IEU PORTAFOLIO de Eduardo Martín Rico Sotomayor

In [119]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import power_transform, FunctionTransformer
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncod
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.dummy import DummyRegressor
from sklearn.preprocessing import PowerTransformer
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score, cross_validate, Repeat
from sklearn.inspection import permutation_importance

from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
df = pd.read_csv('./dataset_Facebook.csv', sep=";")
```

In [3]:
```
df.columns
```

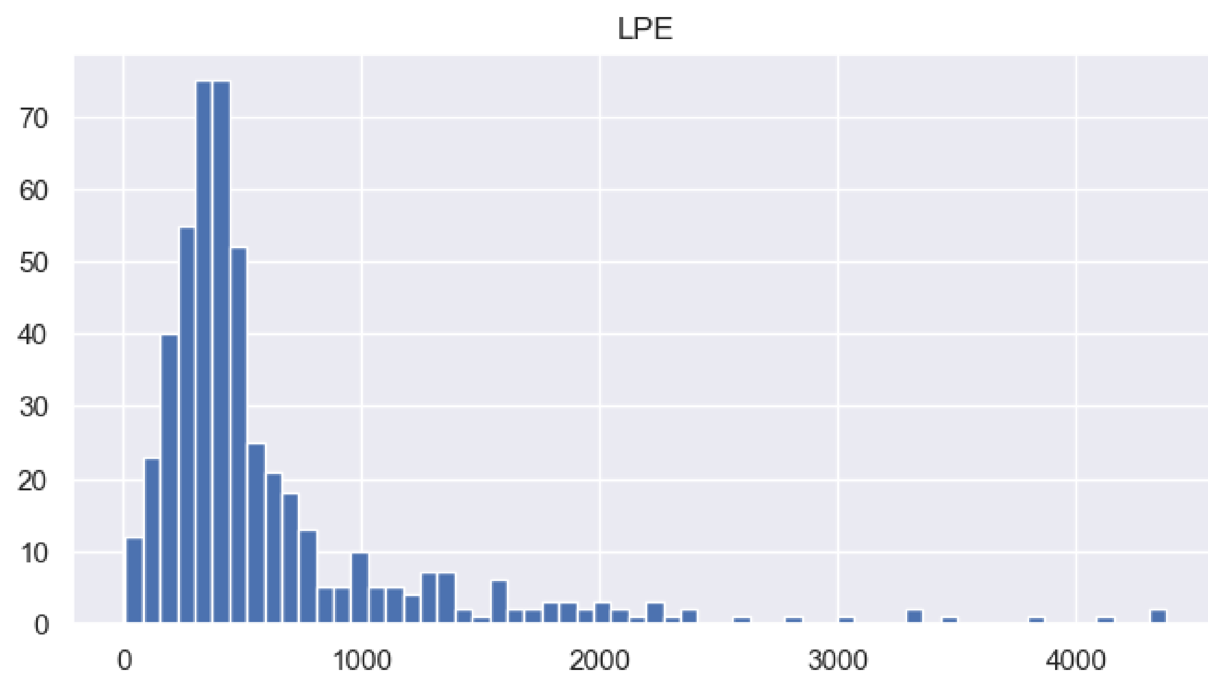
Out[3]:
```
Index(['Page total likes', 'Type', 'Category', 'Post Month', 'Post Weekda
y',
       'Post Hour', 'Paid', 'Lifetime Post Total Reach',
       'Lifetime Post Total Impressions', 'Lifetime Engaged Users',
       'Lifetime Post Consumers', 'Lifetime Post Consumptions',
       'Lifetime Post Impressions by people who have liked your Page',
       'Lifetime Post reach by people who like your Page',
       'Lifetime People who have liked your Page and engaged with your po
st',
       'comment', 'like', 'share', 'Total Interactions'],
      dtype='object')
```

In [4]:
```
columnas = ['Category', 'Page total likes', 'Type', 'Post Month', 'Post Hou
            'Lifetime People who have liked your Page and engaged with your
            ]
df = df[columnas]
df = df.rename(columns={'Lifetime People who have liked your Page and engag
numericas=['Page total likes']
categoricas=['Category', 'Type']
binarias=['Paid']
ordinales=['Post Month', 'Post Weekday', 'Post Hour']
```

In [95]:
```
df_x = df[df.columns.difference(['LPE'])]
df_y = df[['LPE']]
Xtrain, Xtest, Ytrain, Ytest = train_test_split(df_x, df_y, test_size=100,
```

In [242]:
```
df_y.hist(bins=60)
```

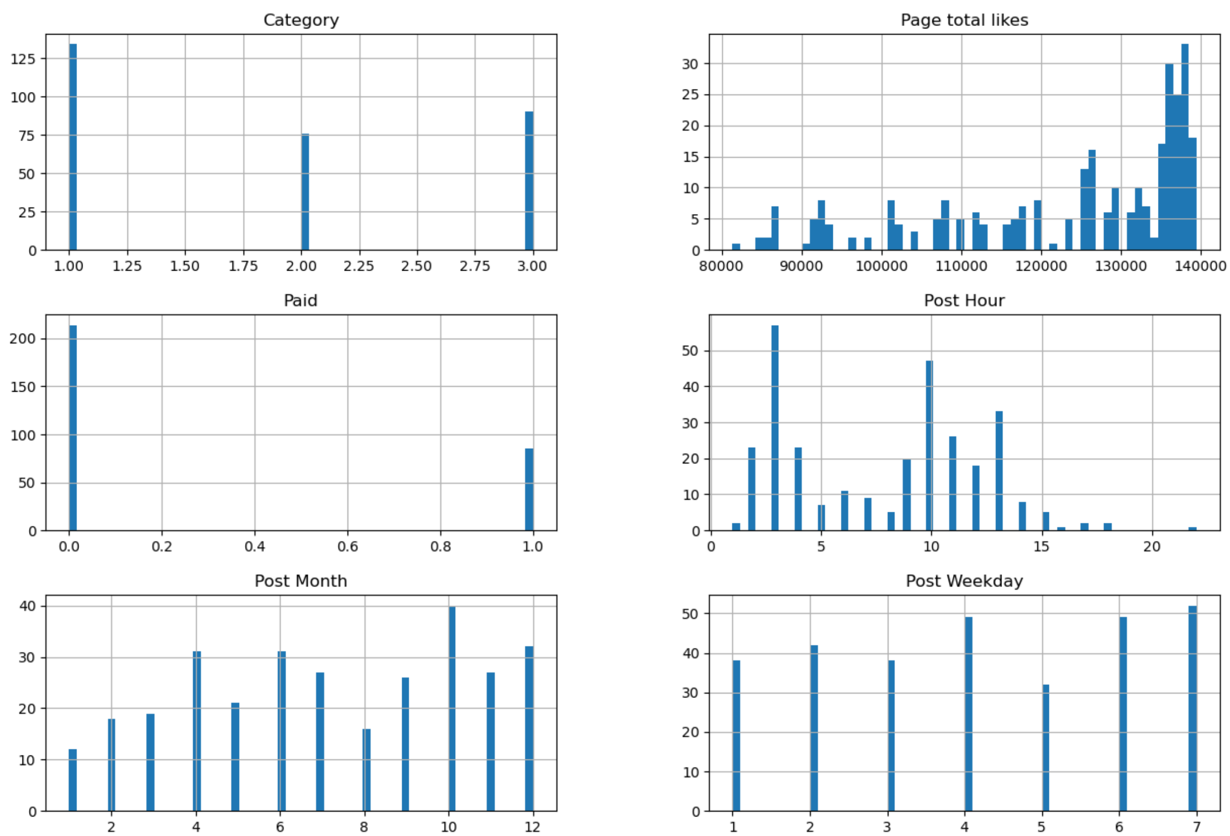
Out[242]:
```
array([[<AxesSubplot:title={'center':'LPE'}>]], dtype=object)
```



In [6]:
```python
def RMSE(y_val, yhat_val):
  diff = np.subtract(y_val, yhat_val)
  n = len(diff)
  diff_pow_2 = diff**2
  sse = np.sum(diff_pow_2)
  mse = sse/n
  rmse = np.sqrt(mse)
  return rmse

