U03 PRÁCTICA 1

MÁS ALGORITMOS DE REGRESIÓN

SISTEMAS DE APRENDIZAJE AUTOMÁTICO

IES SERRA PERENXISA
TORRENT (VALENCIA)

REGRESIÓN CON SVM, CART, BAGGING y BOOST

📌ACTIVIDAD 1: REPASAR ALGORITMOS SVM.

REGRESIÓN LINEAL CON SVM Y UNA SOLA PREDICTORA

Crea el notebook ***saa_u03_p01_a5-<tus_iniciales>.ipynb*** *donde entregar esta actividad.* Utiliza pandas para cargar los datos del fichero ***"50_startups.csv"*** *(puedes utilizar una copia del fichero* ***u02_p03_a1_<tus_iniciales>.ipynb****)*. Utilizaremos como predictora la columna ***"I&D Spend"*** (que significa gasto en I+D) y como target usaremos ***"Profit"*** (beneficios) tal y como hicimos en una práctica anterior.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import pandas as pd
4  from sklearn.preprocessing import StandardScaler
5
6  dataset = pd.read_csv('50_startups.csv')
7  X = dataset.iloc[:, 0].values.reshape(-1,1)          # gasto en I+D
8  y = dataset.iloc[:, -1].values.reshape(-1,1)         # beneficios
```

En todos los procesos aleatorios utiliza una misma semilla, en mi caso usaré ***"123"***. Divide los datos en *train* y *test* dejando el 70% para entrenamiento. Una vez particionados en ***X_train***, ***y_train***, ***X_test***, ***y_test*** crea un ***sklearn.preprocessing.StandardScaler()*** y lo entrenas con ***X_train*** para normalizar ***X_train*** y ***X_test***. También vamos a escalar los ***y_train*** e ***y_test*** con su propio objeto escalador. Este código no te lo paso, debes hacerlo tu mismo. Los parámetros de los objetos escaladores aparecen abajo, no tienen que coincidir porque dependen de los datos que se usen (elección aleatoria) pero si deben ser razonablemente parecidos.

```
Divido en train y test
X_train y X_test escalados con media [72119.13371429] y desviación [45221.19432743]
y_train e y_test escalados con media [109950.90428571] y desviación [39402.92056968]
```

Crea un objeto ***sklearn.Linear.LinearSVR()*** con hiperparámetro ***epsilon*** de ***0.5*** y tu semilla aleatoria y lo llamas ***svm***. Luego lo entrenas. Para graficar la ***SVM*** escribe estas dos funciones, una que calcula los vectores soporte y otra que dibuja los datos *train* y el modelo.

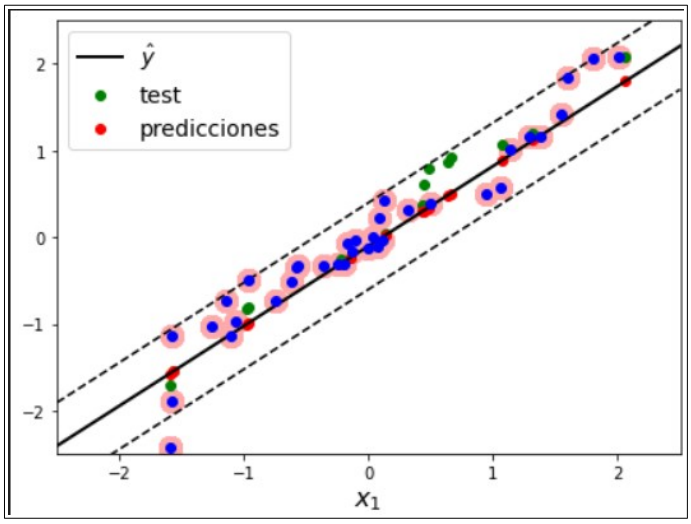
```
5  def calcula_vectores_soporte(svm, X, y):
6      predicciones = svm.predict(X)
7      fuera_del_margen = (np.abs(y - predicciones) >= svm.epsilon)
8      return np.argwhere(fuera_del_margen)
9
10 svm.soporte_ = calcula_vectores_soporte(svm, X_train, y_train)
```

```
12 def plot_svm_regresion(svm, X, y, intervalo_ejes):
13     x1s = np.linspace(intervalo_ejes[0], intervalo_ejes[1], 100).reshape(100, 1)
14     y_pred = svm.predict(x1s)
15     plt.plot(x1s, y_pred, "k-", linewidth=2, label=r"$\hat{y}$")
16     plt.plot(x1s, y_pred + svm.epsilon, "k--")
17     plt.plot(x1s, y_pred - svm.epsilon, "k--")
18     plt.scatter(X[svm.soporte_], y[svm.soporte_], s=180, facecolors='#FFAAAA')
19     plt.plot(X, y, "bo")
20     plt.xlabel(r"$x_1$", fontsize=16)
21     plt.axis(intervalo_ejes)
```

Representa la ***SVM*** entrenada en un gráfico en el intervalo **[-2.5, 2.5]** de *X* y de Y.

```
fig, ejes = plt.subplots(ncols=1, figsize=(7, 5), sharey=True)
plot_svm_regresion(svm=svm, X=X_train, y=y_train, intervalo_ejes=[-2.5, 2.5, -2.5, 2.5])
```

Añade otro gráfico ***scatterplot()*** de los datos de test en color verde con la etiqueta ***"test"***. Haz otro gráfico ***scatterplot()*** de los datos predichos para ***X_test*** en color rojo con la etiqueta ***"predicciones"*** (deben caer sobre la línea principal de la *SVM*).



Por último, muestra los coeficientes y el punto de corte del modelo y su ***score*** (coeficiente de determinación R^2) para datos de *train* y *test*.

```
SVM Coeficientes: [0.92291196]
SVM Corte: [-0.10148506]
R2 train: 0.9202145225816121
R2 test:  0.9467713600753255
```

ENTREGA 1: Muestra:

a) Código y capturas de ejecución.

b) ¿Si ***SVR(kernel="linear")*** es equivalente a ***LinearSVR()*** qué diferencia hay entre ambos?

c) Puesto que es necesario escalar los datos para usar las *máquinas de vector soporte*, es más cómodo utilizar un *pipeline* que nos permita realizar las dos operaciones unificadas. Crea un *pipeline* con ***sklearn.pipeline.make_pipeline()*** que integre un objeto que normalice los datos y un ***LinearSVR()*** con *epsilon* 0.5 y tu semilla aleatoria, lo entrenas y vuelves a calcular el *score* sobre los datos de test ¿Coincide el *score* con el *regresor* anterior?

CUANDO Y COMO USAR CADA POSIBLE REGRESOR SVM

La siguiente tabla muestra un resumen de las diferentes características de varios regresores basados en ***SVM***:

Regresor	¿Cuándo usarlo?	Características clave
SVR	Datos no lineales, flexibilidad con *epsilon*	Soporta *kernel*, útil si < 10000 datos aprox.
NuSVR	Como ***SVR***, controla complejidad con *nu*	Controla cuántos puntos son vectores de soporte
LinearSVR	Datos lineales, eficiente con muchos datos	Más rápido, sin kernels, para muchos datos

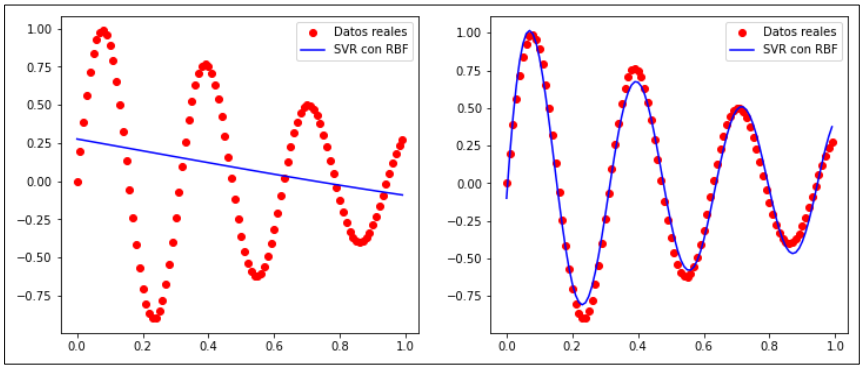
Vamos a probar resultados con datos sintéticos que simulan una función no lineal.

