Código de Python del Parcial 2 de Inteligencia Artificial 1

San Francisco Crime Classification

In [ ]:
```
# El presente archivo contiene la solución realizada en Python.
# Se utilizarán los DataSet contenidos en la carpeta "Data" del repositorio.
# Path: Data/train.csv
# Path: Data/test.csv
```

In [ ]:
```
# Se importan las librerías necesarias para el desarrollo del ejercicio.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB
```

Se importan los datos de entrenamiento y de prueba, desde los archivos CSV. Y se imprimen los primeros 5 registros de cada uno, para verificar que se hayan cargado correctamente.

In [ ]:
```
data_train = pd.read_csv('Data/train.csv')
print(data_train.head())
```

```
                 Dates        Category                     Descript  \
0  2015-05-13 23:53:00        WARRANTS                WARRANT ARREST
1  2015-05-13 23:53:00  OTHER OFFENSES      TRAFFIC VIOLATION ARREST
2  2015-05-13 23:33:00  OTHER OFFENSES      TRAFFIC VIOLATION ARREST
3  2015-05-13 23:30:00   LARCENY/THEFT  GRAND THEFT FROM LOCKED AUTO
4  2015-05-13 23:30:00   LARCENY/THEFT  GRAND THEFT FROM LOCKED AUTO

   DayOfWeek PdDistrict      Resolution                    Address  \
0  Wednesday   NORTHERN  ARREST, BOOKED         OAK ST / LAGUNA ST
1  Wednesday   NORTHERN  ARREST, BOOKED         OAK ST / LAGUNA ST
2  Wednesday   NORTHERN  ARREST, BOOKED  VANNESS AV / GREENWICH ST
3  Wednesday   NORTHERN            NONE   1500 Block of LOMBARD ST
4  Wednesday       PARK            NONE  100 Block of BRODERICK ST

           X          Y
0 -122.425892  37.774599
1 -122.425892  37.774599
2 -122.424363  37.800414
3 -122.426995  37.800873
4 -122.438738  37.771541
```

In [ ]:
```
data_test = pd.read_csv('Data/test.csv')
print(data_test.head())
```

```
   Id                Dates DayOfWeek PdDistrict                   Address  \
0   0  2015-05-10 23:59:00    Sunday    BAYVIEW   2000 Block of THOMAS AV
1   1  2015-05-10 23:51:00    Sunday    BAYVIEW        3RD ST / REVERE AV
2   2  2015-05-10 23:50:00    Sunday   NORTHERN    2000 Block of GOUGH ST
3   3  2015-05-10 23:45:00    Sunday  INGLESIDE  4700 Block of MISSION ST
4   4  2015-05-10 23:45:00    Sunday  INGLESIDE  4700 Block of MISSION ST

            X          Y
0 -122.399588  37.735051
1 -122.391523  37.732432
2 -122.426002  37.792212
3 -122.437394  37.721412
4 -122.437394  37.721412
```

Se procede a realizar el procesamiento de los datos, para poder utilizarlos en el entrenamiento de los modelos de clasificación. Dichos datos se ajustarán para poder utilizarlos en los modelos de Random Forest y Naive Bayes.

Como ya vimos en el ejercicio principal del parcial, los datos de Train contienen valores duplicados en las variables "Dates", "Category", "Descript" y "Address". Por lo tanto, se procede a eliminar los duplicados, para que no afecten el entrenamiento de los modelos.

In [ ]:
```
# Eliminar duplicados con mismas "Dates", "Category", "Descript" y "Address"
print("Antes: ", data_train.shape)
data_train = data_train.drop_duplicates(subset=['Dates', 'Category', 'Descript', 'Address'])
print("Después: ", data_train.shape)
```

```
Antes:  (878049, 9)
Después:  (875357, 9)
```

In [ ]:
```
# Como las columnas Descript y Resolution no se utilizarán en el modelo, se eliminan.
data_train = data_train.drop(['Descript', 'Resolution'], axis=1)
print(data_train.columns)
```

```
Index(['Dates', 'Category', 'DayOfWeek', 'PdDistrict', 'Address', 'X', 'Y'], dtype='object')
```

Otro dato que debemos ajustar que lo pudimos apreciar en el ejercicio principal, es que las variables "X" e "Y" contienen valores atípicos, que se encuentran fuera de los límites de la ciudad de San Francisco. Por lo tanto, se procede a eliminar dichos valores atípicos, para que no afecten el entrenamiento de los modelos.

In [ ]:
```
# Se eliminaran los valores que en X sean mayores a -121.500 y en Y sean mayores a 37.820.

data_train = data_train.drop(data_train[data_train.X > -121.500].index)
data_train = data_train.drop(data_train[data_train.Y > 37.820].index)
print("Datos actuales: ", data_train.shape)
```

```
Datos actuales:  (875290, 7)
```

El siguiente paso es poder dividir la variable "Dates" en 4 variables nuevas, que serán "Year", "Month", "Day" y "Hour". Esto se realiza para poder utilizar dichas variables en el entrenamiento de los modelos.

Se procede a realizar el mismo procesamiento de datos para el conjunto de datos de Test.

In [ ]:
```
# Dividir la columna "Dates" en "Year", "Month", "Day" y "Hour"

data_train_dates = pd.to_datetime(data_train['Dates'])
data_train['Year'] = data_train_dates.dt.year
data_train['Month'] = data_train_dates.dt.month
data_train['Day'] = data_train_dates.dt.day
data_train['Hour'] = data_train_dates.dt.hour
data_train = data_train.drop(['Dates'], axis=1)
print(data_train.columns)
```

```
Index(['Category', 'DayOfWeek', 'PdDistrict', 'Address', 'X', 'Y', 'Year',
       'Month', 'Day', 'Hour'],
      dtype='object')
```

In [ ]:
```
# Se hace lo mismo para el conjunto de datos de prueba.

data_test_dates = pd.to_datetime(data_test['Dates'])
data_test['Year'] = data_test_dates.dt.year
data_test['Month'] = data_test_dates.dt.month
data_test['Day'] = data_test_dates.dt.day
data_test['Hour'] = data_test_dates.dt.hour
data_test = data_test.drop(['Dates'], axis=1)
print(data_test.columns)
```

```
Index(['Id', 'DayOfWeek', 'PdDistrict', 'Address', 'X', 'Y', 'Year', 'Month',
       'Day', 'Hour'],
      dtype='object')
```

Como se puede ver en la salida anterior, el conjunto de Test contiene la variable "Id", que no se encuentra en el conjunto de Train. Por lo tanto, se procede a eliminar dicha variable, para que no afecte el entrenamiento de los modelos.

In [ ]:
```
# Eliminar ID de los datos de prueba.

data_test = data_test.drop(['Id'], axis=1)
print(data_test.columns)
```

```
Index(['DayOfWeek', 'PdDistrict', 'Address', 'X', 'Y', 'Year', 'Month', 'Day',
       'Hour'],
      dtype='object')
```

Se procede a verificar que los conjuntos de datos de Train y Test tengan la misma cantidad de variables, para poder utilizarlos en el entrenamiento de los modelos.

In [ ]:
```
# Verificar igualdad de los conjuntos de datos.

print(data_train.columns)
print(data_test.columns)
```

```
Index(['Category', 'DayOfWeek', 'PdDistrict', 'Address', 'X', 'Y', 'Year',
       'Month', 'Day', 'Hour'],
      dtype='object')
Index(['DayOfWeek', 'PdDistrict', 'Address', 'X', 'Y', 'Year', 'Month', 'Day',
       'Hour'],
      dtype='object')
```

Como vemos el dataset de entrenamiento contiene 1 variable adicional que es la que se quiere predecir.

