Regresión_RandomForest

March 30, 2023

0.1 #Introducción

TFM: Aplicación de ciencia de datos en el sector de producción animal para la predicción y explicación de óptimos en ganado porcino.

Titulo: **Regresión RandomForest**

Autor: **Jose Eduardo Cámara Gómez**

---

1 Preparación y selección de variables para modelos

1.1 Importar paquetes

[1]:
```
# Importación de paquetes
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
from matplotlib.pyplot import figure

from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler

sns.set(style="darkgrid")
```

[2]:
```
from google.colab import files
# Cargamos el fichero del dataset gmd_02.csv
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>

Saving gmd_02.csv to gmd_02.csv
```

[25]:
```python
# Leemos el fichero csv con los datos
df = pd.read_csv('gmd_02.csv', sep=';')
```

[26]:
```python
df.columns
```

[26]:
```
Index(['Unnamed: 0', 'ct_codigo', 'ct_integra', 'ct_tipo', 'ct_raza',
       'ct_fase', 'IncPeso', 'DiasMedios', 'GMD', 'EntradaInicial',
       'EntradaFinal', 'NumAnimales', 'na_rega', 'PesoEntMedio',
       'PesoRecMedio', 'NumBajas', 'GPS_Longitud', 'GPS_Latitud', 'gr_codpos',
       'gr_poblacion', 'KgPiensoTotal', 'semanaEntrada', 'añoEntrada',
       'PorcHembras', 'PiensoCerdaDia', 'na_nombre2'],
      dtype='object')
```

1.2 Agrupar razas similares, para reducir categorías

[27]:
```python
# Revisar la raza si se agrupan las razas con menos ocurrencias
agrupar_razas = {93 : 93, 85 : 93, 90 : 93, 95 : 93, 94 : 93, 82 : 93, 80 : 80,␣
 ↪96 : 80, 88 : 88, 0 : 0, 23 : 0,
                 84 : 0, 66 : 0, 18 : 0, 68 : 88, 7 : 7, 89 : 7, 65 : 7, 15 :␣
 ↪15, 97 : 7, 69 : 69, 81 : 81}
df.replace({'ct_raza' : agrupar_razas}, inplace=True)
```

1.3 Expresar bajas en porcentaje de animales y ponderado por los días.

[28]:
```python
df["bajas"] = df["NumBajas"] / (df["NumAnimales"] * df["DiasMedios"])
```

1.4 Convertir tipos de variables Categóricas y Fecha

[29]:
```python
# Convertimos los tipos
df["ct_integra"] = df["ct_integra"].astype("category")
#df["ct_tipo"] = df["ct_tipo"].astype("category")
df["ct_raza"] = df["ct_raza"].astype("category")
df["ct_fase"] = df["ct_fase"].astype("category")
df['EntradaInicial']= pd.to_datetime(df['EntradaInicial'])
df['EntradaFinal']= pd.to_datetime(df['EntradaFinal'])
df["na_rega"] = df["na_rega"].astype("category")
df["NumBajas"] = df["NumBajas"].astype("int64")
df["gr_codpos"] = df["gr_codpos"].astype("category")
df["gr_poblacion"] = df["gr_poblacion"].astype("category")
df["na_nombre2"] = df["na_nombre2"].astype("category")
```

1.5 Convertir variables categóricas a usar en OneHotEncoding

[30]:
```python
# Funcion para convertir en One Hot Encoding
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    res = pd.concat([original_dataframe, dummies], axis=1)
```

```
    res = res.drop([feature_to_encode], axis=1)
    return(res)
```

1.6 Seleccionar Variables a Utilizar

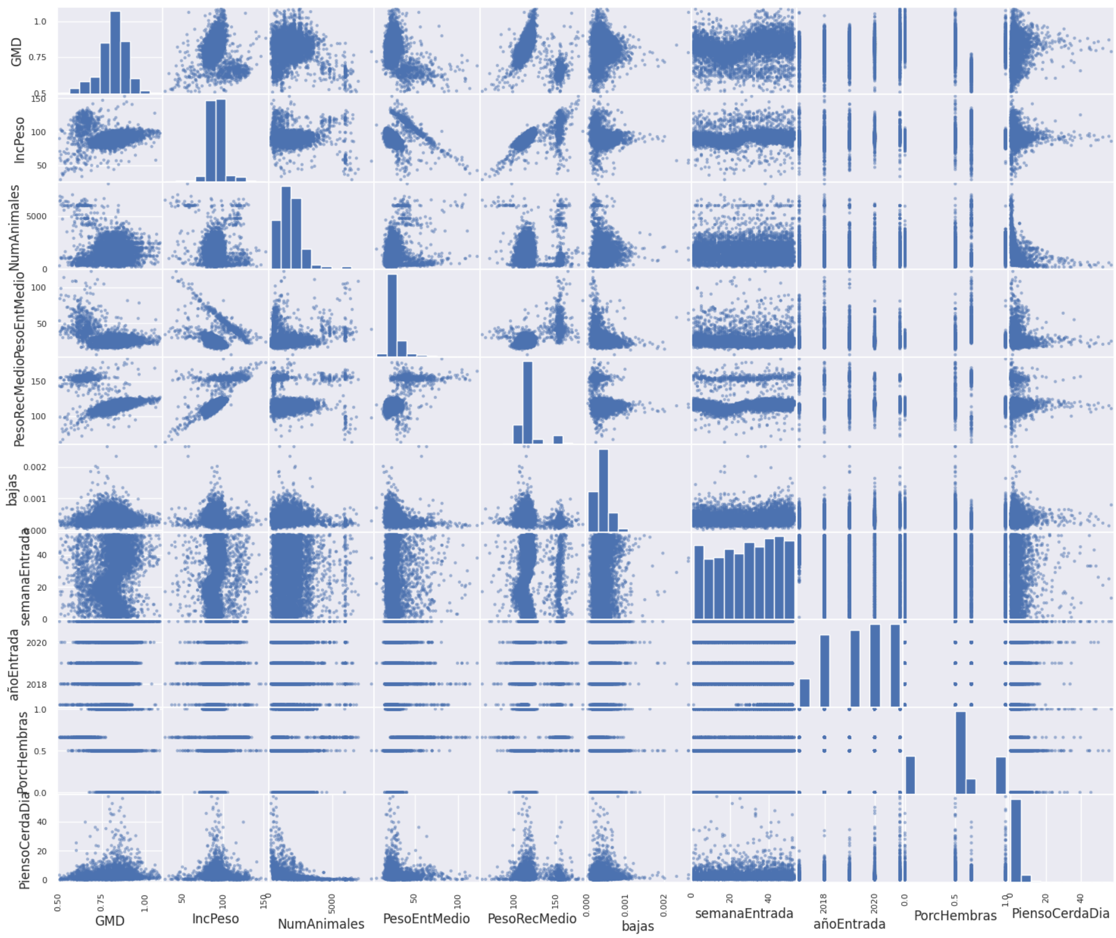
[31]:
```
# Cargamos las variables objetivo y las usadas (15 variables seleccionadas, una␣
↪de ellas categórica con 8 valores).
y = df['GMD']
x0 = df[['ct_integra','ct_tipo', 'ct_raza', 'IncPeso', 'NumAnimales', 'na_rega',
         'PesoEntMedio', 'PesoRecMedio', 'bajas', 'GPS_Longitud', 'GPS_Latitud',
         'semanaEntrada', 'añoEntrada', 'PorcHembras', 'PiensoCerdaDia']]
features_to_encode = ['ct_raza']   # , 'na_rega']
x1 = x0.copy()
x1.drop(['ct_integra','na_rega'], inplace=True, axis=1)
for feature in features_to_encode:
    x1 = encode_and_bind(x1, feature)
```