```
import numpy as np
import matplotlib.pyplot as plt

X = np.arange(0, 1, 0.01).reshape(-1,1)
y = np.sin(20*X) / (1 + 2 * X*X)
plt.scatter(X,y)
```

En primer lugar usa un modelo ***SVR*** con un ***kernel="rbf"***, ***C=1*** y ***gamma=0.1***. **Vamos a jugar a ajustar lo máximo que podamos el modelo a los datos de entrenamiento.** Debes crear una figura de 1 fila y 2 columnas, y entrenar primero el modelo original, calcular el MSE y generar el gráfico de como predice. Luego cambia los parámetros y haz lo mismo dibujando el gráfico de la derecha. Recuera que:

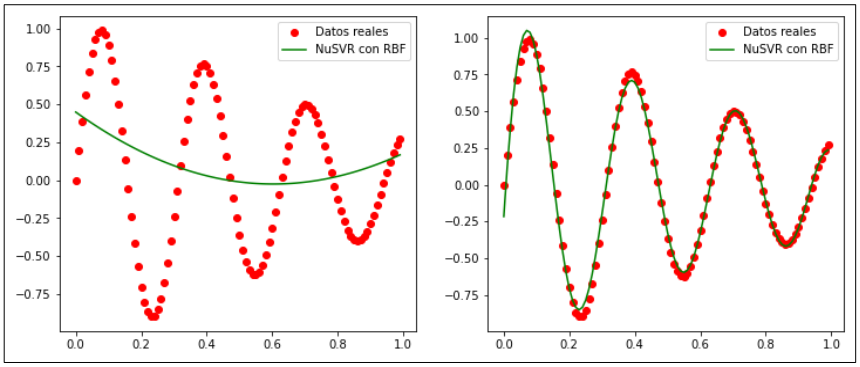
- **C controla el margen (aumentas si *undefitting* y bajas si *overfitting*):**
  - **C pequeño:** permite más errores a cambio de más **suavidad** en la predicción (evita sobreajuste).
  - **C grande:** penaliza más los errores → intenta ajustarse mejor a los datos.
- **epsilon** es la tolerancia al error, define un margen dentro del cual el error no se penaliza.
  - **ε pequeño (por ejemplo 0.01)** predicción más precisa, intenta ajustarse a los datos.
  - **ε grande (1.0 o más)** más tolerancia al error, no se ajusta tanto a los datos y reduce sensibilidad a datos ruidosos (generaliza mejor).
- **gamma define cuanta influencia tienen los puntos individuales en la forma del kernel RBF:**
  - **gamma pequeño (0.01 por ejemplo)** cada punto afecta a una gran región → suave y general.
  - **gamma grande (10 o más):** el modelo se adapta a detalles locales.

```
MSE: 0.240892 %Error: 12.775622%
MSE: 0.005066 %Error: 0.268667%
```

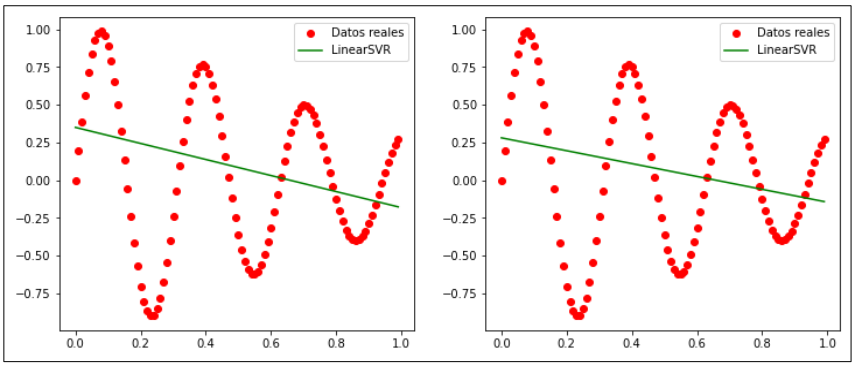


Ahora vamos a realizar lo mismo con ***NuSVR***, que es similar a ***SVR*** pero controlas la complejidad del modelo con el hiperparámetro un que indica el porcentaje de puntos que queremos usar como vectores de soporte. Comienza con ***kernel="rbf"***, ***C=100***, ***gamma=0.1*** y ***nu=0.4***. Entrena y visualiza su predicción. Luego cambia primero un, luego el resto de hiperparámetros hasta conseguir ajustar mucho los datos.

```
MSE: 0.233818 %Error: 12.400478%
MSE: 0.001932 %Error: 0.102484%
```



Ahora lo mismo con ***LinearSVR***, que es más rápido aunque solo sirve si los datos son ajustables linealmente porque no usa kernels. Comienza con ***C=1***, ***epsilon=0.5***. Entrena y visualiza su predicción. Luego cambia los hiperparámetros hasta conseguir ajustar mucho los datos.



ENTREGA 2: Muestra:

a) Código y capturas de ejecución con gráficos previo y posterior y cálculos de MSE.

DETECCIÓN DE ANOMALÍAS CON REGRESORES

Otro uso de los modelos consiste en utilizarlos para detectar outliers. En el caso de las ***SVM*** tenemos a ***OneClassSVM***, que es un clasificador y tenemos un modelo lineal regresor como ***RANSACRegressor***. Veamos un ejemplo de uso:

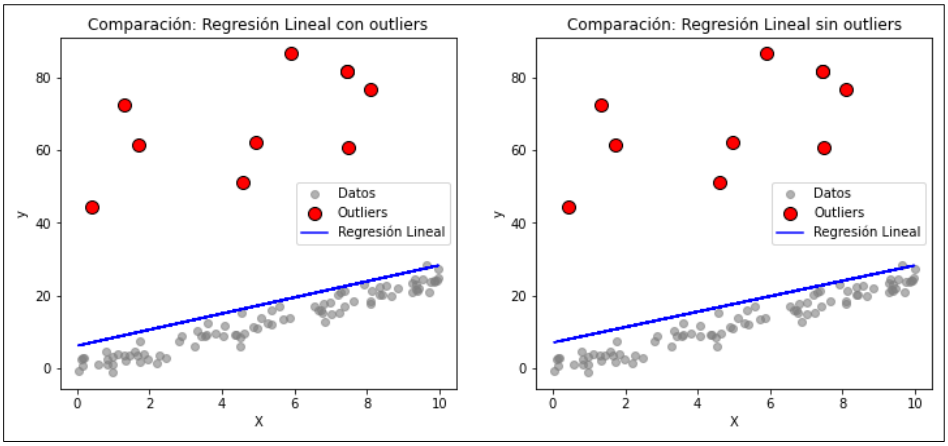
```
import numpy as np
# Generar datos de regresión con ruido
np.random.seed(449)                          # cambia por tu semilla!!
X = np.random.rand(100, 1) * 10              # Característica entre 0 y 10
y = 2.5 * X.flatten() + np.random.randn(100) * 2  # Línea distorsionada con ruido de amplitud 2
# Introducir outliers en los datos
outliers = np.random.randint(0, 100, 10)           # Seleccionamos 10 índices aleatorios
y[outliers] += np.random.randint(20, 50, size=10)  # Aumentamos el valor de esos puntos
```

Ahora entrenamos un modelo lineal usando todos los datos (incluidos outliers), e imprimimos su configuración y generamos un gráfico de datos, outliers y predicciones que hace.

