Reporte

December 6, 2020

0.1 Proyecto 2: Accidentes viales CDMX

0.1.1 Métricas del modelo:

A continuación se muestran las métricas de nuestro mejor modelo.

Después de varias pruebas, tanto el modelo *RandomForestClassifier* como el *DecisionTree* nos dieron resultados relativamente deficientes en prácticamente todas las métricas relevantes, por lo que el modelo final que estaremos evaluando es uno de ***Regresión Logística***.

[3]:
```
#Importando lbirerías:

import pickle
import joblib
import sklearn
import random
import os
import pandas as pd
import numpy as np

from src.pipelines.magic_loop import load_transformation, filter_drop, \
                                     transformation_pipeline,␣
↪train_test_split, NUM_VARS, CAT_VARS
##Cambiar la ubicación del magic_loop

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
```

[22]:
```
from sklearn.metrics import roc_curve, roc_auc_score

import matplotlib.pyplot as plt

%matplotlib inline
```

Cargamos el modelo y los sets de prueba y entrenamiento:

[6]:
```
X_train = pickle.load(open("../output/X_train_diciembre4.pkl", 'rb'))
y_train = pickle.load(open("../output/Y_train_diciembre4.pkl", 'rb'))
X_test = pickle.load(open("../output/X_test_diciembre4.pkl", 'rb'))
y_test = pickle.load(open("../output/Y_test_diciembre4.pkl", 'rb'))
```

[7]:
```
model = joblib.load('../output/finalized_model.sav')
```

[8]:
```
#Confirmando el tamaño de los sets de prueba y entrenamiento:
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(967884, 62) (967884,) (414808, 62) (414808,)
```

[9]:
```
#Confirmando el modelo cargado:
model
```

[9]:
```
LogisticRegression(solver='sag')
```

Cargamos el dataframe generado en *Feature Engineering*:

[11]:
```
# Nos muevos un folder arriba
os.chdir("..")
```

[12]:
```
path = os.getcwd()
df = load_transformation(path)
```

```
Opening feature engineering pickle from output path
Feature Engineering pickle successfully retrieved.
```

[13]:
```
df.head()
```

[13]:

	one_hot__x0_ALVARO OBREGON	one_hot__x0_AZCAPOTZALCO	one_hot__x0_BENITO JUAREZ	one_hot__x0_COYOACAN	one_hot__x0_CUAJIMALPA	one_hot__x0_CUAUHTEMOC	one_hot__x0_GUSTAVO A. MADERO
0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0

```
3                     0.0                             0.0
4                     0.0                             0.0

   one_hot__x0_IZTACALCO  one_hot__x0_IZTAPALAPA  \
0                    0.0                     0.0
1                    0.0                     0.0
2                    0.0                     0.0
3                    0.0                     0.0
4                    0.0                     0.0

   one_hot__x0_MAGDALENA CONTRERAS  …  dia_creacion  hora_simple    sin_hr  \
0                              0.0  …          31.0         13.0 -0.258819
1                              0.0  …          31.0         13.0 -0.258819
2                              0.0  …          31.0         13.0 -0.258819
3                              0.0  …          31.0         18.0 -1.000000
4                              0.0  …          31.0         19.0 -0.965926

         cos_hr     sin_month  cos_month   sin_day   cos_day  bool_llamada  \
0 -9.659258e-01 -2.449294e-16        1.0  0.433884 -0.900969           1.0
1 -9.659258e-01 -2.449294e-16        1.0  0.433884 -0.900969           1.0
2 -9.659258e-01 -2.449294e-16        1.0  0.433884 -0.900969           1.0
3 -1.836970e-16 -2.449294e-16        1.0  0.433884 -0.900969           1.0
4  2.588190e-01 -2.449294e-16        1.0  0.433884 -0.900969           1.0

   label
0      0
1      0
2      0
3      1
4      0

[5 rows x 73 columns]
```

Generamos nuestras predicciones en el conjunto de prueba para la evaluación de métricas:

[14]:
```
predicted_labels = model.predict(X_test)
predicted_scores = model.predict_proba(X_test)
predicted_probs = pd.DataFrame(predicted_scores, columns=["probability_0",␣
↪"probability_1"])
#predicted_probs.head()
```

0.2 Reporte de Métricas:

[16]:
```
from sklearn.metrics import accuracy_score

accuracy_score(y_test, predicted_labels)
```

[16]:
```
0.8048687585581763
```

[17]:
```
#### Precision, recall, thresholds
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds_2 = precision_recall_curve(y_test,␣
 ↪predicted_scores[:,1], pos_label=1)
```

[18]:
```
thresholds_2 = np.append(thresholds_2, 1)
```

[88]:
```
#(precision.shape, recall.shape, thresholds_2.shape)
```

[19]:
```
def get_metrics_report(fpr, tpr, thresholds, precision, recall, thresholds_2):
    df_1 = pd.DataFrame({'threshold': thresholds_2,'precision': precision,
                         'recall': recall})
    df_1['f1_score'] = 2 * (df_1.precision * df_1.recall) / (df_1.precision +␣
 ↪df_1.recall)

    df_2 = pd.DataFrame({'tpr': tpr, 'fpr': fpr, 'threshold': thresholds})
    df_2['tnr'] = 1 - df_2['fpr']
    df_2['fnr'] = 1 - df_2['tpr']

    df = df_1.merge(df_2, on="threshold")

    return df
```

[23]:
```
fpr, tpr, thresholds = roc_curve(y_test, predicted_probs.probability_1,␣
 ↪pos_label=1)
```

[24]:
```
metrics_report = get_metrics_report(fpr, tpr, thresholds, precision, recall,␣
 ↪thresholds_2)
metrics_report
```

[24]:

	threshold	precision	recall	f1_score	tpr	fpr	\
0	0.007078	0.195157	1.000000	0.326579	1.000000	0.999700	
1	0.007100	0.195155	0.999988	0.326576	0.999988	0.999700	
2	0.008389	0.195196	0.999988	0.326633	0.999988	0.999440	
3	0.008395	0.195194	0.999975	0.326630	0.999975	0.999440	
4	0.008405	0.195194	0.999975	0.326630	0.999975	0.999437	
…	…	…	…	…	…	…	
172283	0.667680	0.888889	0.000198	0.000395	0.000198	0.000006	
172284	0.670937	0.941176	0.000198	0.000395	0.000198	0.000003	
172285	0.671903	0.937500	0.000185	0.000371	0.000185	0.000003	
172286	0.674224	1.000000	0.000185	0.000371	0.000185	0.000000	
172287	0.913421	1.000000	0.000012	0.000025	0.000012	0.000000	

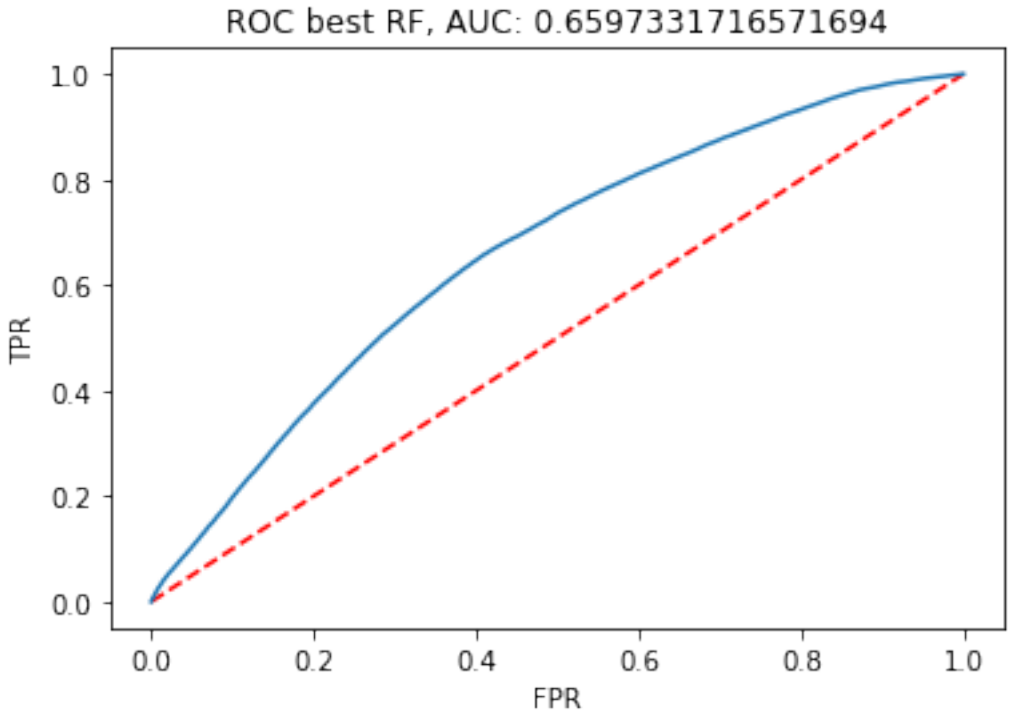
	tnr	fnr

```
0         0.000300  0.000000
1         0.000300  0.000012
2         0.000560  0.000012
3         0.000560  0.000025
4         0.000563  0.000025
…              …         …
172283    0.999994  0.999802
172284    0.999997  0.999802
172285    0.999997  0.999815
172286    1.000000  0.999815
172287    1.000000  0.999988

[172288 rows x 8 columns]
```

0.3 Curva ROC:

[25]:
```
plt.clf()
plt.plot([0,1],[0,1], 'k--', c="red")
plt.plot(fpr, tpr)
plt.title("ROC best RF, AUC: {}".format(roc_auc_score(y_test, predicted_probs.
 ↪probability_1)))
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show();
```



0.4 Matriz de confusión:

[26]:
```
from sklearn.metrics import confusion_matrix

pd.DataFrame(confusion_matrix(y_test, predicted_labels))
#Columnas = Predicción
#renglones = Realidad
```

[26]:

	0	1
0	333679	196
1	80746	187

0.5 Curva de precision y recall:

El promedio diario de llamadas se obtuvo de nuestro EDA-GEDA (554 llamadas diarias), y sabemos que contamos con 20 ambulancias al día.

