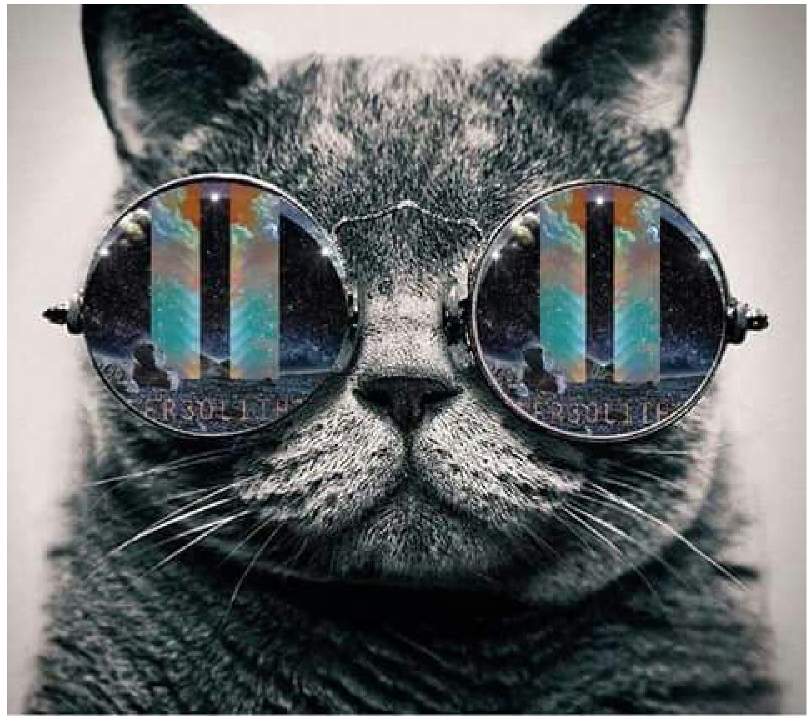
TOULOUSE LAUTREC

APRENDIZAJE AUTOMATICO CON PYTHON

REGRESION LINEAL - COMPARACION DE MODELOS



Ing. Alexander Valdez

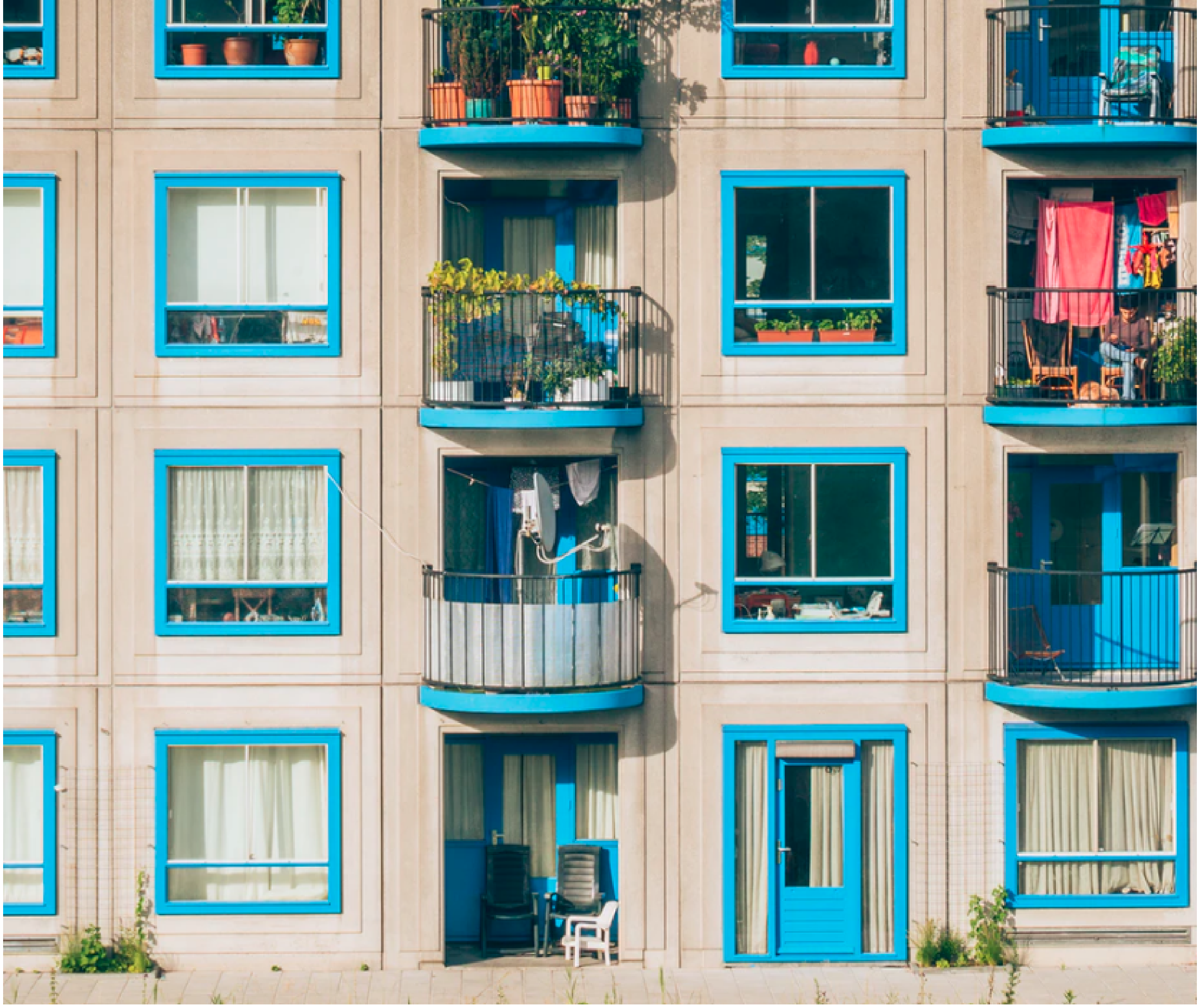
Curso 2290, Clases Lunes y Miercoles 20:00-22:30pm

