Redneuronal_Equipo23

October 30, 2022

#Maestría en Inteligencia Artificial Aplicada ##Curso: Inteligencia Artificial y Aprendizaje Automático ###Tecnológico de Monterrey ###Prof Luis Eduardo Falcón Morales

0.1 Actividad de la Semana 7

###Red Neuronal Artificial - Perceptrón Multicapa : Multilayer Perceptrón (MLP)

Nombres y matrículas de los integrantes del equipo:

- Mateo Comprés, Rafael José. A01793054
- García Hernández, Enrique Ricardo. A01315428
- Chavarría Barrientos, Daniel. A01331204
- García Sabag, Omar Nayib. A01793008

En cada sección deberás incluir todas las líneas de código necesarias para responder a cada uno de los ejercicios.

[ ]:
```
# Incluye aquí todos módulos, librerías y paquetes que requieras.
from sklearn.model_selection import RepeatedKFold, train_test_split,␣
 ↪cross_validate
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import make_scorer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.inspection import permutation_importance
from sklearn.preprocessing import power_transform
import seaborn as sns
from sklearn.compose import TransformedTargetRegressor
```

#**Ejercicio-1.**

Importamos los datos que estaremos analizando

[ ]:
```
#Actualizar con la ruta donde se está almacenando el archivo
df_original = pd.read_csv('dataset_Facebook.csv', sep = ';')
df = df_original.copy() #Hacemos una copia de los datos originales antes de␣
 ↪manipularlos
df.head()
```

[ ]:

	Page total likes	Type	Category	Post Month	Post Weekday	Post Hour	\
0	139441	Photo	2	12	4	3	
1	139441	Status	2	12	3	10	
2	139441	Photo	3	12	3	3	
3	139441	Photo	2	12	2	10	
4	139441	Photo	2	12	2	3	

	Paid	Lifetime Post Total Reach	Lifetime Post Total Impressions	\
0	0.0	2752	5091	
1	0.0	10460	19057	
2	0.0	2413	4373	
3	1.0	50128	87991	
4	0.0	7244	13594	

	Lifetime Engaged Users	Lifetime Post Consumers	\
0	178	109	
1	1457	1361	
2	177	113	
3	2211	790	
4	671	410	

	Lifetime Post Consumptions	\
0	159	
1	1674	
2	154	
3	1119	
4	580	

	Lifetime Post Impressions by people who have liked your Page	\
0	3078	
1	11710	
2	2812	
3	61027	
4	6228	

	Lifetime Post reach by people who like your Page	\
0	1640	
1	6112	

```
2                                                 1503
3                                                32048
4                                                 3200

   Lifetime People who have liked your Page and engaged with your post  \
0                                                  119
1                                                 1108
2                                                  132
3                                                 1386
4                                                  396

   comment    like  share  Total Interactions
0        4    79.0   17.0                 100
1        5   130.0   29.0                 164
2        0    66.0   14.0                  80
3       58  1572.0  147.0                1777
4       19   325.0   49.0                 393
```

Elegimos las columnas que estaremos utilizando para el ejercicio

[ ]:
```
df=df[['Category', 'Page total likes','Type','Post Month','Post Hour','Post␣
 ↪Weekday','Paid','Lifetime People who have liked your Page and engaged with␣
 ↪your post']]
df.head(5)
```

[ ]:
```
   Category  Page total likes     Type  Post Month  Post Hour  Post Weekday  \
0         2            139441   Photo           12          3             4
1         2            139441  Status           12         10             3
2         3            139441   Photo           12          3             3
3         2            139441   Photo           12         10             2
4         2            139441   Photo           12          3             2

   Paid  Lifetime People who have liked your Page and engaged with your post
0   0.0                                                  119
1   0.0                                                 1108
2   0.0                                                  132
3   1.0                                                 1386
4   0.0                                                  396
```

Procedemos a renombrar la variable de respuesta para facilitar la lectura

[ ]:
```
#Renombrar variables
df.rename(columns={'Lifetime People who have liked your Page and engaged with␣
 ↪your post':'LPE'},
                inplace=True)
df.head(5)
```

[ ]:

	Category	Page total likes	Type	Post Month	Post Hour	Post Weekday	\
0	2	139441	Photo	12	3	4	
1	2	139441	Status	12	10	3	
2	3	139441	Photo	12	3	3	
3	2	139441	Photo	12	10	2	
4	2	139441	Photo	12	3	2	

	Paid	LPE
0	0.0	119
1	0.0	1108
2	0.0	132
3	1.0	1386
4	0.0	396

[ ]:
```
df.shape
```

[ ]: (500, 8)

Definimos las variables de entrada y salida

[ ]:
```
input_vars = [] #Variables de entrada
X = df[df.columns[0:7]] #Dataframe con variables de entrada
X.head()
y = df[df.columns[-1:]] #Dataframe con variable de salida
y.head()
```

[ ]:

	LPE
0	119
1	1108
2	132
3	1386
4	396

[ ]:
```
X.head()
```

[ ]:

	Category	Page total likes	Type	Post Month	Post Hour	Post Weekday	\
0	2	139441	Photo	12	3	4	
1	2	139441	Status	12	10	3	
2	3	139441	Photo	12	3	3	
3	2	139441	Photo	12	10	2	
4	2	139441	Photo	12	3	2	

	Paid
0	0.0
1	0.0
2	0.0
3	1.0
4	0.0

#**Ejercicio-2.**

Ahora procedemos a realizar la partición entre los datos de prueba y entrenamiento

[ ]:
```
#Se pide 100 datos de prueba de un total de 500. Esto representa un 20%
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2,␣
↪random_state=42)
```

#**Ejercicio-3.**

Definimos las funciones para realizar los cálculos de los errores

[ ]:
```
def calculate_RMSE(y_real,y_predicted):
    #Raiz cuadrada del error cuadrático medio
    #Root Mean Square Error
    RMSE = np.sqrt(((y_predicted - y_real) ** 2).mean())
    return RMSE

def calculate_MAE(y_real,y_predicted):
    #Error absoluto medio
    #Mean Absolute Error
    MAE = np.mean(np.abs(y_real - y_predicted))
    return MAE

def calculate_MAPE (y_real,y_predicted):
    #Error porcentual absoluto medio (MAPE)
    #Mean Absolute Percentage Error
    MAPE = np.mean(np.abs((y_real - y_predicted) / y_real))
    return MAPE
```

También definimos una función para imprimir las distintas métricas que obtendremos del entrenamiento de los modelos.