[27]:
```
ambulancias_disponibles_al_dia = 20.0
num_llamadas_al_dia = 554.0
```

[28]:
```
k = ambulancias_disponibles_al_dia/num_llamadas_al_dia
print("El valor de k es igual a: ", k)
```

```
El valor de k es igual a:  0.036101083032490974
```

[29]:
```
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds_2 = precision_recall_curve(y_test,␣
 ↪predicted_probs.probability_1, pos_label=1)
```

[30]:
```
thresholds_2 = np.append(thresholds_2, 1)
```

[31]:
```
(precision.shape, recall.shape, thresholds_2.shape)
```

[31]:
```
((346813,), (346813,), (346813,))
```

[32]:
```
from sklearn.metrics import precision_score

def precision_at_k(y_true, y_scores, k):
    threshold = np.sort(y_scores)[::-1][int(k*(len(y_scores)-1))]
    y_pred = np.asarray([1 if i >= threshold else 0 for i in y_scores])

    return precision_score(y_true, y_pred)
```

[33]:
```
from sklearn.metrics import recall_score
def recall_at_k(y_true, y_scores, k):
```

```
    threshold = np.sort(y_scores)[::-1][int(k*(len(y_scores)-1))]
    y_pred = np.asarray([1 if i >= threshold else 0 for i in y_scores])

    return recall_score(y_true, y_pred)
```

[35]:
```
def pr_k_curve(y_true, y_scores, save_target=None):
    k_values = list(np.arange(0.1, 1.1, 0.1))
    pr_k = pd.DataFrame()

    for k in k_values:
        d = dict()
        d['k'] = k
        ## get_top_k es una función que ordena los scores de
        ## mayor a menor y toma los k% primeros
        #top_k = get_top_k(y_scores, k)
        #top_k = y_scores
        # print(precision_at_k(y_true, y_scores, k))
        d['precision'] = precision_at_k(y_true, y_scores, k)#(top_k)
        d['recall'] = recall_at_k(y_true, y_scores, k)#(top_k, predictions)

        pr_k = pr_k.append(d, ignore_index=True)

    # para la gráfica
    fig, ax1 = plt.subplots()
    ax1.plot(pr_k['k'], pr_k['precision'], label='precision')
    ax1.plot(pr_k['k'], pr_k['recall'], label='recall')
    plt.legend()

    if save_target is not None:
        plt.savefig(save_target, dpi=300)
    else:
        plt.show

    return pr_k
```

[36]:
```
precision_at_k(predicted_labels, y_test, k)
```

[36]:
```
0.0023105531736127413
```

[38]:
```
recall_at_k(predicted_labels, y_test, k)
```

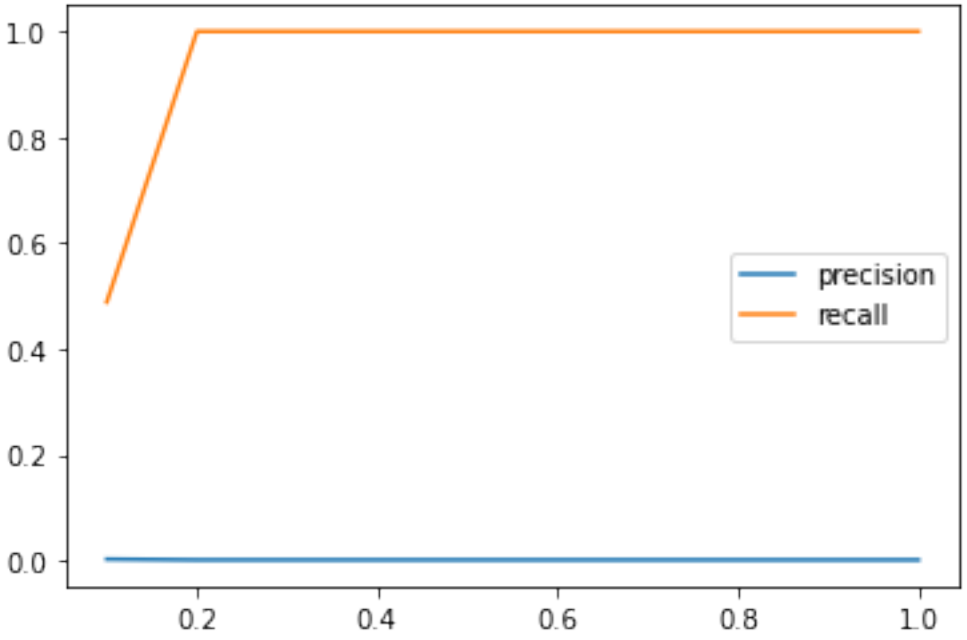
[38]:
```
0.48825065274151436
```

[39]:
```
pr_k_curve(predicted_labels, y_test)
```

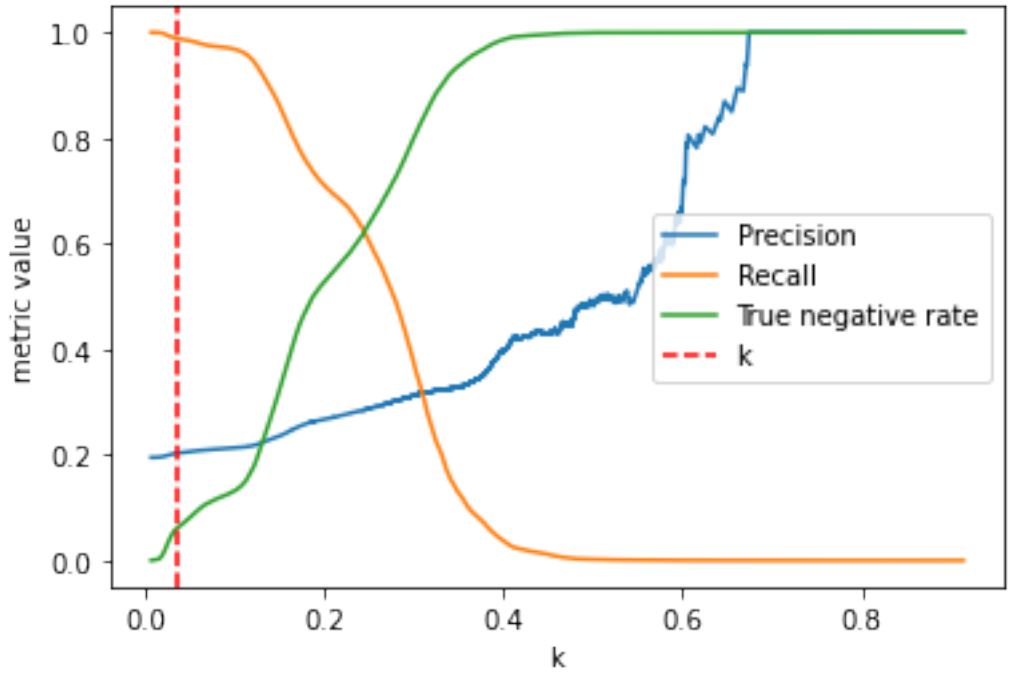
[39]:

	k	precision	recall
0	0.1	0.002311	0.488251

```
1  0.2   0.000923  1.000000
2  0.3   0.000923  1.000000
3  0.4   0.000923  1.000000
4  0.5   0.000923  1.000000
5  0.6   0.000923  1.000000
6  0.7   0.000923  1.000000
7  0.8   0.000923  1.000000
8  0.9   0.000923  1.000000
9  1.0   0.000923  1.000000
```



[47]:
```
plt.plot(metrics_report.threshold, metrics_report.precision, label="Precision")
plt.plot(metrics_report.threshold, metrics_report.recall, label="Recall")
plt.plot(metrics_report.threshold, metrics_report.tnr, label="True negative␣
 ↪rate")
plt.axvline(x=k, color="red", linestyle="--", label="k")
plt.legend(loc="best")
plt.xlabel("k")
plt.ylabel("metric value")
plt.show();
```



Como podemos ver, para el ratio k que obtenemos (sólo 20 ambulancias que se pueden enviar al día) , tenemos un recall a nuestro tope y una precisión baja. Podemos ver que si aumentara k, recall bajaría y la precisión aumentaría, sólo que en este contexto lo importante es el recall.

¿Qué métrica escogemos?

TP = El modelo dice que es llamada falsa, y sí es llamada falsa.

TN = El modelo dice que es llamada no falsa, y no es llamada falsa**.

FP = El modelo dice que es llamada falsa, y no es llamada falsa.

FN = El modelo dice que es llamada no falsa, y sí es llamada falsa.

Decidimos **TNR** dado que nos interesa ayudar a verificar que una llamada que el modelo dice que no es falsa, realmente no es falsa, ya que probablemente se necesite enviar una ambulancia.

De acuerdo a las métricas del negocio, les interesa tener un TNR mayor al 80%. Es decir, de 10 llamadas que el modelo dice que son llamadas legítimas, por lo menos 8 deben ser realmente legítimas.

