Librerías en Python para ciencia de datos.

Ejercicio U10\_E02

MasterD

# Visualización y ajuste de modelos

Ahora vamos a comprobar qué tal te manejas generando algunas gráficas sencillas y construyendo algunos modelos de previsión en base a los hallazgos explorando los datos.

De nuevo, empezaremos leyendo datos de uno de los ficheros incluidos en el directorio de ejercicios de esta unidad. Antes que nada, conviene que cambiemos el directorio de trabajo de nuestra sesión en Python con la función os.chdir().

Ajusta la ruta del directorio a la que estés utilizando en tu ordenador:

**import** **os  
import** **numpy** **as** **np  
import** **pandas** **as** **pd**  
os.chdir("RUTA\_DIRECTORIO\_CUSO/10PythonLibreriasCienciaDatos/Ejercicios")

Ahora ya puedes leer el fichero Auto.csv Recuerda que Pandas ya incluye funciones para leer distintos tipos de fichero, y presta atención a los caracteres delimitadores y a los encabezados.

auto = pd.read\_csv("./Auto.csv", sep=',', header=0)

Comprobemos que los datos se han cargado bien

auto.head()

|  | **mpg** | **cylinders** | **displacement** | **horsepower** | **weight** | **acceleration** | **year** | **origin** | **name** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | 1 | chevrolet chevelle malibu |
| **1** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | 1 | buick skylark 320 |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | 1 | plymouth satellite |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | 1 | amc rebel sst |
| **4** | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | 1 | ford torino |

## Visualización de datos

Vamos a comenzar con algunas gráficas para ver la distribución de las distintas variables que componen nuestro conjunto de datos, utilizando la librería seaborn.

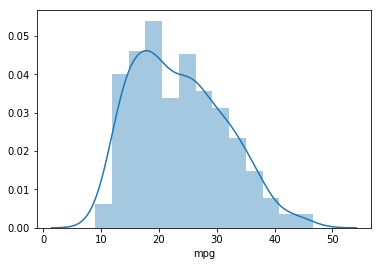
**import** **seaborn** **as** **sns**

*# Si trabajas con Jupyter, recuerda incluir la siguiente directiva*

**%matplotlib** **inline**

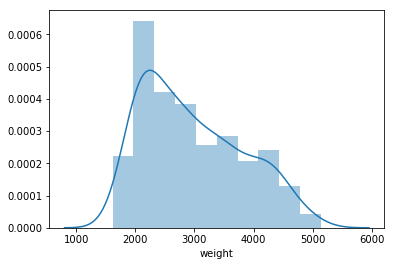
Pinta un gráfico con la distribución de la variable "mpg" (el consumo, *miles per gallon*)

sns.distplot(auto['mpg'])



Ahora muestra la distribución de la variable weight (el peso de cada coche).

sns.distplot(auto['weight'])

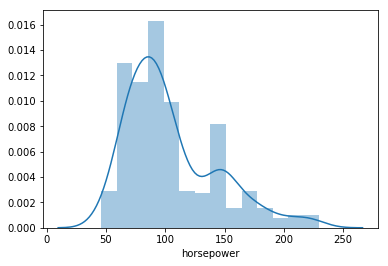


Y ahora con la variable 'horsepower' (la potencia en cv).

¿Qué ocurre al intentar pintar esta última variable? ¿Obtienes un error?

Los datos originales incluyen valores ausentes (NaN) y la librería no sabe cómo manejarlos al tratar de calcular y pintar el histograma. Pero podemos utilizar el método dropna() para descartar todos los valores nulos antes de pintar la gráfica.

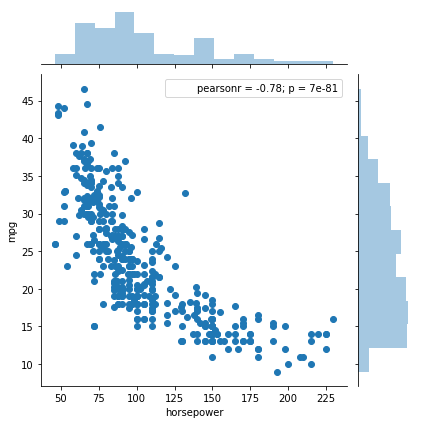
sns.distplot(auto.dropna()['horsepower'])



Muy bien. Ahora vamos a ver cómo influyen estas variables en el consumo.

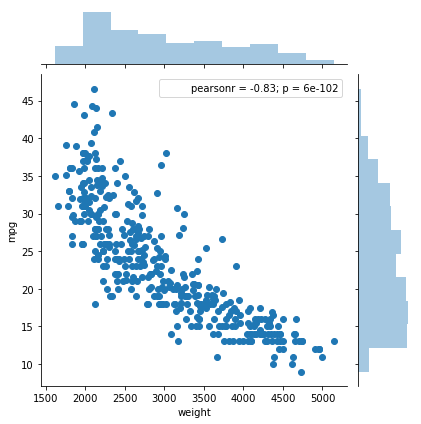
Empieza pintando un gráfico de dispersión con la relación entre potencia y consumo.

sns.jointplot('horsepower', 'mpg', data = auto.dropna())



Y ahora pinta la misma gráfica, pero con la distribución conjunta de peso y consumo.

sns.jointplot('weight', 'mpg', data = auto.dropna())



¿Dirías que existe relación relevante con el consumo de combustible? ¿Dirías que es lineal?

Viendo las gráficas y el valor de los coeficientes de correlación de Pearson, parece que existe una relación significativa de las variables de potencia y peso del vehículo con el consumo de combustible. No obstante, estas gráficas de dispersión también sugieren que la relación es no lineal.

## Ajustando modelos.

Tras explorar gráficamente los datos, vamos a intentar construir algunos modelos que nos permitan dar una estimación del consumo de un vehículo a partir de sus características.

Utilizaremos la librería statsmodels que hemos visto en la unidad 10.

**import** **statsmodels.api** **as** **sm  
import** **statsmodels.formula.api** **as** **smf**

Comienza creando un modelo de regresión lineal simple para predecir la variable consumo ('mpg') a partir de la variable peso ('weight'). Ajusta el modelo y muestra la información de resumen del ajuste.

model\_fn = smf.ols(formula = 'mpg ~ weight', data=auto)

*# Ajustamos los parámetros del modelo (b0, b1)*

mfitted = model\_fn.fit()

*# Veamos el resumen del modelo ajustado*

print(mfitted.summary())

OLS Regression Results

==============================================================================

Dep. Variable: mpg R-squared: 0.692

Model: OLS Adj. R-squared: 0.691

Method: Least Squares F-statistic: 886.6

Date: Sun, 18 Nov 2018 Prob (F-statistic): 5.37e-103

Time: 20:38:09 Log-Likelihood: -1146.0

No. Observations: 397 AIC: 2296.

Df Residuals: 395 BIC: 2304.

Df Model: 1

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

Intercept 46.3174 0.796 58.166 0.000 44.752 47.883

weight -0.0077 0.000 -29.776 0.000 -0.008 -0.007

==============================================================================

Omnibus: 40.133 Durbin-Watson: 0.797

Prob(Omnibus): 0.000 Jarque-Bera (JB): 56.057

Skew: 0.712 Prob(JB): 6.72e-13

Kurtosis: 4.166 Cond. No. 1.13e+04

==============================================================================

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 1.13e+04. This might indicate that there are

strong multicollinearity or other numerical problems.

¿Cuál es el coeficiente para el peso? ¿Y su intervalo de confianza?

¿Existe algún posible problema con el modelo y su ajuste?

Prueba ahora el modelo, prediciendo el valor con una serie de pesos (2000, 3000 y 6000). Pista en caso de errores: ¿es necesario añadir el término constante?

test\_auto = pd.Series(data=[2000, 3000, 6000], name='weight')

test\_auto = sm.add\_constant(test\_auto, prepend=False)

mfitted.predict(test\_auto)

0 30.964188

1 23.287582

2 0.257765

dtype: float64

¿Qué te parece el consumo predicho para un peso de 6000?

En nuestra exploración gráfica anterior, ya pudimos distinguir una relación no lineal con el consumo.

Ajusta un nuevo modelo de regresión, pero esta vez aplicando una transformación logarítmica a la variable peso. Revisa la información de resumen del modelo ajustado.

model\_fn = smf.ols(formula = 'mpg ~ np.log(weight)', data=auto.dropna())

# Ajustamos los parámetros del modelo (b0, b1)

mfitted = model\_fn.fit()

# Veamos el resumen del modelo ajustado

print(mfitted.summary())

OLS Regression Results

==============================================================================

Dep. Variable: mpg R-squared: 0.713

Model: OLS Adj. R-squared: 0.712

Method: Least Squares F-statistic: 967.3

Date: Sun, 18 Nov 2018 Prob (F-statistic): 1.16e-107

Time: 20:44:10 Log-Likelihood: -1116.8

No. Observations: 392 AIC: 2238.

Df Residuals: 390 BIC: 2245.

Df Model: 1

Covariance Type: nonrobust

==================================================================================

coef std err t P>|t| [0.025 0.975]

----------------------------------------------------------------------------------

Intercept 209.9433 6.000 34.990 0.000 198.147 221.740

np.log(weight) -23.4317 0.753 -31.101 0.000 -24.913 -21.951

==============================================================================

Omnibus: 52.088 Durbin-Watson: 0.785

Prob(Omnibus): 0.000 Jarque-Bera (JB): 87.667

Skew: 0.802 Prob(JB): 9.19e-20

Kurtosis: 4.672 Cond. No. 229.

==============================================================================

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

¿Dirías que este modelo es mejor o peor que el anterior?

Vuelve a predecir el consumo con este nuevo modelo para la serie de pesos (2000, 3000, 6000).

test\_auto = pd.Series(data=[2000, 3000, 6000], name='weight')

test\_auto = sm.add\_constant(test\_auto, prepend=False)

mfitted.predict(test\_auto)

0 31.840983

1 22.340230

2 6.098587

dtype: float64

¿Te parece más o menos razonable el consumo para un peso de 6000?

Para terminar, ajusta un nuevo modelo de regresión múltiple. Esta vez utilizando tanto el peso como la potencia como regresores. Aplica la transformación logarítmica a ambos términos. Revisa el resumen de resultados tras el ajuste.

model\_fn = smf.ols(formula = 'mpg ~ np.log(weight) + np.log(horsepower)', data=auto.dropna())

# Ajustamos los parámetros del modelo (b0, b1)

mfitted = model\_fn.fit()

# Veamos el resumen del modelo ajustado

print(mfitted.summary())

OLS Regression Results

==============================================================================

Dep. Variable: mpg R-squared: 0.740

Model: OLS Adj. R-squared: 0.738

Method: Least Squares F-statistic: 552.4

Date: Sun, 18 Nov 2018 Prob (F-statistic): 2.24e-114

Time: 20:46:11 Log-Likelihood: -1097.5

No. Observations: 392 AIC: 2201.

Df Residuals: 389 BIC: 2213.

Df Model: 2

Covariance Type: nonrobust

======================================================================================

coef std err t P>|t| [0.025 0.975]

--------------------------------------------------------------------------------------

Intercept 179.9729 7.420 24.255 0.000 165.384 194.561

np.log(weight) -15.2435 1.478 -10.316 0.000 -18.149 -12.338

np.log(horsepower) -7.6725 1.210 -6.340 0.000 -10.052 -5.293

==============================================================================

Omnibus: 31.511 Durbin-Watson: 0.899

Prob(Omnibus): 0.000 Jarque-Bera (JB): 44.852

Skew: 0.582 Prob(JB): 1.82e-10

Kurtosis: 4.180 Cond. No. 348.

==============================================================================

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

¿Mejora este modelo a los anteriores?