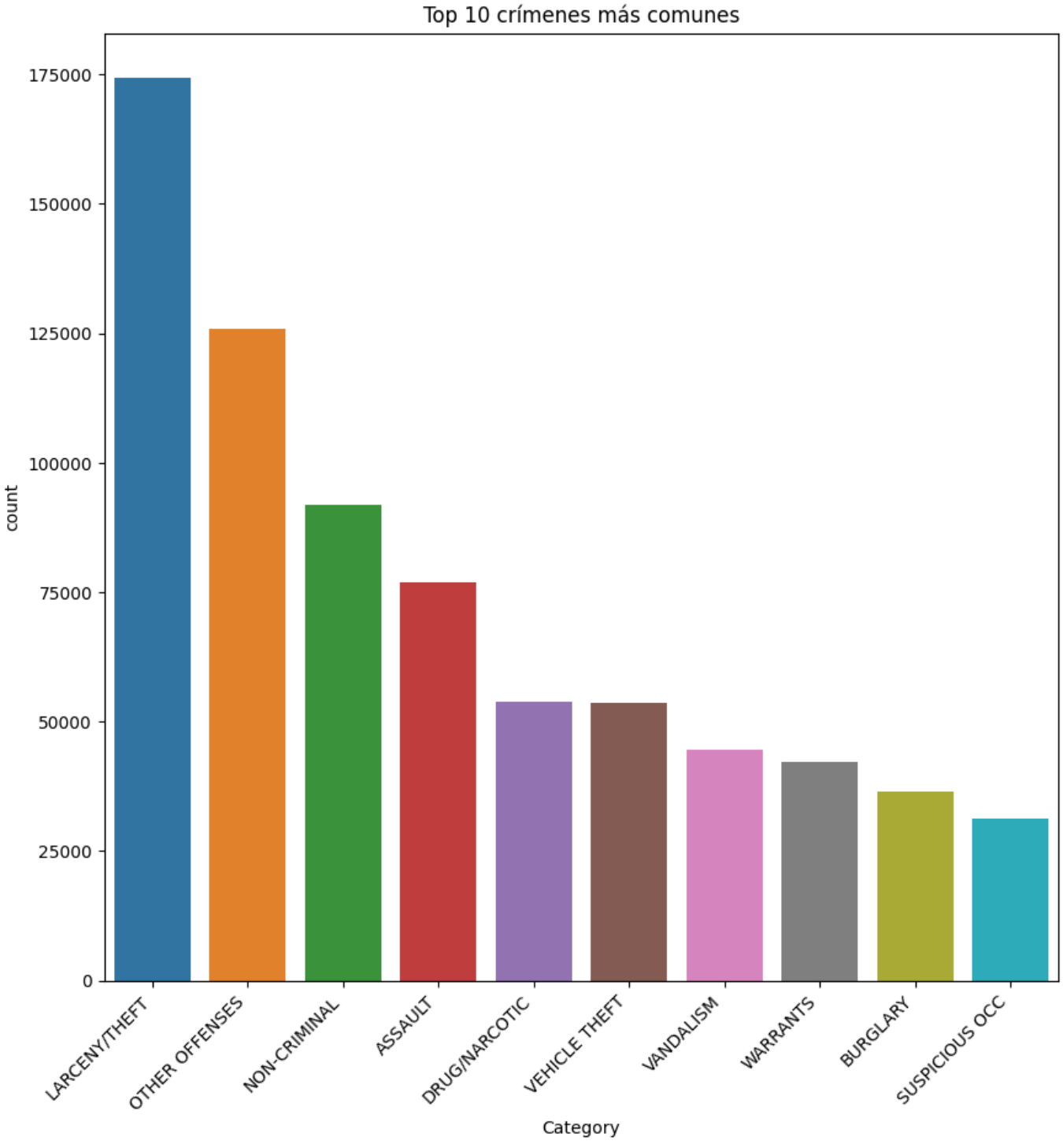
Observación de los datos relevantes de los conjuntos de datos

Se procede a evaluar algunos datos interesantes presentes en el conjunto de Train, como Top 10 crímenes, Días con más crímenes, Valores por distrito, entre otras particularidades.

In [ ]:
```
# Top 10 crímenes más comunes.

print(data_train['Category'].value_counts().head(10))

# Se gráfica la distribución de los crímenes más comunes.

plt.figure(figsize=(10, 10))
sns.countplot(x='Category', data=data_train, order=data_train['Category'].value_counts().head
```

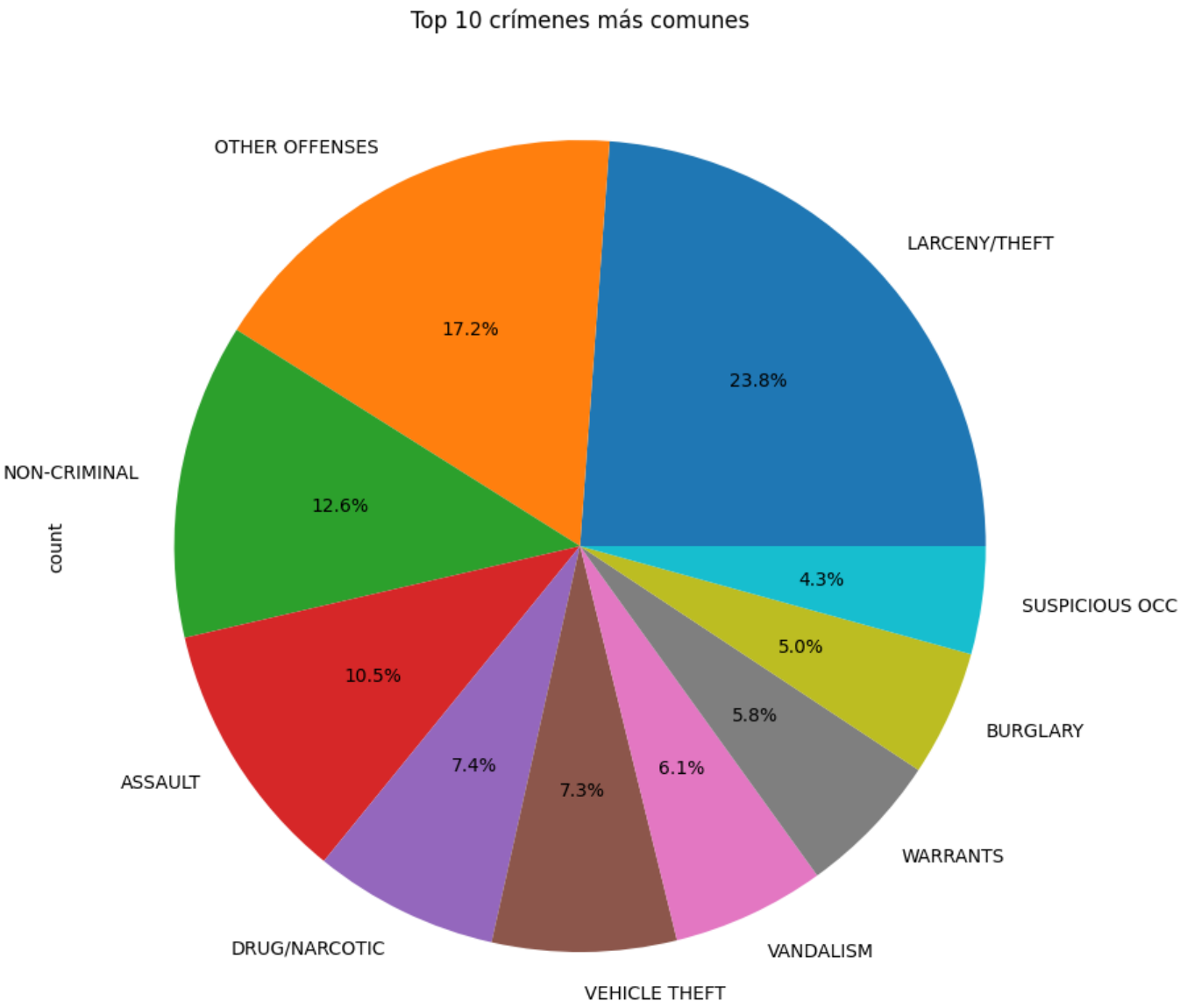
```
plt.xticks(rotation=45, ha='right')
plt.title('Top 10 crímenes más comunes')
plt.show()
```

```
Category
LARCENY/THEFT      174263
OTHER OFFENSES     125913
NON-CRIMINAL        91889
ASSAULT             76787
DRUG/NARCOTIC       53902
VEHICLE THEFT       53664
VANDALISM           44566
WARRANTS            42133
BURGLARY            36594
SUSPICIOUS OCC      31386
Name: count, dtype: int64
```