```
from sklearn.linear_model import LinearRegression
# Ajustar modelo de regresión lineal (con los outliers)
lr = LinearRegression()
lr.fit(X, y)
y_pred_lr = lr.predict(X)
print(f"Recta={lr.intercept_} + {lr.coef_}X")
plt.figure(figsize=(12, 5))
plt.subplot(1,2,1)
plt.scatter(X, y, color="gray", alpha=0.6, label="Datos")
plt.scatter(X[outliers], y[outliers], color="red", label="Outliers", edgecolor="black", s=100)
plt.plot(X, y_pred_lr, color="blue", label="Regresión Lineal")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.title("Comparación: Regresión Lineal con outliers")

from sklearn.svm import OneClassSVM
from sklearn.preprocessing import StandardScaler
```

Completa el código de arriba para que usando ***OneClassSVM***: detecte los *outliers*. Cuando entrenes el modelo y le pidas predicciones, clasificará los datos no *outliers* como clase 1. Por tanto puedes crear una máscara con los datos que son normales ***mascara=oc_svm.predict(X_escalada)*** y usando la máscara puedes quedarte con los datos sin *outliers*: ***X[mascara]*** e ***y[mascara]*** con los que entrenar de nuevo un modelo lineal y mostrar su configuración y predicciones:

```
Recta con outliers=6.228892105186555 + [2.21060338]X
Recta sin outliers=7.107250767393081 + [2.12057461]X
```



El modelo ***RANSAC (Random Sample Consensus)*** es útil cuando tienes datos con ***outliers*** y queremos entrenar un modelo de regresión que sea **robusto** frente a ellos con el objeto ***RANSACRegressor*** que tiene estos hiperparámetros:

- ***estimator=LinearRegression()***: Modelo de regresión a usar dentro de ***RANSAC***.
- ***max_trials=100:*** Número máximo de intentos para encontrar un modelo sin *outliers*.
- ***residual_threshold=10***: Límite para considerar un punto como atípico o no.

Puedes hacer algo parecido a lo que has hecho con ***OneClassSVM***, crear una máscara y usarla para quedarte solo con los datos normales. La forma de hacerlo seria: ***mascara = ransac.inlier_mask_***. El resultado:

ENTREGA 3: Muestra:

a) Capturas de ejecución y el código donde aparezcan fórmulas del modelo y gráficos de predicciones del modelo de regresión lineal con *outliers* y una vez eliminados tras detectarlos con ***OneClassSVM***.

b) Capturas de ejecución y el código donde aparezcan fórmulas del modelo y gráficos de predicciones del modelo de regresión lineal con *outliers* y una vez eliminados tras detectarlos con ***RANSAC***.

UTILIZACIÓN DE KERNELS

La siguiente tabla resumen los diferentes kernels que podemos usar en SVM. En la mayoría de ocasiones el que mejores resultados suele dar es rbf porque tiene la capacidad de adaptarse muy bien a datos complejos. Y en el caso de la regresión, uno de ellos suele dar malos resultados.

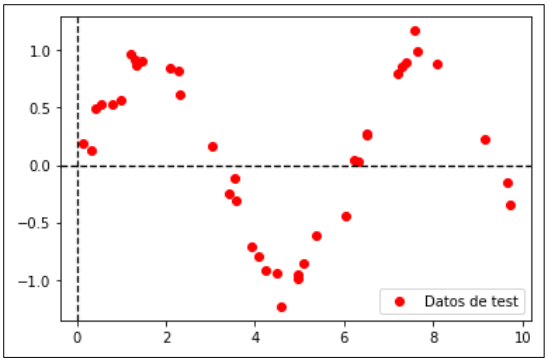
Kernel	Mejor Uso
Lineal	Cuando los datos son separables con una recta
Polinómico	Si la relación entre variables es polinómica
RBF (Gaussiano)	Cuando la separación entre clases es compleja y desconocida
Sigmoide	Para relaciones parecidas a redes neuronales

Vamos a probarlos:

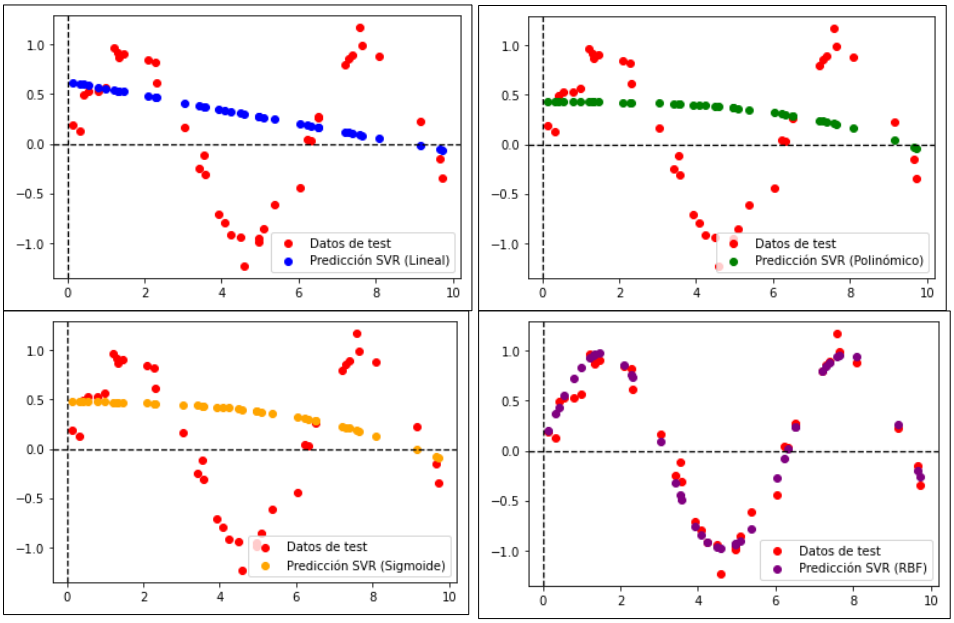
Copia este código en el *notebook* y modifica las líneas 8 y 13 para que los procesos aleatorios sean repetibles en caso de ser necesario de manera que la semilla que elijas dependa de tu nombre y apellidos (`n_letras_nombre concatenar n_letras_apellido1 concatenar n_letras_apellido2`).

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Generar datos artificiales (con ruido)
np.random.seed(449)                                  # Semilla 449 porque Jose = 4 Rosa=4 Rodríguez=9
X = np.sort(10 * np.random.rand(200, 1), axis=0)     # Entrada (200 ejemplos)
y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])  # Salida con ruido

# Dividir en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=449)

# Calcular rango de los datos para calcular la magnitud porcentual del error
rango = y.max() - y.min()

# Graficar resultados
plt.scatter(X_test, y_test, color="red", label="Datos de test")
plt.legend()
plt.axhline(0, color='black', linewidth=1.3, linestyle='--')  # Eje horizontal (y=0)
plt.axvline(0, color='black', linewidth=1.3, linestyle='--')  # Eje vertical
```

Si graficamos los datos de test nos saldrá algo parecido a esto:



Vamos a utilizar un SVR con diferentes kernel pero siempre con los hiperparámetros ***C=1*** y ***epsilon=0.1*** para ver como modeliza estos datos. Para cada cada uno generamos el gráfico scatter de los datos predichos ***(x_test, y_pred)*** en color azul y calculamos el ***MSE*** (con ***"mean_squared_error(y_test, y_pred)"*** y la magnitud porcentual del error comparada con los valores que toman los datos y. Deberías obtener gráficos similares a estos cuatro:



📌ACTIVIDAD 2: UN CASO DE REGRESIÓN MÁS REAL.