[ ]:
```
#Función para imprimir las métricas

def print_metrics(y_predicted, y_real, model_name):

  print(f"-------Métricas {model_name}-------\n")
  print("RMSE: %.4f \n" %(calculate_RMSE(y_predicted, y_real)))
  print("MAE: %.4f \n" %(calculate_MAE(y_predicted, y_real)))
  print("MAPE: %.4f \n" %(calculate_MAPE(y_predicted, y_real)))


#Función para definir los scores que vienen del cross_validate
def print_scores(score, model_name):

  print(f"{model_name} train scores\n")
  print('mean RMSE: %.3f (%.4f) \nmean MAE: %.3f  (%.4f)\nmean MAPE: %.3f (%.
↪4f) \n' % (
```

```
                                                                    np.
↪mean(score['train_RMSE']),
                                                                    np.
↪std(score['train_RMSE']),
                                                                    np.
↪mean(score['train_MAE']),
                                                                    np.
↪std(score['train_MAE']),
                                                                    np.
↪mean(score['train_MAPE']),
                                                                    np.
↪std(score['train_MAPE']),
                                                                    ))

 print(f"\n{model_name} test scores\n")
 print('mean RMSE: %.3f (%.4f) \nmean MAE: %.3f  (%.4f)\nmean MAPE: %.3f (%.
↪4f) \n' % (
                                                                    np.
↪mean(score['test_RMSE']),
                                                                    np.
↪std(score['test_RMSE']),
                                                                    np.
↪mean(score['test_MAE']),
                                                                    np.
↪std(score['test_MAE']),
                                                                    np.
↪mean(score['test_MAPE']),
                                                                    np.
↪std(score['test_MAPE']),
                                                                    ))
```

#**Ejercicio-4.**

Realizamos un análisis exploratorio de los datos para evaluar su distribución y los métodos de limpieza a utilizar.

[ ]:
```
#Análisis de los datos (boxplot, histograma, resumen estadístico, etc)
X_train.describe(include="all")
#Los resultados de Category: 1: Action, 2:Product, 3:Inspiration
#Paid: (yes, no)
```

[ ]:

	Category	Page total likes	Type	Post Month	Post Hour	\
count	400.000000	400.000000	400	400.000000	400.00000	
unique	NaN	NaN	4	NaN	NaN	
top	NaN	NaN	Photo	NaN	NaN	
freq	NaN	NaN	346	NaN	NaN	
mean	1.880000	123692.070000	NaN	7.075000	7.99500	

```
std       0.849945      15813.590103    NaN    3.213283    4.33092
min       1.000000      81370.000000    NaN    1.000000    1.00000
25%       1.000000     115368.000000    NaN    4.000000    3.00000
50%       2.000000     130791.000000    NaN    7.000000    9.00000
75%       3.000000     136393.000000    NaN   10.000000   11.00000
max       3.000000     139441.000000    NaN   12.000000   23.00000
```

```
        Post Weekday        Paid
count     400.000000  399.000000
unique           NaN         NaN
top              NaN         NaN
freq             NaN         NaN
mean        4.132500    0.298246
std         2.038463    0.458062
min         1.000000    0.000000
25%         2.000000    0.000000
50%         4.000000    0.000000
75%         6.000000    1.000000
max         7.000000    1.000000
```

De lo anterior se conluye con lo siguiente:

- Ninguna variable presenta información missing / nula
- No se presentan valores fuera de los umbrales posibles, los valores para los meses son de 1 (enero) a 12 (diciembre)
- No se presentan valores fuera de los umbrales posibles, los valores para las horas son de 1 a 23
- Los días de la semana están OK los valores posibles son de 1 (lunes) a 7 (domingo)

Ahora revisemos las proporciones de las variables categóricas

[ ]:
```
X_train['Category'].value_counts()/X_train['Category'].shape
```

[ ]:
```
1    0.4275
3    0.3075
2    0.2650
Name: Category, dtype: float64
```

[ ]:
```
X_train['Type'].value_counts()/X_train['Type'].shape
```

[ ]:
```
Photo     0.8650
Status    0.0825
Link      0.0400
Video     0.0125
Name: Type, dtype: float64
```

De lo anterior se observa que el tipo “Video” representa apenas un 1%. Agrupemos “Video” con “Link” para que sea al menos un 5% de la población.

[ ]:
```
#Agrupamos Link+Video para que sea al menos el 5% de la población
X_train['Type'] = X_train['Type'].map({'Photo':'Photo','Status':'Status','Link':
 ↪'Link/Video','Video':'Link/Video'})
X_train['Type'].value_counts()/X_train['Type'].shape
```

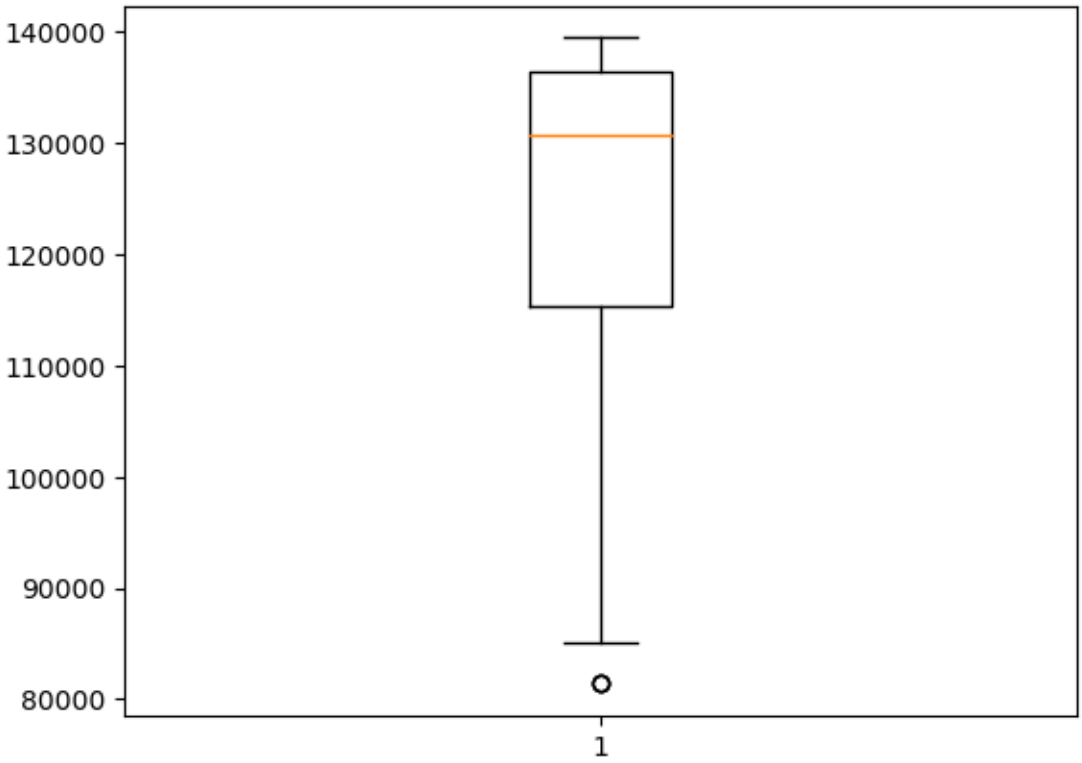
[ ]:
```
Photo         0.8650
Status        0.0825
Link/Video    0.0525
Name: Type, dtype: float64
```

[ ]:
```
X_train['Post Month'].value_counts()/X_train['Post Month'].shape
#Todos los meses tienen al menos 5% de la información
```

[ ]:
```
10    0.1350
7     0.1225
12    0.0975
6     0.0950
4     0.0950
3     0.0775
9     0.0750
8     0.0725
5     0.0700
11    0.0700
2     0.0500
1     0.0400
Name: Post Month, dtype: float64
```