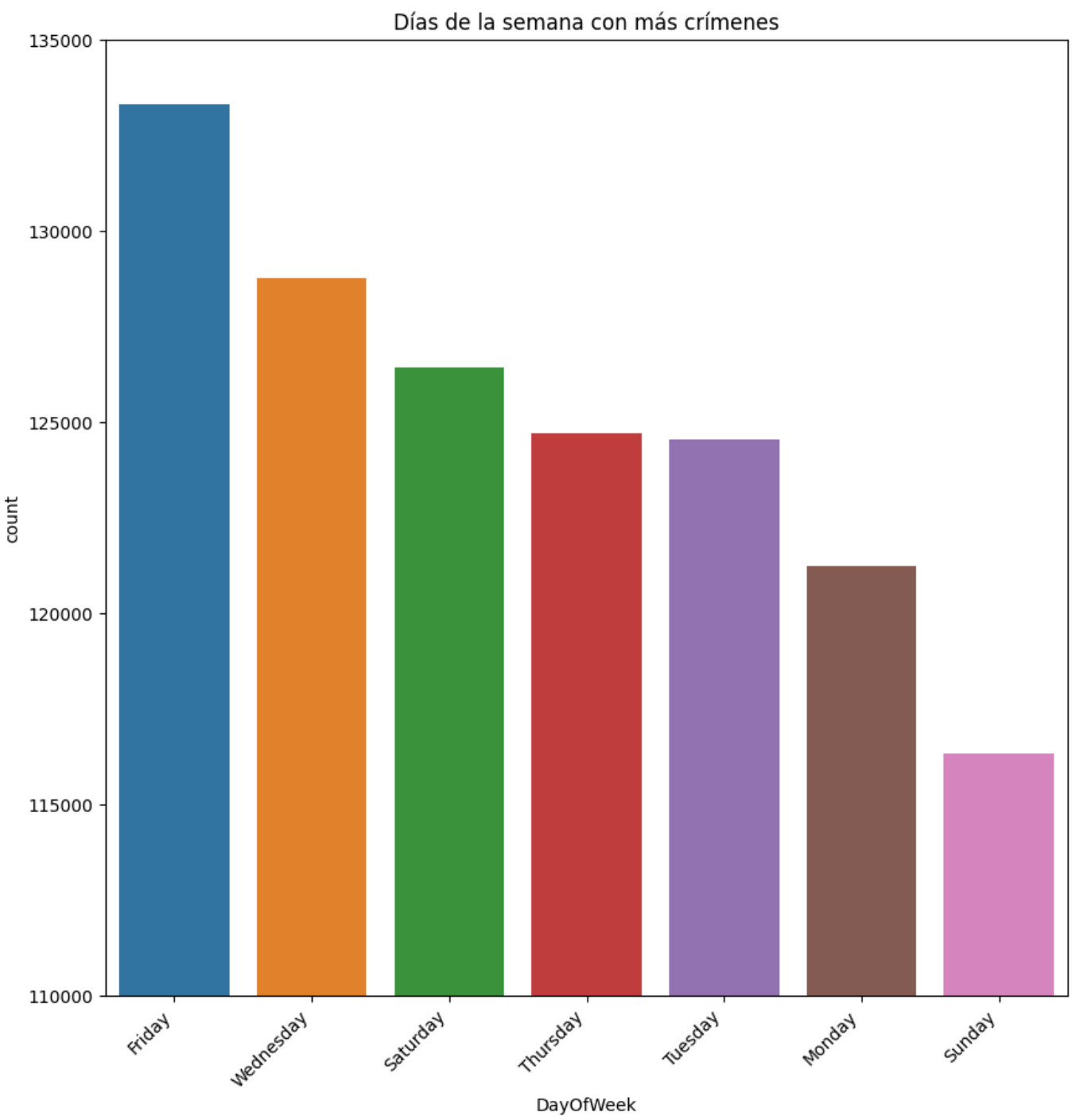
In [ ]:
```
# Gráfica circular con la distribución de los crímenes más comunes.

plt.figure(figsize=(10, 10))
data_train['Category'].value_counts().head(10).plot.pie(autopct='%1.1f%%')
```

```
plt.title('Top 10 crímenes más comunes')
plt.show()
```



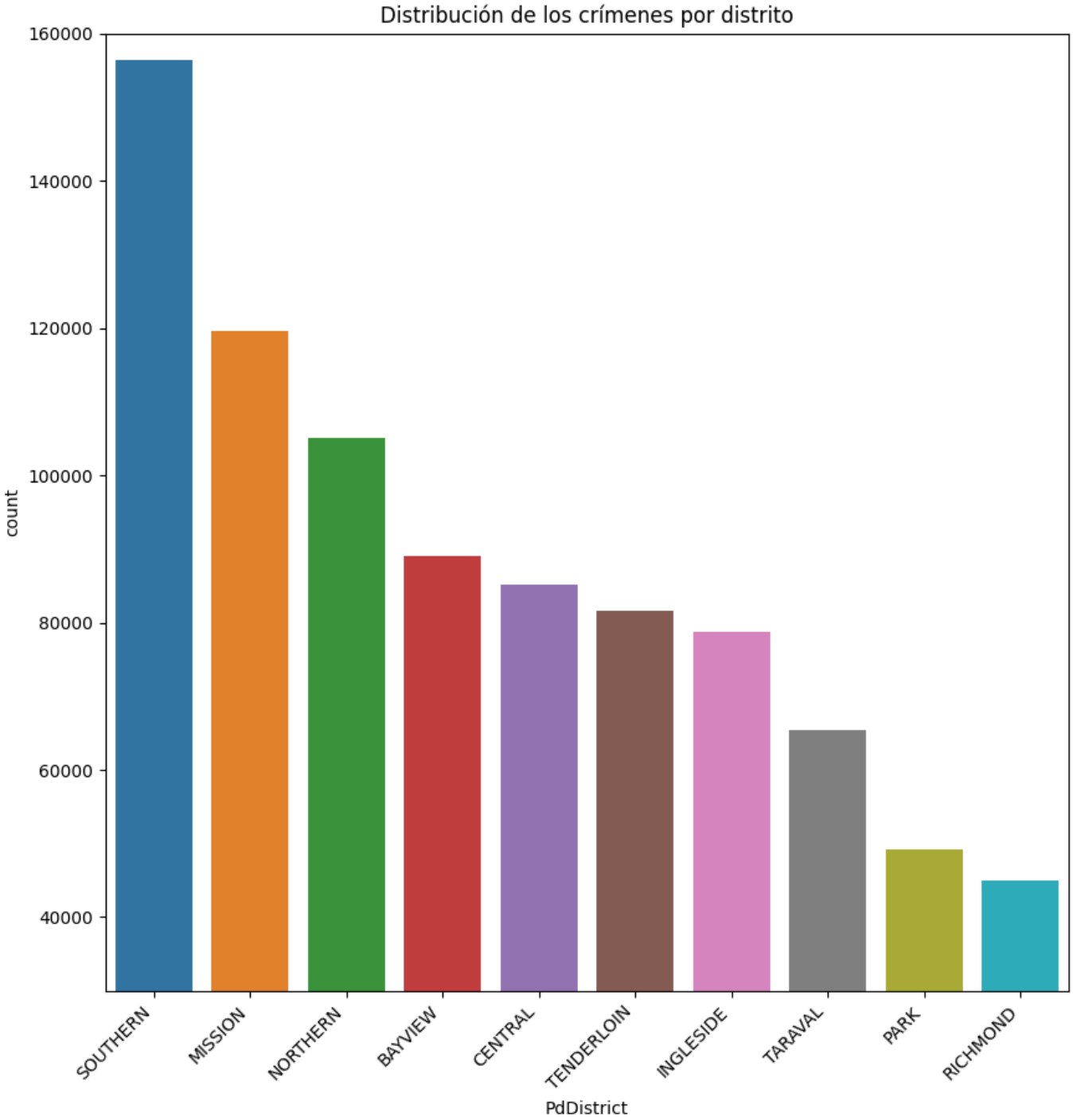
In [ ]:
```
# Días de la semana con más crímenes.

print(data_train['DayOfWeek'].value_counts())

# Se gráfica la distribución de los crímenes por día de la semana.

plt.figure(figsize=(10, 10))
sns.countplot(x='DayOfWeek', data=data_train, order=data_train['DayOfWeek'].value_counts().in
plt.xticks(rotation=45, ha='right')
plt.ylim(110000,135000)
plt.title('Días de la semana con más crímenes')
plt.show()
```

```
DayOfWeek
Friday       133299
Wednesday    128767
Saturday     126410
Thursday     124705
Tuesday      124543
Monday       121230
Sunday       116336
Name: count, dtype: int64
```