[65]:
```
x1.info()
```

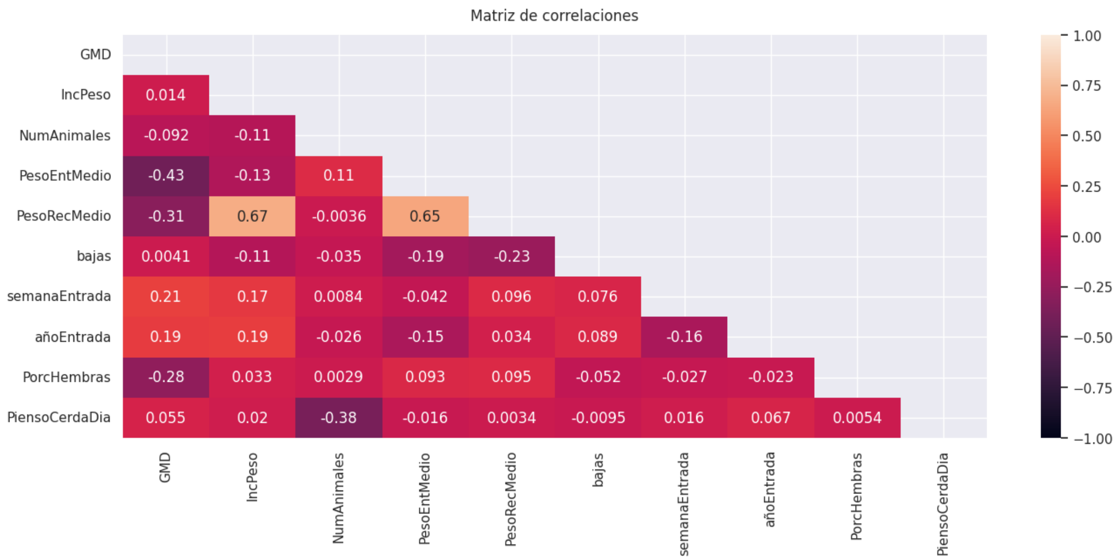
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5277 entries, 0 to 5276
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ct_tipo         5277 non-null   int64
 1   IncPeso         5277 non-null   float64
 2   NumAnimales     5277 non-null   int64
 3   PesoEntMedio    5277 non-null   float64
 4   PesoRecMedio    5277 non-null   float64
 5   bajas           5277 non-null   float64
 6   GPS_Longitud    5277 non-null   float64
 7   GPS_Latitud     5277 non-null   float64
 8   semanaEntrada   5277 non-null   int64
 9   añoEntrada      5277 non-null   int64
 10  PorcHembras     5277 non-null   float64
 11  PiensoCerdaDia  5277 non-null   float64
 12  ct_raza_0       5277 non-null   uint8
 13  ct_raza_7       5277 non-null   uint8
 14  ct_raza_15      5277 non-null   uint8
 15  ct_raza_69      5277 non-null   uint8
 16  ct_raza_80      5277 non-null   uint8
 17  ct_raza_81      5277 non-null   uint8
 18  ct_raza_88      5277 non-null   uint8
 19  ct_raza_93      5277 non-null   uint8
dtypes: float64(8), int64(4), uint8(8)
memory usage: 536.1 KB
```

1.7 Comprobar si hay variables dependientes

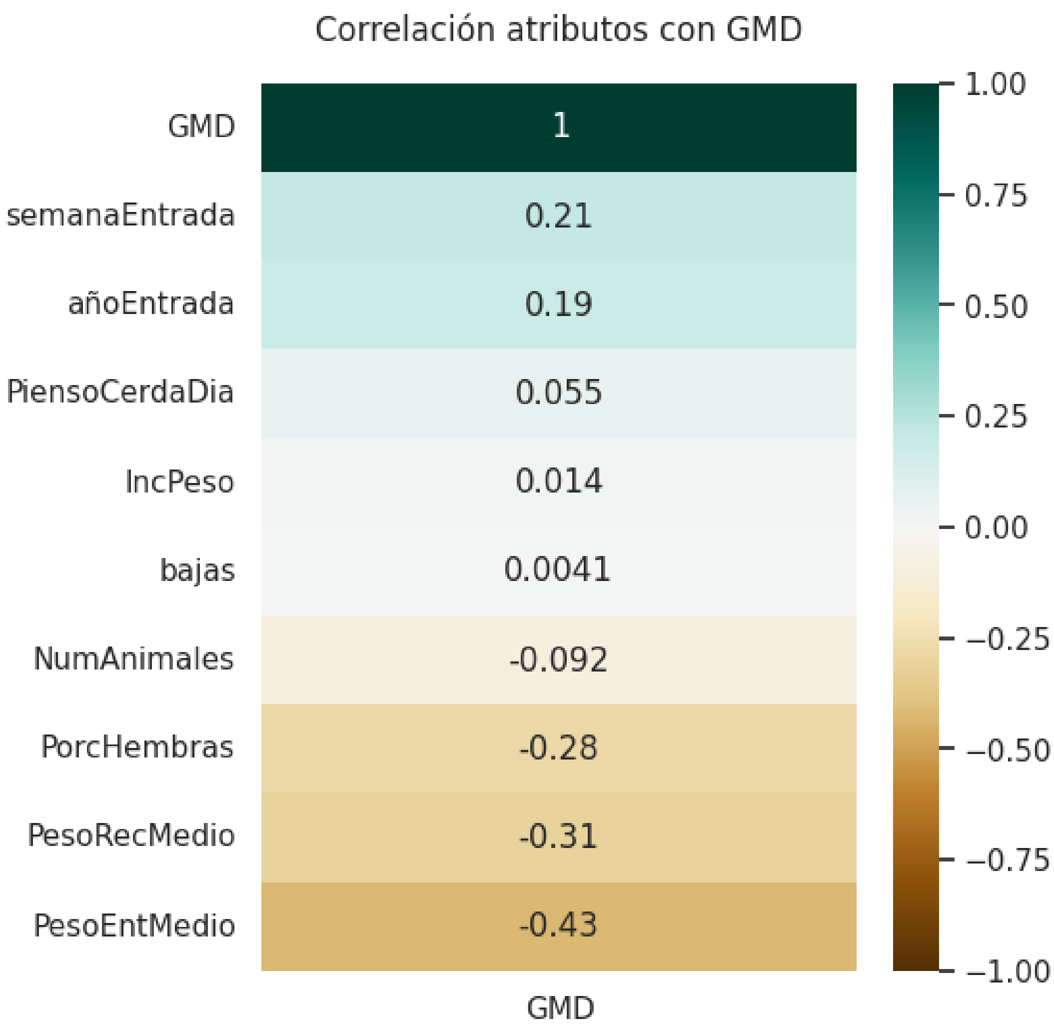
[10]:
```
attributes = ['GMD', 'IncPeso', 'NumAnimales', 'PesoEntMedio', 'PesoRecMedio',␣
 ↪'bajas', 'semanaEntrada', 'añoEntrada', 'PorcHembras', 'PiensoCerdaDia']
pd.plotting.scatter_matrix(df[attributes], figsize=(18, 15))
plt.show()
```



[11]:
```
# Mostramos la matriz de correlaciones entre las principales variables.
plt.figure(figsize=(16, 6))
mask = np.triu(np.ones_like(df[attributes].corr(), dtype=bool))
heatmap = sns.heatmap(df[attributes].corr(), mask=mask, vmin=-1, vmax=1,␣
 ↪annot=True)
heatmap.set_title('Matriz de correlaciones', fontdict={'fontsize':12}, pad=12);
plt.show()
```



[12]:
```
# Mostramos ordenadas las correlaciones de las variables con la variable␣
 ↪Objetivo GMD.
plt.figure(figsize=(5, 6))
heatmap = sns.heatmap(df[attributes].corr()[['GMD']].sort_values(by='GMD',␣
 ↪ascending=False), vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Correlación atributos con GMD', fontdict={'fontsize':12},␣
 ↪pad=16);
```



1.8 Dividir datos en entrenamiento y test

[50]:
```
X_train, X_test, y_train, y_test = train_test_split(x1, y, test_size = 0.2,␣
 ↪random_state = 123)
```

1.9 Escalar datos

[51]:
```
## Vemos de escalar las variables para que no se vean influenciadas por los␣
 ↪outliers.
scaler = RobustScaler()