Procedamos a realizar un diagrama de caja de la variable numérica

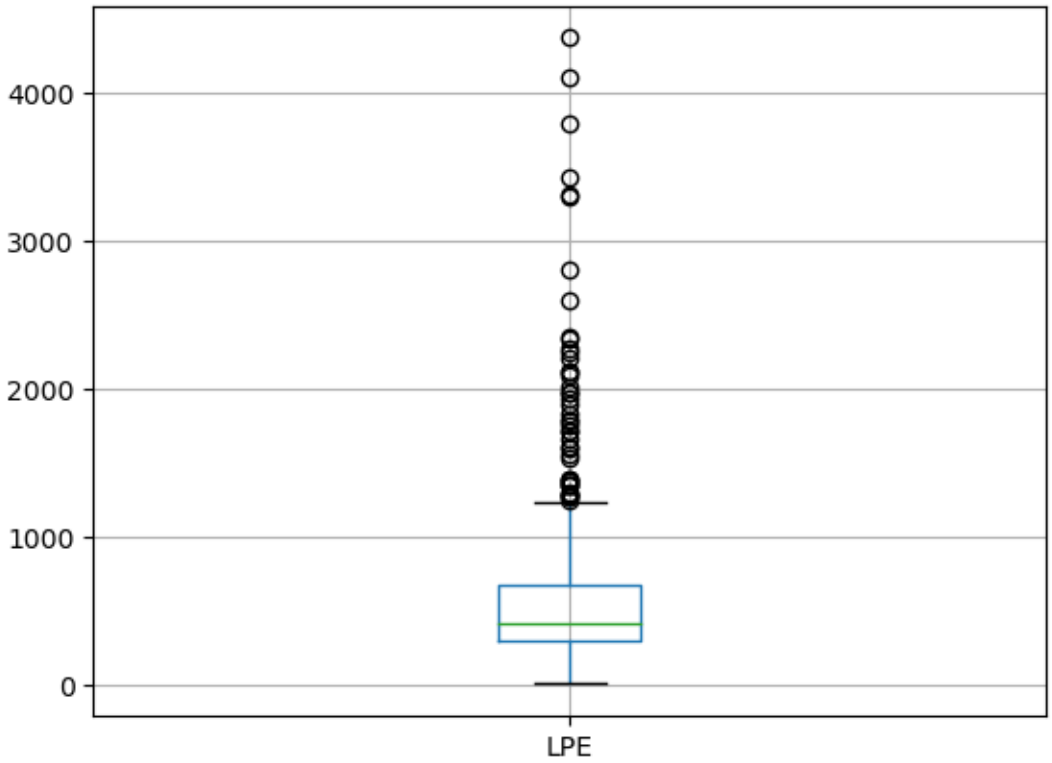
[ ]:
```
plt.boxplot(X_train['Page total likes'])
plt.show()
```



De lo anterior se observa que los datos tienen un sesgo a la izquierda, pero no se observan valores atípicos. Ahora revisemos la variable de salida.

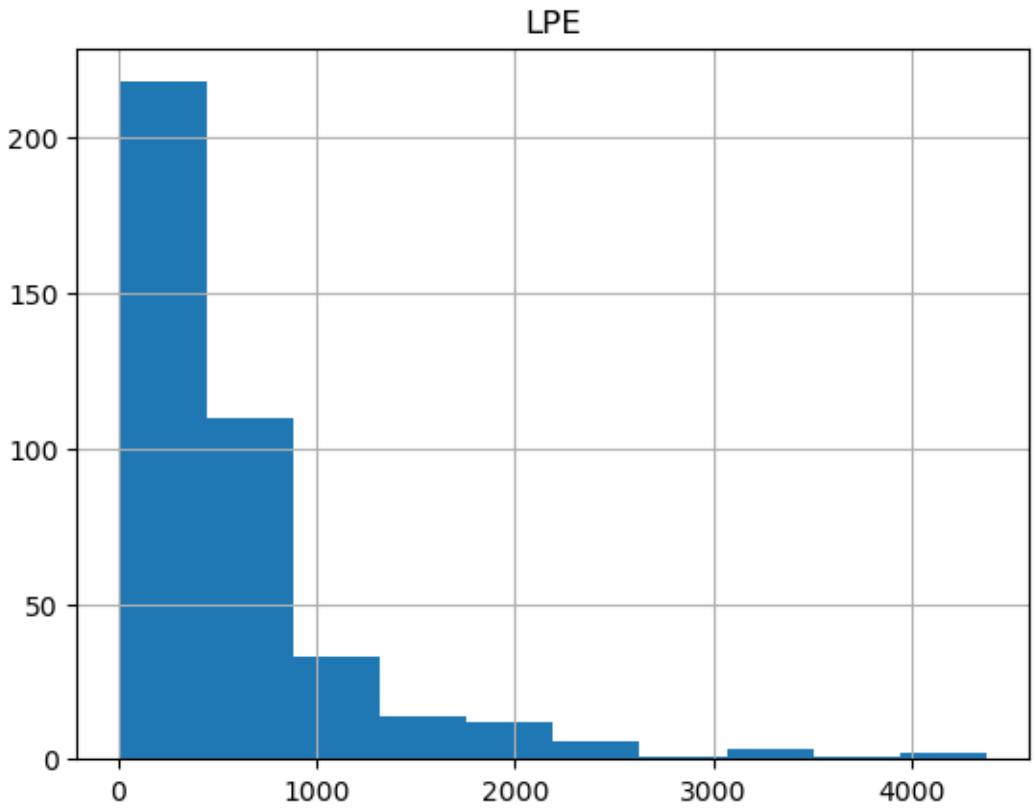
[ ]:
```
y_train.boxplot()
```

[ ]:
```
<AxesSubplot: >
```



[ ]:
```
y_train.hist()
```

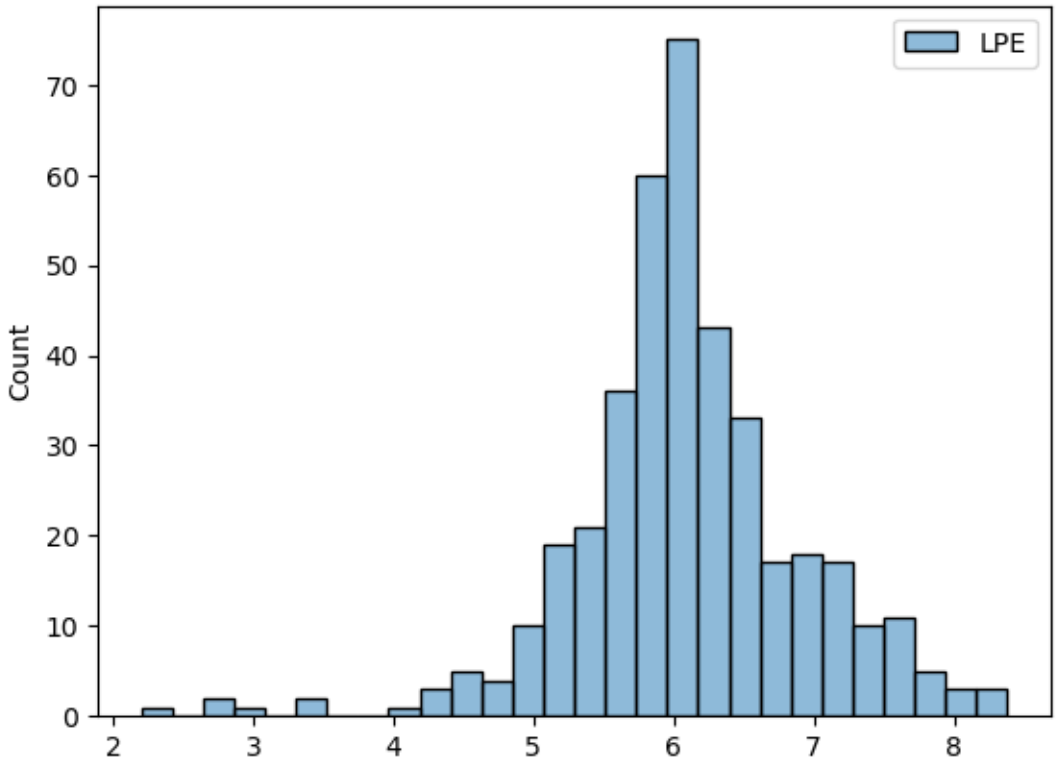
[ ]:
```
array([[<AxesSubplot: title={'center': 'LPE'}>]], dtype=object)
```



De los gráficos anteriores se observa un sesgo a la derecha. También se puede observar del diagrama de caja valores atípicos que podrían afectar el desempeño de nuestro modelo. Esto significa que debemos realizar una transformación a la variable de salida para fines de entrenamiento. Probemos realizando una transformación logarítimica.

