G6

Distribución ideal de los Usos de Suelo en la Ciudad de Buenos Aires

Desafío 3

- *Bautista Alonso Marangone*
- *Gerardo Burgos*
- *Pablo Marcelo Ferrero*
- *Mauro Montrasi*

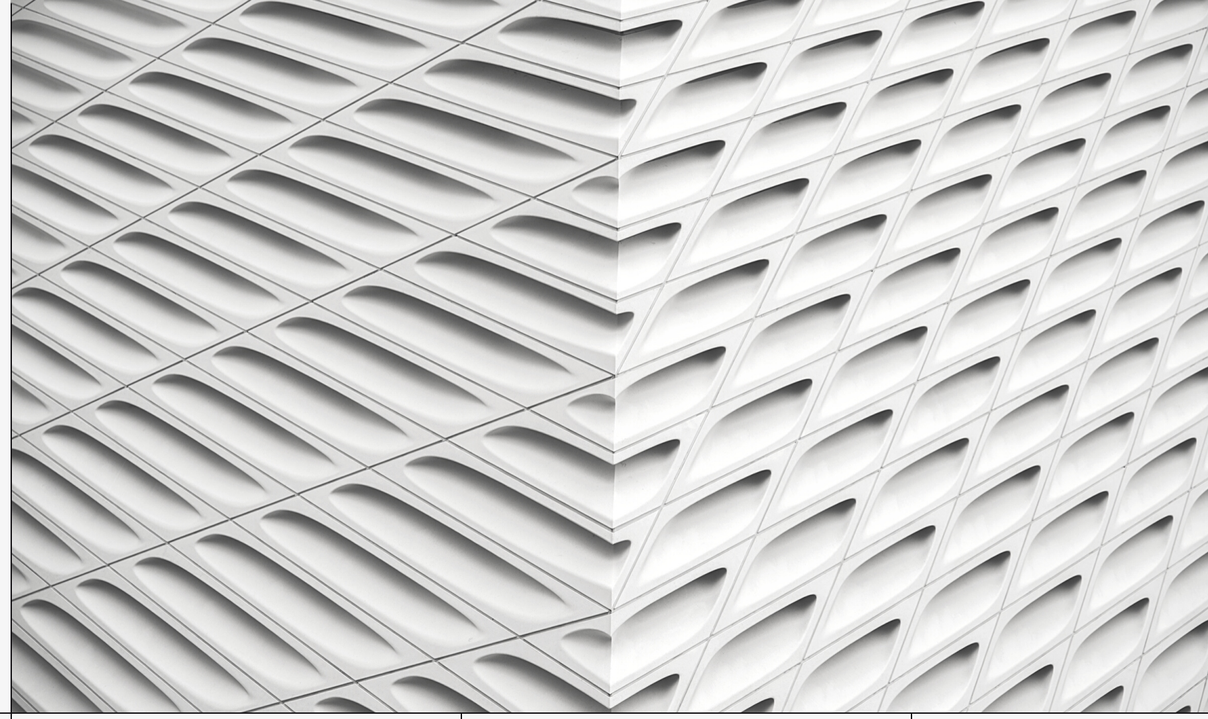
Entendiendo nuestro negocio

Objetivo inmediato:
Determinar, de acuerdo al conjunto de datos seleccionado, (*Relevamiento de Usos de Suelo - 2017 - GCBA*) cómo se distribuyen los diferentes tipos de Usos de Suelo en el territorio de la Ciudad.

Entendiendo nuestro negocio

Objetivo a largo plazo:
Establecer la proyección de dichos usos y predecir los cambios en el mapa. Buscamos la disposición óptima de los Usos, en el tejido de la Ciudad.

Workflow



EXTRACTING DATA

En primera instancia, y una vez fijado el objetivo a lograr en el corto plazo, buscamos los conjuntos de datasets necesarios.

DATA PROCESING

Se eliminaron columnas irrelevantes y se utilizaron métodos reservados de Python. Visualizamos por primera vez para tener una mejor aproximación.

DATA INTEGRATION

Procedimos a la unificación de los conjuntos de datos necesarios para validar la idea. Merge, replace, drop, concat, fillna, entre otros, de las librerias Pandas y Numpy.

