Maestría en Inteligencia Artificial Aplicada

Curso: Inteligencia Artificial y Aprendizaje Automático

Tecnológico de Monterrey

Prof Luis Eduardo Falcón Morales

Adtividad de la Semana 7

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

Nombres y matrículas de los integrantes del equipo:

- Rodrigo López Aguilera - A01793071
- Diego Carrera Nicholls - A00464290
- Guillermo Alfonso Muñiz Hermosillo - A01793101

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

In [ ]:
```
import numpy as np
import pandas as pd


from sklearn.compose import ColumnTransformer, TransformedTargetRegressor
from sklearn.dummy import DummyRegressor
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.inspection import permutation_importance
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import make_scorer
from sklearn.model_selection import train_test_split, cross_val_score, cross_validate, RepeatedKFold, GridSea
from sklearn.neural_network import MLPRegressor, MLPClassifier
from sklearn.pipeline import Pipeline, make_pipeline
```

```
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder, OrdinalEncoder, StandardScaler, LabelEncoder,


import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

In [ ]:
```
SEED=1
```

Ejercicio-1.

En esta tarea considera únicamente la siguiente variable de salida que se concluye que es una de las mejores en el artículo antes citado: 'Lifetime People who have liked a Page and engaged with a post'. Renombra dicha variable como "LPE" . Como variables de entrada selecciona las 7 variables que indican los autores en la Tabla 3 del artículo citado.

In [ ]:
```
df = pd.read_csv('./Facebook_metrics/dataset_Facebook.csv', sep=';', engine='python')
new_columns = {'Lifetime People who have liked your Page and engaged with your post':'LPE'}
df.rename(columns=new_columns, inplace=True)
df.head()
```

Out[ ]:

	Page total likes	Type	Category	Post Month	Post Weekday	Post Hour	Paid	Lifetime Post Total Reach	Lifetime Post Total Impressions	Lifetime Engaged Users	Lifetime Post Consumers	Lifetime Post Consumptions	Lifetim Pos Impression by peopl who hav liked you Pag
0	139441	Photo	2	12	4	3	0.0	2752	5091	178	109	159	3078
1	139441	Status	2	12	3	10	0.0	10460	19057	1457	1361	1674	11710
2	139441	Photo	3	12	3	3	0.0	2413	4373	177	113	154	281
3	139441	Photo	2	12	2	10	1.0	50128	87991	2211	790	1119	6102
4	139441	Photo	2	12	2	3	0.0	7244	13594	671	410	580	6228

In [ ]:
```
df = df[['Category', 'Page total likes', 'Type', 'Post Month', 'Post Hour', 'Post Weekday', 'Paid', 'LPE']].c
df.head()
```

Out[ ]:

	Category	Page total likes	Type	Post Month	Post Hour	Post Weekday	Paid	LPE
0	2	139441	Photo	12	3	4	0.0	119
1	2	139441	Status	12	10	3	0.0	1108
2	3	139441	Photo	12	3	3	0.0	132
3	2	139441	Photo	12	10	2	1.0	1386
4	2	139441	Photo	12	3	2	0.0	396

In [ ]:
```
df['Type'].unique()
```

Out[ ]:
```
array(['Photo', 'Status', 'Link', 'Video'], dtype=object)
```

In [ ]:
```
# labelenconder = LabelEncoder()
# type_encoded = labelenconder.fit_transform(df['Type'])
# df['Type'] = type_encoded
```

In [ ]:
```
df.describe(include='all')
```

Out[ ]:

	Category	Page total likes	Type	Post Month	Post Hour	Post Weekday	Paid	LPE
count	500.000000	500.000000	500	500.000000	500.000000	500.000000	499.000000	500.000000
unique	NaN	NaN	4	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	Photo	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	426	NaN	NaN	NaN	NaN	NaN
mean	1.880000	123194.176000	NaN	7.038000	7.840000	4.150000	0.278557	609.986000
std	0.852675	16272.813214	NaN	3.307936	4.368589	2.030701	0.448739	612.725618
min	1.000000	81370.000000	NaN	1.000000	1.000000	1.000000	0.000000	9.000000
25%	1.000000	112676.000000	NaN	4.000000	3.000000	2.000000	0.000000	291.000000
50%	2.000000	129600.000000	NaN	7.000000	9.000000	4.000000	0.000000	412.000000
75%	3.000000	136393.000000	NaN	10.000000	11.000000	6.000000	1.000000	656.250000
max	3.000000	139441.000000	NaN	12.000000	23.000000	7.000000	1.000000	4376.000000

Ejercicio-2.

Realiza una partición de los datos con 100 datos de Prueba y el resto para entrenamiento y validación.

In [ ]:
```
y = df['LPE']
X = df.loc[:, df.columns != 'LPE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=100, random_state=SEED)
```

In [ ]:
```
print( "X Entrenamiento y validación",  X_train.shape)
print( "X Pruebas", X_test.shape)

print( "Y Entrenamiento y validación", y_train.shape)
print( "Y Pruebas", y_test.shape)
```

```
X Entrenamiento y validación (400, 7)
X Pruebas (100, 7)
Y Entrenamiento y validación (400,)
Y Pruebas (100,)
```

Ejercicio-3.

Definirás tus propias funciones de errores para este problema de regresión. Los errores que utilizarás son la raíz cuadrada del error cuadrático medio RMSE, el error absoluto medio MAE y el error porcentual absoluto medio MAPE.

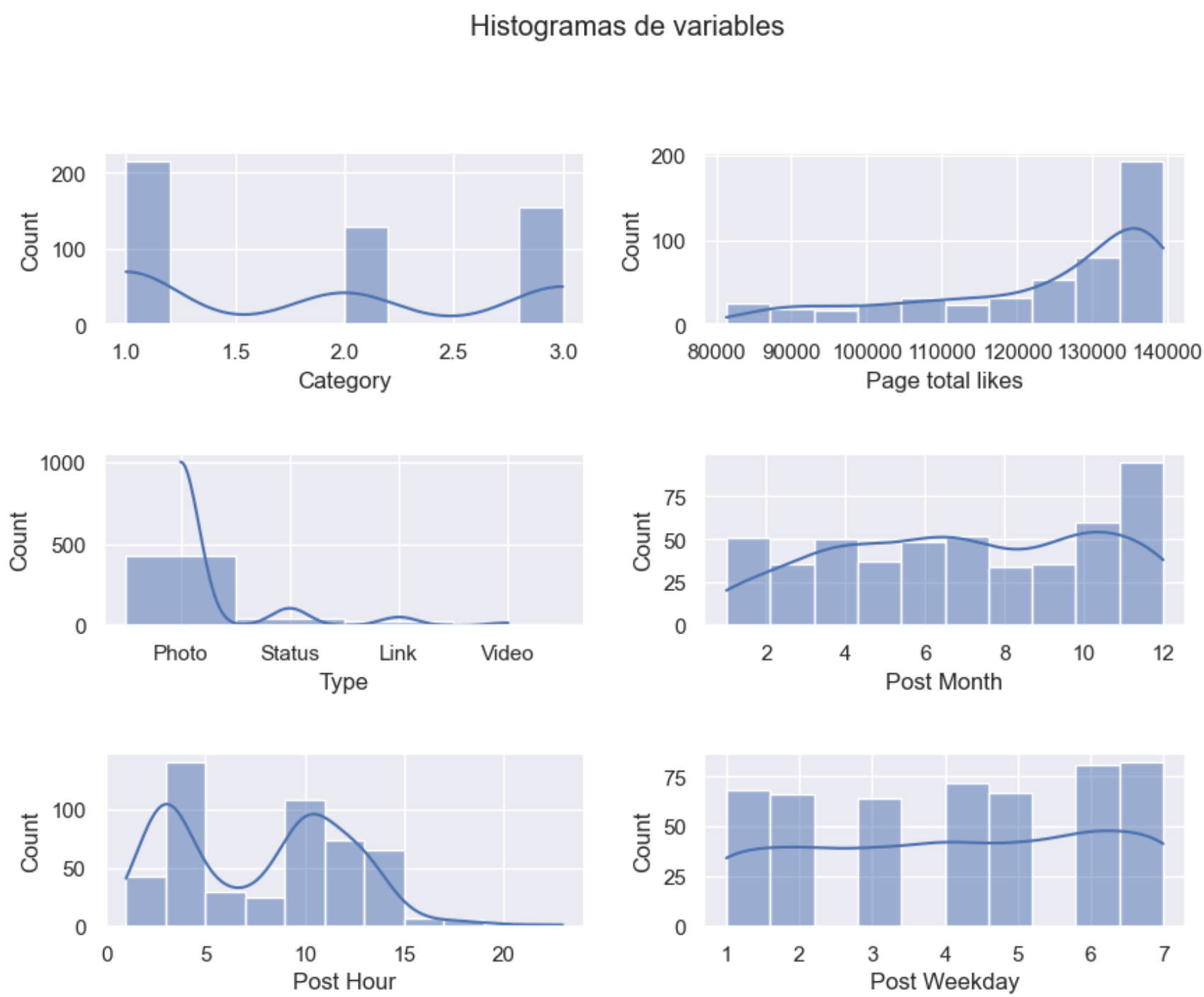
In [ ]:
```
def RMSE(y, yhat) -> float:
    return np.sqrt(np.mean(np.square(y - yhat)))

