SANTANDER PRODUCT RECOMMENDATION

By Sarah Esayas

OBJECTIVE

With 1.5 years of customers' behavior data from Santander Bank, as well as demographic data, we are to predict what kind of product customers would purchase.

Products ranged from payroll accounts to long-term deposits to loans and credit/debit cards.

FIRST LOOK AT OUR DATA…

	fecha_dato	ncodpers	ind_empleado	pais_residencia	sexo	age	fecha_alta	ind_nuevo	antiguedad	indrel	...	ind_hip_fin_ult1
0	2015-01-28	1375586	N	ES	H	35	2015-01-12	0.0	6	1.0	...	0
1	2015-01-28	1050611	N	ES	V	23	2012-08-10	0.0	35	1.0	...	0
2	2015-01-28	1050612	N	ES	V	23	2012-08-10	0.0	35	1.0	...	0
3	2015-01-28	1050613	N	ES	H	22	2012-08-10	0.0	35	1.0	...	0
4	2015-01-28	1050614	N	ES	V	23	2012-08-10	0.0	35	1.0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
99994	2015-01-28	890570	N	ES	V	24	2010-09-02	0.0	58	1.0	...	0
99995	2015-01-28	890561	N	ES	H	25	2010-09-02	0.0	58	1.0	...	0
99996	2015-01-28	890559	N	ES	H	48	2010-09-03	0.0	58	1.0	...	0
99997	2015-01-28	890557	N	ES	V	53	2010-09-02	0.0	58	1.0	...	0
99998	2015-01-28	890498	N	ES	V	47	2010-09-02	0.0	58	1.0	...	0

99999 rows × 48 columns

d_pres_fin_ult1	ind_reca_fin_ult1	ind_tjcr_fin_ult1	ind_valo_fin_ult1	ind_viv_fin_ult1	ind_nomina_ult1	ind_nom_pens_ult1	ind_recibo_ult1
0	0	0	0	0	0.0	0.0	0
0	0	0	0	0	0.0	0.0	0
0	0	0	0	0	0.0	0.0	0
0	0	0	0	0	0.0	0.0	0
0	0	0	0	0	0.0	0.0	0
...	...	...	...	...	...	...	...
0	0	0	0	0	0.0	0.0	1
0	0	0	0	0	0.0	0.0	0
0	0	0	0	0	0.0	0.0	1
0	0	0	0	0	0.0	0.0	0
0	0	0	0	0	0.0	0.0	0

UNDERSTANDING THE DATA

In [2]:

```python
df.dtypes
```

In [3]:

```python
# Let's check if our values corresponds with the data type
for i in df.columns:
    print(f"{i}\n", df[i].unique(), "\n", f"Values = {df[i].unique().size}\n")
```

Column Name	Description	Data Type
fecha_dato FIXED	Data starts at 01/28/2015	~~object~~ DATETIME
ncodpers	Customer code	int64
ind_empleado	0 = A = active 1 = B = ex employed 2= F = subsidiary 3 = N = not employee 4 = P = passive (?)	object
pais_residencia	Customer's Country residence 114 different countries 0 = 'BO' 1 = 'DE' 2 = 'ES' 3 = 'PY'	object
sexo	0 = H = Male 1 = V = Female	object
age FIXED	Age 115 different ages	~~object~~ FLOAT64
fecha_alta FIXED	The date in which the customer became as the first holder of a contract in the bank	~~object~~ DATE/TIME
~~ind_nuevo~~ ~~FIXED~~	New customer Index. 0 = Old customer 1 = New customer if the customer registered in the last 6 months.	~~float64~~ OBJECT
antiguedad	Customer seniority (in months) ??? Does this mean how long they've had it? 249 different numbers	~~object~~ INT64 ?
indrel FIXED	Indicador de Relación (Relationship Indicator) 1 (First/Primary) 99 (Primary customer during the month but not at the end of the month)	~~float64~~ OBJECT
~~ult_fec_cli_1t~~ ~~FIXED~~	~~Last date as primary customer (if he isn't at the end of the month)~~	~~object~~ ~~DATE TIME~~
indrel_1mes FIXED	Customer type at the beginning of the month , 1 (First/Primary customer), 2 (co-owner),	~~float64~~ OBJECT
	1 = I (inactive), P (former customer), R (Potential)	
indresi	Residence index 1 = S (Yes) or 0 = N (No) if the residence country is the same than the bank country)	object
indext	Foreigner index 1 = (S (Yes) or 0 = N (No) if the customer's birth country is different than the bank country)	object
~~conyuemp~~	~~Spouse index.~~ ~~N No~~ ~~S Yes~~ ~~if the customer is spouse of an employee~~	~~object~~
canal_entrada	channel used by the customer to join There are 157 different categories	object
indfall	Deceased index. N/S 0 = N 1 = S	object
tipodom	Address type. 1, primary address Everyone put a primary address	~~float64~~ OBJECT
cod_prov	Province code (customer's address) 53 different province codes	float64
nomprov	Province name 53 different province names	object
ind_actividad_cliente FIXED	Activity index 1, active customer 0, inactive customer	~~float64~~ OBJECT
renta	Gross income of the household mean = $139,646,2 sd = 2.389858e+05 min = 1.202730e+03 max = 2.889440e+07	float64
segmento	segmentation: 0 - VIP, 1 - Individuals / Particulares 2 - college graduated nan	object

ind_ahor_fin_ult1	Saving Account	object
ind_aval_fin_ult1	Guarantees	object
ind_cco_fin_ult1	Current Accounts	object
ind_cder_fin_ult1	Derivada Account	object
ind_cno_fin_ult1	Payroll Account	object
ind_ctju_fin_ult1	Junior Account	object
ind_ctma_fin_ult1	Más particular Account	object
ind_ctop_fin_ult1	particular Account	object
ind_ctpp_fin_ult1	particular Plus Account	object
ind_deco_fin_ult1	Short-term deposits	object
ind_deme_fin_ult1	Medium-term deposits	object
ind_dela_fin_ult1	Long-term deposits	object
ind_ecue_fin_ult1	e-account	object
ind_fond_fin_ult1	Funds	object
ind_hip_fin_ult1	Mortgage	object
ind_plan_fin_ult1	Pensions	object
ind_pres_fin_ult1	Loans	object
ind_reca_fin_ult1	Taxes	object
ind_tjcr_fin_ult1	Credit Card	object
ind_valo_fin_ult1	Securities	object
ind_viv_fin_ult1	Home Account	object
ind_nomina_ult1	Payroll ** nan values	object
ind_nom_pens_ult1	Pensions ** nan values	object
ind_recibo_ult1	Direct Debit	object

ON TO DATA PREPROCESSING!