[48]:
```
negocio = metrics_report[metrics_report.tnr >= 0.8]
negocio
```

[48]:

	threshold	precision	recall	f1_score	tpr	fpr	\
123604	0.299931	0.312883	0.375693	0.341424	0.375693	0.199997	
123605	0.299932	0.312869	0.375669	0.341405	0.375669	0.199997	
123606	0.299933	0.312876	0.375669	0.341409	0.375669	0.199991	

```
123607   0.299933   0.312869  0.375656  0.341400  0.375656  0.199991
123608   0.299934   0.312872  0.375656  0.341402  0.375656  0.199988
…           …          …         …         …         …
172283   0.667680   0.888889  0.000198  0.000395  0.000198  0.000006
172284   0.670937   0.941176  0.000198  0.000395  0.000198  0.000003
172285   0.671903   0.937500  0.000185  0.000371  0.000185  0.000003
172286   0.674224   1.000000  0.000185  0.000371  0.000185  0.000000
172287   0.913421   1.000000  0.000012  0.000025  0.000012  0.000000

             tnr       fnr
123604  0.800003  0.624307
123605  0.800003  0.624331
123606  0.800009  0.624331
123607  0.800009  0.624344
123608  0.800012  0.624344
…          …         …
172283  0.999994  0.999802
172284  0.999997  0.999802
172285  0.999997  0.999815
172286  1.000000  0.999815
172287  1.000000  0.999988

[48684 rows x 8 columns]
```

aequitas_Leo

December 6, 2020

[144]:
```
import datetime
import pandas as pd
import numpy as np
import seaborn as sns
import os
import pickle
import re
import joblib
from aequitas.group import Group
from aequitas.bias import Bias
from aequitas.fairness import Fairness
from aequitas.plotting import Plot
from sklearn.metrics import confusion_matrix
```

[145]:
```
#funciones de apoyo
def load_df(path):
    """
    Recibe el path en donde se encuentra el pickle que se quiere volver a␣
 ↪cargar.
    """
    # Recuperar el pickle
    pkl = pickle.load(open(path, "rb"))

    return pkl


def load_features(path):
    """
    Cargar pickle que se generó durante la transformación
    :param path: Path donde se encuentra el pickle
    :return:
    """
    print("Opening feature engineering pickle from output path")
    output_path = os.path.join(path, "output", "fe_df.pkl")

    # Recuperar el pickle
    incidentes_pkl = load_df(output_path)
```

```
    print("Feature Engineering pickle successfully retrieved.")

    return incidentes_pkl

def filter_drop(df):
    """
    Función para elegir variables relevantes y tirar los datos vacíos de␣
 ↪latitud y longitud.
    :param df: dataframe a transformar
    :return df: dataframe con las columnas relevantes y sin datos nulos
    """
    print("Dropping columns and Nan's (don't worry, it'll be ok)")
    #solo por seguridad nos aseguramos que estén ordenadas (aunque ya están)
    df = df.sort_values(by=["año_creacion", "mes_creacion", "dia_creacion",
                            "hora_simple"])
    return df.drop(columns=["año_creacion", "mes_creacion", "dia_creacion"]).
 ↪dropna()

def train_test_split(df, test_size=.70):
    """
    Función para separar en train y test el dataframe.
    Es un poco manual porque son datos temporales -- y no queremos problemas.
    :param df: dataframe a separar en train y test
    :param test_size: fracción entre 0 y 1 que nos permita separar el dataframe.
 ↪ El default es .70
    :return X_train, y_train, X_test, y_test: los 4 fantásticos.
    """
    if test_size <= 0 or test_size >= 1:
        raise ValueError("Test Size Error. Pick a test size between 0 and 1.")

    #separar features y labels
    print("Performing train test split")
    X = df.drop(columns=["label"]).copy()
    Y = df.label.copy()
    lim = round(df.shape[0] * test_size)  # 70% de train
    X_train, X_test = X[:lim], X[lim:]
    y_train, y_test = Y[:lim], Y[lim:]
    print("Train test split successfully performed")
    return X_train, y_train, X_test, y_test
```

1 Análisis de Sesgo con Aequitas.

El propósito de este notebook es realizar un análisis de sesgo de nuestro modelo utilizando el *framework* de Aequitas.

Métricas elegidas: Con base en en el "Fairness Tree" de Aequitas, y tomando en cuenta que nuestro modelo es un modelo **asistivo**, decidimos elegir las siguientes tres métricas:

- PPR (dado que nos interesa medir la proporción de llamadas que atendemos por colonia)
- FOR Parity (dado que nos interesa apoyar a quienes no estén recibiendo apoyo)
- FNR Parity (dado que nos interesa apoyar con ambulancias a quienes más lo necesiten)

Setup del dataframe. Aequitas necesita un dataframe con cierto número de columnas. Haremos eso aquí.

Pero primero recuperaremos el modelo.

[146]:
```
def load_selected_model(path):
    """
    Function to retrieve the model obtained using the magic loop.
    :param path: path to your model
    :param model: name of your model
    :return model: magic model
    """
    modelo = joblib.load(path)
    #lo hice con joblib;
    return modelo
```

[147]:
```
PATH = '/Users/Leo/Documents/MCD/1o/IDS/Proyectos/proyecto1/Notebooks/
 ↪finalized_model.sav'
model = load_selected_model(PATH)
model
```

[147]:
```
LogisticRegression(solver='sag')
```

Posteriormente recuperamos el set de pruebas.

[148]:
```
X_test = pickle.load(open("/Users/Leo/Documents/MCD/1o/IDS/Proyectos/proyecto1/
 ↪Notebooks/X_test_diciembre4.pkl", "rb"))
y_test = pickle.load(open("/Users/Leo/Documents/MCD/1o/IDS/Proyectos/proyecto1/
 ↪Notebooks/y_test_diciembre4.pkl", "rb"))
```

Ahora podemos predecir en el conjunto de prueba y generar las etiquetas.

[149]:
```
predicted_labels = model.predict(X_test)
predicted_scores = model.predict_proba(X_test)
predicted_probs = pd.DataFrame(predicted_scores, columns=["probability_0",␣
 ↪"probability_1"])
predicted_probs.head()
```

[149]:

	probability_0	probability_1
0	0.751324	0.248676
1	0.733723	0.266277
2	0.847460	0.152540

```
3       0.733723        0.266277
4       0.714466        0.285534
```

[150]:
```
sum(predicted_labels)
```

[150]:
```
383
```

Dado el punto de corte de **0.2903** que seleccionamos anteriormente, podemos generar nuevas etiquetas.

[151]:
```
punto_corte = .2903
new_labels = [0 if score < punto_corte else 1 for score in predicted_probs.
 ↪probability_1]
```

[152]:
```
sum(new_labels)
```

[152]:
```
114477
```

Podemos ver que con éste nuevo punto de corte, se genera una mayor cantidad de etiquetas positivas.

Ya casi estamos listos para el análisis. Ya tenemos nuestras etiquetas, las etiquetas reales *(y_test)*, ahora solo nos falta la columna de nombres de las delegaciones.

[153]:
```
#Funciones tomadas del notebook de EDA-GEDA, para generar el dataframe original
#con las delegaciones en el orden requerido:
CAT_COLS = ["dia_semana", "codigo_cierre", "año_cierre", "mes_cierre", "mes",␣
 ↪"delegacion_inicio",
            "incidente_c4", "clas_con_f_alarma", "tipo_entrada",␣
 ↪"delegacion_cierre", "hora_creacion",
           "hora_cierre"]

DATE_COLS = ["fecha_creacion", "fecha_cierre"]

NUM_COLS = ["latitud", "longitud"]

def ingest_file(file_name):
    """
    Function to retrieve and return the accidents dataset.
    Parameters:
    -----------
    file_name: str
               Path to the file.
    Returns:
    --------
    df: pandas dataframe
    """
    df = pd.read_csv(file_name)
    return df
```

```
def drop_cols(df):
    """
    Function to drop unnnecesary columns in the dataset.
    """
    df.drop(columns = ['folio', 'geopoint', 'mes', 'mes_cierre', 'hora_cierre',␣
 ↪'año_cierre'], inplace = True)
    return df



def fill_na(df):
    """
    Function to fill null values in a dataframe.
    """
    #aquí podemos ir agregando más cosas cuando descubramos
    #cómo imputar valores faltantes para latitud y longitud
    df.fillna({
        'delegacion_inicio': 'No Disponible',
        'delegacion_cierre': 'No Disponible'
              }, inplace = True)
    return df


def categoric_transformation(col,df):
    df[col] = df[col].astype("category")
    return df

def create_categorical(cols, df):
    """
    Function to transform and prepare the categorical features in the dataset.
    """
    #transform to appropriate data type
    for col in cols:
        df = categoric_transformation(col, df)

    return df


def date_transformation(col,df):
    """
    Function to prepare and transform date-type columns.
    """
    df[col] = pd.to_datetime(df[col], dayfirst=True)
    return df
```

```python
def create_date_cols(cols, df):
    for col in cols:
        df = date_transformation(col, df)
    return df



def generate_label(df):
    """
    Function to create a new column indicating whether there was
    a false alarm or not.
    Parameters:
    -----------
    df: pandas dataframe

    Returns:
    --------
    df: pandas dataframe
    """
    #transformamos la columna para solo quedarnos con la letra del código
    df["codigo_cierre"] = df["codigo_cierre"].apply(lambda x: x[1])
    df['label'] = np.where(
        (df.codigo_cierre == 'F') | (df.codigo_cierre == 'N'), 1, 0)
    return df


def clean_hora_creacion(df):
    """
    Function to transform hours with incorrect format to timedelta format.
    """
    horas_raw = df.hora_creacion.values.tolist()
    horas_clean = [datetime.timedelta(days=float(e)) if e.startswith("0.") else␣
 ↪e for e in horas_raw]
    df["hora_creacion"] = horas_clean
    return df


def create_simple_hour(df):
    """
    Function to extract the hour from the column "hora_creacion"
    Parameters:
    -----------
    df: pandas dataframe

    Returns:
    ---------
    df: pandas dataframe with a new column indicating the hour.
```

```
    """
    #la función se podria adaptar para devolver minuto o segundo pero no lo␣
↪considero necesario
    pattern = '\d+' #encuentra uno o más dígitos
    horas_raw = df.hora_creacion.astype(str).values #son así: '22:35:04', '22:
↪50:49', '09:40:11'
    n = len(horas_raw)
    horas_clean = [0]*n #es más rápido reasignar valores que hacer .append()
    for i in range(n):
        hora_raw = horas_raw[i]
        hora_clean = re.match(pattern, hora_raw)[0] #solo queremos la hora,␣
↪esto devuelve un objeto
        horas_clean[i] = hora_clean

    df["hora_simple"] = horas_clean
    return df


def add_date_columns(df):
    """
    Esta función es muy importante puesto que nos ayudará a crear el mes, día y␣
↪año de creación
    del registro. De esta manera podemos prescindir de las fechas de cierre,␣
↪que no tendríamos en tiempo
    real en un modelo.
    Parameters:
    -----------
    df: pandas dataframe

    Returns:
    ---------
    df: pandas dataframe with 4 new columns
    """
    mapping_meses = {1: "Enero", 2: "Febrero", 3: "Marzo", 4: "Abril", 5:␣
↪"Mayo",
                     6: "Junio", 7: "Julio", 8: "Agosto", 9: "Septiembre", 10:
↪ "Octubre",
                     11: "Noviembre", 12: "Diciembre"}


    df["año_creacion"] = df.fecha_creacion.dt.year
    df["mes_creacion"] = df.fecha_creacion.dt.month
    df["dia_creacion"] = df.fecha_creacion.dt.day
    df["mes_creacion_str"] = df.mes_creacion.map(mapping_meses)
    df["año_creacion"] = df["año_creacion"].astype(str)
    return df
```

```python
def create_time_blocks(df):
    """
    Function to group the hour of the day into 3-hour blocks.
    Parameters:
    -----------
    df: pandas dataframe

    Returns:
    ---------
    df: pandas dataframe with a new column indicating the time-block.
    """
    horas_int = set(df.hora_simple.astype(int).values) #estaba como categórico
    f = lambda x: 12 if x == 0 else x
    mapping_hours = {}
    for hora in horas_int:
        grupo = (hora // 3) * 3
        if grupo < 12:
            nombre_grupo = str(f(grupo)) + "-" + str(grupo + 2) + " a.m."
        else:
            hora_tarde = grupo % 12
            nombre_grupo = str(f(hora_tarde)) + "-" + str(hora_tarde + 2) + " p.
 ↪m."
        mapping_hours[hora] = nombre_grupo

    df["espacio_del_dia"] = df["hora_simple"].astype(int).map(mapping_hours)
    return df




def basic_preprocessing(path):
    """
    Function to summarize all the preprocessing done to the data.
    Parameters:
    -----------
    path: str
          Path to your file

    Returns:
    ---------
    df: pandas dataframe
    """
    df = ingest_file(path)
    df = generate_label(df)
```

```
    df = fill_na(df)
    df = clean_hora_creacion(df)
    df = create_categorical(CAT_COLS, df) #transform to appropriate data types
    df = create_date_cols(DATE_COLS, df)
    df = add_date_columns(df)
    df = create_simple_hour(df)
    df = create_time_blocks(df)
    df = drop_cols(df)

    return df
```