def MAE(y, yhat) -> float:
    return np.mean(np.absolute(yhat - y))

def MAPE(y, yhat) -> float:
    return np.mean(np.absolute((y - yhat) / y)) * 100
```

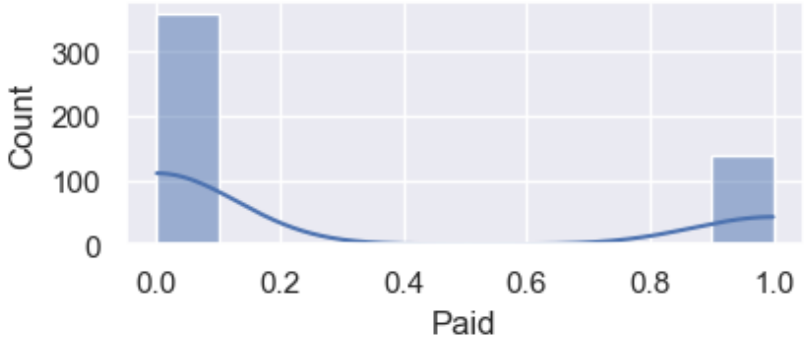
Ejercicio-4.

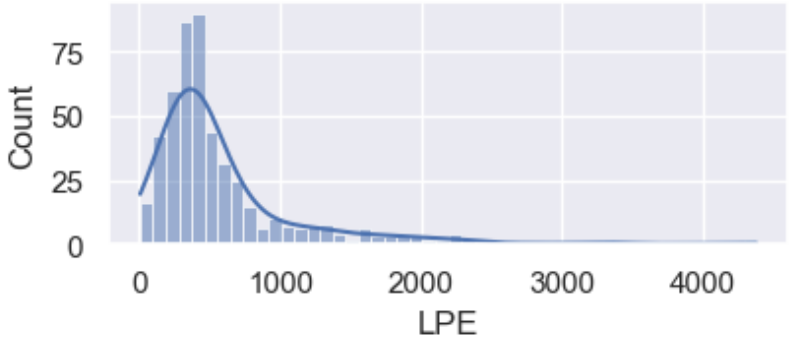
En la página de la UCI, así como en el artículo de los autores previamente citado encuentras información en relación al significado de cada variable. Haz una análisis de tus datos y lleva a cabo

las transformaciones que consideres adecuadas tanto en los datos de entrada, como en las de salida. Utiliza un Pipeline para evitar el filtrado de información.

In [ ]:
```
to_analyze = ['Category', 'Page total likes', 'Type', 'Post Month', 'Post Hour', 'Post Weekday', 'Paid', 'LPE
fig, ax  = plt.subplots(4, 2, figsize=(10, 10))
plt.suptitle('Histogramas de variables')
plt.subplots_adjust(hspace=0.75, wspace=0.25)
for i, col in enumerate(to_analyze):
    sns.histplot(data=df, x=col, ax=ax[i//2, i%2], kde=True).set(xlabel=col)

plt.show()
```

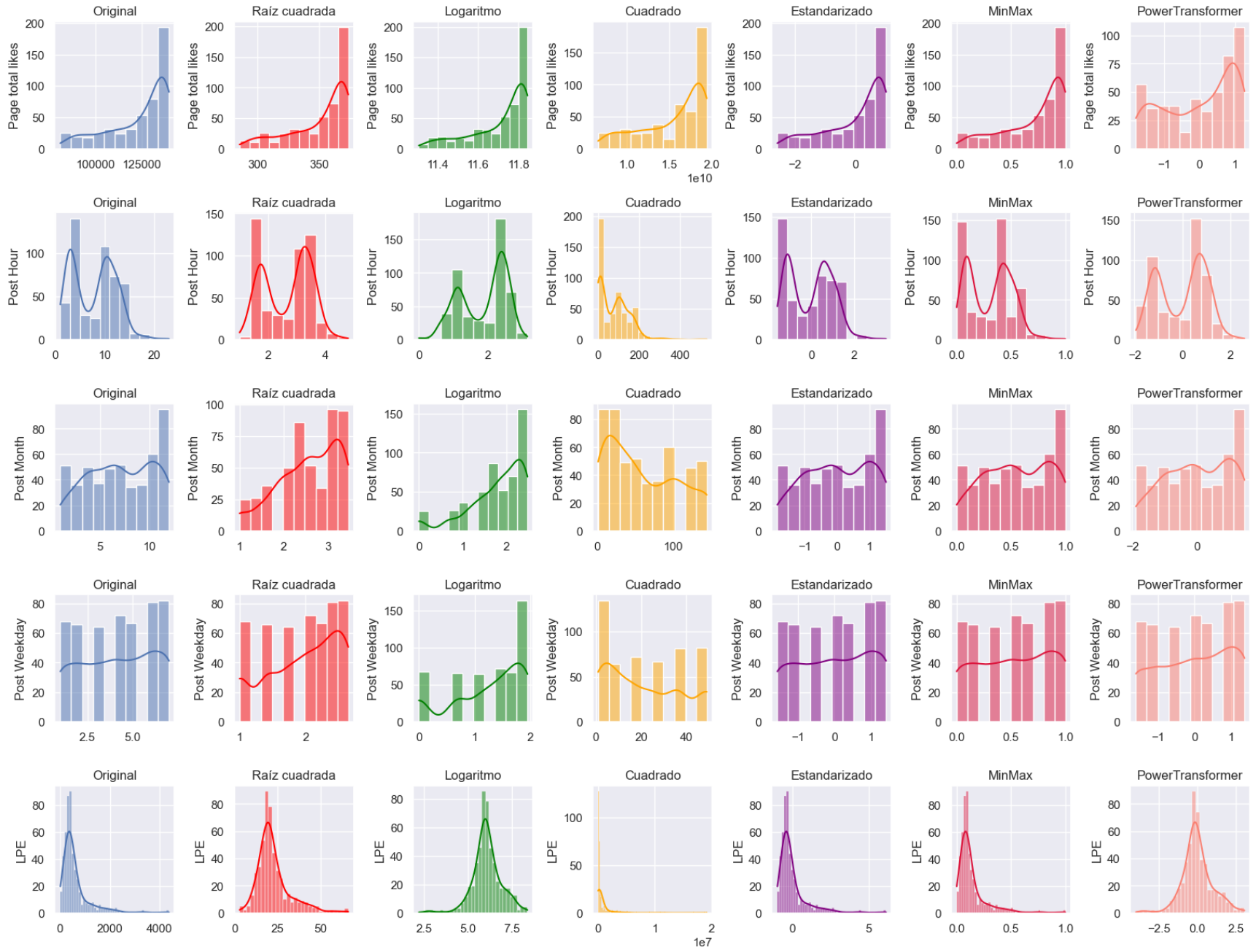

Histogramas de variables
Count
Category
Page total likes
Type
Photo
Status
Link
Video
Post Month
Post Hour
Post Weekday





In [ ]:

```
scaler = StandardScaler()
minmax = MinMaxScaler()
power_transformer = PowerTransformer()
# posibles transformaciones
to_transform_numerical = [
    'Page total likes',
    'Post Hour',
    'Post Month',
    'Post Weekday',
    'LPE'
]

fig, axes = plt.subplots( len(to_transform_numerical), 7, figsize=(20, 15))

plt.subplots_adjust(hspace=0.5, wspace=0.5)
for i, col in enumerate(to_transform_numerical):
    # plt.subplot(len(to_transform_numerical), 4, i*4+1)

    standard_scaled = scaler.fit_transform(df[col].values.reshape(-1, 1)).flatten()
    min_max_scaled  = minmax.fit_transform(df[col].values.reshape(-1, 1)).flatten()
    power_scaled    = power_transformer.fit_transform(df[col].values.reshape(-1, 1)).flatten()

    sns.histplot(data=df, x=col,            ax=axes[i, 0], kde=True).set(title='Original', xlabel='', ylabel=
    sns.histplot(data=np.sqrt(df[col]),     ax=axes[i, 1], kde=True, color="red").set(title='Raíz cuadrada',
    sns.histplot(data=np.log(df[col]),      ax=axes[i, 2], kde=True, color="green").set(title='Logaritmo', xl
    sns.histplot(data=np.power(df[col], 2), ax=axes[i, 3], kde=True, color="orange").set(title='Cuadrado', xl
    sns.histplot(data=standard_scaled,      ax=axes[i, 4], kde=True, color="purple").set(title='Estandarizado
    sns.histplot(data=min_max_scaled,       ax=axes[i, 5], kde=True, color="crimson").set(title='MinMax', xla
    sns.histplot(data=power_scaled,         ax=axes[i, 6], kde=True, color="salmon").set(title='PowerTransfor
```



In [ ]:
```
"""
Category -> categorical
Page total likes -> numerical
```

```
Type -> categorical
Post Month -> discret
Post Hour -> discrete
Post Weekday -> discrete
Paid -> categorical (float)
LPE -> discrete
"""

categorical = ['Category', 'Post Month', 'Post Weekday', 'Post Hour', 'Paid', 'Type']

numerical = ['Page total likes']

X.describe()
```

Out[ ]:

	Category	Page total likes	Post Month	Post Hour	Post Weekday	Paid
count	500.000000	500.000000	500.000000	500.000000	500.000000	499.000000
mean	1.880000	123194.176000	7.038000	7.840000	4.150000	0.278557
std	0.852675	16272.813214	3.307936	4.368589	2.030701	0.448739
min	1.000000	81370.000000	1.000000	1.000000	1.000000	0.000000
25%	1.000000	112676.000000	4.000000	3.000000	2.000000	0.000000
50%	2.000000	129600.000000	7.000000	9.000000	4.000000	0.000000
75%	3.000000	136393.000000	10.000000	11.000000	6.000000	1.000000
max	3.000000	139441.000000	12.000000	23.000000	7.000000	1.000000

In [ ]:
```
X.dtypes
```

Out[ ]:
```
Category              int64
Page total likes      int64
Type                 object
Post Month            int64
Post Hour             int64
Post Weekday          int64
Paid                float64
dtype: object
```

In [ ]:
```
# numeric_pipeline = Pipeline(steps=[
#   ('median', SimpleImputer(strategy='median')),
#   ('scaling', StandardScaler())
```

```
# ])

# categorical_pipeline = Pipeline(steps=[
#   ('imputacionModa', SimpleImputer(strategy='most_frequent')),
# ])


# oneHot_pipeline = Pipeline(steps=[
#   ('oneHot', OneHotEncoder(drop='first'))
# ])

# pipeline = ColumnTransformer(
#   transformers=[
#     ('numeric', numeric_pipeline, numerical),
#     ('categoric', categorical_pipeline, categorical),
#     ('oneHotEncoder', oneHot_pipeline, categorical)
#   ],
#   remainder='passthrough'
# )

# categoricas_numericas = ['Category']


categoricas_numericas = ['Paid']
numericas = ['Page total likes', 'Post Month', 'Post Hour', 'Post Weekday']
onehot = ['Type', 'Category']


pipeline_numericas = Pipeline(steps=[
  ('median_imputer', SimpleImputer(strategy='median')),
  ('power_transformer', PowerTransformer(method='box-cox'))
]);

pipeline_categoricas_numericas = Pipeline(steps=[
  ('mode_imputer', SimpleImputer(strategy='most_frequent')),
]);

pipeline_onehot = Pipeline(steps=[
  ('onehot', OneHotEncoder(drop='first', handle_unknown='ignore'))
]);

column_transformer = ColumnTransformer(
    transformers=[
```

```
        ('categoric', pipeline_categoricas_numericas, categoricas_numericas),
        ('onehot',    pipeline_onehot,                onehot),
        ('numeric',   pipeline_numericas,             numericas),
    ],
    remainder='passthrough'
)


column_transformer_notLR = ColumnTransformer(
  transformers=[
    ('type', pipeline_onehot, onehot)
  ]
)

# ordinal = ['Post Month', 'Post Hour', 'Post Weekday']
# nominal = ['Type', 'Paid', 'Category']
# numerical = ['Page total likes']

# ordinal_pipeline = Pipeline([
#     ("imputer", SimpleImputer(strategy="most_frequent")),
#     ("encoder", OrdinalEncoder())
# ])

# nominal_pipeline = Pipeline([
#     ("imputer", SimpleImputer(strategy="most_frequent")),
#     ("encoder", OneHotEncoder(sparse=True, handle_unknown="ignore"))
# ])

# numerical_pipeline = Pipeline([
#     ("imputer", SimpleImputer(strategy="mean")),
#     ("scaler", StandardScaler())
# ])

# pipeline = ColumnTransformer([
#     ("nominal_preprocessor", nominal_pipeline, nominal),
#     ("ordinal_preprocessor", ordinal_pipeline, ordinal),
#     ("numerical_preprocessor", numerical_pipeline, numerical)
# ])
```

In [ ]:

```
# X_transformed = column_transformer.fit_transform(X_train)
# X_transformednotLR = column_transformer_notLR.fit_transform(X_train)
# X_transformed.shape
```

```
# xtdf = pd.DataFrame(X_transformed)
# xtdf.head()
```

Conclusiones punto 4.

De acuerdo con las diferentes transformaciones numéricas que se hicieron tanto con las variables de entrada y la de salida, se llegó a la conclusión que las variables numéricas, LPE y 'Page total likes', iban a requerir un escalamiento puesto que eran las variables cuyos valores eran altísimos en contraste con las demás variables que eran categóricas, y ordinales discretas. Sin embargo, estas transformaciones solo fueron necesarias para lo que era la regresión lineal por la naturaleza del modelo en sí, si se tienen parámetros que tienen valores muy altos esto puede afectar en el modelo final dando un gran peso a esas variables en particular y dejando de lado a las variables ordinales que solo oscilan en un rango pequeño de valores. Con esto en mente, solamente se escalaron tales variables para ese modelo, en los demás modelos solo fue necesario hacerle una transformación de 'one hot encoder' a la variable de 'Type' debido a que la variable venia con strings como tal y no valores numéricos. Cabe destacar que la transformación que se le hizo a la variable de salida fue logaritmo natural lo cual permitió establecer el mismo espacio de valores que los de la variable de entrada.

Ejercicio-5.

Utiliza la función Dummy para modelos de regresión de scikit-learn con el conjunto que tienes de datos de entrenamiento y validación.

Para ello particiónalos en 100 para validación y 300 para entrenamiento.

Encuentra los errores RMSE, MAE y MAPE para los conjuntos de entrenamiento y validación.

Estos serán tus errores máximos que deberás tomar como referencia en el resto de la actividad.

Consulta su documentación correspondiente:

https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyRegressor.html

In [ ]:
```
X_train_dummy, X_test_dummy, y_train_dummy, y_test_dummy = train_test_split(
    X_train, y_train, test_size=0.25, random_state=SEED
)
print( "X Entrenamiento y validación",  X_train_dummy.shape)
```