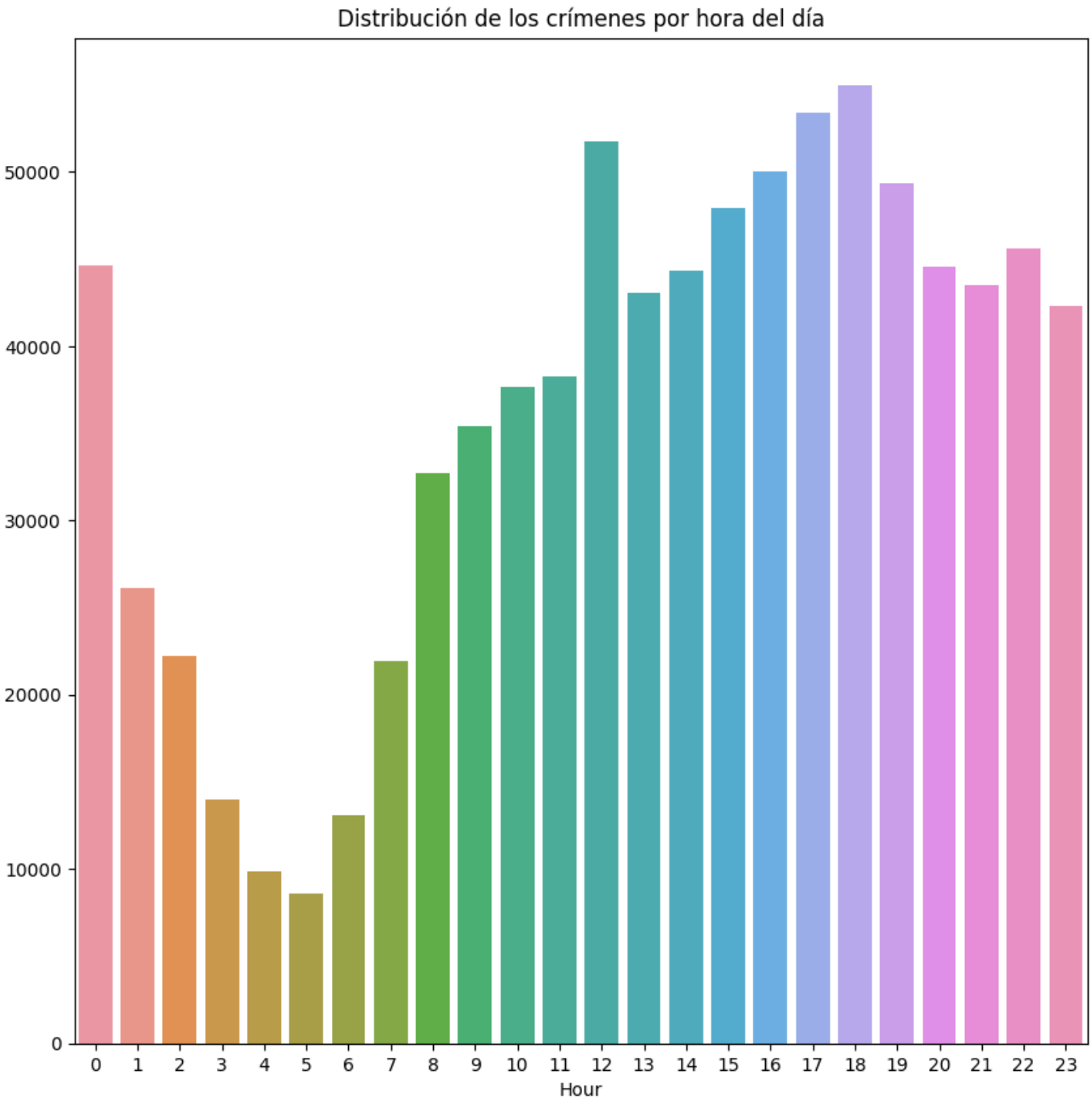
In [ ]:
```
# Distribución de los crímenes por distrito.

print(data_train['PdDistrict'].value_counts())

# Se gráfica la distribución de los crímenes por distrito.

plt.figure(figsize=(10, 10))
sns.countplot(x='PdDistrict', data=data_train, order=data_train['PdDistrict'].value_counts().
plt.xticks(rotation=45, ha='right')
plt.title('Distribución de los crímenes por distrito')
plt.ylim(30000, 160000)
plt.show()
```

```
PdDistrict
SOUTHERN      156446
MISSION       119685
NORTHERN      105065
BAYVIEW        89016
CENTRAL        85242
TENDERLOIN     81604
INGLESIDE      78694
TARAVAL        65351
PARK           49127
RICHMOND       45060
Name: count, dtype: int64
```



In [ ]:
```python
# Continuando con la exploración de los datos, se analiza la distribución de los crímenes por

crimes_by_hour = data_train.groupby('Hour').size()

# Se gráfica la distribución de los crímenes por hora del día.

plt.figure(figsize=(10, 10))
sns.barplot(x=crimes_by_hour.index, y=crimes_by_hour.values)
plt.xticks( ha='center')
```

```
plt.title('Distribución de los crímenes por hora del día')
plt.show()
```



In [ ]:
```
# Top 10 de crímenes por fechas específicas.
print("Top 10 de crímenes por fechas específicas")
print(data_train.groupby(['Day', 'Month']).size().sort_values(ascending=False).head(10))
```

```
Top 10 de crímenes por fechas específicas
Day  Month
1    11       5314
25   1        5253
22   2        4991
11   1        4955
8    2        4941
4    4        4863
8    3        4840
19   4        4834
31   10       4829
10   1        4812
dtype: int64
```

In [ ]:
```
# Tipos de los datos del dataset de entrenamiento.

print(data_train.dtypes)
```

```
Category       object
DayOfWeek      object
PdDistrict     object
Address        object
X             float64
Y             float64
Year            int32
Month           int32
Day             int32
Hour            int32
dtype: object
```

In [ ]:
```
# Le asignamos un valor numérico a cada día de la semana.
data_train['DayOfWeek'] = data_train['DayOfWeek'].map({'Monday':1, 'Tuesday':2, 'Wednesday':3

# Le asignamos un valor numérico a cada distrito.
data_train['PdDistrict'] = data_train['PdDistrict'].map({'SOUTHERN':1, 'MISSION':2, 'NORTHERN
```

In [ ]:
```
# Hacemos lo mismo para el conjunto de datos de prueba.
data_test['DayOfWeek'] = data_test['DayOfWeek'].map({'Monday':1, 'Tuesday':2, 'Wednesday':3,
data_test['PdDistrict'] = data_test['PdDistrict'].map({'SOUTHERN':1, 'MISSION':2, 'NORTHERN':
```

Vemos como van las transformaciones de los datos hasta el momento.