[154]:
```
data = basic_preprocessing('/Users/Leo/Documents/MCD/1o/IDS/Proyectos/proyecto1/
↪datos/incidentes-viales-c5.csv')
```

[155]:
```
data = filter_drop(data)
```

```
Dropping columns and Nan's (don't worry, it'll be ok)
```

[156]:
```
delegaciones = data.delegacion_inicio.iloc[967884:]
```

Confirmamos que nuestros datos tengan las dimensiones correctas:

[157]:
```
print(X_test.shape, y_test.shape, len(new_labels), len(delegaciones))
```

```
(414808, 62) (414808,) 414808 414808
```

Creamos el dataframe con el que se hará el análisis:

[158]:
```
aeq_df = pd.DataFrame({"score": new_labels,
                       "label_value": y_test.values,
                       "delegacion": delegaciones}
                     ).reset_index().iloc[:,1:]
aeq_df.head()
```

[158]:

	score	label_value	delegacion
0	0	0	CUAUHTEMOC
1	0	0	CUAJIMALPA
2	0	1	MIGUEL HIDALGO
3	0	0	CUAJIMALPA
4	0	0	MIGUEL HIDALGO

Damos formato a la columna `delegacion`:

[159]:
```
aeq_df.delegacion = aeq_df.delegacion.apply(lambda x: x.lower().title())
aeq_df["delegacion"] = aeq_df["delegacion"].astype(str)
aeq_df.head()
```

[159]:

	score	label_value	delegacion
0	0	0	Cuauhtemoc
1	0	0	Cuajimalpa
2	0	1	Miguel Hidalgo
3	0	0	Cuajimalpa
4	0	0	Miguel Hidalgo

Eso es todo. Podemos proceder a realizar el análisis de sesgo.

[160]:
```
print(f"Predicciones positivas: {aeq_df['score'].astype(int).sum():,}")
print(f"Etiquetas Positivas y_test: {aeq_df['label_value'].astype(int).sum():
 ↪,}")
print(f"Total de observaciones: {len(aeq_df):,}")
```

```
Predicciones positivas: 114,477
Etiquetas Positivas y_test: 80,933
Total de observaciones: 414,808
```

1.1 Análisis con Group:

[161]:
```
g = Group()
xtab, attrbs = g.get_crosstabs(aeq_df)
xtab
```

```
model_id, score_thresholds 0 {'rank_abs': [114477]}
```

[161]:

	model_id	score_threshold	k	attribute_name	attribute_value	\
0	0	binary 0/1	114477	delegacion	Alvaro Obregon	
1	0	binary 0/1	114477	delegacion	Azcapotzalco	
2	0	binary 0/1	114477	delegacion	Benito Juarez	
3	0	binary 0/1	114477	delegacion	Coyoacan	
4	0	binary 0/1	114477	delegacion	Cuajimalpa	
5	0	binary 0/1	114477	delegacion	Cuauhtemoc	
6	0	binary 0/1	114477	delegacion	Gustavo A. Madero	
7	0	binary 0/1	114477	delegacion	Iztacalco	
8	0	binary 0/1	114477	delegacion	Iztapalapa	
9	0	binary 0/1	114477	delegacion	Magdalena Contreras	
10	0	binary 0/1	114477	delegacion	Miguel Hidalgo	
11	0	binary 0/1	114477	delegacion	Milpa Alta	
12	0	binary 0/1	114477	delegacion	No Disponible	
13	0	binary 0/1	114477	delegacion	Tlahuac	
14	0	binary 0/1	114477	delegacion	Tlalpan	
15	0	binary 0/1	114477	delegacion	Venustiano Carranza	
16	0	binary 0/1	114477	delegacion	Xochimilco	

	tpr	tnr	for	fdr	fpr	…	pprev	fp	\
0	0.543814	0.678411	0.147988	0.695992	0.321589	…	0.367207	7658	
1	0.122367	0.945485	0.155179	0.692437	0.054515	…	0.065724	824	

2	0.133244	0.938898	0.156398	0.695443	0.061102	…	0.073167	1587
3	0.558788	0.682697	0.145367	0.683298	0.317303	…	0.367617	7957
4	0.514692	0.679442	0.215538	0.618188	0.320558	…	0.374490	1815
5	0.118832	0.945976	0.145214	0.713695	0.054024	…	0.064020	1897
6	0.409298	0.777455	0.152596	0.696430	0.222545	…	0.258326	7433
7	0.401696	0.771330	0.153255	0.709274	0.228670	…	0.261405	3235
8	0.581087	0.647450	0.149358	0.690950	0.352550	…	0.401330	19178
9	0.636872	0.638429	0.154435	0.638732	0.361571	…	0.428485	1814
10	0.241119	0.853674	0.173193	0.720311	0.146326	…	0.164403	3706
11	0.506073	0.722584	0.154921	0.671485	0.277416	…	0.325771	511
12	0.875000	0.777778	0.300000	0.086957	0.222222	…	0.696970	2
13	0.566254	0.696828	0.144816	0.663075	0.303172	…	0.359437	2523
14	0.595371	0.629363	0.137383	0.715347	0.370637	…	0.415255	8856
15	0.440281	0.763755	0.144134	0.700151	0.236245	…	0.274370	5569
16	0.571698	0.640691	0.158503	0.690459	0.359309	…	0.405997	4740

	fn	tn	tp	group_label_pos	group_label_neg	group_size	\
0	2806	16155	3345	6151	23813	29964	
1	2625	14291	366	2991	15115	18106	
2	4521	24386	695	5216	25973	31189	
3	2912	17120	3688	6600	25077	31677	
4	1057	3847	1121	2178	5662	7840	
5	5643	33217	761	6404	35114	41518	
6	4676	25967	3240	7916	33400	41316	
7	1975	10912	1326	3301	14147	17448	
8	6184	35220	8578	14762	54398	69160	
9	585	3203	1026	1611	5017	6628	
10	4529	21621	1439	5968	25327	31295	
11	244	1331	250	494	1842	2336	
12	3	7	21	24	9	33	
13	982	5799	1282	2264	8322	10586	
14	2395	15038	3524	5919	23894	29813	
15	3032	18004	2385	5417	23573	28990	
16	1592	8452	2125	3717	13192	16909	

	total_entities	prev
0	414808	0.205280
1	414808	0.165194
2	414808	0.167238
3	414808	0.208353
4	414808	0.277806
5	414808	0.154246
6	414808	0.191596
7	414808	0.189191
8	414808	0.213447
9	414808	0.243060
10	414808	0.190701

```
11          414808  0.211473
12          414808  0.727273
13          414808  0.213867
14          414808  0.198538
15          414808  0.186858
16          414808  0.219824

[17 rows x 26 columns]
```

[162]:
```
#Métricas absolutas
absolute_metrics = g.list_absolute_metrics(xtab)
#absolute_metrics
```

1.1.1 Tabla de conteo de frecuencias por Group:

[163]:
```
xtab[[col for col in xtab.columns if col not in absolute_metrics]]
```

[163]:

	model_id	score_threshold	k	attribute_name	attribute_value
0	0	binary 0/1	114477	delegacion	Alvaro Obregon
1	0	binary 0/1	114477	delegacion	Azcapotzalco
2	0	binary 0/1	114477	delegacion	Benito Juarez
3	0	binary 0/1	114477	delegacion	Coyoacan
4	0	binary 0/1	114477	delegacion	Cuajimalpa
5	0	binary 0/1	114477	delegacion	Cuauhtemoc
6	0	binary 0/1	114477	delegacion	Gustavo A. Madero
7	0	binary 0/1	114477	delegacion	Iztacalco
8	0	binary 0/1	114477	delegacion	Iztapalapa
9	0	binary 0/1	114477	delegacion	Magdalena Contreras
10	0	binary 0/1	114477	delegacion	Miguel Hidalgo
11	0	binary 0/1	114477	delegacion	Milpa Alta
12	0	binary 0/1	114477	delegacion	No Disponible
13	0	binary 0/1	114477	delegacion	Tlahuac
14	0	binary 0/1	114477	delegacion	Tlalpan
15	0	binary 0/1	114477	delegacion	Venustiano Carranza
16	0	binary 0/1	114477	delegacion	Xochimilco

	pp	pn	fp	fn	tn	tp	group_label_pos	group_label_neg
0	11003	18961	7658	2806	16155	3345	6151	23813
1	1190	16916	824	2625	14291	366	2991	15115
2	2282	28907	1587	4521	24386	695	5216	25973
3	11645	20032	7957	2912	17120	3688	6600	25077
4	2936	4904	1815	1057	3847	1121	2178	5662
5	2658	38860	1897	5643	33217	761	6404	35114
6	10673	30643	7433	4676	25967	3240	7916	33400
7	4561	12887	3235	1975	10912	1326	3301	14147
8	27756	41404	19178	6184	35220	8578	14762	54398
9	2840	3788	1814	585	3203	1026	1611	5017