```
print( "X Pruebas", X_test_dummy.shape)

print( "Y Entrenamiento y validación", y_train_dummy.shape)
print( "Y Pruebas", y_test_dummy.shape)
```

```
X Entrenamiento y validación (300, 7)
X Pruebas (100, 7)
Y Entrenamiento y validación (300,)
Y Pruebas (100,)
```

In [ ]:
```
# dummy_regressor = DummyRegressor(strategy='mean')
# X_train_dummy_transformed = pipeline.fit_transform(X_train_dummy)

# dummy_regressor.fit(X_train_dummy_transformed, y_train_dummy)


# y_predicted_dummy = dummy_regressor.predict(X_test_dummy)

# print("RMSE:\t %.2f" % ( RMSE(y_test_dummy, y_predicted_dummy) ) )
# print("MAE:\t %.2f" % ( MAE(y_test_dummy, y_predicted_dummy) ) )
# print("MAPE:\t %.2f" % ( MAPE(y_test_dummy, y_predicted_dummy) ) )
```

In [ ]:
```
pt = PowerTransformer(method='box-cox')
pt.fit(np.array(y_train).reshape(-1,1))
```

Out[ ]:
```
▾ PowerTransformer

PowerTransformer(method='box-cox')
```

In [ ]:
```
# Creamos un Dummy Regressor y lo implementamos en nuestros datos de entrenamiento y validacion
from sklearn.metrics import mean_absolute_percentage_error, mean_squared_error

scores_RMSE = []
scores_MAE = []
scores_MAPE = []
scores_MAPES = []
scores_MSE = []

def y_transformer(y):
    return np.log(y)

def y_inverse_transformer(y):
```

```
    return np.exp(y)

def dummyFuncion(X, Y):

    dummy_regr = DummyRegressor()
    pipe = Pipeline(steps=[('transformer', column_transformer), ('model', dummy_regr)])
    modelo = TransformedTargetRegressor(
        regressor=pipe,
        func=y_transformer,
        inverse_func=y_inverse_transformer
    )

    modelo.fit(X, Y)
    yhat = modelo.predict(X)

    scores_RMSE.append(RMSE(Y, yhat))
    scores_MAE.append(MAE(Y, yhat))
    scores_MAPE.append(MAPE(Y, yhat))
    scores_MAPES.append(mean_absolute_percentage_error(Y, yhat))
    scores_MSE.append(mean_squared_error(Y, yhat))


dummyFuncion(X_train_dummy, y_train_dummy)
dummyFuncion(X_test_dummy, y_test_dummy)
dummyFuncion(X_train, y_train)
dummyFuncion(X_test, y_test)


data = {'RMSE': scores_RMSE, 'MAE': scores_MAE, 'MAPE': scores_MAPE, 'MAPES': scores_MAPES, 'MSE': scores_MSE
scores = pd.DataFrame(data, index=['Dummy', 'DummyV', 'Entrenamiento', 'Validacion'])
scores
```

Out[ ]:

	RMSE	MAE	MAPE	MAPES	MSE
Dummy	642.392359	331.278409	79.357446	0.793574	412667.942602
DummyV	630.303186	374.933421	128.780938	1.287809	397282.105673
Entrenamiento	639.495510	342.315296	91.556930	0.915569	408954.507217
Validacion	623.694508	318.541325	105.506316	1.055063	388994.839377

Ejercicio-6.

Usando los modelos de regresión lineal múltiple, el bosque aleatorio y el perceptrón multicapa con sus valores predeterminados, lleva a cabo su entrenamiento con repeticiones de validación cruzada (RepeatedKFold) y desplegando los errores RMSE, MAE y MAPE.

Recuerda evitar el filtrado de información usando los datos que obtuviste en el ejercicio 2.

Incluye las conclusiones sobre el mejor modelo encontrado en esta primera aproximación. En particular ¿hay alguno sobreentrenado o subentrenado?

NOTA: Recuerda que puedes aumentar en dado caso el número máximo de iteraciones para que todos los modelos converjan.

In [ ]:
```
X_train_transformed = column_transformer.fit_transform(X_train)
X_train_notLR = column_transformer_notLR.fit_transform(X_train)

X_test_transformed = column_transformer.fit_transform(X_test)
X_test_notLR = column_transformer_notLR.fit_transform(X_test)
```

In [ ]:
```
pd.DataFrame(X_train_transformed).describe(include='all')
```

Out[ ]:

	0	1	2	3	4	5	6	7	8
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	4.000000e+02	4.000000e+02	4.000000e+02
mean	0.280000	0.852500	0.095000	0.012500	0.262500	0.310000	-1.909584e-16	-2.020606e-16	-1.953993e-16
std	0.449561	0.355048	0.293582	0.111242	0.440544	0.463072	1.001252e+00	1.001252e+00	1.001252e+00
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-1.794186e+00	-1.937762e+00	-2.043504e+00
25%	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	-9.736749e-01	-9.365562e-01	-1.126900e+00
50%	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	2.977761e-01	-1.498009e-02	4.046259e-01
75%	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	8.917192e-01	8.675760e-01	7.795961e-01
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.265366e+00	1.441096e+00	2.503540e+00

In [ ]:
```
pd.DataFrame(X_test_transformed).describe(include='all')
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	
count	100.000000	100.00000	100.000000	100.000000	100.000000	100.000000	1.000000e+02	1.000000e+02	1.000000e+02	1.00
mean	0.270000	0.85000	0.070000	0.020000	0.250000	0.310000	2.708944e-16	2.042810e-16	2.398082e-16	-2.
std	0.446196	0.35887	0.256432	0.140705	0.435194	0.464823	1.005038e+00	1.005038e+00	1.005038e+00	1.00
min	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	-1.744062e+00	-1.868513e+00	-1.546214e+00	-1.57
25%	0.000000	1.00000	0.000000	0.000000	0.000000	0.000000	-8.346497e-01	-7.587494e-01	-1.189846e+00	-9.09
50%	0.000000	1.00000	0.000000	0.000000	0.000000	0.000000	2.748084e-02	-1.325263e-01	5.618591e-01	1.19
75%	1.000000	1.00000	0.000000	0.000000	0.250000	1.000000	9.480266e-01	9.997144e-01	7.625397e-01	9.66
max	1.000000	1.00000	1.000000	1.000000	1.000000	1.000000	1.345537e+00	1.526768e+00	2.164838e+00	1.35

In [ ]:
```
repeated_k_fold = RepeatedKFold(n_splits=5, n_repeats=3, random_state=SEED)
```

In [ ]:
```
modelos, nombres = [], []
modelos.append(LinearRegression())
nombres.append('Regresión Lineal')

modelos.append(RandomForestRegressor())
nombres.append('Random Forest')

modelos.append(MLPRegressor(max_iter=20000))
nombres.append('Red Neuronal')
```

