Informe Examen Transversal Machine Learning

Introducción

Fase 1 - Entendimiento del caso

Este informe presenta los resultados del análisis de datos sobre el banco monopoly,podemos apreciar en la siguiente imagen el contexto del negocio, una diccionario con las distintas variables que se usaran a lo largo de la exploración de datos, limpieza de datos y fases posteriores ademas de los objetivos generales y específicos propuestos.

Resumen del contexto

El cliente referido como "Banco monopoly" me contrato como analista de datos, Logrando asi identificar distintos aspectos en el contexto de negocios,identificando que se necesita una estrategia para atraer potenciales nuevos clientes al banco para ello, Es importante analizar el uso de los productos financieros. Finalmente, podemos decir que la base de datos tiene 574 variables (columnas) y 51.124 registros (filas).
También es importante destacar que cada fila contiene registros únicos de clientes , todos los datos tratados corresponden a una base que comprende un total de 12 meses de datos de los clientes banco monopoly de forma mensual por cada cliente. Ademas para este caso se usara la metodologia CRISP-DM para una mejor organizacion ya que se dividen en distintas fases y mejor entendimiento del caso.

Dato	Descripción
region	region de residencia
renta	renta de cliente
sexo	sexo
Edad	Edad
Adicional	indicador de tenecia de TC adicionales
Antiguedad	Antiguedad de clientes (meses)
consumo	indicador de credito consumo
debito	indicador de tenecia de TD
ctacte	indicador de cuenta corriente
cuentas	numero de cuentas que tiene el cliente
hipotecario	indicador de credito hipotecario
TC	numero de TC que tiene el cliente

Objetivos generales

- Identificar grupos de clientes con comportamientos similares para personalizar productos y servicios.
- identificar que grupo de clientes que realiza mayor compra de productos y servicios.
- Identificar los productos financieros que generan más ingresos o que tienen mayor aceptación entre los clientes.

Objetivos especificos

- Clasificar a los clientes según su nivel de riesgo de crédito en base a indicador de cuenta corriente
- Predecir el monto total de compras anuales de un cliente en función de su edad.

Fase 2 - Exploración de los datos

En la exploración de los datos no centramos en conocer en dataframe y sus distintas columnas y filas junto con sus distintos tipos de datos que contienen

```
[4] df.columns

Index(['Id', 'Subsegmento', 'Sexo', 'Region', 'Edad', 'Renta', 'Antiguedad',
       'Internauta', 'Adicional', 'Dualidad',
       ...
       'PagoNac_T01', 'PagoInt_T01', 'EeccNac_T01', 'EeccInt_T01', 'UsoL1_T01',
       'UsoL2_T01', 'UsoLI_T01', 'IndRev_T01', 'target', 'Unnamed: 574'],
      dtype='object', length=575)

[5] df.shape

(51124, 575)

[6] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51124 entries, 0 to 51123
Columns: 575 entries, Id to Unnamed: 574
dtypes: float64(509), int64(53), object(13)
memory usage: 224.3+ MB
```

podemos ver en el siguiente codigo los distintos datos con los rangos intercuartiles ademas de el valor min y el valor maximo.

```
[7] df.describe()
```

	Id	Subsegmento	Region	Edad	Renta	Antiguedad	Internauta	Adicional	Dualidad	Monoproducto	...	ColMx_T01	PagoNac_T01	PagoInt_T01	EeccNac_T01	EeccInt_T01	
count	51124.000000	51124.000000	51071.000000	51124.000000	3.775900e+04	51124.000000	51124.000000	51124.000000	51124.000000	51124.000000	...	5.112400e+04	5.112400e+04	5.112400e+04	5.112400e+04	51124.000000	5.1
mean	25562.500000	182.024274	10.828220	38.702879	6.630771e+05	38.896154	0.684199	0.256181	0.381347	0.063141	...	5.237914e+03	7.637553e+04	1.734930e+03	1.939488e+05	7.323155	1.8
std	14758.371918	29.276596	3.392703	13.302573	4.092795e+05	35.672549	0.464839	0.436527	0.485722	0.243218	...	4.852871e+04	1.490256e+05	4.235368e+04	2.884980e+05	108.161194	2.8
min	1.000000	151.000000	1.000000	9.000000	1.000000e+00	6.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000e+00	0.000000e+00	0.000000e+00	-1.861866e+06	-7886.760000	-3.7
25%	12781.750000	160.000000	9.000000	28.000000	4.199990e+05	14.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000e+00	5.000000e+03	0.000000e+00	3.198100e+04	0.000000	1.7
50%	25562.500000	170.000000	13.000000	35.000000	5.670120e+05	25.000000	1.000000	0.000000	0.000000	0.000000	...	0.000000e+00	3.400150e+04	0.000000e+00	9.223050e+04	0.000000	8.1
75%	38343.250000	210.000000	13.000000	46.000000	8.149035e+05	54.000000	1.000000	1.000000	1.000000	0.000000	...	0.000000e+00	9.200000e+04	0.000000e+00	2.359780e+05	0.000000	2.3
max	51124.000000	959.000000	13.000000	104.000000	1.308933e+07	324.000000	1.000000	1.000000	1.000000	1.000000	...	2.072818e+06	8.697782e+06	4.219680e+06	6.911556e+06	3929.810000	6.9

8 rows × 562 columns

En esta imagen se observa las primeras 5 filas y las ultimas 5 filas del dataframe ademas de los distintos datos que contienen cada una de las columnas

```
df.head()
```

	Id	Subsegmento	Sexo	Region	Edad	Renta	Antiguedad	Internauta	Adicional	Dualidad	...	PagoNac_T01	PagoInt_T01	EeccNac_T01	EeccInt_T01	UsoL1_T01	UsoL2_T01	UsoLI_T01	IndRev_T01	target	Unnamed: 574
0	1	160	M	13.0	43	NaN	130	1	1	0	...	33000	0.0	1099866.0	0.0	1099866.0	15080	0.0	R	0	NaN
1	2	160	H	13.0	46	143640.0	69	1	0	0	...	300000	0.0	214592.0	0.0	214592.0	83596	0.0	R	0	NaN
2	3	170	H	13.0	45	929106.0	24	1	1	0	...	216676	0.0	0.0	0.0	7400.0	0	0.0	T	0	NaN
3	4	151	H	13.0	46	172447.0	134	0	1	0	...	60000	0.0	272762.0	0.0	272762.0	10591	0.0	R	0	NaN
4	5	170	H	13.0	46	805250.0	116	0	1	1	...	272925	0.0	249562.0	0.0	75339.0	377782	0.0	R	0	NaN

5 rows × 575 columns

+ Código + Texto

```
df.tail()
```

	Id	Subsegmento	Sexo	Region	Edad	Renta	Antiguedad	Internauta	Adicional	Dualidad	...	PagoNac_T01	PagoInt_T01	EeccNac_T01	EeccInt_T01	UsoL1_T01	UsoL2_T01	UsoLI_T01	IndRev_T01	target	Unnamed: 574
51119	51120	160	H	13.0	51	364978.0	57	1	1	1	...	300000	0.0	478320.0	0.0	478320.0	12668	0.0	R	0	NaN
51120	51121	170	H	13.0	51	625376.0	39	1	0	0	...	166098	0.0	166098.0	0.0	0.0	572363	0.0	R	0	NaN
51121	51122	160	H	13.0	47	806220.0	153	1	1	0	...	18891	0.0	9652.0	0.0	9652.0	16241	0.0	R	0	NaN
51122	51123	160	M	13.0	47	NaN	11	1	0	0	...	26528	0.0	24638.0	0.0	24638.0	84982	0.0	R	0	NaN
51123	51124	170	H	13.0	51	840878.0	75	1	1	0	...	12360	0.0	12360.0	0.0	18500.0	0	0.0	R	0	NaN

5 rows × 575 columns

En estas imágenes se puede apreciar el análisis de esta dos variables las cuales son especificadas en los objetivos específicos

Analisis de la variable Edad

```
[10] df['Edad'].mean()
```

```
38.70287927392223
```

```
df['Edad'].median()
```

```
35.0
```

```
[12] df['Edad'].mode()
```

	Edad
0	27

dtype: int64

Analisis de la variable Ctacte

```
[70] df['Ctacte'].mean()
```

```
0.9258415938757484
```

```
[71] df['Ctacte'].median()
```

```
1.0
```

```
[72] df['Ctacte'].mode()
```