FIRST STEP, CHANGING DATA TYPES:

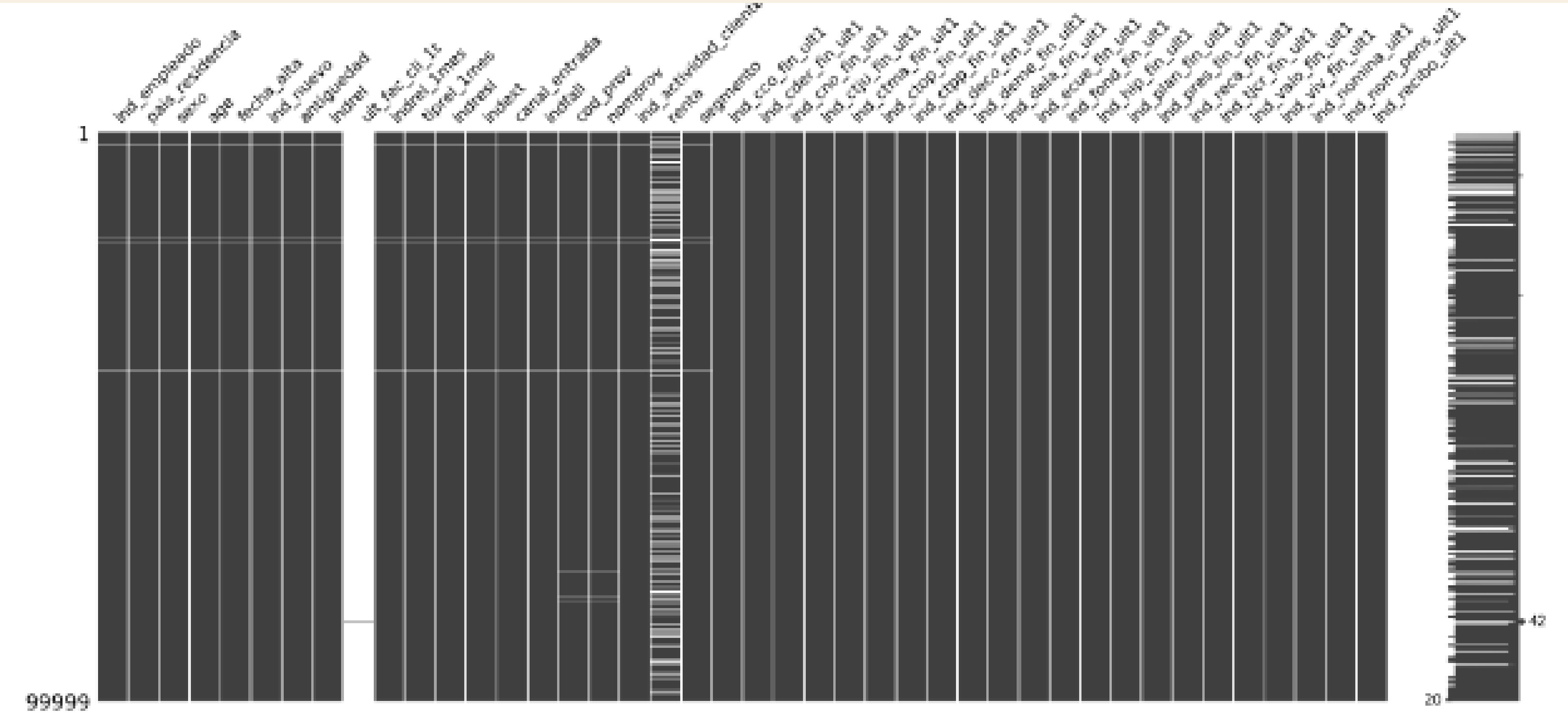
In [5]:

```
# CHANGING DATA TYPES

# Dates
df["fecha_alta"]=pd.to_datetime(df["fecha_alta"])
df["ult_fec_cli_1t"]=pd.to_datetime(df["ult_fec_cli_1t"])

# Numeric
df.loc[df['antiguedad']=='     NA','antiguedad']=None
df["antiguedad"]=pd.to_numeric(df["antiguedad"])
df.loc[df['age']==' NA','age']=None
df["age"]=pd.to_numeric(df["age"])
```

NEXT – MISSING DATA!

```
In [6]:  # MISSING DATA

         missing_data=df.isnull().sum()
         print("Missing data:\n", missing_data)
```

```
Missing data:
 ind_empleado              683
pais_residencia           683
sexo                      683
age                       683
fecha_alta                683
ind_nuevo                 683
antiguedad                683
indrel                    683
ult_fec_cli_1t          99871
indrel_1mes               683
tiprel_1mes               683
indresi                   683
indext                    683
canal_entrada             688
indfall                   683
cod_prov                  769
nomprov                   769
ind_actividad_cliente     683
renta                   18283
segmento                  691
ind_cco_fin_ult1            0
ind_cder_fin_ult1           0
ind_cno_fin_ult1            0
ind_ctju_fin_ult1           0
ind_ctma_fin_ult1           0
ind_ctop_fin_ult1           0
ind_ctpp_fin_ult1           0
ind_deco_fin_ult1           0
ind_deme_fin_ult1           0
ind_dela_fin_ult1           0
ind_ecue_fin_ult1           0
ind_fond_fin_ult1           0
ind_hip_fin_ult1            0
ind_plan_fin_ult1           0
ind_pres_fin_ult1           0
ind_reca_fin_ult1           0
ind_tjcr_fin_ult1           0
ind_valo_fin_ult1           0
ind_viv_fin_ult1            0
ind_nomina_ult1           210
ind_nom_pens_ult1         210
ind_recibo_ult1             0
dtype: int64
```




```
In [8]:  df.drop('ult_fec_cli_1t', axis=1, inplace=True)
```

```
In [9]:
# Let's removing the rows with missing values

missing = ['ind_empleado', 'pais_residencia', 'sexo',
           'age', 'fecha_alta', 'ind_nuevo', 'antiguedad', 'indrel',
           'indrel_1mes', 'tiprel_1mes', 'indresi', 'indext',
           'canal_entrada', 'indfall', 'cod_prov',
           'nomprov', 'ind_actividad_cliente', 'segmento', 'ind_nomina_ult1','ind_nom_pens_ult1']

df = df.dropna(subset=missing)
```

Did not delete missing rows of 'renta', instead used implementation with median:

```
In [11]:
# Let's replace renta's missing with its median

median_renta = df['renta'].median()

df['renta'].fillna(median_renta, inplace=True)
```