Segunda Clase

Presentación del caso

Predecir o estimar el precio de una vivienda puede ser de gran ayuda a la hora de tomar decisiones importantes tales como la adquisición de casa propia . A continuación se presenta un dataset compuesto por **25, 660 registros** para **Argentina y Colombia** adjunto a las siguientes **10 variables**:

```
1.   pais : "Argentina", "Colombia"
2.   provincia_departamento: Provincia o departamento (no ambas)  donde se
ubica el departamento
3.   ciudad: Ciudad donde se ubica el departamento
4.   property_type: "Departamento" (siempre en Argentina), "Apartamento"
(siempre en Colombia)
5.   operation_type: "Venta"
6.   rooms: cantidad de espacios en general dentro del apartamento
7.   bedrooms: cantidad de cuartos donde dormir dentro del apartamento
8.   bathrooms: cantidad de baños dentro del apartamento
9.   surface_total: área total en metros cuadrados del departamento
10.  currency: "USD" (dólar americano)
```



Lectura de los datos

In [ ]:

```
"CELDA N°1"
#Importamos los datos de train y test considerando separador, índice y tipos de datos
import pandas as pd
data=pd.read_csv("https://raw.githubusercontent.com/sebastianVP/Datasets_TOULOUSE_ML/mai
```

In [ ]:

```
"CELDA N°2"
#Comprobamos que la train tenga 25,660 filas y tanto train como test tengan 10 variables
data.shape
data.head()
```

Out[ ]:

Id	pais	provincia_departamento	ciudad	property_type	operation_type	rooms	bedrooms	bathrooms
0	Argentina	Capital Federal	Villa Crespo	Departamento	Venta	2	1	1
1	Argentina	Capital Federal	Palermo	Departamento	Venta	6	4	4
2	Colombia	Atlantico	Barranquilla	Apartamento	Venta	3	3	3
3	Colombia	Valle del Cauca	Cali	Apartamento	Venta	3	3	1
4	Argentina	Capital Federal	Balvanera	Departamento	Venta	3	2	1

Eliminando variables no relevantes

In [ ]:
```
"CELDA N°3"
#Comprobamos que algunas columnas tienen un único valor en train
for col in data.columns:
  if data[col].nunique()<3:
    print('En la columna',col,'hay',data[col].nunique(),'valores distintos')
  else:
     pass #no realizar ninguna acción si es que el número de valores distintos es mayor
```

```
En la columna pais hay 2 valores distintos
En la columna property_type hay 2 valores distintos
En la columna operation_type hay 1 valores distintos
En la columna currency hay 1 valores distintos
```

Las columnas **pais, property_type, operation_type, currency** tienen apenas 1 o 2 valores repetidos en toda la columna.