	Ctacte
0	1

dtype: int64

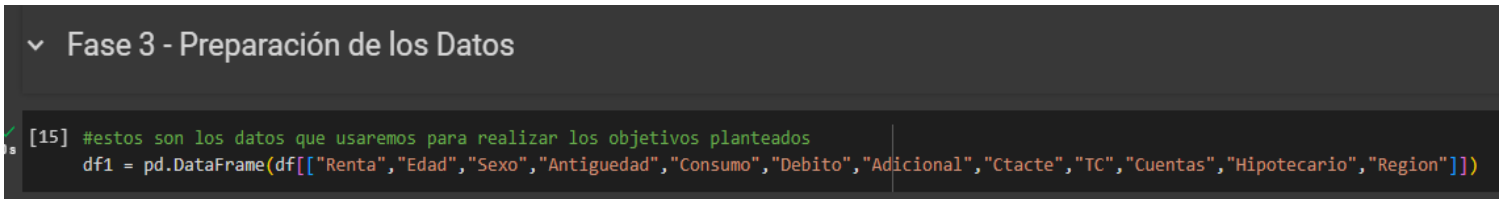
Análisis de datos

Fase 3-Preparación de datos

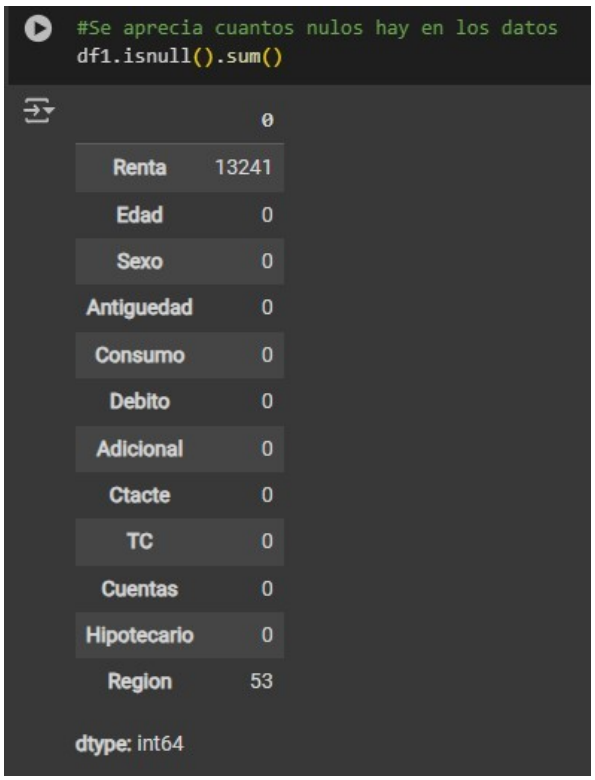
Los datos fueron limpiados y preparados utilizando las siguientes técnicas:

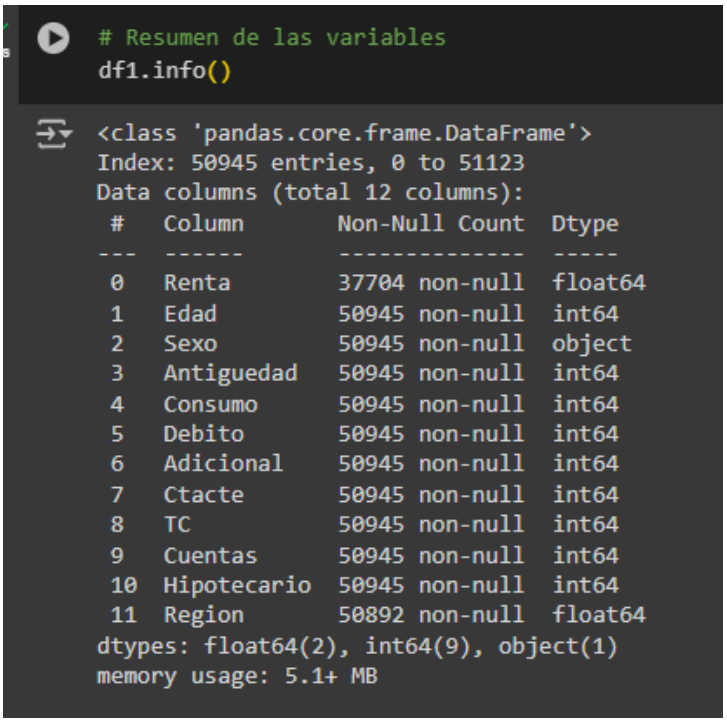
* Detección de valores faltantes
* Eliminación de datos duplicados
* Normalización de los datos
* Manejo de valores atípicos

Podemos apreciar en la siguiente imagen que en la tercera fase se realiza una copia con las variables a utilizar descartando así las que no nos sirven



Siguiendo con la preparación de datos verificamos si hay datos faltantes en las distintas variables en nuestro caso hay datos faltantes en renta y región,ademas se puede apreciar de que tipo de datos son los distintos datos.





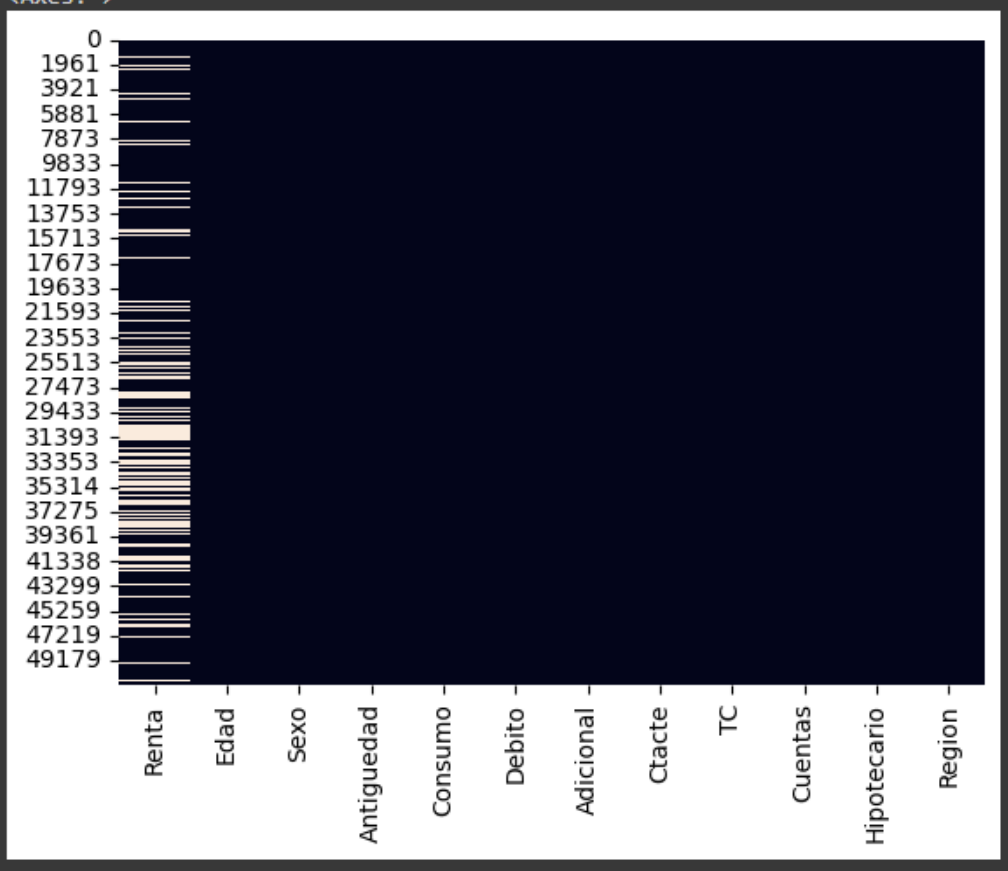
La imagen nos muestra una descripción de las distintas variables como la media, su valor mínimo,su valor Máximo,

```
[18] # Estadísticas descriptivas
     df1.describe()
```

	Renta	Edad	Antiguedad	Consumo	Debito	Adicional	Ctacte	TC	Cuentas	Hipotecario	Region
count	3.770400e+04	50945.000000	50945.000000	50945.000000	50945.000000	50945.000000	50945.000000	50945.000000	50945.000000	50945.000000	50892.000000
mean	6.632504e+05	38.545255	38.682756	0.000883	0.876396	0.255766	0.925842	1.733045	1.408323	0.138012	10.827694
std	4.091703e+05	13.051162	35.410446	0.029708	0.329132	0.436295	0.262031	0.877816	0.550705	0.344916	3.393353
min	1.000000e+00	19.000000	6.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000
25%	4.200000e+05	28.000000	14.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	0.000000	9.000000
50%	5.672410e+05	35.000000	25.000000	0.000000	1.000000	0.000000	1.000000	2.000000	1.000000	0.000000	13.000000
75%	8.150000e+05	46.000000	54.000000	0.000000	1.000000	1.000000	1.000000	2.000000	2.000000	0.000000	13.000000
max	1.308933e+07	80.000000	324.000000	1.000000	1.000000	1.000000	1.000000	12.000000	5.000000	1.000000	13.000000

