Entrenar y evaluar modelos de regresión

Introducción

La regresión es donde los modelos predicen un número.

En el aprendizaje automático, el objetivo de la regresión es crear un modelo que pueda predecir un valor numérico cuantificable, como un precio, una cantidad, un tamaño u otro número escalar.

La regresión es una técnica estadística de fundamental importancia para la ciencia debido a su facilidad de interpretación, robustez y rapidez en el cálculo. Los modelos de regresión proporcionan una base excelente para comprender cómo funcionan las técnicas de aprendizaje automático más complejas.

En situaciones del mundo real, particularmente cuando hay pocos datos disponibles, los modelos de regresión son muy útiles para hacer predicciones. Por ejemplo, si una empresa que alquila bicicletas desea predecir el número esperado de alquileres en un día determinado en el futuro, un modelo de regresión puede predecir este número. Se podría crear un modelo utilizando datos existentes, como el número de bicicletas que se alquilaron en los días en los que también se registraron la temporada, el día de la semana, etc.

¿Qué es la regresión?

La regresión funciona estableciendo una relación entre las variables en los datos que representan características, conocidas como características, de lo que se observa y la variable que estamos tratando de predecir, conocida como etiqueta . Recuerde nuestra empresa que alquila bicicletas y quiere predecir el número esperado de alquileres en un día determinado. En este caso, las características incluyen elementos como el día de la semana, el mes, etc., mientras que la etiqueta es el número de alquileres de bicicletas.

Para entrenar el modelo, comenzamos con una muestra de datos que contiene las características, así como valores conocidos para la etiqueta, por lo que en este caso necesitamos datos históricos que incluyan fechas, condiciones climáticas y el número de alquileres de bicicletas.

Luego, dividiremos esta muestra de datos en dos subconjuntos:

Un conjunto de datos de entrenamiento al que aplicaremos un algoritmo que determina una función que encapsula la relación entre los valores de la característica y los valores de etiqueta conocidos. Un conjunto de datos de validación o prueba que podemos usar para evaluar el modelo usándolo para generar predicciones para la etiqueta y compararlas con los valores de etiqueta conocidos reales. El uso de datos históricos con valores de etiqueta conocidos para entrenar un modelo hace que la regresión sea un ejemplo de aprendizaje automático supervisado .

Un simple ejemplo Tomemos un ejemplo sencillo para ver cómo funciona en principio el proceso de formación y evaluación. Suponga que simplificamos el escenario de modo que usamos una sola característica (temperatura diaria promedio) para predecir la etiqueta de alquiler de bicicletas.

Comenzamos con algunos datos que incluyen valores conocidos para la función de temperatura diaria promedio y la etiqueta de alquiler de bicicletas.

UN SIMPLE EJEMPLO Temperatura Alquileres 56 115 61 126 67 137 72 140 76 152 82 156 54 114 62 129 Ahora seleccionaremos al azar cinco de estas observaciones y las usaremos para entrenar un modelo de regresión. Cuando hablamos de 'entrenar un modelo', lo que queremos decir es encontrar una función (una ecuación matemática; llamémosla f) que pueda usar la función de temperatura (que llamaremos x) para calcular el número de alquileres (que llamaremos y). En otras palabras, necesitamos definir la siguiente función: f (x) = y .

Nuestro conjunto de datos de entrenamiento se ve así:

TABLA 2 X y 56 115 61 126 67 137 72 140 76 152 Vamos a empezar por el trazado de los valores de entrenamiento para X e Y en un gráfico:

Funciones de entrenamiento trazadas contra etiquetas

Ahora necesitamos ajustar estos valores a una función, permitiendo alguna variación aleatoria. Probablemente se puede ver que los puntos trazados forman una línea casi recta diagonal - en otras palabras, hay una aparente lineal relación entre X e Y , así que tenemos que encontrar una función lineal que es la mejor opción para la muestra de datos. Hay varios algoritmos que podemos usar para determinar esta función, que finalmente encontrará una línea recta con una variación general mínima de los puntos graficados; como esto:

Funciones de entrenamiento trazadas contra etiquetas con una línea de regresión

La línea representa una función lineal que se puede usar con cualquier valor de x para aplicar la pendiente de la línea y su intersección (donde la línea cruza el eje y cuando x es 0) para calcular y . En este caso, si extendimos la línea hacia la izquierda, encontraríamos que cuando x es 0, y es alrededor de 20, y la pendiente de la línea es tal que por cada unidad de x que se mueve hacia la derecha, y aumenta alrededor de 1,7. Por tanto, nuestra función f se puede calcular como 20 + 1,7 x .

Ahora que hemos definido nuestra función predictiva, podemos usarla para predecir etiquetas para los datos de validación que retenemos y comparar los valores predichos (que normalmente indicamos con el símbolo ŷ , o "y-hat") con los valores reales conocidos. valores y .

TABLA 3 X y ŷ 82 156 159,4 54 114 111,8 62 129 125,4 Veamos cómo se comparan los valores de yy ŷ en una gráfica:

características de validación trazadas contra etiquetas predichas y reales

Los puntos graficados que están en la línea de la función son los valores ŷ predichos calculados por la función, y los otros puntos graficados son los valores y reales .

Hay varias formas en que podemos medir la varianza entre los valores predichos y reales, y podemos usar estas métricas para evaluar qué tan bien predice el modelo.

Nota

El aprendizaje automático se basa en estadísticas y matemáticas, y es importante conocer los términos específicos que usan los estadísticos y matemáticos (y, por lo tanto, los científicos de datos). Puede pensar en la diferencia entre un valor de etiqueta predicho y el valor de etiqueta real como una medida de error . Sin embargo, en la práctica, los valores "reales" se basan en observaciones de muestra (que a su vez pueden estar sujetas a alguna variación aleatoria). Para dejar en claro que estamos comparando un valor predicho (ŷ) con un valor observado (y), nos referimos a la diferencia entre ellos como los residuos. Podemos resumir los residuos de todas las predicciones de datos de validación para calcular la pérdida general en el modelo como una medida de su rendimiento predictivo.

Una de las formas más comunes de medir la pérdida es elevar al cuadrado los residuos individuales, sumar los cuadrados y calcular la media. Cuadrar los residuos tiene el efecto de basar el cálculo en valores absolutos (ignorando si la diferencia es negativa o positiva) y dar más peso a las diferencias más grandes. Esta métrica se llama Error cuadrático medio .

Para nuestros datos de validación, el cálculo se ve así:

CUADRO 4 y ŷ y - ŷ (y - ŷ) 2 156 159,4 -3,4 11.56 114 111,8 2.2 4.84 129 125,4 3.6 12,96 Suma ∑ 29,36 Significar X 9,79 Entonces, la pérdida para nuestro modelo basado en la métrica MSE es 9.79.

Entonces, ¿eso es bueno? Es difícil saberlo porque el valor de MSE no se expresa en una unidad de medida significativa. Sabemos que cuanto menor es el valor, menor pérdida hay en el modelo; y por lo tanto, mejor es la predicción. Esto hace que sea una métrica útil para comparar dos modelos y encontrar el que funcione mejor.

A veces, es más útil expresar la pérdida en la misma unidad de medida que el valor de la etiqueta predicha, en este caso, la cantidad de alquileres. Es posible hacer esto calculando la raíz cuadrada del MSE, que produce una métrica conocida, como era de esperar, como Root Mean Squared Error (RMSE).

√9,79 = 3,13

Entonces, el RMSE de nuestro modelo indica que la pérdida es un poco más de 3, lo que puede interpretar libremente como que significa que, en promedio, las predicciones incorrectas son incorrectas en alrededor de 3 alquileres.

Hay muchas otras métricas que se pueden utilizar para medir la pérdida en una regresión. Por ejemplo, R 2 (R-Squared) (a veces conocido como coeficiente de determinación) es la correlación entre x y y al cuadrado. Esto produce un valor entre 0 y 1 que mide la cantidad de

Regresión

Las técnicas de aprendizaje automático supervisado implican entrenar un modelo para operar en un conjunto de características y predecir una etiqueta utilizando un conjunto de datos que incluye algunos valores de etiqueta ya conocidos. El proceso de entrenamiento ajusta las características a las etiquetas conocidas para definir una función general que se puede aplicar a las nuevas características para las que las etiquetas son desconocidas y predecirlas. Puede pensar en esta función de esta manera, en la que y representa la etiqueta que queremos predecir y x representa las características que usa el modelo para predecirlo.

y = F (X)

En la mayoría de los casos, x es en realidad un vector que consta de múltiples valores de características, por lo que, para ser un poco más precisos, la función podría expresarse así:

y = F ([X 1 , X 2 , X 3 , . . .])

El objetivo de entrenar el modelo es encontrar una función que realice algún tipo de cálculo de los valores de x que produzca el resultado y . Hacemos esto aplicando un algoritmo de aprendizaje automático que intenta ajustar los valores de x a un cálculo que produce y con una precisión razonable para todos los casos en el conjunto de datos de entrenamiento.

Hay muchos algoritmos de aprendizaje automático para el aprendizaje supervisado y se pueden dividir en dos tipos:

Algoritmos de regresión : algoritmos que predicen unvalor de y que es un valor numérico, como el precio de una casa o el número de transacciones de venta. Algoritmos de clasificación : algoritmos que predicen a qué categoría o clase pertenece una observación. Elvalor y en un modelo de clasificación es un vector de valores de probabilidad entre 0 y 1, uno para cada clase, que indica la probabilidad de que la observación pertenezca a cada clase. En este cuaderno, nos centraremos en la regresión , utilizando un ejemplo basado en un estudio real en el que se recopilaron datos para un esquema de uso compartido de bicicletas y se usaron para predecir el número de alquileres según la estacionalidad y las condiciones climáticas. Usaremos una versión simplificada del conjunto de datos de ese estudio.

Cita : Los datos utilizados en este ejercicio se derivan de Capital Bikeshare y se utilizan de acuerdo con el acuerdo de licencia publicado . Explore los datos El primer paso en cualquier proyecto de aprendizaje automático es explorar los datos que usará para entrenar un modelo. El objetivo de esta exploración es intentar comprender las relaciones entre sus atributos; en particular, cualquier correlación aparente entre las características y la etiqueta que su modelo intentará predecir. Esto puede requerir algo de trabajo para detectar y corregir problemas en los datos (como tratar con valores perdidos, errores o valores atípicos), derivar nuevas columnas de características mediante la transformación o combinación de características existentes (un proceso conocido como ingeniería de características), normalizar características numéricas (valores que puede medir o contar) para que estén en una escala similar y codifiquen

características categóricas (valores que representan categorías discretas) como indicadores numéricos.

Comencemos cargando los datos de uso compartido de bicicletas como un Pandas DataFrame y viendo las primeras filas.

```
import pandas as pd

# load the training dataset
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machir
bike_data = pd.read_csv('daily-bike-share.csv')
bike_data.head()
```

```
--2021-10-13 19:55:57--  https://raw.githubusercontent.com/MicrosoftDocs/mslearn-
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.13
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.1
HTTP request sent, awaiting response... 200 OK
Length: 48800 (48K) [text/plain]
Saving to: ‘daily-bike-share.csv’

daily-bike-share.cs 100%[===================>]  47.66K  --.-KB/s    in 0.01s

2021-10-13 19:55:57 (4.59 MB/s) - ‘daily-bike-share.csv’ saved [48800/48800]
```

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	
0	1	1/1/2011	1	0	1	0	6	0	2	0.3
1	2	1/2/2011	1	0	1	0	0	0	2	0.3
2	3	1/3/2011	1	0	1	0	1	1	1	0.1
3	4	1/4/2011	1	0	1	0	2	1	1	0.2
4	5	1/5/2011	1	0	1	0	3	1	1	0.2

Los datos constan de las siguientes columnas:

instant : un identificador de fila único dteday : la fecha en la que se observaron los datos; en este caso, los datos se recopilaron diariamente; por lo que hay una fila por fecha. temporada : un valor codificado numéricamente que indica la temporada (1: primavera, 2: verano, 3: otoño, 4: invierno) año : el año del estudio en el que se realizó la observación (el estudio se llevó a cabo durante dos años; el año 0 representa 2011 y el año 1 representa 2012) mth : El mes calendario en el que se realizó la observación (1: enero ... 12: diciembre) feriado : un valor binario que indica si la observación se realizó o no en un feriado público) weekday : el día de la semana en el que se realizó la observación (0: domingo ... 6: sábado) día laborable : un valor binario que indica si el día es un día laborable (no un fin de semana o feriado) weathersit : Un valor categórico que indica la situación meteorológica (1: despejado, 2: niebla / nube, 3: lluvia ligera / nieve, 4: lluvia intensa / granizo / nieve / niebla) temp : La temperatura en grados Celsius (normalizada) atemp : La temperatura aparente ("se siente como") en grados Celsius (normalizada) zumbido : el nivel

de humedad (normalizado) velocidad del viento : la velocidad del viento (normalizada) alquileres : el número de alquileres de bicicletas registrados. En este conjunto de datos, los alquileres representan la etiqueta (el valor de y) que nuestro modelo debe estar entrenado para predecir. Las otras columnas son características potenciales (valores x).

Como se mencionó anteriormente, puede realizar alguna ingeniería de funciones para combinar o derivar nuevas funciones. Por ejemplo, agreguemos una nueva columna llamada día al marco de datos extrayendo el componente de día de la columna dteday existente . La nueva columna representa el día del mes del 1 al 31.

```
1 bike_data['day'] = pd.DatetimeIndex(bike_data['dteday']).day
2 bike_data.head(32)
```

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
0	1	1/1/2011	1	0	1	0	6	0	2
1	2	1/2/2011	1	0	1	0	0	0	2
2	3	1/3/2011	1	0	1	0	1	1	1
3	4	1/4/2011	1	0	1	0	2	1	1
4	5	1/5/2011	1	0	1	0	3	1	1
5	6	1/6/2011	1	0	1	0	4	1	1
6	7	1/7/2011	1	0	1	0	5	1	2
7	8	1/8/2011	1	0	1	0	6	0	2

Bien, comencemos nuestro análisis de los datos examinando algunas estadísticas descriptivas clave. Podemos usar el método de descripción del marco de datos para generarlos para las características numéricas, así como para la columna de etiquetas de alquileres .