Por eso eliminamos estas columnas que no aportan data significativa para predecir el precio del departamento, **excepto la columna pais** porque será necesaria después para traer data externa.

In [ ]:
```
"CELDA N°4"
#Eliminamos las columnas mencionadas con el método drop. No olvidar incluir inplace = Tr
data.drop(columns=['property_type', 'operation_type', 'currency'], inplace=True)
```

In [ ]:
```
data.head()
```

Out[ ]:

Id	pais	provincia_departamento	ciudad	rooms	bedrooms	bathrooms	surface_total	price
0	Argentina	Capital Federal	Villa Crespo	2	1	1	37	85000
1	Argentina	Capital Federal	Palermo	6	4	4	300	1590000
2	Colombia	Atlantico	Barranquilla	3	3	3	95	85329
3	Colombia	Valle del Cauca	Cali	3	3	1	60	22846
4	Argentina	Capital Federal	Balvanera	3	2	1	45	80000

Preprocesamiento de los datos

I. Verificación de datos perdidos

In [ ]:
```
"CELDA N°5"
#Verificamos que ninguna columna de train tenga vacíos
for col in data.columns:
  if data[col].isna().sum()>0:
    print('En la columna',col,'hay',data[col].isna().sum(),'valores nulos')
```

Si al ejecutar la celda anterior **no obtenemos resultado** se debe a que no se encontró ninguna columna que tenga datos perdidos.

II. Verificación de outliers

Con ayuda del método **skew** descrita por la distribución de cada variable numérica detectaremos la presencia de **outliers**.

¿Cómo interpretar el valor de Skewness?

Medida estadística que describe la simetría de la distribución alrededor de un promedio. Si el sesgo es igual a cero, la distribución es simétrica; si el sesgo es positivo la distribución una tendrá una cola asimétrica extendida hacia los valores positivos.

In [ ]:
```
data_to_skew = data.select_dtypes(include = ["number"])
```

In [ ]:
```
data_to_skew
```

Out[ ]:

Id	rooms	bedrooms	bathrooms	surface_total	price
0	2	1	1	37	85000
1	6	4	4	300	1590000
2	3	3	3	95	85329
3	3	3	1	60	22846
4	3	2	1	45	80000
...	...	...	...	...	...
25655	3	3	2	61	41288
25656	2	1	1	40	85000
25657	2	1	1	61	185700
25658	3	2	1	53	120000
25659	2	1	1	45	63000

25660 rows × 5 columns

In [ ]:
```
"CELDA N°6"
#Previamente seleccionamos las variables numéricas y en cada una calculamos el skew para
data_to_skew = data.select_dtypes(include = ["number"])

skew = []
for col in data_to_skew.columns:
  skew.append(data_to_skew[col].skew())

skewness=pd.DataFrame(index=data_to_skew.columns) #declaramos como índices a las columna
skewness["Skewness"] = skew
skewness.sort_values(by=["Skewness"], ascending=False) #ordenamos de mayor a menor
```

Out[ ]:

	Skewness
price	6.026587
surface_total	2.081696
bathrooms	1.378065
rooms	1.222084
bedrooms	0.522811

Para corregir el **alto skewness** de las columnas **price y surface_total** aplicaremos una **transformación logarítmica**.