In [ ]:
```
resultados = []
for i in range(len(modelos)):
    pipe = Pipeline(steps=[('transformer', column_transformer), ('model', modelos[i])])
    model = TransformedTargetRegressor(
    regressor=pipe,
    func=np.log,
    inverse_func=np.exp,
    )
    my_metrics = {
      'RMSE': make_scorer(RMSE, greater_is_better=True),
      'MAE': make_scorer(MAE, greater_is_better=True),
      'MAPE': make_scorer(MAPE, greater_is_better=True),
    }
    scores = cross_validate(
```

```
        model,
        X_train,
        y_train,
        cv=repeated_k_fold,
        scoring=my_metrics,
        n_jobs=-1
    )

    # # pipe = make_pipeline((pipeline), (modelos[i]))

    # pipe = Pipeline(steps=[('transformer', pipeline), ('model', modelos[i])])
    # model = TransformedTargetRegressor(
    #     regressor=pipe,
    #     func=y_transformer,
    #     inverse_func=y_inverse_transformer
    # )

    # scores = cross_validate(model,
    #                         X_train,
    #                         y_train,
    #                         cv=repeated_k_fold,
    #                         scoring=my_metrics,
    #                         n_jobs=-1
    #                         )
      resultados.append(scores)

      print("%-20s\tRMSE: %.4f\tMAE: %.4f\tMAPE: %.4f" % (
          nombres[i],
          np.mean(scores['test_RMSE']),
          np.mean(scores['test_MAE']),
          np.mean(scores['test_MAPE'])
        ))
```

```
Regresión Lineal        RMSE: 534.7279  MAE: 284.8613   MAPE: 82.8042
Random Forest           RMSE: 549.5536  MAE: 300.7378   MAPE: 81.2235
Red Neuronal            RMSE: 578.0956  MAE: 320.7677   MAPE: 86.3141
```

Conclusiones punto 6.

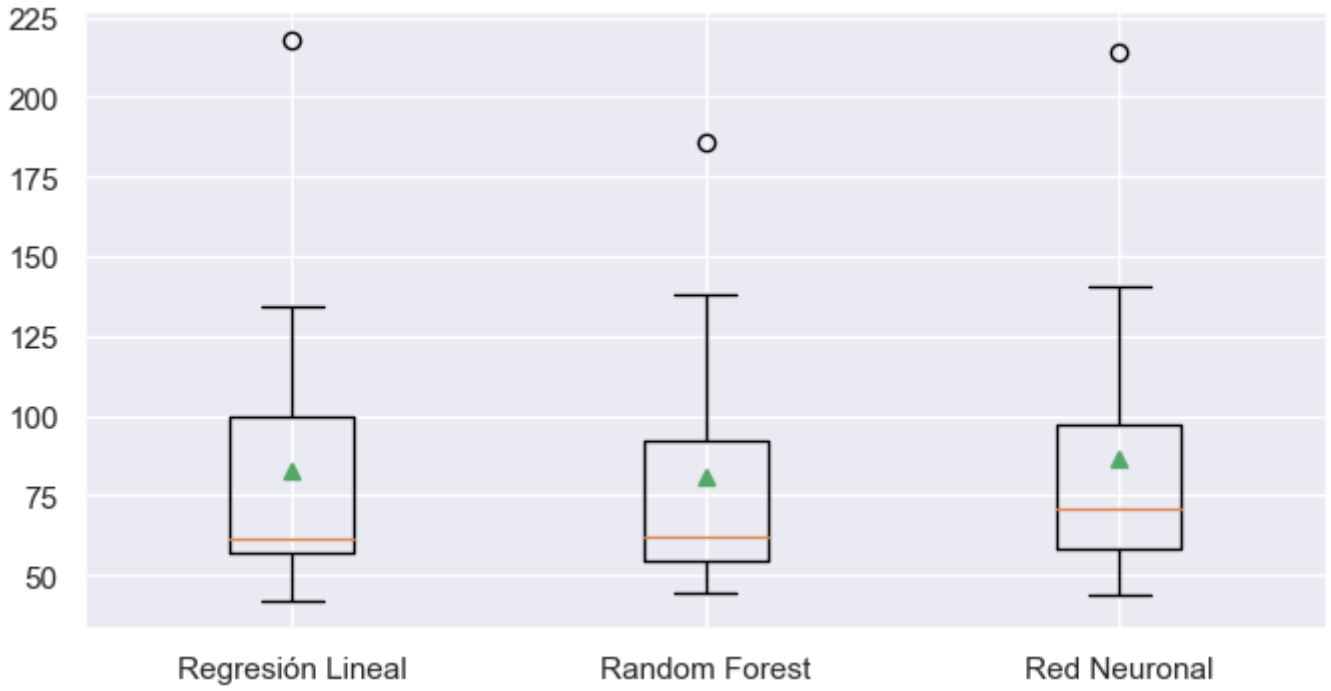
De esta primera iteración de modelos, se encontró que el modelo que mejor rendimiento tuvo en cuanto a la métrica MAPE fue el 'random forest' sin embargo, el erro cuadrático más pequeño se lo llevo el modelo de regresión lineal. De solamente los

resultados de estas métricas en esta primera iteración se puede observar que todos los modelos están subentrenados gracias a los valores obtenidos del error cuadrático RMSE y el porcentaje del error absoluto MAPE.

Ejercicio-7.

Obtener los diagramas de caja y bigote para los errores MAPE de los conjuntos de validación obtenidos. En particular compara estos primeros resultados de MAPE con el mejor resultado que encuentran los autores del artículo citado al inicio. Incluye tus conclusiones.

In [ ]:
```
sns.set(rc={'figure.figsize':(8,4)})
boxPlotsMape = []
for i in range(len(resultados)):
  rr = resultados[i]['test_MAPE']
  boxPlotsMape.append(rr)

plt.boxplot(boxPlotsMape, labels=nombres, showmeans=True)
plt.show()
```



Conclusiones punto 7.

Comparando con los resultados del artículo se puede observar que las métricas obtenidas de cada modelo tienen una gran diferencia al 27% que es el limite que obtienen en el artículo. Por el momento queda correr un análisis con el 'GridsearchCV' para encontrar los mejores hiperparametros de cada modelo y ver si hay alguna mejora con la métrica de MAPE.

In [ ]:

Ejercicio-8.

Usando una búsqueda de malla con validación cruzada (GridSearchCV), busca los mejores hiperparámetros para el modelo MLP.

Al menos deberás realizar la búsqueda en los hiperparámetros "hidden_layer_sizes", "alpha" y "learning_rate_init". Además aplica la validación cruzada con repeticiones (RepeatedKFold). Muestra los mejores hiperparámetros encontrados.

In [ ]:
```
def plot_importance(importance, model_type, title):
  for i,v in enumerate(importance['importances_mean']):
    print('Feature: %0d, Score: %.5f' % (i,v))


  plt.bar(
    [x for x in range(len(importance['importances_mean']))],
    importance['importances_mean']
  )

  plt.title(title)
  plt.suptitle(model_type)
  plt.show()
```

In [ ]:
```
parameters = {
  'model__regressor__hidden_layer_sizes': [(i, i) for i in range(5, 20, 2)],
  'model__regressor__alpha': [0.0001, 0.001, 0.01, 0.1, 0.99],
  'model__regressor__learning_rate_init': [0.001, 0.01, 0.1],
}


mi_regressor = TransformedTargetRegressor(
    regressor=MLPRegressor(max_iter=50000, random_state=SEED),
```

```
    func=np.log1p,
    inverse_func=np.expm1
)

mlp_pipe = Pipeline(steps=[
    ('preprocessor', column_transformer),
    ('model', mi_regressor)
])

mlpGrid = GridSearchCV(
  mlp_pipe,
  parameters,
  cv=repeated_k_fold,
  scoring=make_scorer(MAPE, greater_is_better=True),
  n_jobs=-1
)
mlpGrid.fit(X_train, y_train)

print('Mejor valor de obtenido con la mejor combinación: {:.5f}'.format(mlpGrid.best_score_))
print('Mejor combinación de valores encontrados de los hiperparámetros:', mlpGrid.best_params_)
print('Métrica utilizada:', mlpGrid.scoring)
```

```
Mejor valor de obtenido con la mejor combinación: 98.01500
Mejor combinación de valores encontrados de los hiperparámetros: {'model__regressor__alpha': 0.0001, 'model__
regressor__hidden_layer_sizes': (17, 17), 'model__regressor__learning_rate_init': 0.001}
Métrica utilizada: make_scorer(MAPE)
```