In [ ]:
```
print(data_train.head(10))
```

```
         Category  DayOfWeek  PdDistrict                       Address  \
0        WARRANTS          3           3             OAK ST / LAGUNA ST
1  OTHER OFFENSES          3           3             OAK ST / LAGUNA ST
2  OTHER OFFENSES          3           3     VANNESS AV / GREENWICH ST
3   LARCENY/THEFT          3           3      1500 Block of LOMBARD ST
4   LARCENY/THEFT          3           9     100 Block of BRODERICK ST
5   LARCENY/THEFT          3           7           0 Block of TEDDY AV
6   VEHICLE THEFT          3           7           AVALON AV / PERU AV
7   VEHICLE THEFT          3           4      KIRKWOOD AV / DONAHUE ST
8   LARCENY/THEFT          3          10          600 Block of 47TH AV
9   LARCENY/THEFT          3           5  JEFFERSON ST / LEAVENWORTH ST

            X          Y  Year  Month  Day  Hour
0 -122.425892  37.774599  2015      5   13    23
1 -122.425892  37.774599  2015      5   13    23
2 -122.424363  37.800414  2015      5   13    23
3 -122.426995  37.800873  2015      5   13    23
4 -122.438738  37.771541  2015      5   13    23
5 -122.403252  37.713431  2015      5   13    23
6 -122.423327  37.725138  2015      5   13    23
7 -122.371274  37.727564  2015      5   13    23
8 -122.508194  37.776601  2015      5   13    23
9 -122.419088  37.807802  2015      5   13    23
```

In [ ]:
```
print(data_test.head(10))
```

```
   DayOfWeek  PdDistrict                    Address           X          Y  \
0          7           4   2000 Block of THOMAS AV -122.399588  37.735051
1          7           4        3RD ST / REVERE AV -122.391523  37.732432
2          7           3    2000 Block of GOUGH ST -122.426002  37.792212
3          7           7  4700 Block of MISSION ST -122.437394  37.721412
4          7           7  4700 Block of MISSION ST -122.437394  37.721412
5          7           8     BROAD ST / CAPITOL AV -122.459024  37.713172
6          7           7   100 Block of CHENERY ST -122.425616  37.739351
7          7           7     200 Block of BANKS ST -122.412652  37.739750
8          7           2     2900 Block of 16TH ST -122.418700  37.765165
9          7           5      TAYLOR ST / GREEN ST -122.413935  37.798886

   Year  Month  Day  Hour
0  2015      5   10    23
1  2015      5   10    23
2  2015      5   10    23
3  2015      5   10    23
4  2015      5   10    23
5  2015      5   10    23
6  2015      5   10    23
7  2015      5   10    23
8  2015      5   10    23
9  2015      5   10    23
```

In [ ]:
```
# Se desea convertir la columna "Address" en valores numéricos. Para ello, se utilizará Labell

le = LabelEncoder()
data_train['Address'] = le.fit_transform(data_train['Address'])
data_test['Address'] = le.fit_transform(data_test['Address'])
```

In [ ]:
```
print(data_train['Address'].head(10))
print(data_test['Address'].head(10))
```

```
0    19762
1    19762
2    22661
3     4266
4     1843
5     1505
6    13313
7    18028
8    11378
9    17634
Name: Address, dtype: int32
0     6407
1     9744
2     6336
3    10633
4    10633
5    13799
6     1890
7     5611
8     8090
9    22053
Name: Address, dtype: int32
```

Modelado de los algoritmos de clasificación

Se utilizaran conjuntos reducidos de los datos de Train y Test, para poder realizar las pruebas de los modelos de clasificación. Cada uno será un 10% del total de los datos original.

Random Forest

Se procede a entrenar el modelo de Random Forest, con los datos de Train.

In [ ]:
```
# Datos redimensionados.

data_train_sample = data_train.sample(frac=0.1, random_state=1)
print("Sample train: ",data_train_sample.shape)

data_test_sample = data_test.sample(frac=0.1, random_state=1)
print("Sample test:", data_test_sample.shape)
```

```
Sample train:  (87529, 10)
Sample test: (88426, 9)
```

In [ ]:
```
# Modelo Random Forest.
# Ya existen datos de entrenamiento y de prueba
# La variable objetivo es "Category"

"""
Number of Trees: 100
Criterion: Accuracy
Maximal Depth: 10
Apply Pruning: True
Confidence: 0.1
Apply Pre-Pruning: True
Minimum Gain: 0.01
Minimum Leaf Size: 2
Minimum Size for Split: 4
Number of Prepruning Alternatives: 3
"""
model_rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_depth=10, min_sa

X_train = data_train_sample.drop(['Category'], axis=1)
y_train = data_train_sample['Category']
```

In [ ]:
```
X_test = data_test_sample
```

In [ ]:
```
model_rf.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_leaf=2,
                       min_samples_split=4, random_state=1)
```