Crea el *notebook* ***saa_u03_p01_a2-<tus_iniciales>.ipynb*** donde entregar esta actividad. En el fichero *centro-comercial.csv* tenemos registrada la facturación semanal conseguida en unos 45 centros comerciales de una cadena de supermercados registrada un día concreto de cada semana. Tenemos datos de varios años y queremos tener un modelo de Machine Learning que nos permita predecir las ventas semanales a partir de otros datos que se indiquen.

> **Nota**: *este ejercicio tiene un obstáculo que debes detectar para poder resolverlo de manera correcta (a mi se me ocurren dos posibles soluciones aplicadas en la fase de preprocesamiento y en el ajuste de parámetros, aunque solamente he probado una y es posible que haya más).*

COMPARATIVA DE VARIOS REGRESORES

Carga en un ***DataFrame*** el archivo ***centro-comercial.csv*** y realiza este preprocesamiento básico: análisis exploratorio de datos y selección de características.

PRIMERA ETAPA: EDA.

- Muestra unas filas del dataset y cuantas características y ejemplos tiene.

	centro	fecha	ventas_semanales	festivo	temperatura	precio_gasolina	IPC	desempleo
0	1	05-02-2010	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	12-02-2010	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	19-02-2010	1611968.17	0	39.93	2.514	211.289143	8.106
3	1	26-02-2010	1409727.59	0	46.63	2.561	211.319643	8.106
4	1	05-03-2010	1554806.68	0	46.50	2.625	211.350143	8.106

```
Tiene 8 características y 6435 ejemplos
```

- Cambiar la característica ***fecha*** por 3 características nuevas: ***día de la semana***, ***mes*** y ***año***. Puedes convertir las fechas en tipo ***`Date`*** con la función de pandas ***`to_datetime(fecha, formato)`*** donde tendrás que indicar el formato porque utiliza por defecto el inglés. El formato es un texto con comodines donde ***`%d`*** es el día, ***`%m`*** es el mes y ***`%Y`*** es el año con 4 dígitos. Cuando la tengas convertida a tipo ***`Date`*** puedes extraerle información con expresiones como ***`pd.fecha.dt.year`*** que te devuelve el año.

	centro	ventas_semanales	festivo	temperatura	precio_gasolina	IPC	desempleo	dia	mes	año
0	1	1643690.90	0	42.31	2.572	211.096358	8.106	4	2	2010
1	1	1641957.44	1	38.51	2.548	211.242170	8.106	4	2	2010
2	1	1611968.17	0	39.93	2.514	211.289143	8.106	4	2	2010
3	1	1409727.59	0	46.63	2.561	211.319643	8.106	4	2	2010
4	1	1554806.68	0	46.50	2.625	211.350143	8.106	4	3	2010

- Ahora cuenta y muestra la cantidad de diferentes valores que tiene cada característica con ***`df.nunique()`*** para detectar posibles problemas (que haya 13 meses por ejemplo) o detectar las que quizás sean booleanas (tendrán 2 valores) y en definitiva comprender mejor los datos.

```
dia                1
festivo            2
año                3
mes               12
centro            45
desempleo        349
precio_gasolina  892
IPC             2145
temperatura     3528
ventas_semanales 6435
dtype: int64
```

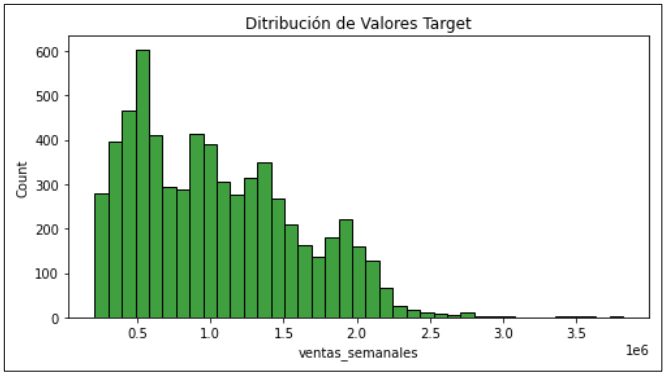
- Definimos una variable ***`target`*** que será ***`"ventas_semanales"`*** y otra de tipo lista llamada ***`predictoras`*** que contenga el resto de características menos el target.

- Intentamos detectar qué características pueden ser pueden ser categóricas y cuales pueden ser numéricas usando el criterio de la cantidad de valores diferentes que tengan. Si una característica tiene menos del 0.5% de sus valores distintos es sospechosa de ser categórica aunque sus valores sean numéricos. Completa el código que crea la lista ***`nf`*** con los nombres de las características numéricas y la lista ***`cf`*** con las categóricas y las imprima:

```
1  # Detectar categóricas por la cantidad de valores diferentes usando el 0.7%
2  nu = df[predictoras].nunique().sort_values()
3  nf = []; cf = [];                          # numéricas (nf) y categóricas (cf)
```

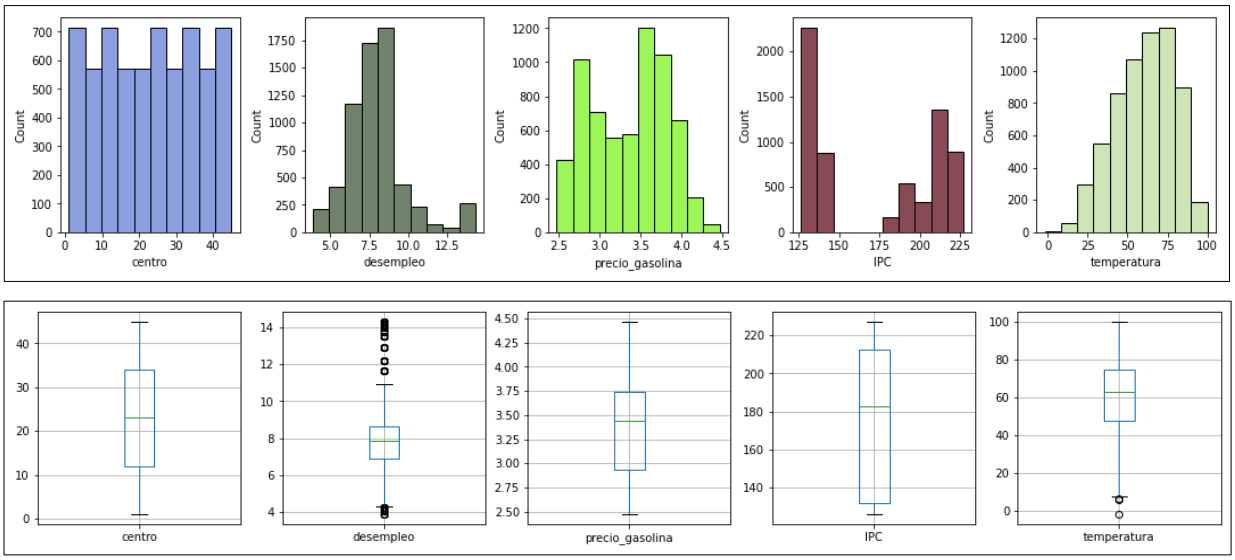
```
El Dataset tiene 5 numéricas y 4 categóricas.
Numéricas: ['centro', 'desempleo', 'precio_gasolina', 'IPC', 'temperatura']
Categóricas: ['dia', 'festivo', 'año', 'mes']
```

- Dibujar un histograma del target con ***seaborn*** por ejemplo usando ***`histplot()`***.