10	5145	26150	3706	4529	21621	1439	5968	25327
11	761	1575	511	244	1331	250	494	1842
12	23	10	2	3	7	21	24	9
13	3805	6781	2523	982	5799	1282	2264	8322
14	12380	17433	8856	2395	15038	3524	5919	23894
15	7954	21036	5569	3032	18004	2385	5417	23573
16	6865	10044	4740	1592	8452	2125	3717	13192

	group_size	total_entities
0	29964	414808
1	18106	414808
2	31189	414808
3	31677	414808
4	7840	414808
5	41518	414808
6	41316	414808
7	17448	414808
8	69160	414808
9	6628	414808
10	31295	414808
11	2336	414808
12	33	414808
13	10586	414808
14	29813	414808
15	28990	414808
16	16909	414808

1.1.2 Tabla de conteo de absolutos por Group:

[164]:
```
xtab[['attribute_name', 'attribute_value']+[col for col in xtab.columns if col␣
 ↪in absolute_metrics]].round(2)
```

[164]:

	attribute_name	attribute_value	tpr	tnr	for	fdr	fpr	fnr	\
0	delegacion	Alvaro Obregon	0.54	0.68	0.15	0.70	0.32	0.46	
1	delegacion	Azcapotzalco	0.12	0.95	0.16	0.69	0.05	0.88	
2	delegacion	Benito Juarez	0.13	0.94	0.16	0.70	0.06	0.87	
3	delegacion	Coyoacan	0.56	0.68	0.15	0.68	0.32	0.44	
4	delegacion	Cuajimalpa	0.51	0.68	0.22	0.62	0.32	0.49	
5	delegacion	Cuauhtemoc	0.12	0.95	0.15	0.71	0.05	0.88	
6	delegacion	Gustavo A. Madero	0.41	0.78	0.15	0.70	0.22	0.59	
7	delegacion	Iztacalco	0.40	0.77	0.15	0.71	0.23	0.60	
8	delegacion	Iztapalapa	0.58	0.65	0.15	0.69	0.35	0.42	
9	delegacion	Magdalena Contreras	0.64	0.64	0.15	0.64	0.36	0.36	
10	delegacion	Miguel Hidalgo	0.24	0.85	0.17	0.72	0.15	0.76	
11	delegacion	Milpa Alta	0.51	0.72	0.15	0.67	0.28	0.49	
12	delegacion	No Disponible	0.88	0.78	0.30	0.09	0.22	0.12	
13	delegacion	Tlahuac	0.57	0.70	0.14	0.66	0.30	0.43	

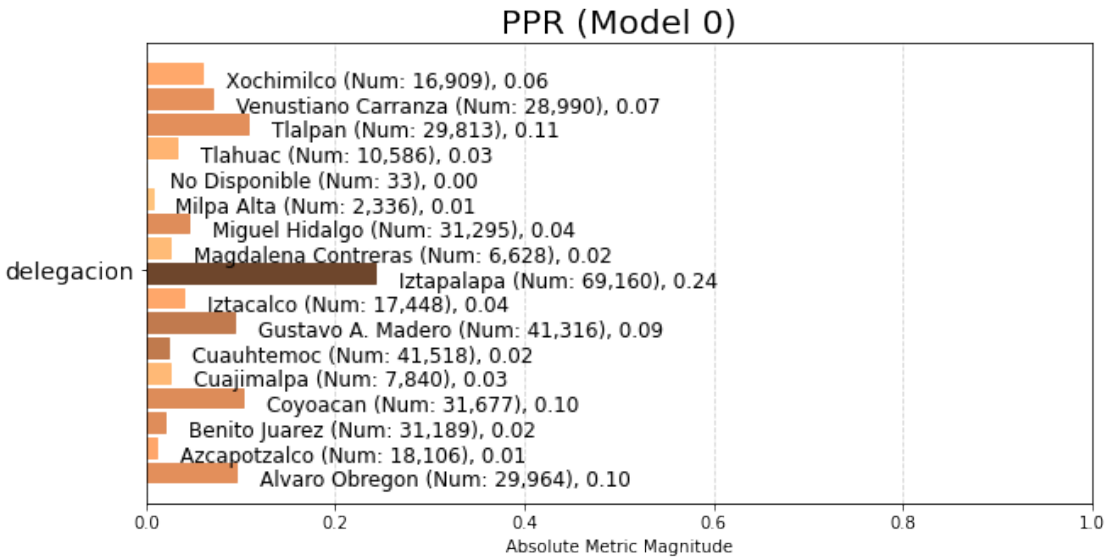
```
14     delegacion              Tlalpan  0.60  0.63  0.14  0.72  0.37  0.40
15     delegacion  Venustiano Carranza  0.44  0.76  0.14  0.70  0.24  0.56
16     delegacion           Xochimilco  0.57  0.64  0.16  0.69  0.36  0.43
```

	npv	precision	ppr	pprev	prev
0	0.85	0.30	0.10	0.37	0.21
1	0.84	0.31	0.01	0.07	0.17
2	0.84	0.30	0.02	0.07	0.17
3	0.85	0.32	0.10	0.37	0.21
4	0.78	0.38	0.03	0.37	0.28
5	0.85	0.29	0.02	0.06	0.15
6	0.85	0.30	0.09	0.26	0.19
7	0.85	0.29	0.04	0.26	0.19
8	0.85	0.31	0.24	0.40	0.21
9	0.85	0.36	0.02	0.43	0.24
10	0.83	0.28	0.04	0.16	0.19
11	0.85	0.33	0.01	0.33	0.21
12	0.70	0.91	0.00	0.70	0.73
13	0.86	0.34	0.03	0.36	0.21
14	0.86	0.28	0.11	0.42	0.20
15	0.86	0.30	0.07	0.27	0.19
16	0.84	0.31	0.06	0.41	0.22

1.1.3 Visualización de métricas por Group:

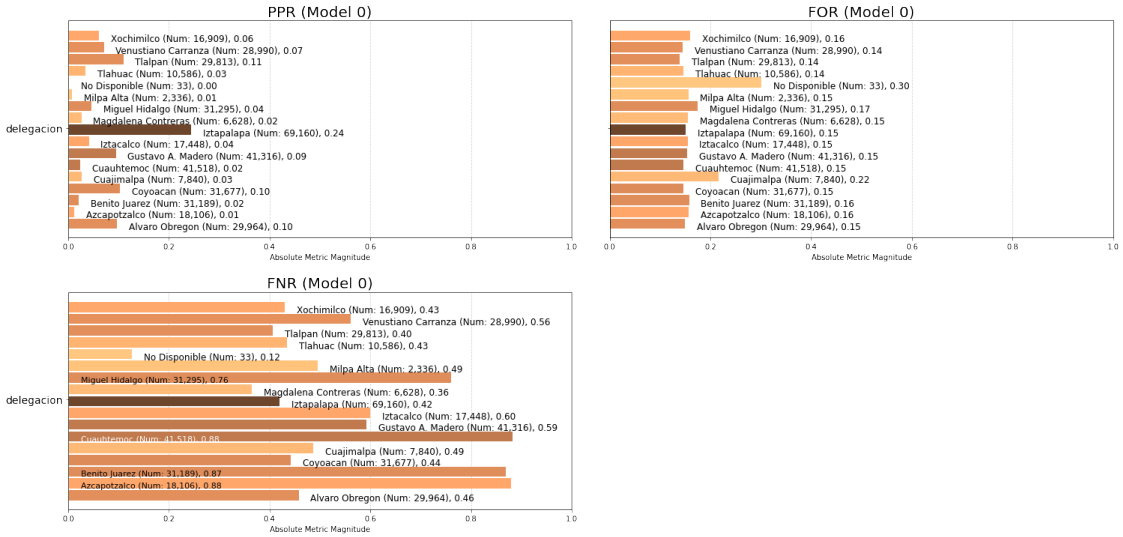
[166]:
```
#En aequitas nuestras métricas son PPR, FOR y FNR.
aeq = Plot()
```

[167]:
```
fdr = aeq.plot_group_metric(xtab, 'ppr')
```



Podemos ver que, excepto por Iztapalapa, en general la mayoría de las colonias tienen una proporción baja de predicciones positivas.

[168]:
```
p = aeq.plot_group_metric_all(xtab, metrics=['ppr','for','fnr'], ncols = 2)
```




```
<Figure size 432x288 with 0 Axes>
```

Interpretación de cada métrica seleccionada: * **PPR** (*Predicted Positive Rate*): Como ya se había mencionado, Iztapalapa es la delegación con más predicción de positivos, por un margen bastante alto. * **FOR** (*False Omission Rate*): Aún contando con pocas observaciones, la categoría "No disponible" de las delegaciones, es la que cuenta con la *FOR* más alta. En segundo lugar se encuentra Cuajimalpa, lo que hace sentido, puesto que cuenta con una de las tasas más altas de llamadas falsas (observado en nuestro EDA/GEDA), por lo que podemos tener llamadas con etiqueta negativa (llamada verdadera), cuando en realidad debía ser una etiqueta positiva (llamada falsa). * **FNR** (*False Negative Rate*): Llama la atención que la tasa de FNR es muy alta para prácticamente todas las delegaciones. Esto es preocupante puesto que al tratarse de falsos negativos (llamadas falsas clasificadas como verdaderas), se pueden estar mandando ambulancias a eventos donde no es necesario. Desgraciadamente, es un comportamiento esperado dada la baja precisión de nuestro modelo.