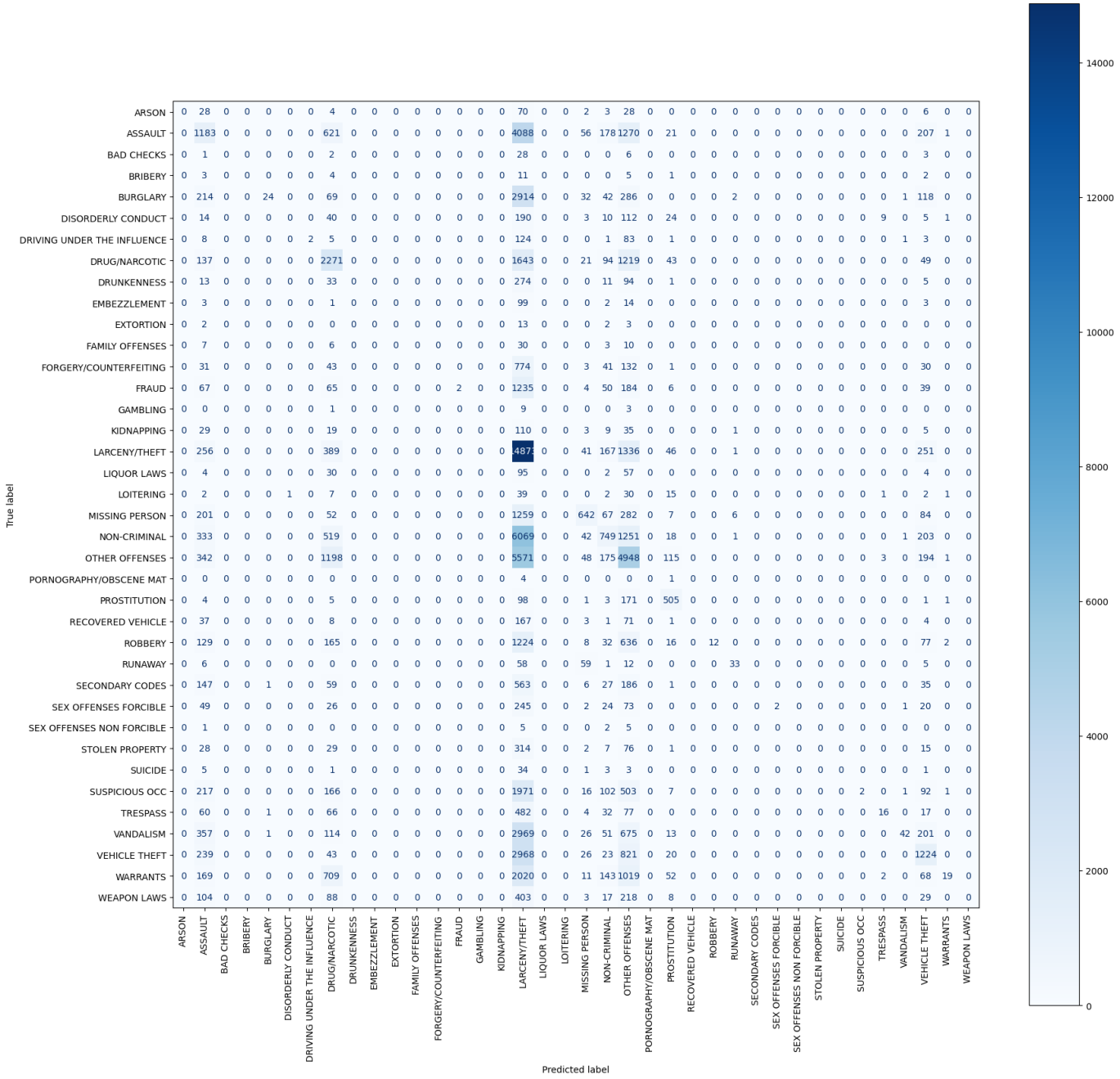
In [ ]:
```
# Se obtiene el score del modelo.
model_rf_pred = model_rf.predict(X_test)

print("Train Accuracy: ", accuracy_score(y_train, model_rf.predict(X_train)))
```

```
Train Accuracy:  0.3033166150647214
```

In [ ]:
```
# Se asigna el valor de model_rf_pred a la columna "Category" del conjunto de datos de prueba

data_test_sample['Category'] = model_rf_pred
```

In [ ]:
```
# Se muestra la matriz de confusión.

disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_train, model_rf.predict(X_t
fig, ax = plt.subplots(1,1, figsize=(20,20))
disp.plot(ax=ax, xticks_rotation=90, cmap=plt.cm.Blues)
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x24ba4d45290>
```



Aquí comienza el modelado Random Forest con todo el dataset de Train y Test

Se procede a entrenar el modelo de Random Forest, con los datos de Train.

In [ ]:
```
# Todo el dataset
model_rf_total = RandomForestClassifier(n_estimators=100, criterion='entropy', max_depth=10, r

X_train_total = data_train.drop(['Category'], axis=1)
y_train_total = data_train['Category']
X_test_total = data_test
```

In [ ]:
```
# Se entrena el modelo con todo el dataset.
model_rf_total.fit(X_train_total, y_train_total)
```

Out[ ]:
```
RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_leaf=2,
                       min_samples_split=4, random_state=1)
```

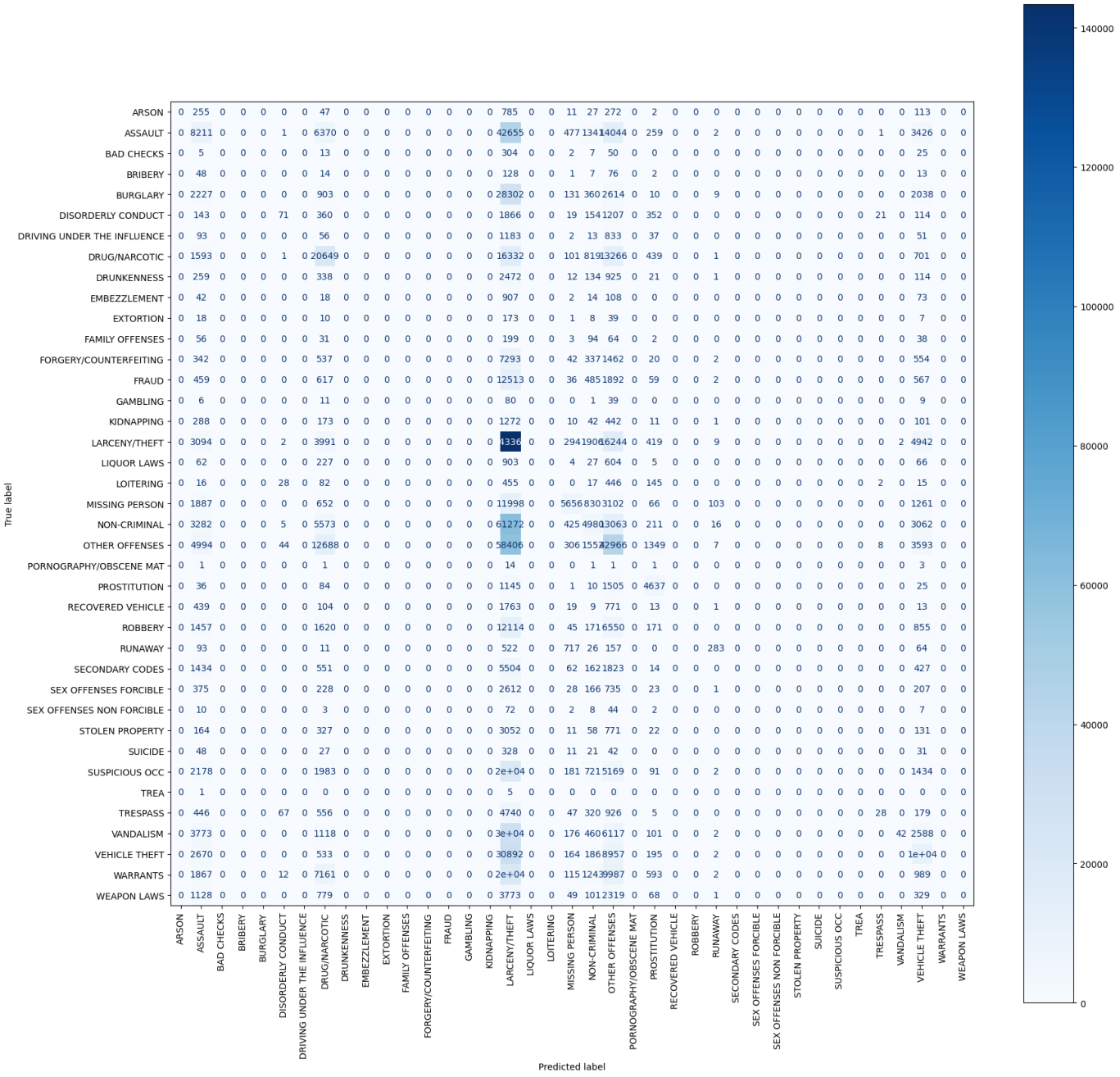
In [ ]:
```
# Se obtiene el score del modelo.
model_rf_pred_total = model_rf.predict(X_test_total)
```

```
print("Train Accuracy: ", accuracy_score(y_train_total, model_rf_total.predict(X_train_total)
```

```
Train Accuracy:  0.2752779078933839
```

In [ ]:
```
data_test['Category'] = model_rf_pred_total

# Se muestra la matriz de confusión.

disp_total = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_train_total, model_rf_
fig, ax = plt.subplots(1,1, figsize=(20,20))
disp_total.plot(ax=ax, xticks_rotation=90, cmap=plt.cm.Blues)
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x24ba9182050>



Naive Bayes

Se procede a entrenar el modelo de Naive Bayes, con los datos de Train.

Aquí comienza el modelado Naive Bayes con el dataset reducido de Train y Test

In [ ]:
```
# Se realiza el modelo de Naive Bayes reducido.
# Se setea el valor de Laplace Correction en true.
```

```
model_nb = GaussianNB()
```

In [ ]:
```
# Se indican los valores de los dataset reducidos nuevamente.