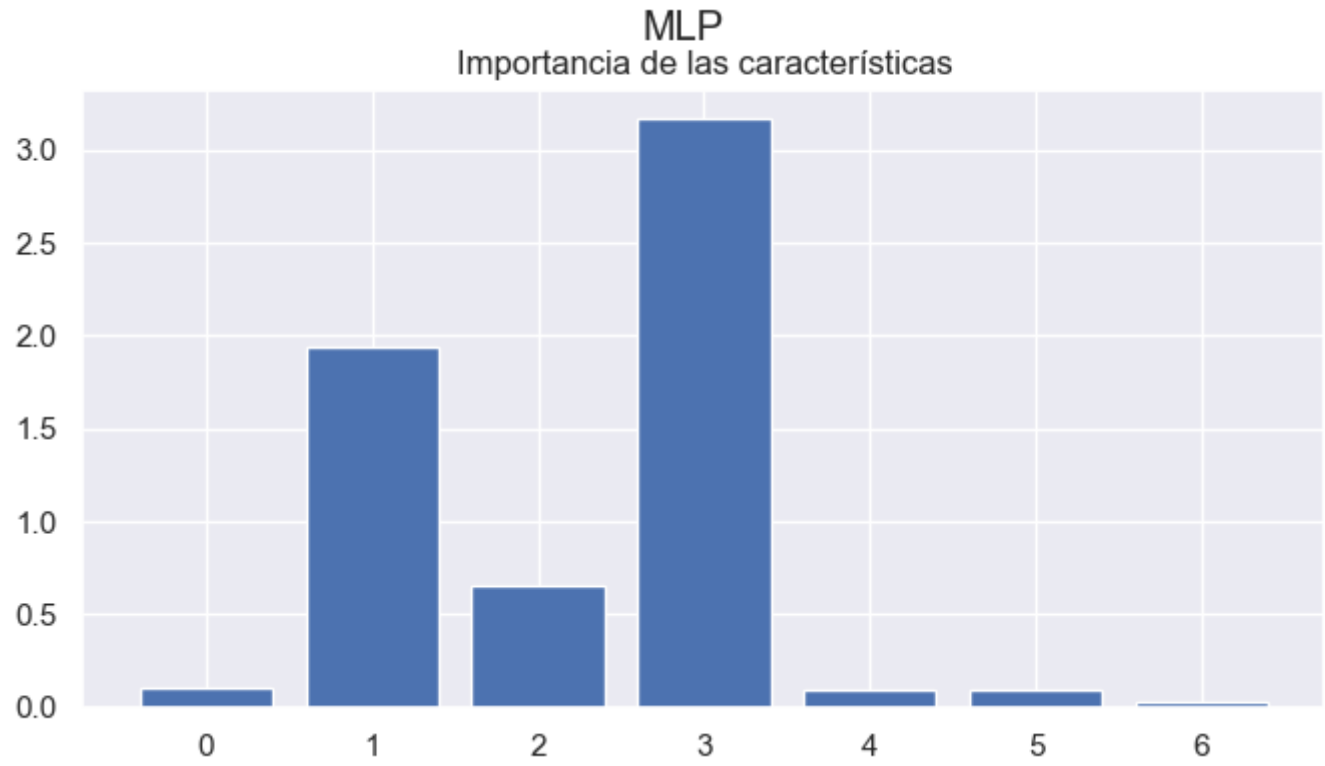
Ejercicio-9.

Con los mejores valores de los hiperparámetros encontrados realiza un análisis de la importancia de los factores. Muestra un diagrama de barras de los resultados e incluye tus conclusiones.

In [ ]:
```
final_MLP = MLPRegressor(
  alpha=0.001,
  hidden_layer_sizes=(17, 17),
  learning_rate_init= 0.001,
  max_iter=50000,
  random_state=SEED
)

finalRegressor = TransformedTargetRegressor(
```

```
  regressor=final_MLP,
  func=np.log1p,
  inverse_func=np.expm1
)

finalPipe = Pipeline(
  steps=[
    ('transformer', column_transformer),
    ('model', finalRegressor)
  ]
)
finalPipe.fit(X_train, y_train)
importance = permutation_importance(finalPipe, X_train, y_train, n_repeats=10)

feature_names = column_transformer_notLR.get_feature_names_out()
print(feature_names)
plot_importance(importance, 'MLP', 'Importancia de las características')
```

```
['type__Type_Photo' 'type__Type_Status' 'type__Type_Video'
 'type__Category_2' 'type__Category_3']
Feature: 0, Score: 0.10164
Feature: 1, Score: 1.93674
Feature: 2, Score: 0.65828
Feature: 3, Score: 3.16375
Feature: 4, Score: 0.09620
Feature: 5, Score: 0.09976
Feature: 6, Score: 0.03383
```



In [ ]:
```
y_hat_train = finalPipe.predict(X_train)
y_hat_test = finalPipe.predict(X_test)

print(
  "TRAIN: RMSE: %.4f\t MAE: %.4f\t MAPE: %.4f" %
  ( RMSE(y_train, y_hat_train), MAE(y_train, y_hat_train), MAPE(y_train, y_hat_train))
)

print(
  "TEST: RMSE: %.4f\t MAE: %.4f\t MAPE: %.4f" %
  ( RMSE(y_test, y_hat_test), MAE(y_test, y_hat_test), MAPE(y_test, y_hat_test))
)
```

```
TRAIN: RMSE: 471.8635    MAE: 251.6068   MAPE: 58.7702
TEST: RMSE: 533.2159     MAE: 281.5281   MAPE: 88.1494
```

Conclusiones punto 9.

Podemos observar con el modelo de red neuronal que el parámetro que mas peso tuvo en los resultados finales fue la variable 'category' con uno de los valores y la segunda que mayor peso tuvo fue la variable 'type' específicamente con el valor de 'status'. Es interesante observar la importancia de estas dos variables y si fuera posible, checar con futuros datos si este comportamiento es el esperado o solamente una peculiaridad de este data set.

In [ ]:

Ejercicio-10.

Repite el ejercicio 8 y 9 para el modelo de bosque aleatorio para buscar sus mejores hiperparámetros (realiza la búsqueda con aquellos hiperparámetros que consideres más adecuados) y usando el conjunto de **Prueba**.

Y realiza igualmente el análisis de importancia de factores con este modelo con un diagrama de barras.