In [ ]:
```
"CELDA N°7"
#Usamos una función lambda para realizar una transformación logarítmica usando numpy sob
import numpy as np
data["price"] = data["price"].map(lambda i: np.log(i) if i > 0 else 0)
print('El skewness después de la transformación logarítmica es: ',data["price"].skew())
```

```
El skewness después de la transformación logarítmica es:  0.8850647322573656
```

In [ ]:
```
"CELDA N°8"
#Usamos una función lambda para realizar una transformación logarítmica usando numpy sob
data["surface_total"] = data["surface_total"].map(lambda i: np.log(i) if i > 0 else 0)
print('El skewness después de la transformación logarítmica es: ',data["surface_total"].
```

```
El skewness después de la transformación logarítmica es:  -0.0539266496430324
```

III. Feature Engineering

Añadimos data externa

Juntando **país y provincia** obtenemos datos estadísticos sobre cada **provincia**

In [ ]:
```
"CELDA N°9"
#Leemos la tabla externa pais_provincia con promedio, medianas y percentiles según provi
pais_provincia=pd.read_csv("https://raw.githubusercontent.com/sebastianVP/Datasets_TOULO

pais_provincia.columns
```

Out[ ]:
```
Index(['pais_provincia', 'promedio_provincia', 'mediana_provincia',
       'percentil10_provincia', 'percentil25_provincia',
       'percentil75_provincia', 'percentil90_provincia'],
      dtype='object')
```

In [ ]:
```
pais_provincia.head()
```

Out[ ]:

	pais_provincia	promedio_provincia	mediana_provincia	percentil10_provincia	percentil25_provincia	percentil
0	Argentina_Bs.As. G.B.A. Zona Norte	169619.4007	130000	60000.0	85000	
1	Argentina_Bs.As. G.B.A. Zona Oeste	114376.3303	85000	54000.0	69000	
2	Argentina_Bs.As. G.B.A. Zona Sur	120535.2642	89000	52000.0	68000	
3	Argentina_Buenos Aires Costa Atlantica	105516.5815	79900	49000.0	62900	
4	Argentina_Buenos Aires Interior	139372.0686	95000	49600.0	70000	

Añadimos **nuevas columnas** al final con ayuda del método **merge**. Previamente **concatenamos el país y provincia**

In [ ]:
```
"CELDA N°10"
```

```
#Para concatenar ambas columnas simplemente usamos el operador + con el guión bajo entre
data['pais_provincia'] = data['pais']+ "_" + data['provincia_departamento']

#Para adjuntar los datos de la tabla externa usamos el método merge especificando left (
data = data.merge(pais_provincia, on='pais_provincia', how='left')
```

In [ ]:
```
data.head()
```

Out[ ]:

	pais	provincia_departamento	ciudad	rooms	bedrooms	bathrooms	surface_total	price	pa
0	Argentina	Capital Federal	Villa Crespo	2	1	1	3.610918	11.350407	Arge
1	Argentina	Capital Federal	Palermo	6	4	4	5.703782	14.279245	Arge
2	Colombia	Atlantico	Barranquilla	3	3	3	4.553877	11.354270	Coloml
3	Colombia	Valle del Cauca	Cali	3	3	1	4.094345	10.036531	Colom
4	Argentina	Capital Federal	Balvanera	3	2	1	3.806662	11.289782	Arge

Juntando **provincia y ciudad** obtenemos datos estadísticos sobre cada **ciudad**

In [ ]:
```
"CELDA N°11"
#Leemos la tabla externa provincia_ciudad con promedio, medianas y percentiles según ciu
provincia_ciudad=pd.read_csv("https://raw.githubusercontent.com/sebastianVP/Datasets_TOU
provincia_ciudad.columns
```

Out[ ]:
```
Index(['provincia_ciudad', 'area_ciudad', 'altura_ciudad', 'habitantes_ciudad',
       'densidad_ciudad', 'promedio_ciudad', 'mediana_ciudad',
       'percentil10_ciudad', 'percentil25_ciudad', 'percentil75_ciudad',
       'percentil90_ciudad'],
      dtype='object')
```

In [ ]:
```
provincia_ciudad.head()
```

Out[ ]:

	provincia_ciudad	area_ciudad	altura_ciudad	habitantes_ciudad	densidad_ciudad	promedio_ciudad	mediana_
0	Antioquia_Medellin	382.00	1495.0	2529403.0	6643.39	263200.0	
1	Antioquia_Envigado	78.78	1575.0	228848.0	2904.00	301400.0	
2	Antioquia_Bello	149.00	1310.0	522264.0	2610.22	182300.0	
3	Antioquia_Sabaneta	15.00	1551.0	82375.0	3638.20	255900.0	
4	Antioquia_Itagui	21.09	1550.0	276744.0	11143.48	176200.0	

Para nuestro caso de **predicción de precios** nos enfocaremos **exclusivamente** en las variables que sean relevantes para este objetivo.

