Regresión lineal simple

El conjunto de datos contiene información sobre las condiciones meteorológicas registradas cada día en varias estaciones meteorológicas de todo el mundo. La información incluye precipitaciones, nevadas, temperaturas, velocidad del viento y si el día incluyó tormentas eléctricas u otras malas condiciones meteorológicas.

Así que nuestra tarea es **predecir la temperatura máxima tomando como entrada la temperatura mínima.**

Información del dataset:

STA: Estación Meteorológica

Date: Fecha de observación

Precip: Precipitación en mm

WindGustSpd: Velocidad máxima de ráfagas de viento en km/h

MaxTemp: Temperatura máxima en grado celsius

MinTemp: Temperatura mínima en grado celsius

MeanTemp: Temperatura media en grado celsius

Snowfall: Nevadas y pellets de hielo en mm

PoorWeather: Una repetición de la columna TSHDSBRSGF

YR: Año de observación

MO: Mes de observación

DA: Día de observación

PRCP: Precipitación en pulgadas y centésimas

DR: Dirección máxima de la ráfaga de viento en decenas de grados

SPD: Velocidad máxima de ráfagas de viento en nudos

MAX: Temperatura máxima en grados farenheit

MIN: Temperatura mínima en grados farenheit

MEA: Temperatura media en grados farenheit

SNF: Nevadas en pulgadas y décimas

SND: Profundidad de nieve (incluye gránulos de hielo) registrada a las 1200 GMT excepto 0000 GMT en el Área del Lejano Oriente asiático en pulgadas y décimas

FT: Superficie cubierta congelada (profundidades en pulgadas)

FB: Base de tierra congelada (profundidad en pulgadas)

FTI: Espesor del suelo congelado (espesor en pulgadas)

ITH: Espesor de hielo en agua (pulgadas y décimas)

PGT: Tiempo pico de ráfagas de viento (horas y décimas)

TSHDSBRSGF: Día con: Trueno; Aguanieve; Granizo; Polvo o arena; Humo o neblina; Soplando nieve; Lluvia; Nieve; Esmalte; Niebla; 0 = No, 1 = Sí

SD3: La profundidad de la nieve a las 0030 GMT incluye gránulos de hielo en pulgadas y décimas

RHX: humedad relativa máxima de 24 horas, en su conjunto

RHN: humedad relativa mínima de 24 horas, en su conjunto

RVG: Caudal de río en pies y décimas

WTE: Equivalente de agua de nieve y hielo en el suelo en pulgadas y centésimas

La comprensión real surge cuando intentamos determinar los parámetros de entrada válidos y los datos de salida previstos. Para esto, necesitamos entender el conjunto de datos y cuáles son relevantes. También necesitamos determinar el parámetro de salida. Por lo tanto, después de una comprensión básica de los datos, finalmente podemos decidir las columnas que no utilizaremos.

Pre-procesado de datos

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
import sklearn.model_selection
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('archs/Weather.csv', low_memory=False)
df.head()
```

Out[2]:

	STA	Date	Precip	WindGustSpd	MaxTemp	MinTemp	MeanTemp	Snowfall	PoorWeather	YR	...	FB	FTI	ITH	PGT	TSHDSBRSGF	SD3	RHX
0	10001	1942-7-1	1.016	NaN	25.555556	22.222222	23.888889	0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	10001	1942-7-2	0	NaN	28.888889	21.666667	25.555556	0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	10001	1942-7-3	2.54	NaN	26.111111	22.222222	24.444444	0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	10001	1942-7-4	2.54	NaN	26.666667	22.222222	24.444444	0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	10001	1942-7-5	0	NaN	26.666667	21.666667	24.444444	0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 31 columns

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

Out[3]:
```
(119040, 31)
```

In [4]:
```
df.describe()
```

Out[4]:

	STA	WindGustSpd	MaxTemp	MinTemp	MeanTemp	YR	MO	DA	DR	SPD	...
count	119040.000000	532.000000	119040.000000	119040.000000	119040.000000	119040.000000	119040.000000	119040.000000	533.000000	532.000000	...
mean	29659.435795	37.774534	27.045111	17.789511	22.411631	43.805284	6.726016	15.797530	26.998124	20.396617	...
std	20953.209402	10.297808	8.717817	8.334572	8.297982	1.136718	3.425561	8.794541	15.221732	5.560371	...
min	10001.000000	18.520000	-33.333333	-38.333333	-35.555556	40.000000	1.000000	1.000000	2.000000	10.000000	...
25%	11801.000000	29.632000	25.555556	15.000000	20.555556	43.000000	4.000000	8.000000	11.000000	16.000000	...
50%	22508.000000	37.040000	29.444444	21.111111	25.555556	44.000000	7.000000	16.000000	32.000000	20.000000	...
75%	33501.000000	43.059000	31.666667	23.333333	27.222222	45.000000	10.000000	23.000000	34.000000	23.250000	...
max	82506.000000	75.932000	50.000000	34.444444	40.000000	45.000000	12.000000	31.000000	78.000000	41.000000	...

8 rows × 24 columns

In [5]:
```
df.drop(["Date"], axis = 1, inplace = True)
df.drop(["PRCP"], axis = 1, inplace = True)
df.drop(["STA"], axis = 1, inplace = True)
df.drop(["Precip"], axis = 1, inplace = True)
df.drop(["Snowfall"], axis = 1, inplace = True)
df.drop(["PoorWeather"], axis = 1, inplace = True)
df.drop(["WindGustSpd"], axis = 1, inplace = True)
df.drop(["MeanTemp"], axis = 1, inplace = True)
df.drop(["YR"], axis = 1, inplace = True)
df.drop(["MO"], axis = 1, inplace = True)
df.drop(["DA"], axis = 1, inplace = True)
df.drop(["DR"], axis = 1, inplace = True)
df.drop(["SPD"], axis = 1, inplace = True)
df.drop(["SND"], axis = 1, inplace = True)
df.drop(["FT"], axis = 1, inplace = True)
df.drop(["FTI"], axis = 1, inplace = True)
df.drop(["FB"], axis = 1, inplace = True)
df.drop(["ITH"], axis = 1, inplace = True)
df.drop(["TSHDSBRSGF"], axis = 1, inplace = True)
df.drop(["PGT"], axis = 1, inplace = True)
df.drop(["SD3"], axis = 1, inplace = True)
df.drop(["RHX"], axis = 1, inplace = True)
df.drop(["RHN"], axis = 1, inplace = True)
df.drop(["RVG"], axis = 1, inplace = True)
df.drop(["WTE"], axis = 1, inplace = True)
df.drop(["MAX"], axis = 1, inplace = True)
df.drop(["MIN"], axis = 1, inplace = True)
df.drop(["MEA"], axis = 1, inplace = True)
df.drop(["SNF"], axis = 1, inplace = True)
df.head()
```

Out[5]:

	MaxTemp	MinTemp
0	25.555556	22.222222
1	28.888889	21.666667
2	26.111111	22.222222
3	26.666667	22.222222
4	26.666667	21.666667

In [6]:
```
df.isnull().values.any()
```

Out[6]: False

Vista estadística del dataset

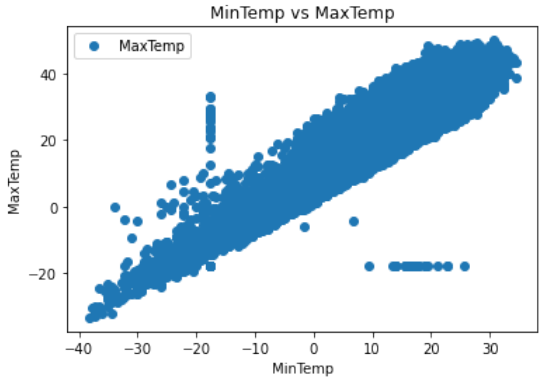
Visualización del conjunto de datos

In [7]:
```
print("max MaxTemp: ",df['MaxTemp'].max())
print("min MaxTemp: ",df['MaxTemp'].min())
print("max MinTemp: ",df['MinTemp'].max())
print("min MinTemp: ",df['MinTemp'].min())
```

```
max MaxTemp:  50.0
min MaxTemp:  -33.33333333
max MinTemp:  34.44444444
min MinTemp:  -38.33333333
```

Graficaremos nuestros puntos de datos en un diagrama en dos dimensiones para ilustrar nuestro dataset y ver si manualmente podemos encontrar alguna relación entre los datos usando el siguiente código:

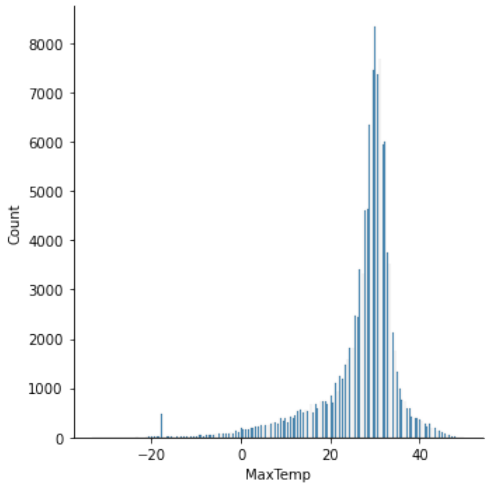
In [8]:
```
df.plot(x='MinTemp', y='MaxTemp', style='o')
plt.title('MinTemp vs MaxTemp')
plt.xlabel('MinTemp')
plt.ylabel('MaxTemp')
plt.show()
```



Hemos tomado “MinTemp” y “MaxTemp” para hacer nuestro análisis. Arriba hay un gráfico en 2 dimensiones entre MinTemp y MaxTemp. Vamos a chequear la temperatura promedio máxima y una vez la ploteamos podemos observar que la temperatura promedio máxima está entre cerca de 25 y 35

In [9]:
```
plt.figure(figsize=(15,10))
plt.tight_layout()
seabornInstance.displot(df['MaxTemp'])
```

Out[9]:
```
<seaborn.axisgrid.FacetGrid at 0x183011f1970>

<Figure size 1080x720 with 0 Axes>
```



La temperatura promedio máxima está entre 25 y 35 grados.

Nuestro siguiente paso es dividir nuestros datos en **"atributos"** y **"etiquetas"**. Los atributos son las **variables independientes**, mientras que las etiquetas son las **variables dependientes** cuyos valores se deben predecir.

En nuestro conjunto de datos, sólo tenemos dos columnas. Queremos predecir el **"MaxTemp"** dependiendo del **"MinTemp"** registrado. Por lo tanto, nuestro conjunto de atributos consistirá en la columna "MinTemp" que se almacena en la variable X, y la etiqueta será la columna "MaxTemp" que se almacena en la variable y.

In [10]:
```
X = df['MinTemp'].values.reshape(-1,1)
y = df['MaxTemp'].values.reshape(-1,1)
df_aux = pd.DataFrame({'X': X.flatten(), 'y': y.flatten()})
df_aux
```

Out[10]:

	X	y
0	22.222222	25.555556
1	21.666667	28.888889
2	22.222222	26.111111
3	22.222222	26.666667
4	21.666667	26.666667
...	...	...
119035	18.333333	28.333333
119036	18.333333	29.444444
119037	18.333333	28.333333
119038	18.333333	28.333333
119039	17.222222	29.444444

119040 rows × 2 columns

A continuación, dividimos el 80% de los datos al conjunto de entrenamiento mientras que el 20% de los datos al conjunto de pruebas usando el código de abajo. La variable test_size es nos permite definir la proporción del conjunto de pruebas.

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
df_aux = pd.DataFrame({'X_train': X_train.flatten(), 'y_train': y_train.flatten()})
df_aux.head()
```

Out[11]:

	X_train	y_train
0	22.222222	27.222222
1	17.777778	32.777778
2	-9.444444	8.888889
3	22.222222	31.666667
4	18.888889	35.000000

In [12]:
```
df_aux = pd.DataFrame({'X_test': X_test.flatten(), 'y_test': y_test.flatten()})
df_aux.head()
```

Out[12]:

	X_test	y_test
0	25.000000	28.888889
1	21.111111	31.111111
2	17.222222	27.222222
3	22.222222	28.888889
4	5.555556	23.333333

Después de dividir los datos en conjuntos de entrenamiento y pruebas, finalmente, es el momento de entrenar nuestro algoritmo. Para ello, necesitamos importar la clase **LinearRegression**, instanciarla y llamar el método **fit()** junto con nuestros datos de entrenamiento.

In [13]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[13]: `LinearRegression()`

Como hemos discutido, el modelo de regresión lineal básicamente encuentra el mejor valor para la intercepción y la pendiente, lo que resulta en la línea que mejor se ajusta a los datos. Para ver el valor de la intercepción y la pendiente calculado por el algoritmo de regresión lineal para nuestro conjunto de datos.

In [14]:
```
'''
Para obtener el interceptor
'''
print(regressor.intercept_)
```

```
[10.66185201]
```

In [15]:
```
'''
Para obtener la pendiente
'''
print(regressor.coef_)
```

```
[[0.92033997]]
```

El resultado debe ser aproximadamente 10.66185201 y 0.92033997 respectivamente. Esto significa que por cada unidad de cambio en la temperatura mínima, el cambio en la temperatura máxima es de alrededor de 0,92%. Ahora que hemos entrenado nuestro algoritmo, es hora de hacer algunas predicciones. Para ello, utilizaremos los datos de nuestras pruebas y veremos con qué precisión nuestro algoritmo predice la puntuación porcentual.