X_train = data_train_sample.drop(['Category'], axis=1)
y_train = data_train_sample['Category']
X_test = data_test_sample
```

In [ ]:
```
# Se entrena el modelo con el dataset reducido.

model_nb.fit(X_train, y_train)
```

Out[ ]:
```
▾ GaussianNB

GaussianNB()
```

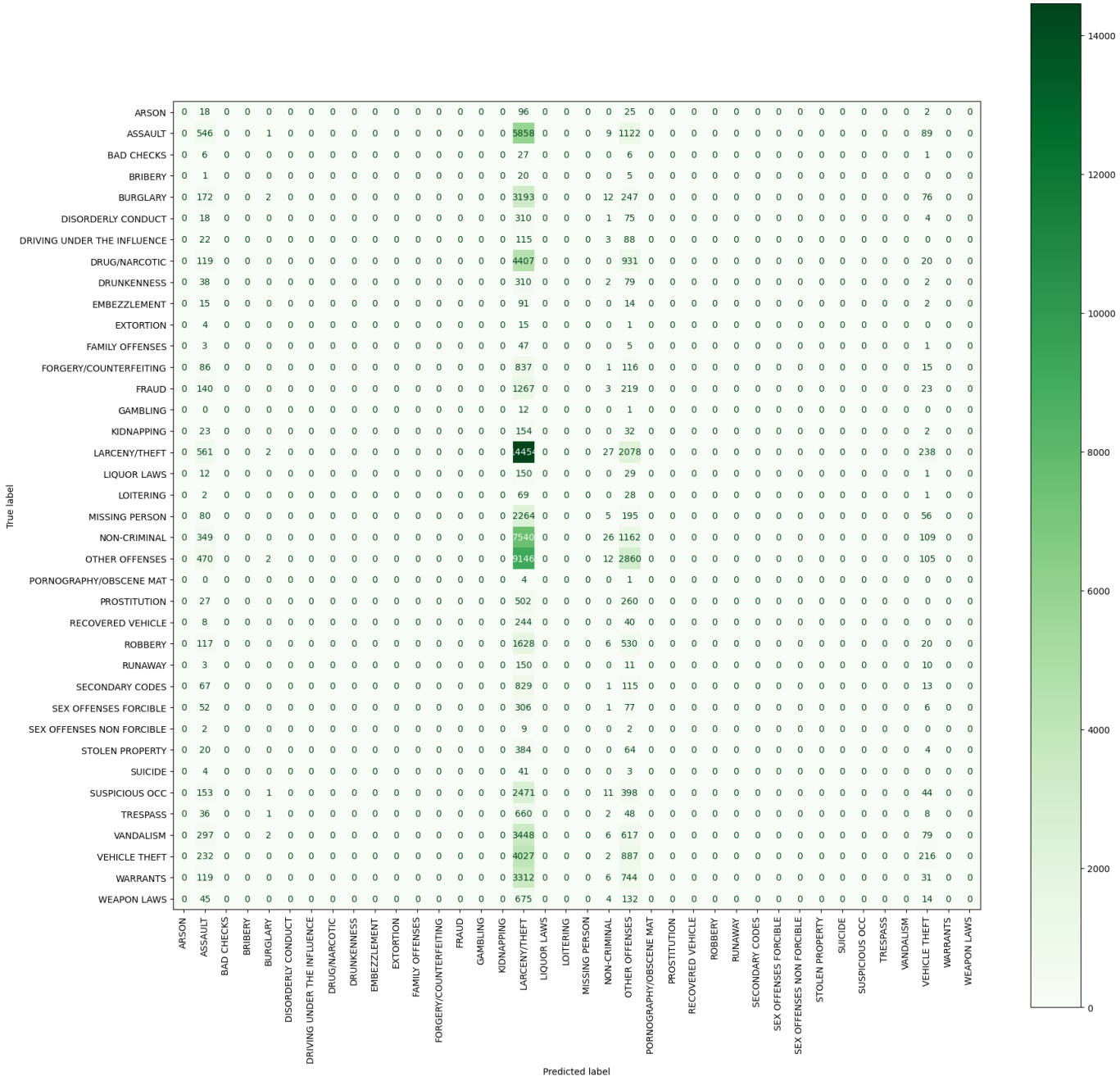
In [ ]:
```
# Se obtiene el score del modelo.
model_nb_pred = model_nb.predict(X_test)
```

In [ ]:
```
print("Train Accuracy: ", accuracy_score(y_train, model_nb.predict(X_train)))

# Se asigna el valor de model_nb_pred a la columna "Category" del conjunto de datos de prueba

data_test_sample['Category'] = model_nb_pred
```

```
Train Accuracy:  0.20683430634418307
```

In [ ]:
```
# Se muestra la matriz de confusión.

disp_nb = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_train, model_nb.predict(
fig, ax = plt.subplots(1,1, figsize=(20,20))
disp_nb.plot(ax=ax, xticks_rotation=90, cmap=plt.cm.Greens)
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x24c053fb950>
```



Aquí comienza el modelado Naive Bayes con todo el dataset de Train y Test

Se procede a entrenar el modelo de Naive Bayes, con los datos de Train.

In [ ]:
```
# Todo el dataset

model_nb_total = GaussianNB()

X_train_total = data_train.drop(['Category'], axis=1)
y_train_total = data_train['Category']

X_test_total = data_test.drop(['Category'], axis=1)