scaler.fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)
```

[52]:
```
# Mostramos las columnas usadas para entrenamiento y sus tipos.
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4221 entries, 3073 to 3582
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ct_tipo         4221 non-null   int64
 1   IncPeso         4221 non-null   float64
 2   NumAnimales     4221 non-null   int64
 3   PesoEntMedio    4221 non-null   float64
 4   PesoRecMedio    4221 non-null   float64
 5   bajas           4221 non-null   float64
 6   GPS_Longitud    4221 non-null   float64
 7   GPS_Latitud     4221 non-null   float64
 8   semanaEntrada   4221 non-null   int64
 9   añoEntrada      4221 non-null   int64
 10  PorcHembras     4221 non-null   float64
 11  PiensoCerdaDia  4221 non-null   float64
 12  ct_raza_0       4221 non-null   uint8
 13  ct_raza_7       4221 non-null   uint8
 14  ct_raza_15      4221 non-null   uint8
 15  ct_raza_69      4221 non-null   uint8
 16  ct_raza_80      4221 non-null   uint8
 17  ct_raza_81      4221 non-null   uint8
 18  ct_raza_88      4221 non-null   uint8
 19  ct_raza_93      4221 non-null   uint8
dtypes: float64(8), int64(4), uint8(8)
memory usage: 461.7 KB
```

2 Modelo RandomForest

2.1 Creación del modelo

[53]:
```
rf = RandomForestRegressor(
            n_estimators = 100,
            criterion    = 'squared_error',
            max_depth    = None,
            max_features = 'sqrt',
            oob_score    = False,
            n_jobs       = -1,
            random_state = 123
         )
rf.fit(X_train_s, y_train)
```

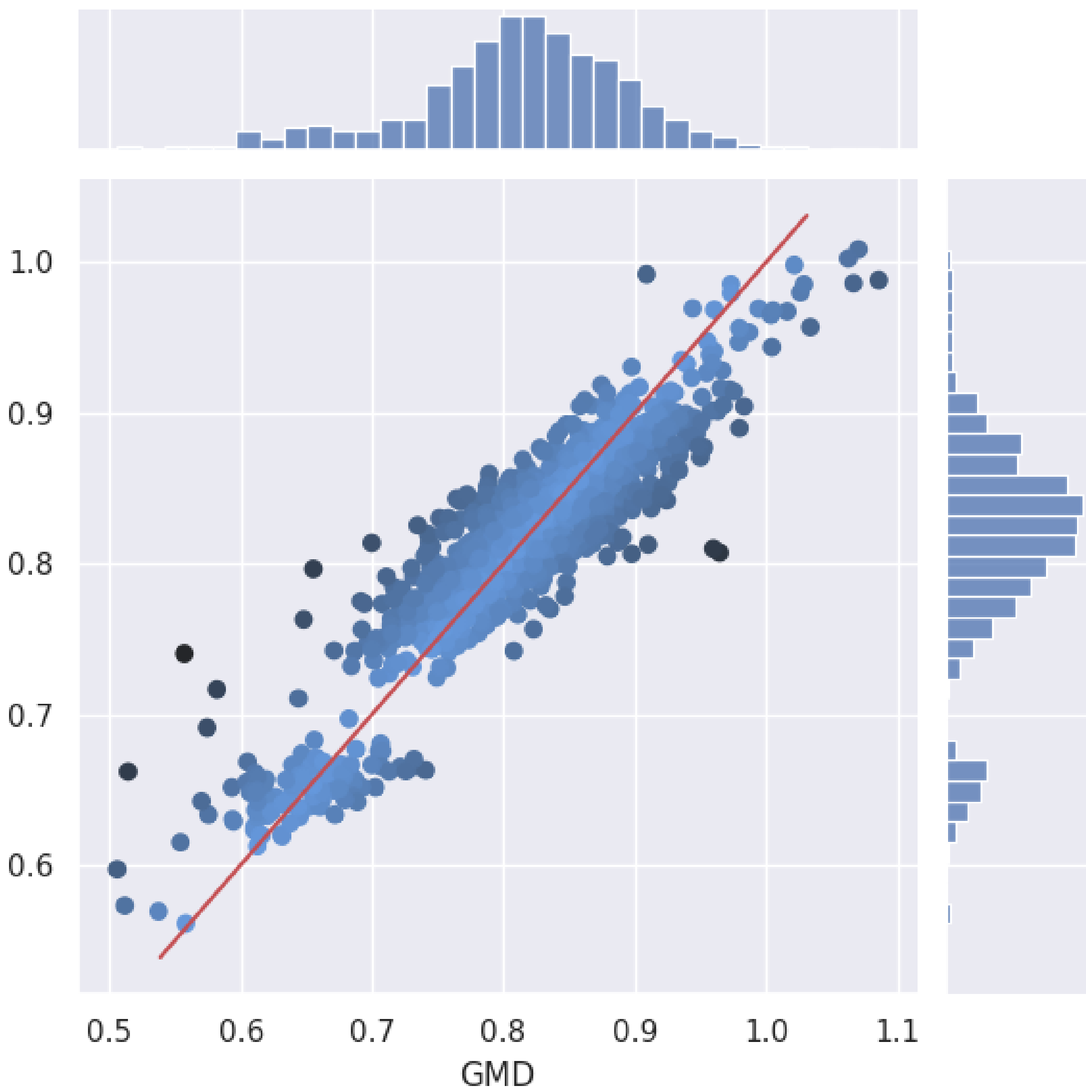
[53]:
```
RandomForestRegressor(max_features='sqrt', n_jobs=-1, random_state=123)
```

2.2 Ver diferencias entre valor real y predicho en Test

[54]:
```
# Función para Graficar diferencias entre valor predicho y real en datos de␣
↪test del modelo pasado
def graficoDiferencias(modelo, X_test_s, y_test):
    y_pred = modelo.predict(X_test_s)
    diferencia = abs(y_pred - y_test)
    g = sns.jointplot(x=y_test, y=y_pred)
    # Draw a line of x=y
    x0, x1 = g.ax_joint.get_xlim()
    y0, y1 = g.ax_joint.get_ylim()
    lims = [max(x0, y0), min(x1, y1)]
    g.ax_joint.plot(lims, lims, '-r')
    g.ax_joint.scatter(x=y_test, y=y_pred, c=diferencia.values, cmap=sns.
↪dark_palette("#69d", reverse=True, as_cmap=True))
    plt.show()
```

[55]:
```
# Graficar las diferencias
print('Score R2:',rf.score(X_test_s, y_test))
graficoDiferencias(rf, X_test_s, y_test)
```

