Anem a practicar i a familiaritzar-nos amb regressions.

Objectius:

- Models de regressió
- Àrbres de regressió
- Random Forest
- Xarxes Neuronals
- Altres models

Durada: 3 dies

Lliurament: Enviar la URL a un repositori anomenat Supervitat_Regressio que contingui la solució. S'ha d'entregar cada Excercici en un mateix fitxer i en un repositori.

In [51]:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

In [52]:

```
pd.set_option('max_columns', None)
raw_df = pd.read_csv('DelayedFlights.csv', sep=",", encoding='utf8')
```

In [53]:

```
raw_df.head()
```

Out[53]:

	Unnamed: 0	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNu
0	0	2008	1	3	4	2003.0	1955	2211.0	2225	WN	335	N712
1	1	2008	1	3	4	754.0	735	1002.0	1000	WN	3231	N772
2	2	2008	1	3	4	628.0	620	804.0	750	WN	448	N428
3	4	2008	1	3	4	1829.0	1755	1959.0	1925	WN	3920	N464
4	5	2008	1	3	4	1940.0	1915	2121.0	2110	WN	378	N726

In [54]:

```
raw_df.describe()
```

Out[54]:

	Unnamed: 0	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrT
count	1.936758e+06	1936758.0	1.936758e+06	1.936758e+06	1.936758e+06	1.936758e+06	1.936758e+06	1.929648e+06	1.936758e
mean	3.341651e+06	2008.0	6.111106e+00	1.575347e+01	3.984827e+00	1.518534e+03	1.467473e+03	1.610141e+03	1.634225e
std	2.066065e+06	0.0	3.482546e+00	8.776272e+00	1.995966e+00	4.504853e+02	4.247668e+02	5.481781e+02	4.646347e
min	0.000000e+00	2008.0	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	1.000000e+00	0.000000e
25%	1.517452e+06	2008.0	3.000000e+00	8.000000e+00	2.000000e+00	1.203000e+03	1.135000e+03	1.316000e+03	1.325000e
50%	3.242558e+06	2008.0	6.000000e+00	1.600000e+01	4.000000e+00	1.545000e+03	1.510000e+03	1.715000e+03	1.705000e
75%	4.972467e+06	2008.0	9.000000e+00	2.300000e+01	6.000000e+00	1.900000e+03	1.815000e+03	2.030000e+03	2.014000e
max	7.009727e+06	2008.0	1.200000e+01	3.100000e+01	7.000000e+00	2.400000e+03	2.359000e+03	2.400000e+03	2.400000e

In [55]:

```
raw_df.shape
```

Out[55]:

```
(1936758, 30)
```

In [56]:

```
raw_df.dtypes
```

Out[56]:

```
Unnamed: 0              int64
Year                    int64
Month                   int64
DayofMonth              int64
DayOfWeek               int64
DepTime               float64
CRSDepTime              int64
ArrTime               float64
CRSArrTime              int64
UniqueCarrier          object
FlightNum               int64
TailNum                object
ActualElapsedTime     float64
CRSElapsedTime        float64
AirTime               float64
ArrDelay              float64
DepDelay              float64
Origin                 object
Dest                   object
Distance                int64
TaxiIn                float64
TaxiOut               float64
Cancelled               int64
CancellationCode       object
Diverted                int64
CarrierDelay          float64
WeatherDelay          float64
NASDelay              float64
SecurityDelay         float64
LateAircraftDelay     float64
dtype: object
```

Nivell 1

- **Exercici 1:**

Crea almenys tres models de regressió diferents per intentar predir el millor possible l'endarreriment dels vols (ArrDelay) de DelayedFlights.csv.

Modelo de regresión lineal. Para poder empezar a trabajar con él, se tratan los datos NaN para la columna ArrDelay reemplazando por el valor de la media de ArrDelay. Se extraen los datos que interesa predecir, el retraso de salida y el retraso de llegada de los aviones. Se creará el modelo y su ajuste posterior para obtener los datos de predicción.