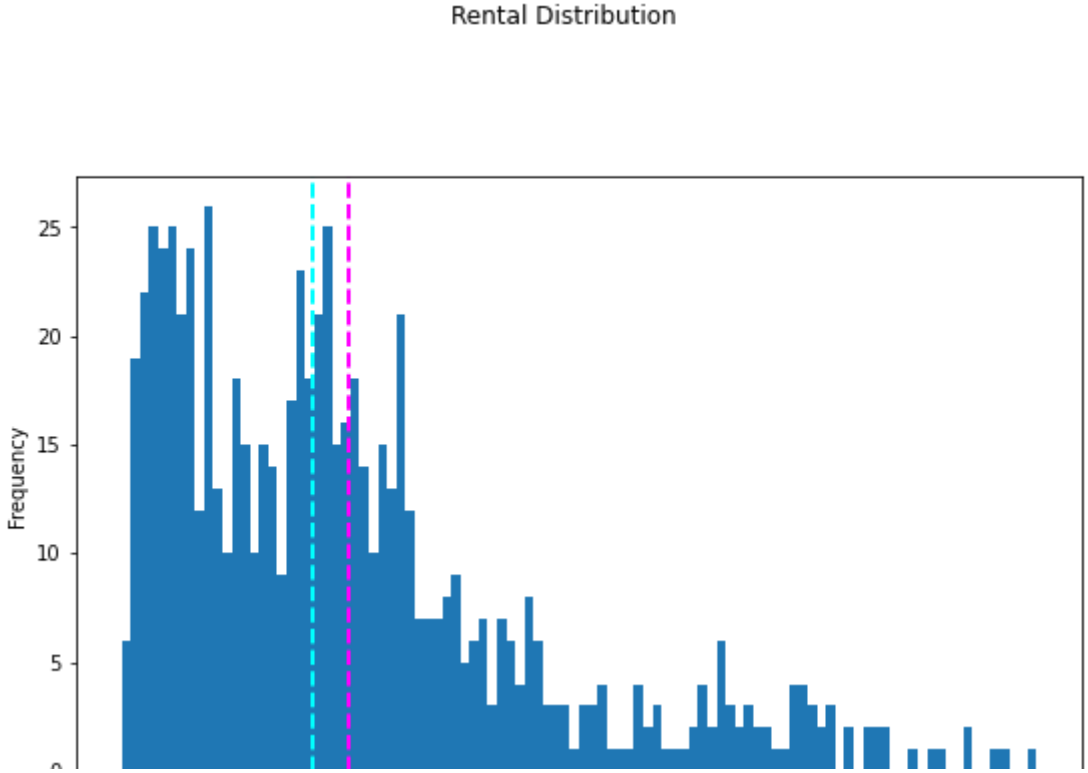
```
1 numeric_features = ['temp', 'atemp', 'hum', 'windspeed']
2 bike_data[numeric_features + ['rentals']].describe()
```

	temp	atemp	hum	windspeed	rentals
count	731.000000	731.000000	731.000000	731.000000	731.000000
mean	0.495385	0.474354	0.627894	0.190486	848.176471
std	0.183051	0.162961	0.142429	0.077498	686.622488
min	0.059130	0.079070	0.000000	0.022392	2.000000
25%	0.337083	0.337842	0.520000	0.134950	315.500000
50%	0.498333	0.486733	0.626667	0.180975	713.000000
75%	0.655417	0.608602	0.730209	0.233214	1096.000000
max	0.861667	0.840896	0.972500	0.507463	3410.000000

Las estadísticas revelan cierta información sobre la distribución de los datos en cada uno de los campos numéricos, incluido el número de observaciones (hay 731 registros), la media, la desviación estándar, los valores mínimo y máximo y los valores de cuartil (los valores de umbral para 25%, 50%, que también es la mediana y el 75% de los datos). A partir de esto, podemos ver que el número medio de alquileres diarios es de alrededor de 848; pero hay una desviación estándar comparativamente grande, lo que indica una gran variación en el número de alquileres por día.

Podríamos tener una idea más clara de la distribución de los valores de los alquileres visualizando los datos. Los tipos de gráficos comunes para visualizar distribuciones de datos numéricos son histogramas y diagramas de caja , así que usemos la biblioteca matplotlib de Python para crear uno de cada uno de estos para la columna de alquileres .

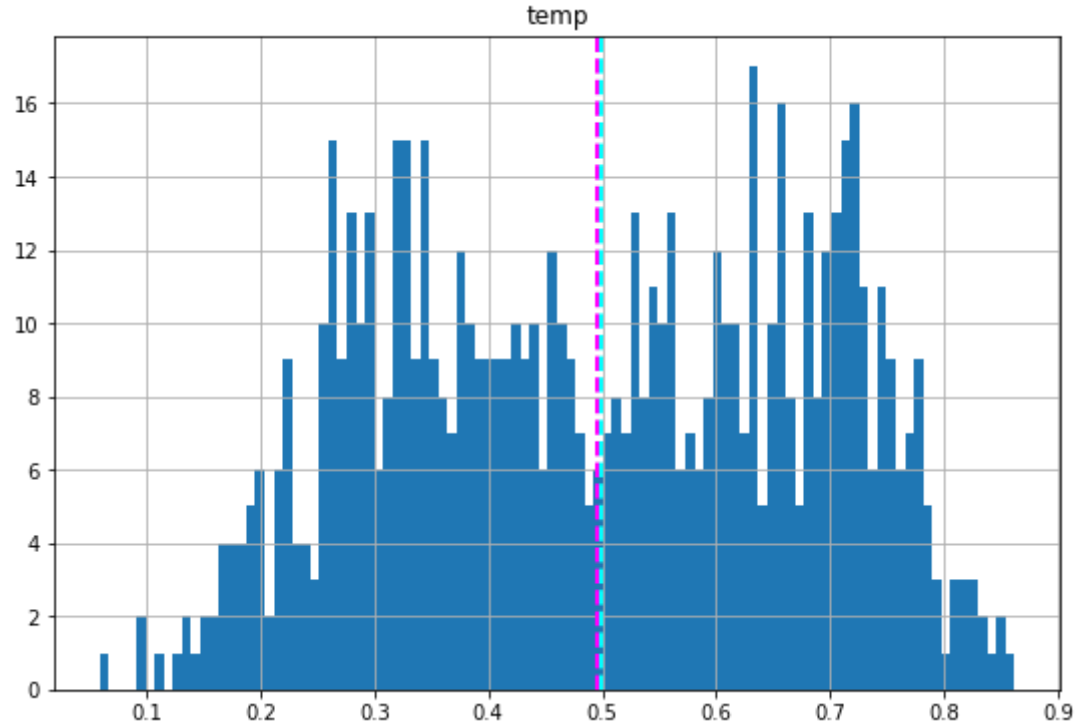
```
import pandas as pd
import matplotlib.pyplot as plt

# This ensures plots are displayed inline in the Jupyter notebook
%matplotlib inline

# Get the label column
label = bike_data['rentals']


# Create a figure for 2 subplots (2 rows, 1 column)
fig, ax = plt.subplots(2, 1, figsize = (9,12))

# Plot the histogram
ax[0].hist(label, bins=100)
ax[0].set_ylabel('Frequency')

# Add lines for the mean, median, and mode
ax[0].axvline(label.mean(), color='magenta', linestyle='dashed', linewidth=2)
ax[0].axvline(label.median(), color='cyan', linestyle='dashed', linewidth=2)

# Plot the boxplot
ax[1].boxplot(label, vert=False)
ax[1].set_xlabel('Rentals')

# Add a title to the Figure
fig.suptitle('Rental Distribution')

# Show the figure
fig.show()

```

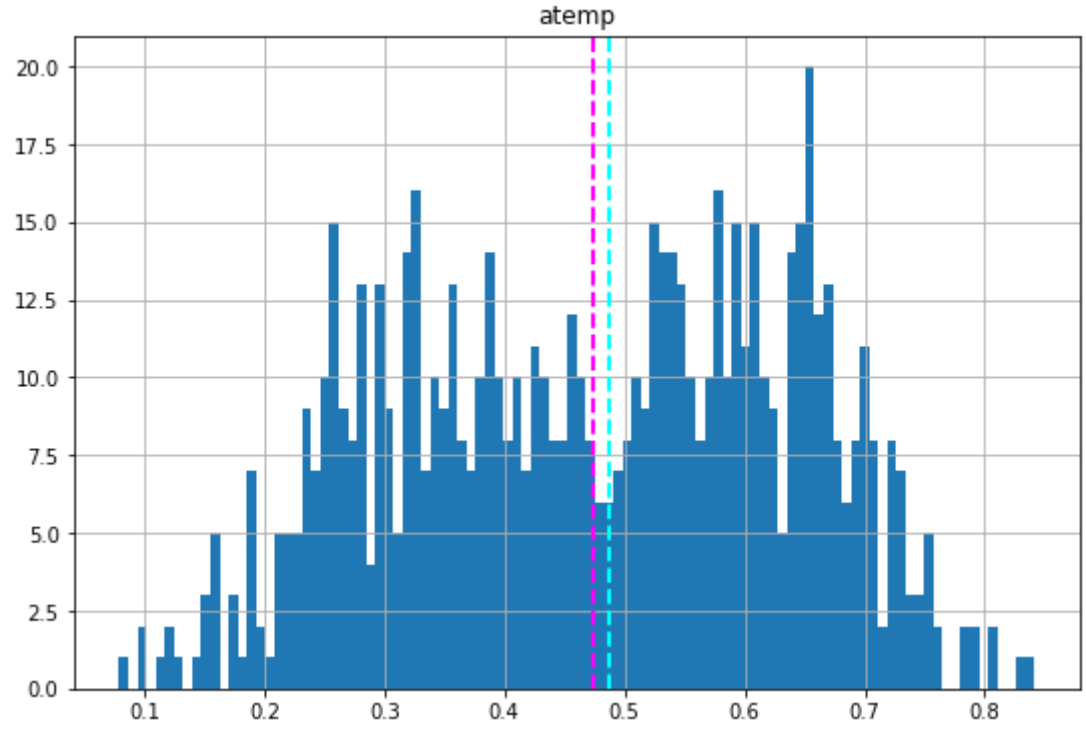


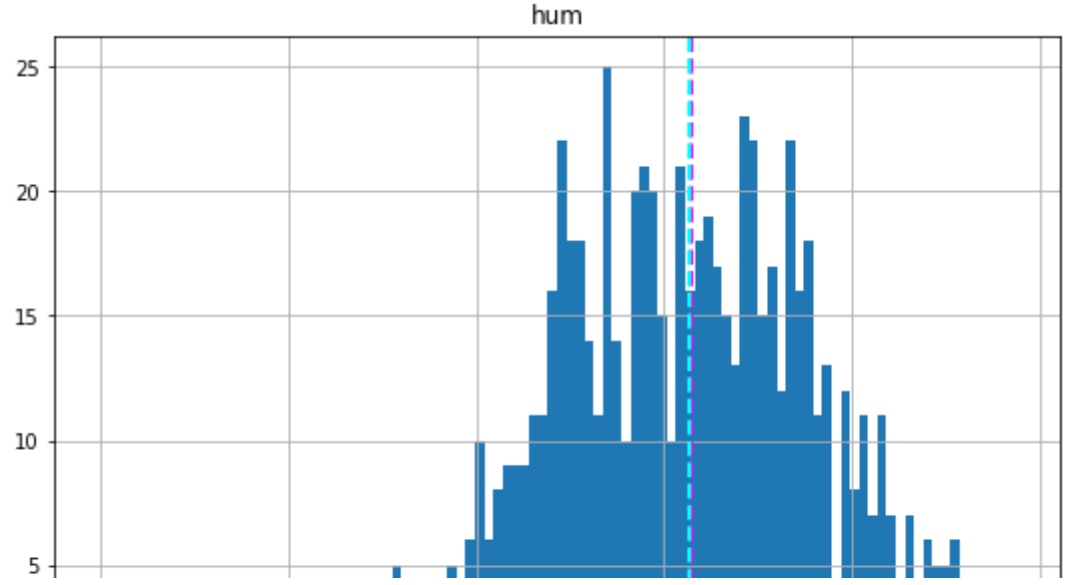
Las parcelas muestran que el número de alquileres diarios oscila entre 0 y poco más de 3.400. Sin embargo, el número medio (y mediano) de alquileres diarios está más cerca del extremo inferior de ese rango, con la mayoría de los datos entre 0 y alrededor de 2200 alquileres. Los pocos valores por encima de este se muestran en el diagrama de caja como círculos pequeños, lo que indica que son valores atípicos , en otras palabras, valores inusualmente altos o bajos más allá del rango típico de la mayoría de los datos.

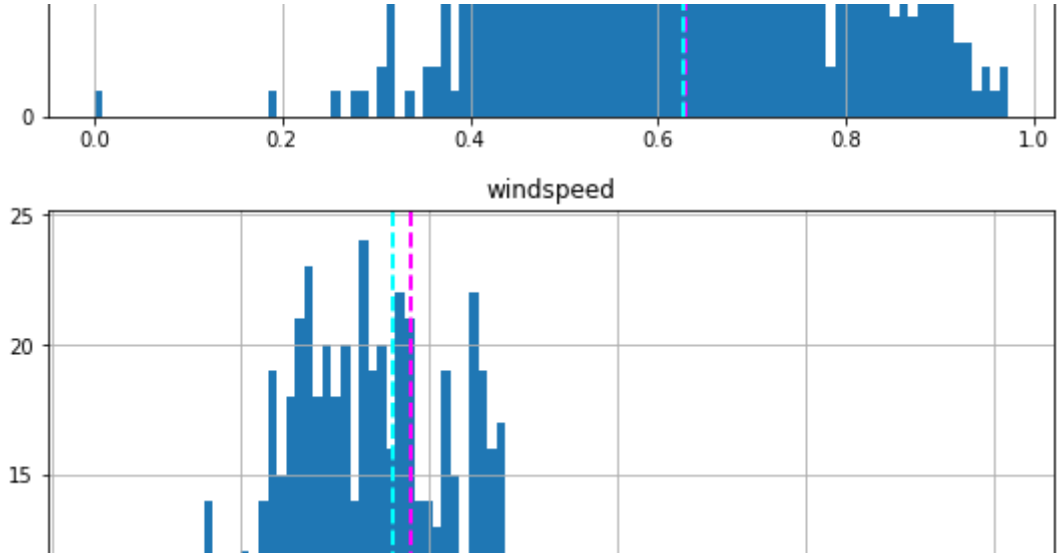
Podemos hacer el mismo tipo de exploración visual de las características numéricas. Creemos un histograma para cada uno de estos.

```
# Plot a histogram for each numeric feature
for col in numeric_features:
    fig = plt.figure(figsize=(9, 6))
    ax = fig.gca()
    feature = bike_data[col]
    feature.hist(bins=100, ax = ax)
    ax.axvline(feature.mean(), color='magenta', linestyle='dashed', linewidth=2)
    ax.axvline(feature.median(), color='cyan', linestyle='dashed', linewidth=2)
    ax.set_title(col)
plt.show()
```