- Completa este código que dibuja las distribuciones de las características numéricas para que aparezca debajo del histograma de cada característica su boxplot y nos permitan visualizar la presencia de *outliers*, tal y como se ve en la figura.

```python
# Visualizar características numéricas
import numpy as np
print('Distribuciones de Características Numéricas'.center(130))
n=4
clr=['r','g','b','g','b','r']
plt.figure(figsize=[15, 6 * math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    sns.histplot(df[nf[i]], bins=10, color=list(np.random.randint([255,255,255])/255))
plt.tight_layout()
plt.show()
```

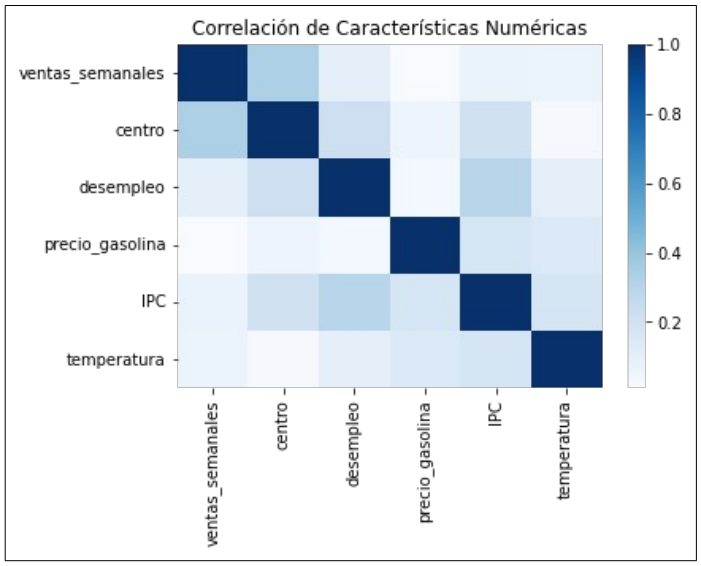


SEGUNDA ETAPA: SELECCIÓN DE CARACTERÍSTICAS.

- Elimina características del *dataframe* que no son útiles, y también debes borrarlas de las listas de predictoras y de cualquier otra lista donde aparezcan.
- Realiza operaciones de limpieza que consideres necesarias. Crea un objeto llamado ***`preprocesador`*** de tipo ***`ColumnTransformer`*** que pueda utilizarse para aplicar transformaciones cuando se necesiten:
  - Tratamiento de valores ausentes.
  - Eliminación de filas repetidas.

- Tratamiento de *outliers*.
- Codificación de variables categóricas.
- Escalado o estandarización.

- Estudia la colinealidad entre las *predictoras* y el *target* y entre cada pareja de predictoras numéricas calculando la matriz de correlaciones de las características numéricas y visualizando un mapa de calor de color azul y blanco. Elimina predictoras que presenten una correlación superior al 60%.



TERCERA ETAPA: PARTICIONAR DATOS

- Inicializa a partir de ahora todos los procesos aleatorios con una semilla aleatoria obtenida como `<letras_de_tu_nombre> concatenar <letras_apellido1> concatenar <letras_apellido2>`, por ejemplo en mi caso sería 449 (Jose = 4, Rosa=4, Rodríguez=9).
- Deja para entrenamiento el 80% de los datos y divide en train y test.

CUARTA ETAPA: ENTRENAMIENTO Y SELECCIÓN DE MODELOS

- Vamos a realizar esta etapa de forma manual. Definimos en un diccionario llamado ***`regresores`*** todos los regresores que vamos a entrenar metidos en un ***`Pipeline`*** donde primero aplicamos el preproceso a los datos y luego entrenamos/predecimos con el modelo.

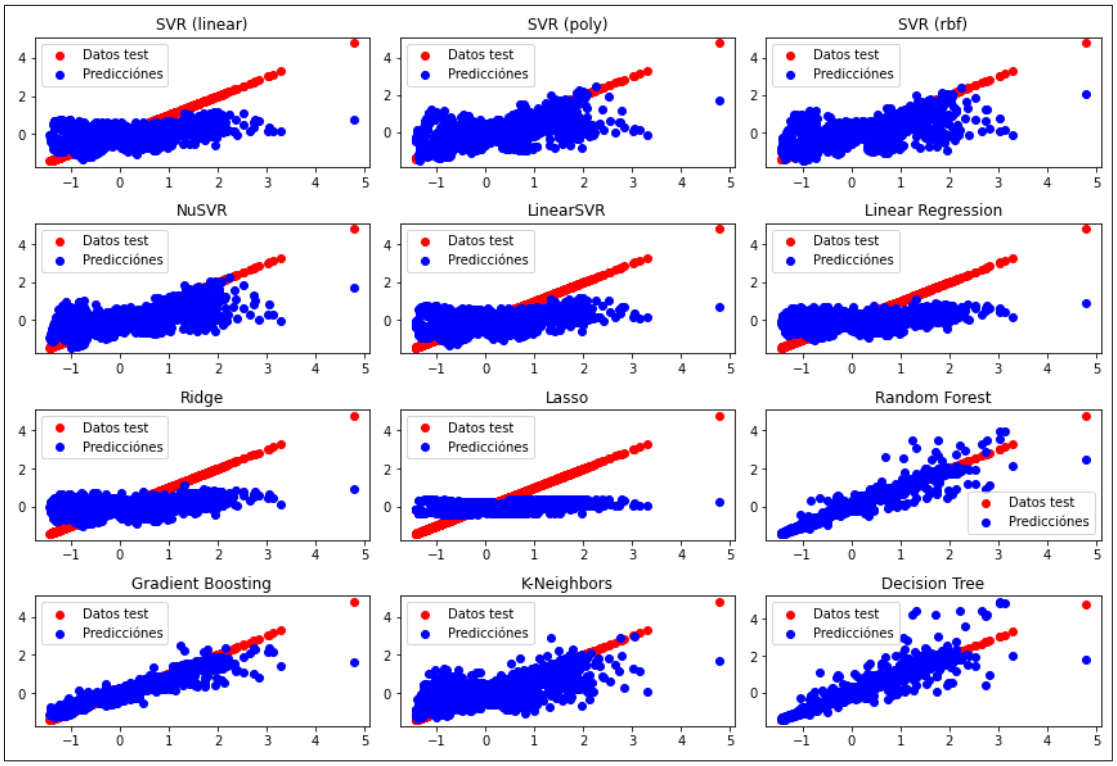
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVR, NuSVR, LinearSVR
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