```
Score R2: 0.8332896022063712
```



[56]:
```
# Analizamos otros errores del método
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import max_error
y_pred = rf.predict(X_test_s)
# Definimos la función con las métricas a mostrar
def mostrar_metricas(y_test, y_pred):
   print("Metr.\t Valor\t\t\t Descripción")
   print("R^2 \t", r2_score(y_test, y_pred), "\t (Coeficiente de␣
 ↪Determinación)")
   print("RMSE\t", mean_squared_error(y_test, y_pred, squared=True), "\t (Raíz␣
 ↪de error cuadrático medio)")
```

```
    print("MAE \t", mean_absolute_error(y_test, y_pred), "\t (Error absoluto␣
 ↪medio)")
    print("MAX \t", max_error(y_test, y_pred), "\t (Error Máximo)")
# Pedimos que muestre las métricas para el modelo de RandomForest
print("Métricas para RandomForest v1")
mostrar_metricas(y_test, y_pred)
```

```
Métricas para RandomForest v1
Metr.    Valor                    Descripción
R^2      0.8332896022063712       (Coeficiente de Determinación)
RMSE     0.0011945882025570303    (Raíz de error cuadrático medio)
MAE      0.026452136729027687     (Error absoluto medio)
MAX      0.1832065150160347       (Error Máximo)
```

2.3 Variables más importantes según modelo

[57]:
```
# Mostrar las variables más importantes
important_features_dict = {}
for idx, val in enumerate(rf.feature_importances_):
    important_features_dict[idx] = val

important_features_list = sorted(important_features_dict,␣
 ↪key=important_features_dict.get, reverse=True)
print(f'Las 10 características más relevantes para la regresión son:')
print('\tOrden\tCaracterística\tImportancia')
for i in range(10):
    print('\t', i+1, '\t', x1.columns[important_features_list[i]], '\t',␣
 ↪important_features_dict.get(important_features_list[i]))
```