[ ]:
```
sns.histplot(np.log(y_train))
```

[ ]:
```
<AxesSubplot: ylabel='Count'>
```



La transformación aplicada resulta adecuada, ya que se observa una distribución uniforme de tipo campana.

Ahora procedamos a definir los piplines

[ ]:
```
#Preprocessor con todos los pipes y transformaciones
# Se estandarizan las columnas numéricas y se hace one-hot-encoding de las␣
 ↪columnas cualitativas, ordinales
from sklearn.preprocessing import MinMaxScaler


numeric_cols = ["Page total likes"]
cat_cols = ["Category","Type"]
bin_cols = ["Paid"]
ord_cols = ["Post Month","Post Hour","Post Weekday"]

#Para la variable numérica, proponemos la mediana 50% de la población arriba y␣
 ↪por debajo del umbral
#Realizamos una transformación z-score
numeric_transformer = Pipeline(
                        steps=[
                            ('imputer', SimpleImputer(strategy='median')),
                            ('scaler', StandardScaler())
```

```
                        ]
                    )

# Transformaciones para las variables categóricas, ordinales y binaria
#Para las variables categóricas, ordinales y binarias proponemos el equivalente␣
 ↪a la moda para distribuciones, es decir, estaremos imputando usando el valor␣
 ↪que más se repite
categorical_transformer = Pipeline(
                            steps=[
                                ('imputer',␣
 ↪SimpleImputer(strategy='most_frequent')),
                                ('onehot',␣
 ↪OneHotEncoder(handle_unknown='ignore'))
                            ]
                        )

preprocessor = ColumnTransformer(
                    transformers=[
                        ('numeric', numeric_transformer, numeric_cols),
                        ('cat', categorical_transformer, cat_cols + bin_cols),
                        ('ord', SimpleImputer(strategy="most_frequent"),␣
 ↪ord_cols)
                                                    ],
                    remainder='passthrough'
                )
```

#Ejercicio-5.

Realizamos el entrenamiento de un modelo dummy, el cual nos arrojará los resultados mínimos a lograr para cada modelo que estaremos probando.

[ ]:
```
#Particionamos el conjunto de entrenamiento y validación en 25% para validación␣
 ↪(100) y el restante para entrenamiento.
X_train_dummy, X_val_dummy, y_train_dummy, y_val_dummy =␣
 ↪train_test_split(X_train, y_train, test_size = 0.25)
```

[ ]:
```
#Entrenamos el Dummy regressor

dummy_pipe = Pipeline (steps = [
      ("transformer", preprocessor),
      ("model", DummyRegressor())
      ])


final_model=TransformedTargetRegressor(regressor=dummy_pipe,
                                       func= np.log,
                                       inverse_func= np.exp)
```

```
final_model.fit(X_train_dummy, y_train_dummy.values.ravel())

train_predictions = final_model.predict(X_train_dummy)
test_predictions = final_model.predict(X_val_dummy)
```

[ ]:
```
#Imprimimos las métricas

print_metrics(train_predictions, y_train_dummy.values.ravel(), "Train Dummy␣
 ↪Regressor")
print_metrics(test_predictions, y_val_dummy.values.ravel(), "Test Dummy␣
 ↪Regressor")
```

```
-------Métricas Train Dummy Regressor-------

RMSE: 599.1055

MAE: 334.1710

MAPE: 0.7554

-------Métricas Test Dummy Regressor-------

RMSE: 732.6770

MAE: 362.1818

MAPE: 0.8188
```

#Ejercicio-6.

Procedemos a entrenar nuestros modelos

[ ]:
```
kfold = RepeatedKFold (n_splits= 5, n_repeats= 3, random_state=42)
```

[ ]:
```
model_names = ["Multiple Linear Regression", "Random Forest", "Multilayer␣
 ↪Perceptron"]
models = [
   LinearRegression(),
    RandomForestRegressor(),
    MLPRegressor(max_iter = 50000)]
```

[ ]:
```
scores = [] # Donde irán los resultados para el boxplot
scorings = {
      'MAPE': 'neg_mean_absolute_percentage_error', #␣
 ↪make_scorer(calculate_MAPE, greater_is_better = False),
      'MAE': make_scorer(calculate_MAE,greater_is_better = False),
      'RMSE': make_scorer(calculate_RMSE,greater_is_better = False)}
```

```
for i in range(len(models)):

    pipe = Pipeline (steps = [
      ("transformer", preprocessor),
      ("model", models[i])
      ])

    final_model=TransformedTargetRegressor(regressor=pipe,
                                    func= np.log,
                                    inverse_func= np.exp)

    score = cross_validate(
      estimator = final_model,
      X = X_train,
      y = y_train.values.ravel(),
      cv=kfold,
      scoring = scorings,
     return_train_score= True,

      n_jobs = -1)



    scores.append(score["test_MAPE"])
    print_scores(score, model_names[i])
```

```
Multiple Linear Regression train scores

mean RMSE: -530.895 (16.9794)
mean MAE: -281.564  (9.7480)
mean MAPE: -0.807 (0.0875)


Multiple Linear Regression test scores

mean RMSE: -544.076 (78.6654)
mean MAE: -291.927  (37.1803)
mean MAPE: -0.873 (0.4138)

Random Forest train scores

mean RMSE: -275.306 (12.9951)
mean MAE: -129.914  (6.3944)
mean MAPE: -0.214 (0.0135)
```

```
Random Forest test scores

mean RMSE: -540.259 (78.2076)
mean MAE: -305.231  (41.3385)
mean MAPE: -0.895 (0.3898)

Multilayer Perceptron train scores

mean RMSE: -445.321 (24.8578)
mean MAE: -238.005  (15.0949)
mean MAPE: -0.558 (0.0677)


Multilayer Perceptron test scores

mean RMSE: -567.728 (91.6200)
mean MAE: -313.944  (50.5724)
mean MAPE: -0.943 (0.3996)
```