DEALING WITH DUPLICATES:

```
In [12]: duplicates=df.duplicated().sum()
         print("Number of Duplicate Records: ", duplicates)

Number of Duplicate Records:  2722

In [13]: df=df.drop_duplicates()

In [14]: duplicates=df.duplicated().sum()
         print("Number of Duplicate Records: ", duplicates)

Number of Duplicate Records:  0
```

INVALID ENTRIES AND OUTLIERS:

In [15]:

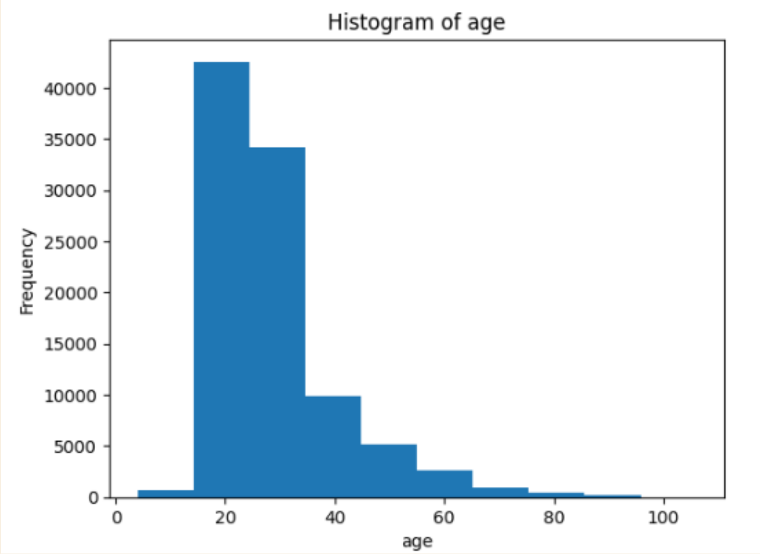
```
features = ['ind_empleado', 'pais_residencia', 'sexo', 'age', 'fecha_alta',
        'ind_nuevo', 'antiguedad', 'indrel', 'indrel_1mes', 'tiprel_1mes',
        'indresi', 'indext', 'canal_entrada', 'indfall', 'cod_prov', 'nomprov',
        'ind_actividad_cliente', 'renta', 'segmento']

df[features].describe()
```

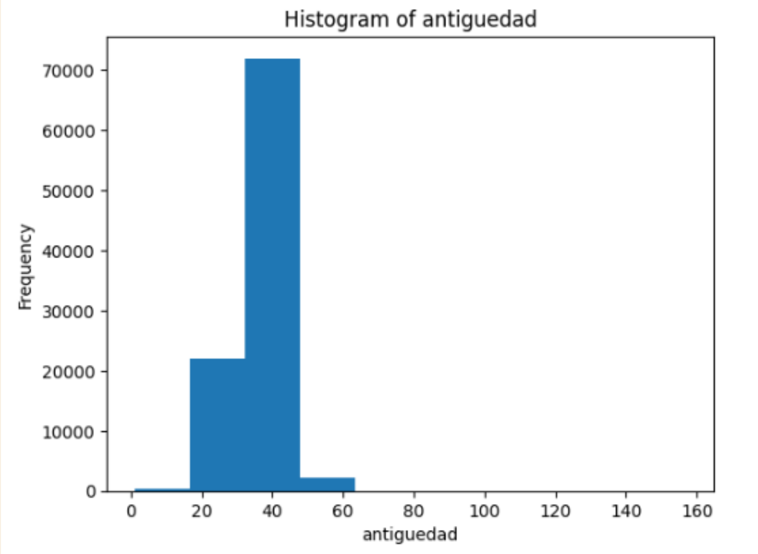
[15]:

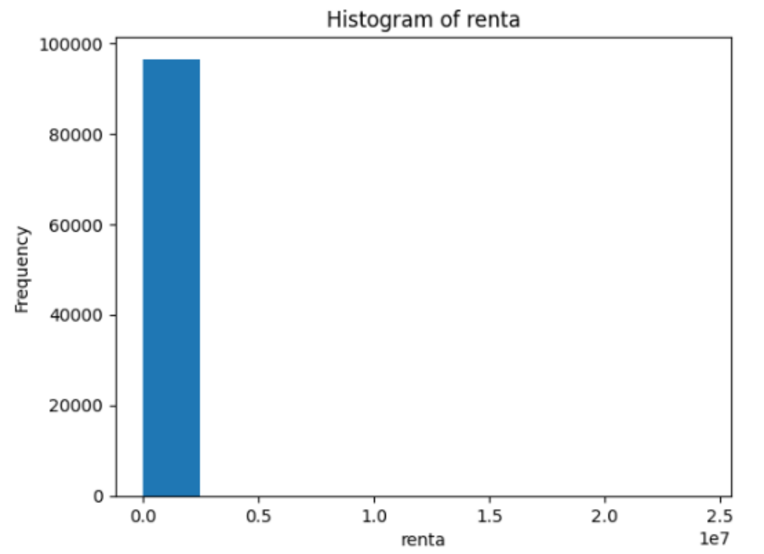
	age	fecha_alta	ind_nuevo	antiguedad	indrel	indrel_1mes	cod_prov	ind_actividad_cliente	renta
count	96495.000000	96495	96495.000000	96495.000000	96495.000000	96495.0	96495.000000	96495.000000	9.649500e+04
mean	29.492481	2012-06-30 21:50:33.446292736	0.000114	36.378776	1.126950	1.0	25.039038	0.417027	1.115668e+05
min	4.000000	2002-06-14 00:00:00	0.000000	1.000000	1.000000	1.0	1.000000	0.000000	2.539800e+03
25%	23.000000	2012-06-18 00:00:00	0.000000	33.000000	1.000000	1.0	11.000000	0.000000	6.683498e+04
50%	25.000000	2012-09-06 00:00:00	0.000000	34.000000	1.000000	1.0	28.000000	0.000000	8.961030e+04
75%	31.000000	2012-10-25 00:00:00	0.000000	37.000000	1.000000	1.0	36.000000	1.000000	1.222534e+05
max	106.000000	2015-01-26 00:00:00	1.000000	157.000000	99.000000	1.0	52.000000	1.000000	2.425324e+07
std	10.649402	NaN	0.010676	6.602483	3.524922	0.0	13.656111	0.493070	1.469700e+05

Indrel_1mes had only 1 value after cleaning, so I dropped this column

OUTLIERS








```python
[18]: # Checking for outliers

def detect_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)
    return (column < lower_bound) | (column > upper_bound)

indep_n = ['age', 'antiguedad', 'renta']

for col in indep_n:
    outliers = detect_outliers(df[col])
    num_outliers = outliers.sum()
    total_values = len(df[col])
    percentage_outliers = (num_outliers / total_values) * 100
    print(f"{col} Outliers: {num_outliers} ({percentage_outliers:.2f}%)\n")
```

```
age Outliers: 10387 (10.76%)

antiguedad Outliers: 21906 (22.70%)

renta Outliers: 7638 (7.92%)
```