Se puede apreciar los valores faltantes de forma más clara gracias a que es un gráfico de calor

```
[22] #Identificamos los missing values visualmente
     sns.heatmap(df1.isnull(), cbar=False)
```



Se procesa la variable renta rellenando los valores faltantes ademas de remplazar las comas por puntos para un mayor procesamiento,ademas se rellenan los datos faltantes Con knn imputer el cual ayuda a rellenar de forma mas uniforme los datos.

exploramos los datos de renta

```
[23] df1.Renta.unique()
```

```
array([    nan, 143640., 929106., ..., 625376., 806220., 840878.])
```

```
df1.Renta.info()
```

```
<class 'pandas.core.series.Series'>
Index: 50945 entries, 0 to 51123
Series name: Renta
Non-Null Count  Dtype
--------------  -----
37704 non-null  float64
dtypes: float64(1)
memory usage: 796.0 KB
```

```
[25] df1.Renta.describe()
```

	Renta
count	3.770400e+04
mean	6.632504e+05
std	4.091703e+05
min	1.000000e+00
25%	4.200000e+05
50%	5.672410e+05
75%	8.150000e+05
max	1.308933e+07

dtype: float64

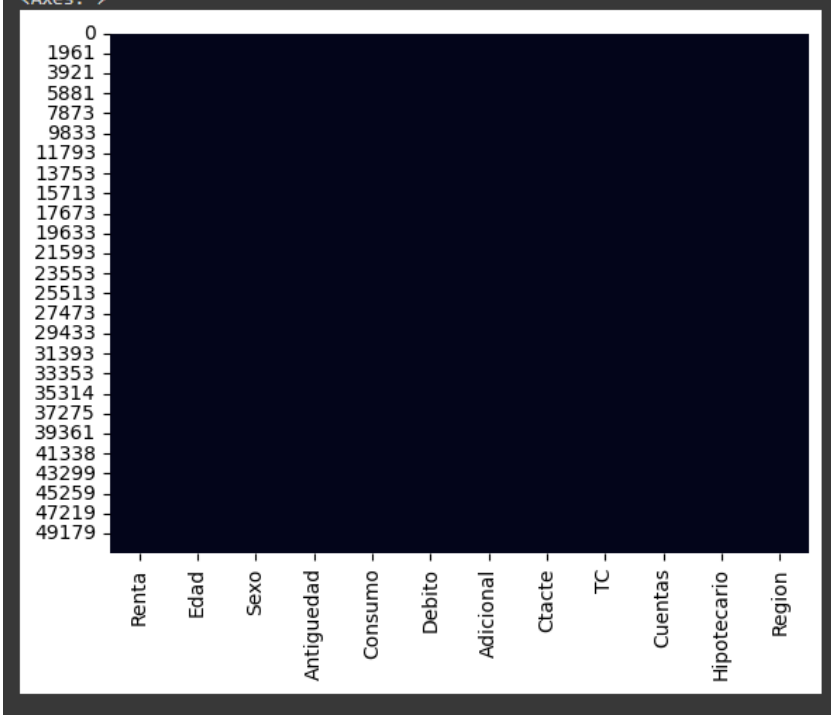
```
[28] ## primero cambiamos las comas por puntos
     df1['Renta'] = df1['Renta'].astype(str).str.replace(',', '.').astype(float)
```

```
[32] #rellenamos los datos faltantes con knn inputer
     knn_imputer = KNNImputer(n_neighbors=3, weights="uniform")
     df1[['Renta']] = knn_imputer.fit_transform(df1[['Renta']])
```

Se observa de forma mas visual el rellena miento de datos con las técnicas antes mencionadas como knn-imputer

```
#comprobamos de forma mas visual los datos faltantes
sns.heatmap(df1.isnull(), cbar=False)
```

```
<Axes: >
```



Se observa que las variables Sexo y Región también contienen datos faltantes pero donde son tan pocos se eliminan sin afectar mucho al dataframe ademas que se verifica por ultima si quedan datos nulos y con ello terminar de limpiar los datos nulos

```
[27] #considerando que los datos faltantes son 53 de region y 1 de sexo los eliminamos
     df1.dropna(subset=['Sexo', 'Region'], inplace=True)
```

```
#verificamos denuevo los datos faltantes y vemos que estan limpios
df1.isna().sum()
```

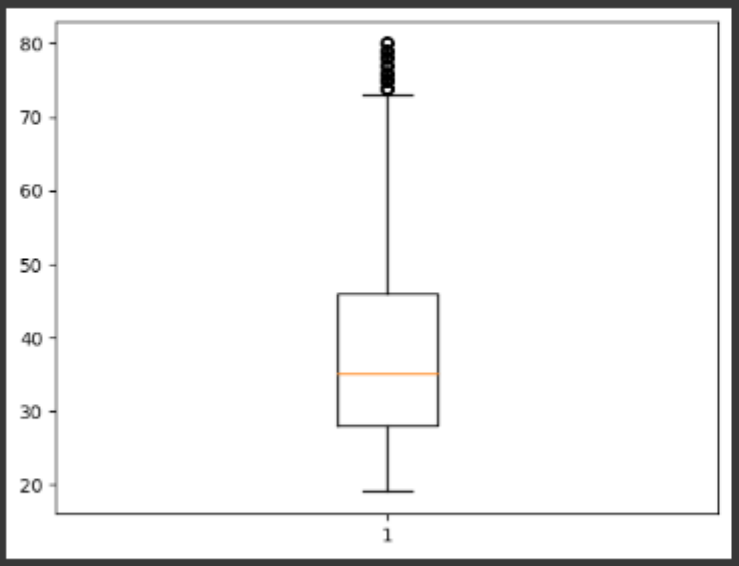
	0
Renta	0
Edad	0
Sexo	0
Antiguedad	0
Consumo	0
Debito	0
Adicional	0
Ctacte	0
TC	0
Cuentas	0
Hipotecario	0
Region	0

dtype: int64

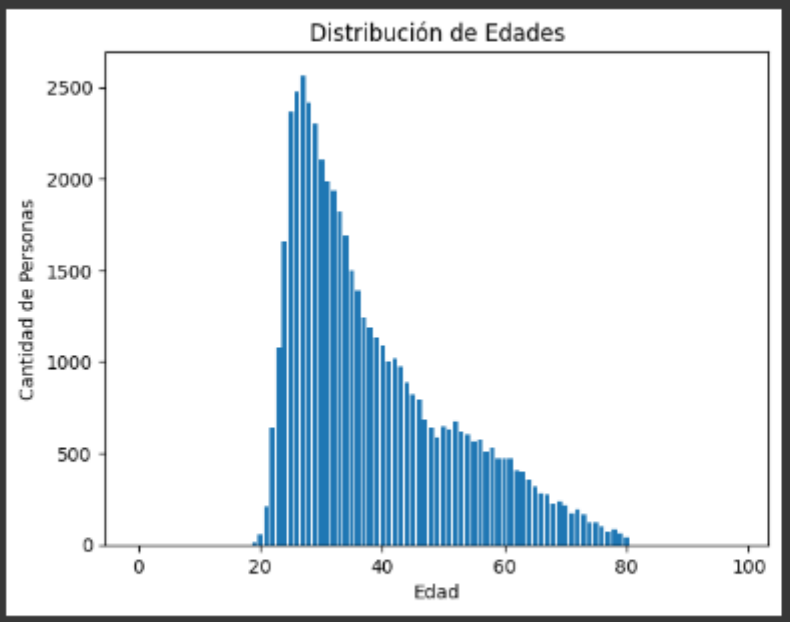
En esta imagen se puede apreciar como se manejan los valor atípicos y eliminación de estos mismos

Manejo de los valores atipicos

```
[65] plt.boxplot(list(df1['Edad']))
     plt.show()
```