ANALYTICAL MODELING

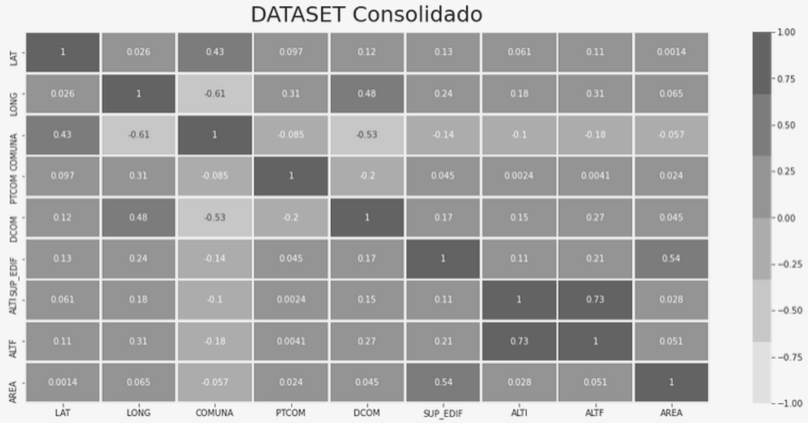
De acuerdo a nuestros datos, buscamos agrupar por categorías de manera relevante. A través del uso de modelos de clasificación KNeighborsClassifier, Cross Validation y

VALIDATION

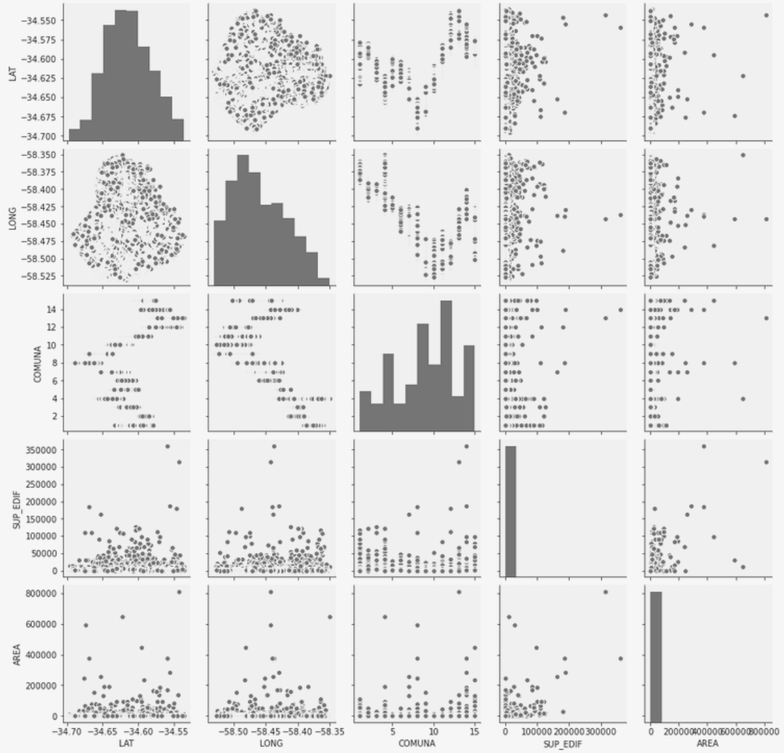
Determinamos los algoritmos y aplicamos libreria ScikitLearn