def MAE(y_val, yhat_val):
  diff = np.subtract(y_val, yhat_val)
  n = len(diff)
  abs_diff = np.abs(diff)
  return (1/n)*(np.sum(abs_diff))

def MAPE(y_true, y_hat):
    return np.mean(np.abs(np.divide((y_true- y_hat),y_true)))*100
```

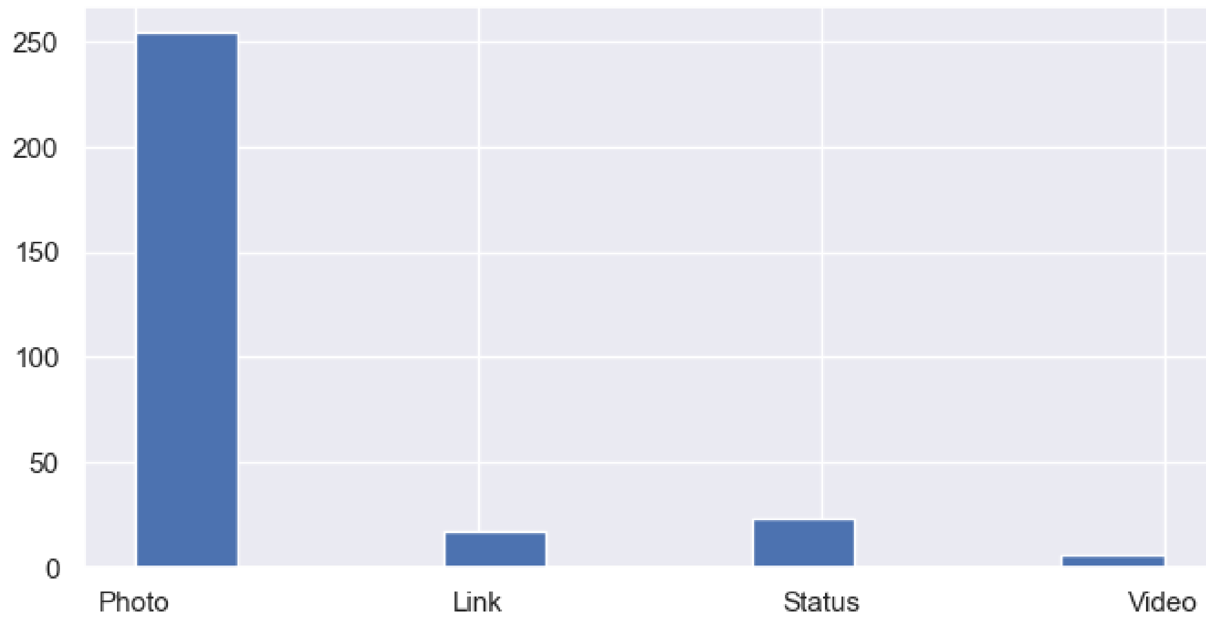
In [7]:
```python
Xtrain.hist(bins=60, figsize=(15, 10))
```

Out[7]:
```
array([[<AxesSubplot:title={'center':'Category'}>,
        <AxesSubplot:title={'center':'Page total likes'}>],
       [<AxesSubplot:title={'center':'Paid'}>,
        <AxesSubplot:title={'center':'Post Hour'}>],
       [<AxesSubplot:title={'center':'Post Month'}>,
        <AxesSubplot:title={'center':'Post Weekday'}>]], dtype=object)
```



In [50]:
```
Xtrain.Type.hist()
```

Out[50]: <AxesSubplot:>



In [96]:
```
cv = RepeatedKFold(n_splits=5, n_repeats=3)

columnas_final = numericas + ordinales + ['Category_1', 'Category_2', 'Cate

box_cox_scale_transformer = make_pipeline(
    PowerTransformer(method='box-cox'),
    MinMaxScaler(feature_range=(1,2))
)

numerical_pipeline = Pipeline(steps = [
     ("imputer", SimpleImputer(strategy='most_frequent')),
     ("box-cox-scaler", box_cox_scale_transformer)

