23515.295918

In [44]:
```
datos.groupby(["model"])[["price","mileage"]].count().head()
```

Out[44]:

	price	mileage
model		
A1	1347	1347
A2	1	1
A3	1929	1929
A4	1381	1381
A5	882	882

In [45]:
```
datos_group = datos.groupby(["model"])
datos_group.agg(Precio_Promedio_USD=("price","mean"), Recuento= ("price","count"), Promedio_Mileage=("mileage","mean"))
```

Out[45]:

model	Precio_Promedio_USD	Recuento	Promedio_Mileage
A1	14327.750557	1347	24498.288048
A2	2490.000000	1	100000.000000
A3	17408.522032	1929	28891.087092
A4	20255.450398	1381	29690.989862
A5	23577.114512	882	23515.295918
A6	22695.385027	748	35005.590909
A7	28865.516393	122	27480.942623
A8	34981.847458	118	18256.872881
Q2	22516.975669	822	10797.816302
Q3	22999.261821	1417	21194.870854
Q5	30445.688712	877	22318.257697
Q7	44788.319899	397	21672.755668
Q8	60115.014493	69	6412.043478
R8	97652.214286	28	12363.000000
RS3	34050.515152	33	25870.545455
RS4	50151.612903	31	21743.806452
RS5	51265.206897	29	11572.758621
RS6	55963.871795	39	28524.641026
RS7	33490.000000	1	56000.000000
S3	20379.444444	18	40500.722222
S4	31248.083333	12	27117.666667
S5	15980.000000	3	53606.666667
S8	33807.750000	4	29441.000000
SQ5	31415.812500	16	42114.875000
SQ7	49269.000000	8	27659.375000
TT	21784.452381	336	28146.770833

In [46]:
```
datos_group = datos.groupby(["model","year"])
datos_group.agg(Precio_Promedio_USD=("price","mean"), Recuento= ("price","count"), Promedio_Mileage=("mileage","mean"))
```

Out[46]:

model	year	Precio_Promedio_USD	Recuento	Promedio_Mileage
A1	2010	9990.000000	1	38000.000000
	2011	6302.000000	5	78740.000000
	2012	8090.761905	21	52973.571429
	2013	8745.982759	58	49486.879310
	2014	10060.084746	59	44822.237288
...	...	...	...	...
TT	2016	18701.784091	88	31269.840909
	2017	20238.145455	55	29097.927273
	2018	25589.600000	10	16954.800000
	2019	33647.954545	66	4907.393939
	2020	35232.739130	23	2419.565217

221 rows × 3 columns

In [47]:

```
datos_group = datos.groupby(["model","year"])
datos_group.agg(Precio_Promedio_USD=("price","mean"), Recuento= ("price","count"), Promedio_Mileage=("mileage","mean")).rese
```

Out[47]:

	model	year	Precio_Promedio_USD	Recuento	Promedio_Mileage
0	A1	2010	9990.000000	1	38000.000000
1	A1	2011	6302.000000	5	78740.000000
2	A1	2012	8090.761905	21	52973.571429
3	A1	2013	8745.982759	58	49486.879310
4	A1	2014	10060.084746	59	44822.237288
...	...	...	...	...	...
216	TT	2016	18701.784091	88	31269.840909
217	TT	2017	20238.145455	55	29097.927273
218	TT	2018	25589.600000	10	16954.800000
219	TT	2019	33647.954545	66	4907.393939
220	TT	2020	35232.739130	23	2419.565217

Ordenar

In [54]:

```
banco_reporte.sort_values("Errors",ascending=False)
```

Out[54]:

	Merchant State	Merchant City	Amount	Operaciones	Errors
9688	TX	Houston	43.799342	4986	80
9567	TX	Dallas	32.667568	2776	64
1752	FL	Orlando	45.808221	2496	56
880	CA	Los Angeles	34.311310	3725	54
1057	CA	Riverside	46.576667	417	53
...	...	...	...	...	...
4175	MA	Wrentham	53.503846	13	0
4176	MA	Yarmouth Port	32.425000	10	0
4177	MD	Aberdeen	50.212400	100	0
4178	MD	Abingdon	30.837742	31	0
11134	WY	Torrington	-28.560000	2	0

3. Pandas avanzado

Merge (Join)

Dataset: S&P 500 Stock Prices

In [1]:
```python
import pandas as pd
```

In [2]:
```python
clientes = {
    "customer_id": [112, 403, 999, 543, 123],
    "name": ["John", "Kelly", "Sam", "April", "Bobbo"],
    "email": ["jman@gmail", "kelly@aol.com", "sports@school.edu", "April@yahoo.com", "HeyImBobbo@msn.com"]
}
clientes = pd.DataFrame(clientes, columns=["customer_id", "name", "email"])
clientes
```

Out[2]:

	customer_id	name	email
0	112	John	jman@gmail
1	403	Kelly	kelly@aol.com
2	999	Sam	sports@school.edu
3	543	April	April@yahoo.com
4	123	Bobbo	HeyImBobbo@msn.com

In [3]:
```python
compras = {
    "customer_id": [403, 112, 543, 999, 654],
    "item": ["soda", "chips", "TV", "Laptop", "Cooler"],
    "cost": [3.00, 4.50, 600, 900, 150]
}
compras = pd.DataFrame(compras, columns=[
                        "customer_id", "item", "cost"])
compras
```

Out[3]:

	customer_id	item	cost
0	403	soda	3.0
1	112	chips	4.5
2	543	TV	600.0
3	999	Laptop	900.0
4	654	Cooler	150.0

In [4]:
```python
# inner es el default
merge_df = pd.merge(clientes, compras, on="customer_id")
merge_df
```

Out[4]:

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0

In [5]:
```python
merge_df = pd.merge(clientes, compras, on="customer_id", how="outer")
merge_df
```

Out[5]:

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0
4	123	Bobbo	HeyImBobbo@msn.com	NaN	NaN
5	654	NaN	NaN	Cooler	150.0

In [6]:
```python
merge_df = pd.merge(clientes, compras, on="customer_id", how="left")
merge_df
```

Out[6]:

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0
4	123	Bobbo	HeyImBobbo@msn.com	NaN	NaN

In [7]:
```
merge_df = pd.merge(clientes, compras, on="customer_id", how="right")
merge_df
```

Out[7]:

	customer_id	name	email	item	cost
0	403	Kelly	kelly@aol.com	soda	3.0
1	112	John	jman@gmail	chips	4.5
2	543	April	April@yahoo.com	TV	600.0
3	999	Sam	sports@school.edu	Laptop	900.0
4	654	NaN	NaN	Cooler	150.0

Modern Pandas

Apply vs Assign con funciones Lambda

In [14]:
```
banco = pd.read_csv("../data/04credit_card.csv")
```

In [15]:
```
banco.assign(Fraud= 1)
```

Out[15]:

	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?	Fraud
0	777	2	2014	10	16	11:11	$88.00	Swipe Transaction	-1.288080e+18	San Francisco	CA	94131.0	5499	NaN	No	1
1	55	3	2007	8	26	10:17	$42.22	Online Transaction	-2.088490e+18	ONLINE	NaN	NaN	4784	NaN	No	1
2	931	2	2012	6	1	13:22	$112.21	Swipe Transaction	-5.162040e+18	Thompsons Station	TN	37179.0	5541	NaN	No	1
3	1537	4	2018	1	29	13:44	$56.35	Online Transaction	3.155110e+16	ONLINE	NaN	NaN	4784	NaN	No	1
4	39	0	2014	5	4	19:09	$100.00	Swipe Transaction	-4.282470e+18	West Boylston	MA	1583.0	4829	NaN	No	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
499995	1722	1	2013	2	23	10:36	$1,034.82	Online Transaction	3.694720e+18	ONLINE	NaN	NaN	4722	NaN	No	1
499996	139	0	2017	6	27	08:09	$81.85	Online Transaction	4.241340e+18	ONLINE	NaN	NaN	4814	NaN	No	1
499997	1358	0	2019	12	21	16:40	$79.00	Chip Transaction	1.799190e+18	New Castle	IN	47362.0	5499	NaN	No	1
499998	1869	0	2018	10	19	10:34	$62.07	Chip Transaction	1.913480e+18	Vancouver	WA	98661.0	5300	NaN	No	1
499999	1212	5	2016	8	14	16:48	$44.83	Swipe Transaction	4.052400e+18	Mesquite	TX	75150.0	7538	NaN	No	1