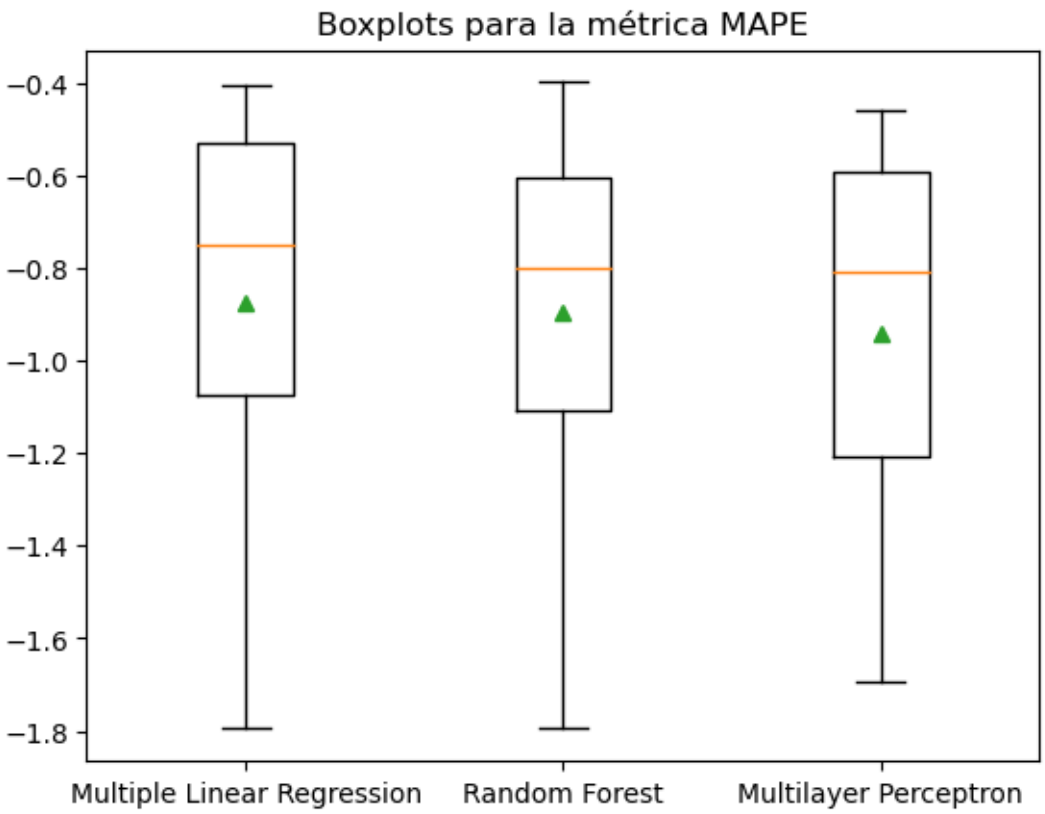
De los resultados obtenidos arriba podemos observar que prácticamente todos los modelos están sobreentrenados. Esto se evidencia porque los valores del MAPE obtenidos del entrenamiento son mucho mas bajos que los que se obtiene del conjunto de validación.

#**Ejercicio-7.**

Realizamos el diagrama de caja de los resultados obtenidos de cada modelo

[ ]:
```
print("\nBoxplot chart\n")
plt.title("Boxplots para la métrica MAPE")
plt.boxplot(scores,labels=model_names,showmeans=True)

plt.show()
```

```
Boxplot chart
```



Del gráfico anterior podemos observar una alta variablidad en la métrica del MAPE para cada uno de los modelos. Por ejemplo, el MAPE varía entre 0.4 y 1.2 aproximdamente, siendo los modelos LR y MLP los que cuentan una ligera menor variación. Con relación a la media, se observa que los tres modelos tienen unos valores similares.

Una posible causa de esta alta variabilidad del MAPE en los modelos podría deberse a que se cuentan con relativamente pocos datos para entrenar (el dataset tiene apenas 500 entradas), como se comentó arriba. Por tanto, es de esperarse que tengamos valores muy diferentes cada vez que corramos los modelos.

Por otro lado, si comparamos estos resultados con lo obtenidos por los autores podemos evidenciar que estos se encuentran muy lejos del 0.27 que se muestra en el artículo. Esto es un indicativo de que estos modelos están subentrenados, pues el mejor MAPE logrado durante el entrenamiento fue de cerca de un 0.40, aún muy lejos del 0.27 que obtuvieron los autores.

#Ejercicio-8.

Ahora procedamos a buscar los valores óptimos para el modelo MLP

[ ]:
```
import warnings
warnings.filterwarnings("ignore")

ColumnPreprocessor = preprocessor
```

```python
model = MLPRegressor(max_iter=100000)

fullPipeline = Pipeline(steps = [('preprocessor',␣
 ↪ColumnPreprocessor),('regressor', model)])

#Transformamos las y
final_model=TransformedTargetRegressor(regressor=fullPipeline,
                                      func= np.log,
                                      inverse_func= np.exp
                                       )

#Definimos las neuronas en cada capa
layers = [i for i in range(1,16,2)]

parameter_space = {
    'regressor__regressor__hidden_layer_sizes': [(i,i) for i in layers],
    'regressor__regressor__alpha': [0.0001,0.001, 0.01, 0.05,1,5,10],
    'regressor__regressor__learning_rate_init': [0.01,0.1,0.5,1,5,10],
}

search = GridSearchCV(final_model,
                      parameter_space,
                      n_jobs=-1,
                      cv=kfold,
                      scoring=make_scorer(calculate_MAPE, greater_is_better =␣
 ↪False)
                      )

search.fit(X_train, y_train.values.ravel())

print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