```
Las 10 características más relevantes para la regresión son:
        Orden   Característica  Importancia
         1       PesoRecMedio     0.22822026064823203
         2       ct_raza_69       0.1153962404420927
         3       ct_tipo          0.11438670034793824
         4       PesoEntMedio     0.08694802628728203
         5       IncPeso          0.07274022404653528
         6       PorcHembras      0.06325723671377499
         7       semanaEntrada    0.05678781772447855
         8       NumAnimales      0.04124358743077506
         9       ct_raza_93       0.040755746297112934
         10      bajas   0.04064345282866445
```

2.4 Optimización de Hiperparámetros

Intentamos ver hasta dónde se pueden optimizar los hiperparámetros haciendo uso de una búsqueda aleatoria entre una gran variadead de valores de esos hiperparámetros.

2.4.1 Definir hiperparámetros a optimizar y con qué posibles valores

[58]:
```
# Hago una optimización de los hiperparámetros para RandomForest
from sklearn.model_selection import RandomizedSearchCV
random_grid = {'bootstrap': [True, False],
               'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110,␣
 ↪None],
               'max_features': [1.0, 'sqrt', 2, 5, 10, 20],
               'min_samples_leaf': [1, 2, 4, 10],
               'min_samples_split': [2, 5, 10, 20],
               'n_estimators': [20, 50, 75, 100, 150, 250, 500, 750]}
```

2.4.2 Lanzar búsqueda de mejores parámetros

[59]:
```
rf_random = RandomizedSearchCV(scoring="neg_mean_squared_error", estimator = rf,
                               param_distributions = random_grid, n_iter = 300,
                               cv = 3, verbose=1, random_state=123, n_jobs = -1)
rf_random.fit(X_train_s, y_train)
```

```
Fitting 3 folds for each of 300 candidates, totalling 900 fits
```

[59]:
```
RandomizedSearchCV(cv=3,
                   estimator=RandomForestRegressor(max_features='sqrt',
                                                   n_jobs=-1,
                                                   random_state=123),
                   n_iter=300, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': [1.0, 'sqrt', 2, 5, 10,
                                                         20],
                                        'min_samples_leaf': [1, 2, 4, 10],
                                        'min_samples_split': [2, 5, 10, 20],
                                        'n_estimators': [20, 50, 75, 100, 150,
                                                         250, 500, 750]},
                   random_state=123, scoring='neg_mean_squared_error',
                   verbose=1)
```

2.4.3 Analizar mejor modelo y su error

[60]:
```
rf_random.best_params_
```

[60]:
```
{'n_estimators': 750,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 5,
 'max_depth': 60,
```

```
 'bootstrap': False}
```

Tras probar aleatoreamente entre 300 combinaciones del rango de hiperparámetros propuesto la mejor solución para optimizar el error cuadrático medio ha sido la que se muestra. La búsqueda de los mejores hiperparámetros tardó en Google Colab 31 minutos, probando las 300 combinaciones para 3 particiones de los datos cada una, usando Cross Validation.

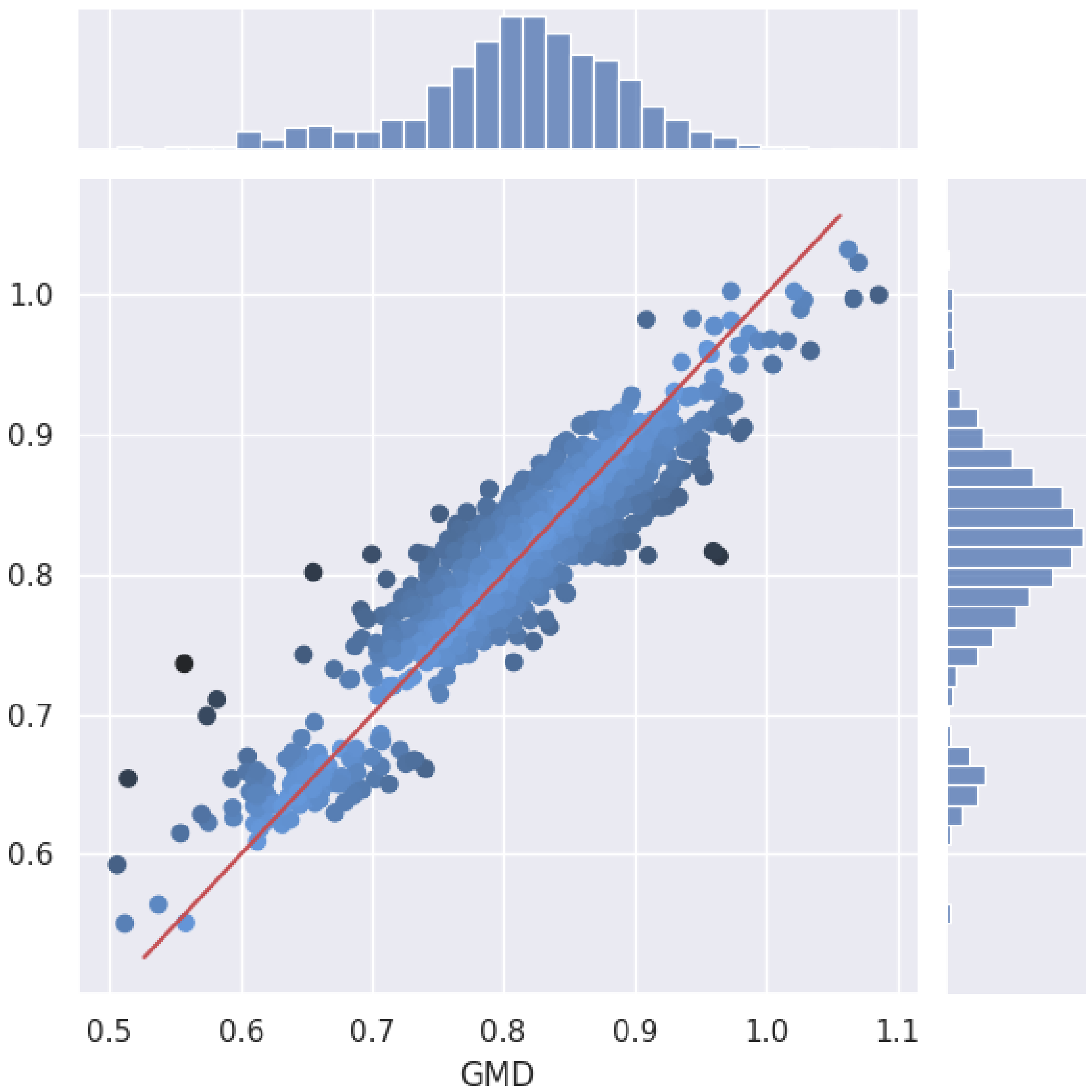
[61]:
```
# Medimos las diferencias de la predicción para los valores de test (no usados␣
 ↪en entrenamiento)
y_pred = rf_random.best_estimator_.predict(X_test_s)
mostrar_metricas(y_test, y_pred)
```