500000 rows × 16 columns

In [16]:
```
banco["Is Fraud?"].value_counts()
```

Out[16]:
```
No     499387
Yes       613
Name: Is Fraud?, dtype: int64
```

In [17]:
```
1 if True else 2
```
Out[17]: 1

In [18]:
```
banco.apply(lambda x: x["Is Fraud?"]=="Yes", axis=1).head()
```
Out[18]:
```
0    False
1    False
2    False
3    False
4    False
dtype: bool
```

In [19]:
```
banco.apply(lambda x: x["Is Fraud?"]=="Yes", axis=1).value_counts()
```
Out[19]:
```
False    499387
True        613
dtype: int64
```

In [20]:
```
%%time
banco.apply(lambda x: 1 if x["Is Fraud?"]=="Yes" else None, axis=1).value_counts()
```
```
Wall time: 3.91 s
```
Out[20]:
```
1.0    613
dtype: int64
```

In [21]:
```
%%time
banco.assign(Fraud= lambda x: [1 if j=="Yes" else None for j in x["Is Fraud?"]]).head()
```
```
Wall time: 216 ms
```
Out[21]:

	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?	Fraud
0	777	2	2014	10	16	11:11	$88.00	Swipe Transaction	-1.288080e+18	San Francisco	CA	94131.0	5499	NaN	No	NaN
1	55	3	2007	8	26	10:17	$42.22	Online Transaction	-2.088490e+18	ONLINE	NaN	NaN	4784	NaN	No	NaN
2	931	2	2012	6	1	13:22	$112.21	Swipe Transaction	-5.162040e+18	Thompsons Station	TN	37179.0	5541	NaN	No	NaN
3	1537	4	2018	1	29	13:44	$56.35	Online Transaction	3.155110e+16	ONLINE	NaN	NaN	4784	NaN	No	NaN
4	39	0	2014	5	4	19:09	$100.00	Swipe Transaction	-4.282470e+18	West Boylston	MA	1583.0	4829	NaN	No	NaN

In [22]:
```
1 banco.assign(Fraud= lambda x: [1 if j=="Yes" else None for j in x["Is Fraud?"]], axis=1)
```

Out[22]:

	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?	Fraud	axis
0	777	2	2014	10	16	11:11	$88.00	Swipe Transaction	-1.288080e+18	San Francisco	CA	94131.0	5499	NaN	No	NaN	1
1	55	3	2007	8	26	10:17	$42.22	Online Transaction	-2.088490e+18	ONLINE	NaN	NaN	4784	NaN	No	NaN	1
2	931	2	2012	6	1	13:22	$112.21	Swipe Transaction	-5.162040e+18	Thompsons Station	TN	37179.0	5541	NaN	No	NaN	1
3	1537	4	2018	1	29	13:44	$56.35	Online Transaction	3.155110e+16	ONLINE	NaN	NaN	4784	NaN	No	NaN	1
4	39	0	2014	5	4	19:09	$100.00	Swipe Transaction	-4.282470e+18	West Boylston	MA	1583.0	4829	NaN	No	NaN	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
499995	1722	1	2013	2	23	10:36	$1,034.82	Online Transaction	3.694720e+18	ONLINE	NaN	NaN	4722	NaN	No	NaN	1
499996	139	0	2017	6	27	08:09	$81.85	Online Transaction	4.241340e+18	ONLINE	NaN	NaN	4814	NaN	No	NaN	1
499997	1358	0	2019	12	21	16:40	$79.00	Chip Transaction	1.799190e+18	New Castle	IN	47362.0	5499	NaN	No	NaN	1
499998	1869	0	2018	10	19	10:34	$62.07	Chip Transaction	1.913480e+18	Vancouver	WA	98661.0	5300	NaN	No	NaN	1
499999	1212	5	2016	8	14	16:48	$44.83	Swipe Transaction	4.052400e+18	Mesquite	TX	75150.0	7538	NaN	No	NaN	1

500000 rows × 17 columns

In [23]:
```
1 banco.assign(Fraud= lambda x: [1 if j=="Yes" else None for j in x["Is Fraud?"]]).count()
```

Out[23]:
```
User              500000
Card              500000
Year              500000
Month             500000
Day               500000
Time              500000
Amount            500000
Use Chip          500000
Merchant Name     500000
Merchant City     500000
Merchant State    444661
Zip               441410
MCC               500000
Errors?             7901
Is Fraud?         500000
Fraud                613
dtype: int64
```

In [24]:

```
banco = pd.read_csv("../data/04credit_card.csv")
```

In [25]:

```
resultado = (
    banco
    .assign(Amount = lambda x: (
            x["Amount"]
            .str.replace("$","", regex=False)
            .str.replace(",","", regex=False)
            .astype("double")
        )
    )
    .assign(Fraud = lambda x: [1 if j=="Yes" else None for j in x["Is Fraud?"]])
    .groupby(["Merchant State","Merchant City"])
    .agg(
        Amount      = ("Amount" , "mean"),
        Operaciones = ("Amount" ,"count"),
        Errors      = ("Errors?","count"),
        Fraud      = ("Fraud","count")
    )
    .reset_index()
    .sort_values("Fraud",ascending=False)
    .assign(Pct_Errors = lambda x: round(x["Errors"] / x["Operaciones"] *100,1))
    .assign(Pct_Fraud = lambda x: round(x["Fraud"] / x["Operaciones"] *100,1))
)

resultado
```

Out[25]:

	Merchant State	Merchant City	Amount	Operaciones	Errors	Fraud	Pct_Errors	Pct_Fraud
3312	Italy	Rome	70.876591	176	7	97	4.0	55.1
540	Algeria	Algiers	131.558000	15	1	14	6.7	93.3
2186	Haiti	Port au Prince	71.598182	11	0	9	0.0	81.8
7798	OH	Strasburg	44.802778	18	1	5	5.6	27.8
596	CA	Berkeley	50.771807	83	1	3	1.2	3.6
...	...	...	...	...	...	...	...	...
3736	LA	Donaldsonville	25.980571	105	3	0	2.9	0.0
3737	LA	Downsville	17.950000	3	0	0	0.0	0.0
3738	LA	Dubach	36.568889	9	0	0	0.0	0.0
3739	LA	Dubberly	403.000000	1	0	0	0.0	0.0
11134	WY	Torrington	-28.560000	2	0	0	0.0	0.0

11135 rows × 8 columns

In [26]:

```
# banco["Merchant State"]=="Italy" & banco["Merchant City"]=="Rome"
```

In [27]:

```
banco[(banco["Merchant State"]=="Italy") & (banco["Merchant City"]=="Rome")].count()
```

Out[27]:

```
User              176
Card              176
Year              176
Month             176
Day               176
Time              176
Amount            176
Use Chip          176
Merchant Name     176
Merchant City     176
Merchant State    176
Zip                 0
MCC               176
Errors?             7
Is Fraud?         176
dtype: int64
```

In [28]:

```
banco[(banco["Merchant State"]=="Italy") & (banco["Merchant City"]=="Rome")]
```

Out[28]:

	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
1441	1656	0	2018	8	2	14:43	$138.17	Chip Transaction	1.715300e+18	Rome	Italy	NaN	3722	NaN	Yes
1678	331	1	2018	10	24	17:52	$177.60	Chip Transaction	6.051400e+18	Rome	Italy	NaN	5310	NaN	Yes
2035	1064	3	2005	11	22	16:40	$2.38	Swipe Transaction	-6.571010e+18	Rome	Italy	NaN	5499	NaN	No
6256	546	1	2018	11	16	11:27	$391.73	Chip Transaction	-8.804510e+18	Rome	Italy	NaN	5712	NaN	Yes
6869	3	3	2017	3	18	13:18	-$78.00	Chip Transaction	-5.162040e+18	Rome	Italy	NaN	5541	NaN	No
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
494023	531	2	2019	7	18	12:34	$241.00	Chip Transaction	4.834900e+17	Rome	Italy	NaN	3504	NaN	Yes
494134	604	1	2017	12	31	14:06	$1.38	Chip Transaction	-7.146670e+18	Rome	Italy	NaN	5970	Bad PIN	Yes
496457	1247	2	2017	6	21	05:50	$62.54	Chip Transaction	7.069580e+18	Rome	Italy	NaN	5812	NaN	No
499681	3	1	2017	11	19	19:30	$2.52	Chip Transaction	8.181290e+18	Rome	Italy	NaN	5661	NaN	Yes
499940	1249	0	2019	3	14	16:36	$54.44	Chip Transaction	-5.162040e+18	Rome	Italy	NaN	5541	NaN	No

Presentar resultados

Map

In [29]:
```
resultado["Amount"].map("${:.2f}".format)
```

Out[29]:
```
3312      $70.88
540      $131.56
2186      $71.60
7798      $44.80
596       $50.77
          ...
3736      $25.98
3737      $17.95
3738      $36.57
3739     $403.00
11134    $-28.56
Name: Amount, Length: 11135, dtype: object
```

In [30]:
```
(
    resultado
    .assign(Amount= lambda x: x["Amount"].map("${:.2f}".format))
    .assign(Pct_Errors= lambda x: x["Pct_Errors"].map("{:.1f}%".format))
    .assign(Pct_Fraud= lambda x: x["Pct_Fraud"].map("{:.1f}%".format))
    .reset_index(drop=True)
)
```

Out[30]:

	Merchant State	Merchant City	Amount	Operaciones	Errors	Fraud	Pct_Errors	Pct_Fraud
0	Italy	Rome	$70.88	176	7	97	4.0%	55.1%
1	Algeria	Algiers	$131.56	15	1	14	6.7%	93.3%
2	Haiti	Port au Prince	$71.60	11	0	9	0.0%	81.8%
3	OH	Strasburg	$44.80	18	1	5	5.6%	27.8%
4	CA	Berkeley	$50.77	83	1	3	1.2%	3.6%
...	...	...	...	...	...	...	...	...
11130	LA	Donaldsonville	$25.98	105	3	0	2.9%	0.0%
11131	LA	Downsville	$17.95	3	0	0	0.0%	0.0%
11132	LA	Dubach	$36.57	9	0	0	0.0%	0.0%
11133	LA	Dubberly	$403.00	1	0	0	0.0%	0.0%
11134	WY	Torrington	$-28.56	2	0	0	0.0%	0.0%

11135 rows × 8 columns

Solución completa

In [31]:
```python
resultado = (
    banco
    .assign(Amount = lambda x: (
            x["Amount"]
            .str.replace("$","", regex=False)
            .str.replace(",","", regex=False)
            .astype("double")
        )
    )
    .assign(Fraud = lambda x: [1 if j=="Yes" else None for j in x["Is Fraud?"]])
    .groupby(["Merchant State","Merchant City"])
    .agg(
        Amount      = ("Amount" , "mean"),
        Operaciones = ("Amount" ,"count"),
        Errors      = ("Errors?","count"),
        Fraud      = ("Fraud","count")
    )
    .reset_index()
    .sort_values("Fraud",ascending=False)
    .assign(Pct_Errors = lambda x: round(x["Errors"] / x["Operaciones"] *100,1))
    .assign(Pct_Fraud = lambda x: round(x["Fraud"] / x["Operaciones"] *100,1))
    .assign(Amount= lambda x: x["Amount"].map("${:.2f}".format))
    .assign(Pct_Errors= lambda x: x["Pct_Errors"].map("{:.1f}%".format))
    .assign(Pct_Fraud= lambda x: x["Pct_Fraud"].map("{:.1f}%".format))
    .reset_index(drop=True)
)

resultado
```

Out[31]:

	Merchant State	Merchant City	Amount	Operaciones	Errors	Fraud	Pct_Errors	Pct_Fraud
0	Italy	Rome	$70.88	176	7	97	4.0%	55.1%
1	Algeria	Algiers	$131.56	15	1	14	6.7%	93.3%
2	Haiti	Port au Prince	$71.60	11	0	9	0.0%	81.8%
3	OH	Strasburg	$44.80	18	1	5	5.6%	27.8%
4	CA	Berkeley	$50.77	83	1	3	1.2%	3.6%
...	...	...	...	...	...	...	...	...
11130	LA	Donaldsonville	$25.98	105	3	0	2.9%	0.0%
11131	LA	Downsville	$17.95	3	0	0	0.0%	0.0%
11132	LA	Dubach	$36.57	9	0	0	0.0%	0.0%
11133	LA	Dubberly	$403.00	1	0	0	0.0%	0.0%
11134	WY	Torrington	$-28.56	2	0	0	0.0%	0.0%

In [32]:
```python
resultado.to_excel("../resultados/analisis_banco.xlsx", index=False)
```

4. Introducción a Matplotlib

Matplotlib

Introducción

In [1]:
```python
import pandas as pd
```

In [2]:
```python
import matplotlib.pyplot as plt
```

In [3]:
```python
import numpy as np
```

In [4]:
```python
import random
```

In [5]:
```python
x_axis = list(range(0,50,1))
```

In [6]:
```python
e_x = [np.exp(x/10) for x in x_axis]
```

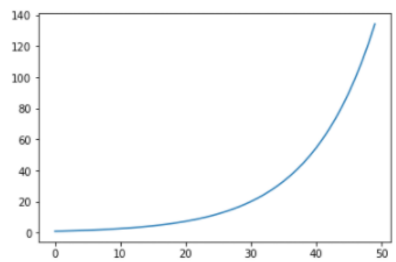
In [7]:
```python
type(x_axis)
```

Out[7]: list

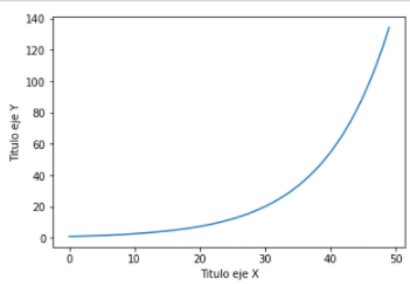
In [8]:
```python
type(e_x)
```

Out[8]: list

In [9]:
```python
plt.plot(x_axis, e_x)
plt.show()
```



In [10]:
```python
plt.xlabel("Titulo eje X")
plt.ylabel("Titulo eje Y")

# Have to plot our chart once again as it doesn't stick after being shown
plt.plot(x_axis, e_x)
plt.show()
```



In [ ]:
```python

```

In [11]:
```python
x_axis = np.arange(0, 6, 0.1)
sin = np.sin(x_axis)
cos = np.cos(x_axis)
```

