Contents

1 PyCaret

PyCaret es una biblioteca de aprendizaje automático de código abierto y de bajo código en Python que automatiza el proceso de aprendizaje automático. Está diseñado para hacer que el proceso de construcción, entrenamiento y despliegue de modelos de aprendizaje automático sea más fácil y más rápido. Con PyCaret, puede construir, entrenar y desplegar rápidamente modelos de aprendizaje automático con sólo unas pocas líneas de código. Proporciona una completa suite de funciones para preprocesamiento, ingeniería de características, selección de modelos y evaluación de modelos. PyCaret también incluye herramientas para ajustar automáticamente los hiperparámetros, crear modelos en ensamblado y el despliegue de modelos en producción.

PyCaret es un envoltorio de Python sobre otras librerías tales como scikit-learn, XGBoost, LightGBM, CatBoost, spaCy, Optuna, Hyperopt, Ray y algunas más.

El diseño y la simplicidad de PyCaret están inspirados en el papel emergente de los científicos de datos ciudadanos, un término utilizado por primera vez por Gartner. Los científicos de datos ciudadanos son usuarios avanzados que pueden realizar tareas analíticas simples y moderadamente sofisticadas que anteriormente habrían requerido más experiencia técnica.

La mayoría de los profesionales del aprendizaje automático comienzan a experimentar con la establecida biblioteca scikit-learn, pero existe una alternativa más sencilla y accesible: PyCaret.

Esta biblioteca tiene muchas ventajas en comparación con scikit-learn, especialmente para personas con experiencia limitada. Ahora veremos una descripción general de las características principales de PyCaret, así como un estudio de un caso centrado en la regresión. Sugiero instalar la

última versión de Anaconda en Windows/macOS/Linux para seguir este tutorial, pero también es compatible con Google Colab. Puedes ejecutar el código en un cuaderno Jupyter o usar tu IDE preferido.

2 Instalar PyCaret

2.1 Con «conda»

```
conda install -c conda-forge pycaret
```

2.2 Con pip

```
# create a conda environment
conda create --name yourenvname python=3.8

# activate conda environment
conda activate yourenvname

# install pycaret
pip install pycaret

# create notebook kernel
python -m ipykernel install --user --name yourenvname --display-name "display-name"
```

2.3 Version pre-release (con soporte para python 3.10, se supone)

```
pip install -U --pre pycaret
```

2.4 Desde el código fuente

```
pip install git+https://github.com/pycaret/pycaret.git#egg=pycaret
#ejecutar test
pytest pycaret
```

2.5 Con docker

Docker usa contenedores para crear entornos virtuales que aíslan una instalación de PyCaret del resto del sistema. El contenedor docker de PyCaret viene con un entorno Notebook preinstalado, que puede compartir recursos con su máquina host (acceder a directorios, usar la GPU, conectarse a Internet, etc.). Las imágenes de PyCaret Docker se prueban para cada versión.

```
docker run -p 8888:8888 pycaret/slim
```

Para una imagen con la versión completa de la imagen docker:

```
docker run -p 8888:8888 pycaret/full
```

Ejecutar docker con un volumen externo, por ejemplo donde vayamos a tener el proyecto, en este caso he puesto el directorio actual de trabajo:

```
docker run --name pycaret -v $PWD:/home/jovyan/work -p 8888:8888 pycaret/full
```

3 El Módulo de Regresión de PyCaret

La regresión es una tarea básica de aprendizaje automático supervisado que estima la relación entre una variable dependiente, y (conocida como objetivo) y variables independientes (conocidas como características).

La regresión se puede utilizar para predecir valores continuos, como el valor de una casa, en lugar de la clasificación, que se utiliza para valores discretos conocidos como clases. El módulo de regresión de PyCaret, que usa sklearn bajo el capó, le permite crear y probar modelos de regresión con unas pocas líneas de código. Incluye una variedad de algoritmos, así como la capacidad de trazar y ajustar hiperparámetros.

Ahora vamos a examinar un caso de estudio de regresión basado en ese módulo.

3.1 Cargando los datos —Dataset—

La base de todo proyecto de aprendizaje automático es la adquisición o creación de un conjunto de datos adecuado. PyCaret incluye una variedad de conjuntos de datos de ejemplo para diferentes tipos de tareas de aprendizaje automático y, en este proyecto, utilizaremos el conjunto de datos de seguros médicos.