```
Metr.    Valor                    Descripción
R^2      0.8412042580961172       (Coeficiente de Determinación)
RMSE     0.0011378745561479255    (Raíz de error cuadrático medio)
MAE      0.025939472844693622     (Error absoluto medio)
MAX      0.17919904826057953      (Error Máximo)
```

El error obtenido es de tan sólo -0.001137…

[62]:
```
print('Score R2:',rf_random.best_estimator_.score(X_test_s, y_test))
graficoDiferencias(rf_random.best_estimator_, X_test_s, y_test)
```

```
Score R2: 0.8412042580961172
```



Hemos obtenido un coeficiente de determinación de 84,12%, mejorando el de los parámetros iniciales en un 0,8%.

Si medimos las diferencias sobre el conjunto de datos de test (no usado en el entrenamiento), el error se mantiene similar al obtenido en la validación cruzada del modelo, por lo que parece que no está sobreajustado y generaliza correctamente.

3 Estimar mejores modelos con LazzyPredict

La librería lazzypredict permite estimar los modelos que mejor representan nuestro modelo según una métrica dada, probando en más de 40 modelos y ofreciendo un ranking de los resultados. No ofrecen los mejores hiperparámetros para cada uno de esos modelos, pero es un buen punto de partida, para seleccionar los modelos más prometedores y realizar sobre ellos la optimización de hiperparámetros, con la que encontrar más rápidamente un buen modelo que se aproxime a la mejor solución disponible con los métodos y variables actuales.

[63]:
```
!pip install lazypredict
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting lazypredict
  Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages
(from lazypredict) (1.4.4)
Requirement already satisfied: xgboost in /usr/local/lib/python3.9/dist-packages
(from lazypredict) (1.7.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages
(from lazypredict) (4.65.0)
Requirement already satisfied: lightgbm in /usr/local/lib/python3.9/dist-
packages (from lazypredict) (3.3.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.9/dist-
packages (from lazypredict) (1.2.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.9/dist-packages
(from lazypredict) (1.1.1)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages
(from lazypredict) (8.1.3)
Requirement already satisfied: wheel in /usr/local/lib/python3.9/dist-packages
(from lightgbm->lazypredict) (0.40.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages
(from lightgbm->lazypredict) (1.10.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages
(from lightgbm->lazypredict) (1.22.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from scikit-learn->lazypredict) (3.1.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-
packages (from pandas->lazypredict) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.9/dist-packages (from pandas->lazypredict) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-
packages (from python-dateutil>=2.8.1->pandas->lazypredict) (1.16.0)
Installing collected packages: lazypredict
Successfully installed lazypredict-0.2.12
```

[64]:
```
import lazypredict
from lazypredict.Supervised import LazyRegressor
# Borramos el modelo que tarda mucho
del lazypredict.Supervised.REGRESSORS[29:32]    # PassiveAggressiveRegressor,␣
 ↪PoissonRegressor, QuantileRegressor
reg = LazyRegressor(verbose=1, ignore_warnings=False, custom_metric=None)
models, predictions = reg.fit(X_train_s, X_test_s, y_train, y_test)
```

```python
print(models)
```

```
  3%|          | 1/39 [00:00<00:17,  2.16it/s]
```

{'Model': 'AdaBoostRegressor', 'R-Squared': 0.7294843224535585, 'Adjusted R-Squared': 0.7242569663657046, 'RMSE': 0.04402749602429004, 'Time taken': 0.4621419906616211}

```
 10%|          | 4/39 [00:00<00:07,  4.97it/s]
```

{'Model': 'BaggingRegressor', 'R-Squared': 0.8087985942459139, 'Adjusted R-Squared': 0.8051038810912456, 'RMSE': 0.037014621034570525, 'Time taken': 0.3705098628997803}
{'Model': 'BayesianRidge', 'R-Squared': 0.6678693280291408, 'Adjusted R-Squared': 0.6614513440297038, 'RMSE': 0.04878454501228551, 'Time taken': 0.0506894588470459}
{'Model': 'DecisionTreeRegressor', 'R-Squared': 0.5652880485358663, 'Adjusted R-Squared': 0.5568878175896994, 'RMSE': 0.05581212484199173, 'Time taken': 0.10013699531555176}
{'Model': 'DummyRegressor', 'R-Squared': -0.00023142543513321456, 'Adjusted R-Squared': -0.01955956892180244, 'RMSE': 0.08465995131903602, 'Time taken': 0.015442609786987305}
{'Model': 'ElasticNet', 'R-Squared': -0.00023142543513321456, 'Adjusted R-Squared': -0.01955956892180244, 'RMSE': 0.08465995131903602, 'Time taken': 0.018407106399536133}

```
 18%|          | 7/39 [00:01<00:03,  8.00it/s]
```

{'Model': 'ElasticNetCV', 'R-Squared': 0.6681781248918917, 'Adjusted R-Squared': 0.66176610798159, 'RMSE': 0.048761861142817164, 'Time taken': 0.16571950912475586}
{'Model': 'ExtraTreeRegressor', 'R-Squared': 0.5951761055935756, 'Adjusted R-Squared': 0.587353421643693, 'RMSE': 0.053859316296242195, 'Time taken': 0.03992414474487305}

```
 26%|          | 10/39 [00:03<00:11,  2.61it/s]
```

{'Model': 'ExtraTreesRegressor', 'R-Squared': 0.8457726097022569, 'Adjusted R-Squared': 0.8427923702762135, 'RMSE': 0.0332436362606322, 'Time taken': 1.9740560054779053}
{'Model': 'GammaRegressor', 'R-Squared': 0.5607752921566995, 'Adjusted R-Squared': 0.552287858188713, 'RMSE': 0.05610107046939275, 'Time taken': 0.1643831729888916}