USAR EL GRID SEARCH X_test

In [ ]:
```
parameters = {
  'model__regressor__n_estimators': [100, 200, 700],
  'model__regressor__max_features': ['sqrt', 'log2'],
  'model__regressor__max_depth' : [4,5,6,7,8],
  'model__regressor__ccp_alpha': [0.0001, 0.001, 0.05],
}

mi_regressor = TransformedTargetRegressor(
    regressor=RandomForestRegressor(random_state=SEED),
    func=np.log1p,
    inverse_func=np.expm1 ,
)

rfr_pipe = Pipeline(steps=[
    ('preprocessor', column_transformer),
    ('model', mi_regressor)
])

rfGrid = GridSearchCV(
  rfr_pipe,
  parameters,
```

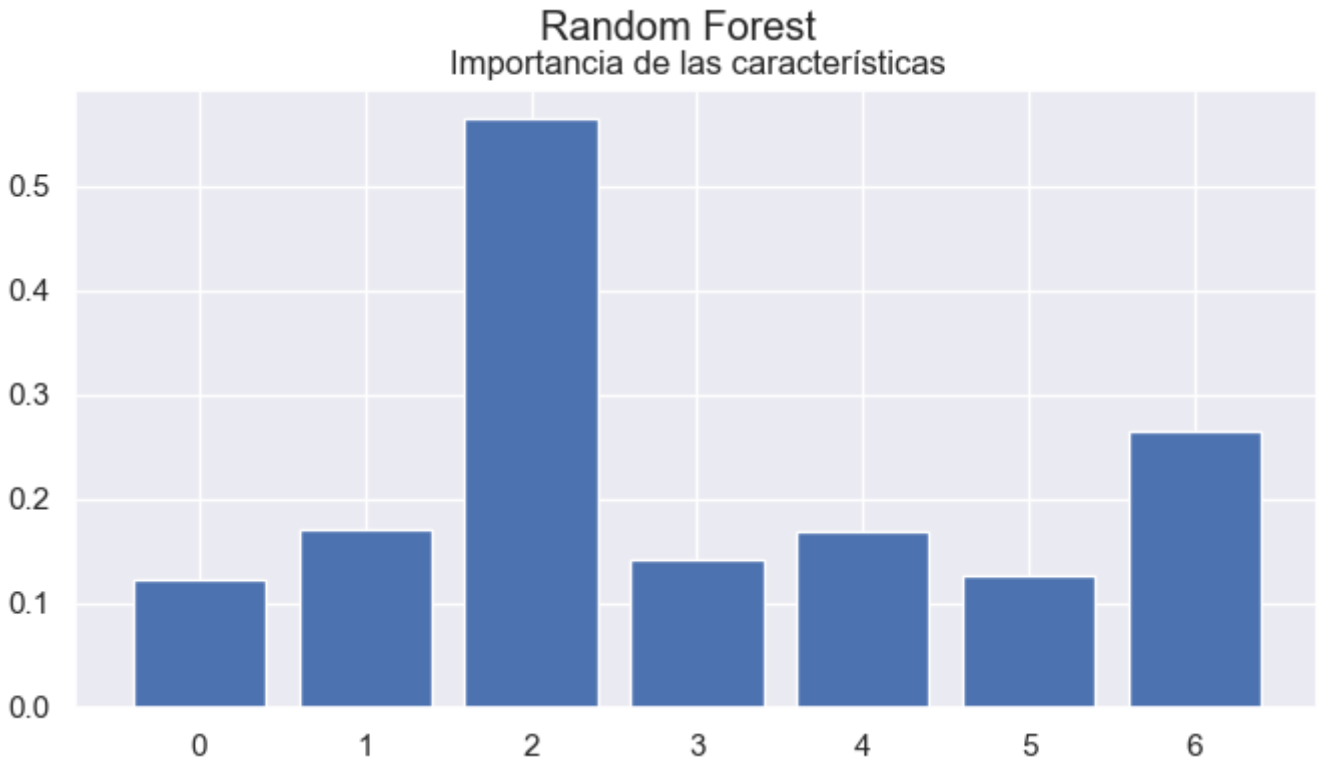
```
  cv=repeated_k_fold,
  scoring=make_scorer(MAPE, greater_is_better=True),
  n_jobs=-1
)
rfGrid.fit(X_test, y_test)

print('Mejor valor de obtenido con la mejor combinación: {:.5f}'.format(rfGrid.best_score_))
print('Mejor combinación de valores encontrados de los hiperparámetros:', rfGrid.best_params_)
print('Métrica utilizada:', rfGrid.scoring)
```

```
Mejor valor de obtenido con la mejor combinación: 105.29730
Mejor combinación de valores encontrados de los hiperparámetros: {'model__regressor__ccp_alpha': 0.001, 'mode
l__regressor__max_depth': 8, 'model__regressor__max_features': 'sqrt', 'model__regressor__n_estimators': 200}
Métrica utilizada: make_scorer(MAPE)
```

In [ ]:

```
final_RFR = RandomForestRegressor(
  ccp_alpha=0.001,
  max_depth=8,
  max_features='sqrt',
  n_estimators=200,
  random_state=SEED
)

finalRegressor = TransformedTargetRegressor(
  regressor=final_RFR,
  func=np.log1p,
  inverse_func=np.expm1
)

finalPipe = Pipeline(
  steps=[
    ('transformer', column_transformer),
    ('model', finalRegressor)
  ]
)
finalPipe.fit(X_test, y_test)
importance = permutation_importance(finalPipe, X_test, y_test, n_repeats=10)

feature_names = column_transformer.get_feature_names_out()
print(feature_names)
plot_importance(importance, 'Random Forest', 'Importancia de las características')
```

```
['categoric__Paid' 'onehot__Type_Photo' 'onehot__Type_Status'
 'onehot__Type_Video' 'onehot__Category_2' 'onehot__Category_3'
 'numeric__Page total likes' 'numeric__Post Month' 'numeric__Post Hour'
 'numeric__Post Weekday']
Feature: 0, Score: 0.12230
Feature: 1, Score: 0.17069
Feature: 2, Score: 0.56364
Feature: 3, Score: 0.14244
Feature: 4, Score: 0.16821
Feature: 5, Score: 0.12592
Feature: 6, Score: 0.26425
```



In [ ]:
```
y_hat_train = finalPipe.predict(X_train)
y_hat_test = finalPipe.predict(X_test)

print(
  "TRAIN: RMSE: %.4f\t MAE: %.4f\t MAPE: %.4f" %
  ( RMSE(y_train, y_hat_train), MAE(y_train, y_hat_train), MAPE(y_train, y_hat_train))
)
```

```
print(
  "TEST: RMSE: %.4f\t MAE: %.4f\t MAPE: %.4f" %
  ( RMSE(y_test, y_hat_test), MAE(y_test, y_hat_test), MAPE(y_test, y_hat_test))
)
```

```
TRAIN: RMSE: 576.5087    MAE: 302.6650   MAPE: 77.8405
TEST: RMSE: 357.3188     MAE: 166.6639   MAPE: 30.7272
```

Ejercicio-11.

Repite el ejercicio 8 y 9 para el modelo de regresión lineal múltiple para buscar sus mejores hiperparámetros (realiza la búsqueda con aquellos hiperparámetros que consideres más adecuados) y usando el conjunto de **Prueba**.

Y realiza igualmente el análisis de importancia de factores con este modelo con un diagrama de barras.

In [ ]:
```
parameters = {
    'model__regressor__fit_intercept': [True, False],
    'model__regressor__copy_X': [True, False],
    'model__regressor__positive': [True, False],
}

mi_regressor = TransformedTargetRegressor(
    regressor=LinearRegression(),
    func=np.log1p,
    inverse_func=np.expm1
)

rlm_pipe = Pipeline(steps=[
    ('preprocessor', column_transformer),
    ('model', mi_regressor)
])

rlmGrid = GridSearchCV(
  rlm_pipe,
  parameters,
  cv=repeated_k_fold,
  scoring=make_scorer(MAPE, greater_is_better=True),
  n_jobs=-1
)
rlmGrid.fit(X_test, y_test)
```