```
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
```

```
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/preprocessing/_function_transformer.py:292: RuntimeWarning:
overflow encountered in exp
  return func(X, **(kw_args if kw_args else {}))
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/model_selection/_search.py:953: UserWarning: One or more of the
test scores are non-finite: [-8.82831130e-001 -9.82840722e-001 -1.07861467e+000
-1.09846037e+000
 -2.63222852e+008 -1.63711420e+004 -9.15457053e-001 -9.14061407e-001
 -1.28352486e+000 -1.09416232e+000 -1.00076033e+000 -9.52283560e-001
 -9.09488133e-001 -8.74348001e-001 -1.21119261e+000 -1.16383131e+000
 -4.22274438e+002 -2.86738104e+001 -9.13133246e-001 -9.71715084e-001
 -1.05175491e+000 -1.23829887e+000 -1.04397010e+000 -1.00000000e+000
 -9.83909959e-001 -8.82609945e-001 -1.14184781e+000 -1.11828603e+000
 -9.47442255e-001 -9.84452928e-001 -8.73042318e-001 -8.93637072e-001
 -1.06420131e+000 -9.09337024e-001 -1.57696611e+060 -4.20375861e+005
 -9.62529403e-001 -9.55272979e-001 -1.08849865e+000 -1.03496212e+000
 -1.79569006e+023 -2.92230113e+212 -9.16497247e-001 -9.16996875e-001
 -1.02519149e+000 -1.00402677e+000 -9.66867230e-001 -9.88089068e-001
 -8.92632732e-001 -9.92543895e-001 -1.15187068e+000 -1.33069988e+000
 -5.92972028e+005 -9.19902673e-001 -8.70427159e-001 -9.26315219e-001
 -1.18502930e+000 -1.54304913e+000 -3.88078463e+006 -1.84056654e+022
 -9.08939166e-001 -9.34105030e-001 -1.27497969e+000 -1.34233336e+000
 -4.28013594e+003 -5.74737812e+002 -8.74614781e-001 -9.11857503e-001
 -1.15844781e+000 -1.02813757e+000 -1.33161075e+009 -9.97886435e-001
 -9.23188018e-001 -8.63183564e-001 -1.03994970e+000 -1.06918841e+000
 -9.95655413e-001 -1.50216099e+045 -9.69197503e-001 -9.36778120e-001
 -1.18745813e+000 -9.80527898e-001 -1.18170410e+006 -4.11176192e+220
 -9.61500923e-001 -9.23343367e-001 -1.09862645e+000 -9.35564359e-001
```

-1.06070910e+000 -1.30383952e+006 -9.42889583e-001 -9.91236407e-001
-9.41965730e-001 -9.74748280e-001 -9.59381704e-001 -7.72239801e+080
-9.17430632e-001 -9.97913138e-001 -1.07404035e+000 -1.07003409e+000
-9.70427355e-001 -1.24095533e+021 -9.19590068e-001 -9.39832917e-001
-1.28531254e+000 -1.22036847e+000 -1.79021029e+009 -1.10125129e+023
-9.13901314e-001 -8.87752843e-001 -1.22373886e+000 -1.26063810e+000
-1.66268996e+005 -8.73353992e+013 -9.29020876e-001 -8.95272937e-001
-1.18862145e+000 -1.02524269e+000 -1.65068174e+002 -2.27924245e+003
-8.86793992e-001 -8.97027402e-001 -1.13904608e+000 -9.78837592e-001
-1.15683297e+002 -1.22618697e+031 -9.29860364e-001 -8.80428651e-001
-1.26987896e+000 -9.62886948e-001 -2.76700486e+006 -1.27260179e+010
-9.67001767e-001 -8.85095672e-001 -1.07649066e+000 -1.09387000e+000
-4.36402994e+045 -3.72545764e+012 -8.55785681e-001 -9.69726284e-001
-1.00522194e+000 -9.37551340e-001 -9.42179687e-001 -5.22285444e+001
-8.97931367e-001 -9.85312000e-001 -1.03645792e+000 -1.17025313e+000
-3.59578762e+004 -4.26651884e+019 -8.86826594e-001 -9.30606780e-001
-1.17074990e+000 -1.22981226e+000 -1.08746811e+000 -8.73568751e+021
-8.97726458e-001 -8.97405785e-001 -1.23372323e+000 -1.00913828e+000
-2.11534614e+007 -2.64052924e+023 -8.67945378e-001 -9.08876369e-001
-1.26495281e+000 -1.01157586e+000 -9.32712352e+008   -inf
-8.84810189e-001 -9.48848965e-001 -9.70062547e-001 -1.10264483e+000
-9.99992611e-001   -inf -9.55792668e-001 -9.01830375e-001
-1.23825623e+000 -9.16353721e-001 -1.18003700e+001 -4.01613205e+128
-9.24144686e-001 -9.24745796e-001 -9.90847158e-001 -9.97066476e-001
-2.51559414e+001 -1.61702647e+008 -9.66093009e-001 -9.75670404e-001
-9.27439374e-001 -9.84010640e-001   -inf   -inf
-9.16700811e-001 -9.75350497e-001 -1.02871637e+000 -1.03799383e+000
-1.44696868e+009 -1.33433495e+020 -8.79714945e-001 -9.53493178e-001
-1.15211808e+000 -9.83828306e-001 -1.14144130e+004 -2.23180631e+002
-8.78764457e-001 -9.43878718e-001 -1.14786081e+000 -9.59672603e-001
            -inf -3.69116471e+113 -9.59948536e-001 -8.30548487e-001
-8.89844975e-001 -1.01353349e+000 -1.77809812e+006 -3.68026174e+018
-9.32298715e-001 -8.78174231e-001 -9.55236699e-001 -1.31430838e+000
-6.08834025e+011   -inf -8.87583780e-001 -9.59557534e-001
-1.04793023e+000 -8.98738782e-001 -2.16884151e+019 -6.30425393e+004
-9.51941700e-001 -1.00400882e+000 -9.50217949e-001 -1.08854714e+000
-1.63021693e+030 -8.44397450e+017 -9.31435003e-001 -1.04104248e+000
-1.10300889e+000 -1.86962298e+013 -1.32374460e+202 -3.15333257e+019
-9.20760134e-001 -9.41658094e-001 -9.61278808e-001 -1.03505747e+000
-4.04520658e+006 -5.39530152e+021 -9.12784700e-001 -9.11681480e-001
-1.17838751e+000 -9.45661195e-001 -5.93754733e+007 -1.14142977e+018
-9.10259422e-001 -9.58386689e-001 -9.34046214e-001 -1.74540712e+000
-5.39026208e+005 -1.16071849e+010 -8.74473435e-001 -8.77602688e-001
-1.50829549e+000 -1.93992551e+000 -7.91693657e+039 -8.87001337e+017
-8.90156831e-001 -9.09850300e-001 -1.76733872e+000 -1.61951600e+000
-2.39849511e+012   -inf -8.93180079e-001 -9.74010776e-001
-2.46260240e+000 -1.54338621e+000 -1.34324421e+000 -3.54910933e+008
-8.45563042e-001 -8.88329092e-001 -1.29068335e+000 -2.26648488e+001

```
             -inf -1.03473893e+020 -9.45878338e-001 -8.50821624e-001
 -2.73976355e+001 -1.43195334e+000 -2.61825167e+036 -8.76420631e+038
 -9.19098282e-001 -9.51391465e-001 -9.79372132e-001 -9.33457498e-001
 -2.61463900e+000 -6.79735836e+020 -9.06543782e-001 -8.89137413e-001
 -9.51641071e-001 -1.19540100e+000 -1.51824098e+008 -4.18308166e+020
 -8.82905989e-001 -9.71648626e-001 -1.10714271e+000 -1.36356702e+000
 -2.50724116e+000 -1.52267093e+104 -8.81964155e-001 -9.09377896e-001
 -1.28998355e+000 -3.36011219e+003 -1.01842802e+004 -7.28682695e+031
 -9.03705594e-001 -8.86404526e-001 -9.69162072e-001 -9.90986232e-001
 -3.97399014e+005 -1.15910763e+217 -8.51799222e-001 -8.99667037e-001
 -1.17597899e+000 -1.11804717e+000 -1.91857852e+023 -7.85199927e+065
 -8.97633299e-001 -9.43541024e-001 -1.21415731e+000 -2.59083619e+001
 -1.79654240e+014 -5.70723600e+006 -8.69629229e-001 -1.04057944e+000
 -3.12457436e+000 -1.03660663e+007 -3.60254984e+009 -7.52361943e+029]
  warnings.warn(
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/model_selection/_search.py:962: RuntimeWarning: invalid value
encountered in subtract
  (array - array_means[:, np.newaxis]) ** 2, axis=1, weights=weights
/Users/rmateoc/opt/anaconda3/envs/python310/lib/python3.10/site-
packages/sklearn/model_selection/_search.py:962: RuntimeWarning: overflow
encountered in square
  (array - array_means[:, np.newaxis]) ** 2, axis=1, weights=weights