# Se entrena el modelo con todo el dataset.

model_nb_total.fit(X_train_total, y_train_total)
```

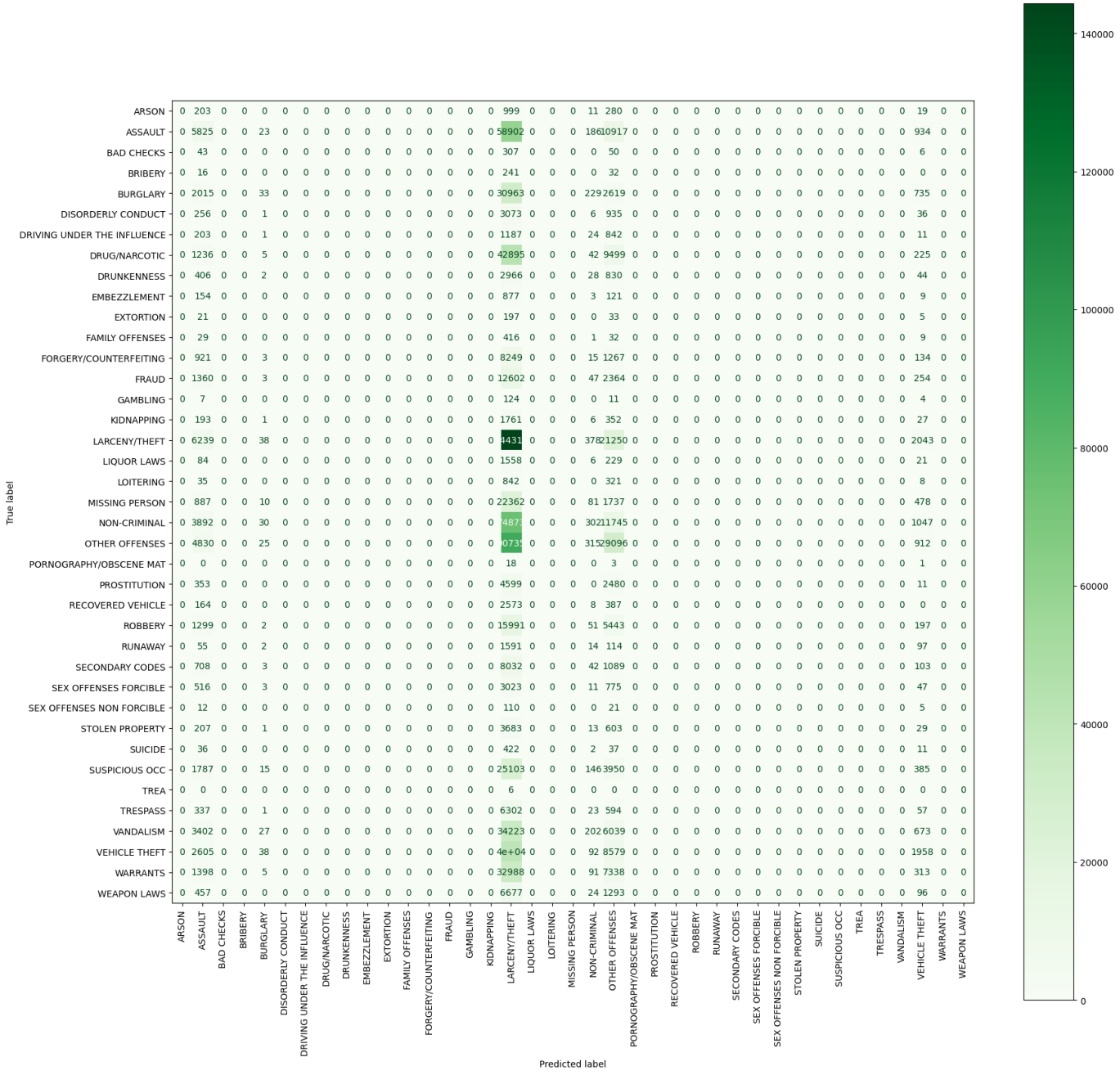
Out[ ]:
```
▾ GaussianNB

GaussianNB()
```

In [ ]:
```
# Se obtiene el score del modelo.

model_nb_pred_total = model_nb_total.predict(X_test_total)

print("Train Accuracy: ", accuracy_score(y_train_total, model_nb_total.predict(X_train_total)
```

```
Train Accuracy:  0.20739297832718298
```

In [ ]:
```
# Se asigna el valor de model_nb_pred_total a la columna "Category" del conjunto de datos de p

data_test['Category'] = model_nb_pred_total

# Se muestra la matriz de confusión.

disp_nb_total = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_train_total, model_
fig, ax = plt.subplots(1,1, figsize=(20,20))
disp_nb_total.plot(ax=ax, xticks_rotation=90, cmap=plt.cm.Greens)
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x24c0b0c0c50>



Resultados de los modelos de clasificación

Random Forest

Se procede a evaluar el modelo de Random Forest, con los datos de Test.

La precisión del modelo es de 30.33% en el conjunto de ejemplo y de 27.53% en el conjunto de completo.

Naive Bayes

Se procede a evaluar el modelo de Naive Bayes, con los datos de Test.

La precisión del modelo es de 20.68% en el conjunto de ejemplo y de 20.74% en el conjunto de completo.

In [ ]:
```
# Se muestran los top 10 resultados del Random Forest por cantidad de crímenes.
# los datos se sacan de model_rf_pred_total

print("Top 10 resultados del Random Forest por cantidad de crímenes")
print(pd.Series(model_rf_pred_total).value_counts().head(10))
```

```
Top 10 resultados del Random Forest por cantidad de crímenes
LARCENY/THEFT     546554
OTHER OFFENSES    160512
DRUG/NARCOTIC      67542
ASSAULT            44185
VEHICLE THEFT      29537
NON-CRIMINAL       16071
MISSING PERSON     10108
PROSTITUTION        8864
RUNAWAY              303
TRESPASS             225
Name: count, dtype: int64
```

In [ ]:
```
# De igual manera, se muestran los top 10 resultados del Naive Bayes por cantidad de crímenes

print("Top 10 resultados del Naive Bayes por cantidad de crímenes")
print(pd.Series(model_nb_pred_total).value_counts().head(10))
```

```
Top 10 resultados del Naive Bayes por cantidad de crímenes
LARCENY/THEFT     692052
OTHER OFFENSES    135111
ASSAULT            43338
VEHICLE THEFT      10968
NON-CRIMINAL        2419
BURGLARY             298
GAMBLING              76
Name: count, dtype: int64
```

In [ ]:
```
# Se realiza la clasificación de los resultados del Naive Bayes.

print("Clasificación de los resultados del Naive Bayes")
print(classification_report(y_train_total, model_nb_total.predict(X_train_total)))
```

```
Clasificación de los resultados del Naive Bayes
c:\Users\user\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\user\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

	precision	recall	f1-score	support
ARSON	0.00	0.00	0.00	1512
ASSAULT	0.14	0.08	0.10	76787
BAD CHECKS	0.00	0.00	0.00	406
BRIBERY	0.00	0.00	0.00	289
BURGLARY	0.12	0.00	0.00	36594
DISORDERLY CONDUCT	0.00	0.00	0.00	4307
DRIVING UNDER THE INFLUENCE	0.00	0.00	0.00	2268
DRUG/NARCOTIC	0.00	0.00	0.00	53902
DRUNKENNESS	0.00	0.00	0.00	4276
EMBEZZLEMENT	0.00	0.00	0.00	1164
EXTORTION	0.00	0.00	0.00	256
FAMILY OFFENSES	0.00	0.00	0.00	487
FORGERY/COUNTERFEITING	0.00	0.00	0.00	10589
FRAUD	0.00	0.00	0.00	16630
GAMBLING	0.00	0.00	0.00	146
KIDNAPPING	0.00	0.00	0.00	2340
LARCENY/THEFT	0.21	0.83	0.34	174263
LIQUOR LAWS	0.00	0.00	0.00	1898
LOITERING	0.00	0.00	0.00	1206
MISSING PERSON	0.00	0.00	0.00	25555
NON-CRIMINAL	0.13	0.00	0.01	91889
OTHER OFFENSES	0.22	0.23	0.22	125913
PORNOGRAPHY/OBSCENE MAT	0.00	0.00	0.00	22
PROSTITUTION	0.00	0.00	0.00	7443
RECOVERED VEHICLE	0.00	0.00	0.00	3132
ROBBERY	0.00	0.00	0.00	22983
RUNAWAY	0.00	0.00	0.00	1873
SECONDARY CODES	0.00	0.00	0.00	9977
SEX OFFENSES FORCIBLE	0.00	0.00	0.00	4375
SEX OFFENSES NON FORCIBLE	0.00	0.00	0.00	148
STOLEN PROPERTY	0.00	0.00	0.00	4536
SUICIDE	0.00	0.00	0.00	508
SUSPICIOUS OCC	0.00	0.00	0.00	31386
TREA	0.00	0.00	0.00	6
TRESPASS	0.00	0.00	0.00	7314
VANDALISM	0.00	0.00	0.00	44566
VEHICLE THEFT	0.18	0.04	0.06	53664
WARRANTS	0.00	0.00	0.00	42133
WEAPON LAWS	0.00	0.00	0.00	8547
accuracy			0.21	875290
macro avg	0.03	0.03	0.02	875290
weighted avg	0.11	0.21	0.11	875290

```
c:\Users\user\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```