])

ord_pipe = Pipeline([
    ('escalaNum', MinMaxScaler(feature_range=(1,2))),
])


cat_pipe = Pipeline([
    ('impModa', SimpleImputer(strategy='most_frequent')),
    ('OneHotE', OneHotEncoder(handle_unknown='ignore'))
])


bin_pipe = Pipeline([
    ('impModa', SimpleImputer(strategy='most_frequent')),
    ('OneHotEncoder', OneHotEncoder(handle_unknown='ignore'))
])


preprocessor_pipeline = ColumnTransformer(transformers=[
    ("numerical_pipeline_input", numerical_pipeline, numericas),
    ("ord_pipe_input", ord_pipe, ordinales),
    ("cat_pipe_input", cat_pipe, categoricas),
    ("binarias", bin_pipe, binarias),
])
#hacer el cambio de Ytrain y Ytest

y_transform = ColumnTransformer(
    transformers = [
      ('numerical_out', numerical_pipeline, ['LPE'])
    ]
)

Ytrain = pd.DataFrame(y_transform.fit_transform(Ytrain))
Ytest = pd.DataFrame(y_transform.fit_transform(Ytest))

#modelos
dummy = Pipeline(
    [
        ("preprocessor", preprocessor_pipeline),
        ("regressor", DummyRegressor()),
    ]
)
```

```
lineal = Pipeline(
    [
        ("preprocessor", preprocessor_pipeline),
        ("regressor", LinearRegression()),
    ]
)

arbol = Pipeline(
    [
        ("preprocessor", preprocessor_pipeline),
        ("regressor", RandomForestRegressor(ccp_alpha=0.8, criterion='squar
    ]
)

mlp = Pipeline(
    [
        ("preprocessor", preprocessor_pipeline),
        ("regressor", MLPRegressor(max_iter=5000)),
    ]
)
```

In [10]:

```
dummy.fit(Xtrain, Ytrain)
lineal.fit(Xtrain, Ytrain)
arbol.fit(Xtrain, Ytrain)
mlp.fit(Xtrain, Ytrain)

models = [dummy, lineal, arbol, mlp]
models_name = ["dummy", "lineal", "arbol", "mlp"]
```

In [11]:

```
from sklearn.model_selection import learning_curve

#Esta función crea la curva de aprendizaje
def create_learning_curve(model, X, y, scoring="max_error"):
    sizes_perc = np.linspace(0.1,1.0,30)
    return learning_curve(
        estimator=model,
        X= X,
        cv=cv,
        y= y.values.ravel(),
        train_sizes=sizes_perc,
        scoring=scoring,
        n_jobs=-1)
```

In [12]:
```
import matplotlib.pyplot as plt

def plot_learning_curve(model, X, y, scoring="max_error", scoring_name="max

    #Antes de graficar, hay que crear la curva, verdad?
    train_sizes, train_scores, val_scores = create_learning_curve(model, X,

    train_avg = np.mean(train_scores, axis = 1)
    val_avg = np.mean(val_scores, axis = 1)

    plt.figure(figsize=(8,7))

    plt.plot(train_sizes, train_avg, color = 'green', marker="o", label="Tr

    plt.plot(train_sizes,val_avg, color="red", marker="+", linestyle="--",

    plt.title(f'Curvas de Aprendizaje de {model_name}')
    plt.xlabel('Tamaño del conjunto de entrenamiento')

    plt.ylabel(f'{scoring_name}')
    plt.legend(loc='lower right')
    plt.show()
```

In [13]:
```
rmse = make_scorer(RMSE)
mae = make_scorer(MAE)
mape = make_scorer(MAPE)
scores = [rmse, mae, mape]
scores_names = ["rmse", "mae", "mape"]
```

In [14]:
```
for idx, model in enumerate(models):
  for idy, score in enumerate(scores):
    plot_learning_curve(model, Xtrain, Ytrain, scoring=score,
                        scoring_name=scores_names[idy], model_name=models_n
```

```
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is t
aken will be eliminated, and the value None will no longer be accepted. S
et `keepdims` to True or False to avoid this warning.
  mode = stats.mode(array)
/Users/emrs/opt/anaconda3/lib/python3.9/site-packages/sklearn/impute/_bas
e.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is t
aken will be eliminated, and the value None will no longer be accepted. S
et `keepdims` to True or False to avoid this warning.
  mode = stats.mode(array)
/Users/emrs/opt/anaconda3/lib/python3.9/site-packages/sklearn/impute/_bas
e.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is t
aken will be eliminated, and the value None will no longer be accepted. S
```

In [15]:
```
y_hat_dummy = dummy.predict(Xtest)
y_hat_lineal = lineal.predict(Xtest)
y_hat_arbol = arbol.predict(Xtest)
y_hat_mlp = mlp.predict(Xtest)

trained_models = [y_hat_dummy, y_hat_lineal, y_hat_arbol, y_hat_mlp]
trained_models_names = ["DUMMY", 'LINEAL', 'ARBOLE', 'MLP']
```

In [16]:
```
scores = [RMSE, MAE, MAPE]
scores_names = ["rmse", "mae", "mape"]
```

In [17]:
```
for idy, y_hats in enumerate(trained_models):
  for idx, score in enumerate(scores):
    print(f"MODELO: {trained_models_names[idy]} -- El error {scores_names[i
```

```
MODELO: DUMMY -- El error rmse es: 0.19521008236065562
MODELO: DUMMY -- El error mae es: 0.14420767977069412
MODELO: DUMMY -- El error mape es: 10.015796131335104
MODELO: LINEAL -- El error rmse es: 2.161907857138881
MODELO: LINEAL -- El error mae es: 16.17593079616144
MODELO: LINEAL -- El error mape es: 11.227621579000608
MODELO: ARBOLE -- El error rmse es: 0.1797735149443987
MODELO: ARBOLE -- El error mae es: 0.12096763185256572
MODELO: ARBOLE -- El error mape es: 8.70333214587322
MODELO: MLP -- El error rmse es: 0.20666398787169957
MODELO: MLP -- El error mae es: 0.1483581082717328
MODELO: MLP -- El error mape es: 10.609602359069768
```

In [18]:
```
def calculate_error_functions(y_real, y_pred):
  results = pd.DataFrame({
      'Error function': [
          'RMSE',
          'MAE',
          'MAPE'
      ],
      'Score': [
          RMSE(y_real, y_pred),
          MAE(y_real, y_pred),
          MAPE(y_real, y_pred)
      ]
  })
  return results