Best parameter (CV score=-0.831):
{'regressor__regressor__alpha': 1, 'regressor__regressor__hidden_layer_sizes':
(7, 7), 'regressor__regressor__learning_rate_init': 0.1}
```

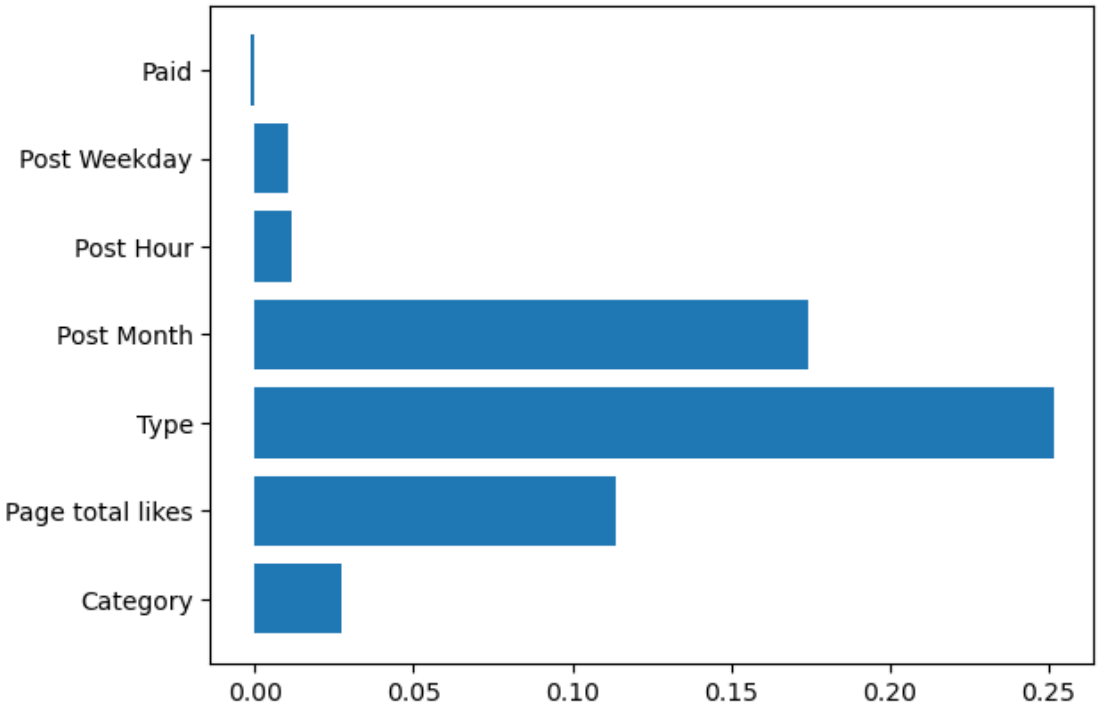
[ ]:
```
search.best_estimator_.fit(X_train, y_train.values.ravel())
predict = search.best_estimator_.predict(X_test)

calculate_MAPE(y_test.values.ravel(), predict)
```

[ ]:
```
0.6301308125122296
```

#**Ejercicio-9.**

[ ]:
```
importance = permutation_importance(search, X_train, y_train.values.ravel(),␣
 ↪n_repeats=10)

# visualicemos la importancia de cada métrica, de las cuales sabemos
# cuáles son las más importantes, de acuerdo a como se definieron al inicio:

for i,v in enumerate(importance['importances_mean']):
        print('Feature: %s, Score: %.5f' % (X.columns[i],v))

plt.barh(X.columns.to_numpy(), importance['importances_mean'])
plt.show()
```

```
Feature: Category, Score: 0.02709
Feature: Page total likes, Score: 0.11392
Feature: Type, Score: 0.25174
Feature: Post Month, Score: 0.17411
Feature: Post Hour, Score: 0.01177
Feature: Post Weekday, Score: 0.01025
Feature: Paid, Score: -0.00141
```



Conclusiones

Del gráfico anterior se puede apreciar que la variable "Post Month" es la que representa mayor importancia respectoa las demás. En segundo lugar se encuentra la variable "Type" con un 30% aproximadamente.

Esto significa que para optimizar el LPE contribuye mucho el es en el que se hace la publicación, así como el tipo y el total de likes de la página. Las demás variables presentan poca importancia respecto a las 3 ya mencionadas.

Sin embargo, es importante destacar que el modelo final resultó con un MAPE de alrededor de 0.95 con el conjunto de prueba. Esto significa que el % de error es muy alto como para poder concluir que estos resultados sean confiables.

Procedamos a evaluar los resultados de los otros modelos.

#Ejercicio-10.

[ ]:
```
ColumnPreprocessor = preprocessor

mi_regressor = RandomForestRegressor()

fullPipeline = Pipeline(steps = [('preprocessor',␣
 ↪ColumnPreprocessor),('regressor', mi_regressor)])

final_model=TransformedTargetRegressor(regressor=fullPipeline,
                                       func= np.log,
                                       inverse_func= np.exp
                                        )

parameter_space = {
    'regressor__regressor__criterion':['squared_error', 'absolute_error',␣
 ↪'poisson'],
    'regressor__regressor__max_depth':[1,3,5,10],
    'regressor__regressor__min_samples_split':[0.1,0.5,1.0],
    'regressor__regressor__max_features':['sqrt', 'log2', None]}

search = GridSearchCV(final_model,
                      parameter_space,
                      n_jobs=-1,
                      cv=kfold,
                     ␣
 ↪scoring=make_scorer(calculate_MAPE,greater_is_better=False)
                      )

search.fit(X_train, y_train.values.ravel())

print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

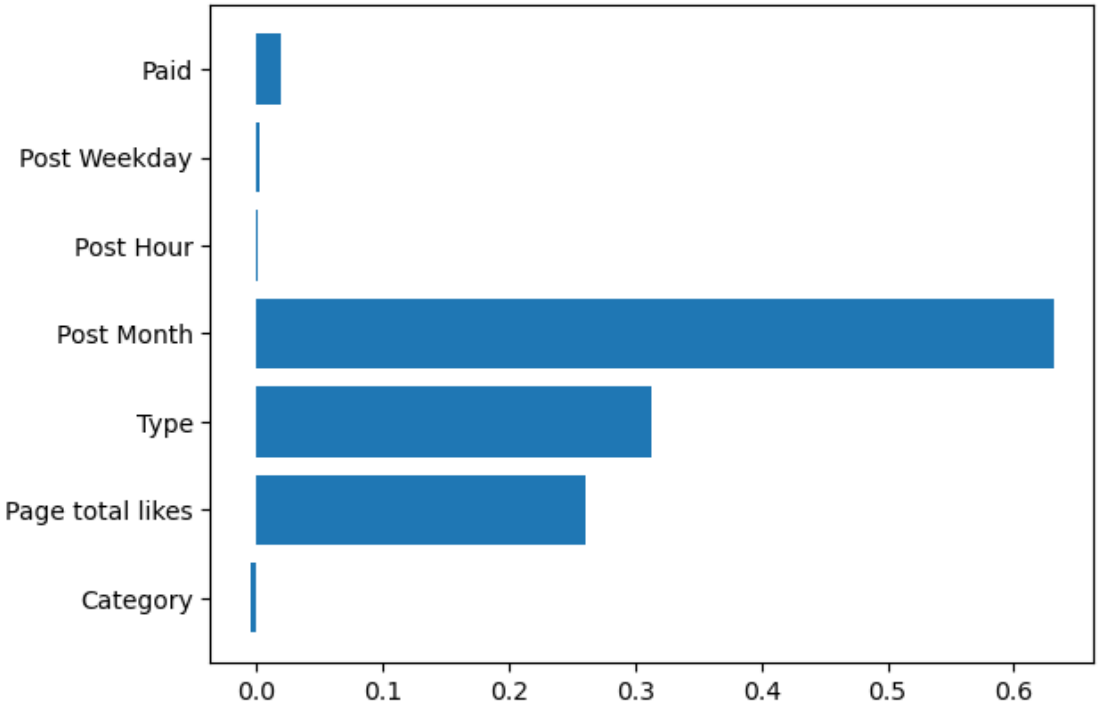
```
Best parameter (CV score=-0.711):
{'regressor__regressor__criterion': 'absolute_error',
'regressor__regressor__max_depth': 5, 'regressor__regressor__max_features':
None, 'regressor__regressor__min_samples_split': 0.1}
```