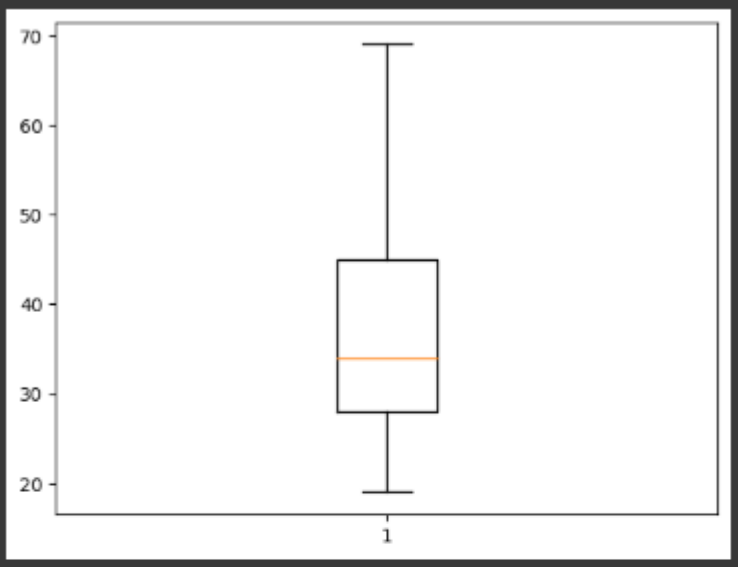



```
# distriución de la variable edad

plt.hist(df1['Edad'], bins=range(0, 100), align='left', rwidth=0.8)
plt.xlabel('Edad')
plt.ylabel('Cantidad de Personas')
plt.title('Distribución de Edades')
plt.show()
```

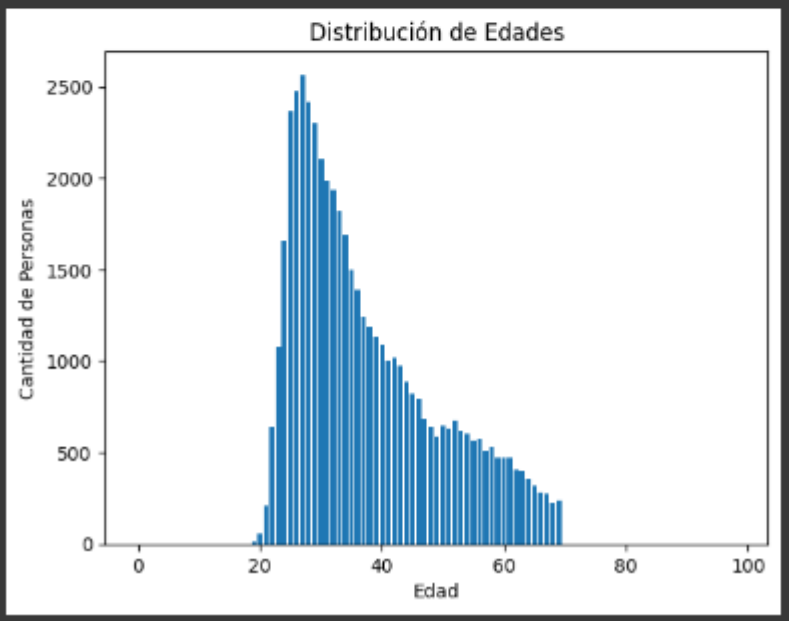



```
[67] # filtramos edaddes mayores a 80 años
     df1 = df1.query('Edad < 70')
```

```
# boxplot despues de quiar ouliers

plt.boxplot(list(df1['Edad']))
plt.show()
```




```
[69] # distribución después de quitar outliers

plt.hist(df1['Edad'], bins=range(0, 100), align='left', rwidth=0.8)
plt.xlabel('Edad')
plt.ylabel('Cantidad de Personas')
plt.title('Distribución de Edades')
plt.show()
```



Análisis de datos

Fase 3 – preparación de datos

Los datos fueron analizados utilizando las siguientes técnicas:

* Visualización de datos
* Análisis de regresión
* Análisis de correlación

Se puede apreciar el análisis de la variable Edad ademas de un histograma mostrando que la mayor cantidad de personas que son parte del banco son de entre 28 y 30 años.

Edad

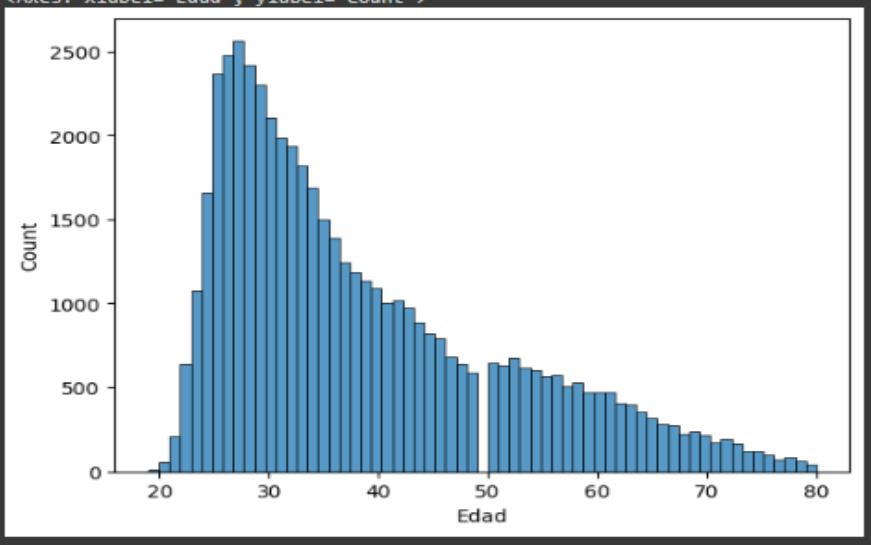
```
df1.Edad.describe()
```

	Edad
count	50892.000000
mean	38.553112
std	13.053192
min	19.000000
25%	28.000000
50%	35.000000
75%	46.000000
max	80.000000

dtype: float64

```
[32] # Histograma
sns.histplot(df1.Edad)
```

```
<Axes: xlabel='Edad', ylabel='Count'>
```



Se puede apreciar el análisis de la variable Sexo ademas de un histograma mostrando que hay mas hombre que mujeres en el banco monopoly

Sexo

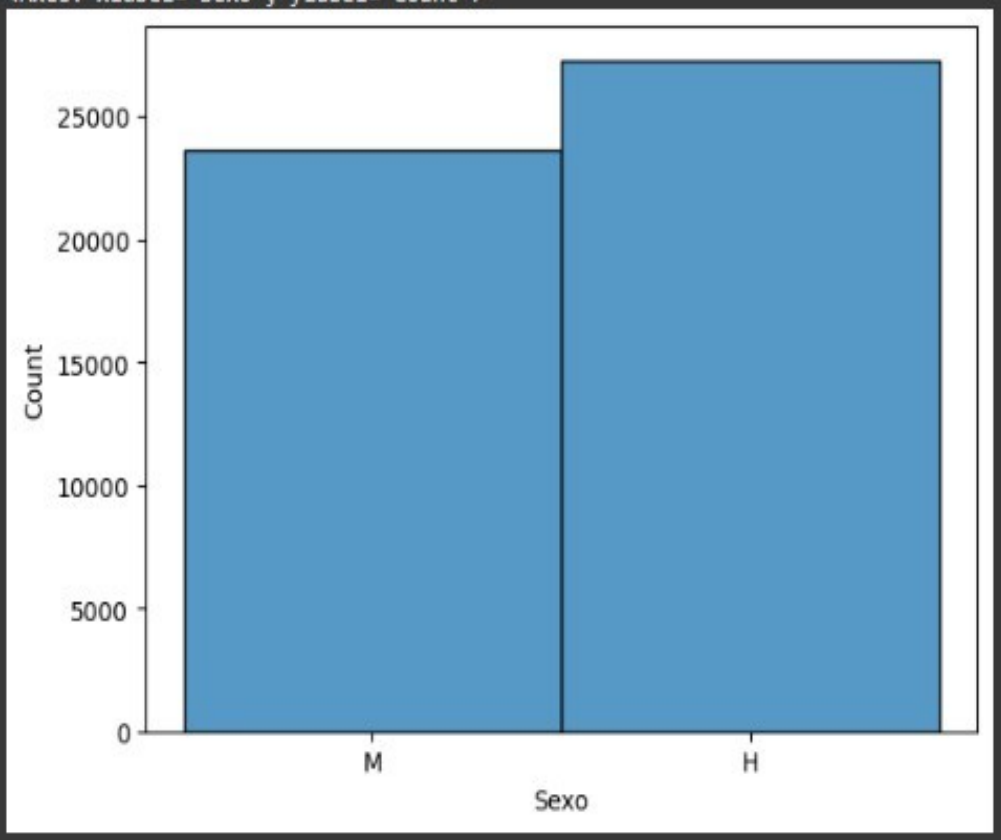
```
df1.Sexo.describe()
```

	Sexo
count	50892
unique	2
top	H
freq	27274

dtype: object

```
[34] # Histograma
sns.histplot(df1.Sexo)
```

<Axes: xlabel='Sexo', ylabel='Count'>



Se puede apreciar el análisis de la variable ctante ademas de un histograma mostrando que es 1 es que poseen una cuenta corriente y 0 que no poseen una cuenta corriente