```
 28%|          | 11/39 [00:12<01:04,  2.31s/it]
```

{'Model': 'GaussianProcessRegressor', 'R-Squared': -5.754981835711436, 'Adjusted R-Squared': -5.885512885676875, 'RMSE': 0.2200087024019368, 'Time taken': 8.994166851043701}

```
 31%|          | 12/39 [00:14<01:01,  2.29s/it]
```

```
{'Model': 'GradientBoostingRegressor', 'R-Squared': 0.8165102364870722,
'Adjusted R-Squared': 0.8129645405737789, 'RMSE': 0.036260491564244754, 'Time
taken': 2.229362964630127}

 33%|        | 13/39 [00:20<01:25,  3.29s/it]

{'Model': 'HistGradientBoostingRegressor', 'R-Squared': 0.8476511539323192,
'Adjusted R-Squared': 0.8447072148778711, 'RMSE': 0.033040556338584305, 'Time
taken': 6.170478820800781}

 36%|        | 14/39 [00:21<01:02,  2.49s/it]

{'Model': 'HuberRegressor', 'R-Squared': 0.6128868887288887, 'Adjusted
R-Squared': 0.6054064421342779, 'RMSE': 0.05266798549079409, 'Time taken':
0.3008115291595459}

 38%|        | 15/39 [00:21<00:46,  1.93s/it]

{'Model': 'KNeighborsRegressor', 'R-Squared': 0.7774162315974172, 'Adjusted
R-Squared': 0.7731150959761113, 'RMSE': 0.03993691484379072, 'Time taken':
0.42801809310913086}

 46%|       | 18/39 [00:24<00:25,  1.24s/it]

{'Model': 'KernelRidge', 'R-Squared': -91.41536249120558, 'Adjusted R-Squared':
-93.20116659731585, 'RMSE': 0.8137665843939424, 'Time taken':
2.8581631183624268}
{'Model': 'Lars', 'R-Squared': 0.6679207205257995, 'Adjusted R-Squared':
0.6615037296180855, 'RMSE': 0.048780770509096766, 'Time taken':
0.025605201721191406}
{'Model': 'LarsCV', 'R-Squared': 0.6681936306928815, 'Adjusted R-Squared':
0.6617819134115845, 'RMSE': 0.04876072182590559, 'Time taken':
0.07918930053710938}
{'Model': 'Lasso', 'R-Squared': -0.00023142543513321456, 'Adjusted R-Squared':
-0.01955956892180244, 'RMSE': 0.08465995131903602, 'Time taken':
0.029938936233520508}

 51%|       | 20/39 [00:24<00:15,  1.22it/s]

{'Model': 'LassoCV', 'R-Squared': 0.6681910306152943, 'Adjusted R-Squared':
0.661779263090952, 'RMSE': 0.048760912873189954, 'Time taken':
0.28230786323547363}
{'Model': 'LassoLars', 'R-Squared': -0.00023142543513321456, 'Adjusted
R-Squared': -0.01955956892180244, 'RMSE': 0.08465995131903602, 'Time taken':
0.07087373733520508}

 62%|      | 24/39 [00:25<00:06,  2.49it/s]

{'Model': 'LassoLarsCV', 'R-Squared': 0.6681936306928815, 'Adjusted R-Squared':
0.6617819134115845, 'RMSE': 0.04876072182590559, 'Time taken':
0.16783523559570312}
{'Model': 'LassoLarsIC', 'R-Squared': 0.6683368877931932, 'Adjusted R-Squared':
0.6619279387650423, 'RMSE': 0.04875019449068347, 'Time taken':
0.07441520690917969}
```

```
{'Model': 'LinearRegression', 'R-Squared': 0.6663880702676847, 'Adjusted
R-Squared': 0.6599414629298621, 'RMSE': 0.04889321021693962, 'Time taken':
0.06666707992553711}
```

```
 64%|      | 25/39 [00:25<00:06,  2.29it/s]
```

```
{'Model': 'LinearSVR', 'R-Squared': 0.5688952733890773, 'Adjusted R-Squared':
0.5605647472709918, 'RMSE': 0.05558007891373986, 'Time taken':
0.5810821056365967}
```

```
 67%|      | 26/39 [00:28<00:11,  1.18it/s]
```

```
{'Model': 'MLPRegressor', 'R-Squared': 0.6629604534566966, 'Adjusted R-Squared':
0.6564476119775989, 'RMSE': 0.0491437392191756, 'Time taken': 2.312641143798828}
```

```
 69%|      | 27/39 [00:31<00:17,  1.43s/it]
```

```
{'Model': 'NuSVR', 'R-Squared': 0.8224590191486887, 'Adjusted R-Squared':
0.819028275557359, 'RMSE': 0.035667861720169364, 'Time taken':
3.309314489364624}
{'Model': 'OrthogonalMatchingPursuit', 'R-Squared': 0.5294344783992626,
'Adjusted R-Squared': 0.5203414248417604, 'RMSE': 0.05806812705734184, 'Time
taken': 0.013873577117919922}
{'Model': 'OrthogonalMatchingPursuitCV', 'R-Squared': 0.6209229536716963,
'Adjusted R-Squared': 0.6135977933561734, 'RMSE': 0.05211845234256676, 'Time
taken': 0.022565841674804688}
```

```
 77%|      | 30/39 [00:31<00:06,  1.34it/s]
```

```
{'Model': 'RANSACRegressor', 'R-Squared': -5.611708736322594e+21, 'Adjusted
R-Squared': -5.720147552483417e+21, 'RMSE': 6341256605.834454, 'Time taken':
0.17681217193603516}
```