In [16]:
```
y_pred = regressor.predict(X_test)
```

In [17]:
```
df_aux = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
df_aux
```

Out[17]:

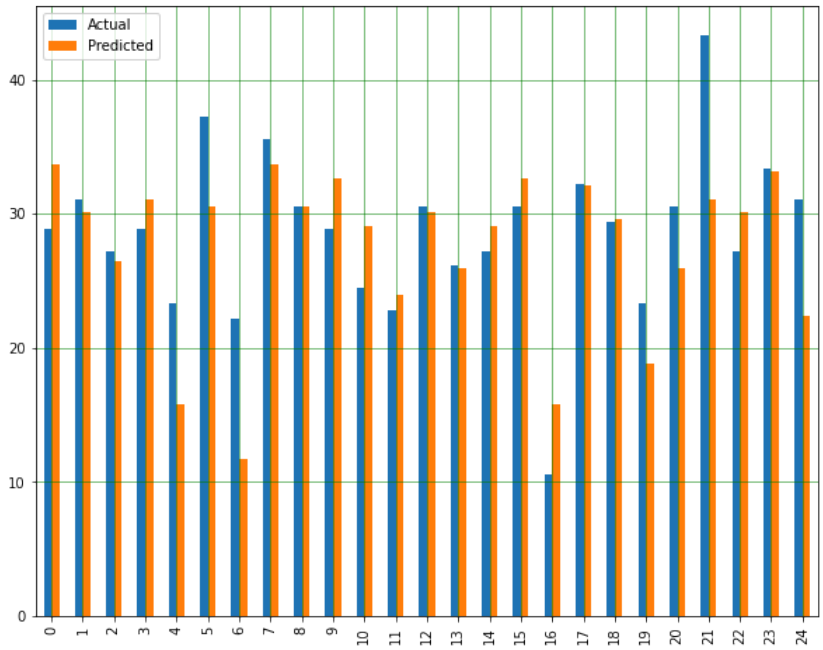
	Actual	Predicted
0	28.888889	33.670351
1	31.111111	30.091251
2	27.222222	26.512151
3	28.888889	31.113851
4	23.333333	15.774852
...	...	...
23803	32.777778	32.136451
23804	32.222222	29.068651
23805	31.111111	32.647751
23806	31.111111	30.602551
23807	36.666667	31.625151

23808 rows × 2 columns

Comparación del valor real y el predecido

También podemos visualizar el resultado de la comparación como un gráfico de barras usando el siguiente código. Como el número de registros es enorme, para fines de representación vamos a tomar 25 registros.

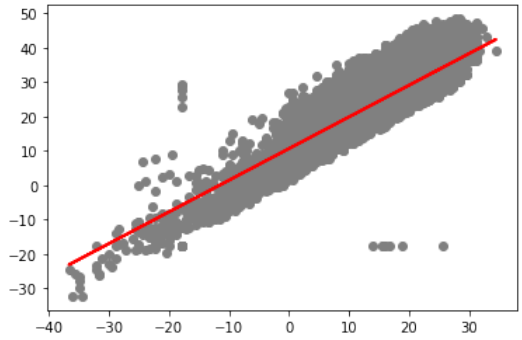
In [18]:
```
df1 = df_aux.head(25)
df1.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()

'''
Gráfico de barras mostrando la comparación de valores reales y predecidos
'''
```



Out[18]: '\nGráfico de barras mostrando la comparación de valores reales y predecidos \n'

Aunque nuestro modelo no es muy preciso, los porcentajes predichos se acercan a los reales. Trazaremos nuestra línea recta con los datos de la prueba:

In [19]:
```
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```



Predicción vs datos de prueba

La línea recta del gráfico anterior muestra que nuestro algoritmo es correcto. El paso final es evaluar el desempeño del algoritmo. Este paso es especialmente importante para comparar el rendimiento de los diferentes algoritmos en un determinado dataset. Para los algoritmos de regresión, se utilizan comúnmente tres métricas de evaluación:

In [20]:
```
print('Error Absoluto Medio:',metrics.mean_absolute_error(y_test, y_pred))
print('Error Cuadratico Medio:', metrics.mean_squared_error(y_test, y_pred))
print('Raíz del error cuadrático medio:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Error Absoluto Medio: 3.1993291783785835
Error Cuadratico Medio: 17.631568097568532
Raíz del error cuadrático medio: 4.198996082109215
```

Se puede ver que el valor Raíz del error cuadrático medio 4,19, lo cual es más del 10% del valor medio de los porcentajes de toda la temperatura, es decir, 22,41. Esto significa que nuestro algoritmo no fue muy preciso pero aún así puede hacer predicciones razonablemente buenas.