Mostramos la matriz de confusión como referencia:

[169]:
```
pd.DataFrame(confusion_matrix(y_test, new_labels))
```

[169]:

	0	1
0	254570	79305
1	45761	35172

1.1.4 Análisis de Sesgo.

Primero pondremos como referencia a **Iztalapa** por ser la colonia con más predicciones positivas

[170]:
```
bias = Bias()
```

[171]:
```
bdf = bias.get_disparity_predefined_groups(xtab, original_df = aeq_df,
                                ref_groups_dict = {'delegacion': 'Iztapalapa'},
                                alpha=0.05)
```

`get_disparity_predefined_group()`

[172]:
```
bdf
```

[172]:

	model_id	score_threshold	k	attribute_name	attribute_value	\
0	0	binary 0/1	114477	delegacion	Alvaro Obregon	
1	0	binary 0/1	114477	delegacion	Azcapotzalco	
2	0	binary 0/1	114477	delegacion	Benito Juarez	
3	0	binary 0/1	114477	delegacion	Coyoacan	
4	0	binary 0/1	114477	delegacion	Cuajimalpa	
5	0	binary 0/1	114477	delegacion	Cuauhtemoc	
6	0	binary 0/1	114477	delegacion	Gustavo A. Madero	
7	0	binary 0/1	114477	delegacion	Iztacalco	
8	0	binary 0/1	114477	delegacion	Iztapalapa	
9	0	binary 0/1	114477	delegacion	Magdalena Contreras	
10	0	binary 0/1	114477	delegacion	Miguel Hidalgo	
11	0	binary 0/1	114477	delegacion	Milpa Alta	
12	0	binary 0/1	114477	delegacion	No Disponible	
13	0	binary 0/1	114477	delegacion	Tlahuac	
14	0	binary 0/1	114477	delegacion	Tlalpan	
15	0	binary 0/1	114477	delegacion	Venustiano Carranza	
16	0	binary 0/1	114477	delegacion	Xochimilco	

	tpr	tnr	for	fdr	fpr	…	\
0	0.543814	0.678411	0.147988	0.695992	0.321589	…	
1	0.122367	0.945485	0.155179	0.692437	0.054515	…	
2	0.133244	0.938898	0.156398	0.695443	0.061102	…	
3	0.558788	0.682697	0.145367	0.683298	0.317303	…	
4	0.514692	0.679442	0.215538	0.618188	0.320558	…	
5	0.118832	0.945976	0.145214	0.713695	0.054024	…	
6	0.409298	0.777455	0.152596	0.696430	0.222545	…	
7	0.401696	0.771330	0.153255	0.709274	0.228670	…	
8	0.581087	0.647450	0.149358	0.690950	0.352550	…	
9	0.636872	0.638429	0.154435	0.638732	0.361571	…	
10	0.241119	0.853674	0.173193	0.720311	0.146326	…	
11	0.506073	0.722584	0.154921	0.671485	0.277416	…	
12	0.875000	0.777778	0.300000	0.086957	0.222222	…	
13	0.566254	0.696828	0.144816	0.663075	0.303172	…	

```
14  0.595371  0.629363  0.137383  0.715347  0.370637  …
15  0.440281  0.763755  0.144134  0.700151  0.236245  …
16  0.571698  0.640691  0.158503  0.690459  0.359309  …

    ppr_ref_group_value  pprev_ref_group_value  precision_ref_group_value  \
0             Iztapalapa             Iztapalapa                 Iztapalapa
1             Iztapalapa             Iztapalapa                 Iztapalapa
2             Iztapalapa             Iztapalapa                 Iztapalapa
3             Iztapalapa             Iztapalapa                 Iztapalapa
4             Iztapalapa             Iztapalapa                 Iztapalapa
5             Iztapalapa             Iztapalapa                 Iztapalapa
6             Iztapalapa             Iztapalapa                 Iztapalapa
7             Iztapalapa             Iztapalapa                 Iztapalapa
8             Iztapalapa             Iztapalapa                 Iztapalapa
9             Iztapalapa             Iztapalapa                 Iztapalapa
10            Iztapalapa             Iztapalapa                 Iztapalapa
11            Iztapalapa             Iztapalapa                 Iztapalapa
12            Iztapalapa             Iztapalapa                 Iztapalapa
13            Iztapalapa             Iztapalapa                 Iztapalapa
14            Iztapalapa             Iztapalapa                 Iztapalapa
15            Iztapalapa             Iztapalapa                 Iztapalapa
16            Iztapalapa             Iztapalapa                 Iztapalapa

    fdr_ref_group_value  for_ref_group_value  fpr_ref_group_value  \
0             Iztapalapa            Iztapalapa            Iztapalapa
1             Iztapalapa            Iztapalapa            Iztapalapa
2             Iztapalapa            Iztapalapa            Iztapalapa
3             Iztapalapa            Iztapalapa            Iztapalapa
4             Iztapalapa            Iztapalapa            Iztapalapa
5             Iztapalapa            Iztapalapa            Iztapalapa
6             Iztapalapa            Iztapalapa            Iztapalapa
7             Iztapalapa            Iztapalapa            Iztapalapa
8             Iztapalapa            Iztapalapa            Iztapalapa
9             Iztapalapa            Iztapalapa            Iztapalapa
10            Iztapalapa            Iztapalapa            Iztapalapa
11            Iztapalapa            Iztapalapa            Iztapalapa
12            Iztapalapa            Iztapalapa            Iztapalapa
13            Iztapalapa            Iztapalapa            Iztapalapa
14            Iztapalapa            Iztapalapa            Iztapalapa
15            Iztapalapa            Iztapalapa            Iztapalapa
16            Iztapalapa            Iztapalapa            Iztapalapa

    fnr_ref_group_value  tpr_ref_group_value  tnr_ref_group_value  \
0             Iztapalapa            Iztapalapa            Iztapalapa
1             Iztapalapa            Iztapalapa            Iztapalapa
2             Iztapalapa            Iztapalapa            Iztapalapa
3             Iztapalapa            Iztapalapa            Iztapalapa
```

```
4            Iztapalapa          Iztapalapa          Iztapalapa
5            Iztapalapa          Iztapalapa          Iztapalapa
6            Iztapalapa          Iztapalapa          Iztapalapa
7            Iztapalapa          Iztapalapa          Iztapalapa
8            Iztapalapa          Iztapalapa          Iztapalapa
9            Iztapalapa          Iztapalapa          Iztapalapa
10           Iztapalapa          Iztapalapa          Iztapalapa
11           Iztapalapa          Iztapalapa          Iztapalapa
12           Iztapalapa          Iztapalapa          Iztapalapa
13           Iztapalapa          Iztapalapa          Iztapalapa
14           Iztapalapa          Iztapalapa          Iztapalapa
15           Iztapalapa          Iztapalapa          Iztapalapa
16           Iztapalapa          Iztapalapa          Iztapalapa

    npv_ref_group_value
0            Iztapalapa
1            Iztapalapa
2            Iztapalapa
3            Iztapalapa
4            Iztapalapa
5            Iztapalapa
6            Iztapalapa
7            Iztapalapa
8            Iztapalapa
9            Iztapalapa
10           Iztapalapa
11           Iztapalapa
12           Iztapalapa
13           Iztapalapa
14           Iztapalapa
15           Iztapalapa
16           Iztapalapa

[17 rows x 46 columns]
```

1.1.5 Tabla de análisis de sesgo por Bias (por mayoría):

[173]:
```
majority_bdf = bias.get_disparity_major_group(xtab, original_df=aeq_df)
```

```
get_disparity_major_group()
```

[174]:
```
majority_bdf[['attribute_name', 'attribute_value'] +  bias.
↪list_disparities(majority_bdf)].round(2)
```

[174]:
```
   attribute_name      attribute_value  ppr_disparity  pprev_disparity  \
0      delegacion       Alvaro Obregon           0.40             0.91
1      delegacion         Azcapotzalco           0.04             0.16
```

2	delegacion	Benito Juarez	0.08	0.18
3	delegacion	Coyoacan	0.42	0.92
4	delegacion	Cuajimalpa	0.11	0.93
5	delegacion	Cuauhtemoc	0.10	0.16
6	delegacion	Gustavo A. Madero	0.38	0.64
7	delegacion	Iztacalco	0.16	0.65
8	delegacion	Iztapalapa	1.00	1.00
9	delegacion	Magdalena Contreras	0.10	1.07
10	delegacion	Miguel Hidalgo	0.19	0.41
11	delegacion	Milpa Alta	0.03	0.81
12	delegacion	No Disponible	0.00	1.74
13	delegacion	Tlahuac	0.14	0.90
14	delegacion	Tlalpan	0.45	1.03
15	delegacion	Venustiano Carranza	0.29	0.68
16	delegacion	Xochimilco	0.25	1.01

	precision_disparity	fdr_disparity	for_disparity	fpr_disparity	\
0	0.98	1.01	0.99	0.91	
1	1.00	1.00	1.04	0.15	
2	0.99	1.01	1.05	0.17	
3	1.02	0.99	0.97	0.90	
4	1.24	0.89	1.44	0.91	
5	0.93	1.03	0.97	0.15	
6	0.98	1.01	1.02	0.63	
7	0.94	1.03	1.03	0.65	
8	1.00	1.00	1.00	1.00	
9	1.17	0.92	1.03	1.03	
10	0.90	1.04	1.16	0.42	
11	1.06	0.97	1.04	0.79	
12	2.95	0.13	2.01	0.63	
13	1.09	0.96	0.97	0.86	
14	0.92	1.04	0.92	1.05	
15	0.97	1.01	0.97	0.67	
16	1.00	1.00	1.06	1.02	

	fnr_disparity	tpr_disparity	tnr_disparity	npv_disparity
0	1.09	0.94	1.05	1.00
1	2.10	0.21	1.46	0.99
2	2.07	0.23	1.45	0.99
3	1.05	0.96	1.05	1.00
4	1.16	0.89	1.05	0.92
5	2.10	0.20	1.46	1.00
6	1.41	0.70	1.20	1.00
7	1.43	0.69	1.19	1.00
8	1.00	1.00	1.00	1.00
9	0.87	1.10	0.99	0.99
10	1.81	0.41	1.32	0.97

11	1.18	0.87	1.12	0.99
12	0.30	1.51	1.20	0.82
13	1.04	0.97	1.08	1.01
14	0.97	1.02	0.97	1.01
15	1.34	0.76	1.18	1.01
16	1.02	0.98	0.99	0.99

Análisis por minoría. Ahora podemos analizar el sesgo a partir de la colonia con *menos* predicciones positivas. Veamos los resultados.