In [12]:
```python
type(x_axis)
```

Out[12]: numpy.ndarray

In [13]:
```python
type(sin)
```

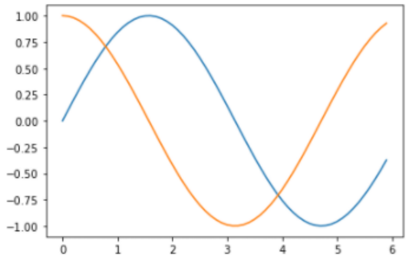
Out[13]: numpy.ndarray

In [14]:
```python
type(cos)
```

Out[14]: numpy.ndarray

In [15]:

```python
plt.plot(x_axis, sin)
plt.plot(x_axis, cos)

plt.show()
```



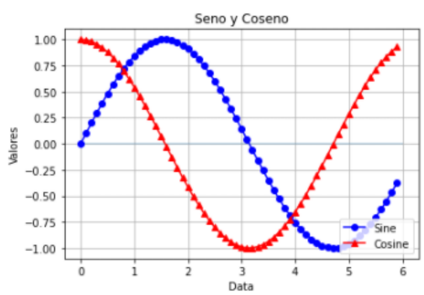
Estilo

In [18]:

```python
x_axis = np.arange(0, 6, 0.1)
sin = np.sin(x_axis)
cos = np.cos(x_axis)
```

In [19]:

```python
plt.hlines(0, 0, 6, alpha=0.25)
sine_handle = plt.plot(x_axis, sin, marker ='o', color='blue', label="Sine")
cosine_handle = plt.plot(x_axis, cos, marker='^', color='red', label="Cosine")
plt.legend(loc="lower right")
plt.title("Seno y Coseno")
plt.xlabel("Data")
plt.ylabel("Valores")
plt.grid()
plt.savefig("../resultados/04lineas.png")
plt.show()
```



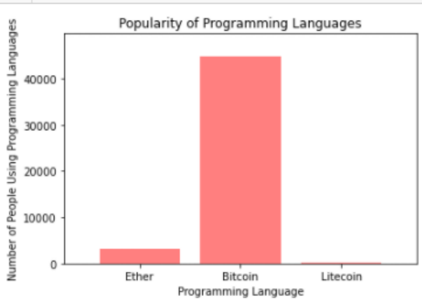
Barras

In [20]:

```python
precio = [3126, 44750, 163]
x_axis = np.arange(len(precio))
```

In [21]:

```python
plt.bar(x_axis, precio, color='r', alpha=0.5, align="center")
plt.xticks(x_axis, ["Ether", "Bitcoin", "Litecoin"])
plt.xlim(-0.75, len(x_axis)-0.25)
plt.ylim(0, max(precio)+5000)
plt.title("Popularity of Programming Languages")
plt.xlabel("Programming Language")
plt.ylabel("Number of People Using Programming Languages")
plt.show()
```

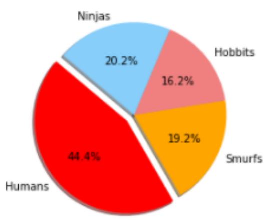


Pie

https://interworks.com/blog/rcurtis/2018/01/19/friends-dont-let-friends-make-pie-charts/

In [27]:

```
labels = ["Humans", "Smurfs", "Hobbits", "Ninjas"]

sizes = [220, 95, 80, 100]

colors = ["red", "orange", "lightcoral", "lightskyblue"]

explode = (0.1, 0, 0, 0)

plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct="%1.1f%%", shadow=True, startangle=140)

plt.show()
```



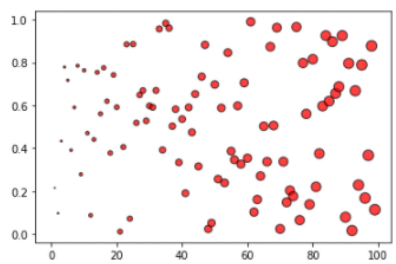
Dispersión

In [30]:

```
x_limit = 100
x_axis = np.arange(0, x_limit, 1)
data = [random.random() for value in x_axis]
```

In [31]:

```
plt.scatter(x_axis, data, marker="o", facecolors="red", edgecolors="black",
            s=x_axis, alpha=0.75)
plt.show()
```



5. Matplotlib Avanzado

Matplotlib & Pandas

In [1]:
```python
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```python
datos = pd.read_csv("../data/03audi.csv")
```

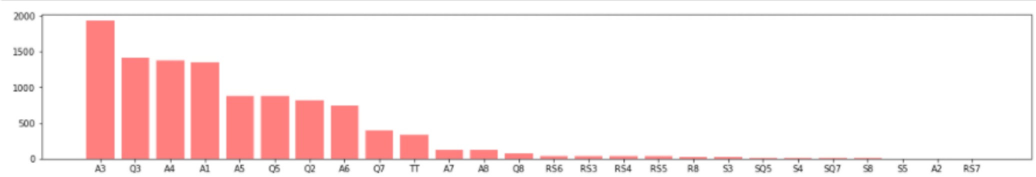
In [3]:
```python
datos.head()
```

Out[3]:

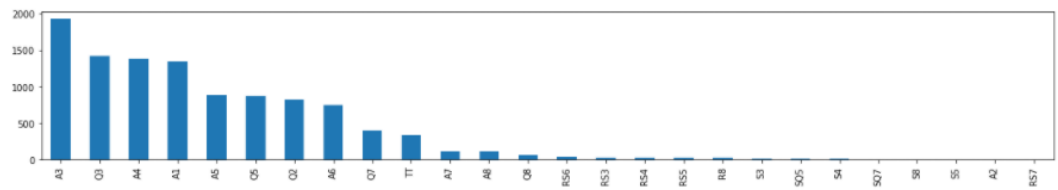
	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0

In [4]:
```python
resultado = datos["model"].value_counts()
```

In [5]:
```python
plt.figure(figsize=(20,3))
plt.bar(resultado.index, resultado, color='r', alpha=0.5, align="center")
plt.show()
```



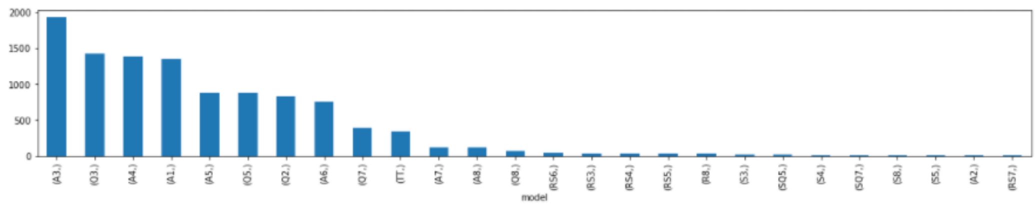
In [6]:
```python
resultado.plot(kind="bar", figsize=(20,3))
plt.show()
```



Group by

In [12]:
```
(
    pd.read_csv("../data/03audi.csv")
    [["model"]]
    .value_counts()
    .plot(kind="bar", figsize=(20,3))
)
```

Out[12]: <AxesSubplot:xlabel='model'>



In [13]:
```
resultado = pd.read_csv("../data/03audi.csv").groupby("model")[["model"]].agg(conteo= ("model","count"))
```

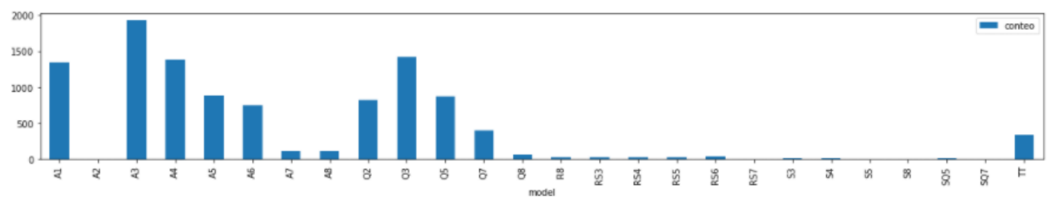
In [14]:
```
resultado.head()
```

Out[14]:

	conteo
model	
A1	1347
A2	1
A3	1929
A4	1381
A5	882

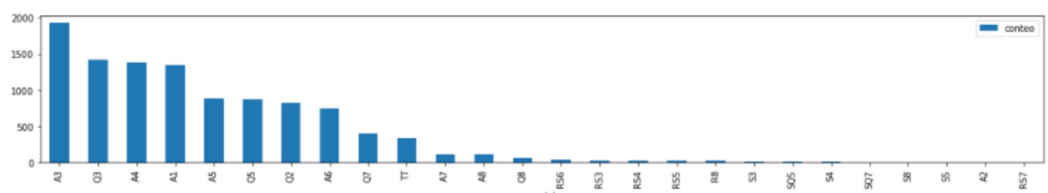
In [15]:
```
resultado.plot(kind="bar", figsize=(20,3))
```