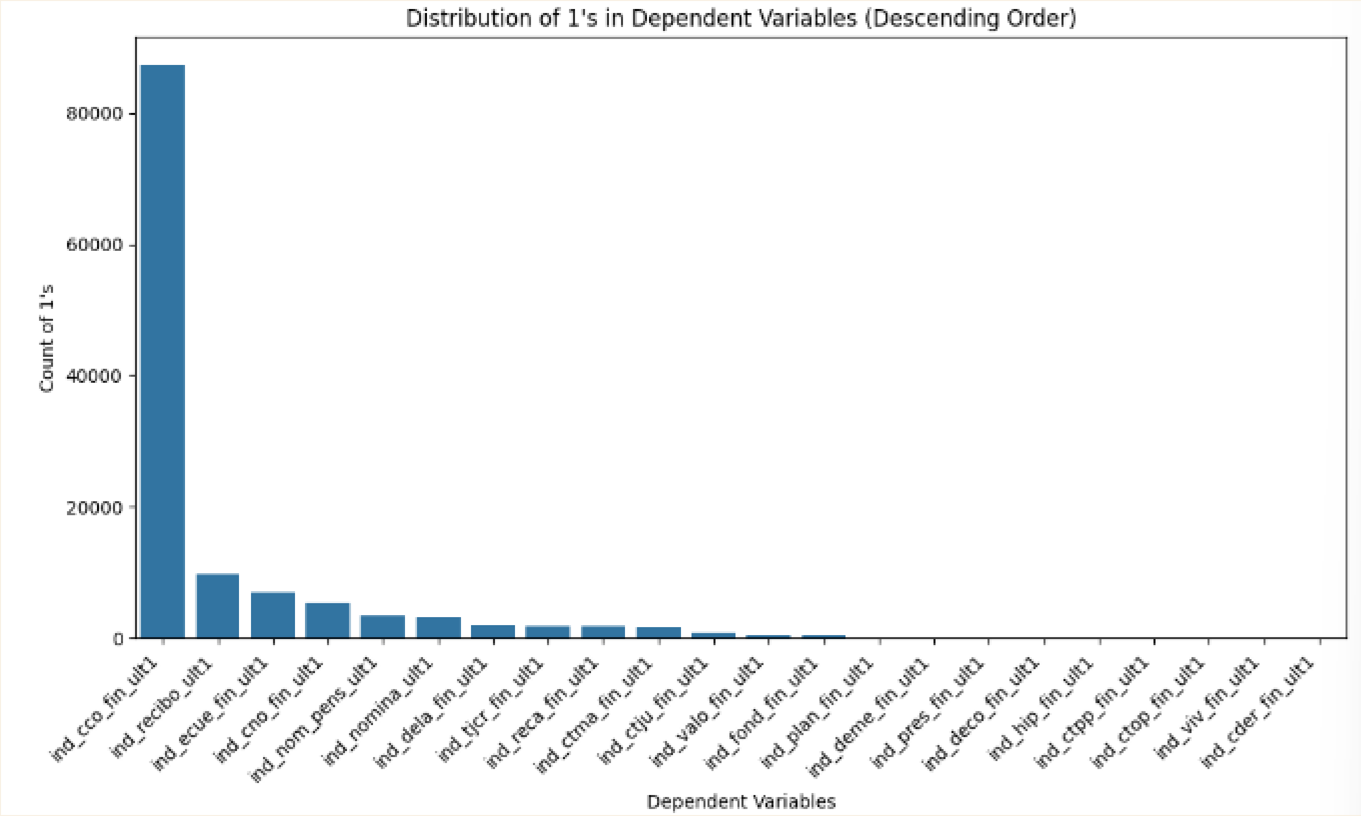
OUTLIERS

```
[25]: def handle_outliers(col):
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - (1.5 * IQR)
        upper_bound = Q3 + (1.5 * IQR)

        df.loc[df[col]>upper_bound, col] = Q3
        df.loc[df[col]<lower_bound, col] = Q1

      for col in indep_n:
        handle_outliers(col)

      for col in indep_n:
        d = detect_outliers(df[col]).sum()
        print(f"{col} Outliers: {d}\n")
```

```
age Outliers: 0

antiguedad Outliers: 0

renta Outliers: 0
```

DATA VISUALIZATION

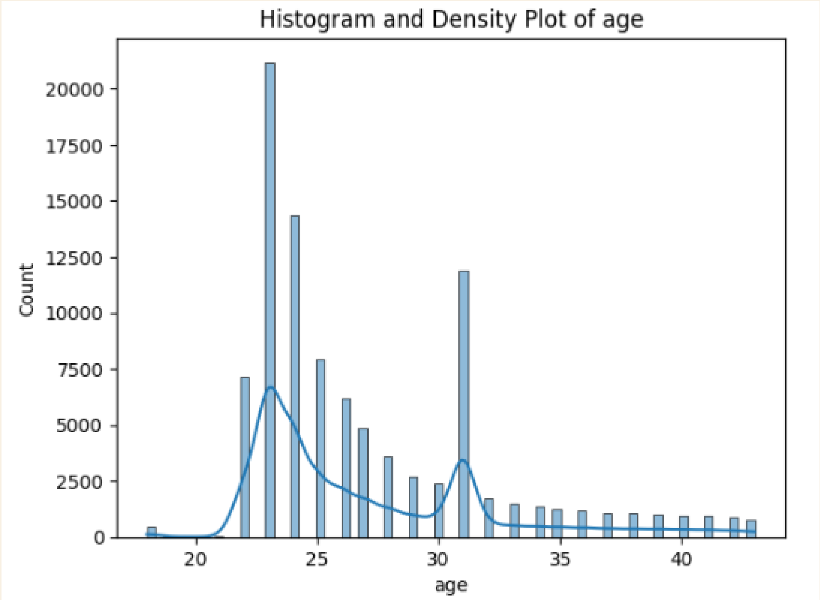
CHECKING TO SEE HOW MANY PRODUCTS WERE BOUGHT:

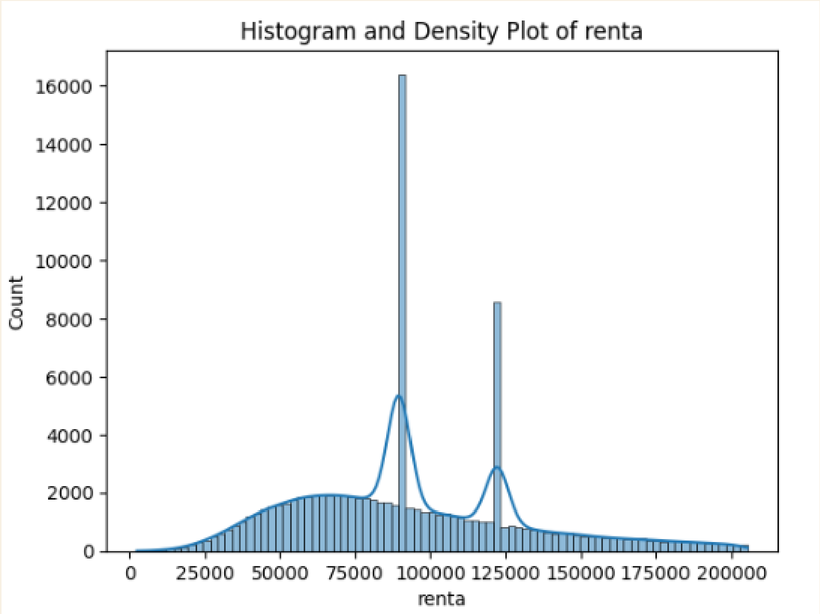


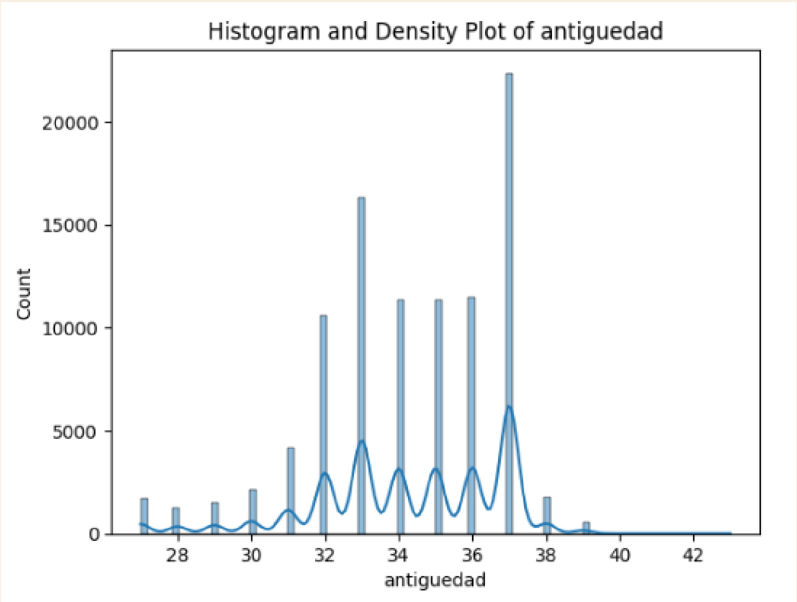

```
ind_cco_fin_ult1     87193.0
ind_recibo_ult1       9695.0
ind_ecue_fin_ult1     7053.0
ind_cno_fin_ult1      5355.0
ind_nom_pens_ult1     3522.0
ind_nomina_ult1       3268.0
ind_dela_fin_ult1     2087.0
ind_tjcr_fin_ult1     1851.0
ind_reca_fin_ult1     1827.0
ind_ctma_fin_ult1     1576.0
ind_ctju_fin_ult1      925.0
ind_valo_fin_ult1      456.0
ind_fond_fin_ult1      370.0
ind_plan_fin_ult1      126.0
ind_deme_fin_ult1       34.0
ind_pres_fin_ult1       19.0
ind_deco_fin_ult1       18.0
ind_hip_fin_ult1        11.0
ind_ctpp_fin_ult1       10.0
ind_ctop_fin_ult1        6.0
ind_viv_fin_ult1         6.0
ind_cder_fin_ult1        4.0
dtype: float64
```

Then chose to focus on products that had > 1000 counts.

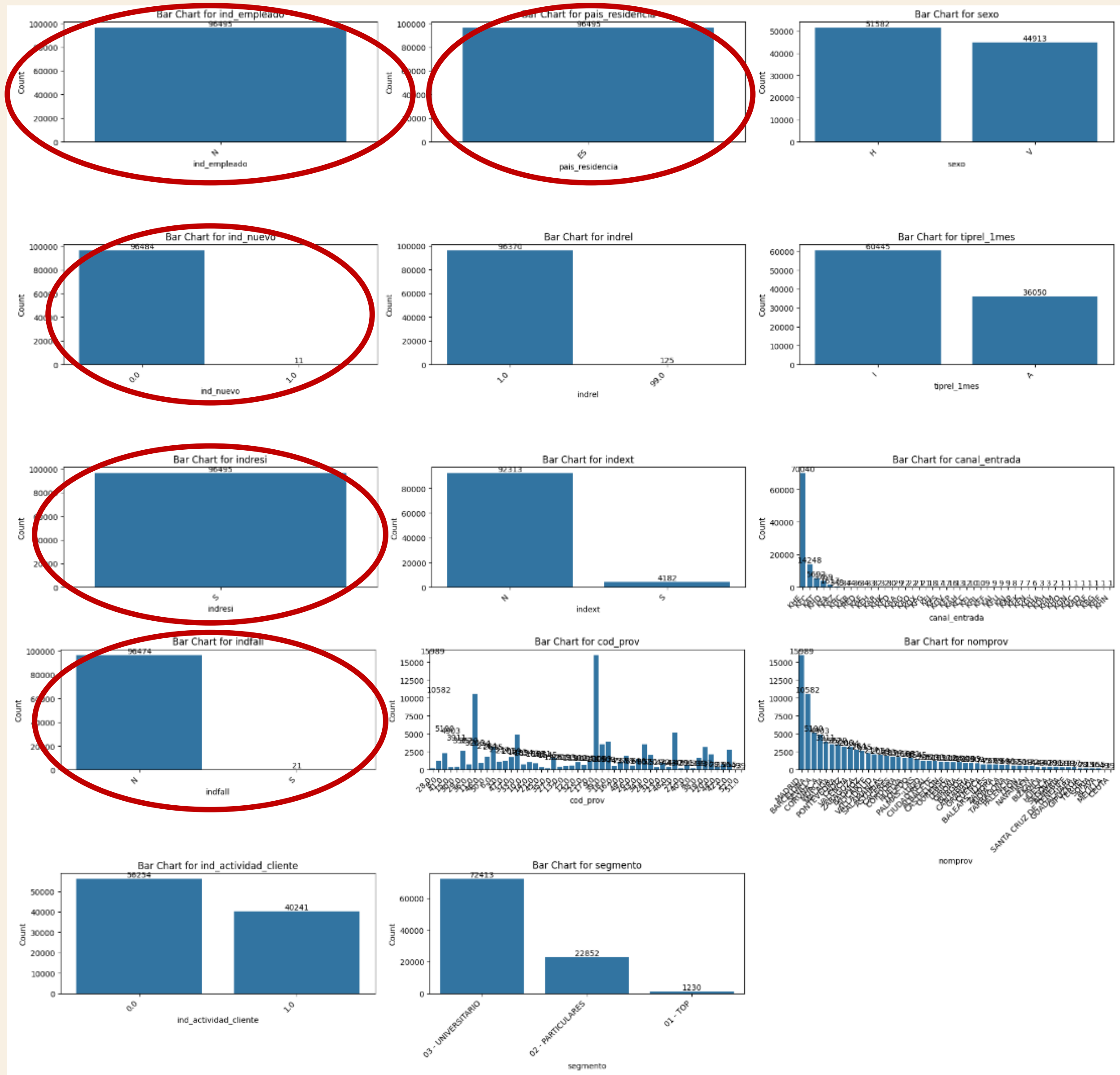
CHECKING FREQUENCY OF NUMERICAL COLUMNS:



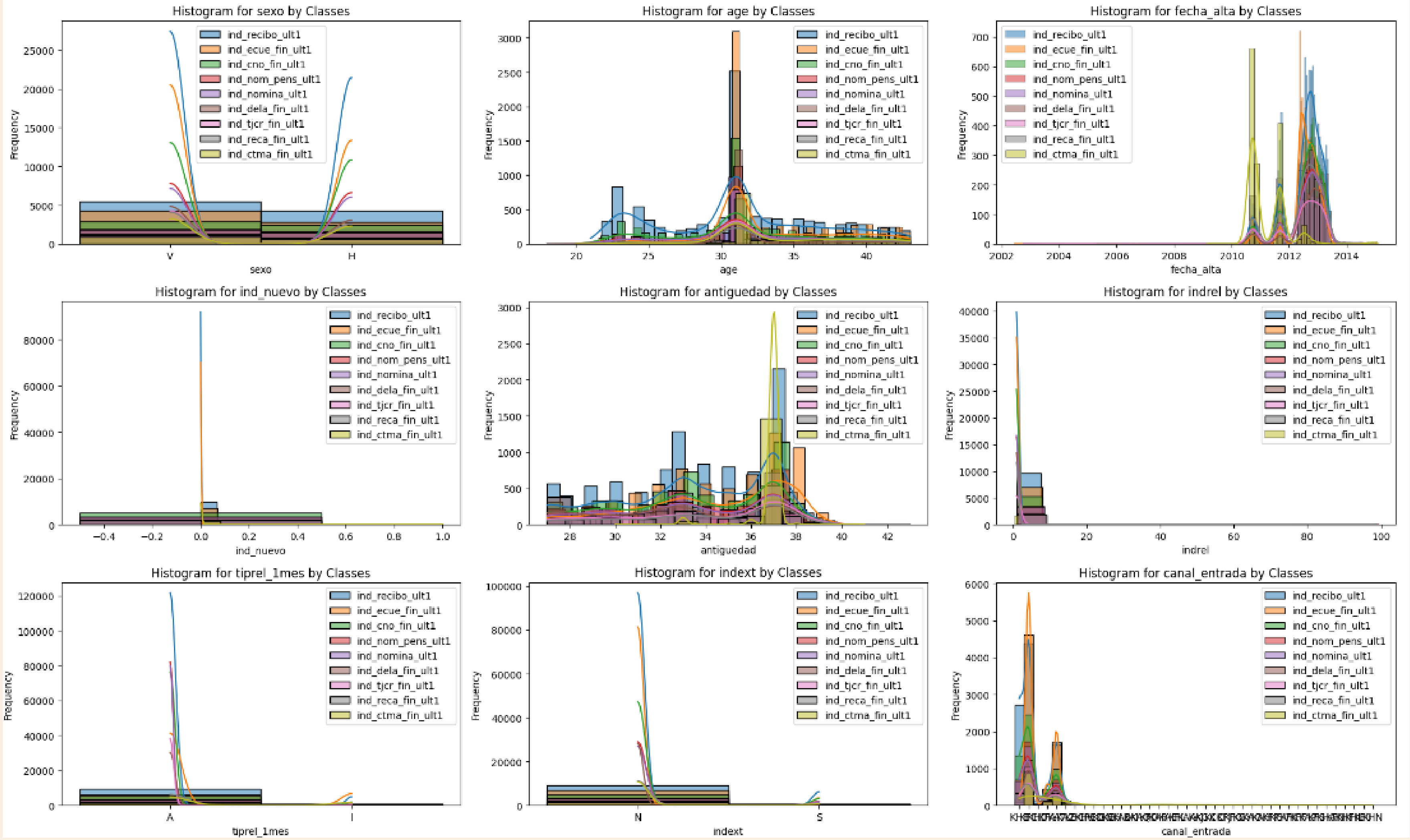


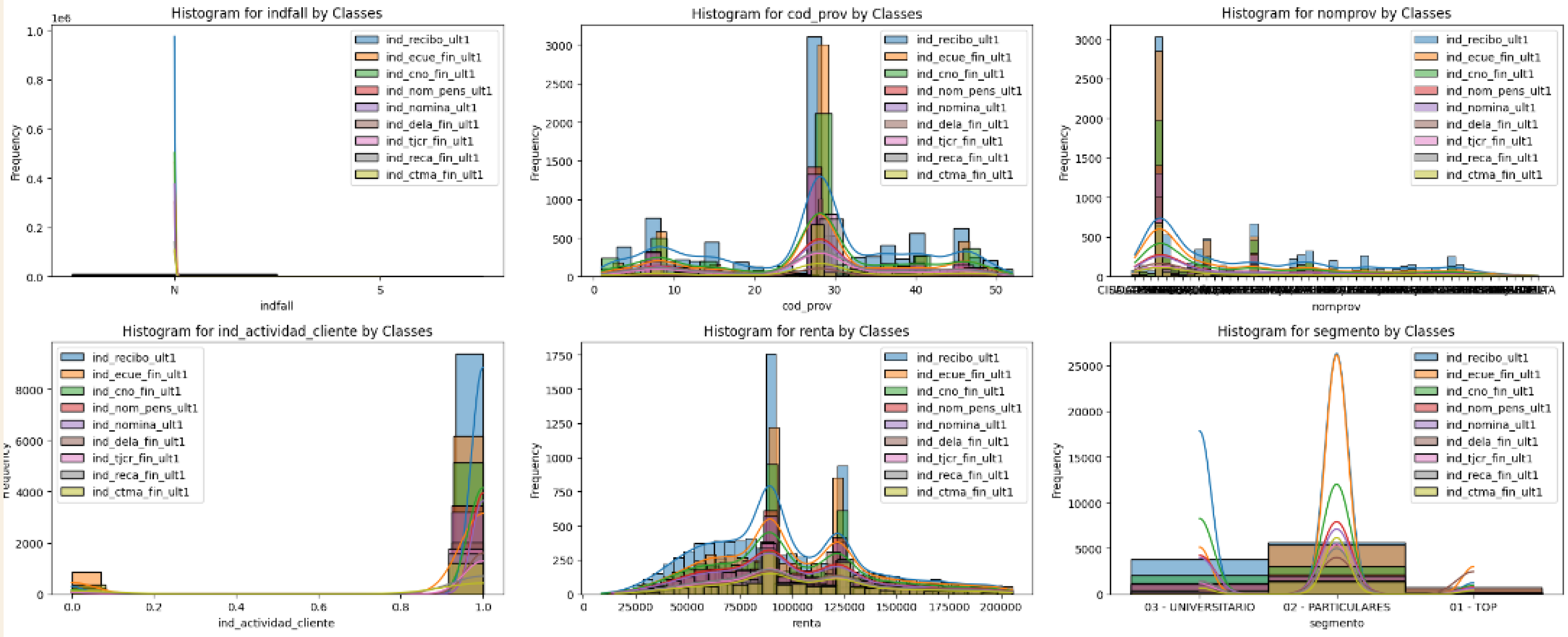


CATEGORICAL COLUMNS:



VISUALIZING RELATIONSHIPS

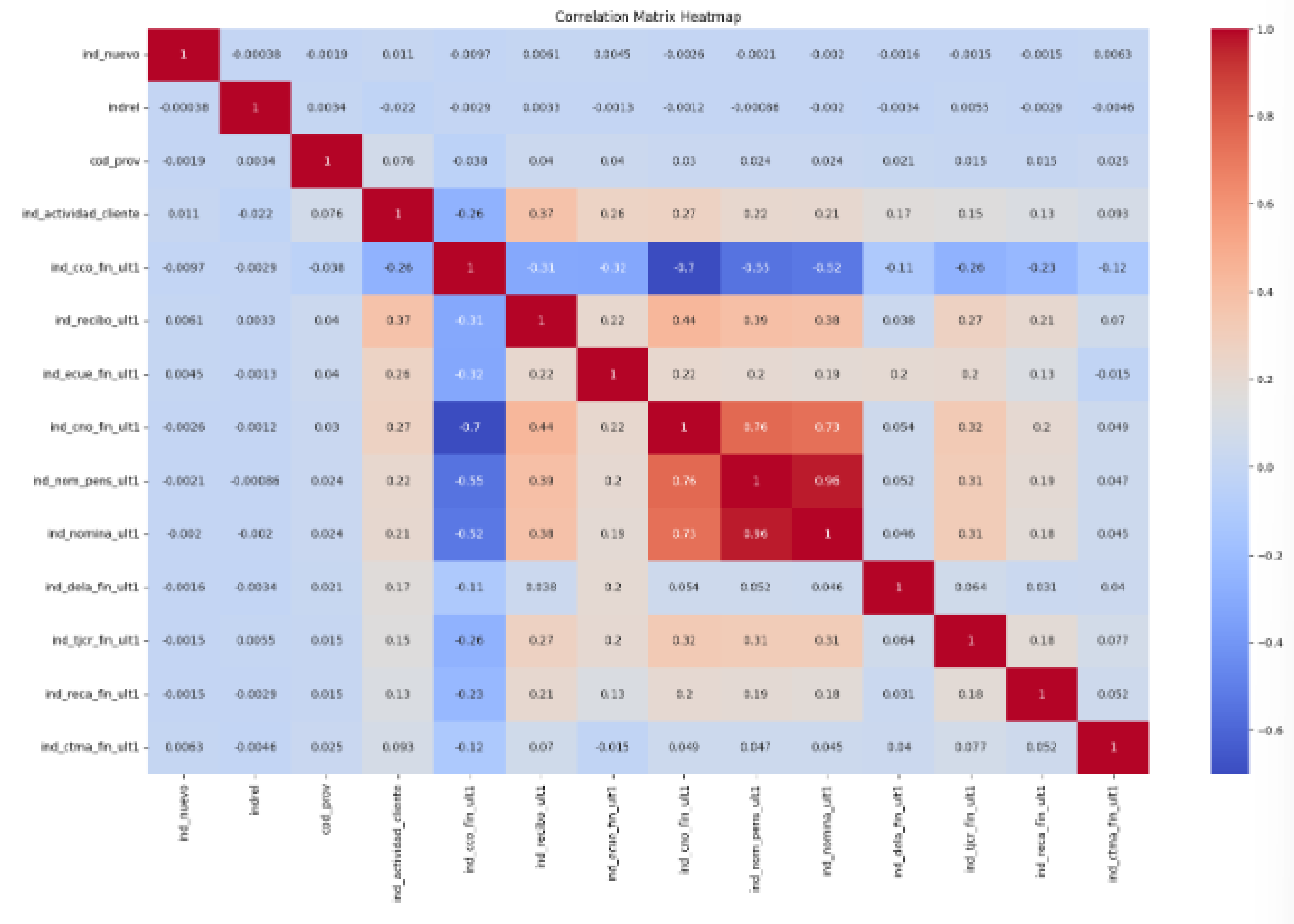




Histogram for indfall by Classes
Histogram for cod_prov by Classes
Histogram for nomprov by Classes
Histogram for ind_actividad_cliente by Classes
Histogram for renta by Classes
Histogram for segmento by Classes
Frequency
indfall
cod_prov
nomprov
ind_actividad_cliente
renta
segmento
ind_recibo_ult1
ind_ecue_fin_ult1
ind_cno_fin_ult1
ind_nom_pens_ult1
ind_nomina_ult1
ind_dela_fin_ult1
ind_tjcr_fin_ult1
ind_reca_fin_ult1
ind_ctma_fin_ult1
03 - UNIVERSITARIO
02 - PARTICULARES
01 - TOP

CORRELATION WITH NUMERICAL VARIABLES

CORRELATION WITH ENCODED CATEGORICAL VARIABLES



MACHINE LEARNING

SELECTED FEATURES AND TARGET VARIABLES

- *'sexo_encoded'* = Gender of the customer (categorical: 0 for male, 1 for female)
- 'age' = age of customer
- 'antiguedad' = seniority/tenure of customer in months
- 'tiprel_1mes' = customer relation type ate beginning of month (0 = active, 1 = inactive)
- 'cod_prov' = province code of customer's residence
- 'ind_actividad_cliente' = indicator if customer is active or not (0 = ACTIVE, 1 = inactive)
- 'renta' = gross income of household the customer resides