calculate_error_functions(Ytest.values.ravel(), y_hats).head()
```

Out[18]:

	Error function	Score
0	RMSE	0.206664
1	MAE	0.148358
2	MAPE	10.609602

In [150]:
```
models = {
    "LinearRegression": lineal,
    "RandomForest": arbol,
    "MultilayerPerceptron": mlp
}

metrics = {
    "RMSE": make_scorer(RMSE),
    "MAE": make_scorer(MAE),
    "MAPE": make_scorer(MAPE)
}

results, names = [], []

for name, model in models.items():

  kfold = RepeatedKFold(n_splits=5, n_repeats=3)

  model_results = cross_validate(model, Xtrain, Ytrain.values.ravel(), scor

  results.append(model_results)
  names.append(name)

  results_df = pd.DataFrame({
      'Error function': [
          'RMSE',
          'MAE',
          'MAPE'
      ],
      'Validation Mean': [
          np.mean(model_results['test_RMSE']),
          np.mean(model_results['test_MAE']),
          np.mean(model_results['test_MAPE'])
      ],
      'Validation Std': [
          np.std(model_results['test_RMSE']),
          np.std(model_results['test_MAE']),
          np.std(model_results['test_MAPE'])
      ],
      'Training Mean': [
          np.mean(model_results['train_RMSE']),
          np.mean(model_results['train_MAE']),
          np.mean(model_results['train_MAPE'])
      ],
      'Training Std': [
          np.std(model_results['train_RMSE']),
          np.std(model_results['train_MAE']),
          np.std(model_results['train_MAPE'])
      ]
  })
  print(f"\n")
  print(name)
  print(results_df.head())
  print(f"\n")
```

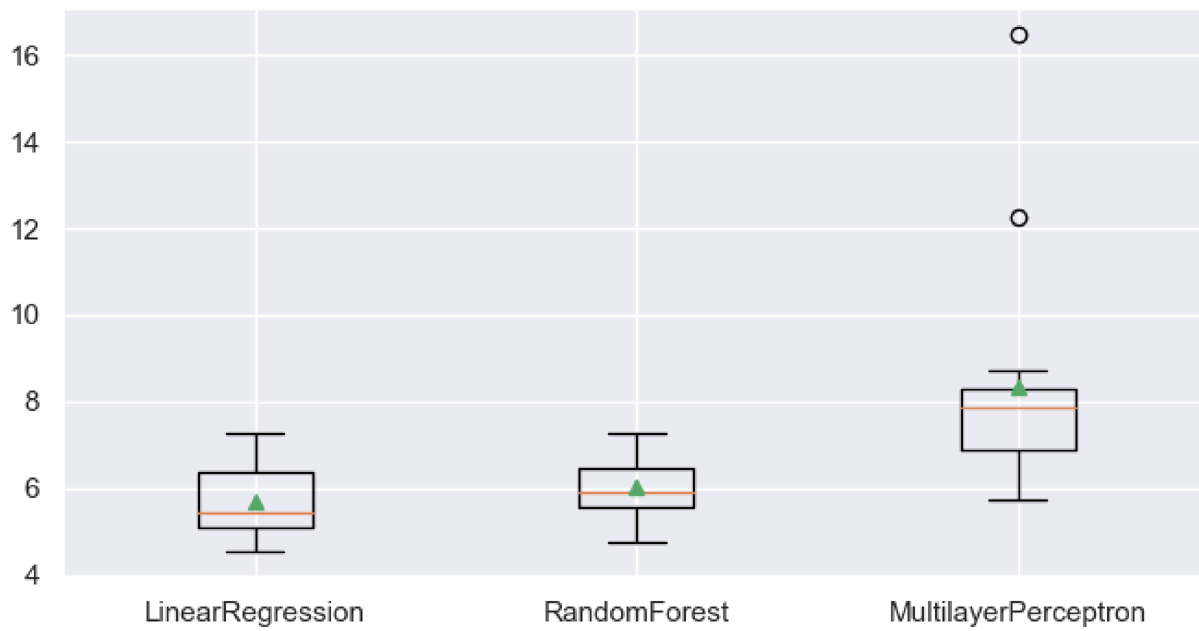
```
/Users/emrs/opt/anaconda3/lib/python3.9/site-packages/sklearn/impute/_bas
```

```
e.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is t
aken will be eliminated, and the value None will no longer be accepted. S
et `keepdims` to True or False to avoid this warning.
  mode = stats.mode(array)
/Users/emrs/opt/anaconda3/lib/python3.9/site-packages/sklearn/impute/_bas
e.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is t
aken will be eliminated, and the value None will no longer be accepted. S
et `keepdims` to True or False to avoid this warning.
  mode = stats.mode(array)
/Users/emrs/opt/anaconda3/lib/python3.9/site-packages/sklearn/impute/_bas
e.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
```

MODELO MLP

- MLP: las métricas obnetidas con los datos de validación no difieren significativamente con respecto a las mismas métricas pero en el conjunto de entrenamiento
- RF: Está sobre entrenado porque tiene un error bajo en el conjunto de entrenamiento y un error de más del doble en el conjunto de validación, es decir, no generaliza
- LR: la desviación estandar aumenta para el conjunto de validación, lo que indica que no está generalizando correctamente

In [52]:
```
sns.set(rc={'figure.figsize':(8,4)})

box = list()
for i in range(len(results)):
  temp = results[i]['test_MAPE']
  box.append(np.abs(temp))
plt.boxplot(box, labels=names, showmeans=True)

plt.show()
```



In [21]:
```
#MLP
parameters = {
    "regressor__hidden_layer_sizes": [(i,) for i in range(1, 4)],
    "regressor__alpha": np.linspace(0.0001, 0.0250, 4),
    "regressor__learning_rate_init": np.linspace(0.001, 0.250, 4),
    "regressor__activation" : ['identity', 'logistic', 'tanh', 'relu'],
    #"regressor__solver": ['lbfgs', 'sgd', 'adam']
}

mlp_grid = GridSearchCV(mlp, parameters, scoring=make_scorer(MAPE), cv=cv,
print(model.get_params().keys())
mlp_grid.fit(Xtrain, Ytrain.values.ravel())
```