Out[15]: <AxesSubplot:xlabel='model'>



In [16]:
```
(
    pd.read_csv("../data/03audi.csv")
    .groupby("model")
    [["model"]]
    .agg(conteo= ("model","count"))
    .sort_values("conteo", ascending=False)
    .plot(kind="bar", figsize=(20,3))
)
```

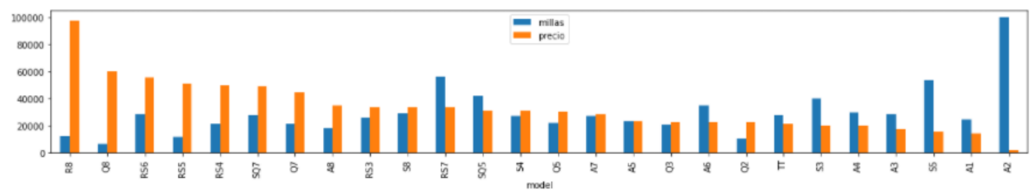
Out[16]: <AxesSubplot:xlabel='model'>



In [17]:

```
(
    pd.read_csv("../data/03audi.csv")
    .groupby("model")
    [["mileage","price"]]
    .agg(millas= ("mileage","mean"), precio=("price","mean"))
    .sort_values("precio", ascending=False)
    .plot(kind="bar", figsize=(20,3))
)
```

Out[17]: `<AxesSubplot:xlabel='model'>`



In [ ]:

```

```

In [ ]:

```

```

Series de tiempo

Utilizando los datos de acciones, genera las líneas de open price por año para Apple y Amazon

In [18]:

```
apple = pd.read_csv("../data/05aapl.csv")
```

In [19]:

```
amazon = pd.read_csv("../data/05amzn.csv")
```

In [20]:

```
apple.head()
```

Out[20]:

	symbol	date	open	high	low	close	volume
0	AAPL	02/01/2014	79.3828	79.5756	78.8601	79.0185	58791957
1	AAPL	03/01/2014	78.9799	79.0999	77.2042	77.2828	98303870
2	AAPL	06/01/2014	76.7785	78.1142	76.2285	77.7042	103359151
3	AAPL	07/01/2014	77.7599	77.9942	76.8464	77.1481	79432766
4	AAPL	08/01/2014	76.9728	77.9371	76.9556	77.6371	64686685

In [21]:
```
amazon.head()
```
Out[21]:

	symbol	date	open	high	low	close	volume
0	AMZN	02/01/2014	398.80	399.36	394.02	397.97	2140246
1	AMZN	03/01/2014	398.29	402.71	396.22	396.44	2213512
2	AMZN	06/01/2014	395.85	397.00	388.42	393.63	3172207
3	AMZN	07/01/2014	395.04	398.47	394.29	398.03	1916684
4	AMZN	08/01/2014	398.47	403.00	396.04	401.92	2316903

In [22]:
```
apple.append(amazon)
```
Out[22]:

	symbol	date	open	high	low	close	volume
0	AAPL	02/01/2014	79.3828	79.5756	78.8601	79.0185	58791957
1	AAPL	03/01/2014	78.9799	79.0999	77.2042	77.2828	98303870
2	AAPL	06/01/2014	76.7785	78.1142	76.2285	77.7042	103359151
3	AAPL	07/01/2014	77.7599	77.9942	76.8464	77.1481	79432766
4	AAPL	08/01/2014	76.9728	77.9371	76.9556	77.6371	64686685
...	...	...	...	...	...	...	...
1002	AMZN	22/12/2017	1172.0800	1174.6200	1167.8300	1168.3600	1585054
1003	AMZN	26/12/2017	1168.3600	1178.3200	1160.5500	1176.7600	2005187
1004	AMZN	27/12/2017	1179.9100	1187.2900	1175.6100	1182.2600	1867208
1005	AMZN	28/12/2017	1189.0000	1190.1000	1184.3800	1186.1000	1841676
1006	AMZN	29/12/2017	1182.3500	1184.0000	1167.5000	1169.4700	2688391

2014 rows × 7 columns

In [23]:
```
acciones = apple.append(amazon)
```
In [24]:
```
acciones.dtypes
```
Out[24]:
```
symbol     object
date       object
open      float64
high      float64
low       float64
close     float64
volume      int64
dtype: object
```

In [25]:
```
pd.to_datetime(acciones["date"])
```
Out[25]:
```
0      2014-02-01
1      2014-03-01
2      2014-06-01
3      2014-07-01
4      2014-08-01
          ...
1002   2017-12-22
1003   2017-12-26
1004   2017-12-27
1005   2017-12-28
1006   2017-12-29
Name: date, Length: 2014, dtype: datetime64[ns]
```
In [26]:
```
acciones["date"] = pd.to_datetime(acciones["date"])
```
In [27]:
```
acciones.dtypes
```
Out[27]:
```
symbol            object
date      datetime64[ns]
open             float64
high             float64
low              float64
close            float64
volume             int64
dtype: object
```
In [28]:
```
acciones.head()
```
Out[28]:

	symbol	date	open	high	low	close	volume
0	AAPL	2014-02-01	79.3828	79.5756	78.8601	79.0185	58791957
1	AAPL	2014-03-01	78.9799	79.0999	77.2042	77.2828	98303870
2	AAPL	2014-06-01	76.7785	78.1142	76.2285	77.7042	103359151
3	AAPL	2014-07-01	77.7599	77.9942	76.8464	77.1481	79432766
4	AAPL	2014-08-01	76.9728	77.9371	76.9556	77.6371	64686685

In [29]:
```
acciones["date"].dt.year
```
Out[29]:
```
0       2014
1       2014
2       2014
3       2014
4       2014
        ...
1002    2017
1003    2017
1004    2017
1005    2017
1006    2017
Name: date, Length: 2014, dtype: int64
```

In [30]:
```
acciones["year"] = acciones["date"].dt.year

acciones.head()
```
Out[30]:

	symbol	date	open	high	low	close	volume	year
0	AAPL	2014-02-01	79.3828	79.5756	78.8601	79.0185	58791957	2014
1	AAPL	2014-03-01	78.9799	79.0999	77.2042	77.2828	98303870	2014
2	AAPL	2014-06-01	76.7785	78.1142	76.2285	77.7042	103359151	2014
3	AAPL	2014-07-01	77.7599	77.9942	76.8464	77.1481	79432766	2014
4	AAPL	2014-08-01	76.9728	77.9371	76.9556	77.6371	64686685	2014

In [31]:
```
resultado = acciones.groupby(["symbol","year"])[["open"]].mean().reset_index()
```

In [32]:
```
resultado
```
Out[32]:

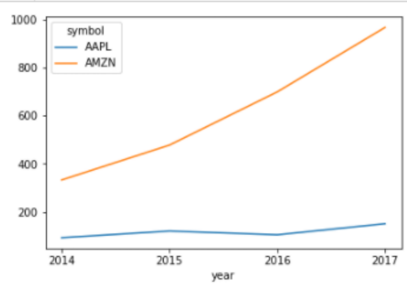
	symbol	year	open
0	AAPL	2014	92.219736
1	AAPL	2015	120.169206
2	AAPL	2016	104.507698
3	AAPL	2017	150.482560
4	AMZN	2014	332.798433
5	AMZN	2015	478.126230
6	AMZN	2016	699.756587
7	AMZN	2017	968.253825

In [33]:
```
resultado = resultado.pivot(index='year',columns='symbol',values="open")
```