SELECTED FEATURES AND TARGET VARIABLES

- *'ind_cco_fin_ult1'* = Current accounts
- 'ind_recibo_ult1' = Direct debits
- 'ind_ecue_fin_ult1' = e-account
- 'ind_cno_fin_ult1' = Payroll account
- 'ind_nom_pens_ult1' = Pensions
- 'ind_nomina_ult1' = Payroll
- 'ind_dela_fin_ult1' = Taxes
- 'ind_tjcr_fin_ult1' = Credit Cards
- 'ind_reca_fin_ult1' = Taxes
- 'ind_ctma_fin_ult1' = Long-term deposits

PERFORMANCE METRICS

I'll be using evaluation metrics suitable for multi-label classification tasks, such as:

- Hamming loss
- Jaccard similarity score
- Precision, recall, and F1-score for each product
- Micro/macro averaged metrics

MODEL SELECTION

I've decided to try two different models for this task:

- **K-Nearest Neighbors (KNN) Model**
  - I chose this due to its ease of interpretation in robustness to outliers.
  - KNN works by classifying instances based on the majority class among their k nearest neighbors
- **Random Forest Model**
  - I chose this for its ability to handle outliers and complex relationships in the data.
  - RF builds multiple decision trees and combines their predictions through voting to make the final prediction.

KNN

```
[111]: X = df[features]
       y = df[new_targets]

       # Splitting the dataset
       X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=470)
```

```
[112]: # Preprocessing for numerical variables

       numerical_features = ['age', 'antiguedad', 'renta']

       numerical_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='median')),
                                               ('scaler', StandardScaler())])
```

```
[113]: # Preprocessing for categorical variables

       categorical_features = ['sexo_encoded', 'tiprel_1mes_encoded', 'cod_prov', 'ind_actividad_cliente']

       categorical_transformer = Pipeline(steps = [('imputer', SimpleImputer(strategy='most_frequent')),
                                                   ('onehot', OneHotEncoder(handle_unknown='ignore'))])
```

```
[114]: # Combining Preprocessing

       preprocessor = ColumnTransformer(transformers=[('num', numerical_transformer, numerical_features),
                                                      ('cat', categorical_transformer, categorical_features)])
```

```
[115]: knn_pipeline=Pipeline(steps=[('preprocessor', preprocessor),
                                    ('classifier', KNeighborsClassifier(n_neighbors=4))])
```

```
[116]: # Train
       knn_pipeline.fit(X_train,y_train)
```

```
Classification Report For KNN:
              precision    recall  f1-score   support

           0       0.92      0.93      0.93     17367
           1       0.37      0.12      0.18      1944
           2       0.46      0.15      0.22      1426
           3       0.20      0.03      0.05      1100
           4       0.19      0.01      0.03       711
           5       0.18      0.01      0.03       669
           6       0.35      0.04      0.07       447
           7       0.12      0.00      0.01       391
           8       0.14      0.00      0.01       360
           9       0.21      0.06      0.09       288

   micro avg       0.88      0.68      0.77     24703
   macro avg       0.31      0.14      0.16     24703
weighted avg       0.74      0.68      0.69     24703
 samples avg       0.84      0.81      0.81     24703


Accuracy Score: 0.7577076532462822
```

RANDOM FOREST

[93]:
```python
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df['sexo_encoded'] = label_encoder.fit_transform(df['sexo'])
```

[94]:
```python
label_encoder = LabelEncoder()
df['tiprel_1mes_encoded'] = label_encoder.fit_transform(df['tiprel_1mes'])
```

[96]:
```python
features = ['sexo_encoded', 'age', 'antiguedad', 'tiprel_1mes_encoded', 'cod_prov',
            'ind_actividad_cliente', 'renta']
```

[97]:
```python
from sklearn.ensemble import RandomForestClassifier

X = df[features]
Y = df[new_targets]

clf = RandomForestClassifier(n_estimators=10)
clf = clf.fit(X, Y)
```

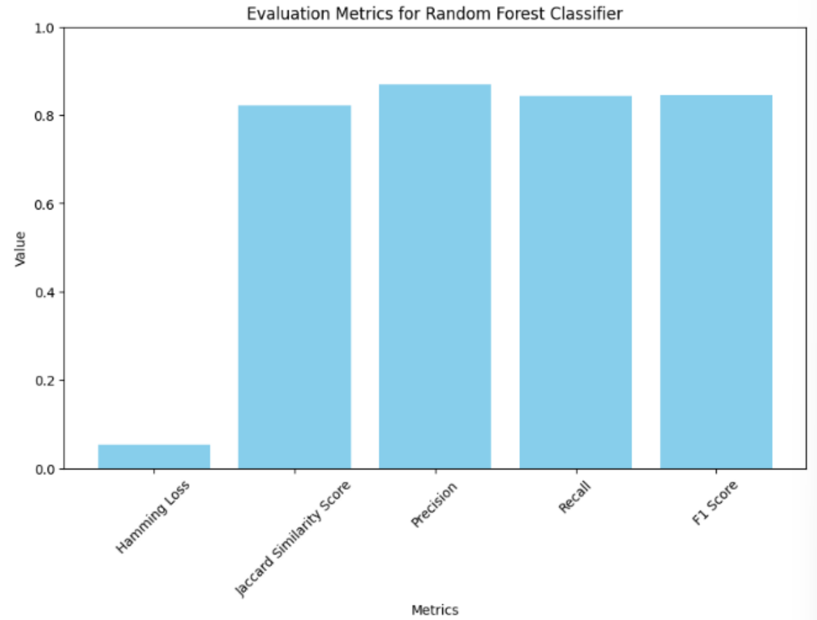
[98]:
```python
# Split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

[99]:
```python
# Initialize
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

[100]:
```python
rf_classifier.fit(X_train, y_train)
```

```
Accuracy: 0.7688481268459506
```

```
Hamming Loss: 0.05405461422871651
Jaccard Similarity Score: 0.8225817276493478
Precision: 0.8698946281450557
Recall: 0.843348779482776
F1 Score: 0.8448592325166241
```



CONCLUSION

Overall, the Random Forest model demonstrates stronger predictive performance and is better suited for predicting customer product purchases.

In summary, the Random Forest model shows promise in accurately predicting customer product purchases, and efforts should be focused on refining this model for deployment in real-world applications.