In [57]:

```
raw_df['ArrDelay']=raw_df['ArrDelay'].fillna(raw_df['ArrDelay'].mean())
```

In [58]:

```
x = np.array(raw_df['DepDelay']).reshape((-1, 1))
y = np.array(raw_df['ArrDelay'])
```

In [59]:

```
print (x,y)
```

```
[[ 8.]
 [19.]
 [ 8.]
 ...
 [80.]
 [11.]
 [ 7.]] [-14.   2.  14. ...  99.   9.  -5.]
```

In [60]:

```
model = LinearRegression().fit(x, y)
r_sq = model.score(x, y)
```

In [61]:

```
print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
```

```
coefficient of determination: 0.8995207508716909
intercept: -1.2579166978369898
slope: [1.00631293]
```

In [62]:

```
y_pred = model.predict(x)
print('predicted response:', y_pred, sep='\n')
```

```
predicted response:
[ 6.79258674 17.86202898  6.79258674 ... 79.24711772  9.81152553
  5.78627381]
```

Se prepara el modelo de regresión polynomial. Este modelo requiere una tranformación del input para incluir el valor de x^2 en el array X. Una vez realizado, se ajusta al modelo y se obtienen las predicciones.

In [63]:

```
x_2 = PolynomialFeatures(degree=2, include_bias=False).fit_transform(x)
```

In [64]:

```
print (x_2)
```

```
[[   8.   64.]
 [  19.  361.]
 [   8.   64.]
 ...
 [  80. 6400.]
 [  11.  121.]
 [   7.   49.]]
```

In [65]:

```
model_2 = LinearRegression().fit(x_2, y)
```

In [66]:

```
r_sq_2 = model_2.score(x_2, y)
print('coefficient of determination:', r_sq_2)
print('intercept:', model_2.intercept_)
print('coefficients:', model_2.coef_)
```

```
coefficient of determination: 0.8995862528489823
intercept: -1.5303942104420258
coefficients: [ 1.01579534e+00 -2.90498437e-05]
```

In [67]:

```
y_pred_2 = model_2.predict(x_2)
print (y_pred_2)
```

```
[ 6.59410929 17.75923019  6.59410929 ... 79.54731371  9.63983946
  5.5787497 ]
```

Advanced Linear Regression con statsmodels. Una forma de obtener una regresión lineal que aporta más detalles que la regresión lineal

In [68]:

```
x_3 = sm.add_constant(x_2)
```

In [69]:

```
print (x_3)
```

```
[[1.00e+00 8.00e+00 6.40e+01]
 [1.00e+00 1.90e+01 3.61e+02]
 [1.00e+00 8.00e+00 6.40e+01]
 ...
 [1.00e+00 8.00e+01 6.40e+03]
 [1.00e+00 1.10e+01 1.21e+02]
 [1.00e+00 7.00e+00 4.90e+01]]
```

In [70]:

```
model_3 = sm.OLS(y, x_3)
```

In [71]:

```
results = model_3.fit()
```

In [72]:

```
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.900
Model:                            OLS   Adj. R-squared:                  0.900
Method:                 Least Squares   F-statistic:                 8.675e+06
Date:                Fri, 06 May 2022   Prob (F-statistic):               0.00
Time:                        15:21:18   Log-Likelihood:            -8.3412e+06
No. Observations:             1936758   AIC:                         1.668e+07
Df Residuals:                 1936755   BIC:                         1.668e+07
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.5304      0.018    -83.730      0.000      -1.566      -1.495
x1             1.0158      0.000   2822.323      0.000       1.015       1.017
x2         -2.905e-05   8.17e-07    -35.544      0.000   -3.07e-05   -2.74e-05
==============================================================================
Omnibus:                   912921.943   Durbin-Watson:                   1.724
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        281254026.155
Skew:                           1.045   Prob(JB):                         0.00
Kurtosis:                      61.999   Cond. No.                     3.40e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.4e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [73]:

```
print('coefficient of determination:', results.rsquared)
print('adjusted coefficient of determination:', results.rsquared_adj)
print('regression coefficients:', results.params)
```

```
coefficient of determination: 0.8995862528489823
adjusted coefficient of determination: 0.8995861491562105
regression coefficients: [-1.53039421e+00  1.01579534e+00 -2.90498437e-05]
```

In [74]:

```
print('predicted response:', results.predict(x_3), sep='\n')
```

```
predicted response:
[ 6.59410929 17.75923019  6.59410929 ... 79.54731371  9.63983946
  5.5787497 ]
```

Se realiza un cuarto modelo de regresión, esa vez random forest. Para ello, hay que transformar primero los datos a tipo Integer (random forest solo acepta integer como valor). Después se realiza un train test split y se usará para realizar la predicción de los retrasos.

In [75]:

```
X = x.round(0).astype(int)
Y = y.round(0).astype(int)
```

In [76]:

```
X_train, X_test, y_train, y_test = train_test_split(x, y , test_size=0.33, random_state=42)
```

In [77]:

```
model_4 = RandomForestRegressor(random_state = 42)
```

In [78]:

```
model_4.fit(X_train, y_train)
```

Out[78]:

```
RandomForestRegressor(random_state=42)
```

In [79]:

```
y_pred_4 = model_4.predict(X_test)
```

In [80]:

```
print('predicted response:', y_pred_4, sep='\n')
```

```
predicted response:
[  8.77399531 434.97427653  19.56738872 ...   6.12044374   4.03793344
  29.93782911]
```

In [81]:

```
r_sq_4 = model_4.score(X, Y)
print (r_sq_4)
```

```
0.9001126119515857
```

- **Exercici 2:**

Compara'ls en base al MSE i al R2.

In [82]:

```
print('Mean squared error for all models:')
print('Linear Regression model:', mean_squared_error(y, y_pred))
print('Polynomial Regression:', mean_squared_error(y, y_pred_2))
print('Advanced Linear Regression:', mean_squared_error(y, y_pred))
print('Random Forest:', mean_squared_error(y_test, y_pred_4))
```

```
Mean squared error for all models:
Linear Regression model: 322.5925190396327
Polynomial Regression: 322.3822224058516
Advanced Linear Regression: 322.5925190396327
Random Forest: 323.67522778814583
```

In [83]:

```
print('Coefficient of determination for all models:' )
print('Linear Regression model:', r_sq)
print('Polynomial Regression:', r_sq_2)
print('Advanced Linear Regression: 0.900')
print('Random Forest:', r_sq_4)
```

```
Coefficient of determination for all models:
Linear Regression model: 0.8995207508716909
Polynomial Regression: 0.8995862528489823
Advanced Linear Regression: 0.900
Random Forest: 0.9001126119515857
```

- **Exercici 3:**

Entrena'ls utilitzant els diferents paràmetres que admeten.

Linear Model con n_jobs para controlar el tamaño y el tiempo de ejecución, ya que es un dataset bastsante amplio. El siguiente es controlar con fit_intercept, que se encargará de encontar el mejor fiting.

In [84]:

```
linear_model = LinearRegression(n_jobs=-1, normalize=True).fit(x, y)
sq1 = linear_model.score(x, y)
print('coefficient of determination:', sq1)
```

```
coefficient of determination: 0.8995207508716909
/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_base.py:141: FutureWarning: 'normalize' was de
precated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the pr
evious behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipel
ine as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
```

In [85]:

```
linear_model_2 = LinearRegression(fit_intercept=False).fit(x, y)
sq2 = linear_model_2.score(x, y)
print('coefficient of determination:', sq2)
```

```
coefficient of determination: 0.8992227601508445
```

In [86]:

```
linear_model_3 = LinearRegression(copy_X=False).fit(x, y)
sq3 = linear_model_3.score(x, y)
print('coefficient of determination:', sq3)
```

```
coefficient of determination: 0.3112766341229223
```

Entreno de modelos con randomforest. Se controla con max_features el tamaño del split que sea igual a sqrt. También se crea u segundo modelo, que por cuestiones de tamao construye 50 estimaciones.

In [87]:

```
random_forest_model = RandomForestRegressor(max_features = 'sqrt', random_state = 42).fit(X_train,y_train)
rf_sq = random_forest_model.score(X, y)
print('coefficient of determination:', rf_sq)
```

```
coefficient of determination: 0.9001240244300632
```

In [88]:

```
random_forest_model_2 = RandomForestRegressor(n_estimators = 50, random_state = 42, bootstrap = True, verbose=2
rf_sq2 = random_forest_model_2.score(X, y)
print('coefficient of determination:', rf_sq2)
```

```
building tree 1 of 50
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.5s remaining:    0.0s
building tree 2 of 50
building tree 3 of 50
building tree 4 of 50
building tree 5 of 50
building tree 6 of 50
building tree 7 of 50
building tree 8 of 50
building tree 9 of 50
building tree 10 of 50
building tree 11 of 50
building tree 12 of 50
building tree 13 of 50
building tree 14 of 50
building tree 15 of 50
building tree 16 of 50
building tree 17 of 50
building tree 18 of 50
building tree 19 of 50
building tree 20 of 50
building tree 21 of 50
building tree 22 of 50
building tree 23 of 50
building tree 24 of 50
building tree 25 of 50
building tree 26 of 50
building tree 27 of 50
building tree 28 of 50
building tree 29 of 50
building tree 30 of 50
building tree 31 of 50
building tree 32 of 50
building tree 33 of 50
building tree 34 of 50
building tree 35 of 50
building tree 36 of 50
building tree 37 of 50
building tree 38 of 50
building tree 39 of 50
building tree 40 of 50
building tree 41 of 50
building tree 42 of 50
building tree 43 of 50
building tree 44 of 50
building tree 45 of 50
building tree 46 of 50
building tree 47 of 50
building tree 48 of 50
building tree 49 of 50
building tree 50 of 50
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:   27.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
coefficient of determination: 0.9001170518469798
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    4.6s finished
```

In [89]:

```
random_forest_model_3 = RandomForestRegressor(n_estimators = 200, max_depth=2, random_state=0).fit(X_train,y_tr
rf_sq3 = random_forest_model_3.score(X, y)
print('coefficient of determination:', rf_sq3)
```

```
coefficient of determination: 0.7662742824995042
```

Finalmente, el modelo Polynomial

In [91]:

```
model_2 = LinearRegression(fit_intercept=False).fit(X, y)
r_sq2 = model_2.score(X, y)
print('coefficient of determination:', r_sq2)
```

```
coefficient of determination: 0.8992227601508445
```

In [92]:

```
model_3 = LinearRegression(copy_X=False).fit(X, y)
r_sq3 = model_3.score(X, y)
```

- **Exercici 4:**

Compara el seu rendiment utilitzant l'aproximació traint/test o utilitzant totes les dades (validació interna).

Regresión lineal simple. Resultados impresos en un Dataset.

In [93]:

```
X_train, X_test, y_train, y_test = train_test_split(x, y , test_size=0.33, random_state=42)
```

In [94]:

```
model = LinearRegression().fit(X_train, y_train)
model.score(X_train, y_train)
pred = model.predict(X_test)
```

In [95]:

```
df = pd.DataFrame({'Original data': y_test, 'Predicted data': pred})
df
```

Out[95]:

	Original data	Predicted data
0	71.0	8.798748
1	548.0	562.393604
2	9.0	19.870645
3	52.0	53.086336
4	22.0	5.779139
...	...	...
639126	51.0	80.262811
639127	158.0	166.824916
639128	6.0	6.785675
639129	-3.0	4.772603
639130	42.0	29.936006

639131 rows × 2 columns

Random Forest con resultados impresos en un dataset

In [96]:

```
model2 = RandomForestRegressor(random_state = 42).fit(X_train,y_train)
model2.score(X_train, y_train)
pred2=model.predict(X_test)
```

In [97]:

```
df2 = pd.DataFrame({'Original data': y_test, 'Predicted data': pred2})
df2
```

Out[97]:

	Original data	Predicted data
0	71.0	8.798748
1	548.0	562.393604
2	9.0	19.870645
3	52.0	53.086336
4	22.0	5.779139
...	...	...
639126	51.0	80.262811
639127	158.0	166.824916
639128	6.0	6.785675
639129	-3.0	4.772603
639130	42.0	29.936006

639131 rows × 2 columns

Polynomial con resultados en un nuevo dataset.

In [99]:

```
X = PolynomialFeatures(degree=4, include_bias=False).fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
model = LinearRegression().fit(X_train, y_train)
pred3 = model.predict(X_test)
```

In [100…

```
df3 = pd.DataFrame({'Original data': y_test, 'Predicted data': pred3})
df3
```

Out[100…

	Original data	Predicted data
0	9.0	20.758196
1	-3.0	5.251061
2	13.0	20.758196
3	11.0	14.564190
4	19.0	17.662649
...	...	...
387347	147.0	161.458917
387348	9.0	17.662649
387349	36.0	33.111803
387350	41.0	42.347762
387351	72.0	60.747623

387352 rows × 2 columns

Nivell 2

- **Exercici 5:**

Realitza algun procés d'enginyeria de variables per millorar-ne la predicció.

Nivell 3

- **Exercici 6:**

No utilitzis la variable DepDelay a l'hora de fer prediccions.