Este conjunto de datos se origina en el libro Aprendizaje automático con R de Brett Lantz y contiene información sobre seguros de salud. La variable objetivo, y, representa los cargos de seguro para cada persona, y las características son propiedades, como la edad, el sexo y el índice de masa corporal (IMC, en inglés BMI).

Los datos del mundo real rara vez son tan simples, pero trabajar con conjuntos de datos de juguetes nos ayuda a comprender los conceptos y la metodología antes de pasar a casos más complejos.

Aquí hay una descripción para cada variable del conjunto de datos:

- age (edad): edad del beneficiario principal
- sex (sexo): sexo del contratista de seguros - female (femenino), male (masculino)
- bmi (imc): Índice de masa corporal, que proporciona una comprensión de los pesos corporales que son relativamente altos o bajos en relación con la altura. Un índice objetivo del peso corporal (kg/m^2) utilizando la relación entre la altura y el peso, idealmente de 18,5 a 24,9
- children (hijos): Número de hijos cubiertos por el seguro de salud / Número de dependientes
- smoker (fumador): fuma
- region (región): área residencial del beneficiario en los EE. UU., noreste, sureste, suroeste, noroeste.
- charges (cargos): costos médicos individuales facturados por el seguro de salud

Para obtener los datos, usaremos la función get_data de pycaret:

```
from pycaret.datasets import get_data

data = get_data('insurance')
```

Ejemplo de datos:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

```
data.info()
```

que produce esta salida:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

La función `get_data` devuelve un dataframe pandas, por lo que podemos usar la función `info()` de pandas para obtener algunos detalles sobre el conjunto de datos.

Como podemos ver, hay 1338 registros y cero valores nulos. La mayoría de los conjuntos de datos del mundo real tienen algunos valores nulos y pueden requerir cierta ingeniería de características, pero en este caso no tenemos que lidiar con eso.

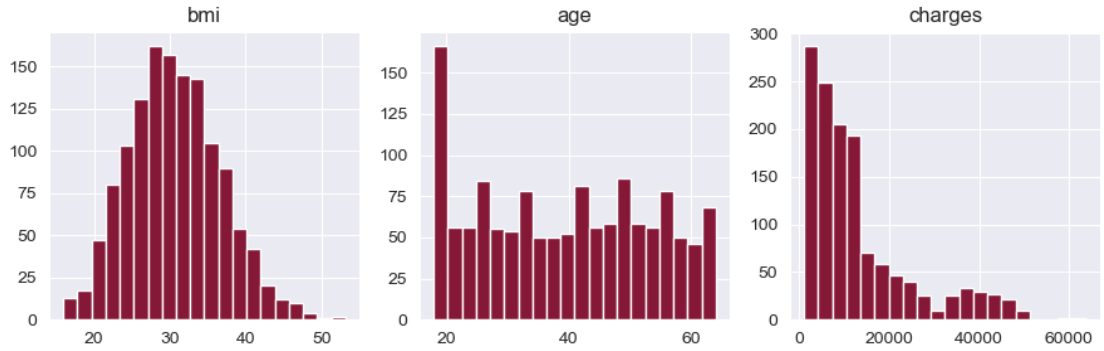
3.2 Análisis Exploratorio de Datos —Exploratory Data Analysis (EDA)—

Después de cargar el conjunto de datos, normalmente necesitaremos examinar y comprender sus propiedades básicas. Esto se conoce como análisis exploratorio de datos y se puede lograr con varias herramientas y métodos, como el trazado.

Comenzamos trazando los histogramas de las variables numéricas.