DATA WRANGLING

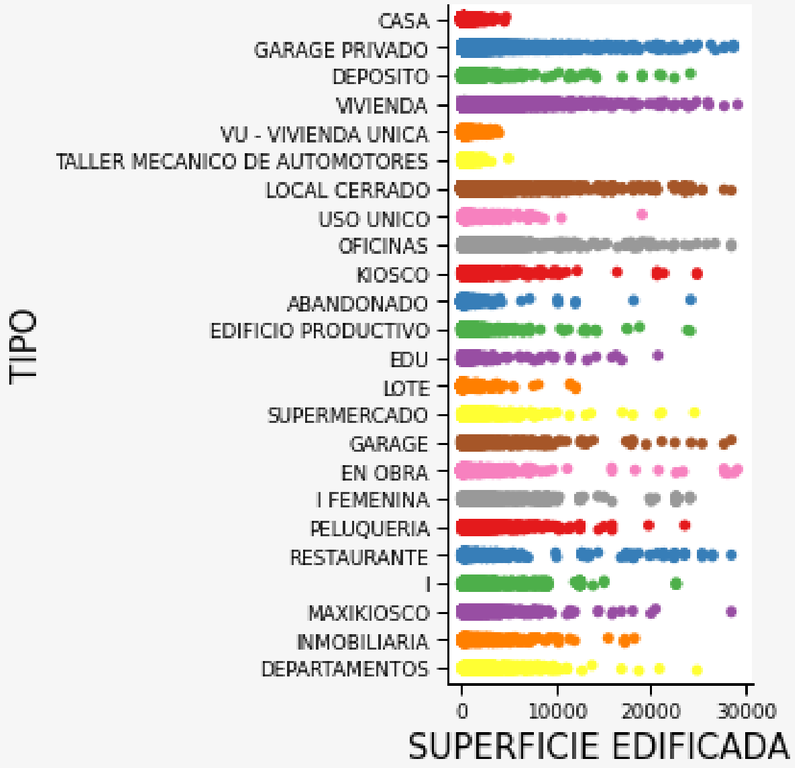
Desafío 3

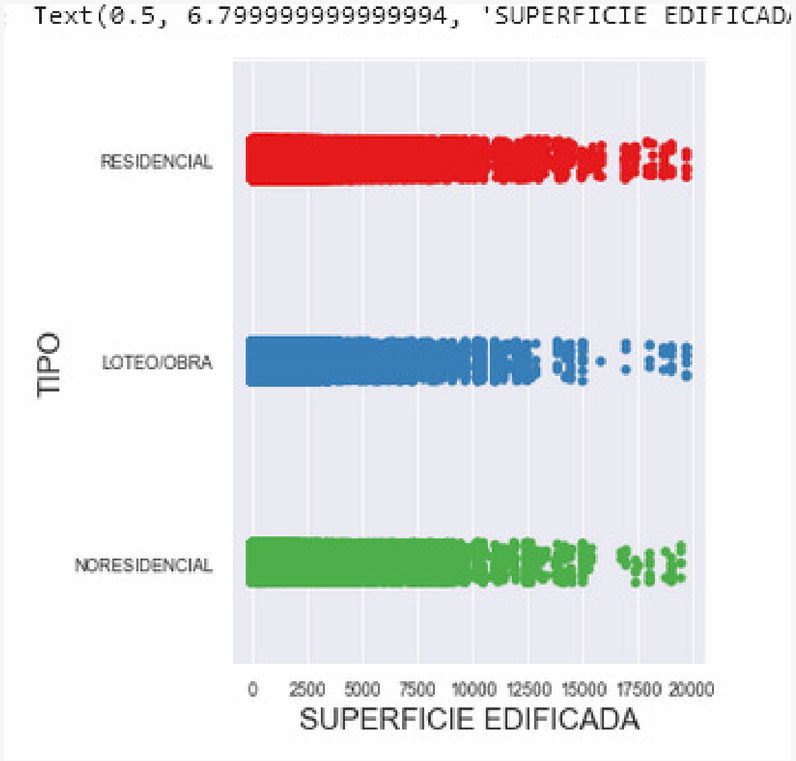



```
In [38]: #TEST Y REDUCCION EN Y
         alfa = data_merged3.PREDICT.value_counts()
         d2 = data_merged3[data_merged3.PREDICT.isin(alfa.index[alfa>1000])]
         d2.PREDICT.value_counts(), d2.shape

Out[38]: (VIVIENDA                          118413
          GARAGE PRIVADO                   90999
          CASA                             79030
          LOCAL CERRADO                    19918
          VU - VIVIENDA UNICA              12567
          DEPOSITO                          4474
          EDIFICIO PRODUCTIVO               4356
          OFICINAS                          4344
          I FEMENINA                        3369
          TALLER MECANICO DE AUTOMOTORES    2788
          PELUQUERIA                        2095
          SUPERMERCADO                      1821
          I                                 1659
          ABANDONADO                        1632
          INMOBILIARIA                      1577
          KIOSCO                            1494
          EN OBRA                           1417
          EDU                               1373
          LOTE                              1363
          MAXIKIOSCO                        1252
          DEPARTAMENTOS                     1224
          GARAGE                            1218
          RESTAURANTE                       1150
          USO UNICO                         1134
          Name: PREDICT, dtype: int64,
          (360667, 68))
```



PRIMER ACERCAMIENTO A LA VARIABLE PREDICTIVA





Optamos por la reducción de nuestra variableY, de modo que en un primer acercamiento contaba con mas de 500000 features. Finalmente, a traves de distintos pasos, damos cuenta que es posible englobarlos en 3 grandes categorías.