Por ello vamos a **filtrar** esas columnas del dataset provincia_ciudad.

In [ ]:
```
"CELDA N°12"
#Filtramos solo las columnas relativas al precio con el método loc
provincia_ciudad = provincia_ciudad.loc[:,['provincia_ciudad',
                                           'promedio_ciudad',
                                           'mediana_ciudad',
                                           'percentil10_ciudad',
                                           'percentil25_ciudad',
```

```
                                'percentil75_ciudad',
                                'percentil90_ciudad']]
```

In [ ]:

```

```

Añadimos **nuevas columnas** al final con ayuda del método **merge**. Previamente **concatenamos la provincia y ciudad**

In [ ]:

```
"CELDA N°13"
#Para concatenar ambas columnas simplemente usamos el operador + con el guión bajo entre
data['provincia_ciudad'] = data['provincia_departamento']+ "_" + data['ciudad']

#Para adjuntar los datos de la tabla externa usamos el método merge especificando left (
data = data.merge(provincia_ciudad, on='provincia_ciudad', how='left')
```

In [ ]:

```
data.head()
```

Out[ ]:

	pais	provincia_departamento	ciudad	rooms	bedrooms	bathrooms	surface_total	price	pa
0	Argentina	Capital Federal	Villa Crespo	2	1	1	3.610918	11.350407	Arge
1	Argentina	Capital Federal	Palermo	6	4	4	5.703782	14.279245	Arge
2	Colombia	Atlantico	Barranquilla	3	3	3	4.553877	11.354270	Coloml
3	Colombia	Valle del Cauca	Cali	3	3	1	4.094345	10.036531	Colom
4	Argentina	Capital Federal	Balvanera	3	2	1	3.806662	11.289782	Arge

5 rows × 22 columns

In [ ]:

```
data.shape
```

Out[ ]:

```
(25660, 22)
```

IV. Supuestos para Modelos de Regresión

Estandarización de las variables numéricas predictoras

Vamos a **estandarizar** usando StandardScaler modificando la distribución de los datos para asegurar la **normalidad** de los datos.

StandardScale

estándariza los datos eliminando la media y escalando los datos de forma que su varianza sea igual a 1. Este tipo de escalado suele denominarse frecuentemente "normalización" de los datos. Veamos un ejemplo. Partimos del siguiente conjunto de datos:

In [ ]:

```
"CELDA N°14"
#Seleccionamos solo las columnas numéricas de la data total
data_to_standar = data.select_dtypes(include = ["number"])

#Aplicamos la librería StandardScaler
```

```
from sklearn.preprocessing import StandardScaler
data_standar = StandardScaler().fit_transform(data_to_standar)
data = pd.DataFrame(data=data_standar, columns=data_to_standar.columns)
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(25660, 17)
```

Multicolinealidad

La multicolinealidad ocurre cuando las variables independientes (predictores) en un modelo de regresión están correlacionadas.

Las variables independientes deberían ser eso, independientes. Y esto se debe a que si el grado de correlación entre las variables independientes es alto, no podremos aislar la relación entre cada variable independiente y la variable dependiente (respuesta).

¡Si no podremos aislar los efectos podríamos confundir sus efectos!

Es decir, cuando las variables independientes están muy correlacionadas los cambios en una variable están asociados con cambios en otra variable y, por tanto, los coeficientes de regresión del modelo ya no van a medir el efecto de una variable independiente sobre la respuesta manteniendo constante, o sin variar, el resto de predictores.

La multicolinealidad provoca 3 tipos de problemas:

El valor de los coeficientes de regresión del modelo cambia si incluyes o no otras variables independientes, y por lo tanto dificulta la interpretación del modelo. Se reduce su precisión de nuestras estimaciones (aumenta el error estándar de los coeficientes de regresión). La significación estadística (p-valor) de los coeficientes de regresión del modelo se vuelve menos confiable, como consecuencia del ítem anterior, y por lo tanto es difícil identificar variables independientes a incluir en el modelo. Recuerda "¿CÓMO SELECCIONAR LAS VARIABLES ADECUADAS PARA TU MODELO?

La multicolinealidad, o correlaciones altas entre las variables independientes, puede detectarse a veces observando la matriz de correlación. Otras veces, la multicolinealidad es más sutil, siendo una combinación lineal no obvia de dos o más de las variables independientes. En este último vaso, podemos utilizar el factor de inflación de la varianza (VIF) para detectar la multicolinealidad. Los VIF comienzan en 1 y no tienen límite superior.

Hay diferentes puntos críticos que los estadísticos consideran para hablar de problemas de multicolinealidad (¿un VIF>5 o VIF>10? ¡ouch!). Pero la regla (tácita) más general es la siguiente:

- VIF=1 significa que no existe correlación entre esta variable independiente y cualquier otra.
- 1< VIF < 5 sugiere una correlación moderada pero no sería necesario resolverla.
- VIF > 5 son niveles críticos de multicolinealidad.

Vamos a verificar la **multicolinearidad** utilizando el método del **Variance Inflation Factor(VIF)**.

Para mayor información consultar la siguiente página

In [ ]:
```
"CELDA N°15"
#Aplicamos variance_inflation_factor sobre todas las columnas para obtener un dataframe
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
vif = pd.DataFrame()
vif["features"] = data.columns
vif["vif_Factor"] = [variance_inflation_factor(data.values, i) for i in range(data.shape
vif
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Out[ ]:

	features	vif_Factor
0	rooms	4.824583
1	bedrooms	6.871245
2	bathrooms	3.298516
3	surface_total	7.867226
4	price	4.845436
5	promedio_provincia	304.722250
6	mediana_provincia	231.320329
7	percentil10_provincia	114.810387
8	percentil25_provincia	199.430680
9	percentil75_provincia	336.628552
10	percentil90_provincia	274.802734
11	promedio_ciudad	1137.260505
12	mediana_ciudad	217.258956
13	percentil10_ciudad	59.271647
14	percentil25_ciudad	1.017143
15	percentil75_ciudad	219.300268
16	percentil90_ciudad	101.878999

Se evidencia que los datos externos superan por mucho el límite permisible. Así como también algunas variables originales.

Para corregirlo haremos uso del **PCA** (sin considerar el precio).

In [ ]:

```
"CELDA N°16"
#Separamos las variables predictoras del target
X = data.drop('price',axis=1)
y = data['price']
```

In [ ]:

```
"CELDA N°17"
#Aplicamos PCA sobre las variables predictoras y actualizamos X para obtener un nuevo da
import numpy as np
from sklearn.decomposition import PCA
pca = PCA(n_components=6)
components=pca.fit_transform(X)
X=pd.DataFrame(data=components,columns=['PCA1','PCA2','PCA3','PCA4','PCA5','PCA6'])

vif = pd.DataFrame()
vif["features"] = X.columns
vif["vif_value"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif
```

Out[ ]:

	features	vif_value
0	PCA1	1.0
1	PCA2	1.0
2	PCA3	1.0
3	PCA4	1.0
4	PCA5	1.0
5	PCA6	1.0

Ahora ya hemos cumplido los **supuestos** para ejecutar un modelo de **regresión**:

- Distribución normal (sin outliers) de las variables predictoras
- Independencia entre las variables predictoras

V. Preparamos los datos para el modelo

Finalmente, con ayuda de la librería **train_test_split** dividimos la data de **train**: 85% para entrenamiento y **15%** para validación

In [ ]:
```
"CELDA N°18"
#Con la librería train_test_split generamos los datos de entrenamiento y validación
from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid= train_test_split(X,y,test_size = 0.15,random_state=1
```

Modelamiento y Evaluación

Importamos las librerías de **Scikit Learn** para realizar:

1. Regresión Lasso
2. Regresión Ridge

In [ ]:
```
"CELDA N°19"
#Obtenemos los modelos de la librería linear_models de sklearn
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge

#Desactivamos los mensajes de advertencia para mejor visibilidad de los resultados de ca
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
"CELDA N°20"
#Utilizaremos  el indicador denominado MSLE para obtener el error logarítmico promedio
from sklearn.metrics import mean_squared_log_error
```

In [ ]:
```
"CELDA N°21"
#Obtenemos el MSLE aplicando Lasso variando el valor de alpha
for i in range(1,10):
    lasso = Lasso(alpha=i/100).fit(X_train,y_train)
    lasso_predictions=lasso.predict(X_valid)
    print("Mi MSLE es: ", mean_squared_log_error(abs(y_valid),abs(lasso_predictions)), "
```

```
Mi MSLE es:  0.05450044535546668 cuando alpha es:  0.01
```

```
Mi MSLE es:  0.054883735821263074 cuando alpha es:  0.02
Mi MSLE es:  0.05567818466692163 cuando alpha es:  0.03
Mi MSLE es:  0.056846166848497866 cuando alpha es:  0.04
Mi MSLE es:  0.05842181813836395 cuando alpha es:  0.05
Mi MSLE es:  0.06036424753898551 cuando alpha es:  0.06
Mi MSLE es:  0.06271178101536869 cuando alpha es:  0.07
Mi MSLE es:  0.06545731883217724 cuando alpha es:  0.08
Mi MSLE es:  0.06860346677978614 cuando alpha es:  0.09
```

Para mayor detalle del porqué un valor acordado para **alpha = 0.01** sin basarnos en una base **teórica** acudir al video **minuto 26** de la clase dictada por **AndreNg en Standford**.

In [ ]:
```
"CELDA N°22"
#Obtenemos el MSLE aplicando Lasso variando el valor de alpha
for i in range(1,10):
    ridge = Ridge(alpha=i/100).fit(X_train,y_train)
    ridge_predictions=ridge.predict(X_valid)
    print("Mi MSLE es: ", mean_squared_log_error(abs(y_valid),abs(ridge_predictions)), "
```

```
Mi MSLE es:  0.054490346410072596 cuando alpha es:  0.01
Mi MSLE es:  0.05449033719651285 cuando alpha es:  0.02
Mi MSLE es:  0.05449032798380175 cuando alpha es:  0.03
Mi MSLE es:  0.05449031877193933 cuando alpha es:  0.04
Mi MSLE es:  0.054490309560925536 cuando alpha es:  0.05
Mi MSLE es:  0.0544903003507604 cuando alpha es:  0.06
Mi MSLE es:  0.054490291141443874 cuando alpha es:  0.07
Mi MSLE es:  0.054490281932975955 cuando alpha es:  0.08
Mi MSLE es:  0.054490272725356656 cuando alpha es:  0.09
```

Cross Validation

Para evaluar la **robustez** del modelo vamos a utilizar **Repeated KFoldes** para aumentar el número de iteraciones en la divisiíón de **KFolds**.

Puedes encontrar mayor información en este enlace

In [ ]:
```
"CELDA N°23"
#Aplicamos las librerías correspondientes para ejecutar el Repeated KFoldes y Cross Vali

from numpy import mean
from numpy import std
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score

#Realizamos 5 particiones y 3 repeticiones
cv = RepeatedKFold(n_splits=5, n_repeats=3)

#Guardamos los resultados de aplicar el MSLE en la variable scores
scores = -cross_val_score(Ridge(alpha=0.01), abs(X), abs(y), scoring='neg_mean_squared_l

#Finalmente obtenemos el promedio y desviación estándar de los resultados
print('Promedio y Desviación del Error: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Promedio y Desviación del Error: 0.083 (0.002)
```

MLo **ideal** es que el error logarítmico promedio sea **mínimo**, por ello el Promedio debe ser **cercano a cero** (al igual que la desviación).

De no cumplise alguna de estas condiciones es una alerta para mejorar el modelo para que sea más **robusto** y mejore su precisión para diferentes datos.

- Error cuadrático medio (MSE): una de las funciones de pérdida más utilizadas, MSE toma la media de las diferencias al cuadrado entre los valores previstos y reales para calcular el valor de pérdida para su modelo de predicción. Funciona mejor cuando se realiza un análisis de referencia y se tiene un conjunto de datos de un orden de magnitud similar.
- Error logarítmico cuadrático medio (MSLE): MSLE adopta un enfoque similar al MSE, pero utiliza un logaritmo para compensar los grandes valores atípicos en un conjunto de datos y los trata como si estuvieran en la misma escala. Esto es más valioso si busca un modelo equilibrado con errores porcentuales similares.

In [ ]:
```

```