```
otE__handle_unknown', 'preprocessor__cat_pipe_input__OneHotE__sparse', 'p
reprocessor__binarias__memory', 'preprocessor__binarias__steps', 'preproc
essor__binarias__verbose', 'preprocessor__binarias__impModa', 'preprocess
or__binarias__OneHotEncoder', 'preprocessor__binarias__impModa__add_indic
ator', 'preprocessor__binarias__impModa__copy', 'preprocessor__binarias__
impModa__fill_value', 'preprocessor__binarias__impModa__missing_values',
'preprocessor__binarias__impModa__strategy', 'preprocessor__binarias__imp
Moda__verbose', 'preprocessor__binarias__OneHotEncoder__categories', 'pre
processor__binarias__OneHotEncoder__drop', 'preprocessor__binarias__OneHo
tEncoder__dtype', 'preprocessor__binarias__OneHotEncoder__handle_unknow
n', 'preprocessor__binarias__OneHotEncoder__sparse', 'regressor__activati
on', 'regressor__alpha', 'regressor__batch_size', 'regressor__beta_1', 'r
egressor__beta_2', 'regressor__early_stopping', 'regressor__epsilon', 're
gressor__hidden_layer_sizes', 'regressor__learning_rate', 'regressor__lea
rning_rate_init', 'regressor__max_fun', 'regressor__max_iter', 'regressor
__momentum', 'regressor__n_iter_no_change', 'regressor__nesterovs_momentu
m', 'regressor__power_t', 'regressor__random_state', 'regressor__shuffl
e', 'regressor__solver', 'regressor__tol', 'regressor__validation_fractio
n', 'regressor__verbose', 'regressor__warm_start'])
```

In [22]:
```
mlp_grid.best_score_
```

Out[22]:
```
13.878469977784977
```

In [23]:
```
mlp_grid.best_params_
```

Out[23]:
```
{'regressor__activation': 'identity',
 'regressor__alpha': 0.0167,
 'regressor__hidden_layer_sizes': (2,),
 'regressor__learning_rate_init': 0.001}
```

In [24]:
```
#LINEAL
parameters = {
    "regressor__fit_intercept": [True, False],
    "regressor__positive": [True, False],
    "regressor__normalize": [True, False],
    "regressor__copy_X": [True, False],
}


lineal_grid = GridSearchCV(lineal, parameters, scoring=make_scorer(MAPE), c
print(model.get_params().keys())
lineal_grid.fit(Xtrain, Ytrain.values.ravel())
```

```
dict_keys(['memory', 'steps', 'verbose', 'preprocessor', 'regressor', 'pr
eprocessor__n_jobs', 'preprocessor__remainder', 'preprocessor__sparse_thr
eshold', 'preprocessor__transformer_weights', 'preprocessor__transformer
s', 'preprocessor__verbose', 'preprocessor__verbose_feature_names_out',
'preprocessor__numerical_pipeline_input', 'preprocessor__ord_pipe_input',
'preprocessor__cat_pipe_input', 'preprocessor__binarias', 'preprocessor__
numerical_pipeline_input__memory', 'preprocessor__numerical_pipeline_inpu
t__steps', 'preprocessor__numerical_pipeline_input__verbose', 'preprocess
or__numerical_pipeline_input__imputer', 'preprocessor__numerical_pipeline
_input__box-cox-scaler', 'preprocessor__numerical_pipeline_input__imputer
__add_indicator', 'preprocessor__numerical_pipeline_input__imputer__cop
y', 'preprocessor__numerical_pipeline_input__imputer__fill_value', 'prepr
ocessor__numerical_pipeline_input__imputer__missing_values', 'preprocesso
r__numerical_pipeline_input__imputer__strategy', 'preprocessor__numerical
_pipeline_input__imputer__verbose', 'preprocessor__numerical_pipeline_inp
ut__box-cox-scaler__memory', 'preprocessor__numerical_pipeline_input__box
-cox-scaler__steps', 'preprocessor__numerical_pipeline_input__box-cox-sca
ler__verbose', 'preprocessor__numerical_pipeline_input__box-cox-scaler__p
owertransformer', 'preprocessor__numerical_pipeline_input__box-cox-scaler
```

In [25]:
```
lineal_grid.best_score_
```

Out[25]:
```
6.045486642422903
```

In [26]:
```
lineal_grid.best_params_
```

Out[26]:
```
{'regressor__copy_X': True,
 'regressor__fit_intercept': True,
 'regressor__normalize': False,
 'regressor__positive': True}
```

In [27]:
```
#ARBOL
parameters = {
    "regressor__ccp_alpha": np.linspace(start=0.0, stop=1.0, num=10),
    "regressor__criterion": ["squared_error", "absolute_error", "poisson"],
    "regressor__max_depth": list(range(2,11)) + [None],
    "regressor__min_samples_split": range(2,7)
}


arbol_grid = GridSearchCV(arbol, parameters, scoring=make_scorer(MAPE), cv=
print(model.get_params().keys())
arbol_grid.fit(Xtrain, Ytrain.values.ravel())
```