```
 90%|      | 35/39 [00:35<00:02,  1.51it/s]
```

```
{'Model': 'RandomForestRegressor', 'R-Squared': 0.8322351812395536, 'Adjusted
R-Squared': 0.8289933489929749, 'RMSE': 0.03467194562106037, 'Time taken':
3.8039724826812744}
{'Model': 'Ridge', 'R-Squared': 0.6679289247918978, 'Adjusted R-Squared':
0.6615120924207267, 'RMSE': 0.04878016792272903, 'Time taken':
0.014435291290283203}
{'Model': 'RidgeCV', 'R-Squared': 0.667928924792042, 'Adjusted R-Squared':
0.6615120924208737, 'RMSE': 0.048780167922718436, 'Time taken':
0.0195159912109375}
{'Model': 'SGDRegressor', 'R-Squared': 0.6680895261414477, 'Adjusted R-Squared':
0.6616757971779974, 'RMSE': 0.04876837058650609, 'Time taken':
0.020760536193847656}
{'Model': 'SVR', 'R-Squared': 0.7056216216914125, 'Adjusted R-Squared':
0.6999331506129857, 'RMSE': 0.04592833705970492, 'Time taken':
0.13870549201965332}
{'Model': 'TransformedTargetRegressor', 'R-Squared': 0.6663880702676847,
'Adjusted R-Squared': 0.6599414629298621, 'RMSE': 0.04889321021693962, 'Time
taken': 0.02387833595275879}
```

```
{'Model': 'TweedieRegressor', 'R-Squared': 0.5589925924906035, 'Adjusted
R-Squared': 0.5504707102198905, 'RMSE': 0.05621480506031914, 'Time taken':
0.026605606079101562}

100%|      | 39/39 [00:36<00:00,  1.06it/s]

{'Model': 'XGBRegressor', 'R-Squared': 0.8400534872446372, 'Adjusted R-Squared':
0.8369627333749684, 'RMSE': 0.03385440261176636, 'Time taken':
0.8272502422332764}
{'Model': 'LGBMRegressor', 'R-Squared': 0.8476133664338951, 'Adjusted
R-Squared': 0.8446686971862408, 'RMSE': 0.033044653654034827, 'Time taken':
0.1853652000427246}


                                      Adjusted R-Squared  \
Model
HistGradientBoostingRegressor                       0.84
LGBMRegressor                                       0.84
ExtraTreesRegressor                                 0.84
XGBRegressor                                        0.84
RandomForestRegressor                               0.83
NuSVR                                               0.82
GradientBoostingRegressor                           0.81
BaggingRegressor                                    0.81
KNeighborsRegressor                                 0.77
AdaBoostRegressor                                   0.72
SVR                                                 0.70
LassoLarsIC                                         0.66
LassoLarsCV                                         0.66
LarsCV                                              0.66
LassoCV                                             0.66
ElasticNetCV                                        0.66
SGDRegressor                                        0.66
RidgeCV                                             0.66
Ridge                                               0.66
Lars                                                0.66
BayesianRidge                                       0.66
LinearRegression                                    0.66
TransformedTargetRegressor                          0.66
MLPRegressor                                        0.66
OrthogonalMatchingPursuitCV                         0.61
HuberRegressor                                      0.61
ExtraTreeRegressor                                  0.59
LinearSVR                                           0.56
DecisionTreeRegressor                               0.56
GammaRegressor                                      0.55
TweedieRegressor                                    0.55
OrthogonalMatchingPursuit                           0.52
DummyRegressor                                     -0.02
```

```
LassoLars                                      -0.02
ElasticNet                                     -0.02
Lasso                                          -0.02
GaussianProcessRegressor                       -5.89
KernelRidge                                   -93.20
RANSACRegressor          -572014755248341726032.00

                                           R-Squared          RMSE  \
Model
HistGradientBoostingRegressor                   0.85          0.03
LGBMRegressor                                   0.85          0.03
ExtraTreesRegressor                             0.85          0.03
XGBRegressor                                    0.84          0.03
RandomForestRegressor                           0.83          0.03
NuSVR                                           0.82          0.04
GradientBoostingRegressor                       0.82          0.04
BaggingRegressor                                0.81          0.04
KNeighborsRegressor                             0.78          0.04
AdaBoostRegressor                               0.73          0.04
SVR                                             0.71          0.05
LassoLarsIC                                     0.67          0.05
LassoLarsCV                                     0.67          0.05
LarsCV                                          0.67          0.05
LassoCV                                         0.67          0.05
ElasticNetCV                                    0.67          0.05
SGDRegressor                                    0.67          0.05
RidgeCV                                         0.67          0.05
Ridge                                           0.67          0.05
Lars                                            0.67          0.05
BayesianRidge                                   0.67          0.05
LinearRegression                                0.67          0.05
TransformedTargetRegressor                      0.67          0.05
MLPRegressor                                    0.66          0.05
OrthogonalMatchingPursuitCV                     0.62          0.05
HuberRegressor                                  0.61          0.05
ExtraTreeRegressor                              0.60          0.05
LinearSVR                                       0.57          0.06
DecisionTreeRegressor                           0.57          0.06
GammaRegressor                                  0.56          0.06
TweedieRegressor                                0.56          0.06
OrthogonalMatchingPursuit                       0.53          0.06
DummyRegressor                                 -0.00          0.08
LassoLars                                      -0.00          0.08
ElasticNet                                     -0.00          0.08
Lasso                                          -0.00          0.08
GaussianProcessRegressor                       -5.75          0.22
KernelRidge                                   -91.42          0.81
RANSACRegressor          -5611708736322594144256.00 6341256605.83
```

	Time Taken
Model	
HistGradientBoostingRegressor	6.17
LGBMRegressor	0.19
ExtraTreesRegressor	1.97
XGBRegressor	0.83
RandomForestRegressor	3.80
NuSVR	3.31
GradientBoostingRegressor	2.23
BaggingRegressor	0.37
KNeighborsRegressor	0.43
AdaBoostRegressor	0.46
SVR	0.14
LassoLarsIC	0.07
LassoLarsCV	0.17
LarsCV	0.08
LassoCV	0.28
ElasticNetCV	0.17
SGDRegressor	0.02
RidgeCV	0.02
Ridge	0.01
Lars	0.03
BayesianRidge	0.05
LinearRegression	0.07
TransformedTargetRegressor	0.02
MLPRegressor	2.31
OrthogonalMatchingPursuitCV	0.02
HuberRegressor	0.30
ExtraTreeRegressor	0.04
LinearSVR	0.58
DecisionTreeRegressor	0.10
GammaRegressor	0.16
TweedieRegressor	0.03
OrthogonalMatchingPursuit	0.01
DummyRegressor	0.02
LassoLars	0.07
ElasticNet	0.02
Lasso	0.03
GaussianProcessRegressor	8.99
KernelRidge	2.86
RANSACRegressor	0.18