RESIDENCIAL, NO RESIDENCIAL, OBRA/LOTE.

CAMBIAMOS EL LABEL ENCODING, DE RES/ NO RES / LOT

Algoritmos

VARIABLE Y.

De acuerdo con nuestro Dataframe, apuntamos a definir nuestra vector taget Y ('RAMA'), que nos habla sobre el Uso de Suelo en la matriz de features X, parcela de la ciudad.

TRAIN, SPLIT, TEST.

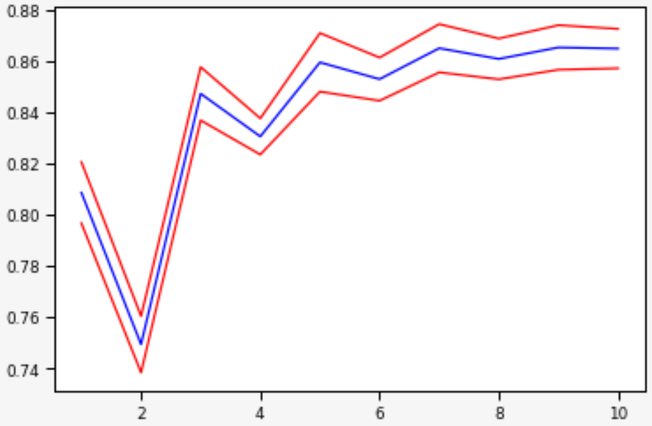
Armamos los conjuntos de entrenamiento y test con una proporción 70-30 y una vez que el modelo está entrenado, vamos a predecir las etiquetas del conjunto de test

EVALUACIÓN, DESARROLLO Y PERFORMANCE

Calculamos accuracy sobre el conjunto de test, la matriz de confusión y visualizamos la performance.

K - NEAREST NEIGHBOURS (KNN)

PRUEBA 20 SAMPLES



En cada iteración, instanciamos el modelo con un hiperparámetro distinto

cross_val_scores nos devuelve un array de 5 resultados, uno por cada partición que hizo automáticamente CV

Para cada valor de n_neighbours, creamos un diccionario con el valor de n_neighbours y la media y el desvío de los scores

Incorporamos los límites inferior y superior, restando y sumando el valor del desvío estándar, respectivamente

PRECISION, ACCURACY, F1-SCORE

Elegimos el modelo óptimo que indique indicado cross validation

Instanciamos y entrenamos el modelo

Sensitivity, Specificity, Precision, Evaluamos qué accuracy obtenemos en train

Lo utilizamos para predecir en test

```
#El tema de este numero de Accuracy, es que es que sabemos que las clases intra variable no estan bien balanceadas,
```

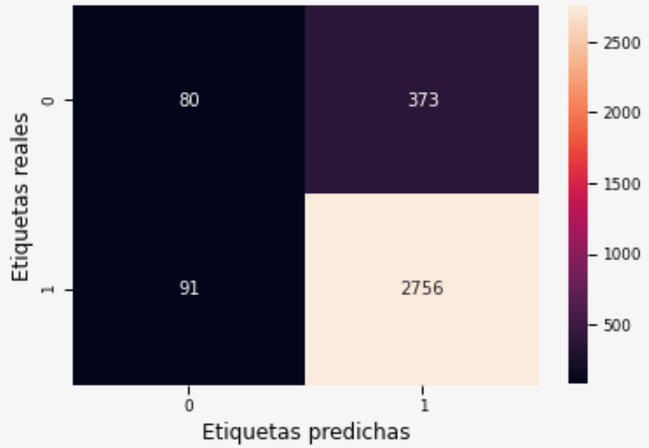
```
# Recall, Es el número de elementos identificados correctamente como positivos del total de positivos verdaderos.
print('Recall=',recall_score(y_test, y_pred,average='micro').round(2))
```

```
Recall= 0.86
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.45      0.16      0.24       453
           2       0.88      0.97      0.92      2847

    accuracy                           0.86      3300
   macro avg       0.66      0.56      0.58      3300
weighted avg       0.82      0.86      0.83      3300
```