[175]:
```
min_bdf = bias.get_disparity_min_metric(xtab, original_df=aeq_df)
```

```
get_disparity_min_metric()
```

1.1.6 Tabla de análisis de sesgo por Bias (por minoría):

[176]:
```
min_bdf[['attribute_name', 'attribute_value'] +  bias.
↪list_disparities(min_bdf)].round(2)
```

[176]:

	attribute_name	attribute_value	ppr_disparity	pprev_disparity	\
0	delegacion	Alvaro Obregon	478.39	5.74	
1	delegacion	Azcapotzalco	51.74	1.03	
2	delegacion	Benito Juarez	99.22	1.14	
3	delegacion	Coyoacan	506.30	5.74	
4	delegacion	Cuajimalpa	127.65	5.85	
5	delegacion	Cuauhtemoc	115.57	1.00	
6	delegacion	Gustavo A. Madero	464.04	4.04	
7	delegacion	Iztacalco	198.30	4.08	
8	delegacion	Iztapalapa	1206.78	6.27	
9	delegacion	Magdalena Contreras	123.48	6.69	
10	delegacion	Miguel Hidalgo	223.70	2.57	
11	delegacion	Milpa Alta	33.09	5.09	
12	delegacion	No Disponible	1.00	10.89	
13	delegacion	Tlahuac	165.43	5.61	
14	delegacion	Tlalpan	538.26	6.49	
15	delegacion	Venustiano Carranza	345.83	4.29	
16	delegacion	Xochimilco	298.48	6.34	

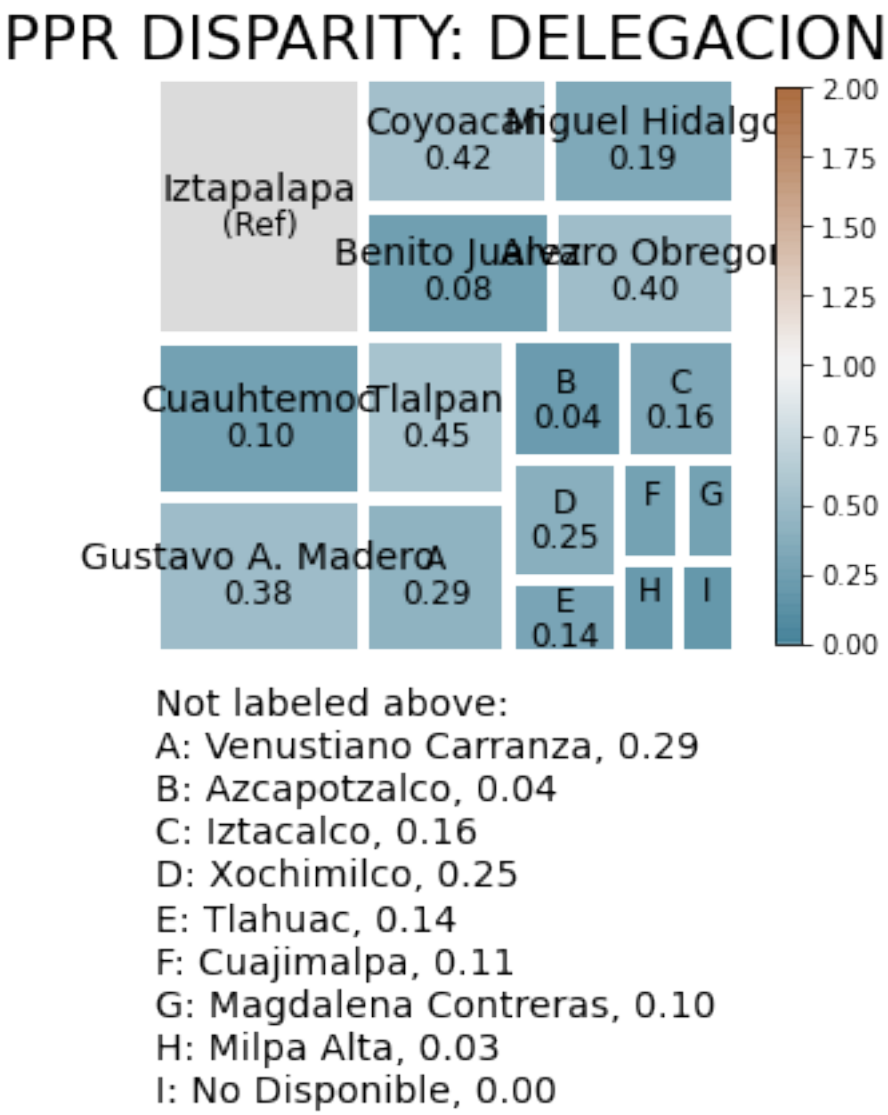
	precision_disparity	fdr_disparity	for_disparity	fpr_disparity	\
0	1.09	8.00	1.08	5.95	
1	1.10	7.96	1.13	1.01	
2	1.09	8.00	1.14	1.13	
3	1.13	7.86	1.06	5.87	
4	1.37	7.11	1.57	5.93	
5	1.02	8.21	1.06	1.00	
6	1.09	8.01	1.11	4.12	
7	1.04	8.16	1.12	4.23	

8	1.10	7.95	1.09	6.53
9	1.29	7.35	1.12	6.69
10	1.00	8.28	1.26	2.71
11	1.17	7.72	1.13	5.14
12	3.26	1.00	2.18	4.11
13	1.20	7.63	1.05	5.61
14	1.02	8.23	1.00	6.86
15	1.07	8.05	1.05	4.37
16	1.11	7.94	1.15	6.65

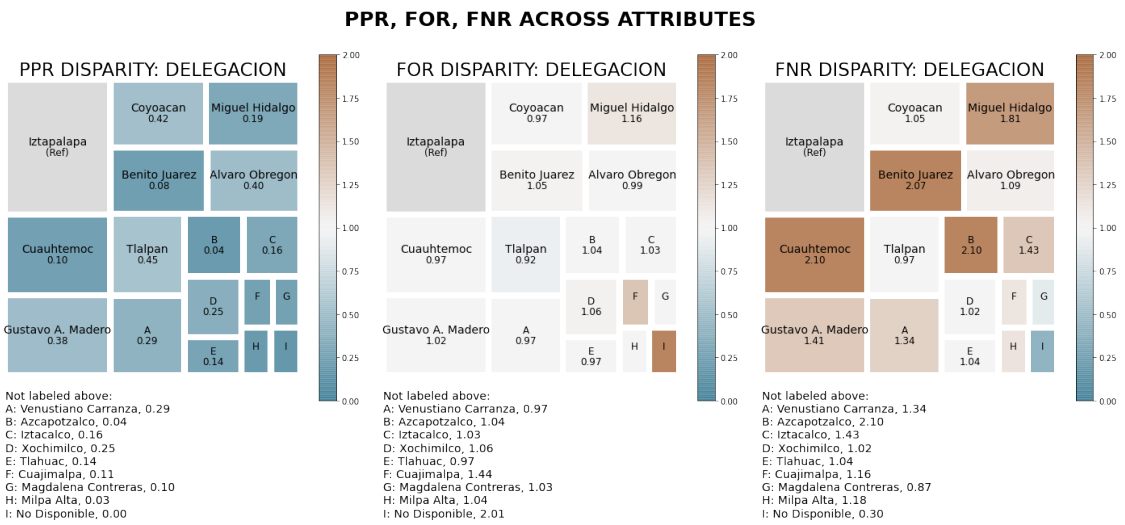
	fnr_disparity	tpr_disparity	tnr_disparity	npv_disparity
0	3.65	4.58	1.08	1.22
1	7.02	1.03	1.50	1.21
2	6.93	1.12	1.49	1.21
3	3.53	4.70	1.08	1.22
4	3.88	4.33	1.08	1.12
5	7.05	1.00	1.50	1.22
6	4.73	3.44	1.24	1.21
7	4.79	3.38	1.23	1.21
8	3.35	4.89	1.03	1.22
9	2.91	5.36	1.01	1.21
10	6.07	2.03	1.36	1.18
11	3.95	4.26	1.15	1.21
12	1.00	7.36	1.24	1.00
13	3.47	4.77	1.11	1.22
14	3.24	5.01	1.00	1.23
15	4.48	3.71	1.21	1.22
16	3.43	4.81	1.02	1.20

1.2 Visualización de sesgo por Bias:

[177]:
```
ppr_disparity = aeq.plot_disparity(bdf, group_metric='ppr_disparity',
                                   attribute_name='delegacion',
                                   significance_alpha=0.05)
```



[178]:
```
p = aeq.plot_disparity_all(majority_bdf,  metrics=['ppr','for','fnr'],␣
↪significance_alpha=0.05)
```



Usando a Iztapalapa como referencia (por tener el número más alto observaciones), observamos lo siguiente en las métricas de interés:

- **PPR**: Todas las delegaciones tienen un *PPR* más bajo (comportamiento esperado).
- **FOR**: Parecería que la delegación Iztapalapa es relativamente media, hay varias delegaciones por encima y por debajo respecto a esta métrica. La más lejana hacia arriba es No disponible seguida de Cuajimalpa, y la más lejana hacia abajo es Tlalpan, por un margen relativamente pequeño. Esto podría significar que mandamos más ambulancias en vano a Cuajimalpa.
- **FNR**: Se observa una mayor disparidad. Ahora "No disponible" es la más lejana hacia bajo, seguida por Magdalena Contreras por un margen relativamente pequeño respecto a Iztapalapa. Las más lejanas hacia arriba son Azcapotzalco y Cuauhtémoc.