Ctacte

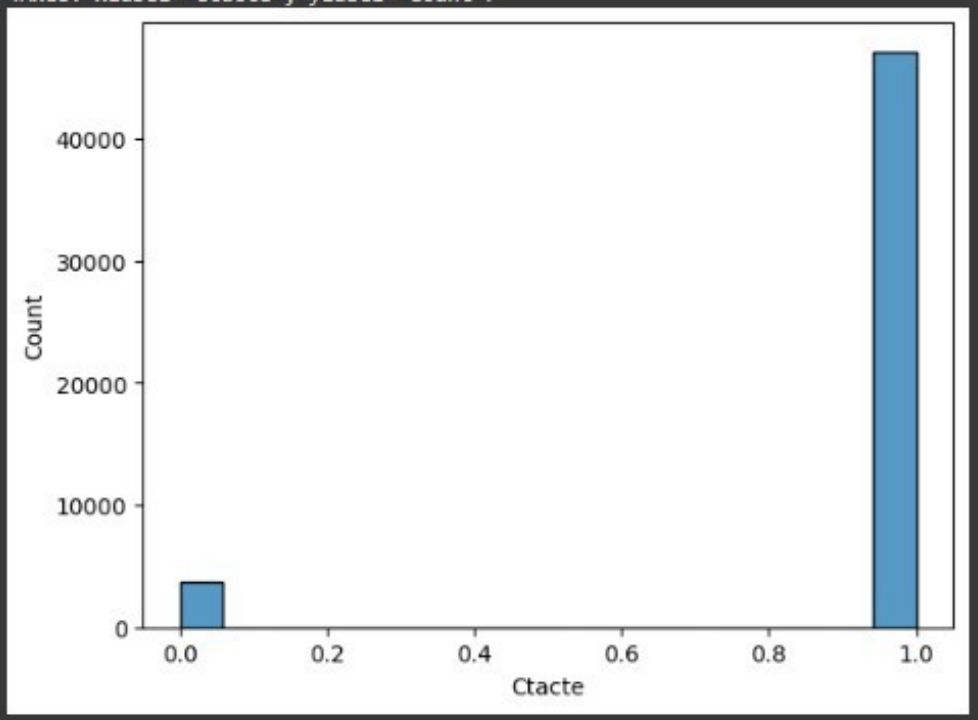
```
df1.Cuentas.describe()
```

	Cuentas
count	50892.000000
mean	1.408473
std	0.550788
min	1.000000
25%	1.000000
50%	1.000000
75%	2.000000
max	5.000000

dtype: float64

```
[42] # histograma
sns.histplot(df1.Ctacte)
```

```
<Axes: xlabel='Ctacte', ylabel='Count'>
```



transformación de la variable sexo a numérica a través de la tecnica de label encoder para que 1 se hombre y 0 mujer, tambien se creo dos tablas nuevas annual_purchases y risk_levels para el uso del modelado y se normalizo risk level para su mejor uso en los modelados, ademas se normaliza la data para su posterior uso Con modelos no supervisados

Transformation of variables

Transformation of the sex variable into numeric through the label encoder technique so that 1 is a man and 0 is a woman.

```
label_encoder = preprocessing.LabelEncoder()
df1['Sexo']= label_encoder.fit_transform(df1['Sexo'])
df1['Sexo'].unique()
```

```
array([1, 0])
```

```
#Data normalization
scaler = Normalizer()
df1_normalize = scaler.fit_transform(df1)
df1_normalize
```

```
array([[9.99999978e-01, 6.48322296e-05, 1.50772627e-06, ...,
        1.50772627e-06, 0.00000000e+00, 1.96004415e-05],
       [9.99999829e-01, 3.20245002e-04, 0.00000000e+00, ...,
        6.96184788e-06, 6.96184788e-06, 9.05040224e-05],
       [9.99999998e-01, 4.84336555e-05, 0.00000000e+00, ...,
        1.07630346e-06, 1.07630346e-06, 1.39919449e-05],
       ...,
       [9.99999980e-01, 5.82967417e-05, 0.00000000e+00, ...,
        1.24035621e-06, 0.00000000e+00, 1.61246307e-05],
       [9.99999997e-01, 7.08631360e-05, 1.50772630e-06, ...,
        1.50772630e-06, 0.00000000e+00, 1.96004419e-05],
       [9.99999994e-01, 6.06508907e-05, 0.00000000e+00, ...,
        1.18923315e-06, 0.00000000e+00, 1.54600310e-05]])
```

Creating tables for later use

The annual purchases table is created for the use of its prediction in the model

```
# Create column "Annual_Purchases" as the sum of consumption variables
df1['Annual_Purchases'] = df1['Consumo'] + df1['Debito'] + df1['Adicional'] + df1['TC']

#See the first results
print(df1[['Consumo', 'Debito', 'Adicional', 'TC', 'Annual_Purchases']].head())
```

	Consumo	Debito	Adicional	TC	Annual_Purchases
0	0	1	1	3	5
1	0	0	0	1	1
2	0	1	1	2	4
3	0	0	1	2	3
4	0	0	1	3	4

```python
#Function to calculate the risk level
def calculate_risk_level(row):
    #Define rules based on customer characteristics
    if row["Renta"] < 30000 or row["Consumo"] > 1:
        return "Alto"
    elif row["Renta"] >= 30000 and row["Renta"] < 70000 and row["Consumo"] <= 1:
        return "Medio"
    else:
        return "Bajo"

#Apply the function to each row of the DataFrame to create the new column
df1["Risk level"] = df1.apply(calculate_risk_level, axis=1)

#Show the result with the new column
print(df1)
```

```
                Renta  Edad  Sexo  Antiguedad  Consumo  Debito  Adicional  \
0      663250.351147    43     1         130        0       1          1
1      143640.000000    46     0          69        0       0          0
2      929106.000000    45     0          24        0       1          1
3      172447.000000    46     0         134        0       0          1
4      805250.000000    46     0         116        0       0          1
...              ...   ...   ...         ...      ...     ...        ...
51119  364978.000000    51     0          57        0       1          1
51120  625376.000000    51     0          39        0       1          0
51121  806220.000000    47     0         153        0       1          1
51122  663250.351147    47     1          11        0       0          0
51123  840878.000000    51     0          75        0       1          1

       Ctacte  TC  Cuentas  Hipotecario  Region  Annual_Purchases Risk level
0           1   3        1            0    13.0                 5       Bajo
1           1   1        1            1    13.0                 1       Bajo
2           1   2        1            1    13.0                 4       Bajo
3           0   2        1            0    13.0                 3       Bajo
4           1   3        2            1    13.0                 4       Bajo
...       ...  ..      ...          ...     ...               ...        ...
51119       1   3        2            0    13.0                 5       Bajo
51120       1   1        1            0    13.0                 2       Bajo
51121       1   2        1            0    13.0                 4       Bajo
51122       1   1        1            0    13.0                 1       Bajo
51123       1   2        1            0    13.0                 4       Bajo

[49548 rows x 14 columns]
```

```python
#Encode the "Risk level" column as a numeric variable
df1["Risk level"] = df1["Risk level"].map({"Alto": 2, "Medio": 1, "Bajo": 0})
```