# Definir los diferentes regresores
regresores = {
    'SVR (linear)': Pipeline( [('pre', preprocesador), ('regresor', SVR(kernel='linear', C=1.0, epsilon=0.1))] ),
    'SVR (poly)': Pipeline( [('pre', preprocesador), ('regresor', SVR(kernel='poly', C=1.0, epsilon=0.1))] ),
    'SVR (rbf)': Pipeline( [('pre', preprocesador), ('regresor', SVR(kernel='rbf', C=1.0, epsilon=0.1)  )] ),
    'NuSVR': Pipeline( [('pre', preprocesador), ('regresor', NuSVR(nu=0.5, kernel='rbf', C=1.0))] ),
    'LinearSVR': Pipeline( [('pre', preprocesador), ('regresor', LinearSVR(C=1.0, epsilon=0.1, random_state=449))] ),
    'Linear Regression': Pipeline( [('pre', preprocesador), ('regresor', LinearRegression() )] ),
    'Ridge': Pipeline( [('pre', preprocesador), ('regresor', Ridge(alpha=1.0) )] ),
    'Lasso': Pipeline( [('pre', preprocesador), ('regresor', Lasso(alpha=0.1) )] ),
    'Random Forest': Pipeline( [('pre', preprocesador), ('regresor', RandomForestRegressor(n_estimators=100, random_state=449))] ),
    'Gradient Boosting': Pipeline( [('pre', preprocesador), ('regresor',GradientBoostingRegressor(n_estimators=100, random_state=449) )] ),
    'K-Neighbors': Pipeline( [('pre', preprocesador), ('regresor', KNeighborsRegressor(n_neighbors=5) )] ),
    'Decision Tree': Pipeline( [('pre', preprocesador), ('regresor', DecisionTreeRegressor(random_state=449) )] )
}
```

- Ahora definimos listas (`mse_train, mse_test, r2_train, r2_test`) para registrar el error que comete cada modelo con *train* y con *test* para poder comprobar el desempeño de cada uno y detectar situaciones no deseables. Completa o adapta el código para generar salida como esta:

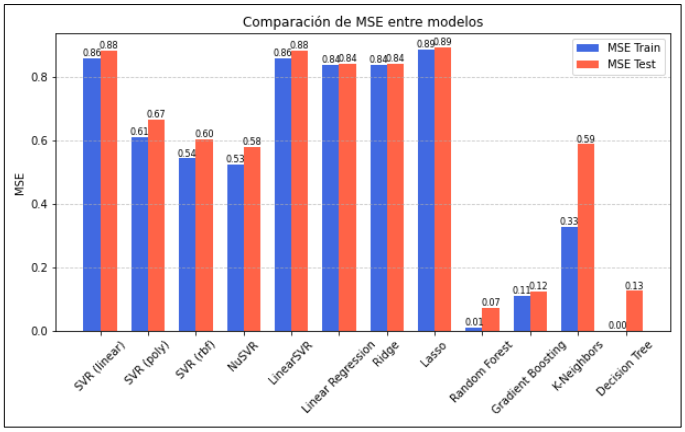
```
1  # Entrenar y evaluar cada modelo
2  from sklearn.metrics import mean_squared_error, r2_score
3  mse_train = []
4  mse_test = []
5  r2_train = []
6  r2_test = []
7  plt.figure(figsize=(12, 10))
8  for i, (nombre, modelo) in enumerate( regresores.items() ):
9      modelo.fit(X_train, y_train_escalado)   # modelo.fit(X_train, y_train) si ni el y_train ni el y_test estuviesen escalados
10     y_pred = modelo.predict(X_train).reshape(-1,1)
11     # y_pred = s_y.inverse_transform(y_pred.reshape(-1, 1)).ravel() si el y_test no estuviese escalado
```

```
Modelo: SVR (linear) - MSE: 0.883998 $R^2$: 0.109534
Modelo: SVR (poly) - MSE: 0.665140 $R^2$: 0.329994
Modelo: SVR (rbf) - MSE: 0.603999 $R^2$: 0.391582
Modelo: NuSVR - MSE: 0.580604 $R^2$: 0.415148
Modelo: LinearSVR - MSE: 0.884647 $R^2$: 0.108880
Modelo: Linear Regression - MSE: 0.842781 $R^2$: 0.151053
Modelo: Ridge - MSE: 0.842804 $R^2$: 0.151030
Modelo: Lasso - MSE: 0.894370 $R^2$: 0.099087
Modelo: Random Forest - MSE: 0.072431 $R^2$: 0.927039
Modelo: Gradient Boosting - MSE: 0.123625 $R^2$: 0.875471
Modelo: K-Neighbors - MSE: 0.588710 $R^2$: 0.406983
Modelo: Decision Tree - MSE: 0.125870 $R^2$: 0.873209
```

- Además de imprimir los *scores* (***MSE*** y ***R2***) para cada modelo y registrarlos en las listas, vas a generar un gráfico donde se vea:
  - Un *scatter* de color rojo que representa cada valor de ***y_test***. Esto genera una línea recta de color rojo formada por los valores de test: los puntos ***(y_test, y_test)*** siempre ocuparán la diagonal del gráfico.
  - Otro *scatter* de color azul representa cada valor de ***y_test*** y cada valor predicho para ese ***y_test***. Así podremos hacernos una idea de cómo de próximas están las predicciones del regresor a sus valores reales.
  - Puedes seleccionar uno de los gráficos del ***plt.figure(figsize=(12,10))*** de la línea 7 ejecutando antes de crear los gráficos, la sentencia ***plt.subplot(5,3,i+1)*** dentro del bucle.



- Fuera del bucle vamos a comparar el ***MSE*** de los modelos con un gráfico de barras donde se vea *train* y *test*. Esto nos ayudará a detectar posibles ***underfitting*** y ***overfitting*** y ver desempeño de los modelos.



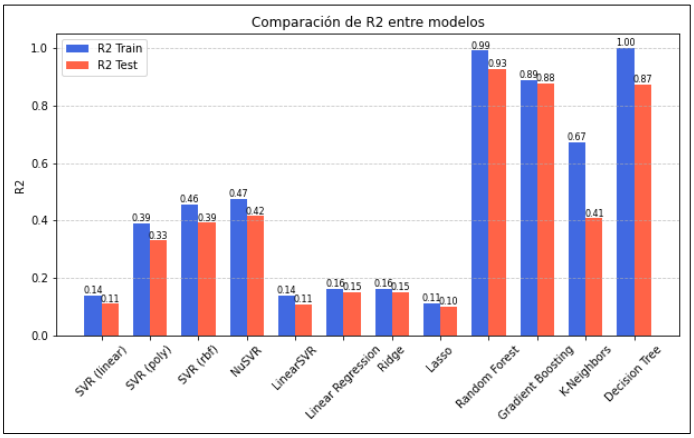
ENTREGA 4: Muestra:

a) Etapas y pasos de la actividad con el texto, el código y los gráficos y resultados que se generan en un notebook de Jupyter.

b) Responde ***mirando tu gráfico de MSE***:

- El peor modelo parece ser: _____
- El mejor modelo parece ser: _____
- Cuál es el que tiene más *overfitting*: _____

- De manera similar haz un gráfico de barras donde aparezca el R2 de cada modelo.



ENTREGA 5: Responde:

a) Responde ***mirando tu gráfico de R2***:

- El peor modelo y sus ***R2*** de *train* y *test*: _____
- El mejor modelo y sus ***R2*** de *train* y *test*: _____
- Cuál es el que tiene más *overfitting*: _____

- Escoge uno de los modelos de máquinas de soporte vectorial y ajusta sus hiperparámetros a mano hasta que mejores su R2 con respecto el valor inicial.
- Si ***K-NN*** tiene *overfitting*, intenta mejorar la configuración del modelo de la misma forma.

ENTREGA 6: Muestra:

a) Nombre del modelo ***SVR*** que has escogido.

b) ***MSE*** y ***R2*** original del ***SVR***: _____

c) Código que realice los cambios a la configuración de los modelos ***SVR*** y ***K-NN***, los entrene y calcule sus nuevos ***scores***.

d) Gráfico de puntos de datos de test y predicciones de ambos.

e) ***MSE*** y ***R2*** mejorado de ambos: _____

📌ACTIVIDAD 3: ÁRBOLES DE DECISIÓN PARA REGRESIÓN.

Crea el *notebook* ***saa_u03_p01_a3-<tus_iniciales>.ipynb*** donde entregar esta actividad. Los árboles de decisión son propensos a presentar *overfitting*. En *scikit-learn* podemos reducirlo ajustando ciertos hiperparámetros. En el caso de ***DecisionTreeRegressor***:

- Antes del entrenamiento:
  - Limitar la profundidad del árbol (***max_depth***)
  - Restringir el número mínimo de muestras por nodo (***min_samples_split***, ***min_samples_leaf***)
- Poda después del entrenamiento:
  - Reducir la complejidad con "***ccp_alfa***".

Utiliza los modelos ***DecisionTreeRegressor***, ***GradientBoostingRegressor***, ***XGBRegressor*** y ***RandomForestRegressor*** para entrenarlos con los datos del ejercicio anterior intentando mejorar los resultados en caso de haberlos usado ya, o bajar el *overfitting* en caso de no haberlos usado.

ENTREGA 7: Muestra Código, gráficos y capturas de ejecución de:

a) Carga de datos y preprocesamiento (si es necesario).
b) Entrenamiento y configuración del ***DecisionTreeRegresor***: desempeño inicial, cambio de hiperparámetros y desempeño final.
c) Igual para el ***GradientBoostingRegressor***.
d) Igual para ***XGBRegressor***.
e) Igual para ***RandomForestregressor***.
f) Importancia o influencia de cada característica en alguno de los modelos.

📌ACTIVIDAD 4: APLICAR SVR, ÁRBOLES Y RAMDOMFOREST.

Crea el *notebook* ***saa_u03_p01_a4-<tus_iniciales>.ipynb*** donde entregar esta actividad. Intenta encontrar un modelo regresor basado en ***SVR*** o ***CART*** o ***RandomForestRegressor*** de ***sklearn.ensemble*** que mejore los resultados del tasador de pisos que elegiste en una práctica de la unidad anterior.

ENTREGA 8: Muestra código y capturas de ejecución de configuración y medición de desempeño con gráficos de:

a) Un regresor con ***SVM***.
b) Un regresor basado en árboles de decisión.
c) Un regresor basado en ***RandomForestRegressor*** o e ***GradientBoostingRegressor***.

En alguno de ellos muestra importancia o influencia de cada característica usada.

📌ACTIVIDAD 5: REGRESIÓN A PARTIR DE FOTOGRAFÍAS.

DEFINIR PROBLEMA Y RECOPILAR DATOS

Crea el *notebook* ***saa_u03_p01_a5-<tus_iniciales>.ipynb*** donde entregar esta actividad. Necesitamos consensuar por votación 2 posibles problemas (lo que escoja la mayoría de la clase gana) más que nada por obtener suficiente cantidad de datos de alguno de los problemas:

a) Predecir la edad de una persona: Si nadie está en contra de aportar fotografías personales, cada alumno buscará 10 fotografías suyas o de conocidos (propias, familia, amigos, …) realizadas en diferentes edades y las etiquetará con la edad que tenía en ese momento la persona que aparece ***"edad_<tus_iniciales>-<num_foto>.jpg"*** o ***"edad_<tus_iniciales>-<num_foto>.png"***. En el caso de descargar de Internet las imágenes o de generarlas con aplicaciones tened cuidado porque al buscar os pueden aparecer las mismas fotografías para diferentes edades. La cara debe cubrir casi toda la foto (sin paisaje de fondo: ajustar el borde de abajo a la barbilla y los laterales a las orejas y el borde superior al pelo) y la persona debe estar mirando de frente.

b) Predecir la peligrosidad de un animal en un rango de 0 a 10: 10 significa que te puede matar o desgraciar si te engancha y 0 que no te va a dañar (al menos en principio). En caso de escoger esta opción cada uno buscará, procesará y aportará 10 fotografías de cabezas de animales de todo tipo (serpientes, insectos, felinos, osos, tiburones, ovejas, gatitos, …) con el nombre del fichero siguiendo el formato ***"peligo_<tus_iniciales>-<num_foto>.jpg"*** o bien formato ***"peligo_<tus_iniciales>-<num_foto>.png".***

Nota: *este enfoque no tiene visos de dar buenos resultados. Lo ideal sería extraer característics de cada fotografía (zonas de ojos, boca, nariz, orejas, dientes) creando embeddings y codificando estos rasgos a través de deep learning y luego usarlos para realizar las predicciones, pero vamos a probar a ver que tal nos va, al fin y al cabo es una excusa para probar regresores.*

Una vez que tengas las fotografías debes procesarlas. Te paso el siguiente código:

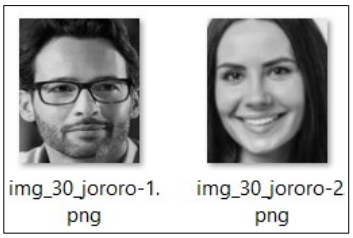
```python
import numpy as np
import pandas as pd
import re
import cv2              # 🔄 Instalar opencv: pip install opencv-python
import os

carpeta = "./edad"                          # 🔄 ** CAMBIA!! ** Ruta de carpeta donde están las imágenes
archivo_salida = "josrosrod_imagenes.csv"   # 🔄 ** CAMBIA!! ** Ruta de archivo donde guardar datos como .csv
# Recorrer todas las imágenes en la carpeta
datos_procesados = []
patron = r"^\d+_.*\.(jpg|png)$"
for nombre_archivo in os.listdir(carpeta):
    ruta_completa = os.path.join(carpeta, nombre_archivo)
    if not re.match(patron, nombre_archivo): # Si el fichero no es imagen con dato numérico antes de _
        continue
    edad = int(nombre_archivo.split("_")[0])
    # Leer la imagen en escala de grises, escalarla, guardar la procesada
    imagen = cv2.imread(ruta_completa, cv2.IMREAD_GRAYSCALE)
    if imagen is None:
        print(f"No se pudo leer: {nombre_archivo}")
        continue
    imagen_escalada = cv2.resize(imagen, (92,112), interpolation=cv2.INTER_AREA) # Escala a 92x112
    cv2.imwrite(os.path.join(carpeta, "img_" + nombre_archivo), imagen_escalada)
    imagen_normalizada = (imagen_escalada / 255.0).astype(np.float32)        # Normalizar a [0,1]
    datos_procesados.append({"edad":edad, "imagen": imagen_normalizada})     # Añadir datos a datos_procesados

# Convertir datos_procesados a DataFrame y guardar como csv
df = pd.DataFrame([ {"edad": item["edad"],
                     "imagen": ",".join(map(str, np.ravel(item["imagen"])))}
                    for item in datos_procesados])
df.to_csv(archivo_salida, index=False)
print(f"✅ Se guardaron {len(datos_procesados)} imágenes procesadas en '{archivo_salida}'")
```

Lo que hace el código es definir rutas (que debes adaptar para tu uso) en las variables ***carpeta*** (ruta relativa para alcanzar el lugar donde están las imágenes) y ***archivo_salida*** (pathname relativo que define el archivo .csv donde se van a guardar los datos).

Las imágenes de ***carpeta*** se transforman usando la librería ***opencv*** (quizás debas instalarla) en información numérica de la siguiente manera: obtendremos una imagen en escala de grises de dimensiones 92x112 píxels (ancho x alto) que se almacenan como valores float de 32 bits sin signo entre 0 y 256 normalizados a float en el intervalo [0,1]. La columna target de cada foto será la primera característica del dataset. Por ejemplo podemos tener ficheros como estos:

Y al ejecutar el script obtenermos el resultado y generamos:

```
✅ Se guardaron 2 imágenes procesadas en 'josrosrod_imagenes.csv'
```



Y un fichero .csv donde anotamos los datos de cada ejemplo, algo como esto:

```
edad,imagen
30,"0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.21568628,0.22352941,0.1882353,0.20784314,0.21960784,0.
30,"0.92941177,0.8862745,0.8666667,0.7254902,0.45882353,0.32941177,0.38039216,0.3882353,
```

Debes subir tus fotografías debidamente etiquetadas a la siguiente carpeta: 📁 carpeta. Pero antes comprueba que son correctos y de paso realizas la carga de datos. Para ello prueba a cargar tus imágenes en un ***DataFrame*** de pandas y mostrar sus datos y visualizar alguna de las imágenes que contiene. El siguiente ejemplo visualiza la primera fotografía usando ***opencv*** y aunque sale en su propia ventana nos sirve para comprobar que las hemos generado y cargado bien:

```python
df = pd.read_csv(archivo_salida)
# Convertir la imagen de string a numpy array
df["imagen"] = df["imagen"].apply(lambda x: np.array(list(map(float, x.split(",")))).reshape(112,92))
#Imprimir datos de la primera cara
print(f"Edad: {df.iloc[0]['edad']} Imagen: {df.iloc[0]['imagen']}")
pixel_array = (df.iloc[0]["imagen"] * 255).astype(np.uint8)
cv2.imshow(f"Edad:", pixel_array)
```

Este otro código define un método al que indicas (un array de imágenes, un array con sus etiquetas y desde que imagen hasta qué imagen quieres visualizar). El método usa ***matplotlib*** para visualizarlas añadiendo etiquetas con el dato (la edad en este ejemplo) en un recuadro rojo en la esquina superior izquierda) y con el índice que ocupa en el ***DataFrame*** (una caja de color verde en la esquina inferior izquierda).

Y aquí está ese otro código:

```python
import matplotlib.pyplot as plt
def print_imagenes(imgs, targets, desde, hasta):
    # configuramos el tamaño de las imágenes por pulgadas
    fig = plt.figure(figsize=(30, 24))
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
    for i in range(desde, hasta):
        # graficamos las imagenes en una matriz de 25x20
        p = fig.add_subplot(25, 20, i + 1, xticks=[], yticks=[])
        p.imshow(imgs[i], cmap="gray")
        # etiquer imágenes con target e índice
        p.text(0, 14, str(targets[i]), bbox=dict(facecolor='red', alpha=0.5))
        p.text(0, 100, str(i), bbox=dict(facecolor='green', alpha=0.5))

print_imagenes(df.iloc[:]["imagen"], df.iloc[:]["edad"], 0, 2)
```

ENTRENAR VARIOS REGRESORES Y MEDIR SU DESEMPEÑO

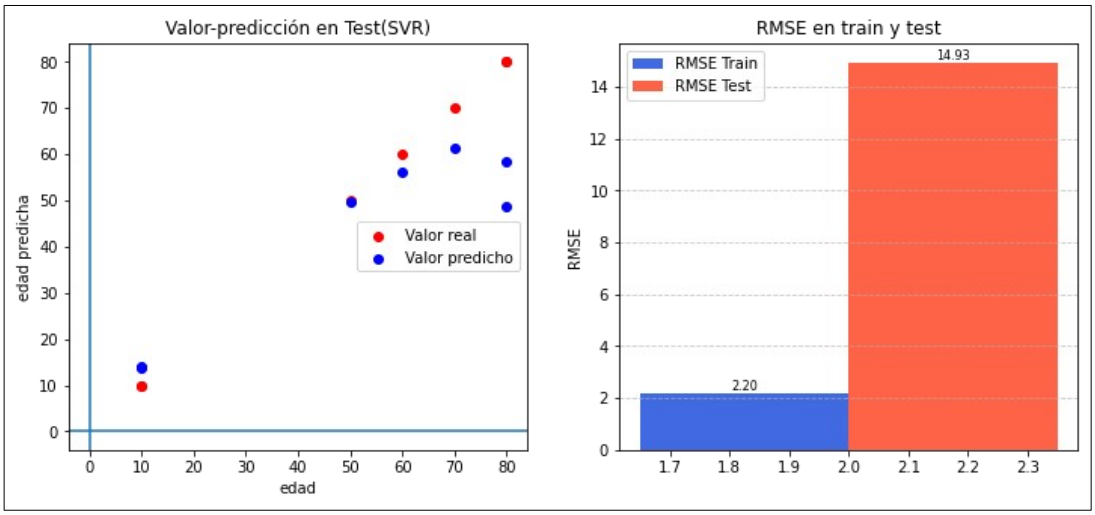
Ahora vamos a utilizar varios regresores para ver el desempeño que somos capaces de conseguir en esta tarea. Debes probar todos los regresores que importamos en esta figura:

```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
```

En primer lugar necesitamos transformar la característica imagen de cada cara en una característica por cada pixel, para ello:

```python
# Preparar características del dataframe
y = df['edad']
df_pixels = df["imagen"].apply(lambda img: img.flatten())  # Aplana cada imagen
df_pixels = pd.DataFrame(df_pixels.tolist())               # Expandir en columnas
df_final = pd.concat([df["edad"], df_pixels], axis=1)      # Unir con la edad
X = df_final.drop(columns=["edad"])                        # Características (píxeles)
```

Como es algo que haremos en todos los modelos, voy a pasarte el código de un método que nos ahorrará trabajo. Solo tenemos que pasar en cada llamada los valores ***y_train***, ***y_test***, ***y_train_predicho***, ***y_test_predicho*** y el nombre del modelo. Los valores reales y las predicciones deben pasarse sin escalar para que se entiendan bien los gráficos. La figura se obtiene con SVR y solo 31 fotos originales:



El código del método es este:

```python
def resumen_resultado(y_train, y_train_predicho, y_test, y_test_predicho, nombre_modelo="modelo"):
    rmse_train = mean_squared_error(y_train, y_train_predicho)**0.5
    rmse_test = mean_squared_error(y_test, y_test_predicho)**0.5
    print(f"RMSE en train de {nombre_modelo}: {rmse_train:.6f}")
    print(f"RMSE en test de {nombre_modelo}: {rmse_test:.6f}")
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.scatter(y_test, y_test, color="red", label="Valor real")
    plt.scatter(y_test, y_test_predicho, color="blue", label="Valor predicho")
    plt.xlabel("edad")
    plt.ylabel("edad predicha")
    plt.axvline()
    plt.axhline()
    plt.title(f"Valor-predicción en Test({nombre_modelo})")
    plt.legend()
    plt.subplot(1, 2, 2)
    bar_width = 0.35
    bars1 = plt.bar(2 - bar_width/2, rmse_train, width=bar_width, label="RMSE Train", color="royalblue")
    bars2 = plt.bar(2 + bar_width/2, rmse_test, width=bar_width, label="RMSE Test", color="tomato")
    # Anotar valores encima de las barras
    for bars in [bars1, bars2]:
        for bar in bars:
            yval = bar.get_height()
            plt.text(bar.get_x() + bar.get_width()/2, yval, f"{yval:.2f}", ha="center", va="bottom", fontsize=8)
    plt.ylabel("RMSE")
    plt.title("RMSE en train y test")
    plt.legend()
    plt.grid(axis="y", linestyle="--", alpha=0.7)
```

ENTREGA 9:

a) Añade a la carpeta compartida tus 10 fotografías con el formato indicado.
b) Adapta el código propuesto, lo entregas y lo ejecutas.
c) Entrenas y pruebas el modelo ***SVR***.
d) Entrenas y pruebas el modelo ***DecisionTreeRegressor***.
e) Entrenas y pruebas el modelo ***KneighborsRegressor***.
f) Entrenas y pruebas el modelo ***BaggingRegressor***.
g) Entrenas y pruebas el modelo ***RandomForestRegressor***.
h) Entrenas y pruebas el modelo ***AdaBoostRegressor***.
i) Entrenas y pruebas el modelo ***GradientBoostingRegressor***.
j) Comenta en cada uno de ellos los ajustes de hiperparámetros que has intentado para evitar que tengan *underfitting* o bien *overfitting*. Si no consigues solucionarlo deja la mejor configuración.