MATRIZ DE CONFUSIÓN



Desafío 3

REGRESION LOGÍSTICA

PASO 1

```
In [115]: ## Standarizamos la matriz de *features*
          from sklearn.preprocessing import StandardScaler
          scaler = StandardScaler()
          x1 = scaler.fit_transform(x1)

In [116]: # Verificamos que las variables ahora tengan media 0 y desvío 1.
          print('Medias:', np.mean(x1, axis=0).round(2))
          print('Desvio:', np.std(x1, axis=0).round(2))

          Medias: [ 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.  0. -0. -0.  0. -0.  0. -0. -0.]
          Desvio: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
           1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
           1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

In [117]: # Importamos la clase
          from sklearn.linear_model import LogisticRegression
          # Instanciamos un objeto de esa clase
          logistic_regression = LogisticRegression()
          # Ajustamos esta instancia con los datos de entrenamiento
          logistic_regression.fit(x1,y1)

Out[117]: LogisticRegression()

In [118]: logistic_regression.predict(x1)

Out[118]: array([2, 2, 2, ..., 2, 2, 2], dtype=int8)

In [119]: logistic_regression.predict_proba(x1).shape

Out[119]: (10000, 2)
```

Donde la columna 0 es la probabilidad de pertenencia a la clase 0 y la columna 1 es la probabilidad de pertenencia a la clase 1.

PASO 2

```
[120]: # cHEAQUEAMOS Sumamos el valor de cada columna para cada fila
       logistic_regression.predict_proba(x1).sum(axis=1)

t[120]: array([1., 1., 1., ..., 1., 1., 1.])
```

Si nos quedamos sólamente con la columna 1, podremos hacer la comparación que mencionamos recién:

```
[121]: # Nos quedamos sólamente con la columna 1
       prob_1 = logistic_regression.predict_proba(x1)[:,1]
       # Comparamos con 0.5
       prob_1 >= 0.5

t[121]: array([ True,  True,  True, ...,  True,  True,  True])

[122]: # Comparamos con el método .predict()
       (prob_1 >= 0.5) == logistic_regression.predict(x1)

t[122]: array([False, False, False, ..., False, False, False])
```

Vemos que es lo mismo para todos los casos obtener predicciones con el método `.predict()` que obtener la estimación de las probabilidades `.predict_proba()` y luego compararlo con el valor de umbral 0.5.
En la práctica a veces es necesario modificar el valor de umbral por diversos motivos, por lo que no hay que perder de vista que los clasificadores Learn tienen el método `.predict_proba()` que nos permite jugar manualmente con el valor de umbral. Si no es necesario modificar este valor, `.predict()` obtendremos los mismo resultados que umbralizando con 0.5.

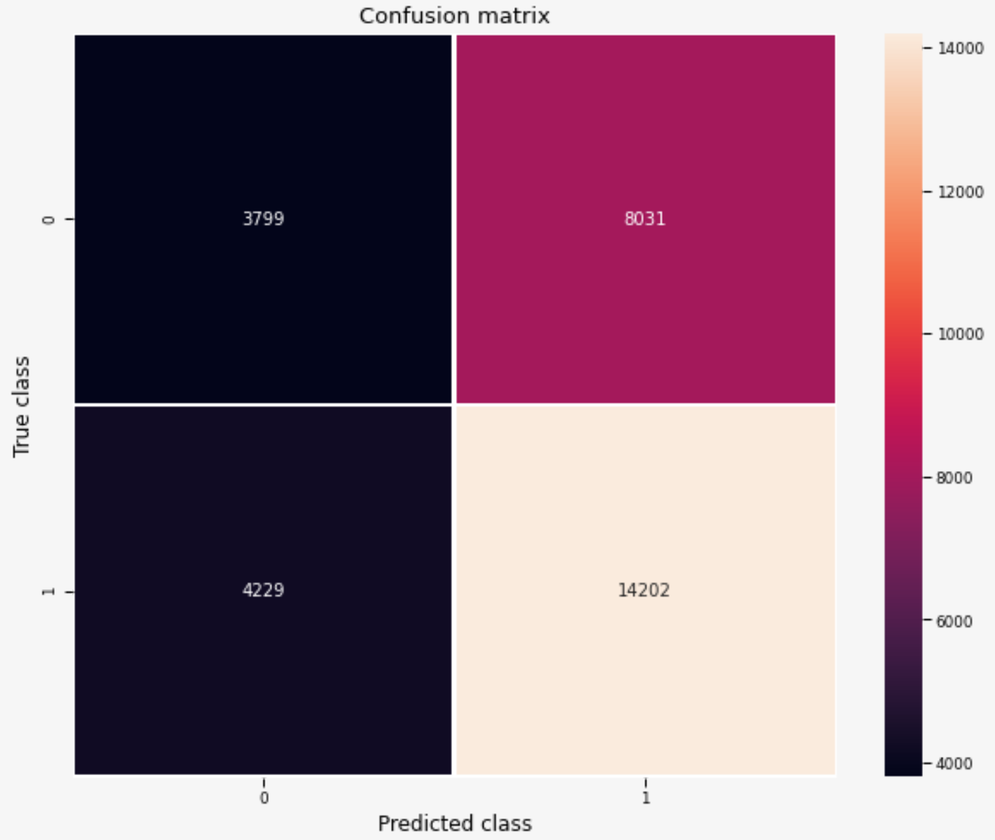
Una vez que ya tenemos las etiquetas puestas por el modelo (es decir $\hat{y}$) debemos comparar esas predicciones con los valores reales (y)