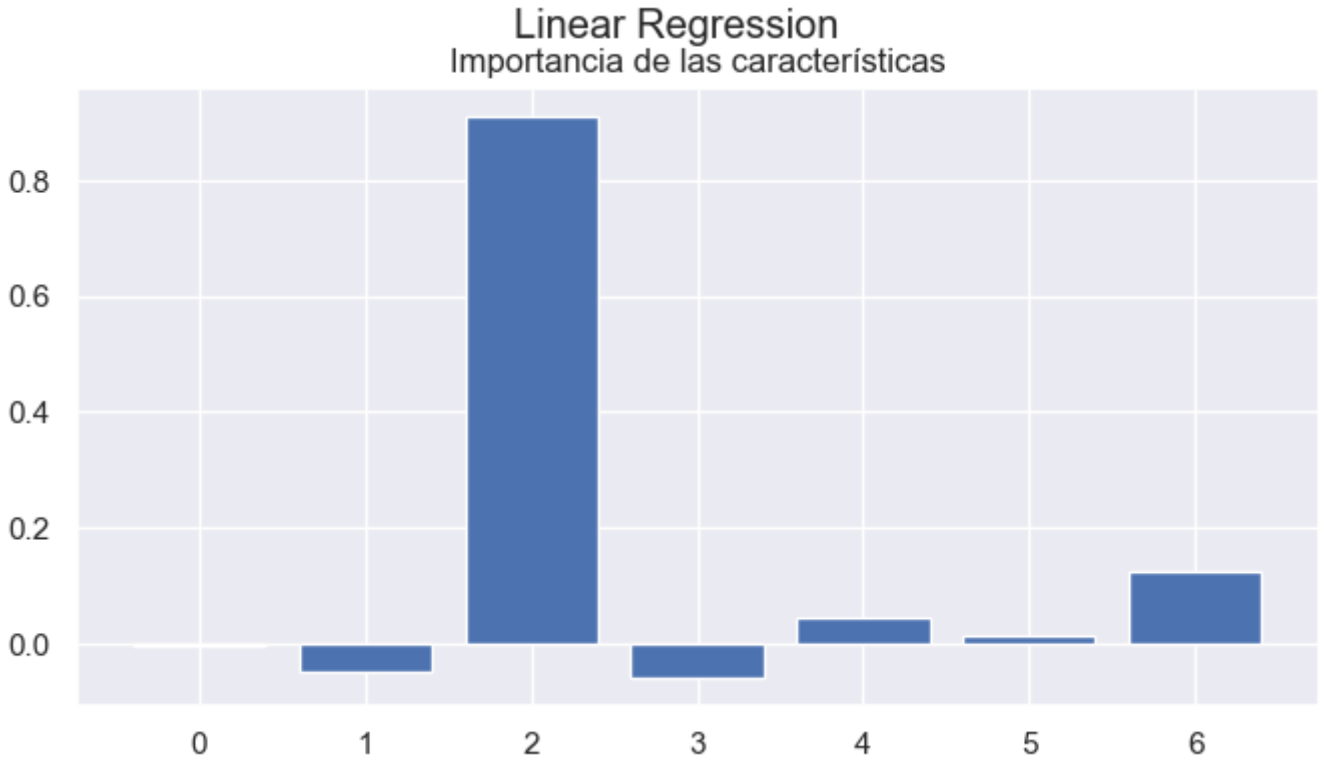
```
print('Mejor valor de obtenido con la mejor combinación: {:.5f}'.format(rlmGrid.best_score_))
print('Mejor combinación de valores encontrados de los hiperparámetros:', rlmGrid.best_params_)
print('Métrica utilizada:', rlmGrid.scoring)
```

```
Mejor valor de obtenido con la mejor combinación: 107.64131
Mejor combinación de valores encontrados de los hiperparámetros: {'model__regressor__copy_X': True, 'model__r
egressor__fit_intercept': False, 'model__regressor__positive': False}
Métrica utilizada: make_scorer(MAPE)
```

In [ ]:

```
final_LR = LinearRegression(
  copy_X=True,
  fit_intercept=False,
  positive=False,
)

finalRegressor = TransformedTargetRegressor(
  regressor=final_LR,
  func=np.log1p,
  inverse_func=np.expm1,
)

finalPipe = Pipeline(
  steps=[
    ('transformer', column_transformer),
    ('model', finalRegressor)
  ]
)
finalPipe.fit(X_test, y_test)
importance = permutation_importance(finalPipe, X_test, y_test, n_repeats=20, n_jobs=-1)

feature_names = column_transformer.get_feature_names_out()
print(feature_names)
plot_importance(importance, 'Linear Regression', 'Importancia de las características')
```

```
['categoric__Paid' 'onehot__Type_Photo' 'onehot__Type_Status'
 'onehot__Type_Video' 'onehot__Category_2' 'onehot__Category_3'
 'numeric__Page total likes' 'numeric__Post Month' 'numeric__Post Hour'
 'numeric__Post Weekday']
Feature: 0, Score: -0.00480
Feature: 1, Score: -0.04795
Feature: 2, Score: 0.91084
Feature: 3, Score: -0.05817
Feature: 4, Score: 0.04519
Feature: 5, Score: 0.01485
Feature: 6, Score: 0.12600
```



In [ ]:
```
y_hat_train = finalPipe.predict(X_train)
y_hat_test = finalPipe.predict(X_test)

print(
  "TRAIN: RMSE: %.4f\t MAE: %.4f\t MAPE: %.4f" %
  ( RMSE(y_train, y_hat_train), MAE(y_train, y_hat_train), MAPE(y_train, y_hat_train))
)
```

```
print(
  "TEST: RMSE: %.4f\t MAE: %.4f\t MAPE: %.4f" %
  ( RMSE(y_test, y_hat_test), MAE(y_test, y_hat_test), MAPE(y_test, y_hat_test))
)
```

```
TRAIN: RMSE: 601.7524    MAE: 348.6141   MAPE: 99.9628
TEST: RMSE: 532.5092     MAE: 310.2518   MAPE: 87.8504
```

Conclusiones punto 10 y 11.

RF – Con el análisis de Random Forest se puede observar que, de nuevo, la variable con mas importancia es la de 'type' con el valor de 'status'. Estos son resultados similares a los obtenidos con la red neuronal. Sin embargo, a diferencia de la red neuronal ahora se tiene un poco de mayores pesos en las otras variables con excepción de categoría.

LR – Finalmente con el análisis de regresión lineal, cuyas variables fueron preprocesadas con la intención de facilitar ese algoritmo, se puede de nuevo observar la importancia de la misma variable 'type' y con el valor de 'status' sin embargo en este caso en particular se puede observar que también cae un mayor peso a otro valor de esta misma variable la cual es 'video'. Con esto se puede concluir que de las 7 variables de entrada que se escogieron para el análisis la variable de 'type' es la que mayor peso tiene en las predicciones finales. Esto puede ser un punto de partida para otro análisis de exploración sobre esta variable en particular y como afecta las predicciones finales.

Ejercicio-12.

Compara tus resultados con los obtenidos por los autores del artículo de Moro-Rita-Vala con respecto a MAPE. Incluye tus conclusiones finales de la actividad.

En conclusión, ninguno de los modelos resulto acercarse a la métrica de MAPE que obtienen en el articulo de Moro-Rita-Vala. La razón de esto en primera son los modelos utilizados en el articulo y los modelos utilizados para esta actividad, la manera en la que llegan a la métrica del 27% es utilizando una SVM la cual no fue considerada para nuestro análisis. Por parte de la actividad, fue interesante considerar el preprocesamiento que deben tener los datos antes de alimentarse a un algoritmo de aprendizaje como la regresión lineal. La importancia del rango de datos puede tener una gran influencia en el rendimiento y la precisión del modelo la cual puede afectar los resultados enormemente. De igual manera, es importante hacer notar que la naturaleza del algoritmo SVM es capaz de tomar el espacio de las variables de entrada y transformarlo a otro plano (usualmente dimensiones mayores) que pueda ser utilizado con mayor facilidad para este problema de regresión. Al momento de regresar un resultado

puede regresarlo en el mismo espacio que entraron las variables de entrada, con esto se puede evitar todo el análisis que se llevó a cabo para el procesamiento de datos como lo fue para el algoritmo de regresión lineal.

In [ ]:

Fin de la Actividad de la semana 7.