```
/Users/emrs/opt/anaconda3/lib/python3.9/site-packages/sklearn/impute/_bas
e.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is t
aken will be eliminated, and the value None will no longer be accepted. S
et `keepdims` to True or False to avoid this warning.
  mode = stats.mode(array)
/Users/emrs/opt/anaconda3/lib/python3.9/site-packages/sklearn/impute/_bas
e.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is t
aken will be eliminated, and the value None will no longer be accepted. S
et `keepdims` to True or False to avoid this warning.
  mode = stats.mode(array)
/Users/emrs/opt/anaconda3/lib/python3.9/site-packages/sklearn/impute/_bas
e.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
```

In [28]:
```
arbol_grid.best_score_
```

Out[28]:
```
7.354632720025369
```

In [29]:
```
arbol_grid.best_params_
```

Out[29]:
```
{'regressor__ccp_alpha': 0.8888888888888888,
 'regressor__criterion': 'squared_error',
 'regressor__max_depth': 10,
 'regressor__min_samples_split': 3}
```

In [30]:
```
arbol.named_steps.preprocessor.transformers_[2][1]['OneHotE'].get_feature_n
```

Out[30]:
```
array(['x0_1', 'x0_2', 'x0_3', 'x1_Link', 'x1_Photo', 'x1_Status',
       'x1_Video'], dtype=object)
```

In [31]:
```
arbol.named_steps.preprocessor.transformers_[3][1]['OneHotEncoder'].get_fea
```

Out[31]:
```
array(['x0_0.0', 'x0_1.0'], dtype=object)
```

In [61]:
```
arbol.named_steps.preprocessor.transformers_
```

Out[61]:
```
[('numerical_pipeline_input',
  Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')),
                  ('box-cox-scaler',
                   Pipeline(steps=[('powertransformer',
                                    PowerTransformer(method='box-cox')),
                                   ('minmaxscaler',
                                    MinMaxScaler(feature_range=(1,
2)))]))]),
  ['Page total likes']),
 ('ord_pipe_input',
  Pipeline(steps=[('escalaNum', MinMaxScaler(feature_range=(1, 2)))]),
  ['Post Month', 'Post Weekday', 'Post Hour']),
 ('cat_pipe_input',
  Pipeline(steps=[('impModa', SimpleImputer(strategy='most_frequent')),
                  ('OneHotE', OneHotEncoder(handle_unknown='ignore'))]),
  ['Category', 'Type']),
 ('binarias',
  Pipeline(steps=[('impModa', SimpleImputer(strategy='most_frequent')),
                  ('OneHotEncoder', OneHotEncoder(handle_unknown='ignor
e'))]),
  ['Paid'])]
```

In [155]:
```
Xtrain.columns
```

Out[155]:
```
Index(['Category', 'Page total likes', 'Paid', 'Post Hour', 'Post Month',
       'Post Weekday', 'Type'],
      dtype='object')
```

In [114]:
```
linear_model.coef_
```

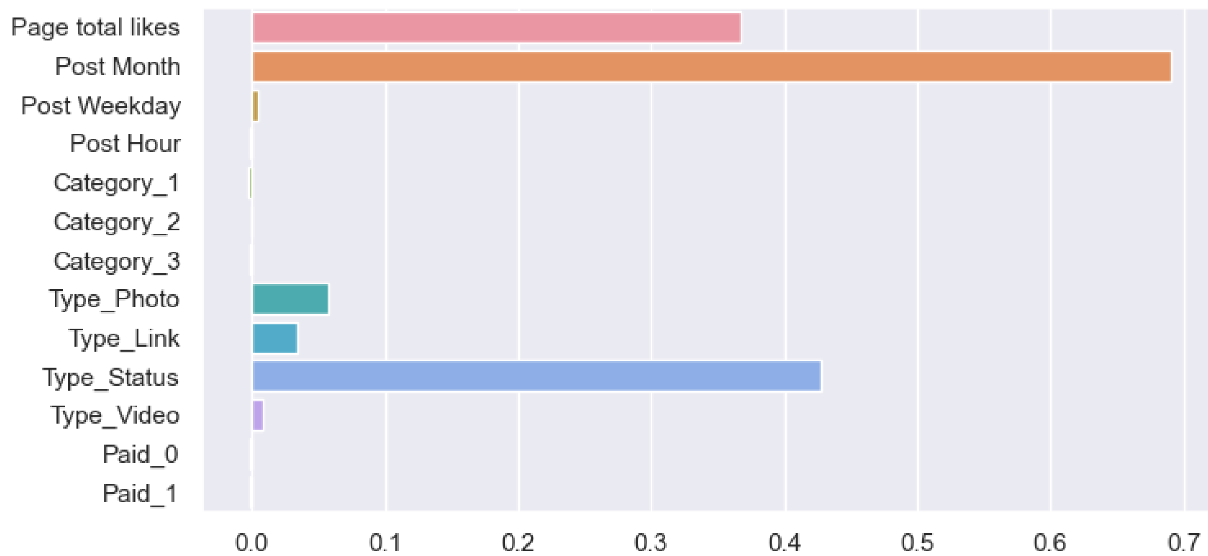
Out[114]:
```
array([ 0.21297199, -0.34452245, -0.03659269, -0.01951701,  0.00625872,
       -0.00040268, -0.0066059 , -0.2092255 , -0.03736772,  0.19451242,
        0.09955574, -0.01896932,  0.01896932])
```

Hiperparámetros LR

- Los mejores hiperparámetros fueron: copy_X= True, fit_intercept= True, normalize= True,positive= False
- Similar al artículo, las características más relevantes son 'Type_status', 'Page total likes' y 'Post Month', sin embargo, la más relevante es 'Post Month'
- A diferencia del modelo RF las características 'Category' y 'Paid' son completamente insignificantes

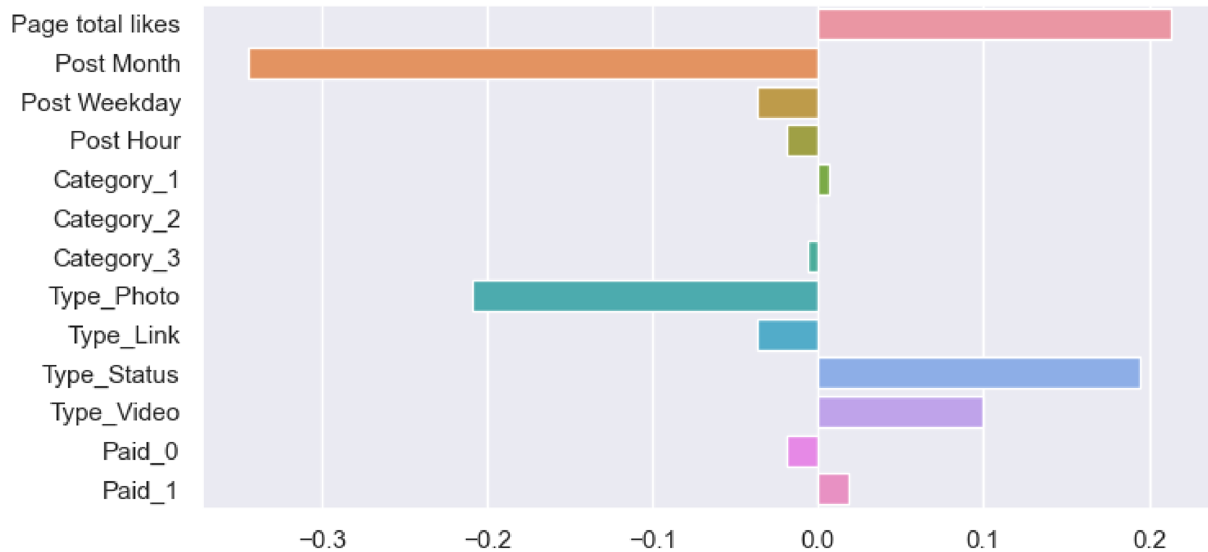
In [107]:
```
linear_model = LinearRegression(copy_X= True, fit_intercept= True, normaliz
linear_model.fit(preprocessor_pipeline.fit_transform(Xtrain), Ytrain.values