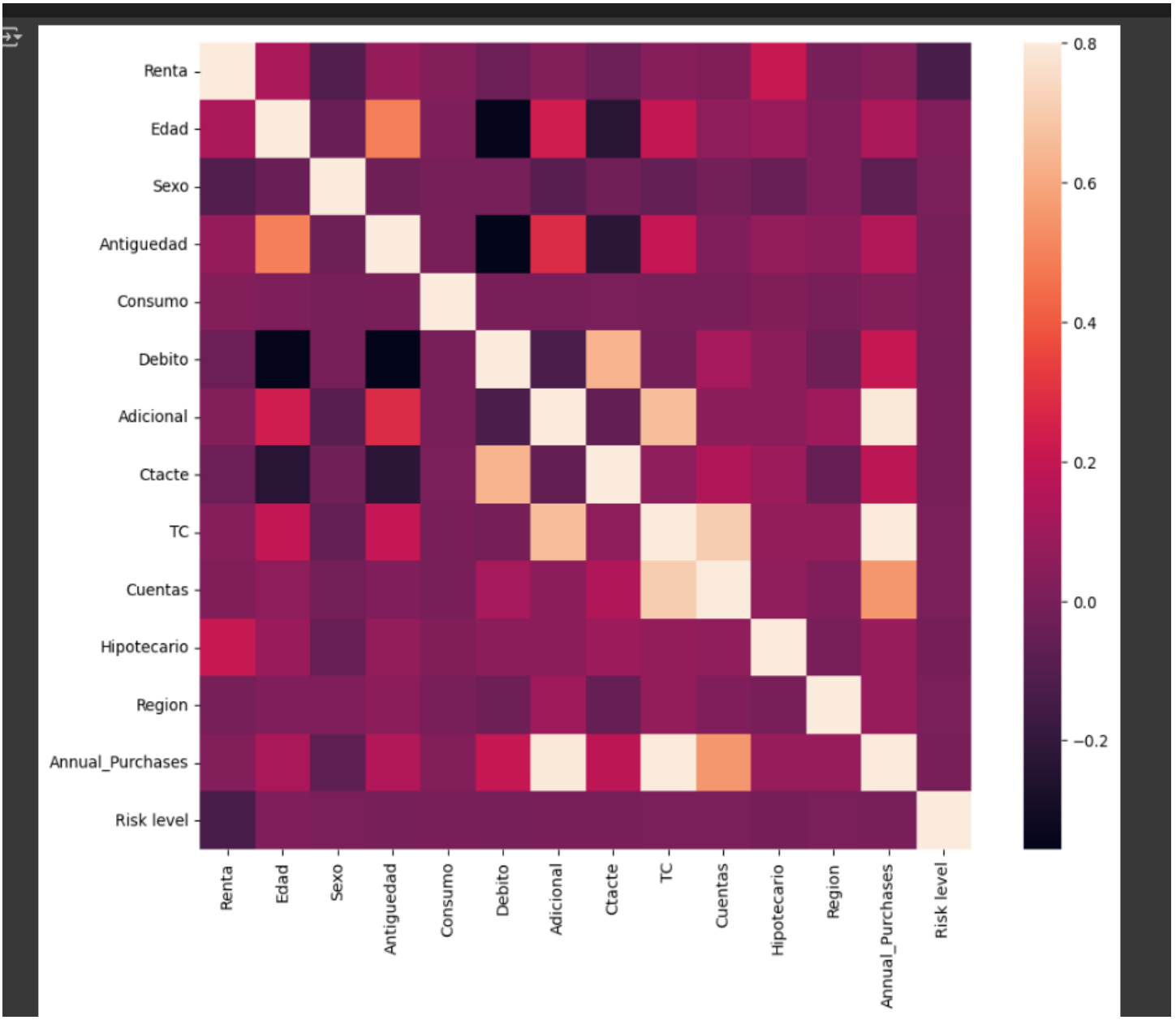
+ Código + Texto

```
[126] df1.head()
```

	Renta	Edad	Sexo	Antiguedad	Consumo	Debito	Adicional	Ctacte	TC	Cuentas	Hipotecario	Region	Annual_Purchases	Risk level
0	663250.351147	43	1	130	0	1	1	1	3	1	0	13.0	5	0
1	143640.000000	46	0	69	0	0	0	1	1	1	1	13.0	1	0
2	929106.000000	45	0	24	0	1	1	1	2	1	1	13.0	4	0
3	172447.000000	46	0	134	0	0	1	0	2	1	0	13.0	3	0
4	805250.000000	46	0	116	0	0	1	1	3	2	1	13.0	4	0

Pasos siguientes: Generar código con df1 | Ver gráficos recomendados | New interactive sheet

Se puede apreciar una matriz de correlación el cual ve que tanto tiene relación las variables entre ellas, entre valores positivos y valores negativos.



Fase 4 modelado

En esta fase veremos los distintos modelos que se hicieron,tanto modelos de regresión como de clasificación, también se vera los resultados de cada modelo y eligiendo el mejor resultado de los modelos

Se puede apreciar en la imagen que se importan las distintas librerías ademas de dividir la data de entrenamiento y prueba en este caso estamos dividiendo la data 80/20,ademas podemos apreciar la variable dependiente y la independiente.
En este caso la separación es para los modelos de regresión ya que los de clasificación los objetivos son diferente.

```
[ ]  # import the libraries to use
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.ensemble import RandomForestRegressor
     from sklearn.svm import SVR
```

```
[ ]  #Separate features and purpose
     X = df1[['Edad']]
     y = df1['Annual_Purchases']

     # Dividir en conjunto de entrenamiento y prueba
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Se puede apreciar en esta otra imagen que se cambio tanto el test_size a 0.3, lo cual se puede decir que la data esta dividida 70/30,ademas solo se bota una columna que es la objetivo,ademas se normaliza los datos de entrenamiento con MINMAXSCALER y se aplica PCA para la reducción de entidades.

```
     # import the libraries to use
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression,Ridge,Lasso
     from sklearn.ensemble import RandomForestRegressor
     from sklearn.svm import SVR
     from sklearn.decomposition import PCA
```

```
[ ]  #Separate features and purpose
     X = df1.drop(columns=['Annual_Purchases'])  #We eliminate the target column from the features dataset
     y = df1['Annual_Purchases']  #This is the variable to predict

     #Split into training and testing
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

     scaler = MinMaxScaler()

     #Normalize training data
     X_train_scaled = scaler.fit_transform(X_train)

     # Normalize test data
     X_test_scaled = scaler.transform(X_test)

     # Apply PCA to training and test data
     pca = PCA(n_components=0.95)
     X_train_pca = pca.fit_transform(X_train_scaled)
     X_test_pca = pca.transform(X_test_scaled)
```

Para los modelos de regresión se usaron los modelos de:

1)Linear regression

2)Random Forest

3)Vector Support for Regression

Como podemos apreciar esta es las primeras pruebas con los modelos de regresión lineal,random forest y vector suport para regresión, se puede ver que en términos de precisión ningún modelo es muy bueno en ello ya que en esto modelos no hay hiperparametros aplicado y por eso da tan mal resultados.

Evaluation of the linear regression model

```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Regresión Lineal - MAE: {mae}, MSE: {mse}, R²: {r2}")
```

```
Regresión Lineal - MAE: 0.9417050958846915, MSE: 1.499932295701075, R²: 0.018291244307029997
```

Evaluation of the Random Forest model

```
[86] mae_rf = mean_absolute_error(y_test, y_pred_rf)
     mse_rf = mean_squared_error(y_test, y_pred_rf)
     r2_rf = r2_score(y_test, y_pred_rf)

     print(f"Random Forest - MAE: {mae_rf}, MSE: {mse_rf}, R²: {r2_rf}")
```

```
Random Forest - MAE: 0.916625245950206, MSE: 1.4830743458067883, R²: 0.029324806996284147
```

Evaluation of the Vector Support for Regression

```
[87] # Evaluar el modelo
     mae_svr = mean_absolute_error(y_test, y_pred_svr)
     mse_svr = mean_squared_error(y_test, y_pred_svr)
     r2_svr = r2_score(y_test, y_pred_svr)

     print(f"SVR - MAE: {mae_svr}, MSE: {mse_svr}, R²: {r2_svr}")
```

```
SVR - MAE: 0.8918789687229067, MSE: 1.539386637117293, R²: -0.0075317028549266585
```

Como podemos apreciar esta es las segundas pruebas con los modelos de regresión lineal,random forest y vector support para regresión,pero en este caso esta normalizado los datos con mixmaxscaler y también esta aplicado PCA para una reducción de dimensionalidad ,pero ademas hay aplicado distintos hiperparametros o variaciones de la regresión lineal.
El mejor modelo en base a la precisión es el de vector support for regression con una precisión de 0,94.

Evaluation of the linear regression model

```
[ ]  mae = mean_absolute_error(y_test, y_pred)
     mse = mean_squared_error(y_test, y_pred)
     r2 = r2_score(y_test, y_pred)

     print(f"Regresión Lineal - MAE: {mae}, MSE: {mse}, R²: {r2}")
```

```
Regresión Lineal - MAE: 0.5011986553971476, MSE: 0.3751625811728299, R²: 0.7556794244452327
```