PASO 3

```
In [124]: # Usando numpy
          (y == y_pred).sum() / len(y)

Out[124]: 0.8665

In [125]: # Usando Scikit-Learn
          from sklearn.metrics import accuracy_score
          accuracy_score(y, y_pred)

Out[125]: 0.8665

In [126]: logistic_regression.intercept_

Out[126]: array([2.06592299])

In [127]: logistic_regression.coef_

Out[127]: array([[ 0.03531311, -0.13597989,  0.1524142 , -0.11098254, -0.07857536,
                   0.06985277, -0.2629344 , -0.40912912, -0.09147308, -0.16642873,
                  -0.03964529, -0.02896097, -0.0702566 ,  0.08506617,  0.01695031,
                  -0.15109513,  0.04324969, -0.02326039, -0.09334519, -0.10139432,
                  -0.24861599, -0.02199634, -0.0246871 ,  0.03307038, -0.14603362,
                  -0.05368589,  0.13524041, -0.01691365,  0.03923185,  0.10452437,
                  -0.01176175, -0.02820986, -0.13516859, -0.05199935, -0.07443382,
                   0.13189015,  0.05096412, -0.20900127,  0.08180597,  0.02365044,
                   0.1114879 , -0.27618049,  0.03274333,  0.13097081, -0.12967129,
                  -0.07213854,  0.03226942, -0.09715861,  0.09556964,  0.05012476,
                   0.04299512,  0.04750498, -0.09008318,  0.09728229, -0.05199935,
                  -0.10144161, -0.09397213, -0.01989001,  0.01695031, -0.02327127,
                  -0.06984423, -0.01080131, -0.05238509,  0.06304349,  0.19345443,
                   0.05565303, -0.01691365,  0.20645821]])
```

Desafío 3

Naive Bayes




```
In [105]: from sklearn.model_selection import train_test_split
          from sklearn.naive_bayes import GaussianNB
          from sklearn.naive_bayes import MultinomialNB
          from sklearn.metrics import accuracy_score
          from sklearn.metrics import confusion_matrix

In [106]: t2=  t.drop(["LAT","LONG"], axis=1).copy()
          t2.shape
          t2.Y.unique()
          t2

Out[106]:
```

	SUP_EDIF	PTCOM	DCOM	ALTI	ALTF	AREA	ALMAGRO	BALVANERA	BARRACAS	BELGRANO	...	COM_7.0	COM_8.0	COM_9.0	COM_10.0	CO
0	737.0	182.445	12.7	19.5	23.5	180.2	0	0	0	0	...	0	0	0	0	
72	451.0	182.445	12.7	19.5	23.5	326.2	0	0	0	0	...	0	0	0	0	
75	451.0	182.445	12.7	16.5	19.5	326.2	0	0	0	0	...	0	0	0	0	
77	451.0	182.445	12.7	0.0	17.2	326.2	0	0	0	0	...	0	0	0	0	
97	289.0	182.445	12.7	16.5	19.5	319.6	0	0	0	0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1358874	2198.0	149.510	23.7	25.8	29.8	252.9	0	0	0	0	...	0	0	0	0	
1358939	6937.0	149.510	23.7	41.0	45.0	736.8	0	0	0	0	...	0	0	0	0	
1358942	1372.0	149.510	23.7	22.8	25.8	186.5	0	0	0	0	...	0	0	0	0	
1358952	1135.0	149.510	23.7	0.0	31.2	172.8	0	0	0	0	...	0	0	0	0	
1358963	200.0	149.510	23.7	0.0	22.8	144.3	0	0	0	0	...	0	0	0	0	

91697 rows × 67 columns

test_size = 0.33

Desafío 3