lineal_importancia = permutation_importance(linear_model,
                                            preprocessor_pipeline.fit_trans
                                            Ytest.values.ravel(), n_repeats
sns.barplot(x=lineal_importancia['importances_mean'], y=columnas_final)
```

Out[107]: <AxesSubplot:>



In [115]:
```
sns.barplot(x=linear_model.coef_, y=columnas_final) #no usar porque no está
```

Out[115]: <AxesSubplot:>

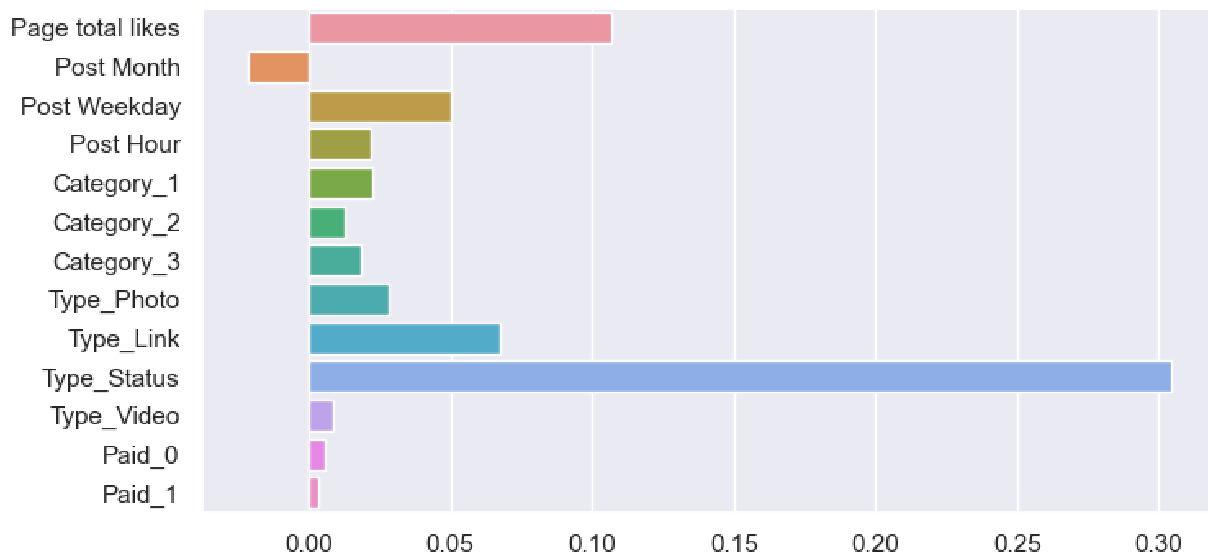


Hiperparámetros RF

- Los mejores hiperparámetros fueron: ccp_alpha=0, criterion='squared_error', max_depth=10, min_samples_split=3
- Al igual que el artículo, las características más relevantes son 'Type_status' y 'Page total likes', al igual que en la publicación de Moro-Rita-Vala, 'Type Status' es aproximadamente tres veces más importante que los otros tipos juntos
- Es interesante mostrar que dentro de la característica 'Category' no hay un valor que sobresalga entre ellas
- En general la característica de 'Category' no tiene mucho impacto cuando lo comparamos con la categoría de 'Type'
- A diferencia del modelo MLP este modelo no toma en cuenta ni a 'Paid' ni a 'Category'

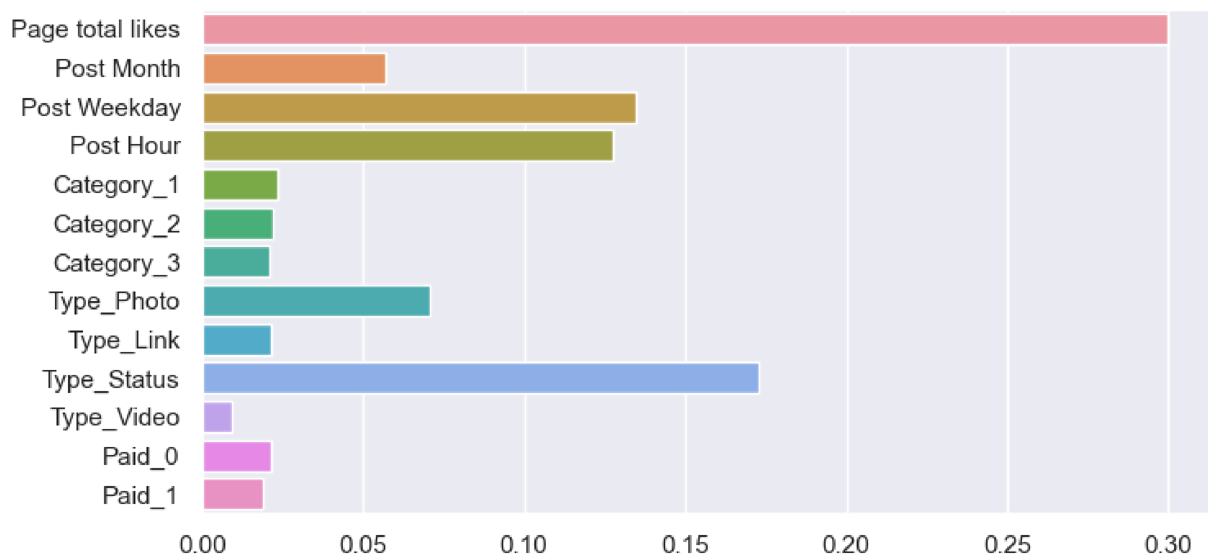
In [224]:
```
tree_model = RandomForestRegressor(ccp_alpha=0, criterion='squared_error',
tree_model= tree_model#arbol
#tree_model.fit(Xtrain, Ytrain.values.ravel());
tree_model.fit(preprocessor_pipeline.fit_transform(Xtrain), Ytrain.values.r