```
[ ]  mae = mean_absolute_error(y_test, y_pred_ridge)
     mse = mean_squared_error(y_test, y_pred_ridge)
     r2 = r2_score(y_test, y_pred_ridge)

     print(f"Regresión ridge - MAE: {mae}, MSE: {mse}, R²: {r2}")
```

```
Regresión ridge - MAE: 0.5011998357946627, MSE: 0.375162692692177, R²: 0.7556793518194634
```

```
     mae = mean_absolute_error(y_test, y_pred_lasso)
     mse = mean_squared_error(y_test, y_pred_lasso)
     r2 = r2_score(y_test, y_pred_lasso)

     print(f"Regresión lasso - MAE: {mae}, MSE: {mse}, R²: {r2}")
```

```
Regresión lasso - MAE: 0.6197141634224229, MSE: 0.6311689064887659, R²: 0.588958072461475
```

Evaluation of the Random Forest model

```
     mae_rf = mean_absolute_error(y_test, y_pred_rf)
     mse_rf = mean_squared_error(y_test, y_pred_rf)
     r2_rf = r2_score(y_test, y_pred_rf)

     print(f"Random Forest - MAE: {mae_rf}, MSE: {mse_rf}, R²: {r2_rf}")
```

```
Random Forest - MAE: 0.3988162647809196, MSE: 0.30966366997112765, R²: 0.7983348823349372
```

Evaluation of the Vector Support for Regression

```
[ ]  mae_svr = mean_absolute_error(y_test, y_pred_svr)
     mse_svr = mean_squared_error(y_test, y_pred_svr)
     r2_svr = r2_score(y_test, y_pred_svr)

     print(f"SVR - MAE: {mae_svr}, MSE: {mse_svr}, R²: {r2_svr}")
```

```
SVR - MAE: 0.12781205595864387, MSE: 0.08239739146141005, R²: 0.9463395895104234
```

Para el modelo de clasificación se hicieron dos pruebas para un mejor acuraccy,para la primera prueba se uso el modelo en distribución de 80/20.

Se puede apreciar en esta otra imagen que se cambio tanto el test_size a 0.3, lo cual se puede decir que la data esta dividida 70/30,ademas se normaliza los datos de entrenamiento con standarScaler y se utiliza técnicas de reducción de dimensionalidad como PCA y se utiliza GridSearch y SMOTE lo que ayuda a una distribución mas equitativa en los modelos.

```
[ ]  #import the libraries to use
     from sklearn.model_selection import GridSearchCV
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.svm import SVC
```

```
     # Variables predictoras
     z = df1.drop(columns=['Risk level'])  # Eliminamos la columna target del dataset de features
     i = df1['Risk level']  # Esta es la variable a predecir

     # Dividir en entrenamiento y prueba
     z_train, z_test, i_train, i_test = train_test_split(z, i, test_size=0.2, random_state=42, stratify=i)
```

```
[ ]  #Import the libraries to use
     from sklearn.model_selection import GridSearchCV,cross_val_score
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.svm import LinearSVC
     from sklearn.decomposition import PCA
     from imblearn.over_sampling import SMOTE
```

```
     z = df1.drop(columns=['Risk level'])  #We eliminate the target column from the features dataset
     i = df1['Risk level']  #This is the variable to predict

     #Split into training and testing
     z_train, z_test, i_train, i_test = train_test_split(z, i, test_size=0.3, random_state=42, stratify=i)

     # Normalize the data
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(z_train)
     X_test_scaled = scaler.transform(z_test)

     #To reduce the dimensionality of the data
     pca = PCA(n_components=0.95)
     X_train_pca = pca.fit_transform(X_train_scaled)
     X_test_pca = pca.transform(X_test_scaled)

     # Aplicar SMOTE a los datos de entrenamiento
     smote = SMOTE(random_state=42)
     X_train_smote, i_train_smote = smote.fit_resample(X_train_pca, i_train)
```

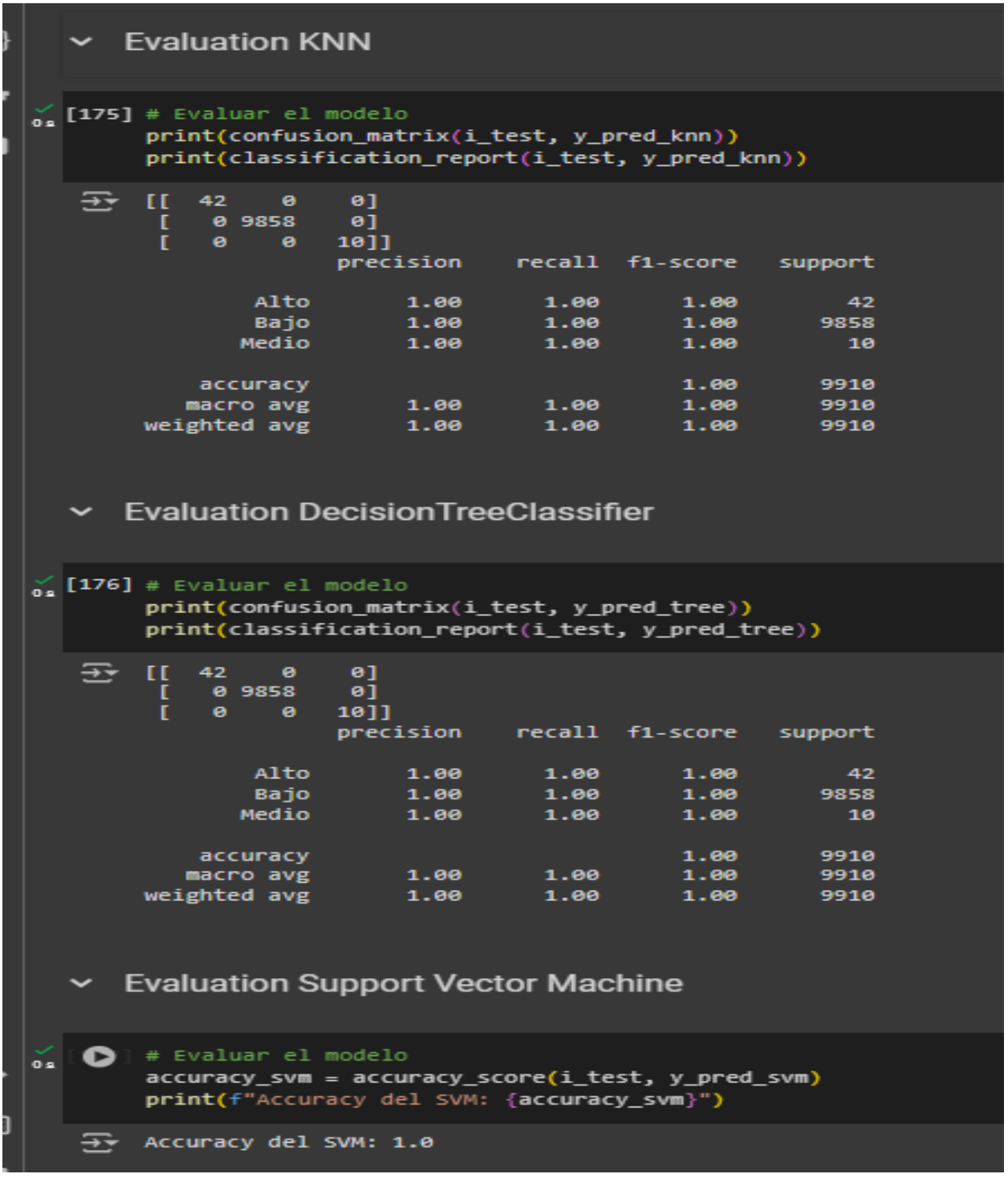
Para los modelos de clasificación se usaron los modelos de:

1)K-Nearest Neighbors

2)DecisionTreeClassifier

3)Vector Support Machine

Como se puede apreciar en esta primera prueba todo esta dando overfiting,lo cual dice que los modelos no están aprendiendo lo cual significa que hay que revisar los modelos y mejorarlos.



Resultados de la segunda prueba ahora con el modelo con estandarización ,PCA ,SMOTE y GridSeach ademas de que los modelos tiene hiperparametros en el knn se utilizo de scoring acuracy y en el random forest y en el support vector machine se uso Roc_Auc.

El mejor modelo en este caso en el support vector machine,ya que tiene un mejor rendimiento en clasificar 0,1 y 2

KNN

- Rendimiento del modelo en el conjunto de prueba: 1.00 (exactitud)
- Informe de clasificación:
- Clase 0: precisión 1.00, recall 1.00, f1-score 1.00
- Clase 1: precisión 0.13, recall 0.13, f1-score 0.13
- Clase 2: precisión 0.60, recall 0.59, f1-score 0.59
- Matriz de confusión:
- Clase 0: 14760 aciertos, 9 errores, 18 errores
- Clase 1: 6 aciertos, 2 errores, 7 errores
- Clase 2: 22 aciertos, 4 errores, 37 errores

DecisionTreeClassifier

- Rendimiento del modelo en el conjunto de prueba: 0.99 (exactitud)
- Informe de clasificación:
- Clase 0: precisión 1.00, recall 0.99, f1-score 1.00
- Clase 1: precisión 0.06, recall 0.13, f1-score 0.08
- Clase 2: precisión 0.41, recall 0.70, f1-score 0.52
- Matriz de confusión:
- Clase 0: 14703 aciertos, 30 errores, 54 errores
- Clase 1: 4 aciertos, 2 errores, 9 errores
- Clase 2: 15 aciertos, 4 errores, 44 errores

Support Vector Machine

- Rendimiento del modelo en el conjunto de prueba: 1.00 (exactitud)
- Informe de clasificación:
- Clase 0: precisión 1.00, recall 1.00, f1-score 1.00
- Clase 1: precisión 0.62, recall 0.87, f1-score 0.72
- Clase 2: precisión 0.97, recall 0.92, f1-score 0.94
- Matriz de confusión:
- Clase 0: 14784 aciertos, 3 errores, 0 errores
- Clase 1: 0 aciertos, 13 aciertos, 2 errores
- Clase 2: 0 aciertos, 5 errores, 58 aciertos

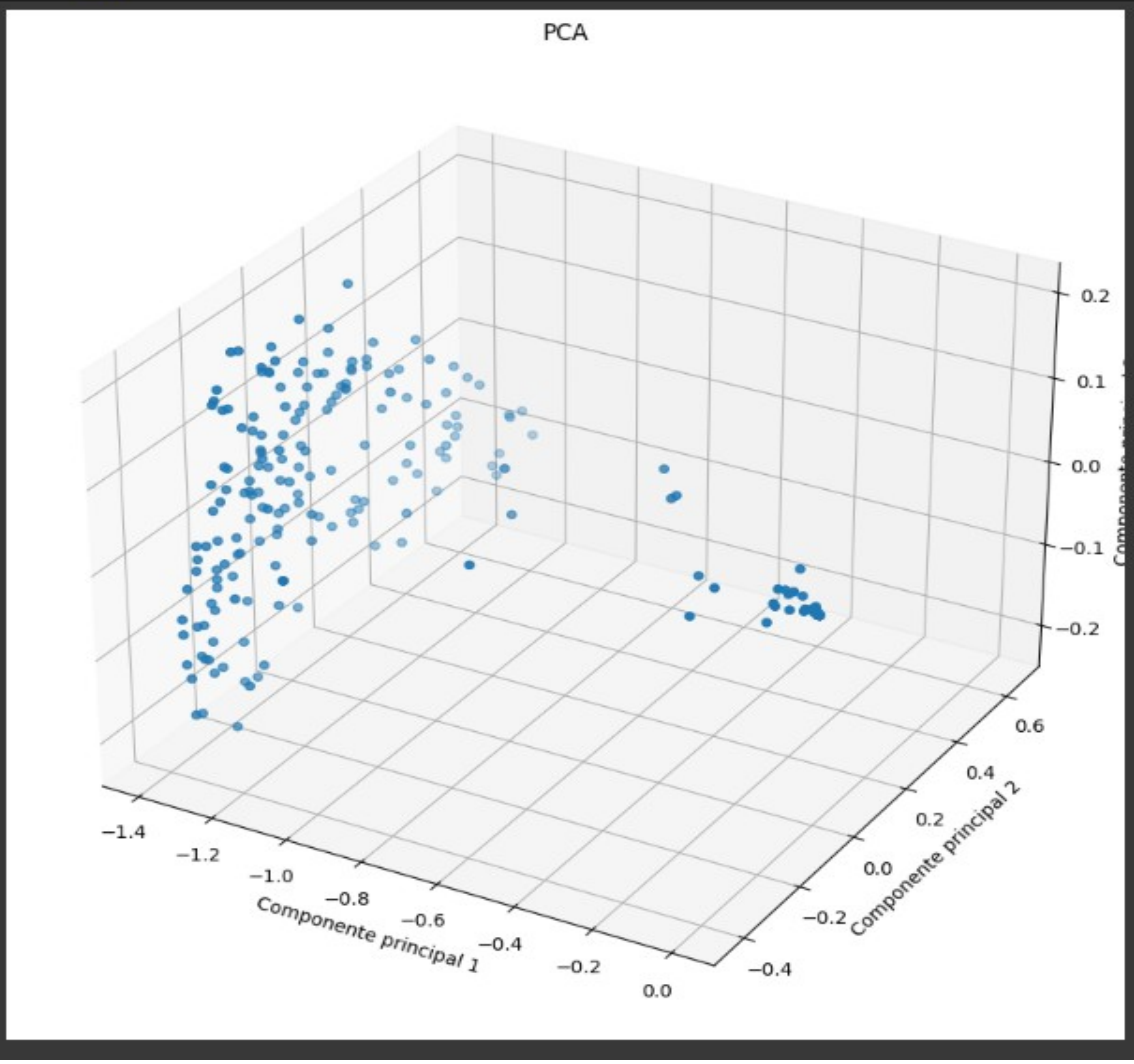
Modelo no supervisado

PCA

Uso de PCA para reducir dimensionalidad para mejorar la estructura de datos y los resultados a la hora del entrenamiento del modelo

```
[67] #se selecciona el número de componentes principales como en este caso tenemos 12 variables vamos a elegir 3 componentes principales para evitar el sobreajuste
     pca = PCA(n_components=3)

[68] df1_pca = pca.fit_transform(df1_normalize)
```

```
fig = plt.figure(figsize=(10, 12))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(df1_pca[:, 0], df1_pca[:, 1], df1_pca[:, 2])
ax.set_xlabel('Componente principal 1')
ax.set_ylabel('Componente principal 2')
ax.set_zlabel('Componente principal 3')
ax.set_title('PCA')

plt.show()
```




```
[73] print(pca.components_)

[[ 7.04787790e-01 -5.61923173e-01 -7.84105438e-03 -3.97636313e-01
   9.01383651e-10 -1.37593098e-02 -2.72619006e-03 -1.44469576e-02
  -2.57812285e-02 -2.22462095e-02 -1.13945700e-03 -1.66655539e-01]
 [-3.96741493e-02 -5.84792493e-01 -2.78423955e-03  8.02824046e-01
  -7.48694100e-11 -1.54589366e-02  7.45506718e-04 -1.41536344e-02
  -2.11904684e-02 -2.14221009e-02  8.95512149e-04 -1.02766221e-01]
 [ 1.25204525e-01 -1.56480020e-01  3.01259163e-02  2.25624700e-02
  -2.13828920e-10  5.48849414e-02  1.58047609e-02  5.03197160e-02
   9.61826579e-02  6.54150924e-02  4.61553337e-03  9.69059045e-01]]
```

Se utiliza kmeans como visualización y uso de los datos del pca para su separación de los datos y agrupación

K-means

```
[81] # Utiliza los datos del PCA como entrada para k-means
     kmeans = KMeans(n_clusters=3)
     kmeans.fit(df1_pca)
```

```
KMeans
KMeans(n_clusters=3)
```

```
[82] # Obtén los centroides de los clusters y las etiquetas de los datos
     centroides = kmeans.cluster_centers_
     etiquetas = kmeans.labels_
```

```
[ ] # Muestra los datos de k-means por pantalla
    print("Centroides de los clusters:")
    print(centroides)
```

```
Centroides de los clusters:
[[ 4.62648402e-03 -2.32640999e-06  3.76503668e-06]
 [-1.34542773e+00 -1.76109058e-01  2.31687949e-03]
 [-1.31961438e+00  3.03724593e-01 -6.92268573e-03]]
```

```
[84] print("\nEtiquetas de los datos:")
     print(etiquetas)
```

```
Etiquetas de los datos:
[0 0 0 ... 0 0 0]
```

```
# Muestra un gráfico de dispersión de los datos de k-means
plt.scatter(df1_pca[:, 0], df1_pca[:, 1], c=etiquetas)
plt.scatter(centroides[:, 0], centroides[:, 1], c='red', marker='*', s=200)
plt.title("K-means")
plt.show()
```