temp
0
2
4
6
8
10
12
14
16
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


atemp
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

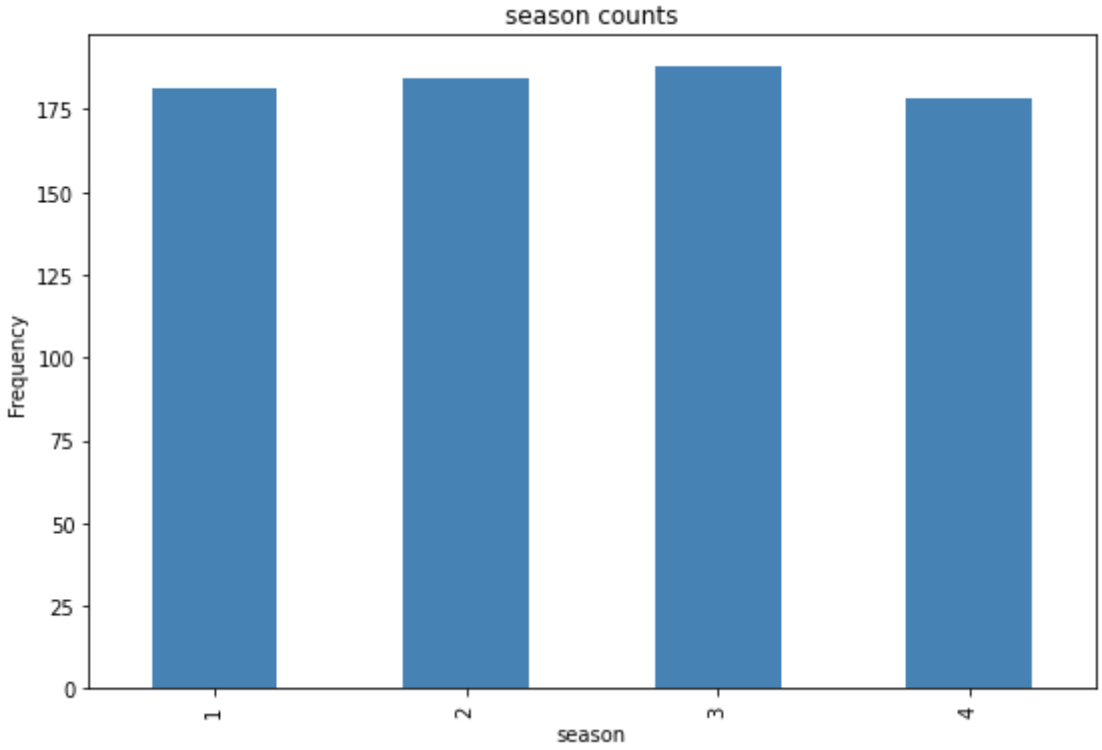

hum
5
10
15
20
25

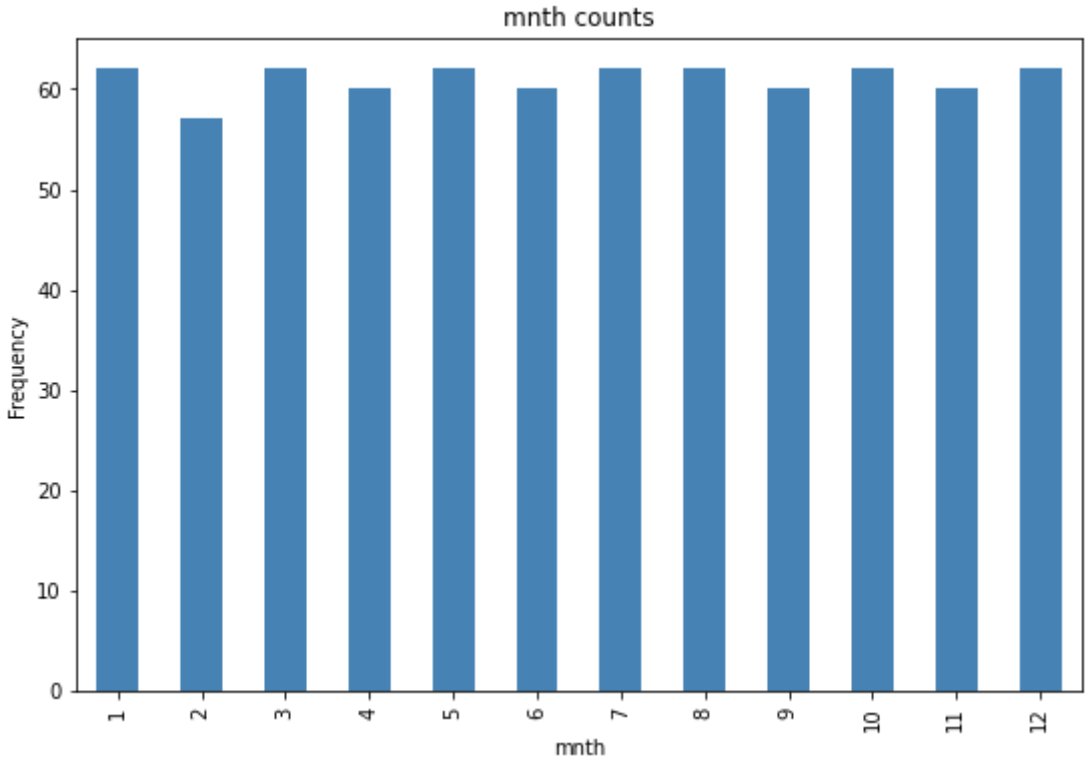


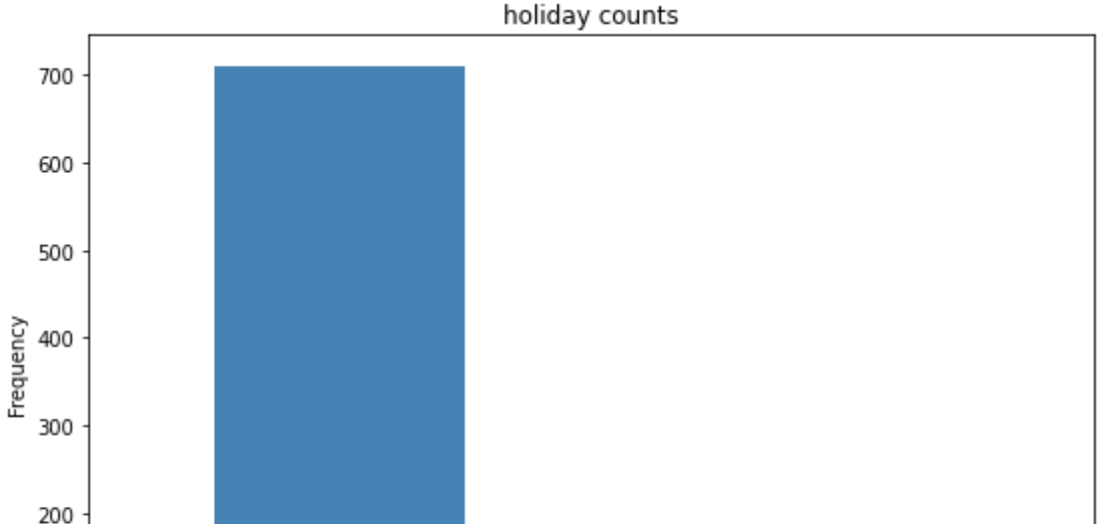
Las características numéricas parecen estar distribuidas de manera más normal , con la media y la mediana más cerca del centro del rango de valores, coincidiendo con los valores que ocurren con mayor frecuencia.

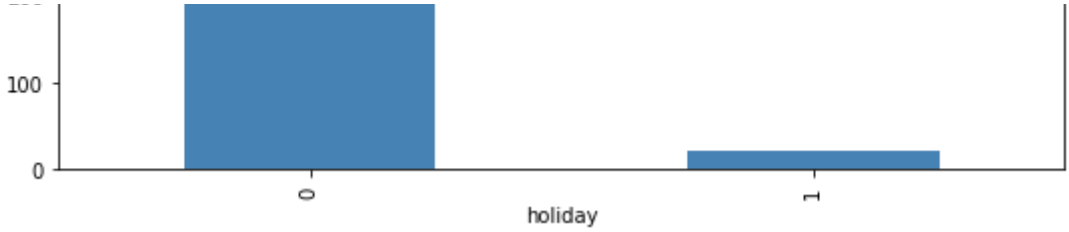
Hemos explorado la distribución de los valores numéricos en el conjunto de datos, pero ¿qué pasa con las características categóricas? Estos no son números continuos en una escala, por lo que no podemos usar histogramas; pero podemos trazar un gráfico de barras que muestre el recuento de cada valor discreto para cada categoría.

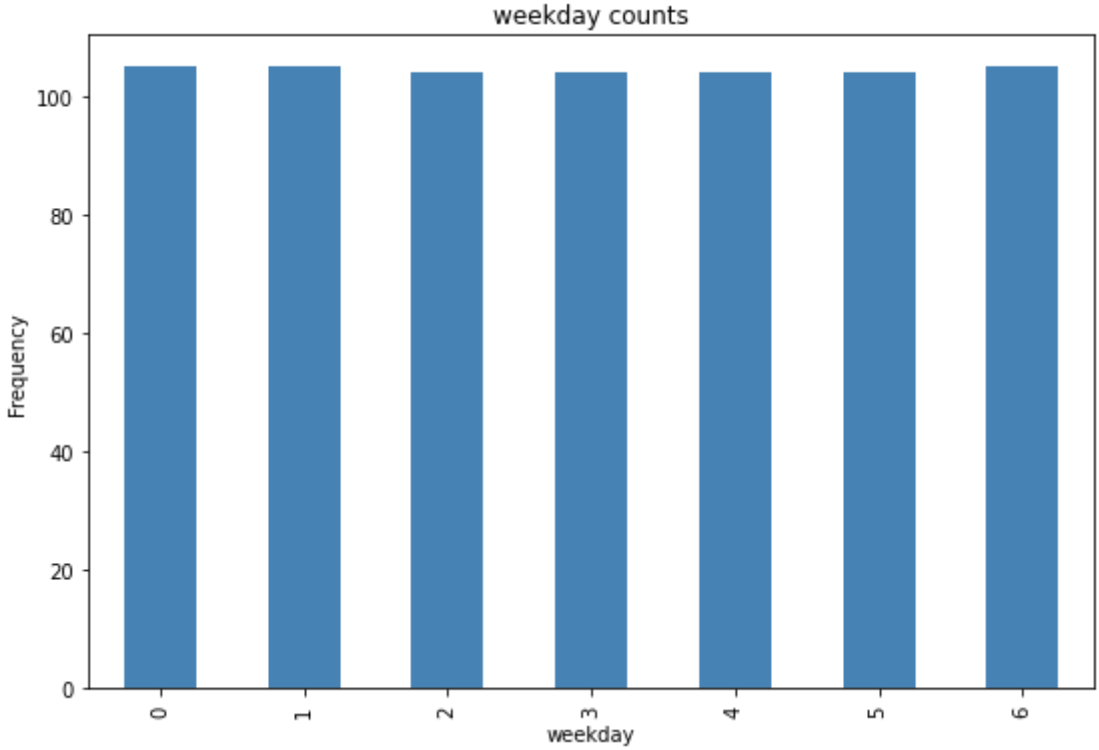
```
import numpy as np

# plot a bar plot for each categorical feature count
categorical_features = ['season','mnth','holiday','weekday','workingday','weathersit

for col in categorical_features:
    counts = bike_data[col].value_counts().sort_index()
    fig = plt.figure(figsize=(9, 6))
    ax = fig.gca()
    counts.plot.bar(ax = ax, color='steelblue')
    ax.set_title(col + ' counts')
    ax.set_xlabel(col)
    ax.set_ylabel("Frequency")
plt.show()