[ ]:
```
search.best_estimator_.fit(X_train, y_train.values.ravel())
predict = search.best_estimator_.predict(X_test)

calculate_MAPE(y_test.values.ravel(), predict)
```

[ ]:
```
importance = permutation_importance(search, X_train, y_train.values.ravel(),␣
 ↪n_repeats=10)

for i,v in enumerate(importance['importances_mean']):
        print('Feature: %s, Score: %.5f' % (X.columns[i],v))
```

```
plt.barh(X.columns.to_numpy(), importance['importances_mean'])
plt.show()
```

```
Feature: Category, Score: -0.00496
Feature: Page total likes, Score: 0.25997
Feature: Type, Score: 0.31315
Feature: Post Month, Score: 0.63148
Feature: Post Hour, Score: 0.00063
Feature: Post Weekday, Score: 0.00214
Feature: Paid, Score: 0.01980
```



Conclusiones

Con este modelo se obtuvieron resultados similares al MLP, donde se observa que las 3 variables de mayor importancia son Post Month, Type y Page total likes.

#Ejercicio-11.

[ ]:
```
ColumnPreprocessor = preprocessor


mi_regressor = LinearRegression()
```

```
fullPipeline = Pipeline(steps = [('preprocessor',␣
 ↪ColumnPreprocessor),('regressor', mi_regressor)])

final_model=TransformedTargetRegressor(regressor=fullPipeline,
                                       func= np.log,
                                       inverse_func= np.exp
                                        )


parameter_space = {
    'regressor__regressor__fit_intercept': [True, False],
    'regressor__regressor__copy_X': [True, False],
}

search = GridSearchCV(final_model,
                      parameter_space,
                      n_jobs=-1,
                      cv=kfold,
                     ␣
 ↪scoring=make_scorer(calculate_MAPE,greater_is_better=False)
                      )

search.fit(X_train, y_train.values.ravel())

print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

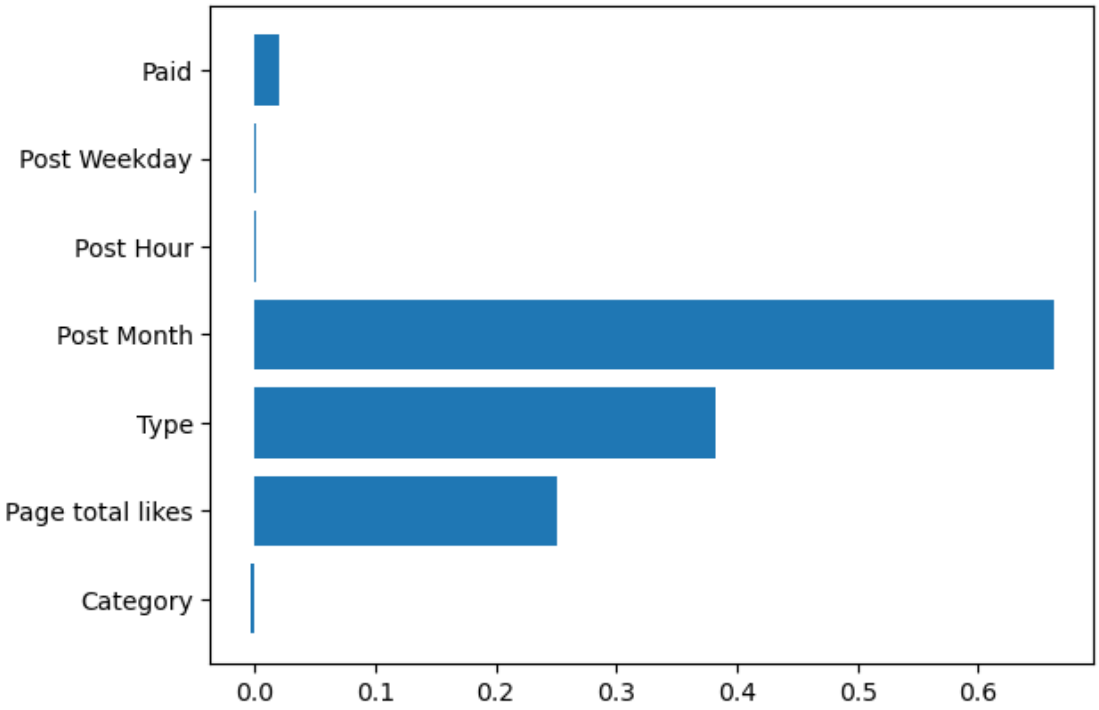
```
Best parameter (CV score=-0.771):
{'regressor__regressor__copy_X': True, 'regressor__regressor__fit_intercept':
True}
```

[ ]:
```
search.best_estimator_.fit(X_train, y_train.values.ravel())
predict = search.best_estimator_.predict(X_test)

calculate_MAPE(y_test.values.ravel(), predict)
```

[ ]:
```
importance = permutation_importance(search, X_train, y_train.values.ravel(),␣
 ↪n_repeats=10)

for i,v in enumerate(importance['importances_mean']):
        print('Feature: %s, Score: %.5f' % (X.columns[i],v))

plt.barh(X.columns.to_numpy(), importance['importances_mean'])
plt.show()
```

```
Feature: Category, Score: -0.00352
Feature: Page total likes, Score: 0.25125
Feature: Type, Score: 0.38191
```

```
Feature: Post Month, Score: 0.66261
Feature: Post Hour, Score: 0.00140
Feature: Post Weekday, Score: 0.00061
Feature: Paid, Score: 0.02050
```



Conclusiones

En este último modelo también se logran los mismos resultados que en los dos anteriores. De nuevo se puede observar que las 3 variables de mayor importancia son Post Month, Type y Page total likes.

#Ejercicio-12.

Al momento de correr estos modelos se pudo observar que los tres llegaron a conclusiones muy similares, donde las 3 variables que representan mayor importancia para el LPE son:

- Post Month
- Type
- Page total Likes

Lo anterior coincide con los hallazgos de los autores, aunque con magnitudes de importancia un tanto diferentes.

Sin embargo, en comparación con lo obtenido por los autores para el MAPE, se observa que estos modelos están subentrenados puesto que ninguno estuvo cerca del mejor MAPE logrado por los autores, el cual fue de un 0.27 aproximadamente. Esto hace que las predicciones de nuestros modelos no sean lo suficientemente confiables, por tener MAPES con los datos de prueba muy altos.

Una de las posibles razones de esta variación se deba a que los autores utilizaron otro modelo (SVM), además que su conjunto de datos contaba con alrededor de 800 registros, mientras el utilizado para este ejercicio fue de solo 500. Esto se debe a que varias de las entradas fueron eliminadas por confidencialidad, según se describía en la fuente de donde se obtuvieron los datos.

###**Fin de la Actividad de la semana 7.**