In [34]:
```
resultado
```
Out[34]:

symbol	AAPL	AMZN
year		
2014	92.219736	332.798433
2015	120.169206	478.126230
2016	104.507698	699.756587
2017	150.482560	968.253825

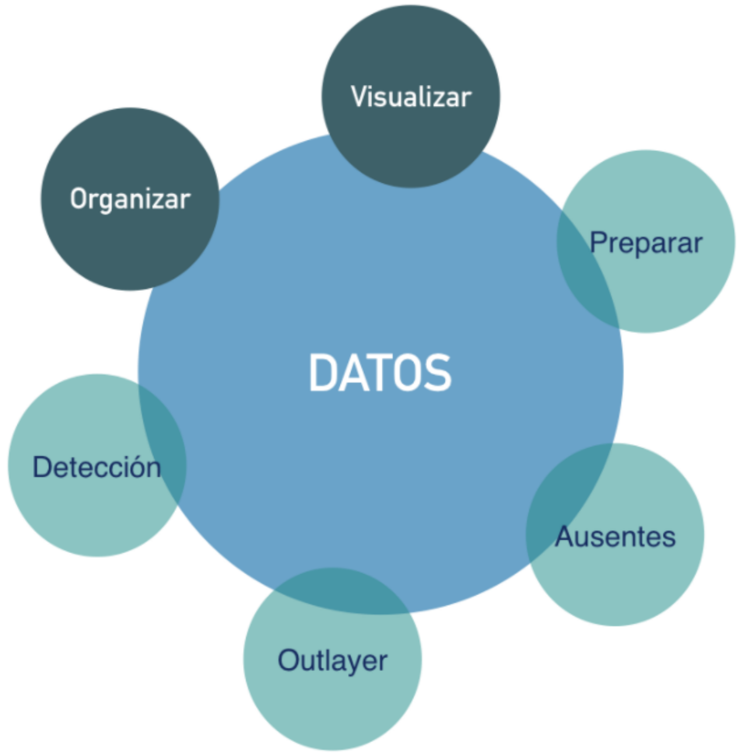
In [35]:
```
resultado.plot(kind="line",xticks=resultado.index)
plt.show()
```



6. EDA & ETL

Exploratory Data Análysis (EDA)

Análisis exploratorio de los datos



CLOUD TECHNOLOGY TRAINING

In [2]:
```python
# Dependencias
import pandas as pd
import matplotlib.pyplot as plt
```

1) Cargar datos

In [3]:
```python
datos = pd.read_csv("../data/06Bank.csv")
```

2) Descriptivos de los datos

In [4]:
```python
datos.shape
```

Out[4]:
```
(10000, 14)
```

In [5]:
```python
datos.columns
```

Out[5]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [13]:
```python
datos.describe()
```

Out[13]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	H
count	10000.0	10000.0	10000.0	10000.0	10000.0	9998.0	10000.0	
mean	5000.5	15690940.5694	650.5288	38.9218	5.0128	76489.1271754347	1.5302	
std	2886.8956799071675	71936.18612274883	96.65329873613061	10.487806451704591	2.892174377049708	62397.39772884749	0.5816543579989936	0.45584046
min	1.0	15565701.0	350.0	18.0	0.0	0.0	1.0	
25%	2500.75	15628528.25	584.0	32.0	3.0	0.0	1.0	
50%	5000.5	15690738.0	652.0	37.0	5.0	97198.54000000001	1.0	
75%	7500.25	15753233.75	718.0	44.0	7.0	127647.84	2.0	
max	10000.0	15815690.0	850.0	92.0	10.0	250898.09	4.0	

In [14]:
```python
pd.set_option('display.float_format', str)
```

In [15]:
```python
datos.describe()
```

Out[15]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	H
count	10000.0	10000.0	10000.0	10000.0	10000.0	9998.0	10000.0	
mean	5000.5	15690940.5694	650.5288	38.9218	5.0128	76489.1271754347	1.5302	
std	2886.8956799071675	71936.18612274883	96.65329873613061	10.487806451704591	2.892174377049708	62397.39772884749	0.5816543579989936	0.45584046
min	1.0	15565701.0	350.0	18.0	0.0	0.0	1.0	
25%	2500.75	15628528.25	584.0	32.0	3.0	0.0	1.0	
50%	5000.5	15690738.0	652.0	37.0	5.0	97198.54000000001	1.0	
75%	7500.25	15753233.75	718.0	44.0	7.0	127647.84	2.0	
max	10000.0	15815690.0	850.0	92.0	10.0	250898.09	4.0	

In [16]:
```python
datos.head()
```

Out[16]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	1	15634602	Hargrave	619	France	Female	42	2	0.0	1	1	1	101348.88
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
2	3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57
3	4	15701354	Boni	699	France	Female	39	1	0.0	2	0	0	93826.63
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1

In [17]:
```python
datos.tail()
```

Out[17]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSa
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.0	2	1	0	9627
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	1	10169
9997	9998	15584532	Liu	709	France	Female	36	7	0.0	1	0	1	4208
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	0	9288
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	0	3819

3) Detectar valores perdidos

In [18]:
```
datos.count()
```

Out[18]:
```
RowNumber          10000
CustomerId         10000
Surname            10000
CreditScore        10000
Geography          10000
Gender             10000
Age                10000
Tenure             10000
Balance             9998
NumOfProducts      10000
HasCrCard          10000
IsActiveMember     10000
EstimatedSalary     9998
Exited             10000
dtype: int64
```

In [19]:
```
# Tenemos valores perdidos, vamos a separarlos de nuestro resto de datos
```

In [32]:
```
filtro = ((pd.isna(datos["Balance"])) | (pd.isna(datos["EstimatedSalary"])))
```

In [33]:
```
errores = datos[filtro]
```

In [34]:
```
errores
```

Out[34]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
493	494	15725679	Hsia	531	France	Female	47	6	NaN	1	0	0	194998.34
635	636	15633648	Jideofor	696	Spain	Female	51	5	0.0	2	1	0	NaN
802	803	15681554	Alley	614	Germany	Female	31	7	NaN	2	1	1	NaN

In [35]:
```
datos = datos.dropna(how="any")
```

In [37]:
```
1 datos.count()
```

Out[37]:
```
RowNumber          9997
CustomerId         9997
Surname            9997
CreditScore        9997
Geography          9997
Gender             9997
Age                9997
Tenure             9997
Balance            9997
NumOfProducts      9997
HasCrCard          9997
IsActiveMember     9997
EstimatedSalary    9997
Exited             9997
dtype: int64
```

4) Análisis univariado - Exploración visual & detectar atípicos

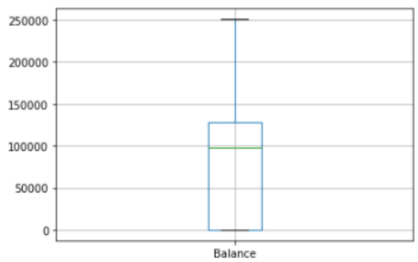
In [39]:
```
1 datos.describe()
```

Out[39]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	
count	9997.0	9997.0	9997.0	9997.0	9997.0	9997.0	9997.0	
mean	5001.807242172652	15690943.764429329	650.5398619585876	38.92057617285185	5.012503751125338	76496.77838351465	1.5301590477143143	0.705511
std	2886.341332425962	71943.79761817645	96.65864728071975	10.488073875864002	2.8925231856049805	62395.82831675655	0.5816795037900425	0.4558352
min	1.0	15565701.0	350.0	18.0	0.0	0.0	1.0	
25%	2503.0	15628523.0	584.0	32.0	3.0	0.0	1.0	
50%	5002.0	15690743.0	652.0	37.0	5.0	97208.46	1.0	
75%	7501.0	15753248.0	718.0	44.0	7.0	127649.64	2.0	
max	10000.0	15815690.0	850.0	92.0	10.0	250898.09	4.0	