arbol_importancia = permutation_importance(tree_model,
                                           preprocessor_pipeline.fit_transf
                                           Ytest.values.ravel(), n_repeats=
sns.barplot(x=arbol_importancia['importances_mean'], y=columnas_final)
```

Out[224]:
```
<AxesSubplot:>
```



In [234]:
```
sns.barplot(x=tree_model.feature_importances_, y=columnas_final) #este no p
```

Out[234]: <AxesSubplot:>

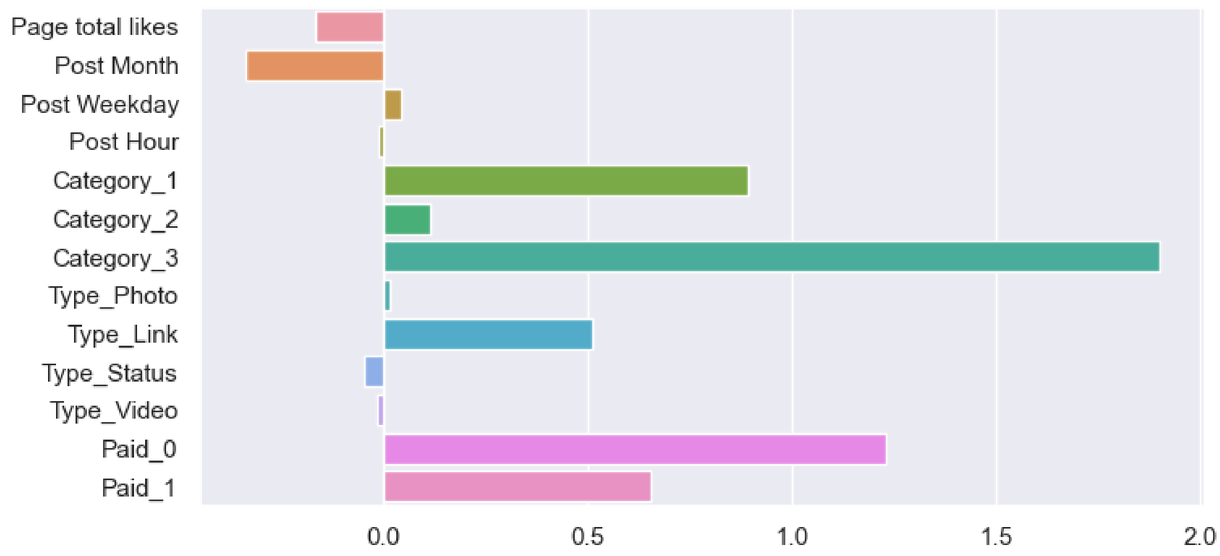


Hiperparámetros MLP

- Los mejores hiperparámetros fueron: activation= 'tanh', alpha= 0.0084, hidden_layer_sizes= (3,), learning_rate_init= 0.001
- De la importancia de los factores podemos ver que, a diferencia del artículo, que los posts que tienen una categoría de 'inspiration', que son de tipo link, que no son pagados son las tres características que tienen más peso para hacer una predicción

In [226]:
```
mlp_model = MLPRegressor(activation= 'tanh', alpha= 0.0084, hidden_layer_si
mlp_model.fit(preprocessor_pipeline.fit_transform(Xtrain), Ytrain.values.ra
#print(mlp_model.coefs_)
mlp_importancia = permutation_importance(mlp_model,
                                         preprocessor_pipeline.fit_transfor
                                         Ytest.values.ravel(), n_repeats=60
sns.barplot(x=mlp_importancia['importances_mean'], y=columnas_final)
```

Out[226]: <AxesSubplot:>



In [227]:
```
lineal_pred = linear_model.predict(preprocessor_pipeline.fit_transform(Xtes
mlp_pred = mlp_model.predict(preprocessor_pipeline.fit_transform(Xtest))
tree_pred = tm_fit.predict(Xtest)
```

In [238]:
```
MAPE(Ytest.values.ravel(), lineal_pred)
```

Out[238]: 10.449249740324966

In [239]:
```
MAPE(Ytest.values.ravel(), mlp_pred)
```

Out[239]: 14.821607192457941

In [240]:
```
MAPE(Ytest.values.ravel(), tree_pred)
```

Out[240]: 9.811595721550626

In [231]:
```
def calculate_mape(actual, predicted) -> float:

    # Convert actual and predicted
    # to numpy array data type if not already
    if not all([isinstance(actual, np.ndarray),
                isinstance(predicted, np.ndarray)]):
        actual, predicted = np.array(actual),
        np.array(predicted)

    # Calculate the MAPE value and return
    return round(np.mean(np.abs((
      actual - predicted) / actual)) * 100, 2)
```

In [232]:
```
calculate_mape(Ytest.values.ravel(), tree_pred)
```

Out[232]: 9.81

Conclusiones

- Obtuvimos tres modelos con sus respectivos hiperparámetros y MAPE's
    - Regresión Lineal: MAPE= 10.44%
    - Random Forest: MAPE= 9.81%
    - Multilayer Perceptron: MAPE= 14.82%
- Con respecto al modelo (SVM) de Moro-Rita-Vala que tiene un MAPE de 27.2% podemos concluir que nuestro peor modelo (MLP) tiene un error mucho más bajo 14.82%
- Al igual que Moro-Rita-Vala para su modelo, nuestros modelos RL y RF también tienen pesos altos para las categorías 'Page Total Likes' y 'Type' (tipo de contenido), es decir, son las características más importantes para los tres modelos, SVM, RL y RF, con esas variables independientes y con ese dataset de entrenamiento.

In [ ]: