Apuntes del [Curso de Matemáticas para Data Science: Estadística Descriptiva]() en Platzi

¿Para qué sirve la estadística descriptiva?

La estadística descriptiva sirve para 2 cosas:

- Anális exploratorio de la información.
- Preprocesamiento de la información antes de tener un modelo de machine learning.

Estadística descriptiva vs. inferencial

Estadística descriptiva: resumir un historial de datos.

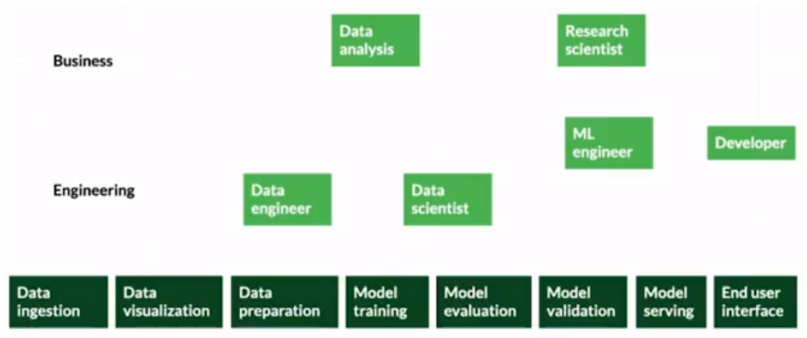
Estadística inferencial: predecir con datos.

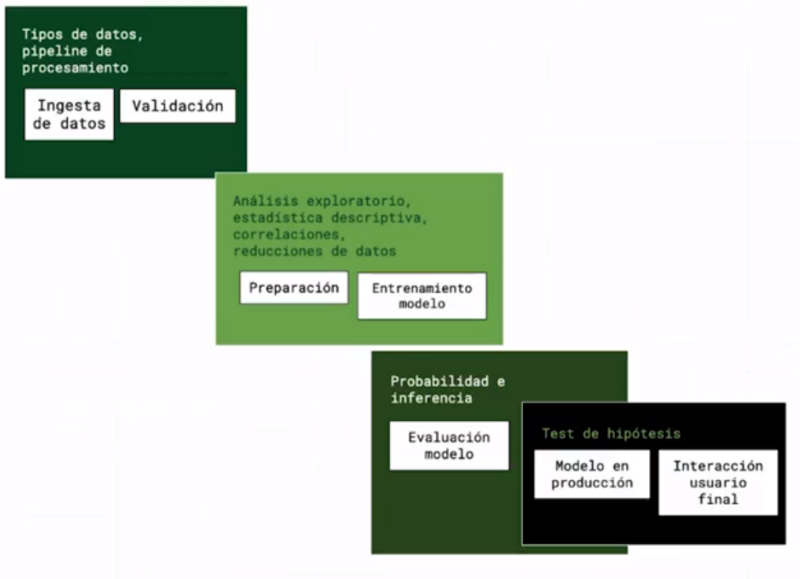
¿Por qué aprender estadística?

- Resumir grandes cantidades de información para tomar mejores decisiones.
- Responder preguntas con relevancia social.
- Reconocer patrones en los datos.

Flujo de trabajo en data science

Flujo del manejo de datos





La estadística descriptiva está presente en los 2 primeros bloques, mientras que la inferencial está en los últimos.

Flujo

- Estadísticos para ingesta y procesamiento:
  1. ¿Cuáles son los tipos de datos con los que estás trabajando? (identifícalos)
  2. Define el flujo de procesamiento de los datos para que sean útiles
- Estadísticos para analítica exploración:
  1. Análisis exploratorio de los datos (correlaciones y reducciones)

Estadística descriptiva para analítica

Tipos de datos en estadística inferencial

Puedes ver los tipos de datos con `dtypes`

- **Datos categóricos:** ordinales (object) y nominales (bool)
- **Datos numéricos:** discretos (int64) y continuos (float64)

Dataset con esos tipos de datos: https://www.kaggle.com/lepchenkov/usedcarscatalog

```
import pandas as pd

df = pd.read_csv("cars.csv")
df
```

Tipos de datos del csv

```
df.dtypes
```

Generar un conjunto completo de estadísticos descriptivos (fundamentales) del dataset:

```
df.describe()
```

Medidas de tendencia central

- Media → promedio (puede ser afectado por algo que se sale bastante del promedio)
- Mediana → dato central
- Moda → dato que más se repite (es de ayuda un diagrama de frecuencias y no aplica para datos numéricos continuos)

Todos los estadísticos descriptivos se pueden representar en este diagrama (distribuciones).

Media o promedio → mean(df)

$\frac{1}{N} \sum\_i = 1^N x_i$

Mediana → median(df)

Cuando hay valores atípicos fuertes que puedan sesgar la media, es mejor emplear la mediana.

Mediana impar: $x\_(n+1)/2^{\text{ordered}}$

Mediana par: $\frac{x\_n/2^{\text{ordered}} + x\_n/2+1^{\text{ordered}}}{2}$

Siempre y cuando los valores del DataFrame estén ordenados

Moda

x_k **donde** $\text{Freq}(x_k) = \max(\text{Freq}(x_i))$

Aplicación y notas en Python (Deepnote)

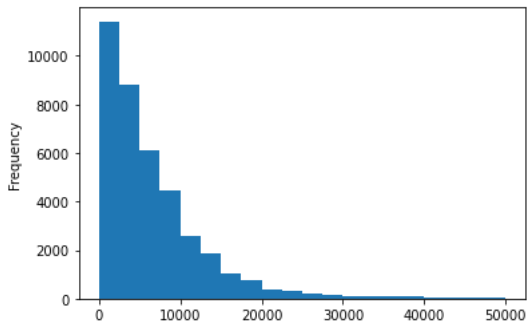
```
import pandas as pd

df = pd.read_csv('cars.csv')
```

```
#Media
df['price_usd'].mean()
```

```
#Mediana
```

```
#Grafico en pandas de un histograma de frecuencia
df['price_usd'].plot.hist(bins=20) #bins da los intervalos de valores en los que se trazará el diagrama
```



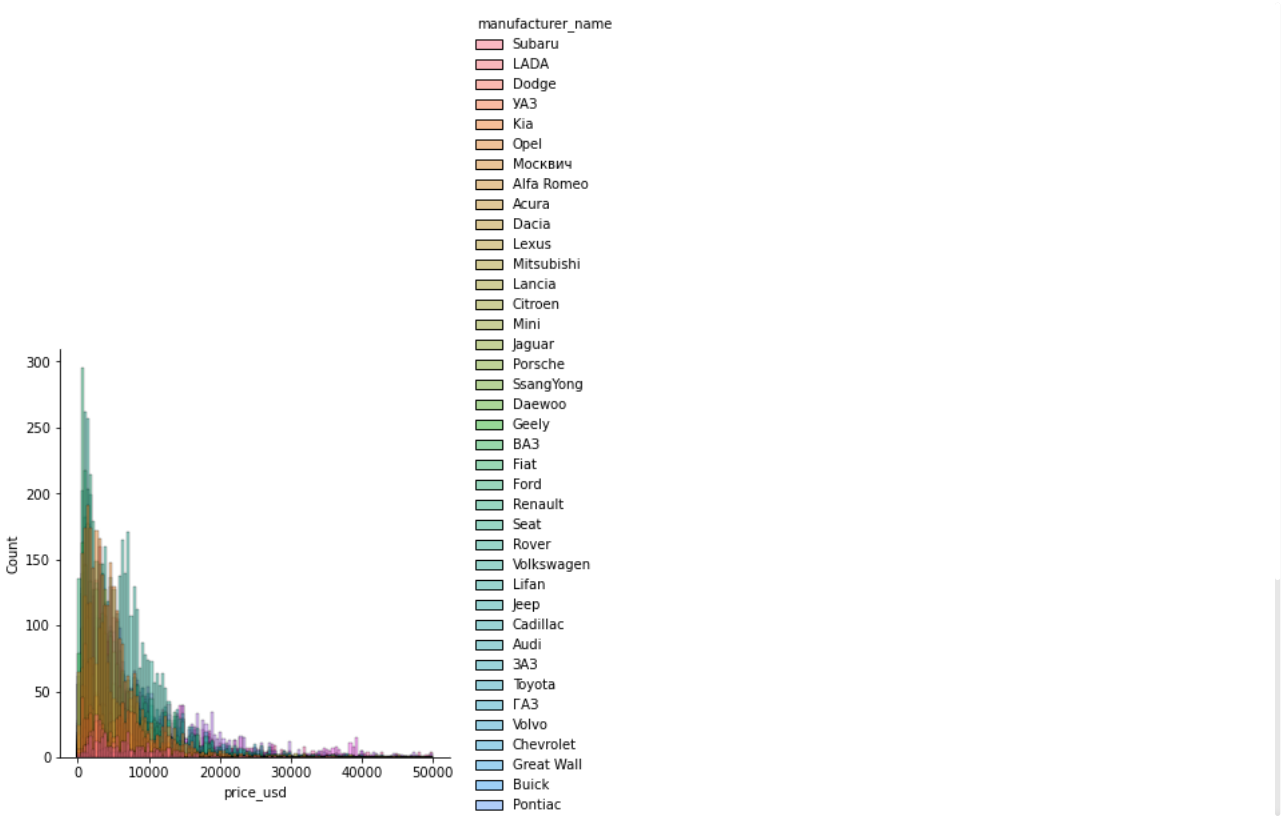
Debido a que hay un fuerte sesgo de los autos con precios más elevados, la **mediana** funciona mejor que la **media**.

Resutla más interersante analizar los precios por marcas:

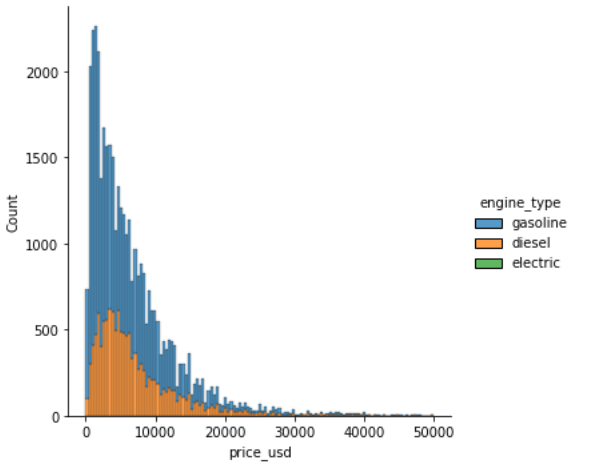
- **Pro tip**: usar seaborn

```
import seaborn as sns

#distribution plot para hacer un histograma con las marcas de carros
sns.displot(df, x='price_usd', hue='manufacturer_name') #hue crea un histograma por cada una de las categorías de manufactorer_name
```




```
#Histograma, de barras apiladadf con el tipo de combustible que necesitan
sns.displot(df, x='price_usd', hue='engine_type', multiple='stack')
```

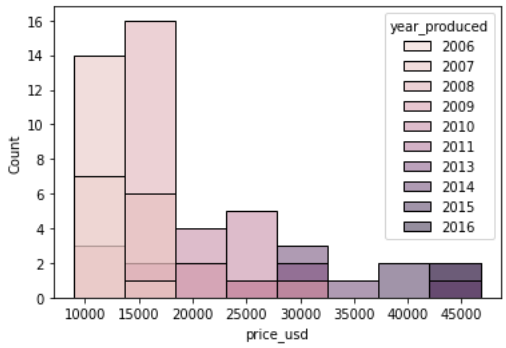


No se ven los autos eléctricos, así que hay que contar cuántos hay

```
df.groupby('engine_type').count()
```

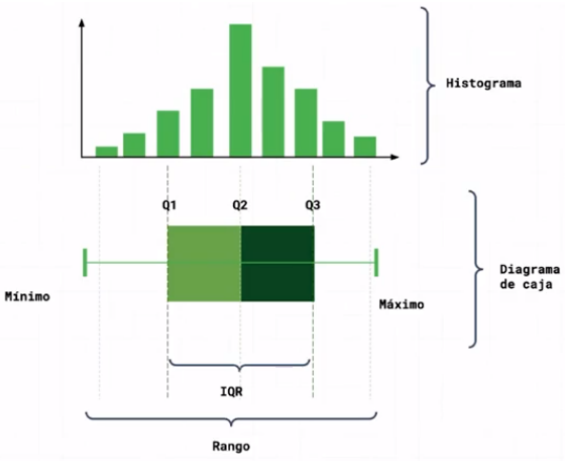
Para hacer una inspección de una marca en específico se pueden aplicar filtros

```
Q7_df = df[(df['manufacturer_name'] == 'Audi') & (df['model_name'] == 'Q7')]
sns.histplot(Q7_df, x='price_usd', hue='year_produced')
```



Medidas de dispersión

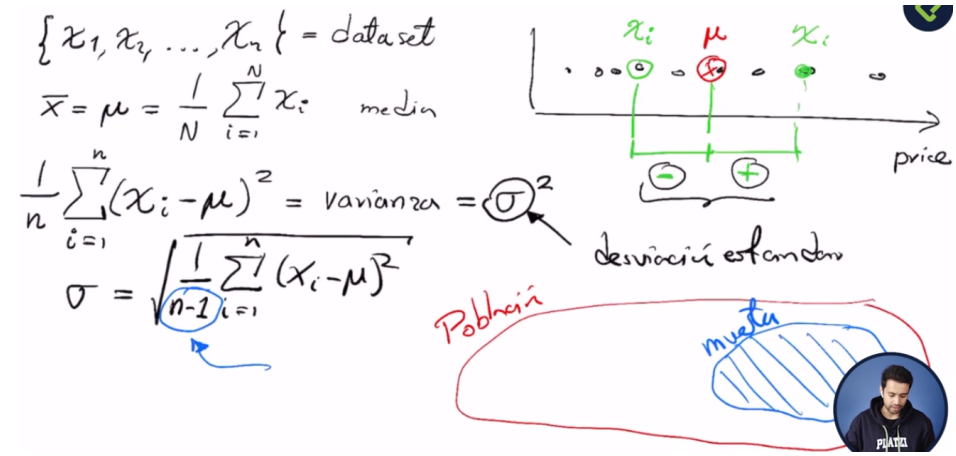
- Rango: abarca desde el mínimo hasta el máximo en el conjunto de datos (todos los datos)
- Rango intercuartil (IQR): 4 subdivisiones homogéneas de los datos
- Desviación estándar



El diagrama de caja es la visualización para representar simplificadamente la dispersión de los datos en referencia a la mediana.

Desviación estándar

Es la medida de dispersión de datos más utilizada. Matemáticamente es la raíz de la media del error cuadrático. Cuando se está trabajando con muestras en vez de la población, se hace una corrección haciendo que se divida para `n-1` .



Si los datos siguen una distribución normal, si se considera todos los datos que están en un rango de `promedio ± 3*desviación estándar` se estaría abarcando el 99.72% de los datos de la distribución. Los puntos que se salen de eso no corresponden con el patrón y se conocen como outliers y a veces se los descarga. En otras palabras, si los datos están más allá de `3*std` , se descartan.

Si siguen una distribución asimétrica, no aplica lo anterior, si no que se crean funciones para determinar qué datos son relevantes. Pero generalmente solo se usa para desviación estándar para distribuciones simétricas.

Medidas de dispersión en Python

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

df = pd.read_csv('cars.csv')
```

Calcular la desviación estándar

```
df['price_usd'].std()
```

```
#Rango = valor max - valor min
rango = df['price_usd'].max() - df['price_usd'].min()
rango
```

```
#Quartiles
median = df['price_usd'].median()
Q1 = df['price_usd'].quantile(q=0.25) #toma el primer 25% de todos los datos
Q3 = df['price_usd'].quantile(q=0.75)
min_val = df['price_usd'].quantile(q=0)
max_val = df['price_usd'].quantile(q=1)
print(min_val, Q1, median, Q3, max_val)
```

```
1.0 2100.0 4800.0 8990.0 50000.0
```

```
iqr = Q3 - Q1
iqr
```

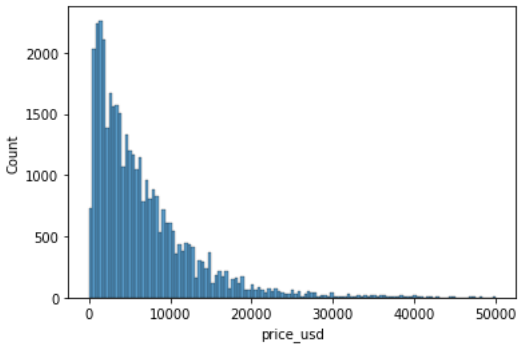
Límites para detección de outliers (con datos simétricamente distribuídos)

```
minlimit = Q1 - 1.5*iqr
maxlimit = Q3 + 1.3*iqr
print(minlimit, maxlimit)
```

```
-8235.0 17947.0
```

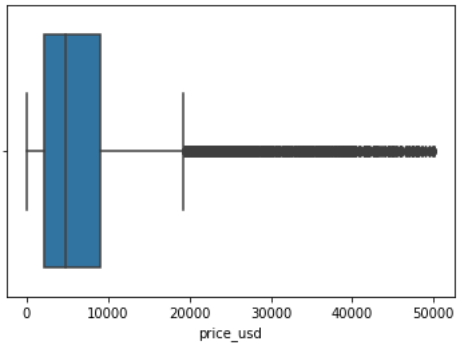
El valor es negativo porque se está aplicando una ecuación de una distribución simética a una no simétrica.

```
sns.histplot(df['price_usd'])
```

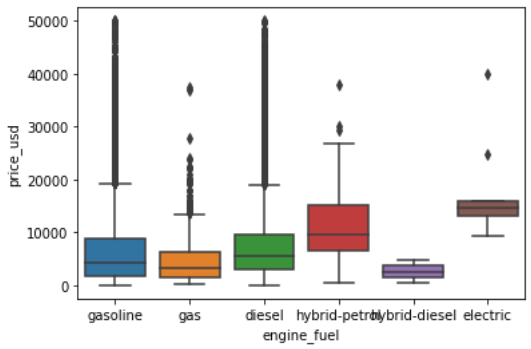



```
sns.boxplot(df['price_usd'])
```

```
/shared-libs/python3.7/py/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12,
  FutureWarning
```




```
sns.boxplot(x='engine_fuel', y='price_usd', data=df)
```



Exploración visual de los datos

Hay que saber qué gráfico es el correcto para enseñar los datos en cuestión.

Esta página tiene todos los gráficos y explica en qué consiste cada uno de ellos y deja ejemplos de uso reales. También te permite clasificarlos en función de los inputs y formas.

Enlace: https://datavizproject.com/

Diagramas de dispersión en el análisis de datos

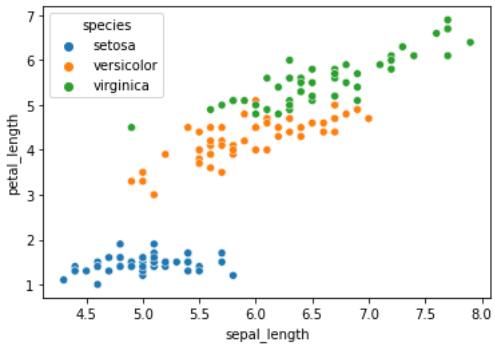
Para esto se va a trabajar con un nuevo dataset clásico. Se trata de iris y lo que trae es datos de atributos especiales de unas flores llamadas iris, vienen en 3 especies distintas. El dataset viene por defecto en seaborn.

```
import pandas as pd
import seaborn as sns

iris = sns.load_dataset('iris')
iris.head()
```

Documentación de scatterplot (diagrama de dispersión): http://seaborn.pydata.org/generated/seaborn.scatterplot.html

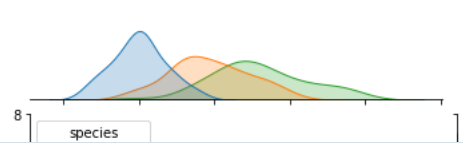
```
sns.scatterplot(data=iris, x='sepal_length', y='petal_length', hue='species')
```



Jointplot muestra un scatterplot y la distribución de los datos.

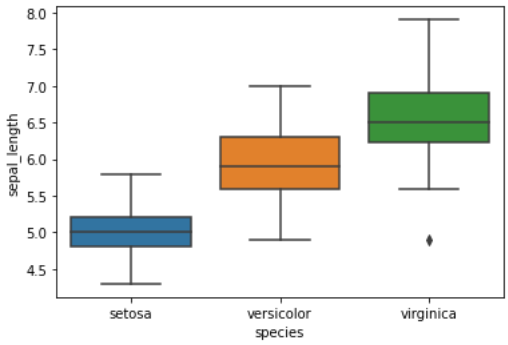
Documentación de jointplot: http://seaborn.pydata.org/generated/seaborn.jointplot.html

```
sns.jointplot(data=iris, x='sepal_length', y='petal_length', hue='species')
```



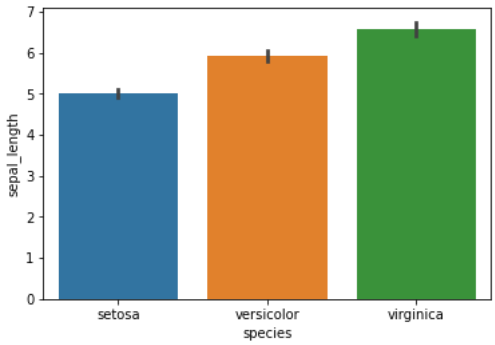
Documentación de boxplot: http://seaborn.pydata.org/generated/seaborn.boxplot.html

```
sns.boxplot(data=iris, x='species', y='sepal_length')
```



Documentación de barplot: http://seaborn.pydata.org/generated/seaborn.barplot.html

```
sns.barplot(data=iris, x='species', y='sepal_length')
```



Estadística en la ingesta de datos

Pipelines de procesamiento para variables numéricas

Escalamiento lineal

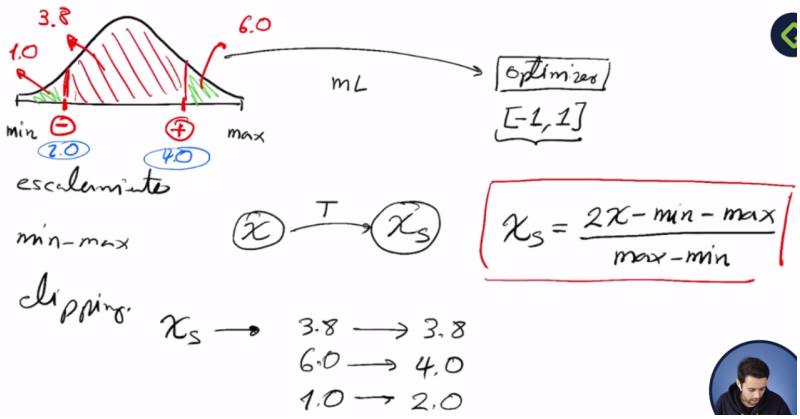
Es importante normalizar los datos (hacer escalamiento lineal), antes de pasarlos por un modelo de machine learning. Esto porque los modelos son eficientes si están en el mismo rango `[-1, 1]`. Si no están en ese rango, tienen que ser transformados (escalados).

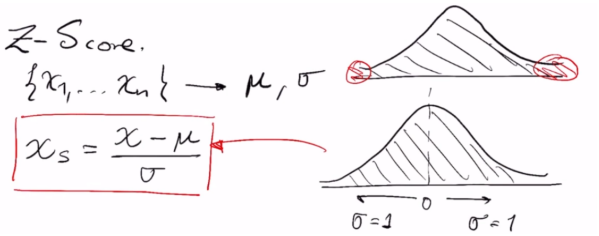
Hay diferentes tipos de escalamiento lineal (max-min, Clipping, Z-score, Winsorizing, etc.). Se los usa normalmente cuando la data es simétrica o está uniformemente distribuida.

Tipos de escalamiento lineal

- **Min-max**: hace una transformación para que los datos entren en el rango `[-1, 1]` por medio de una fórmula. Es uno de los más usados. **Funciona mejor para datos uniformemente distribuidos**.
- **Clipping**: fuerza a que los datos que se salen del rango, se transformen en datos dentro del mismo. Este método no es muy recomendado porque descarta valores outliers que puede que estén funcionando bien.

- **Winsorizing**: una variación del clipping que usa los quartils como extremos.
- **Z-Score**: es uno de los más comunes porque está basado en el promedio y desviación estándar. **Funciona mejor para datos distribuidos "normalmente" (forma de campana de Gauss)**.





Transformaciones lineales en Python

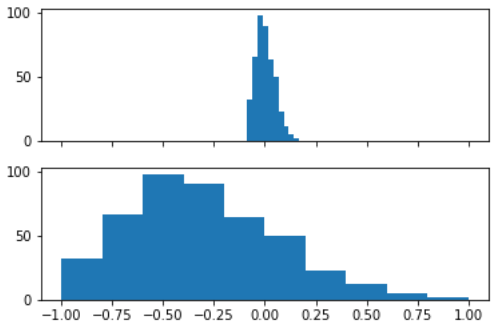
Se utilizará este dataset de scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html

```
import timeit #para medir el tiempo de ejecución de los modelos
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model #datasets para descargar un modelo y linear_model para hacer una regresión lineal

X, y = datasets.load_diabetes(return_X_y=True) #carga el dataset
raw = X[:, None, 2] #transformación en las dimensiones para que se ajuste al formato de entrada del preprocesamiento
```

```
#reglas de escalamiento lineal, aplicamos max-min
max_raw = max(raw)
#raw = datos crudos
min_raw = min(raw)
scaled = (2*raw - max_raw - min_raw)/(max_raw - min_raw)

# es importante tener una noción de los datos originales antes y después de escalarlos:
fig, axs = plt.subplots(2, 1, sharex=True)
axs[0].hist(raw)
axs[1].hist(scaled)
```



```
# modelos para entrenamiento
def train_raw():
    linear_model.LinearRegression().fit(raw, y)

def train_scaled():
    linear_model.LinearRegression().fit(scaled, y)
```

```
raw_time = timeit.timeit(train_raw, number=100) #repite la ejecución del código 100 veces y sobre eso calcula el tiempo
scaled_time = timeit.timeit(train_scaled, number=100)
print(f'train raw: {raw_time}')
print(f'train scaled: {scaled_time}')
```

```
train raw: 0.07411030999355717
train scaled: 0.06660442000429612
```

Se puede ver como al normalizar los datos, el algoritmo se vuelve más eficiente.

Scikit Learn tiene una parte de prepocesamiento, en su documentación encontrarás cómo estandarizar datos numéricos y catergóricos.

Utilidades de Scikit Learn: https://scikit-learn.org/stable/modules/preprocessing.html

---

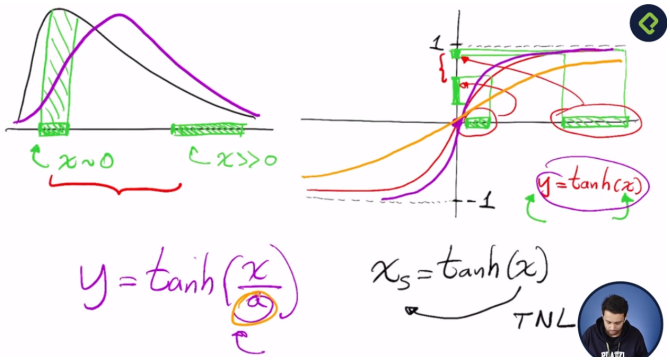
Transformación no lineal

Cuando la data no es simétrica o uniforme, sino que está muy sesgada, se le aplica una transformación para que tengan una distribución simétrica y se pueda aplicar los escalamientos lineales.

Hay diferentes tipos de de funciones no lineales: logaritmos, sigmoides, polinomiales, etc. Estas funciones se les puede aplicar a los datos para transformarlos y hacerlos homogéneos.

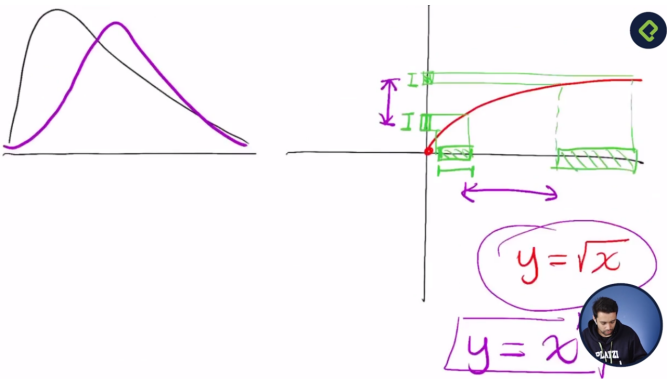
Tanh(x)

La tanh siempre está en un rango de -1 a 1 en Y, por lo que, cuando los valores de X son muy altos, estarán muy cercanos al |1|. También se podría calibrar los datos para ajustar la curva, dividiéndolos por un parámetro a.



Raíz cuadrada

Otras funciones polinomiales, por ejemplo la raíz cuadrada (x½), también pueden hacer que una distribución se normalice.

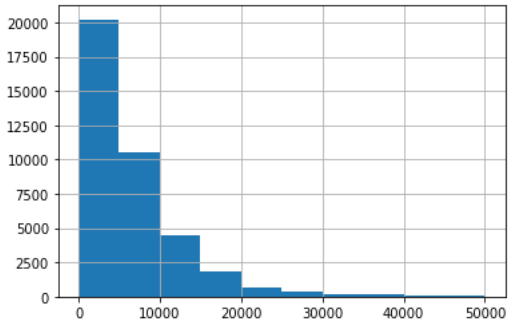


Hay un abanico gigante de maneras para hacer estas transformaciones con funciones no lineales. Los anteriores fueron solo 2 ejemplos. Las más famosas son la tanh y la sigmoide porque ambas amplian los valores importantes y acercan al 0, mientras que a los outliers los minimizan y acercan a -1 y 1.

Transformaciones no lineales en Python

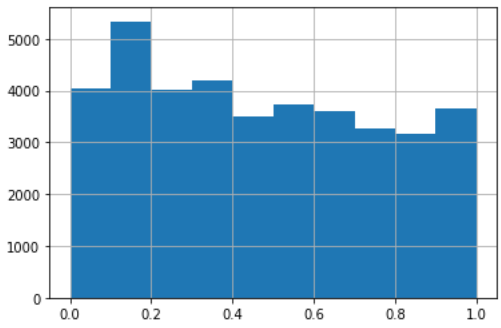
```
df = pd.read_csv('cars.csv')
```

```
# Acá se puede apreciar como la distribución está fuertemente sesgada
df.price_usd.hist()
```




```
# Transformación con tanh(x)

# Esta línea toma la columna y le aplica a toda una función matemática
p = 10000
df.price_usd.apply(lambda x: np.tanh(x/p)).hist()
```



En esta documentación encontrarás diversas maneras de hacer esas transformaciones no lineales. De esa manera podrás aplicar las funciones que trae Scikit Learn para hacer las transformaciones.

Mapear datos a una distribución Gaussiana: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_map_data_to_normal.html

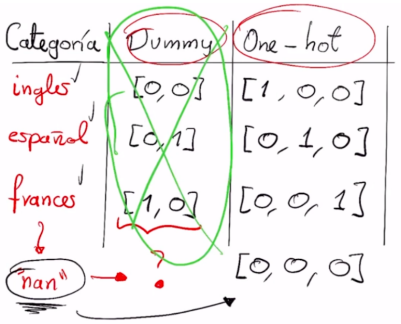
Pipelines de procesamiento para variables categóricas

Cuando se tiene variables categóricas se hace un mapeo numérico. Para eso hay 2 métodos, de manera que sean fácilmente interpretables en modelos de machine learning:

- **Dummy**: es la representación más compacta que se puede tener de los datos. Es mejor usarla cuando los inputs son variables linealmente independientes (no tienen un grado de correlación significativo). Es decir, las cuando se sabe que las categorías son independientes entre sí.
- **One-hot**: es más extenso. Permite incluir categorías que no estaban en el dataset inicialmente. De forma que si se filtra una categoría que no estaba incluida, igual se pueda representar numéricamente y no de error en el modelo (este modelo es más cool y es el que se usa).

Hay errores en la notación de Pandas y los tratan como que ambos modelos son lo mismo, pero en la realidad el Dummy no se usa. Aún así, en Pandas el método es `.get_dummies()`.

Ejemplo de aplicación de ambos:



Procesamiento de datos categóricos en Python

```
import pandas as pd

df = pd.read_csv('cars.csv')
```

Documentación de Pandas dummies: https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

```
pd.get_dummies(df['engine_type'])
```

Documentación de One-hot con Scikit: https://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features

```
import sklearn.preprocessing as preprocessing

encoder = preprocessing.OneHotEncoder(handle_unknown='ignore')
```

```
encoder.fit(df[['engine_type']].values)
```

```
encoder.transform([['gasoline'],['diesel'], ['aceite']]).toarray()
```

Las variables numéricas discretas (números enteros) también pueden ser codificadas como categóricas

```
encoder.fit(df[['year_produced']].values)
```

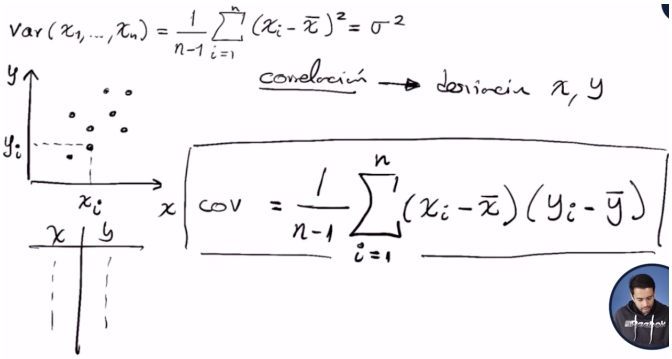
```
encoder.transform([[2016], [2009], [190]]).toarray()
```

En este caso la dimensionalidad del dataset se ve demasiado afectada, así que hay que buscar hacer una reducción de los datos.

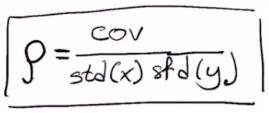
Correlaciones: covarianza y coeficiente de correlación

Si 2 variables están correlacionadas, estarían aportando la misma información, por lo que no sería útil tener las 2 variables en el modelo si su correlación es muy alta.

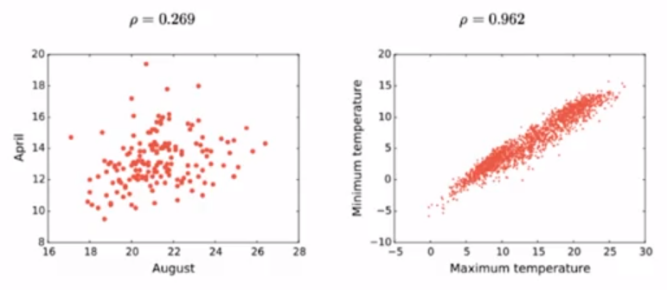
La forma de encontrar las correlaciones es usando al covarianza:



Pero como las escalas de X y Y pueden ser distintas, entonces se usa el coeficiente de correlación (ρ):

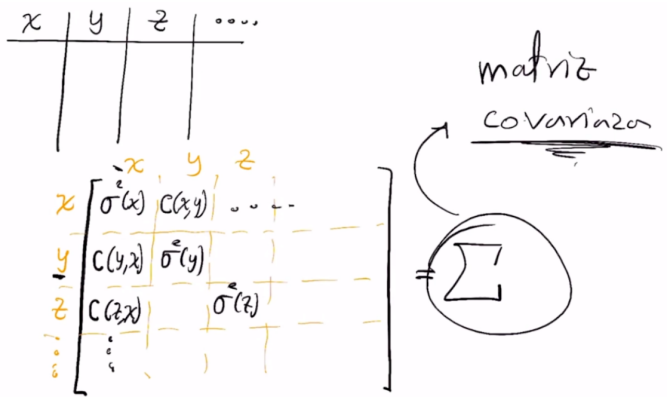


Mientas más alto sea el coeficiente de correlación (más cercano a 1), más alta es la correlación y viceversa (más cercano a 0), y si el valor es cercano a -1, entonces hay una correlación inversa:



Matriz de covarianza

Cuando hay muchas variables (lo que pasa normalmente), se debe calcular todas las posibles covarianzas de las parejas de datos del dataset. El resultado de este cálculo, representado en una matriz, es la matriz de covarianza.



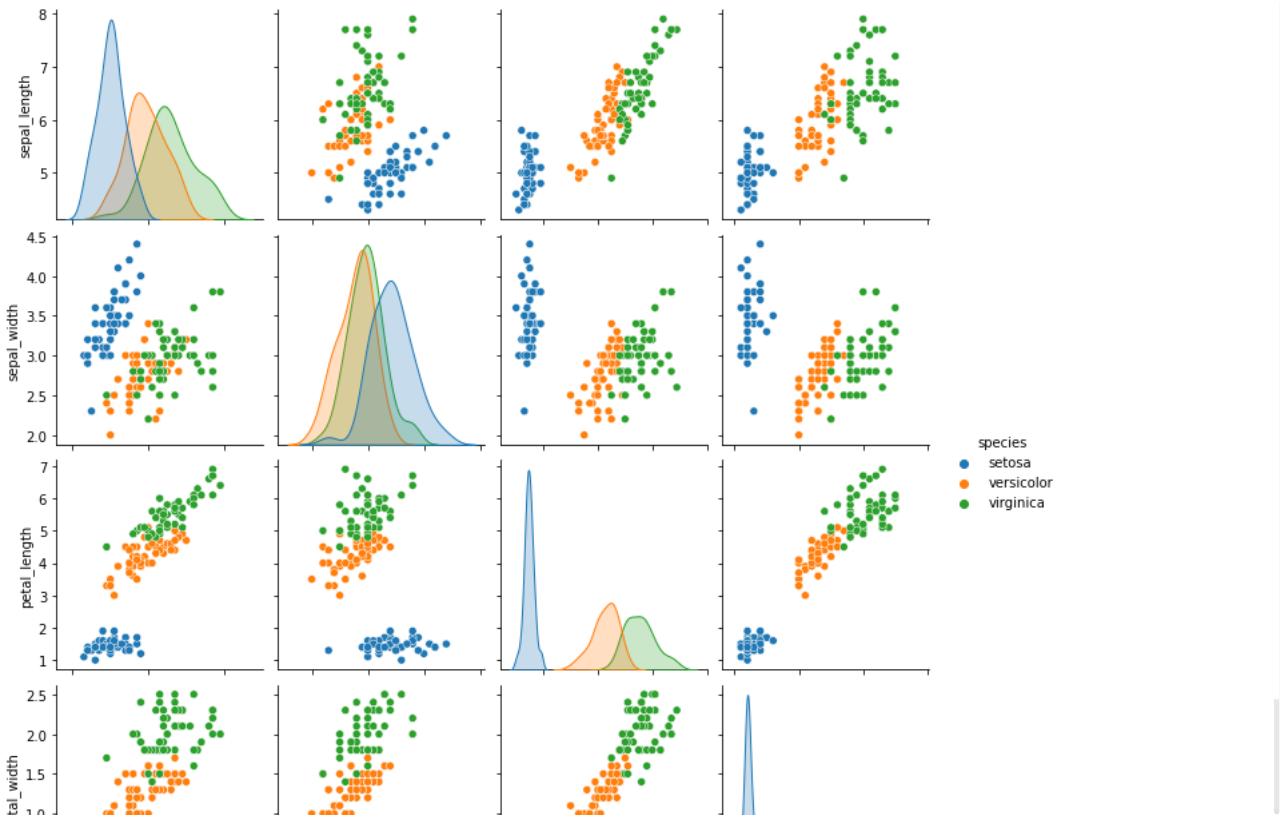
Siempre se la usa en los análisis exploratorios de datos.

Matriz de covarianza en Python

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
```

```
iris = sns.load_dataset('iris')
```

```
sns.pairplot(iris, hue='species') #este gráfico no sirve si hay demasiadas variables
```

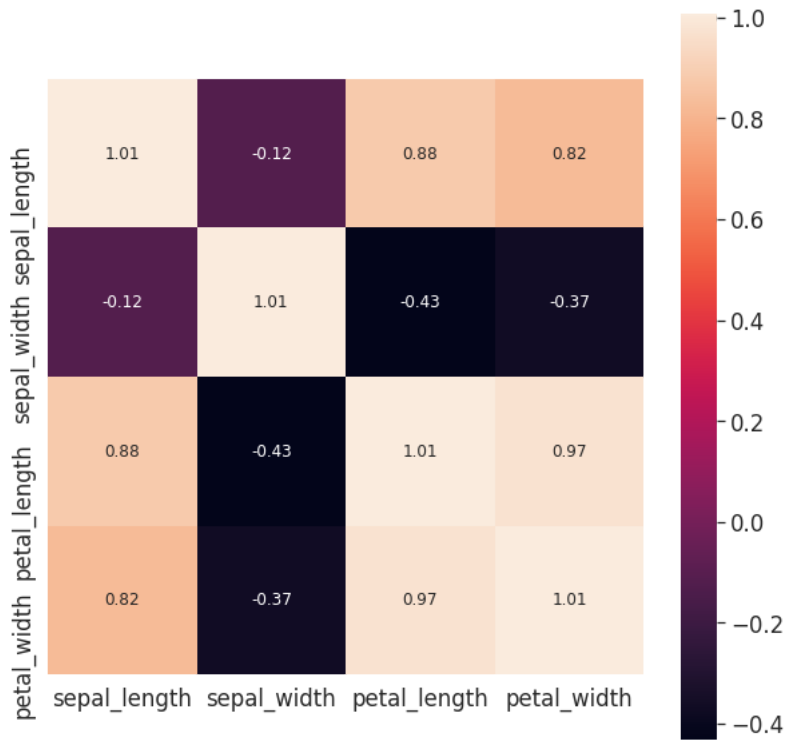



```
iris.columns
```

```
scaler = StandardScaler()
scaled = scaler.fit_transform(
    iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
)
scaled.T
```

```
covariance_matrix = np.cov(scaled.T)
covariance_matrix
```

```
# Mapa de calor de la matriz de covarianza
plt.figure(figsize=(10,10))
sns.set(font_scale=1.5)
hm = sns.heatmap(covariance_matrix,
                 cbar=True,
                 annot=True,
                 square=True,
                 fmt='.2f',
                 annot_kws={'size': 12},
                 yticklabels=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
                 xticklabels=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'])
```



Este proceso se conoce como reducción de datos y consiste en eliminar datos altamente correlacionados , se basa en la matriz de covarianza y se le llama análisis de componentes principales.

Bonus: comandos de Pandas y Seaborn usados en el curso

|Código|▶ Función| |---|---| |**Pandas**|| | `pd.read_csv('ruta')` |→ Lee el DataFrame y lo puedes almacenar en una variable {df}| | `df.dtypes` |→ Los tipos de todas las columnas| | `df.describe()` |→ Conjunto completo de estadísticos descriptivos (fundamentales)| | `df.groupby('categoria').count()` |→ Agrupa los datos por la variable categórica y los cuenta| | `df['categoria'].mean()` |→ Saca la media de todos los datos de esa categoría| | `df['category'].plot.hist(bins=20)` |→ Histograma de frecuencua con intervalos de valores (bins)| |---|---| |**Seaborn**|| | `sns.scatterplot(data=dataset, x='x_column', y='y_column')` |→ Grafica un scatterplot (diagrama de dispersión). Si añades `hue='categoría'` , aparece en colores clasificados.| | `sns.jointplot(data=dataset, x='x_column', y='y_column')` |→ Grafica un scatterplot y sus distribuciones. Si añades `hue='categoría'` , aparece en colores clasificados.| | `sns.boxplot(data=iris, x='species', y='sepal_length')` |→ Grafica los cuartiles.| | `sns.barplot(data=iris, x='species', y='sepal_length')` |→ Grafica un diagrama de barras.|