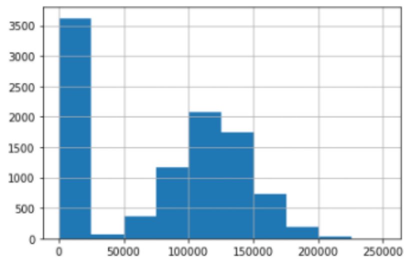
In [42]:
```
1 datos.boxplot(column=["Balance"])
```

Out[42]: <AxesSubplot:>

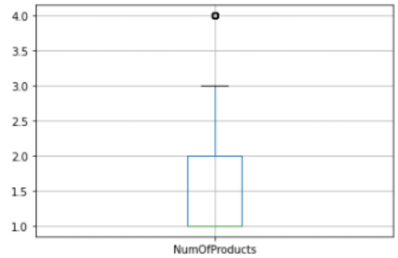


In [50]:
```
1 datos["Balance"].hist()
```

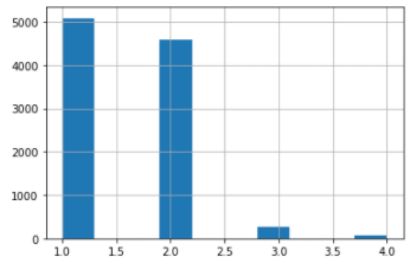
Out[50]: <AxesSubplot:>



In [44]:
```
datos.boxplot(column=["NumOfProducts"])
```
Out[44]: <AxesSubplot:>



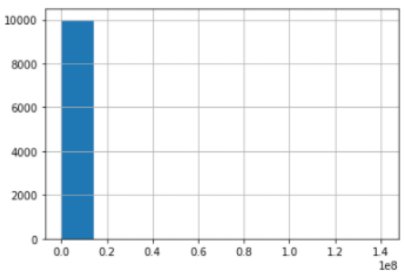
In [49]:
```
datos["NumOfProducts"].hist()
```
Out[49]: <AxesSubplot:>



In [59]:
```
datos["NumOfProducts"].describe()
```
Out[59]:
```
count                 9997.0
mean      1.5301590477143143
std       0.5816795037900425
min                      1.0
25%                      1.0
50%                      1.0
75%                      2.0
max                      4.0
Name: NumOfProducts, dtype: float64
```

In [45]:
```
datos.boxplot(column=["EstimatedSalary"])
```
Out[45]: <AxesSubplot:>

In [48]:
```
datos["EstimatedSalary"].hist()
```
Out[48]: <AxesSubplot:>



In [56]:
```
datos["EstimatedSalary"].describe()
```
Out[56]:
```
count                9997.0
mean     114176.58612383706
std     1408696.1778245952
min                   11.58
25%                51016.02
50%               100236.02
75%                149399.7
max             140831200.0
Name: EstimatedSalary, dtype: float64
```

In [58]:
```
(datos["EstimatedSalary"]/1000000).describe()
```
Out[58]:
```
count                 9997.0
mean      0.11417658612383755
std        1.4086961778245952
min                 1.158e-05
25%      0.051016019999999995
50%       0.10023602000000001
75%       0.14939970000000002
max                  140.8312
Name: EstimatedSalary, dtype: float64
```

In [60]:
```
max(datos["EstimatedSalary"])
```
Out[60]: 140831200.0

In [65]:
```
revisar = datos[datos["EstimatedSalary"]==140831200]
```

In [67]:
```
datos = datos[datos["EstimatedSalary"]!=140831200]
```

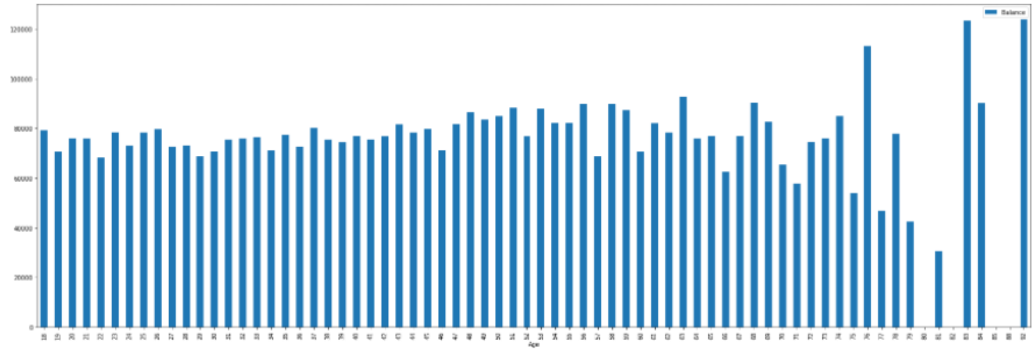
In [68]:
```
datos.head()
```
Out[68]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	1	15634602	Hargrave	619	France	Female	42	2	0.0	1	1	1	101348.88
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
2	3	15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57
3	4	15701354	Boni	699	France	Female	39	1	0.0	2	0	0	93826.63
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1

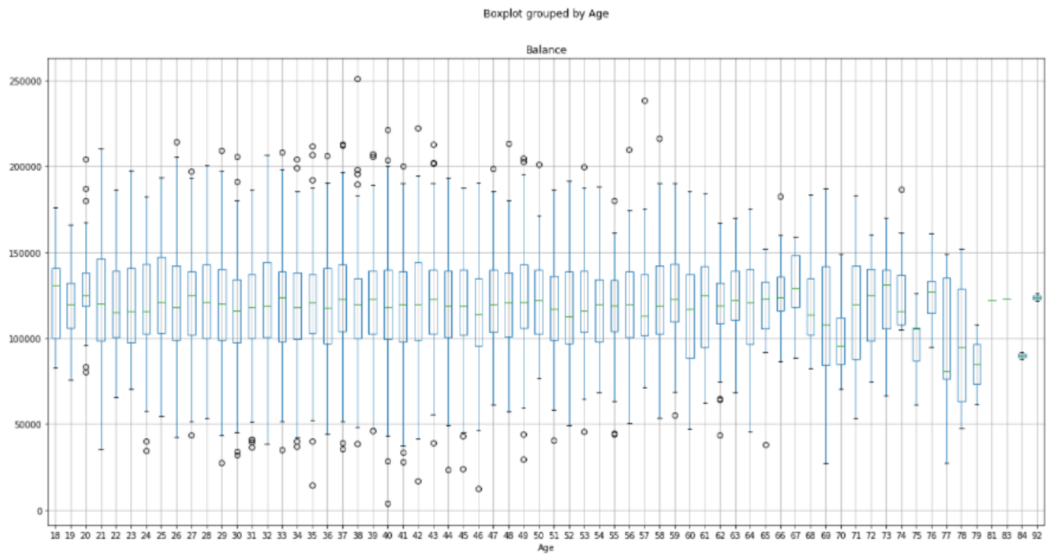
In [ ]:

5) Análisis bivariado

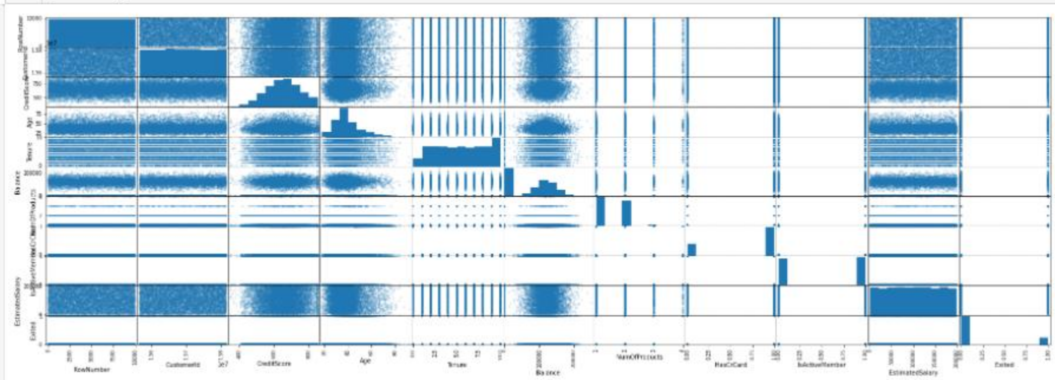
In [87]:
```
datos.groupby("Age")[["Balance"]].mean().plot(kind="bar", figsize=(30,10))
plt.show()
```



In [32]:
```
datos.query("Balance>0").boxplot(by="Age",column="Balance", figsize=(20,10))
plt.show()
```

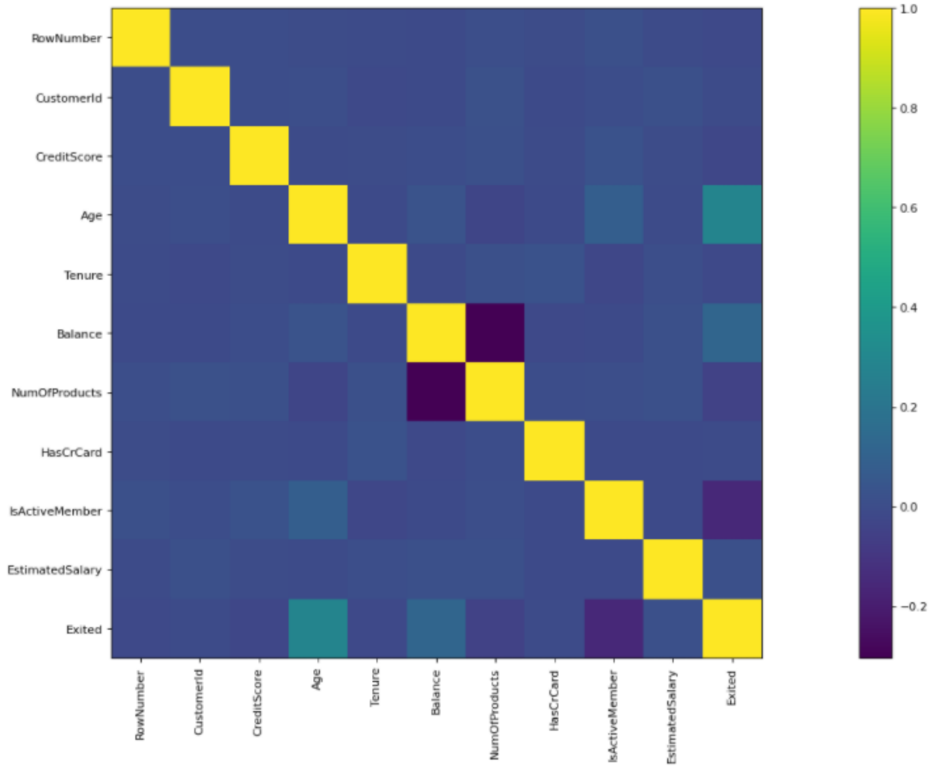


In [33]:
```
pd.plotting.scatter_matrix(datos, alpha=0.2, figsize=(30,10))
plt.show()
```



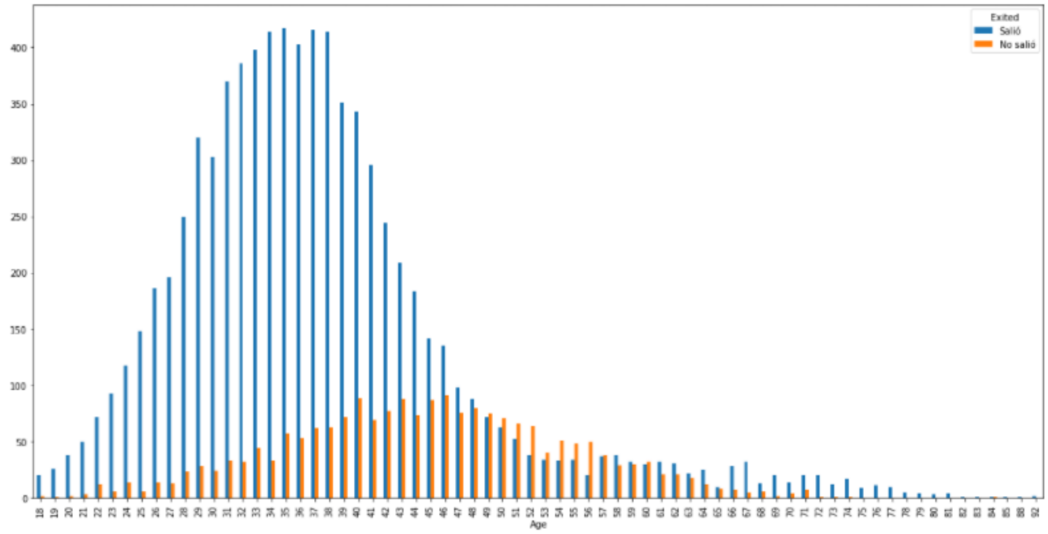
In [34]:

```
corr = datos.corr()
plt.figure(num=None, figsize=(30, 10), dpi=80, facecolor='w', edgecolor='k')
corrMat = plt.matshow(corr, fignum = 1)
plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
plt.yticks(range(len(corr.columns)), corr.columns)
plt.gca().xaxis.tick_bottom()
plt.colorbar(corrMat)
plt.show()
```

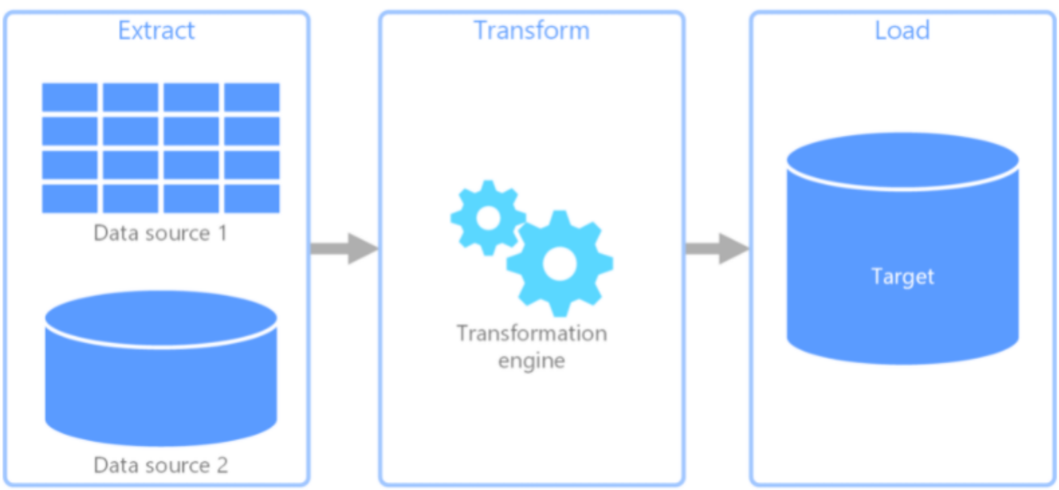


In [35]:

```python
(
    datos
    .groupby(["Age","Exited"])
    .agg(Conteo = ("Exited","count"))
    .reset_index()
    .pivot(index='Age',columns='Exited',values="Conteo")
    .rename(columns={0:"Salió",1:"No salió"})
    .plot(kind="bar", figsize=(20,10))
)
plt.show()
```



ETL



In [36]:

```python
import sqlalchemy
from sqlalchemy import create_engine
```

In [37]:

```python
engine = create_engine("sqlite:///../resultados/bank.sqlite")
```

In [38]:

```python
datos.to_sql("datos", engine,if_exists="replace")
```

7. Temas selectos

Deployment

Algunos ejemplos selectos del tipo de implementaciones que se pueden realizar

Jupyter notebook -> Script

Reportes automáticos

ETL automatizado

API - flask