```

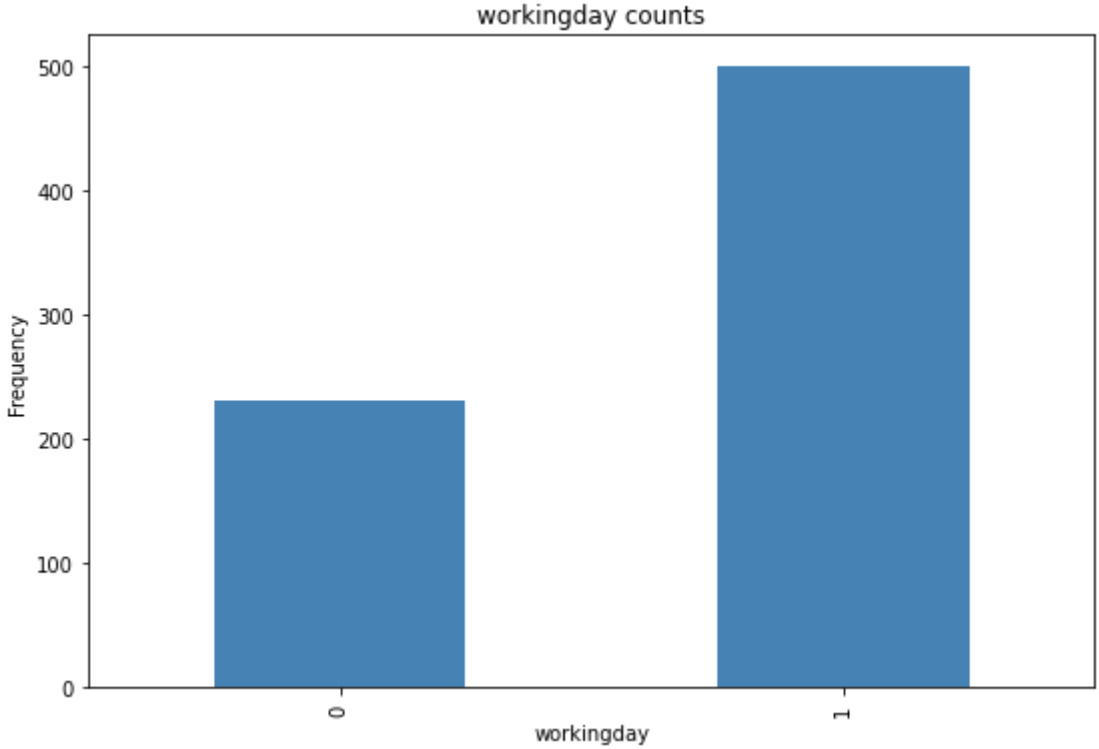

season counts
Frequency
0
25
50
75
100
125
150
175
1
2
3
4
season

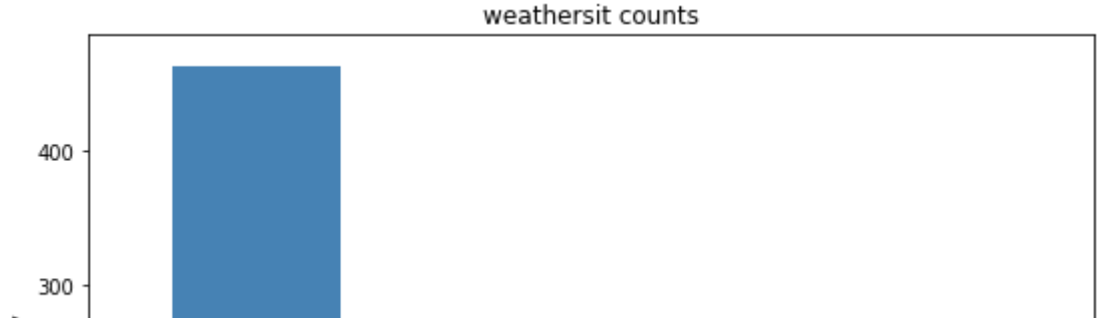

mnth counts
Frequency
0
10
20
30
40
50
60
1
2
3
4
5
6
7
8
9
10
11
12
mnth

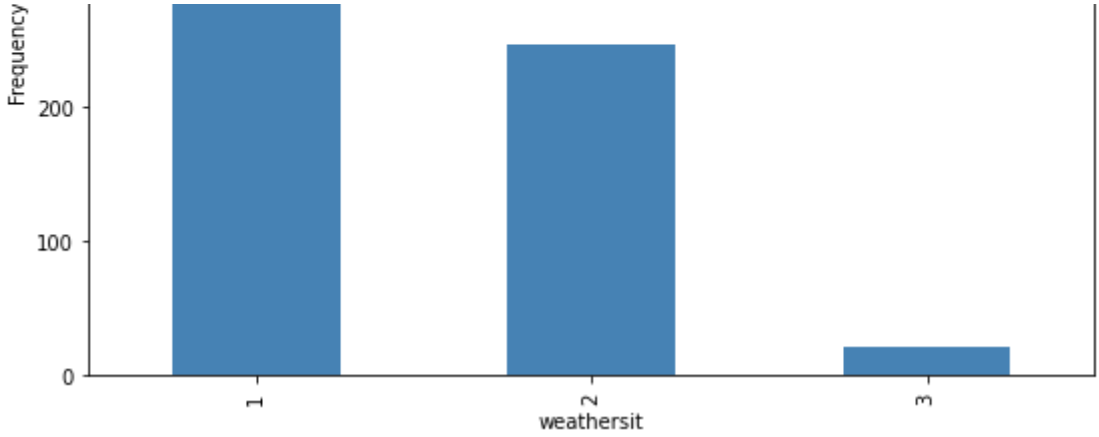

holiday counts
Frequency
200
300
400
500
600
700

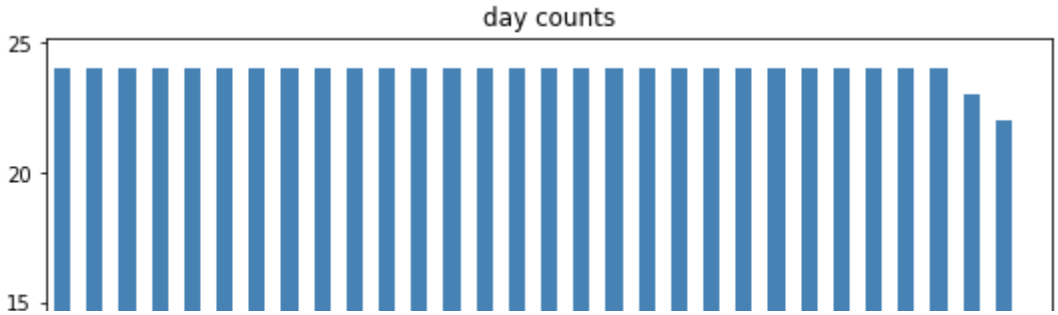

100
0
0
1
holiday


weekday counts
Frequency
100
80
60
40
20
0
0
1
2
3
4
5
6
weekday


workingday counts
Frequency
500
400
300
200
100
0
0
1
workingday


weathersit counts
400
300



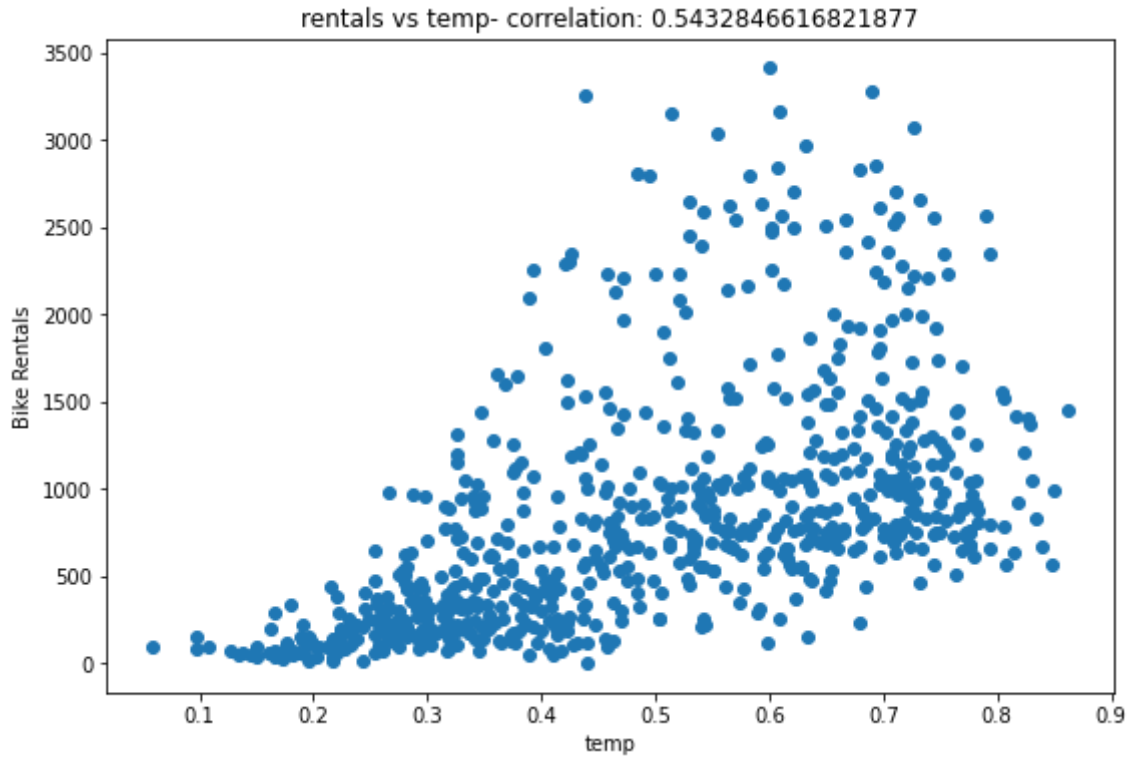


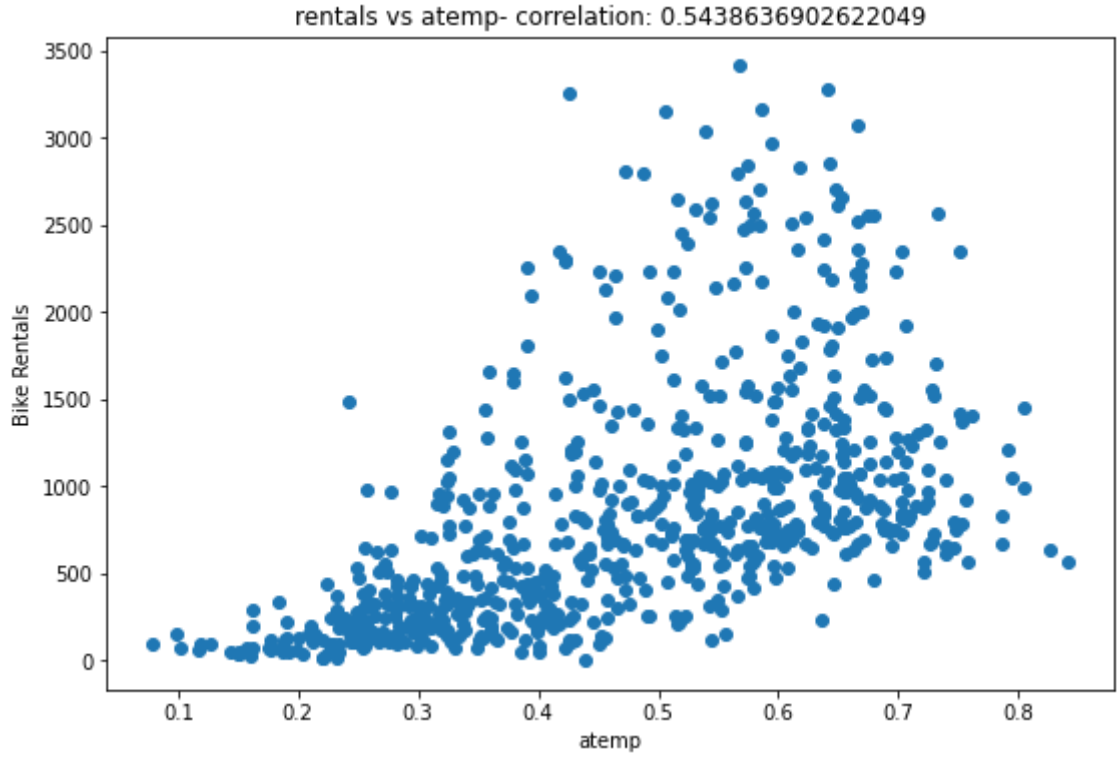
Muchas de las características categóricas muestran una distribución más o menos uniforme (lo que significa que hay aproximadamente el mismo número de filas para cada categoría). Las excepciones a esto incluyen:

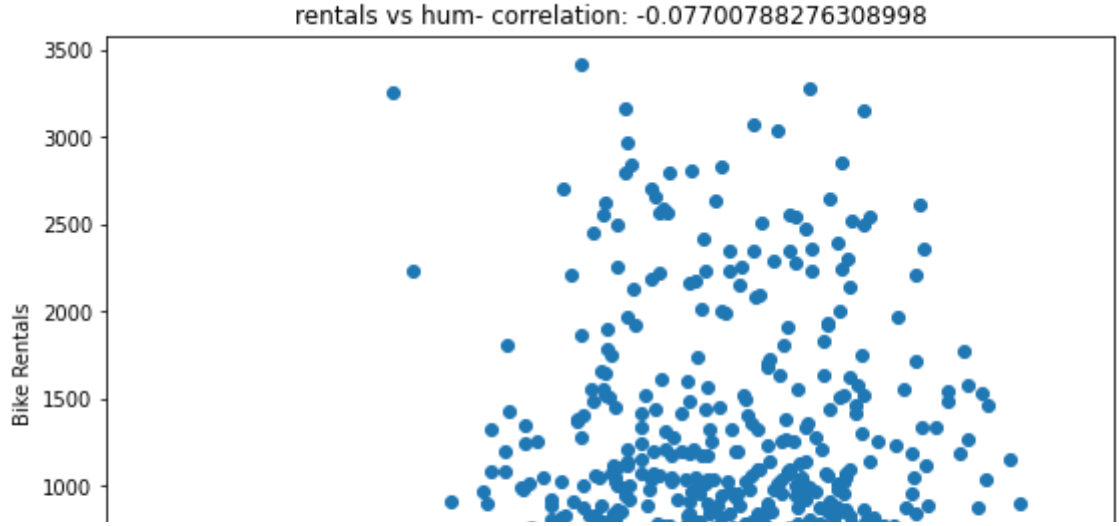
feriados : hay muchos menos días que son feriados que días que no lo son. jornada laboral : hay más días laborables que no laborables. weathersit : La mayoría de los días son de categoría 1 (despejados), siendo la categoría 2 (niebla y nubes) la siguiente más común. Hay comparativamente pocos días de categoría 3 (lluvia ligera o nieve) y ningún día de categoría 4 (lluvia intensa, granizo o niebla). Ahora que sabemos algo sobre la distribución de los datos en nuestras columnas, podemos comenzar a buscar relaciones entre las características y la etiqueta de alquileres que queremos poder predecir.

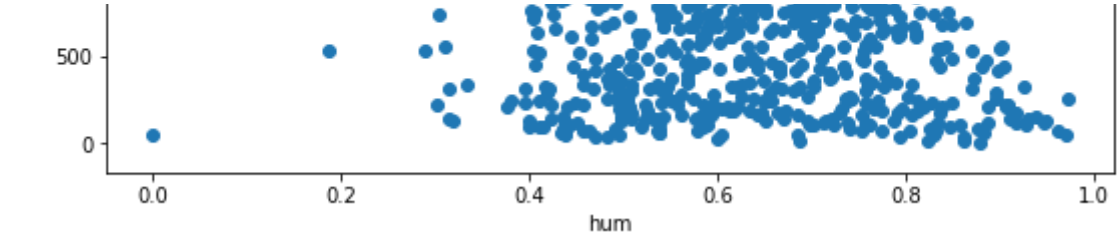
Para las características numéricas, podemos crear diagramas de dispersión que muestren la intersección de los valores de características y etiquetas. También podemos calcular la estadística de correlación para cuantificar la relación aparente.

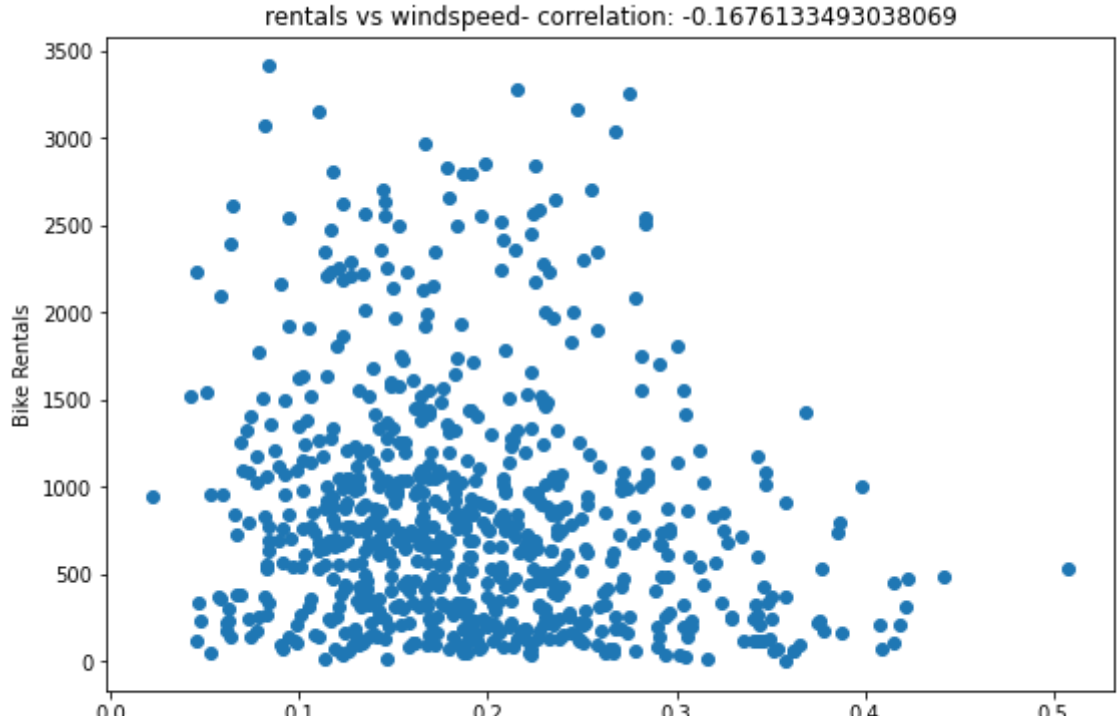
```
for col in numeric_features:
    fig = plt.figure(figsize=(9, 6))
    ax = fig.gca()
    feature = bike_data[col]
    label = bike_data['rentals']
    correlation = feature.corr(label)
    plt.scatter(x=feature, y=label)
    plt.xlabel(col)
    plt.ylabel('Bike Rentals')
    ax.set_title('rentals vs ' + col + '- correlation: ' + str(correlation))
plt.show()

```


rentals vs temp- correlation: 0.5432846616821877
Bike Rentals
temp
0
500
1000
1500
2000
2500
3000
3500
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


rentals vs atemp- correlation: 0.5438636902622049
Bike Rentals
atemp
0
500
1000
1500
2000
2500
3000
3500
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


rentals vs hum- correlation: -0.07700788276308998
Bike Rentals
1000
1500
2000
2500
3000
3500



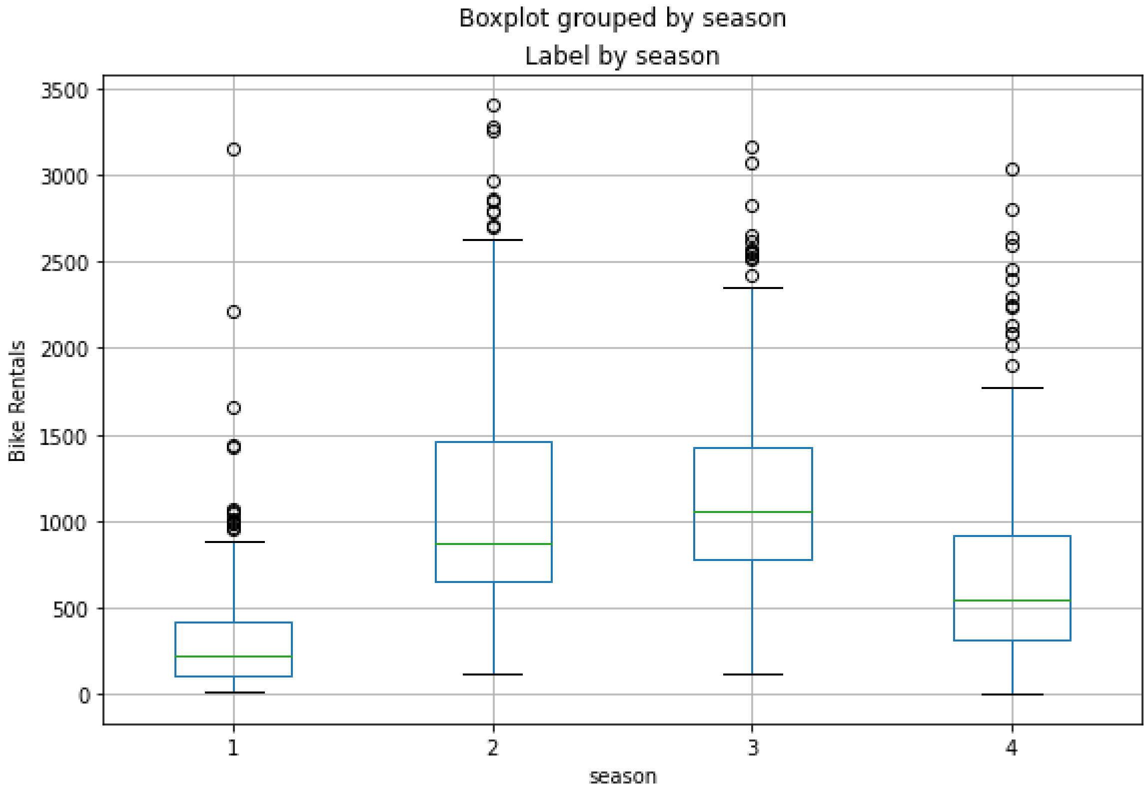


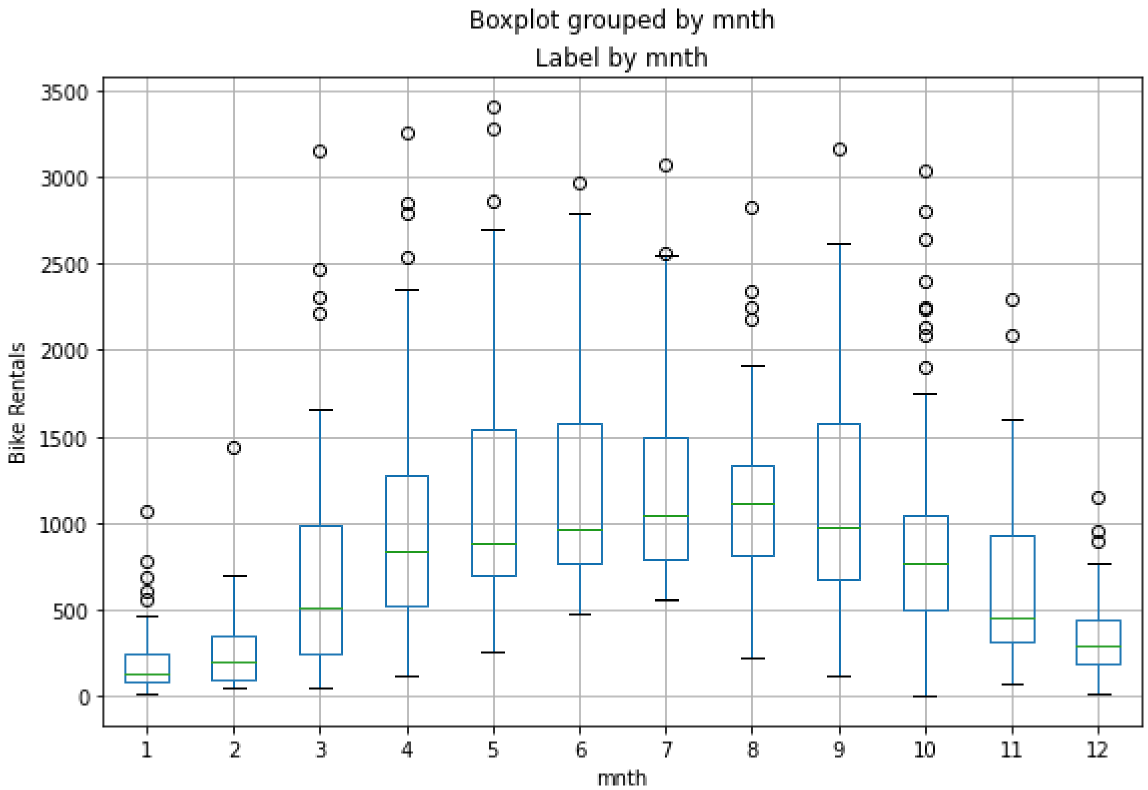
Los resultados no son concluyentes, pero si observa de cerca los diagramas de dispersión de temperatura y temperatura , puede ver una tendencia diagonal vaga que muestra que los recuentos de alquileres más altos tienden a coincidir con temperaturas más altas; y un valor de correlación de poco más de 0,5 para ambas características respalda esta observación. Por el contrario, las gráficas de zumbido y velocidad del viento muestran una correlación ligeramente negativa, lo que indica que hay menos alquileres en días con alta humedad o velocidad del viento.

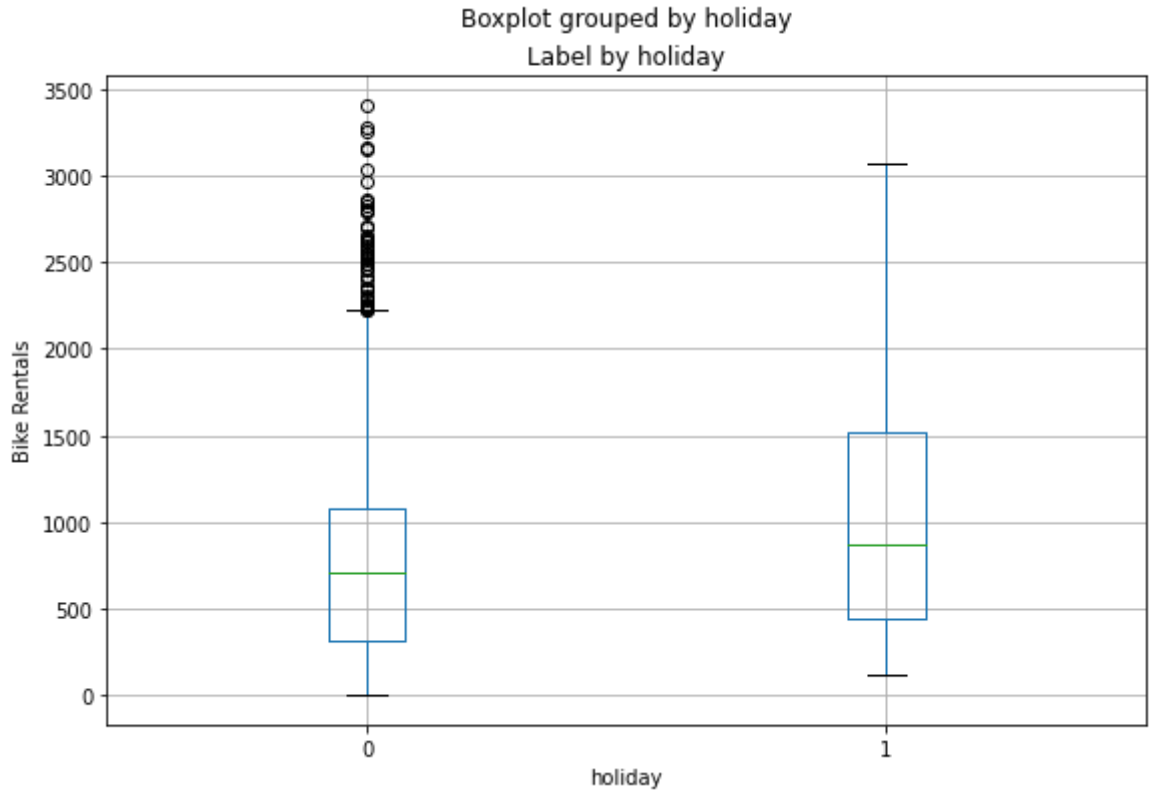
Ahora comparemos las características categóricas con la etiqueta. Haremos esto mediante la creación de diagramas de caja que muestren la distribución de los recuentos de alquiler para cada categoría.

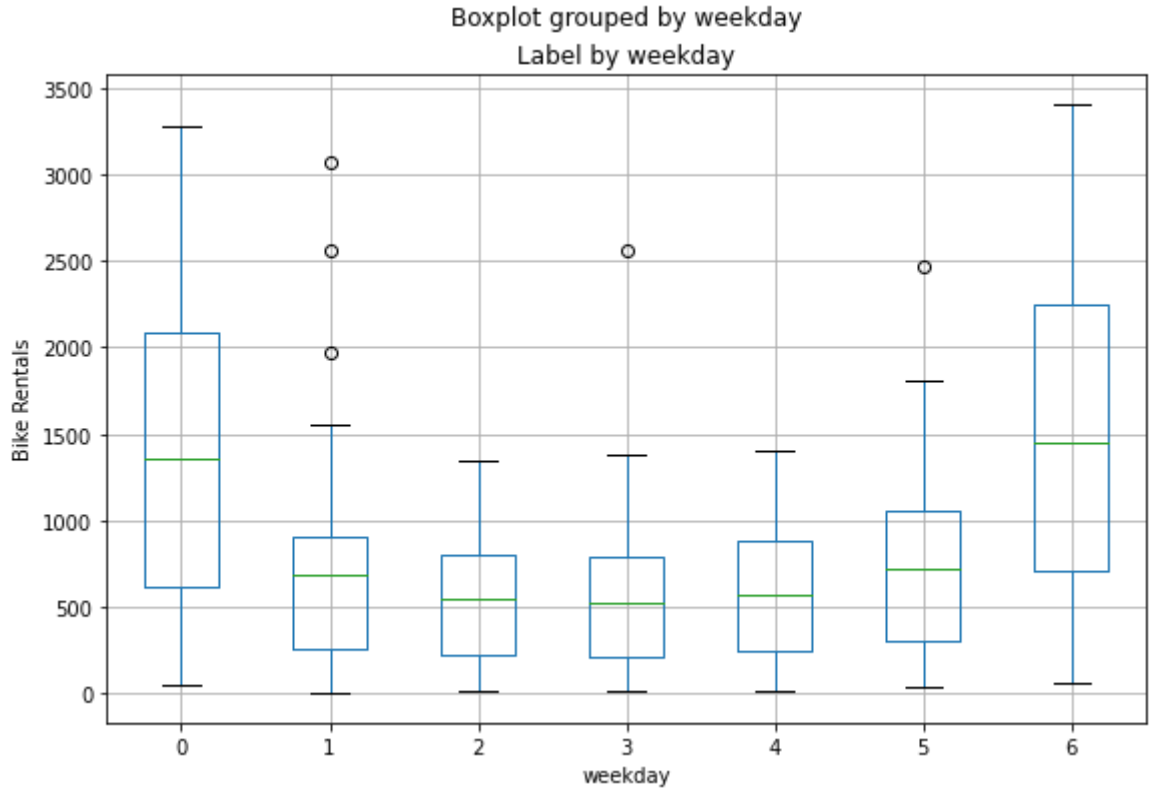
```
# plot a boxplot for the label by each categorical feature
for col in categorical_features:
    fig = plt.figure(figsize=(9, 6))
    ax = fig.gca()
    bike_data.boxplot(column = 'rentals', by = col, ax = ax)
    ax.set_title('Label by ' + col)
    ax.set_ylabel("Bike Rentals")
plt.show()
```

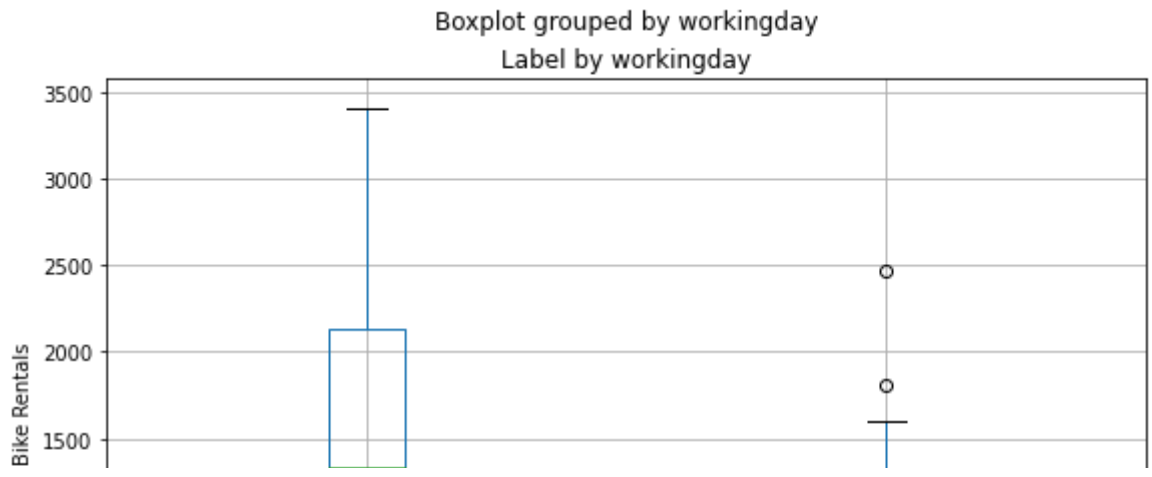
```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecat
  return array(a, dtype, copy=False, order=order)
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecat
  return array(a, dtype, copy=False, order=order)
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecat
  return array(a, dtype, copy=False, order=order)
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecat
  return array(a, dtype, copy=False, order=order)
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecat
  return array(a, dtype, copy=False, order=order)
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecat
  return array(a, dtype, copy=False, order=order)
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecat
  return array(a, dtype, copy=False, order=order)
```

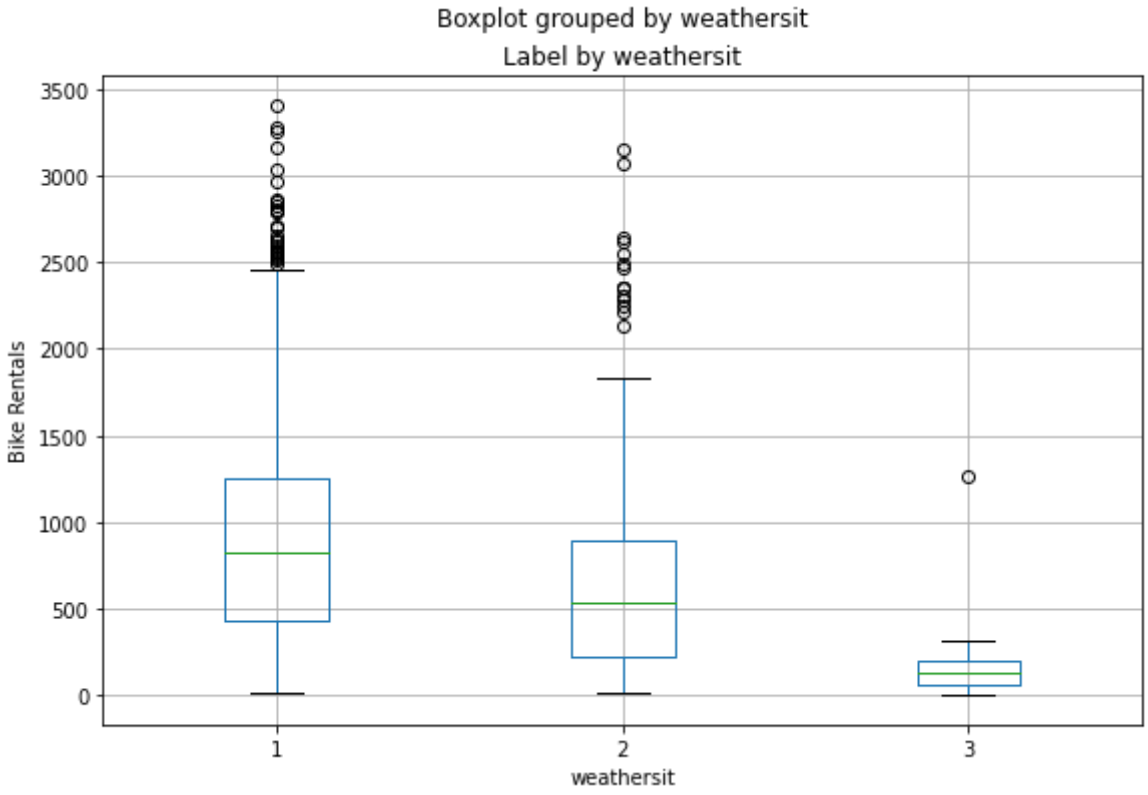


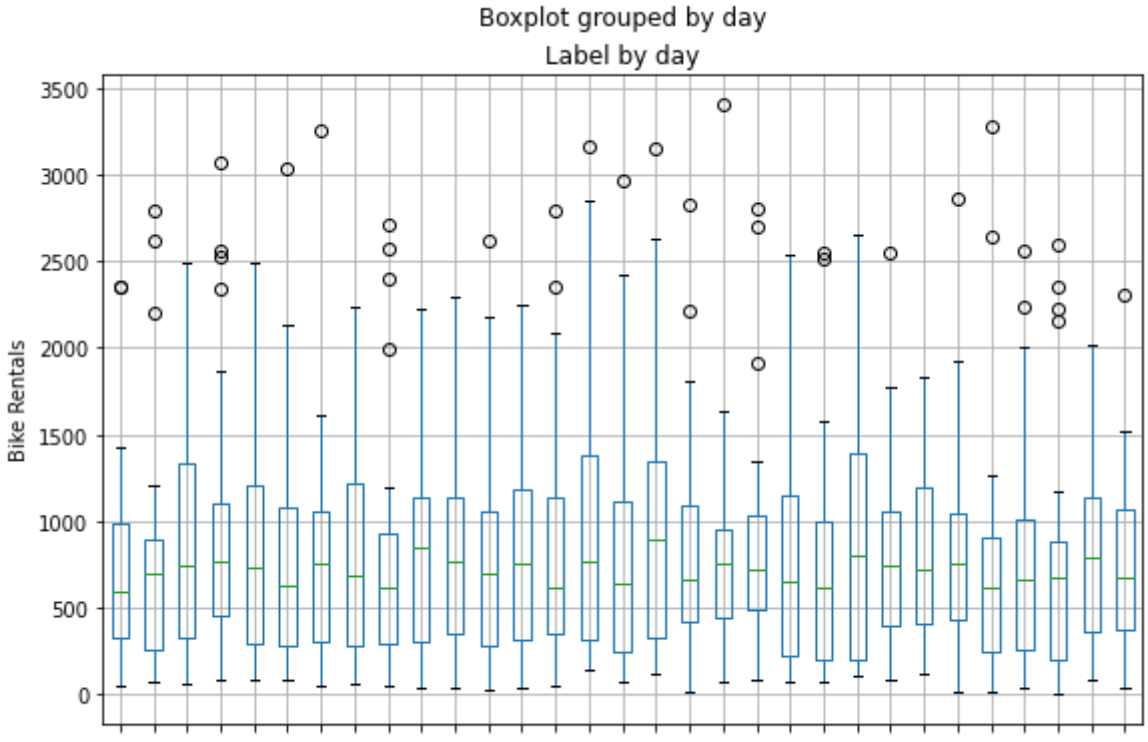



Boxplot grouped by holiday
Label by holiday
Bike Rentals
3500
3000
2500
2000
1500
1000
500
0
0
1
holiday


Boxplot grouped by weekday
Label by weekday
Bike Rentals
3500
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
weekday


Boxplot grouped by workingday
Label by workingday
Bike Rentals
3500
3000
2500
2000
1500





Las gráficas muestran alguna variación en la relación entre algunos valores de categoría y alquileres. Por ejemplo, existe una clara diferencia en la distribución de los alquileres los fines de semana (día laborable 0 o 6) y los de la semana laboral (día laborable 1 a 5). Del mismo modo,

existen diferencias notables entre las categorías de días festivos y días laborables . Existe una tendencia notable que muestra diferentes distribuciones de alquiler en los meses de verano y otoño en comparación con los meses de primavera e invierno. La categoría weathersit también parece marcar la diferencia en la distribución del alquiler. La función de día que creamos para el día del mes muestra poca variación, lo que indica que probablemente no predice la cantidad de alquileres.

Entrenar un modelo de regresión Ahora que hemos explorado los datos, es hora de usarlos para entrenar un modelo de regresión que use las características que hemos identificado como potencialmente predictivas para predecir la etiqueta de alquileres . Lo primero que debemos hacer es separar las características que queremos usar para entrenar el modelo de la etiqueta que queremos que prediga.

```
1 # Separate features and labels
2 X, y = bike_data[['season','mnth', 'holiday','weekday','workingday','weathersit','te
3 print('Features:',X[:10], '\nLabels:', y[:10], sep='\n')
```

```
Features:
[[1.        1.        0.        6.        0.        2.        0.344167
  0.363625  0.805833  0.160446 ]
 [1.        1.        0.        0.        0.        2.        0.363478
  0.353739  0.696087  0.248539 ]
 [1.        1.        0.        1.        1.        1.        0.196364
  0.189405  0.437273  0.248309 ]
 [1.        1.        0.        2.        1.        1.        0.2
  0.212122  0.590435  0.160296 ]
 [1.        1.        0.        3.        1.        1.        0.226957
  0.22927   0.436957  0.1869   ]
 [1.        1.        0.        4.        1.        1.        0.204348
  0.233209  0.518261  0.0895652]
 [1.        1.        0.        5.        1.        2.        0.196522
  0.208839  0.498696  0.168726 ]
 [1.        1.        0.        6.        0.        2.        0.165
  0.162254  0.535833  0.266804 ]
 [1.        1.        0.        0.        0.        1.        0.138333
  0.116175  0.434167  0.36195  ]
 [1.        1.        0.        1.        1.        1.        0.150833
  0.150888  0.482917  0.223267 ]]

Labels:
[331 131 120 108  82  88 148  68  54  41]
```

Después de separar el conjunto de datos, ahora tenemos numerosas matrices llamadas X que contienen las características e y que contienen las etiquetas.

Nos pudimos entrenar un modelo utilizando todos los datos; pero es una práctica común en el aprendizaje supervisado dividir los datos en dos subconjuntos; un conjunto (normalmente más grande) con el que entrenar el modelo y un conjunto de "retención" más pequeño con el que validar el modelo entrenado. Esto nos permite evaluar qué tan bien se desempeña el modelo cuando se usa con el conjunto de datos de validación comparando las etiquetas predichas con

las etiquetas conocidas. Es importante dividir los datos al azar (en lugar de decir, tomar el primer 70% de los datos para entrenar y guardar el resto para validación). Esto ayuda a garantizar que los dos subconjuntos de datos sean estadísticamente comparables (por lo que validamos el modelo con datos que tienen una distribución estadística similar a los datos en los que se entrenó).

Para dividir aleatoriamente los datos, usaremos la función train_test_split en la biblioteca scikit-learn . Esta biblioteca es uno de los paquetes de aprendizaje automático más utilizados para Python.

```
1 from sklearn.model_selection import train_test_split
2
3 # Split data 70%-30% into training set and test set
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
5
6 print ('Training Set: %d rows\nTest Set: %d rows' % (X_train.shape[0], X_test.shape[
```

```
Training Set: 511 rows
Test Set: 220 rows
```

Ahora tenemos los siguientes cuatro conjuntos de datos:

X_train : los valores de características que usaremos para entrenar el modelo y_train : Las etiquetas correspondientes que usaremos para entrenar el modelo. X_test : los valores de características que usaremos para validar el modelo y_test : las etiquetas correspondientes que usaremos para validar el modelo Ahora estamos listos para entrenar un modelo ajustando un algoritmo de regresión adecuado a los datos de entrenamiento. Usaremos un algoritmo de regresión lineal , un punto de partida común para la regresión que funciona tratando de encontrar una relación lineal entre los valores X y la etiqueta y . El modelo resultante es una función que define conceptualmente una línea en la que se cruzan todas las combinaciones posibles de valores X e y.

En Scikit-Learn, los algoritmos de entrenamiento están encapsulados en estimadores , y en este caso usaremos el estimador LinearRegression para entrenar un modelo de regresión lineal.

```
1 # Train the model
2 from sklearn.linear_model import LinearRegression
3
4 # Fit a linear regression model on the training set
5 model = LinearRegression().fit(X_train, y_train)
6 print (model)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

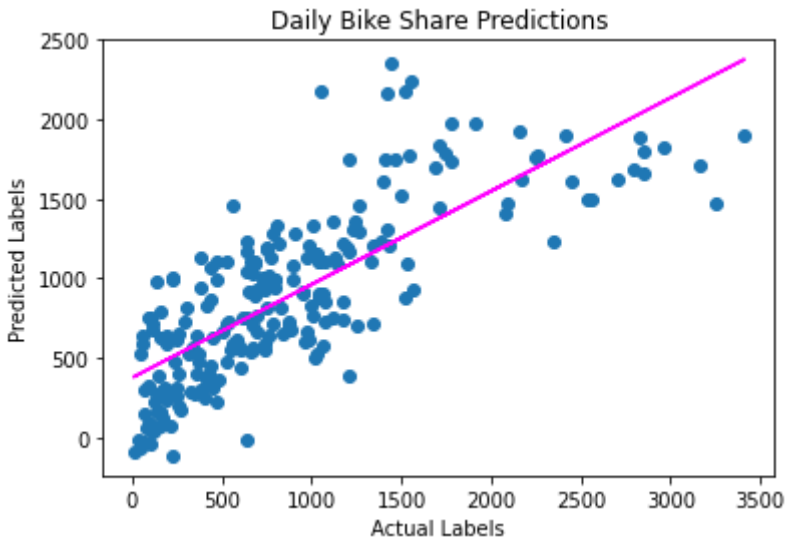
Evaluar el modelo entrenado Ahora que hemos entrenado el modelo, podemos usarlo para predecir los recuentos de alquiler de las funciones que retuvimos en nuestro conjunto de datos de validación. Luego, podemos comparar estas predicciones con los valores reales de la etiqueta para evaluar qué tan bien (¡o no!) Está funcionando el modelo.

```
import numpy as np

predictions = model.predict(X_test)
np.set_printoptions(suppress=True)
print('Predicted labels: ', np.round(predictions)[:10])
print('Actual labels   : ' ,y_test[:10])
```

```
Predicted labels:  [1896. 1184. 1007.  -28.  314.  385.  475.  590. 1476.  -22.]
Actual labels   :  [2418  754  222   47  244  145  240  555 3252   38]
```

Comparar cada predicción con su correspondiente valor real de "verdad fundamental" no es una forma muy eficiente de determinar qué tan bien está prediciendo el modelo. Veamos si podemos obtener una mejor indicación visualizando un diagrama de dispersión que compare las predicciones con las etiquetas reales. También superpondremos una línea de tendencia para tener una idea general de qué tan bien se alinean las etiquetas predichas con las etiquetas verdaderas.

```
import matplotlib.pyplot as plt

%matplotlib inline

plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```



Hay una tendencia diagonal definida, y las intersecciones de los valores predichos y reales generalmente siguen el camino de la línea de tendencia; pero hay una gran diferencia entre la

función ideal representada por la línea y los resultados. Esta varianza representa los residuos del modelo; en otras palabras, la diferencia entre la etiqueta predicha cuando el modelo aplica los coeficientes que aprendió durante el entrenamiento a los datos de validación y el valor real de la etiqueta de validación. Estos residuos, cuando se evalúan a partir de los datos de validación, indican el nivel de error esperado cuando el modelo se utiliza con datos nuevos para los que se desconoce la etiqueta.

Puede cuantificar los residuos calculando una serie de métricas de evaluación de uso común. Nos centraremos en los siguientes tres:

Error cuadrático medio (MSE) : la media de las diferencias cuadráticas entre los valores predichos y reales. Esto produce una métrica relativa en la que cuanto menor sea el valor, mejor será el ajuste del modelo. Error cuadrático medio (RMSE) : La raíz cuadrada del MSE. Esto produce una métrica absoluta en la misma unidad que la etiqueta (en este caso, el número de alquileres). Cuanto menor sea el valor, mejor será el modelo (en un sentido simplista, representa el número medio de alquileres en los que las predicciones son incorrectas). Coeficiente de determinación (generalmente conocido como R cuadrado o R2): una métrica relativa en la que cuanto mayor sea el valor, mejor será el ajuste del modelo. En esencia, esta métrica representa qué parte de la varianza entre los valores de etiqueta predichos y reales es capaz de explicar el modelo.

```
 1 from sklearn.metrics import mean_squared_error, r2_score
 2
 3 mse = mean_squared_error(y_test, predictions)
 4 print("MSE:", mse)
 5
 6 rmse = np.sqrt(mse)
 7 print("RMSE:", rmse)
 8
 9 r2 = r2_score(y_test, predictions)
10 print("R2:", r2)
```

```
MSE: 201972.55947035592
RMSE: 449.4135728595165
R2: 0.6040454736919191
```

Así que ahora hemos cuantificado la capacidad de nuestro modelo para predecir el número de alquileres. Definitivamente tiene cierto poder predictivo, ¡pero probablemente podamos hacerlo mejor!

Descubra nuevos modelos de regresión

Experimentando con modelos Los modelos de regresión a menudo se eligen porque funcionan con pequeñas muestras de datos, son robustos, fáciles de interpretar y existe una variedad.

La regresión lineal es la forma más simple de regresión, sin límite para el número de características utilizadas. La regresión lineal se presenta en muchas formas, a menudo

nombradas por la cantidad de características utilizadas y la forma de la curva que se ajusta.

Los árboles de decisión adoptan un enfoque paso a paso para predecir una variable. Si pensamos en nuestro ejemplo de bicicleta, el árbol de decisiones puede ser primero ejemplos divididos entre los que están durante la primavera / verano y el otoño / invierno, haga una predicción basada en el día de la semana. Los lunes de primavera / verano pueden tener una tarifa de alquiler de bicicletas de 100 por día, mientras que los lunes de otoño / invierno pueden tener una tarifa de alquiler de 20 por día.

Los algoritmos de conjuntos construyen no solo un árbol de decisiones, sino una gran cantidad de árboles, lo que permite mejores predicciones sobre datos más complejos. Los algoritmos de conjuntos, como Random Forest, se utilizan ampliamente en el aprendizaje automático y la ciencia debido a sus sólidas capacidades de predicción.

Los científicos de datos a menudo experimentan con el uso de diferentes modelos. En el siguiente ejercicio, experimentaremos con diferentes tipos de modelos para comparar cómo funcionan con los mismos datos.

Regresión: experimentar con modelos adicionales En el cuaderno anterior, usamos modelos de regresión simple para observar la relación entre las características de un conjunto de datos de alquiler de bicicletas. En este cuaderno, experimentaremos con modelos más complejos para mejorar nuestro rendimiento de regresión.

Comencemos cargando los datos de uso compartido de bicicletas como un Pandas DataFrame y viendo las primeras filas. También dividiremos nuestros datos en conjuntos de datos de prueba y entrenamiento.

```
# Import modules we'll need for this notebook
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# load the training dataset
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machir
bike_data = pd.read_csv('daily-bike-share.csv')
bike_data['day'] = pd.DatetimeIndex(bike_data['dteday']).day
numeric_features = ['temp', 'atemp', 'hum', 'windspeed']
categorical_features = ['season','mnth','holiday','weekday','workingday','weathersit
bike_data[numeric_features + ['rentals']].describe()
print(bike_data.head())


# Separate features and labels
# After separating the dataset, we now have numpy arrays named **X** containing the
X, y = bike_data[['season','mnth', 'holiday','weekday','workingday','weathersit','te

```

```
24 # Split data 70%-30% into training set and test set
25 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
26
27 print ('Training Set: %d rows\nTest Set: %d rows' % (X_train.shape[0], X_test.shape[
28
```

```
--2021-10-13 20:02:45--  https://raw.githubusercontent.com/MicrosoftDocs/mslearn-
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.13
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.1
HTTP request sent, awaiting response... 200 OK
Length: 48800 (48K) [text/plain]
Saving to: ‘daily-bike-share.csv.1’

daily-bike-share.cs 100%[===================>]  47.66K  --.-KB/s    in 0.01s

2021-10-13 20:02:45 (4.60 MB/s) - ‘daily-bike-share.csv.1’ saved [48800/48800]

   instant    dteday  season  yr  ...       hum  windspeed  rentals  day
0        1  1/1/2011       1   0  ...  0.805833   0.160446      331    1
1        2  1/2/2011       1   0  ...  0.696087   0.248539      131    2
2        3  1/3/2011       1   0  ...  0.437273   0.248309      120    3
3        4  1/4/2011       1   0  ...  0.590435   0.160296      108    4
4        5  1/5/2011       1   0  ...  0.436957   0.186900       82    5

[5 rows x 15 columns]
Training Set: 511 rows
Test Set: 220 rows
```

Ahora tenemos los siguientes cuatro conjuntos de datos:

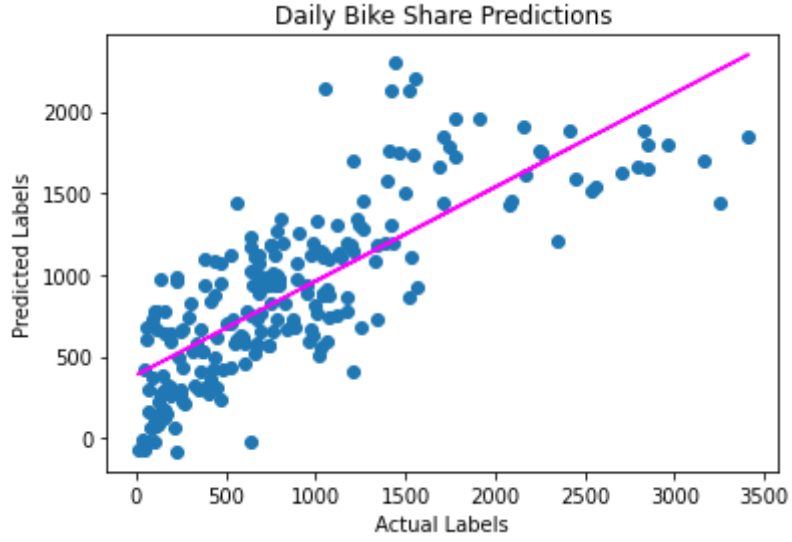
X_train : los valores de características que usaremos para entrenar el modelo y_train : Las etiquetas correspondientes que usaremos para entrenar el modelo. X_test : los valores de características que usaremos para validar el modelo y_test : las etiquetas correspondientes que usaremos para validar el modelo Ahora estamos listos para entrenar un modelo ajustando un algoritmo de regresión adecuado a los datos de entrenamiento.

Experimentar con algoritmos El algoritmo de regresión lineal que usamos la última vez para entrenar el modelo tiene cierta capacidad predictiva, pero hay muchos tipos de algoritmos de regresión que podríamos probar, incluidos:

Algoritmos lineales : no solo el algoritmo de regresión lineal que usamos anteriormente (que técnicamente es un algoritmo de mínimos cuadrados ordinarios), sino otras variantes como Lasso y Ridge . Algoritmos basados en árboles : algoritmos que construyen un árbol de decisiones para llegar a una predicción. Algoritmos de conjunto : algoritmos que combinan las salidas de múltiples algoritmos de base para mejorar la generalización. Nota : Para obtener una lista completa de estimadores de Scikit-Learn que encapsulan algoritmos para el aprendizaje automático supervisado, consulte la documentación de Scikit-Learn . Hay muchos algoritmos para elegir, pero para la mayoría de los escenarios del mundo real, la hoja de trucos del estimador de Scikit-Learn puede ayudarlo a encontrar un punto de partida adecuado. Pruebe otro algoritmo lineal Intentemos entrenar nuestro modelo de regresión usando un algoritmo Lasso . Podemos hacer esto simplemente cambiando el estimador en el código de entrenamiento.

```
from sklearn.linear_model import Lasso

# Fit a lasso model on the training set
model = Lasso().fit(X_train, y_train)
print (model, "\n")

# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```

```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)

MSE: 201155.70593338404
RMSE: 448.5038527519959
R2: 0.6056468637824488
```



Pruebe un algoritmo de árbol de decisión Como alternativa a un modelo lineal, hay una categoría de algoritmos de aprendizaje automático que utiliza un enfoque basado en el árbol en el que las características del conjunto de datos se examinan en una serie de evaluaciones, cada uno de cuyos resultados en una rama en un árbol de decisiones basadas en el valor de la característica. Al final de cada serie de ramas hay nodos hoja con el valor de etiqueta predicho en función de los valores de las características.

Es más fácil ver cómo funciona esto con un ejemplo. Entrenemos un modelo de regresión de árbol de decisión utilizando los datos de alquiler de bicicletas. Después de entrenar el modelo, el código siguiente imprimirá la definición del modelo y una representación de texto del árbol que

```
1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.tree import export_text
3
4 # Train the model
5 model = DecisionTreeRegressor().fit(X_train, y_train)
6 print (model, "\n")
7
8 # Visualize the model tree
9 tree = export_text(model)
10 print(tree)
```

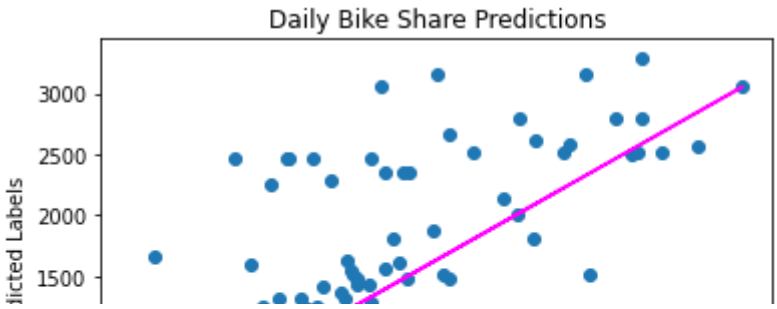
```
|   |   |   |   |   |   |   |   |--- feature_6 >  0.61
|   |   |   |   |   |   |   |   |   |--- value: [1417.00]
|   |   |   |   |   |   |   |   |--- feature_1 >  5.50
|   |   |   |   |   |   |   |   |   |--- feature_7 <= 0.53
|   |   |   |   |   |   |   |   |   |   |--- value: [1182.00]
|   |   |   |   |   |   |   |   |   |--- feature_7 >  0.53
|   |   |   |   |   |   |   |   |   |   |--- feature_6 <= 0.70
|   |   |   |   |   |   |   |   |   |   |   |--- feature_9 <= 0.14
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |   |--- feature_9 >  0.14
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- feature_6 >  0.70
|   |   |   |   |   |   |   |   |   |   |   |--- value: [1045.00]
|   |   |   |   |   |   |   |--- feature_9 >  0.20
|   |   |   |   |   |   |   |   |--- feature_6 <= 0.60
|   |   |   |   |   |   |   |   |   |--- value: [529.00]
|   |   |   |   |   |   |   |   |--- feature_6 >  0.60
|   |   |   |   |   |   |   |   |   |--- value: [533.00]
|   |   |   |--- feature_8 >  0.83
|   |   |   |   |--- feature_5 <= 2.50
|   |   |   |   |   |--- feature_9 <= 0.10
|   |   |   |   |   |   |--- value: [258.00]
|   |   |   |   |   |--- feature_9 >  0.10
|   |   |   |   |   |   |--- feature_1 <= 5.50
|   |   |   |   |   |   |   |--- feature_7 <= 0.51
|   |   |   |   |   |   |   |   |--- value: [692.00]
|   |   |   |   |   |   |   |--- feature_7 >  0.51
|   |   |   |   |   |   |   |   |--- feature_3 <= 2.50
|   |   |   |   |   |   |   |   |   |--- value: [678.00]
|   |   |   |   |   |   |   |   |--- feature_3 >  2.50
|   |   |   |   |   |   |   |   |   |--- feature_8 <= 0.85
|   |   |   |   |   |   |   |   |   |   |--- value: [547.00]
|   |   |   |   |   |   |   |   |   |--- feature_8 >  0.85
|   |   |   |   |   |   |   |   |   |   |--- value: [536.00]
|   |   |   |   |   |--- feature_1 >  5.50
```

```
|   |   |   |   |   |   |--- feature_3 <= 3.50
|   |   |   |   |   |   |   |--- feature_7 <= 0.55
|   |   |   |   |   |   |   |   |--- value: [438.00]
|   |   |   |   |   |   |   |--- feature_7 >  0.55
|   |   |   |   |   |   |   |   |--- feature_3 <= 2.50
|   |   |   |   |   |   |   |   |   |--- value: [477.00]
|   |   |   |   |   |   |   |   |--- feature_3 >  2.50
|   |   |   |   |   |   |   |   |   |--- value: [480.00]
|   |   |   |   |   |   |--- feature_3 >  3.50
|   |   |   |   |   |   |   |--- value: [555.00]
|   |   |   |--- feature_5 >  2.50
|   |   |   |   |--- feature_7 <= 0.49
|   |   |   |   |   |--- value: [126.00]
|   |   |   |   |--- feature_7 >  0.49
|   |   |   |   |   |--- feature_8 <= 0.88
|   |   |   |   |   |   |--- value: [315.00]
|   |   |   |   |   |--- feature_8 >  0.88
|   |   |   |   |   |   |--- feature_9 <= 0.25
|   |   |   |   |   |   |   |--- value: [254.00]
|   |   |   |   |   |   |--- feature_9 >  0.25
|   |   |   |   |   |   |   |--- feature_1 <= 9.50
|   |   |   |   |   |   |   |   |--- value: [204.00]
|   |   |   |   |   |   |   |--- feature_1 >  9.50


|   |   |   |   |   |   |   |   |--- value: [217.00]
```

Entonces ahora tenemos un modelo basado en árboles; pero ¿es bueno? Evaluémoslo con los datos de la prueba.

```
# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```

```
MSE: 261535.9818181818
RMSE: 511.40588754743703
R2: 0.48727512259636285
```



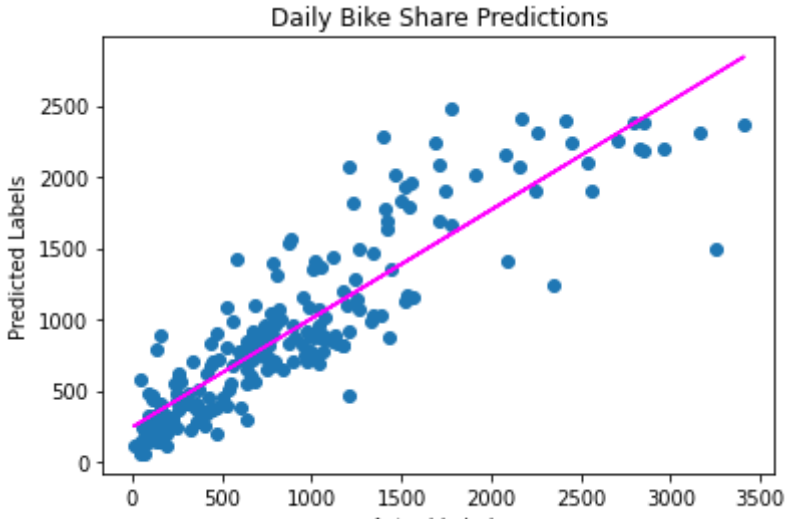
El modelo basado en árboles no parece haber mejorado con respecto al modelo lineal, entonces, ¿qué más podríamos probar?

Pruebe un algoritmo de conjunto Los algoritmos de conjunto funcionan combinando varios estimadores de base para producir un modelo óptimo, ya sea aplicando una función agregada a una colección de modelos base (a veces denominada embolsado) o construyendo una secuencia de modelos que se basan entre sí para mejorar el rendimiento predictivo (denominado refuerzo).

Por ejemplo, probemos un modelo de bosque aleatorio, que aplica una función de promedio a varios modelos de árbol de decisión para un mejor modelo general.

```
from sklearn.ensemble import RandomForestRegressor

# Train the model
model = RandomForestRegressor().fit(X_train, y_train)
print (model, "\n")

# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)

MSE: 112053.27708136362
RMSE: 334.74359901477374
R2: 0.7803265831538294
```

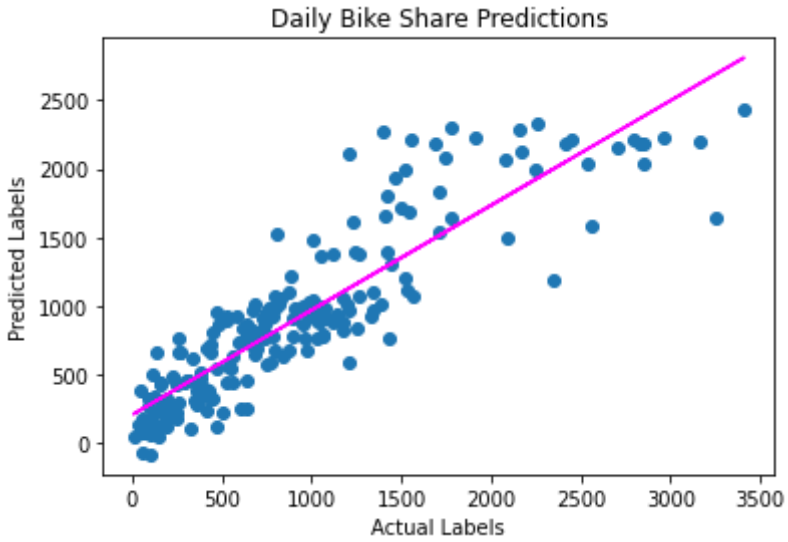


Por si fuera poco, también vamos a probar un impulso conjunto algoritmo. Usaremos un estimador Gradient Boosting, que como un algoritmo Random Forest construye múltiples árboles, pero en lugar de construirlos todos de forma independiente y tomar el resultado promedio, cada árbol se construye sobre las salidas del anterior en un intento de reducir incrementalmente el pérdida (error) en el modelo.

```
# Train the model
from sklearn.ensemble import GradientBoostingRegressor

# Fit a lasso model on the training set
model = GradientBoostingRegressor().fit(X_train, y_train)
print (model, "\n")

# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
```

```
23 z = np.polyfit(y_test, predictions, 1)
24 p = np.poly1d(z)
25 plt.plot(y_test,p(y_test), color='magenta')
26 plt.show()
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)

MSE: 104456.2820362519
RMSE: 323.1969709577302
R2: 0.7952200151246885
```



Mejorar modelos con hiperparámetros

Los modelos simples con conjuntos de datos pequeños a menudo se pueden ajustar en un solo paso, mientras que los conjuntos de datos más grandes y los modelos más complejos deben ajustarse utilizando repetidamente el modelo con datos de entrenamiento y comparando el resultado con la etiqueta esperada. Si la predicción es lo suficientemente precisa, consideramos que el modelo está entrenado. Si no es así, ajustamos ligeramente el modelo y volvemos a hacer un bucle.

Los hiperparámetros son valores que cambian la forma en que se ajusta el modelo durante estos bucles. La tasa de aprendizaje, por ejemplo, es un hiperparámetro que establece cuánto se ajusta un modelo durante cada ciclo de entrenamiento. Una alta tasa de aprendizaje significa que un modelo se puede entrenar más rápido, pero si es demasiado alto, los ajustes pueden ser tan grandes que el modelo nunca se 'ajusta con precisión' y no es óptimo.

Procesamiento previo de datos

El preprocesamiento se refiere a los cambios que realiza en sus datos antes de pasarlos al modelo. Hemos leído anteriormente que el preprocesamiento puede implicar la limpieza de su conjunto de datos. Si bien esto es importante, el preprocesamiento también puede incluir cambiar el formato de sus datos, por lo que es más fácil de usar para el modelo. Por ejemplo, los datos descritos como 'rojo', 'naranja', 'amarillo', 'lima' y 'verde' pueden funcionar mejor si se convierten a un formato más nativo de las computadoras, como números que indiquen la cantidad de rojo y la cantidad de verde.

Funciones de escala El paso de preprocesamiento más común es escalar entidades para que caigan entre cero y uno. Por ejemplo, el peso de una bicicleta y la distancia que una persona viaja en bicicleta pueden ser dos números muy diferentes, pero al escalar ambos números entre cero y uno, los modelos pueden aprender de manera más efectiva a partir de los datos.

Usar categorías como características En el aprendizaje automático, también puede utilizar funciones categóricas como "bicicleta", "patineta" o "coche". Estas características están representadas por valores 0 o 1 en vectores one-hot , vectores que tienen un 0 o 1 para cada valor posible. Por ejemplo, bicicleta, patineta y coche pueden ser respectivamente (1,0,0), (0,1,0) y (0,0,1).

Regresión: optimizar y guardar modelos En el cuaderno anterior, usamos modelos de regresión complejos para observar la relación entre las características de un conjunto de datos de alquiler de bicicletas. En este portátil veremos si podemos mejorar aún más el rendimiento de estos modelos.

Comencemos cargando los datos de uso compartido de bicicletas como un Pandas DataFrame y viendo las primeras filas. Como de costumbre, también dividiremos nuestros datos en conjuntos de datos de prueba y entrenamiento.

```
# Import modules we'll need for this notebook
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# load the training dataset
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machir
bike_data = pd.read_csv('daily-bike-share.csv')
bike_data['day'] = pd.DatetimeIndex(bike_data['dteday']).day
numeric_features = ['temp', 'atemp', 'hum', 'windspeed']
categorical_features = ['season','mnth','holiday','weekday','workingday','weathersit
bike_data[numeric_features + ['rentals']].describe()
print(bike_data.head())

```

```
19
20 # Separate features and labels
21 # After separating the dataset, we now have numpy arrays named **X** containing the
22 X, y = bike_data[['season','mnth', 'holiday','weekday','workingday','weathersit','te
23
24 # Split data 70%-30% into training set and test set
25 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_sta
26
27 print ('Training Set: %d rows\nTest Set: %d rows' % (X_train.shape[0], X_test.shape[
28
```

```
--2021-10-13 20:05:40--  https://raw.githubusercontent.com/MicrosoftDocs/mslearn-
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.13
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.1
HTTP request sent, awaiting response... 200 OK
Length: 48800 (48K) [text/plain]
Saving to: ‘daily-bike-share.csv.2’

daily-bike-share.cs 100%[===================>]  47.66K  --.-KB/s    in 0.01s

2021-10-13 20:05:40 (4.49 MB/s) - ‘daily-bike-share.csv.2’ saved [48800/48800]

   instant    dteday  season  yr  ...       hum  windspeed  rentals  day
0        1  1/1/2011       1   0  ...  0.805833   0.160446      331    1
1        2  1/2/2011       1   0  ...  0.696087   0.248539      131    2
2        3  1/3/2011       1   0  ...  0.437273   0.248309      120    3
3        4  1/4/2011       1   0  ...  0.590435   0.160296      108    4
4        5  1/5/2011       1   0  ...  0.436957   0.186900       82    5

[5 rows x 15 columns]
Training Set: 511 rows
Test Set: 220 rows
```

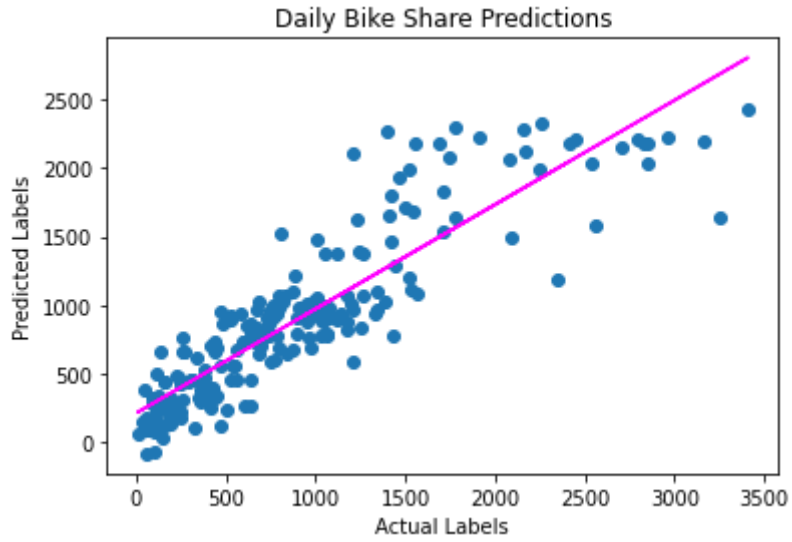
Ahora tenemos los siguientes cuatro conjuntos de datos:

X_train : los valores de características que usaremos para entrenar el modelo y_train : Las etiquetas correspondientes que usaremos para entrenar el modelo. X_test : los valores de características que usaremos para validar el modelo y_test : las etiquetas correspondientes que usaremos para validar el modelo Ahora estamos listos para entrenar un modelo ajustando un algoritmo de conjunto de impulso , como en nuestro último cuaderno. Recuerde que un estimador de aumento de gradiente es como un algoritmo de bosque aleatorio, pero en lugar de construir todos los árboles de forma independiente y tomar el resultado promedio, cada árbol se basa en las salidas del anterior en un intento de reducir gradualmente la pérdida (error) en el modelo.

```
1 # Train the model
2 from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
3
4
5 # Fit a lasso model on the training set
6 model = GradientBoostingRegressor().fit(X_train, y_train)
7 print (model, "\n")
```

```
 8
 9 # Evaluate the model using the test data
10 predictions = model.predict(X_test)
11 mse = mean_squared_error(y_test, predictions)
12 print("MSE:", mse)
13 rmse = np.sqrt(mse)
14 print("RMSE:", rmse)
15 r2 = r2_score(y_test, predictions)
16 print("R2:", r2)
17
18 # Plot predicted vs actual
19 plt.scatter(y_test, predictions)
20 plt.xlabel('Actual Labels')
21 plt.ylabel('Predicted Labels')
22 plt.title('Daily Bike Share Predictions')
23 # overlay the regression line
24 z = np.polyfit(y_test, predictions, 1)
25 p = np.poly1d(z)
26 plt.plot(y_test,p(y_test), color='magenta')
27 plt.show()
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)

MSE: 104143.33418012297
RMSE: 322.71246362686855
R2: 0.795833529754882
```



Optimizar los hiperparámetros Eche un vistazo a la definición del estimador GradientBoostingRegressor en la salida anterior y observe que, al igual que los otros estimadores que probamos anteriormente, incluye una gran cantidad de parámetros que controlan la forma en que se entrena el modelo. En el aprendizaje automático, el término parámetros se refiere a

valores que se pueden determinar a partir de datos; los valores que especifique para afectar el comportamiento de un algoritmo de entrenamiento se denominan más correctamente hiperparámetros .

Los hiperparámetros específicos de un estimador varían según el algoritmo que encapsula el estimador. En el caso del estimador GradientBoostingRegressor , el algoritmo es un conjunto que combina múltiples árboles de decisión para crear un modelo predictivo general. Puede obtener información sobre los hiperparámetros de este estimador en la documentación de Scikit-Learn .

No entraremos en los detalles de cada hiperparámetro aquí, pero funcionan juntos para afectar la forma en que el algoritmo entrena un modelo. En muchos casos, los valores predeterminados proporcionados por Scikit-Learn funcionarán bien; pero puede haber alguna ventaja al modificar los hiperparámetros para obtener un mejor rendimiento predictivo o reducir el tiempo de entrenamiento.

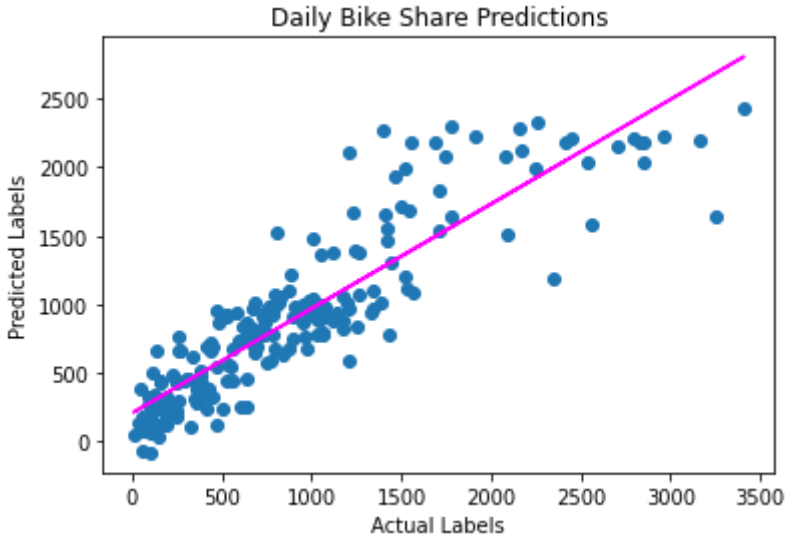
Entonces, ¿cómo sabe qué valores de hiperparámetros debe usar? Bueno, en ausencia de una comprensión profunda de cómo funciona el algoritmo subyacente, deberá experimentar. Afortunadamente, SciKit-Learn proporciona una forma de ajustar los hiperparámetros probando múltiples combinaciones y encontrando el mejor resultado para una métrica de rendimiento determinada.

Intentemos usar un enfoque de búsqueda de cuadrícula para probar combinaciones de una cuadrícula de valores posibles para los hiperparámetros learning_rate y n_estimators del estimador GradientBoostingRegressor .

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, r2_score

# Use a Gradient Boosting algorithm
alg = GradientBoostingRegressor()

# Try these hyperparameter values
params = {
 'learning_rate': [0.1, 0.5, 1.0],
 'n_estimators' : [50, 100, 150]
 }

# Find the best hyperparameter combination to optimize the R2 metric
score = make_scorer(r2_score)
gridsearch = GridSearchCV(alg, params, scoring=score, cv=3, return_train_score=True)
gridsearch.fit(X_train, y_train)
print("Best parameter combination:", gridsearch.best_params_, "\n")

# Get the best model
model=gridsearch.best_estimator_
print(model, "\n")

# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
```

```
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```

```
Best parameter combination: {'learning_rate': 0.1, 'n_estimators': 100}

GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)

MSE: 103751.49217028235
RMSE: 322.1047844572979
R2: 0.7966017114217425
```



Preprocesar los datos Entrenamos un modelo con datos que se cargaron directamente desde un archivo de origen, con resultados solo moderadamente exitosos.

En la práctica, es común realizar algún preprocesamiento de los datos para que sea más fácil para el algoritmo ajustar un modelo. Existe una amplia gama de transformaciones de preprocesamiento que puede realizar para preparar sus datos para el modelado, pero nos limitaremos a algunas técnicas comunes:

Escala de características numéricas La normalización de entidades numéricas para que estén en la misma escala evita que las entidades con valores grandes produzcan coeficientes que afecten desproporcionadamente a las predicciones. Por ejemplo, suponga que sus datos incluyen las siguientes características numéricas:

A B C 3 480 sesenta y cinco La normalización de estas características a la misma escala puede dar como resultado los siguientes valores (suponiendo que A contiene valores de 0 a 10, B contiene valores de 0 a 1000 y C contiene valores de 0 a 100):

A B C 0,3 0,48 0,65 Hay varias formas de escalar datos numéricos, como calcular los valores mínimo y máximo para cada columna y asignar un valor proporcional entre 0 y 1, o usar la desviación media y estándar de una variable distribuida normalmente para mantener la misma extensión de valores en una escala diferente.

Codificación de variables categóricas Los modelos de aprendizaje automático funcionan mejor con características numéricas en lugar de valores de texto, por lo que generalmente necesita convertir características categóricas en representaciones numéricas. Por ejemplo, suponga que sus datos incluyen la siguiente característica categórica.

Tamaño S METRO L Puede aplicar codificación ordinal para sustituir un valor entero único para cada categoría, como este:

Tamaño 0 1 2 Otra técnica común es utilizar una codificación en caliente para crear características binarias individuales (0 o 1) para cada valor de categoría posible. Por ejemplo, podría usar la codificación one-hot para traducir las posibles categorías en columnas binarias como esta:

Tamaño_S Talla M Talla L 1 0 0 0 1 0 0 0 1 Para aplicar estas transformaciones de preprocesamiento al alquiler de bicicletas, haremos uso de una función de Scikit-Learn llamada canalizaciones . Estos nos permiten definir un conjunto de pasos de preprocesamiento que terminan con un algoritmo. Luego, puede ajustar toda la canalización a los datos, de modo que el modelo encapsule todos los pasos de preprocesamiento, así como el algoritmo de regresión. Esto es útil, porque cuando queremos usar el modelo para predecir valores a partir de nuevos datos, necesitamos aplicar las mismas transformaciones (basadas en las mismas distribuciones estadísticas y codificaciones de categorías utilizadas con los datos de entrenamiento).

Nota : El término canalización se usa ampliamente en el aprendizaje automático, ¡a menudo para significar cosas muy diferentes! En este contexto, lo estamos usando para referirnos a objetos de canalización en Scikit-Learn, pero es posible que lo vea en otro lugar para significar algo más.

```
1 # Train the model
2 from sklearn.compose import ColumnTransformer
3 from sklearn.pipeline import Pipeline
4 from sklearn.impute import SimpleImputer
5 from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Define preprocessing for numeric columns (scale them)
numeric_features = [6,7,8,9]
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

# Define preprocessing for categorical features (encode them)
categorical_features = [0,1,2,3,4,5]
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Create preprocessing and training pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('regressor', GradientBoostingRegressor())])


# fit the pipeline to train a linear regression model on the training set
model = pipeline.fit(X_train, (y_train))
print (model)
```

```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('num',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
                                                                   StandardScaler


                                                                   verbose=False),
                                                  [6, 7, 8, 9]),
                                                 ('cat',
                                                  Pipeline(memory=None,
                                                           steps=[('onehot',
                                                                   OneHotEncoder(
                                                learning_rate=0.1, loss='ls',
                                                max_depth=3, max_features=None,
                                                max_leaf_nodes=None,
                                                min_impurity_decrease=0.0,
                                                min_impurity_split=None,
                                                min_samples_leaf=1,
                                                min_samples_split=2,
                                                min_weight_fraction_leaf=0.0,
                                                n_estimators=100,
                                                n_iter_no_change=None,
                                                presort='deprecated',
                                                random_state=None, subsample=1.0,
                                                tol=0.0001, validation_fraction=0.1,
```