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
import sys

from pycaret.datasets import get_data
```

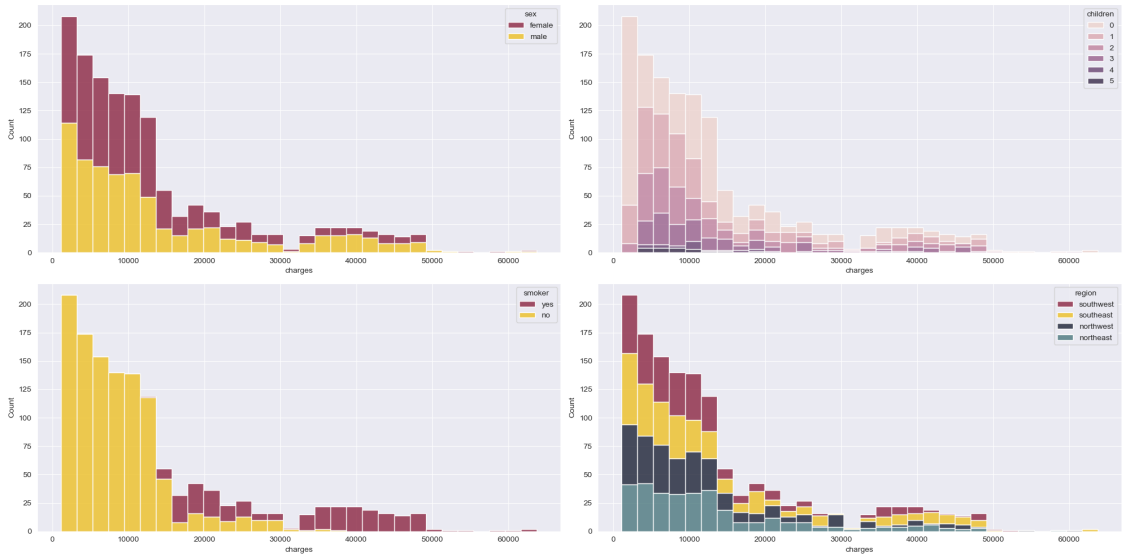
```
data = get_data('insurance')

sns.set_style('darkgrid')
colors = ['#851836', '#EDBD17', '#0E1428', '#407076', '#4C5B61']
sns.set_palette(sns.color_palette(colors))

numerical = ['bmi', 'age', 'charges']
data[numerical].hist(bins=20, layout=(1, 3), figsize=(9,3))

plt.tight_layout()
plt.savefig(sys.stdout.buffer)
#plt.show()
```



Aquí, estamos usando la función integrada hist() de pandas para trazar un histograma de edad, IMC y cargos. Esto nos ayuda a comprender mejor la distribución de valores para estas variables numéricas.

La variable IMC tiene una distribución cercana a la normal, mientras que la variable cargos está sesgada a la derecha. Las distribuciones sesgadas pueden ser un problema para los algoritmos de aprendizaje automático, por lo que nos ocuparemos de eso más adelante.

Ahora, seremos un poco creativos al trazar el histograma de la variable de destino, es decir, los cargos del seguro, con barras apiladas que representan diferentes categorías de las variables categóricas. Logramos esto usando la función histplot() de la biblioteca seaborn:



Los fumadores tienen cargos significativamente más altos, y podemos ver que los hombres tienen costes médicos más altos con más frecuencia que las mujeres.

Ahora que hemos obtenido información útil de EDA, comencemos el proceso de regresión con PyCaret sobre estos datos.

3.3 Initializing a PyCaret Environment

La función setup() de PyCaret inicializa el entorno y prepara la implementación y los datos de modelado de aprendizaje automático. Hay dos parámetros necesarios, un conjunto de datos y la variable objetivo. Después de ejecutar la función, se infiere el tipo de cada característica y se realizan varias tareas de preprocesamiento en los datos.