1.3 Análisis de Justicia (*Fairness*):

[179]:
```
f = Fairness()
```

[180]:
```
fdf = f.get_group_value_fairness(bdf)
```

[181]:
```
parity_determinations = f.list_parities(fdf)
```

[182]:
```
fdf[['attribute_name', 'attribute_value'] + absolute_metrics + bias.
 ↪list_disparities(fdf) + parity_determinations].round(2)
```

[182]:

	attribute_name	attribute_value	tpr	tnr	for	fdr	fpr	fnr	\
0	delegacion	Alvaro Obregon	0.54	0.68	0.15	0.70	0.32	0.46	
1	delegacion	Azcapotzalco	0.12	0.95	0.16	0.69	0.05	0.88	
2	delegacion	Benito Juarez	0.13	0.94	0.16	0.70	0.06	0.87	
3	delegacion	Coyoacan	0.56	0.68	0.15	0.68	0.32	0.44	
4	delegacion	Cuajimalpa	0.51	0.68	0.22	0.62	0.32	0.49	
5	delegacion	Cuauhtemoc	0.12	0.95	0.15	0.71	0.05	0.88	

6	delegacion	Gustavo A. Madero	0.41	0.78	0.15	0.70	0.22	0.59
7	delegacion	Iztacalco	0.40	0.77	0.15	0.71	0.23	0.60
8	delegacion	Iztapalapa	0.58	0.65	0.15	0.69	0.35	0.42
9	delegacion	Magdalena Contreras	0.64	0.64	0.15	0.64	0.36	0.36
10	delegacion	Miguel Hidalgo	0.24	0.85	0.17	0.72	0.15	0.76
11	delegacion	Milpa Alta	0.51	0.72	0.15	0.67	0.28	0.49
12	delegacion	No Disponible	0.88	0.78	0.30	0.09	0.22	0.12
13	delegacion	Tlahuac	0.57	0.70	0.14	0.66	0.30	0.43
14	delegacion	Tlalpan	0.60	0.63	0.14	0.72	0.37	0.40
15	delegacion	Venustiano Carranza	0.44	0.76	0.14	0.70	0.24	0.56
16	delegacion	Xochimilco	0.57	0.64	0.16	0.69	0.36	0.43

	npv	precision	…	Equalized Odds	FNR Parity	Supervised Fairness	\
0	0.85	0.30	…	True	True	True	
1	0.84	0.31	…	False	False	False	
2	0.84	0.30	…	False	False	False	
3	0.85	0.32	…	True	True	True	
4	0.78	0.38	…	True	True	False	
5	0.85	0.29	…	False	False	False	
6	0.85	0.30	…	False	False	False	
7	0.85	0.29	…	False	False	False	
8	0.85	0.31	…	True	True	True	
9	0.85	0.36	…	True	True	True	
10	0.83	0.28	…	False	False	False	
11	0.85	0.33	…	False	True	False	
12	0.70	0.91	…	False	False	False	
13	0.86	0.34	…	True	True	True	
14	0.86	0.28	…	True	True	True	
15	0.86	0.30	…	False	False	False	
16	0.84	0.31	…	True	True	True	

	Statistical Parity	Precision Parity	TypeII Parity	FPR Parity	\
0	False	True	True	True	
1	False	True	False	False	
2	False	True	False	False	
3	False	True	True	True	
4	False	True	False	True	
5	False	True	False	False	
6	False	True	False	False	
7	False	True	False	False	
8	True	True	True	True	
9	False	True	True	True	
10	False	True	False	False	
11	False	True	True	False	
12	False	False	False	False	
13	False	True	True	True	
14	False	True	True	True	

```
15              False              True          False       False
16              False              True           True        True

    TNR Parity  FOR Parity  NPV Parity
0         True        True        True
1        False        True        True
2        False        True        True
3         True        True        True
4         True       False        True
5        False        True        True
6         True        True        True
7         True        True        True
8         True        True        True
9         True        True        True
10       False        True        True
11        True        True        True
12        True       False        True
13        True        True        True
14        True        True        True
15        True        True        True
16        True        True        True

[17 rows x 38 columns]
```

Podemos ver la equidad a nivel de atributos:

[183]:
```
gaf = f.get_group_attribute_fairness(fdf)
gaf
```

[183]:
```
   model_id score_threshold attribute_name  Statistical Parity  Impact Parity  \
0         0      binary 0/1     delegacion               False          False

   FDR Parity  FPR Parity  FOR Parity  FNR Parity  TPR Parity  TNR Parity  \
0       False       False       False       False       False       False

   NPV Parity  Precision Parity  TypeI Parity  TypeII Parity  Equalized Odds  \
0        True             False         False          False           False

   Unsupervised Fairness  Supervised Fairness
0                  False                False
```

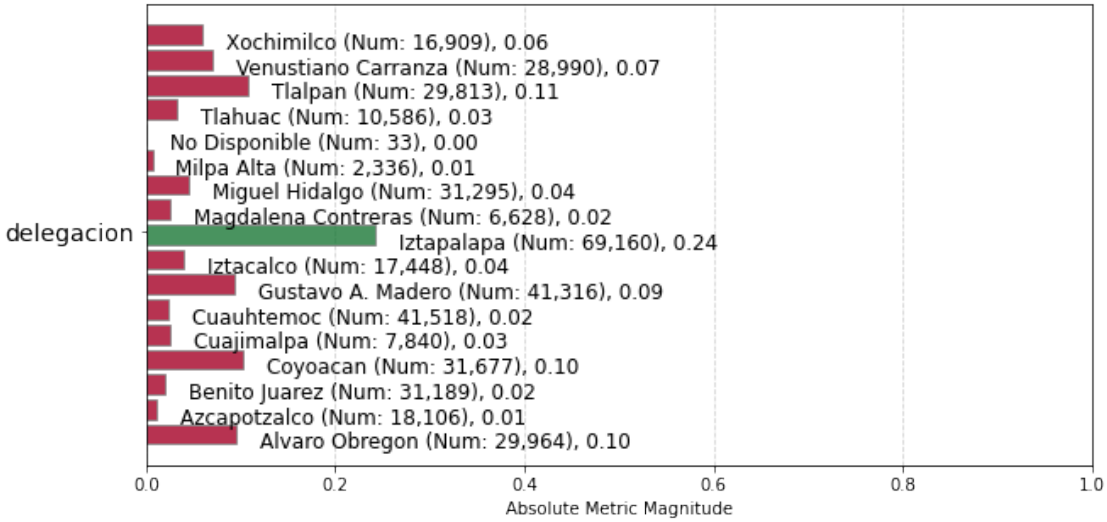
Podemos ver que nuestro modelo es injusto para prácticamente todas las métricas, con excepción de **NPV** (Negative Predictive Value).

[184]:
```
gof = f.get_overall_fairness(fdf)
gof
```

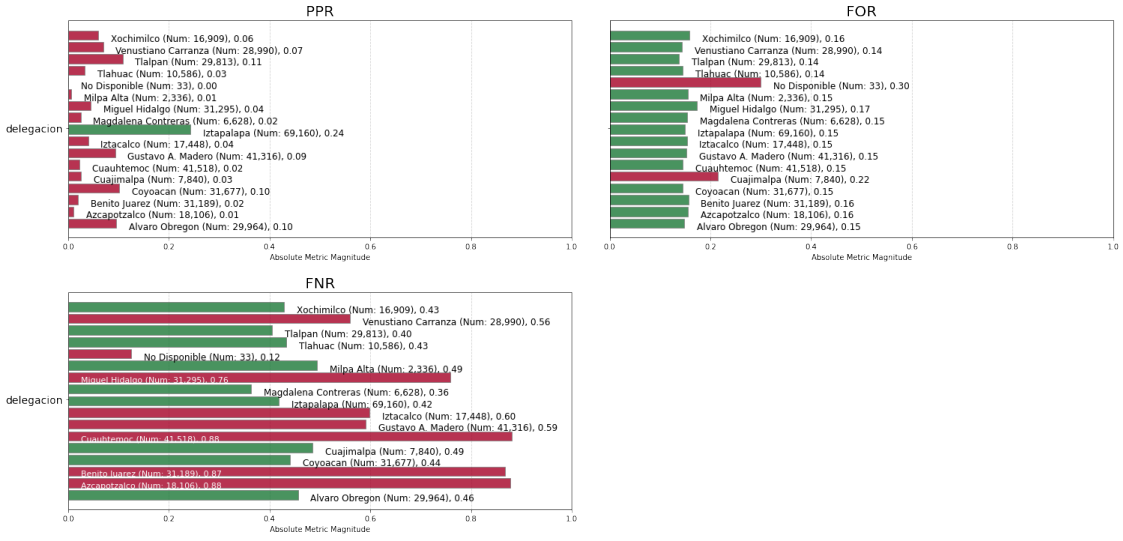
[184]:
```
{'Unsupervised Fairness': False,
 'Supervised Fairness': False,
 'Overall Fairness': False}
```

1.4 Visualización de Equidad (*Fairness*):

[185]:
```
z = aeq.plot_fairness_group(fdf, group_metric='ppr')
```



[186]:
```
z = aeq.plot_fairness_group_all(fdf, metrics=['ppr','for','fnr'], ncols = 2)
```




```
<Figure size 432x288 with 0 Axes>
```

Similar a lo que se había observado anteriormente, se puede concluir lo siguiente:

- **PPR**: Iztapalapa parecería ser la delegación con más propensión a las etiquetas positivas, esto puede ser peligroso pues se pueden dejar de enviar ambulancias en ocasiones que sí sería necesario. Se observa una gran disparidad con el resto de las delegaciones respecto a Iztapalapa. En la gráfica se observa de color verde por ser el grupo de referencia.
- **FOR**: Cuajimalpa parecería ser la delegación que resulta más beneficiada con la *FOR* más alta. Se podrían estar mandando ambulancias en vano. Basado en los colores, parece ser de las métricas con mayor justicia de nuestro modelo.
- **FNR**: Se observa una menor justicia de nuestro modelo, es un comportamiento esperado dada la baja precisión de nuestro modelo.

[ ]: