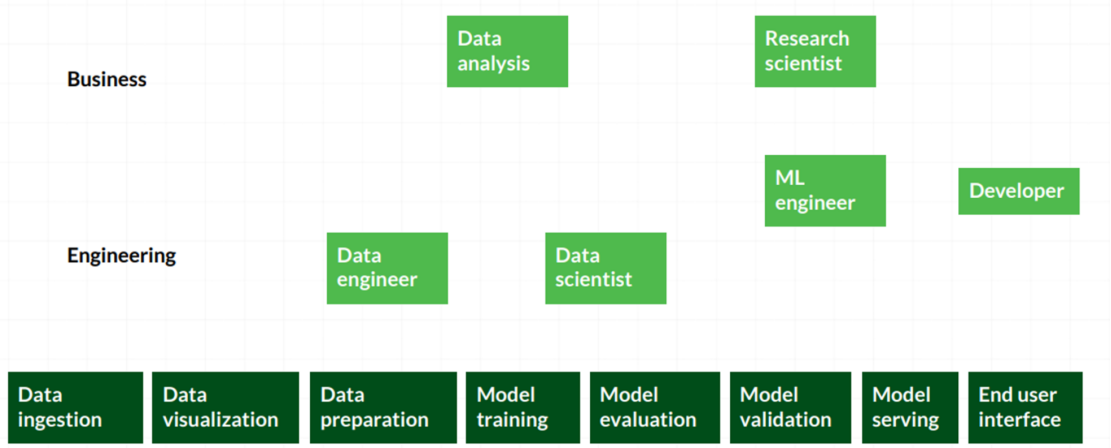
Estadística Descriptiva (Normalización de Datos) & Code

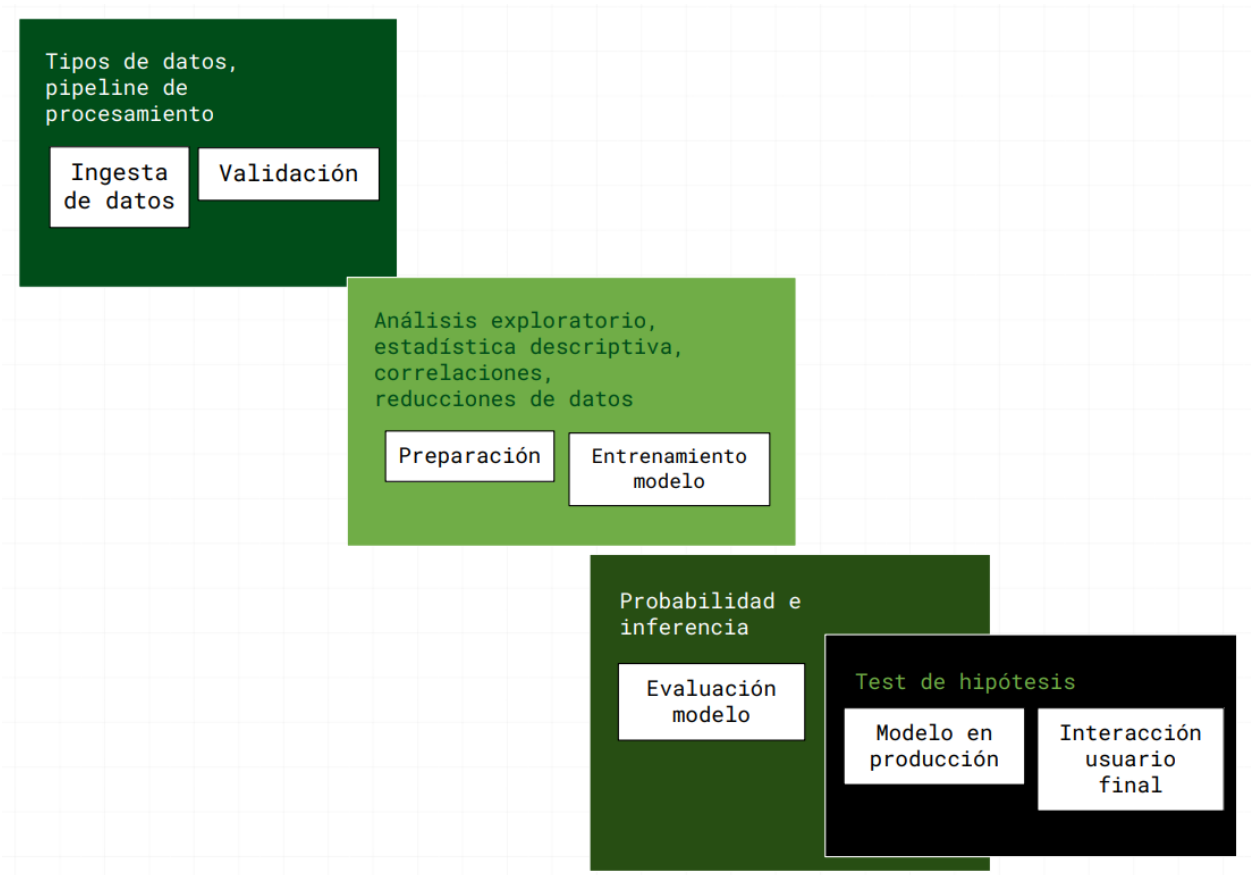
Type:	Notes
Category:	Data Data Science
Position:	**Curso de Matemáticas para Data Science: Estadística Descriptiva**
📕 Biblioteca.	Data Science

Indice

Flujo de Trabajo en Data Science



Pasos y roles en el flujo de trabajo de Data Science (Design Patterns in Machine Learning).



Tipos de Datos

Categóricos

(Genero, categoría de película, método de pago etc..)

- Ordinal
- Nominal

Numéricos

(Edad, altura, temperatura)

- Discretos
- Continuos

```
# Cargando Libreria y Archivo
import pandas as pd
df = pd.read_csv('...')

# Muestra los tipos de datos que tiene el dataframe
df.dtypes
```

```
#Aquí vemos que los tipos de datos se identifican de la siguiente
# Categoricos: object, bool
# Numéricos: int64 (discreto), float64 (contínuo)
```

Medidas de Tendencia Central

Media o Promedio

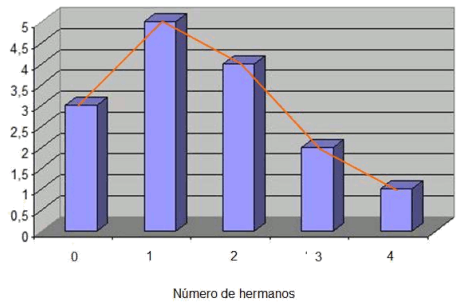
⚠️ Susceptible a valores atípicos u outliers

$$Media = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Moda

Dato que más se repite, se usa una tabla de frecuencias

$$Moda = Freq(x_k) = Max(Freq(x_k))$$



Mediana

Se toma el dato central del dataset ordenado

Mediana Impar (ordenada)

$$Med(x) = X\left\lfloor \frac{n+1}{2} \right\rfloor$$

Mediana Par (ordenada)

$$Med(x) = \frac{X\left\lfloor \frac{n}{2} \right\rfloor + X\left\lfloor \frac{n}{2} + 1 \right\rfloor}{2}$$

Medidas de Tendencias Central en Python

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

```
# Mediana
df['price_usd'].median()

# Moda a traves de un grafico
df['price_usd'].plot.hist(bins=20)

sns.displot(df,x='price_usd', hue='manufacturer_name')
```

Medidas de Dispersión

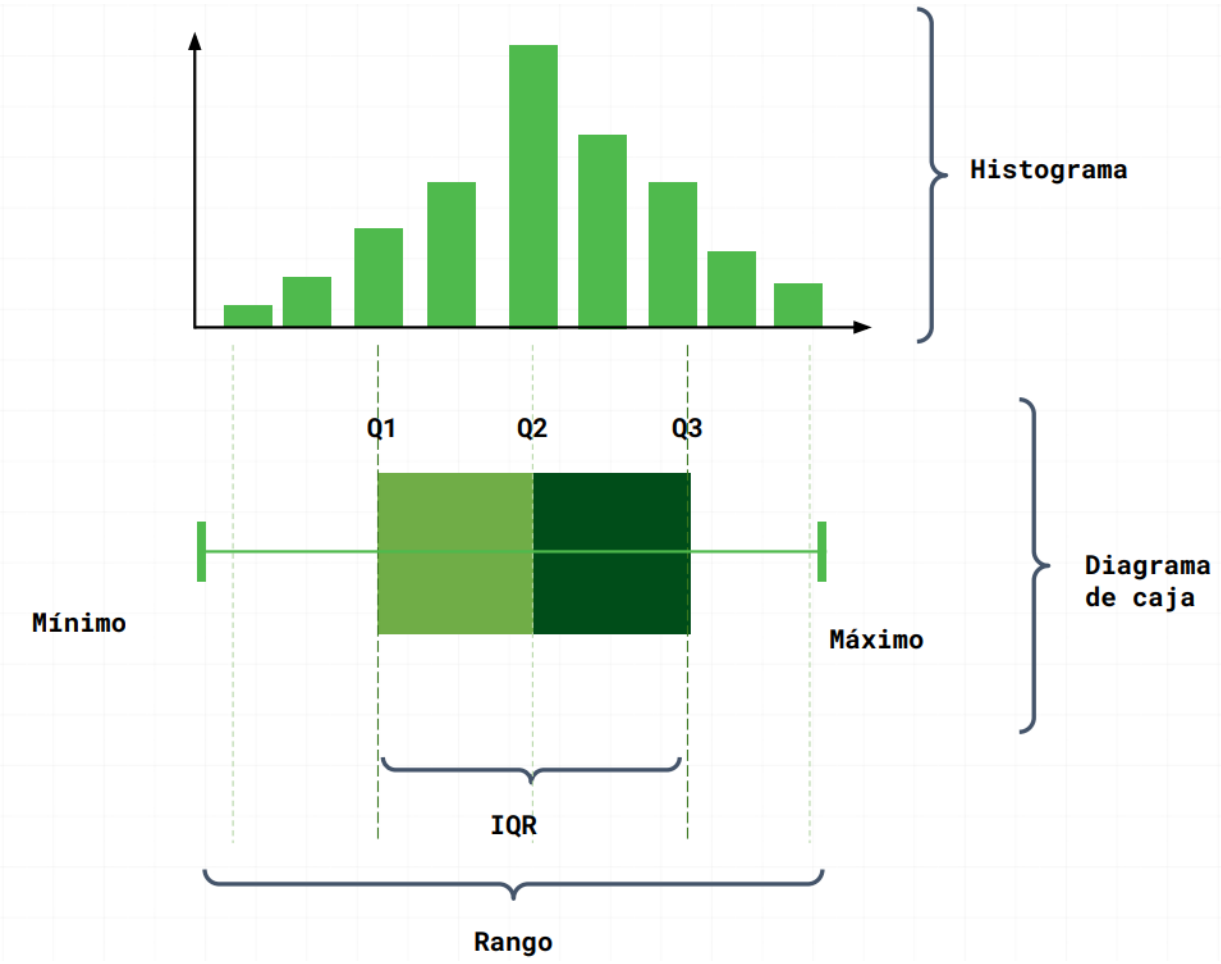
Rango, Cuartiles y Diagrama caja y bigotes

¿Qué tanto se desvían tus datos de su media?

- Rango
  - Intervalo entre valor máximo y mínimo
- Cuartiles
  - Nuestro valor se divide en **cuatro** partes iguales
  - Q1: 25% de los datos es igual o menor
  - Q2: 50%
  - Q3: 75%
  - Distancia inter-cuartil: distancia entre 1er cuartil y el 3er cuartil.

$$IQR = Q_3 - Q_1$$

- También hay **Quantile** es una generalización de los cuartiles en los que se divide el dataset en partes iguales.
- El **diagrama de cajas** representa de manera visual los cuartiles, Inter cuartiles y el rango.



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

Varianza y Desviación estándar

Varianza

$$\sigma^2 = \frac{1}{n}\sum(x_i - \mu)^2$$

- La varianza es el promedio del cuadrado de las distancias entre un punto medio y el punto a evaluar (μ es la media)

Desviación estándar

$$\sigma = \sqrt{\sigma^2}$$

- La desviación estándar es 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`.

💡 La desviación estándar nos dice, en promedio, **cuánto se desvían los datos de la media**. Si la desviación estándar es pequeña, significa que los datos están mayormente cerca de la media. Si es grande, los datos están más dispersos.

Límites para detección de outliers (datos simétricamente distribuidos)

💡 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.

Datos Simétricos

- Tomaremos únicamente los datos entre $Q_1 - 1.5 * IQR$
- Y $Q_3 + 1.5 * IQR$
- Siendo $IQR = Q_3 - Q_1$

Datos Asimétricos

Si los datos son no simétricos, entonces cambian los límites, y pasan a ser

- $Q_1 - 1.5 * f(IQR)$
- $Q_3 + 1.5 * g(IQR)$
- Siendo f y g, funciones que modelizan la asimetría de los datos.

Medidas de Dispersión en Python

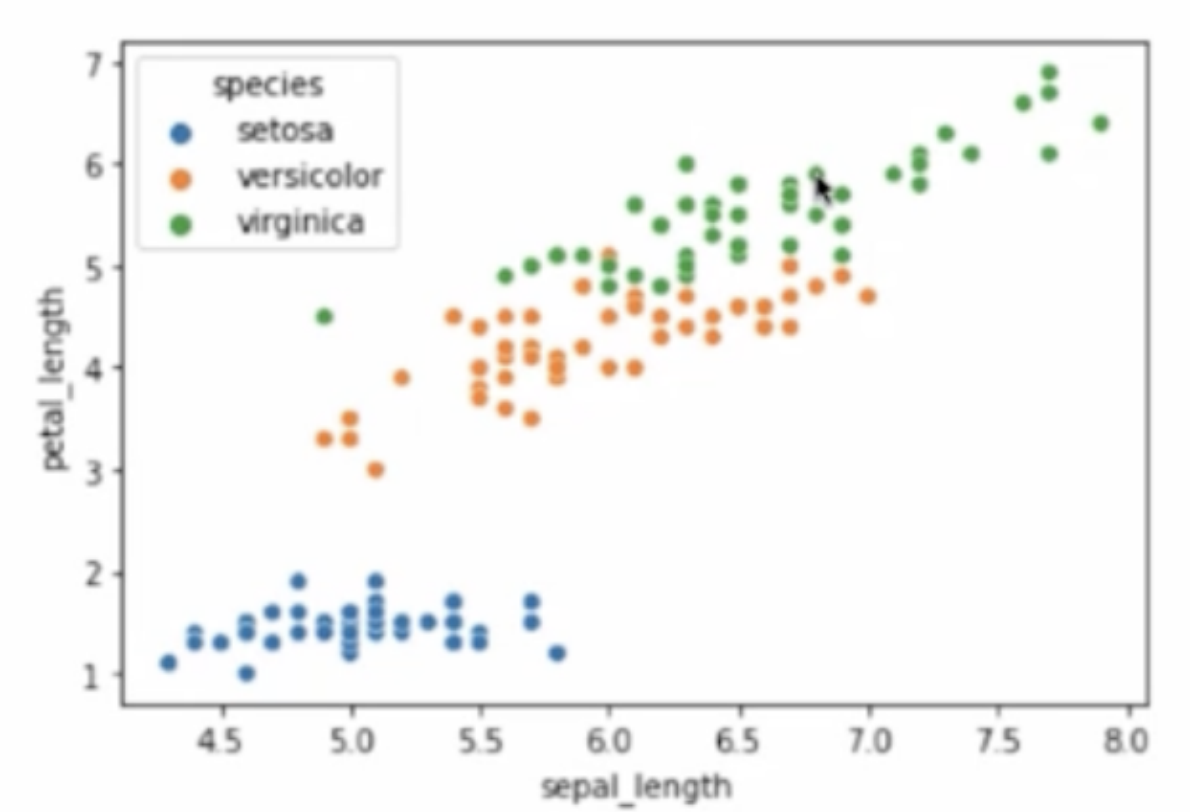
```
# Desviacion Estandar
df['price_usd'].std()

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

# Quartiles
median = df['price_usd'].median()
Q1 = df['price_usd'].quantile(q=0.25) #toma el primer 25% de todo
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)

# Rango inter quartilico
iqr = Q3 - Q1

# Outliers
inlimit = Q1 - 1.5*iqr
maxlimit = Q3 + 1.5*iqr
print('rango para detección de outliers: {}, {}'.format(minlimit,
```

Exploración visual de datos

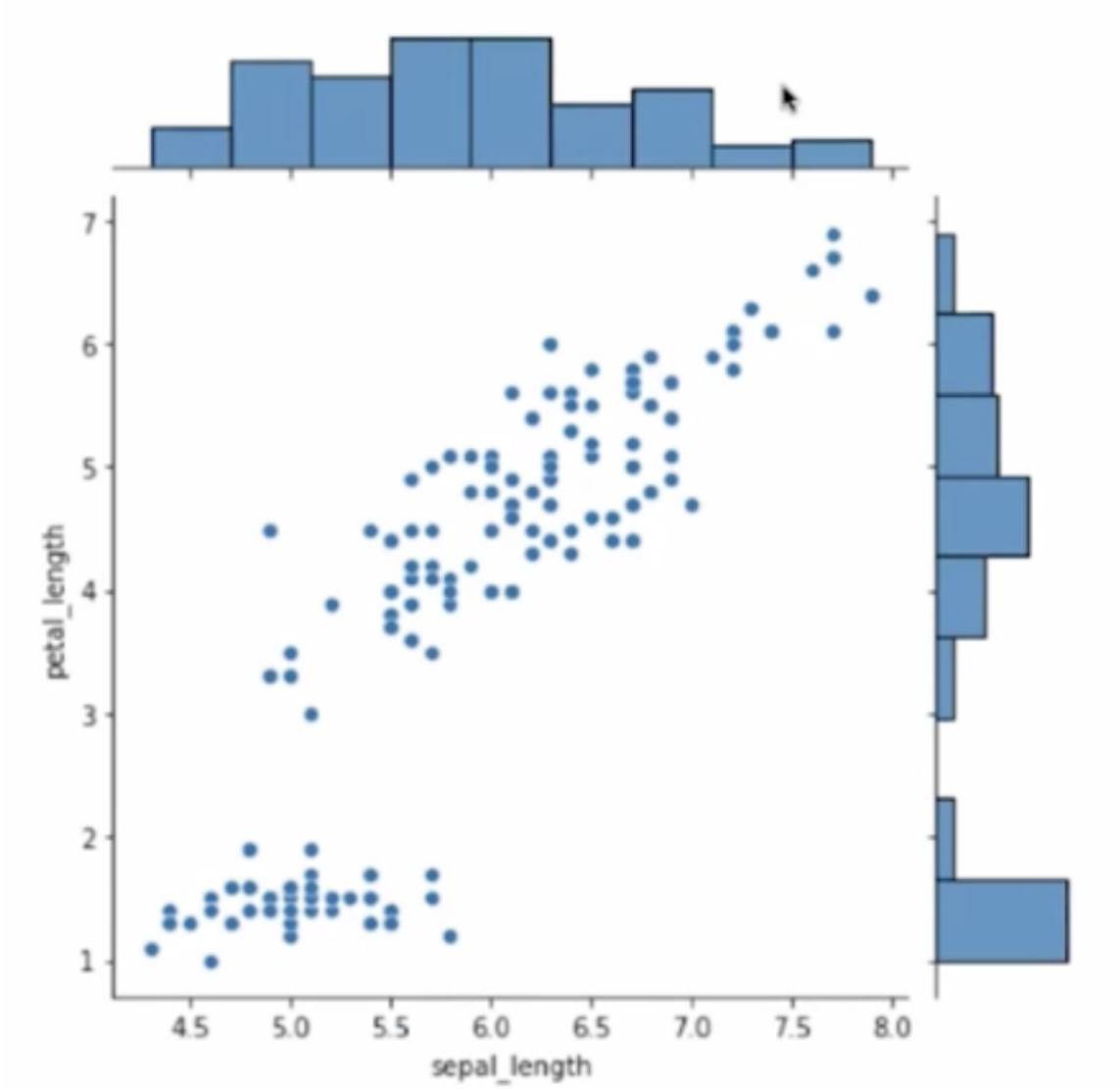
Una imagen vale más que mil palabras. Pero una **buena imagen**…

- Recursos
  - https://www.data-to-viz.com/
  - https://datavizproject.com/
- Diagrama de dispersión
  - También conocido como: Scatter plot
  - Permite contrastar dos variables de nuestro set de datos.

- La visualización es detallada porque muestra los atributos de cada elemento en el conjunto de datos.



- Join plot
  - Es un scatter plot que en cada eje muestra un histograma con la segregación de los datos



Pipelines de procesamiento de datos

Datos Numéricos - Normalización de datos

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.

- En todos los casos se intenta transformar una variable en otra variable que tiene el mismo significado pero en otro rango
  - Transformar de $X \rightarrow X_s$

Tipos de Escalamiento

- **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**.

$$X_s = \frac{2X - min - max}{max - min}$$

- **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.
  - Los valores mínimos y máximos se pueden escoger de manera arbitraria.
  - En caso de que $X > X_{max}; X_s = X_{max}$
  - Y en caso de que $X < X_{min}; X_s = X_{min}$
- **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. **Funcion mejor para datos distribuidos "normalmente" (forma de campana de Gauss)**.

$$X_s = \frac{X - \mu}{\sigma}$$

Escalamiento Lineal en Python

Se utilizará este dataset de scikit-learn: https://scikit-learn.org/stable/

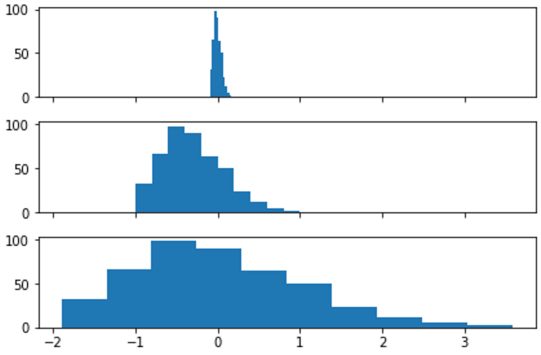
Normalización de Datos

```
import timeit
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
from sklearn import datasets,
```

Entrenamiento del Modelo vs Tiempos de ejecución

```
# modelos para entrenamiento
def train_raw():
    linear_model.LinearRegress
```

```
X, y = datasets.load_diabetes(
raw = X[:, None, 2]

# escalamiento max-min
max_raw = max(raw)
min_raw = min(raw)
scaled = (2*raw - max_raw -min

# normalización Z-score
avg = np.average(raw)
std = np.std(raw)
z_scaled = (raw - avg)/std

# Creacion del grafico
fig, axs = plt.subplots(3, 1,

axs[0].hist(raw)
axs[1].hist(scaled)
axs[2].hist(z_scaled)
```




```
def train_scaled():
    linear_model.LinearRegress

def train_z_scaled():
    linear_model.LinearRegress


raw_time = timeit.timeit(train
scaled_time = timeit.timeit(tr
z_scaled_time = timeit.timeit(
print('trainning time for raw
print('trainning time for scal
print('trainning time for z_sc
```

```
trainning time for raw data : 0.07451489300001413
trainning time for scaled data : 0.06742551799834473
trainning time for z_scaled data : 0.06812811800045893
```

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

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

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

- max-min scaling: mejor para datos uniformemente distribuidos
- z-score scaling: mejor para datos distribuidos "normalmente" (forma de campana de gauss)

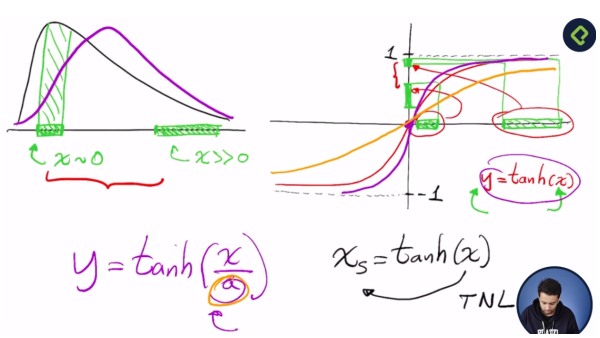
Transformaciones no lineales

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

> 💡 La idea básica es aplicar una transformación que transforme nuestros datos, en datos simétricos
>
> - $X_s = f(X)$ siendo f una función no lineal auxiliar.

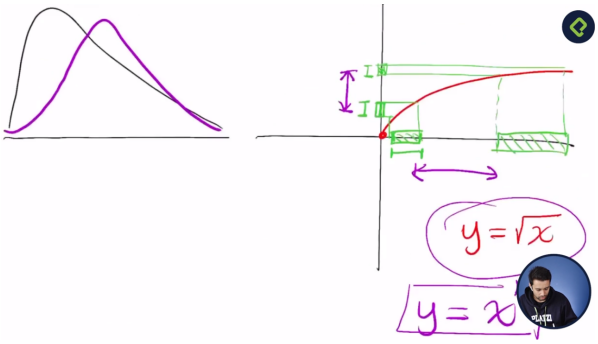
Tanh(x)

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.



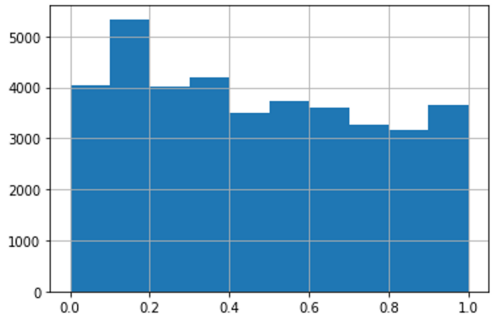
Hay un abanico gigante de maneras para hacer estas transformaciones con funciones no lineales. Los anteriores fueron solo 2 ejemplos. Las más famosas son la tanh y la sigmoide porque ambas amplían los valores importantes y acercan al 0, mientras que a los outliers los minimizan y acercan a -1 y 1.

https://scikit-learn.org/stable/auto_examples/preprocessing/plot_map_data_to_normal.html

https://www.youtube.com/watch?v=pKtkkL7tmmQ

Transformaciones no lineales en Python

```
# Trabajando con el mismo dataset'
# Transformación con tanh(x)

# Esta línea toma la columna y le
p = 10000
df.price_usd.apply(lambda x: np.ta
```



Datos Categóricos - Mapeos Numericos

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()`.

- ⚠️ El One Hot encoding puede aumentar demasiado la dimensionalidad del dataset y eso puede afectar el rendimiento

Mapeos Numéricos en Python

```
# Pandas
import pandas as pd
pd.get_dummies(df['engine_type'])

# Scikit Learn
import sklearn.preprocessing as preprocessing
encoder = preprocessing.OneHotEncoder(handle_unknown='ignore')
encoder.fit(df[['engine_type']].values)
```

```
encoder.transform([['gasoline'],['diesel'], ['aceite']]).toarray(
ncoder.fit(df[['year_produced']].values)
encoder.transform([[2016], [2009], [190]]).toarray()
```

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

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

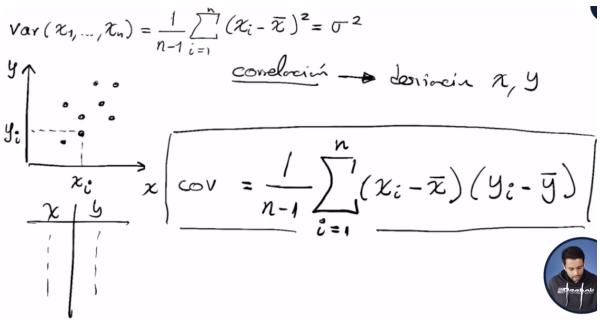
Correlaciones: covarianza y coeficiente de correlación

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.

- ⚠️ Correlación no implica causa (puede que sí, puede que no)

La forma de encontrar las correlaciones es usando al covarianza:

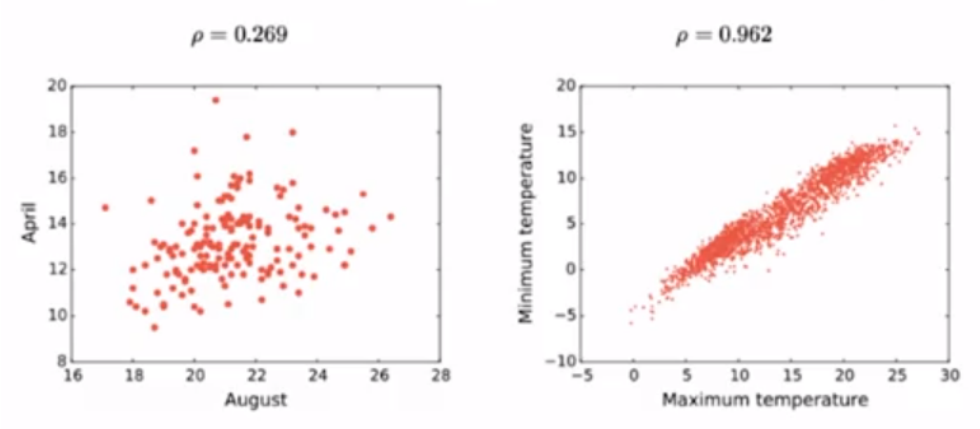
$$Cov = \frac{1}{n-1}\sum(X_i - \bar{X})(Y_i - \bar{Y})$$



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

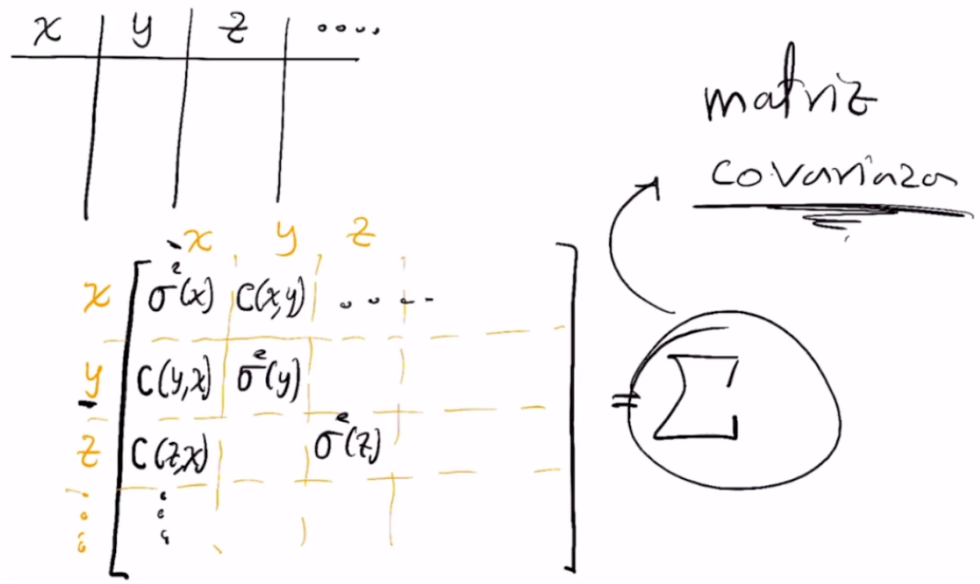
$$p = \frac{Cov}{std(X)std(y)}$$

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:



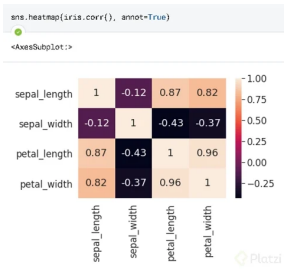
Matriz de covarianza

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.



Matriz de Covarianza en Python

```
# Usando Seaborn
# Metodo 1
sns.heatmap(iris.corr(), annot=True)

# Metodo 2
```



```
corr_matrix = iris.select_dtypes(include='numbe
sns.heatmap(corr_matrix, annot=True)
plt.show()

# Usando Skelearn
scaler = StandardScaler()
scaled = scaler.fit_transform(
    iris[['sepal_length', 'sepal_width', 'petal
)

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', '
                 xticklabels=['sepal_length', '
```

PCA → Análisis de componentes principales

Se trabaja que a mayor cantidad de varianza, mayor cantidad de información se tiene.

- Permiten reducir el número de componentes y la dimensionalidad
- Se buscan obtener las “direcciones” en las que la varianza es mayor y más representativa.

- Es una transformación lineal de un set de datos con gran cantidad de de variables/columnas a otro set de datos con menos variables y menos dimensiones pero manteniendo la “representatividad”

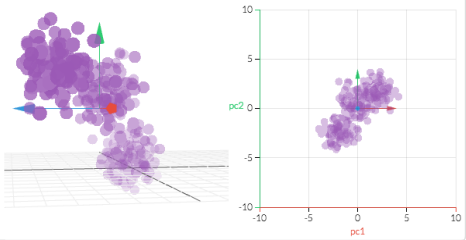
Reducción de Dimensionalidad

1. Estandarizar datos
2. Matriz de covarianza
3. Eigenvalores & Eigenvectores
4. Proyección de los datos

Principal Component Analysis explained visually

Principal component analysis (PCA) is a technique used to emphasize variation and bring out strong patterns in a dataset. It's often used to make data easy to explore and visualize.

https://setosa.io/ev/principal-component-analysis/



https://www.youtube.com/watch?v=AniiwysJ-2Y

Bonus

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.

References:

slides-curso-estadistica-descriptiva.pdf

https://datavizproject.com/

https://www.data-to-viz.com/

https://scikit-learn.org/stable/

https://scikit-learn.org/stable/auto_examples/preprocessing/plot_map_data_to_normal.html

https://www.youtube.com/watch?v=pKtkkL7tmmQ

https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

https://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features