```
from pycaret.regression import *

reg = setup(
    data=data,
    target='charges',
    train_size=0.8,
    session_id=10,
    normalize=True,
    transform_target=True
)
```

co	Description	Value
0	session_id	10
1	Target	charges
2	Original Data	(1338, 7)
3	Missing Values	False
4	Numeric Features	2
5	Categorical Features	4

Continúa en la siguiente página

Continúa de la página anterior

co	Description	Value
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(1070, 14)
10	Transformed Test Set	(268, 14)
11	Shuffle Train-Test	True
12	Stratify Train-Test	False
13	Fold Generator	KFold
14	Fold Number	10
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False
18	Experiment Name	reg-default-name
19	USI	bd4e
20	Imputation Type	simple
21	Iterative Imputation Iteration	None
22	Numeric Imputer	mean
23	Iterative Imputation Numeric Model	None
24	Categorical Imputer	constant
25	Iterative Imputation Categorical Model	None
26	Unknown Categoricals Handling	least_frequent
27	Normalize	True
28	Normalize Method	zscore
29	Transformation	False
30	Transformation Method	None
31	PCA	False
32	PCA Method	None
33	PCA Components	None
34	Ignore Low Variance	False
35	Combine Rare Levels	False
36	Rare Level Threshold	None
37	Numeric Binning	False
38	Remove Outliers	False
39	Outliers Threshold	None
40	Remove Multicollinearity	False
41	Multicollinearity Threshold	None
42	Clustering	False
43	Clustering Iteration	None
44	Polynomial Features	False
45	Polynomial Degree	None
46	Trignometry Features	False
47	Polynomial Threshold	None
48	Group Features	False

Continúa en la siguiente página

Continúa de la página anterior

co	Description	Value
49	Feature Selection	False
50	Features Selection Threshold	None
51	Feature Interaction	False
52	Feature Ratio	False
53	Interaction Threshold	None
54	Transform Target	True
55	Transform Target Method	box-cox

Después de ejecutar la función setup() en nuestros datos, los resultados muestran la canalización de preprocesamiento aplicada al conjunto de datos. Algunos aspectos destacados de este pipeline son:

1. Tipos de datos inferidos. Podemos ver que cuatro características han sido correctamente identificadas como categóricas, y el resto como numéricas. En caso de que PyCaret no lo haga correctamente, podemos definirlos en la función setup() nosotros mismos, utilizando los parámetros categorical_features y numeric_features.

2. División de entrenamiento/prueba. El conjunto de datos se ha dividido en un conjunto de entrenamiento y prueba, ya que es una práctica estándar en el aprendizaje automático. El tamaño del conjunto de trenes se ha establecido en el 80% del conjunto de datos original, lo que significa que el 80% de los datos se utilizarán para entrenar el modelo de aprendizaje automático y el resto para probar su precisión.

3. Normalización de Características Numéricas. Muchos algoritmos de regresión que requieren que las funciones se normalicen para que funcionen como se espera. Las características normalizadas tienen $\mu = 0$ y $\sigma = 1$. El método estándar para lograrlo es reemplazar cada valor con su puntaje z asociado, que se define como $z = \frac{z-\mu}{\sigma}$.

4. Codificación One-Hot de características categóricas Algunos algoritmos de aprendizaje automático aceptan características categóricas y otros que no, por lo que es mejor convertirlos en características numéricas mediante la codificación one-hot. La codificación one-hot elimina las características categóricas y las reemplaza con variables binarias adicionales, una para cada categoría, menos una (para evitar la trampa de la variable ficticia).

5. Transformación de destino Como hemos notado en la sección EDA, la variable de destino está sesgada a la derecha. Esto podría causar problemas, ya que muchos algoritmos de regresión esperan que los datos tengan una distribución normal para funcionar de manera óptima. La función setup() incluye la opción de transformar el objetivo para tener una distribución cercana a la normal. Las transformaciones también se pueden aplicar a las funciones si es necesario, pero no fue necesario en este caso.

Hay varios otros parámetros avanzados en la función `setup()`, por lo que si tiene curiosidad, no dude en consultar la sección correspondiente de sus documentos que repasa cada parte en detalle.

3.4 Ver los datos preprocesados

La función get_config('X') devuelve el conjunto de datos de características después de que se le haya aplicado la canalización de preprocesamiento

```
get_config('X')
```

1338 filas × 14 columnas

Podemos ver que las características numéricas se han normalizado con el método de puntuación z y las características categóricas se han codificado con codificación one-hot. Es importante verificar que el procesamiento previo se haya completado con éxito, ya que, en algunos casos, nuestro conjunto de datos podría no estar tan limpio como el que se usó en este ejemplo. En caso de que falle la canalización de preprocesamiento, es posible que obtengamos resultados incorrectos e inesperados de los modelos de aprendizaje automático.

3.5 Comparando diferentes modelos

Hay numerosos algoritmos de regresión disponibles y no siempre es obvio cuál es el óptimo para nuestro conjunto de datos. La única forma de encontrar el mejor modelo es probar varios de ellos y comparar los resultados. Afortunadamente, PyCaret proporciona la función compare_models(), que compara fácilmente una variedad de modelos diferentes:

```
best = compare_models(sort='RMSE')
```

Después de ejecutar la función compare_models(), se muestran los resultados. Esta tabla puede parecer intimidante, pero en realidad es bastante simple de entender. La primera columna contiene el nombre de cada modelo, y el resto de las columnas son varias métricas.

Puede concentrarse en RMSE por ahora, que significa Root Mean Squared Error. RMSE es una métrica ampliamente utilizada para la regresión, y se define como la raíz cuadrada de la diferencia cuadrada promediada entre el valor real y el predicho por el modelo:

$$RMSE = \sqrt{\frac{1}{N}\sum Ni = 1(x_i - \hat{x_i})^2}$$

Cuanto menor sea el valor de RMSE, más preciso será nuestro modelo. En este caso, el mejor modelo es el modelo Gradient Boosting Regressor, con un valor RMSE de 4368.4047.

3.6 Creación de un modelo con PyCaret

La función create_model() le permite crear un modelo de regresión basado en el algoritmo de su preferencia. En este caso, usaremos Gradient Boosting Regressor, ya que tuvo el mejor rendimiento de compare_models() anterior.

La función create_model() utiliza la validación cruzada de k veces para evaluar la precisión del modelo. En este método, el conjunto de datos primero se divide en k submuestras, una submuestra se conserva para la validación y el resto se usa para entrenar el modelo. Este proceso se repite varias veces, y cada submuestra se usa solo una vez como datos de validación.

```
model = create_model('gbr', cross_validation=True, fold=10)
```

col	age	bmi	sex_female	child_0	child_1	child_2	child_3	child_4	child_5	smoker_no	northeast	northwest	southea
0	-1.423959	-0.457049	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	-1.494665	0.498336	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
2	-0.787608	0.373013	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
3	-0.434080	-1.302572	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0
4	-0.504786	-0.297547	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1333	0.767917	0.042616	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0
1334	-1.494665	0.197235	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0
1335	-1.494665	0.999627	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1336	-1.282548	-0.798839	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0
1337	1.545679	-0.266623	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0

Method	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2049.5818	20421350.9926	4370.9223	0.8629	0.3560	0.1638	0.0160
rf	Random Forest Regressor	2148.1937	20860760.2193	4463.4103	0.8585	0.3816	0.1857	0.0560
lightgbm	Light Gradient Boosting Machine	2320.8815	21164314.7458	4491.1725	0.8568	0.3771	0.1910	0.0550
catboost	CatBoost Regressor	2272.1176	21920375.9207	4552.2118	0.8525	0.3685	0.1737	0.7090
ada	AdaBoost Regressor	3028.9828	22202851.6593	4619.7146	0.8498	0.4583	0.3918	0.0110
et	Extra Trees Regressor	2290.2154	24332567.3742	4858.0821	0.8341	0.4075	0.2014	0.0490
xgboost	Extreme Gradient Boosting	2782.4022	35645139.9000	5681.2366	0.7593	0.4167	0.2362	0.0950
dt	Decision Tree Regressor	2883.4270	38905351.2137	6207.1844	0.7276	0.4946	0.3120	0.0050
omp	Orthogonal Matching Pursuit	5700.3404	59762727.6470	7668.1283	0.5919	0.6876	0.6901	0.0070
ridge	Ridge Regression	4081.9423	63909181.2000	7873.9212	0.5655	0.4260	0.2620	0.0050
br	Bayesian Ridge	4088.1831	64170144.6909	7889.7825	0.5637	0.4259	0.2620	0.0050
lar	Least Angle Regression	4106.0225	64908348.6235	7935.2674	0.5587	0.4259	0.2619	0.0060
lr	Linear Regression	4106.0354	64908770.4000	7935.2942	0.5587	0.4259	0.2619	0.0080
huber	Huber Regressor	4245.0332	81231444.5110	8865.0706	0.4478	0.4356	0.2068	0.0080
knn	K Neighbors Regressor	4982.9582	81987651.9475	8946.7009	0.4452	0.5405	0.3290	0.0120
par	Passive Aggressive Regressor	6250.5322	114585710.1368	10361.4084	0.2406	0.6238	0.5523	0.0060
en	Elastic Net	8276.7225	165075368.0000	12754.6845	-0.1198	0.9128	0.9605	0.0050
llar	Lasso Least Angle Regression	8385.7427	166526887.1876	12811.0623	-0.1297	0.9245	0.9895	0.0060
lasso	Lasso Regression	8385.7422	166526895.2000	12811.0627	-0.1297	0.9245	0.9895	0.0080

col	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	1153.5021	3234575.3253	1798.4925	0.9704	0.2460	0.1493
1	2726.0526	31665967.0393	5627.2522	0.7558	0.4797	0.1887
2	2378.3264	29063760.2546	5391.0815	0.8309	0.3298	0.1569
3	2079.7411	21902085.8132	4679.9664	0.8806	0.4493	0.1497
4	1791.8977	17011091.0753	4124.4504	0.8976	0.3362	0.1576
5	1521.1686	9602958.3059	3098.8640	0.9157	0.2453	0.1530
6	1971.6365	15482810.4199	3934.8203	0.8652	0.3269	0.1721
7	2608.5165	31293027.6163	5594.0171	0.8249	0.4281	0.1597
8	2300.7854	25535340.4020	5053.2505	0.8118	0.4252	0.1699
9	1964.1911	19421893.6738	4407.0278	0.8760	0.2937	0.1813
Mean	2049.5818	20421350.9926	4370.9223	0.8629	0.3560	0.1638
SD	458.6997	8928114.5410	1147.3402	0.0571	0.0801	0.0129

Después de entrenar el modelo, se muestran los resultados de la validación cruzada. Establecimos el número de pliegues (k_0) en 10, por lo que en este caso, tenemos una validación cruzada de diez pliegues. Podemos ver las métricas de cada pliegue y la media y la desviación estándar de todos los pasos.

Si ha usado sklearn antes, notará que una línea de código con PyCaret equivale a varias líneas con sklearn..

3.7 Ajustar un modelo

La función tune_model() ajusta los hiperparámetros de un modelo determinado y genera los resultados. Los hiperparámetros son ajustes del modelo que se pueden modificar y pueden tener un efecto positivo o negativo en su precisión.

tune_model() utiliza el método de búsqueda de cuadrícula aleatoria para ajustar y optimizar el modelo probando una muestra aleatoria de los hiperparámetros. Podemos definir una cuadrícula con valores específicos para los hiperparámetros utilizando el parámetro custom_grid.

También podemos definir el número de iteraciones con el parámetro n_iter. Se selecciona un valor aleatorio de la cuadrícula definida de hiperparámetros para cada iteración y se prueba mediante validación cruzada k-fold.

```
params = {
    'learning_rate': [0.01, 0.1],
    'max_depth': [5, 6, 7, 8],
    'subsample': [0.6, 0.7, 0.8],
    'n_estimators' : [100, 300, 400, 500]
}

tuned_model = tune_model(
    model,
    optimize='RMSE',
    fold=10,
    custom_grid=params,
```

```
    n_iter=20
)
```

col	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	1245.5549	4026621.4141	2006.6443	0.9631	0.2423	0.1569
1	2583.1972	30189293.0509	5494.4784	0.7672	0.4800	0.1834
2	2442.0266	30135598.8806	5489.5900	0.8247	0.3410	0.1730
3	1997.9252	21734060.8504	4661.9804	0.8816	0.4503	0.1487
4	1946.5765	16879740.1085	4108.4961	0.8984	0.3409	0.1819
5	1488.9834	9451504.0446	3074.3299	0.9170	0.2606	0.1656
6	2025.9735	15546849.0444	3942.9493	0.8646	0.3295	0.1775
7	2387.5058	28945218.9003	5380.0761	0.8381	0.4243	0.1528
8	2317.8041	26198464.5189	5118.4436	0.8069	0.4456	0.1837
9	1843.0738	17015176.7972	4124.9457	0.8914	0.2845	0.1729
Mean	2027.8621	20012252.7610	4340.1934	0.8653	0.3599	0.1696
SD	404.6465	8560903.9358	1083.9623	0.0545	0.0807	0.0123

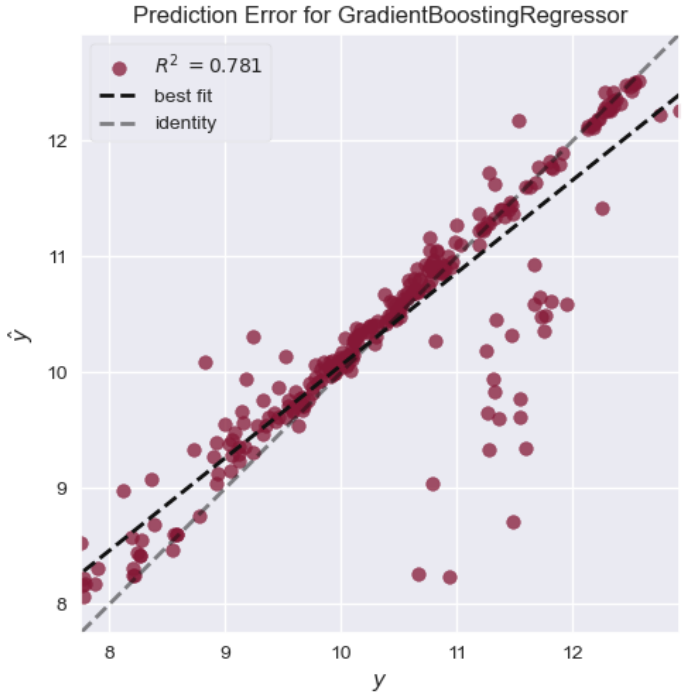
Como podemos ver en los resultados de la validación cruzada, el ajuste de hiperparámetros aumentó ligeramente la precisión del modelo. La mejora es pequeña, pero experimentar con un número de iteraciones más alto o una cuadrícula con diferentes valores de hiperparámetros puede generar mejores resultados.

3.8 Trazar/dibujar el rendimiento del modelo

PyCaret incluye una función plot_model() que nos permite visualizar la precisión de nuestro modelo y otras propiedades. La función incluye una variedad de gráficos que nos ayudan a evaluar y comprender mejor nuestro modelo. En comparación con las bibliotecas subyacentes utilizadas para generar estos gráficos (sklearn, pandas y matplotlib), el uso de PyCaret es significativamente más rápido y sencillo de trabajar.

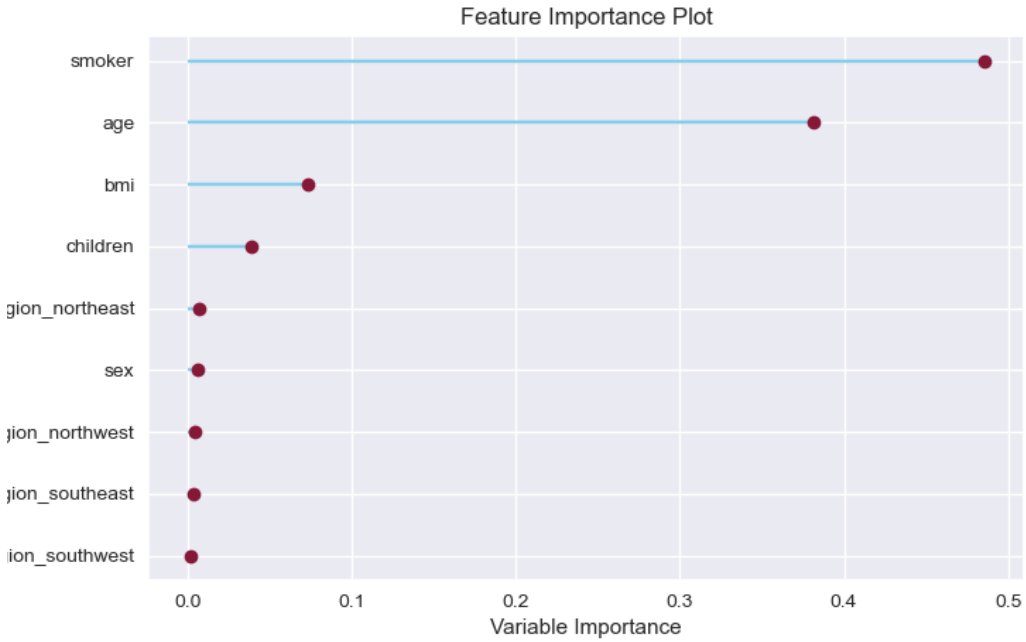
Primero, trazaremos el error de las predicciones en el conjunto de prueba:

```
plot_model(tuned_model, plot='error')
```



Second, we'll plot the importance of each feature:

```
plot_model(tuned_model, plot='feature')
```



En la sección anterior de EDA, vimos que ser fumador conduce a cargos de seguro significativamente más altos, y ahora, en el gráfico de importancia de características, vemos que ser fumador

tiene el valor predictivo más alto. Además, también podemos ver que la edad y el IMC también parecen jugar un papel importante.

3.9 Hacer predicciones sobre nuevos datos

El objetivo final de cada proyecto de aprendizaje automático del mundo real es hacer predicciones sobre nuevos datos, donde se desconoce el valor de la variable objetivo. Puede lograrlo utilizando la función `predict_model()`, que devuelve un dataframe pandas con las predicciones.

Vamos a crear un pequeño conjunto de datos sintéticos y probar nuestro modelo y ver cómo predice los cargos de seguro:

```
cols =  ['age', 'sex', 'bmi', 'children', 'smoker', 'region']

records = [
       [30, 'male', 20, 0, 'no', 'southeast'],
       [30, 'male', 20, 0, 'yes', 'southeast'],
       [30, 'male', 35, 0, 'yes', 'southeast'],
       [70, 'male', 35, 0, 'yes', 'southeast'],
       [30, 'female', 20, 0, 'no', 'southeast'],
       [30, 'female', 20, 0, 'yes', 'southeast'],
       [30, 'female', 35, 0, 'yes', 'southeast'],
       [70, 'female', 35, 0, 'yes', 'southeast']
]

new_data = pd.DataFrame(data=records, columns=cols)

predict_model(tuned_model, new_data)
```

col	age	sex	bmi	children	smoker	region	Label
0	30	male	20	0	no	southeast	4043.350231
1	30	male	20	0	yes	southeast	17007.642015
2	30	male	35	0	yes	southeast	35749.960178
3	70	male	35	0	yes	southeast	45790.897563
4	30	female	20	0	no	southeast	4503.047383
5	30	female	20	0	yes	southeast	17208.037478
6	30	female	35	0	yes	southeast	35853.324929
7	70	female	35	0	yes	southeast	45870.135872

Podemos ver que se prevé que los jóvenes no fumadores con un IMC bajo tengan los cargos más bajos según nuestro modelo. Por otro lado, se prevé que a los que son mayores, obesos y fuman se les cobrará diez veces más. Esos resultados están en línea con la EDA y la gráfica de importancia de la característica

3.10 Interpretación del modelo

La capacidad de interpretar los resultados de un modelo de aprendizaje automático le permite evitar depender de un "modelo de caja negra", en los que no se entiende exactamente cómo funcionan.

PyCaret incluye la función interpret_model() que proporciona un gráfico de interpretación para un modelo dado. Esta función requiere la biblioteca SHAP (SHapley Additive exPlanations) para funcionar, por lo que tendremos que instalarla primero.

```
pip install shap
```

Después de instalar la biblioteca SHAP, podemos crear una gráfica de interpretación para nuestro modelo. El Gradient Boosting Regressor no es compatible con la función interpret_model(), por lo que crearemos otro modelo basado en el algoritmo XGBoost e interpretaremos ese modelo en su lugar.

```
pip install xgboost
```

Para interpretar el modelo, usaremos el tipo de gráfico "motivo":

```
xgb = create_model('xgboost', cross_validation=True, verbose=False)

interpret_model(xgb, plot='reason', observation=32)
```

```
from pycaret.regression import *

from pycaret.datasets import get_data

data = get_data('insurance')

reg = setup(
  data=data,
  target='charges',
  train_size=0.8,
  session_id=10,
  normalize=True,
  transform_target=True
)
xgb = create_model('xgboost', cross_validation=True, verbose=False)

interpret_model(xgb, plot='reason', observation=32,save=True)
```

idx	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

idx	Description	Value
0	Session id	10
1	Target	charges
2	Target type	Regression
3	Original data shape	(1338, 7)
4	Transformed data shape	(1338, 10)
5	Transformed train set shape	(1070, 10)
6	Transformed test set shape	(268, 10)
7	Ordinal features	2
8	Numeric features	3
9	Categorical features	3
10	Preprocess	True
11	Imputation type	simple
12	Numeric imputation	mean
13	Categorical imputation	mode
14	Maximum one-hot encoding	25
15	Encoding method	None
16	Normalize	True
17	Normalize method	zscore
18	Transform target	True
19	Transform target method	yeo-johnson
20	Fold Generator	KFold
21	Fold Number	10
22	CPU Jobs	-1
23	Use GPU	False
24	Log Experiment	False
25	Experiment Name	reg-default-name
26	USI	bf7b

https://www.knime.com/blog/how-to-manage-python-environments-conda-and-knime

https://www.learndatasci.com/tutorials/introduction-pycaret-machine-learning/

https://machinelearningmastery.com/pycaret-for-machine-learning/

https://medium.com/low-code-for-advanced-data-science/machine-learning-in-knime-with-pycaret

https://shap.readthedocs.io/en/latest/

https://www.pycaret.org/tutorials/html/CLF101.html

https://anderfernandez.com/blog/category/machine-learning/