Descripció Anem a practicar i a familiaritzar-nos amb algoritmes de classificació.

Nivell 1

In [109…
```
import pandas as pd
import xgboost as xgb
import numpy as np

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

In [110…
```
# Importar el Data set
raw_df = pd.read_csv('DelayedFlights.csv')
```

In [111…
```
raw_df
```

Out[111…

	Unnamed: 0	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	...	TaxiIn
0	0	2008	1	3	4	2003.0	1955	2211.0	2225	WN	...	4.0
1	1	2008	1	3	4	754.0	735	1002.0	1000	WN	...	5.0
2	2	2008	1	3	4	628.0	620	804.0	750	WN	...	3.0
3	4	2008	1	3	4	1829.0	1755	1959.0	1925	WN	...	3.0
4	5	2008	1	3	4	1940.0	1915	2121.0	2110	WN	...	4.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1936753	7009710	2008	12	13	6	1250.0	1220	1617.0	1552	DL	...	9.0
1936754	7009717	2008	12	13	6	657.0	600	904.0	749	DL	...	15.0
1936755	7009718	2008	12	13	6	1007.0	847	1149.0	1010	DL	...	8.0
1936756	7009726	2008	12	13	6	1251.0	1240	1446.0	1437	DL	...	13.0
1936757	7009727	2008	12	13	6	1110.0	1103	1413.0	1418	DL	...	8.0

1936758 rows × 30 columns

In [112…
```
# para eliminar entradas Nan y se reemplazan por la media de los valores de todo el set
raw_df['ArrTime']= raw_df['ArrTime'].fillna(raw_df['ArrTime'].mean())
```

In [113…
```
# se seleccionan las columnas que se quieren utilizar. Se seleccionan las columnas de salida programada,
# salida real, llegada programada y llegada real.
df = raw_df[['UniqueCarrier', 'DepTime', 'CRSDepTime' , 'ArrTime', 'CRSArrTime']]
```

In [114…
```
df.head(5)
```

Out[114…

	UniqueCarrier	DepTime	CRSDepTime	ArrTime	CRSArrTime
0	WN	2003.0	1955	2211.0	2225
1	WN	754.0	735	1002.0	1000
2	WN	628.0	620	804.0	750
3	WN	1829.0	1755	1959.0	1925
4	WN	1940.0	1915	2121.0	2110

In [115…
```
df['UniqueCarrier'].value_counts()
```

Out[115…
```
WN    377602
AA    191865
MQ    141920
UA    141426
OO    132433
DL    114238
XE    103663
CO    100195
US     98425
EV     81877
NW     79108
FL     71284
YV     67063
B6     55315
OH     52657
9E     51885
AS     39293
F9     28269
HA      7490
AQ       750
Name: UniqueCarrier, dtype: int64
```

In [116…
```
# finalmente, por cuestiones técnicas decido trabajar con los datos de una compañía y predecir
# la puntualidad de una compañía especifica en lugar de todo el set.
carriers = df[(df.UniqueCarrier == 'WN')]['UniqueCarrier']
wn = df[df['UniqueCarrier'].isin(carriers)]
```

In [117…
```
wn.head(5)
```

Out[117…

	UniqueCarrier	DepTime	CRSDepTime	ArrTime	CRSArrTime
0	WN	2003.0	1955	2211.0	2225
1	WN	754.0	735	1002.0	1000
2	WN	628.0	620	804.0	750
3	WN	1829.0	1755	1959.0	1925
4	WN	1940.0	1915	2121.0	2110

In [118…
```
wn.dtypes
```

Out[118…
```
UniqueCarrier     object
DepTime          float64
CRSDepTime         int64
ArrTime          float64
CRSArrTime         int64
dtype: object
```

In [119…
```
wn.describe()
```

Out[119…

	DepTime	CRSDepTime	ArrTime	CRSArrTime
count	377602.000000	377602.000000	377602.000000	377602.000000
mean	1564.304196	1516.528898	1648.512906	1668.651927
std	424.057374	402.096921	524.499495	455.023485
min	1.000000	600.000000	1.000000	5.000000
25%	1243.000000	1220.000000	1350.000000	1355.000000
50%	1616.000000	1545.000000	1734.000000	1730.000000
75%	1920.000000	1845.000000	2041.000000	2028.750000
max	2400.000000	2235.000000	2400.000000	2355.000000

- Exercici 1:

Crea almenys tres models de classificació diferents per intentar predir el millor possible l'endarreriment dels vols (ArrDelay) de DelayedFlights.csv. Considera si el vol ha arribat tard o no (ArrDelay > 0).

Decision Tree Model

In [120…
```
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [121…
```
# creación del objeto
clf = DecisionTreeClassifier()
# Train del Decision Tree Classifer
clf = clf.fit(X ,y)
# prediccíones de test dataset
y_pred = clf.predict(X)
```

In [122…
```
y_pred
```

Out[122…
```
array([2205, 1000,  750, ..., 2215, 2005, 1920])
```

In [123…
```
# Accuracy de modelo Decision Tree
accuracy_decision_tree = metrics.accuracy_score(y, y_pred)
```

In [124…
```
# Matriz de confianza
matrix_decision_tree = metrics.confusion_matrix(y, y_pred)
```

In [125…
```
# Separación de la matriz de confianza
confusion_decision_tree = metrics.confusion_matrix(y, y_pred)
print(confusion_decision_tree)
#[fila, columna]
TP_DT = confusion_decision_tree[1, 1]
TN_DT = confusion_decision_tree[0, 0]
FP_DT = confusion_decision_tree[0, 1]
FN_DT = confusion_decision_tree[1, 0]
```

```
[[1041    1    0 ...    1    0    0]
 [   6  511    0 ...    0    0    0]
 [   2    1  474 ...    0    0    0]
 ...
 [   3    0    2 ...  853    1    0]
 [   4    0    3 ...    5  993    0]
 [   7    0    1 ...    6   14 1437]]
```

In [126…
```
#classification accuracy
classification_accuracy_decision_tree = (TP_DT + TN_DT) / float(TP_DT + TN_DT + FP_DT + FN_DT)
print(classification_accuracy_decision_tree)
print(metrics.accuracy_score(y , y_pred))
```

```
0.9955099422706863
0.9028156630526322
```

In [127…
```
#classification error
classification_error_decision_tree = (FP_DT + FN_DT) / float(TP_DT + TN_DT + FP_DT + FN_DT)
print(classification_error_decision_tree)
print(1 - metrics.accuracy_score(y , y_pred))
```

```
0.004490057729313663
0.09718433694736783
```

In [128…
```
# sensibilidad
sensitivity_decision_tree = (TP_DT) / float(FN_DT + TP_DT)
print(sensitivity_decision_tree)
print(metrics.recall_score(y , y_pred, average='micro'))
```

```
0.988394584139265
0.9028156630526322
```

In [129…
```
#especifidad
specificity_decision_tree = (TN_DT) / (TN_DT + FP_DT)
print(specificity_decision_tree)
```

```
0.9990403071017274
```

In [130…
```
# ratio de falsos positivos
false_positive_rate_decision_tree = (FP_DT) / float(TN_DT + FP_DT)
print(false_positive_rate_decision_tree)
print(1 - specificity_decision_tree)
```

```
0.0009596928982725527
0.0009596928982725794
```

In [131…
```
# precisión
precision_decision_tree = (TP_DT) / float(TP_DT + FP_DT)
print(precision_decision_tree)
print(metrics.precision_score(y , y_pred, average='micro'))
```

```
0.998046875
0.9028156630526322
```

KNN

In [132…
```
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [133…
```
# Objeto KNN
classifier = KNeighborsClassifier()
# Fit
classifier.fit(X, y)
# predicciones
y_pred = classifier.predict(X)
```

In [134…
```
y_pred
```

Out[134…
```
array([2205, 1000,  800, ..., 2215, 1950, 1920])
```

In [135…
```
# Accuracy de modelo KNN
accuracy_knn = metrics.accuracy_score(y , y_pred)
print(accuracy_knn)
```

```
0.5089909481411645
```

In [136…
```
# Separación de la matriz de confianza
confusion_knn = metrics.confusion_matrix(y, y_pred)
print(confusion_knn)
#[fila, columna]
TP_KNN = confusion_knn[1, 1]
TN_KNN = confusion_knn[0, 0]
FP_KNN = confusion_knn[0, 1]
FN_KNN = confusion_knn[1, 0]
```

```
[[795  32  19 ...  21  22  87]
 [ 75 324  12 ...   4   3  21]
 [ 61  20 275 ...   4   8  13]
 ...
 [ 61  11  16 ... 408  66  60]
 [ 94  11  18 ...  57 555  87]
 [152  37  30 ...  71  71 927]]
```

In [137…
```
#classification accuracy
classification_accuracy_knn =(TP_KNN + TN_KNN) / float(TP_KNN + TN_KNN + FP_KNN + FN_KNN)
print(metrics.accuracy_score(y , y_pred))
print(metrics.accuracy_score(y , y_pred))
```

```
0.5089909481411645
0.5089909481411645
```

In [138…
```
# Error de clasificación
classification_error_knn = (FP_KNN + FN_KNN) / float(TP_KNN + TN_KNN + FP_KNN + FN_KNN)
print(classification_error_knn)
print(1 - metrics.accuracy_score(y , y_pred))
```

```
0.08727569331158239
0.4910090518588355
```

In [139…
```
#Sensibilidad
sensitivity_knn = (TP_KNN) / float(FN_KNN + TP_KNN)
print(sensitivity_knn)
print(metrics.recall_score(y , y_pred, average='micro'))
```

```
0.8120300751879699
0.5089909481411645
```

In [140…
```
# Especificidad
specificity_knn = TN_KNN / (TN_KNN + FP_KNN)
print(specificity_knn)
```

```
0.9613059250302297
```

In [141…
```
# Ratio de falsos positvos
false_positive_rate_knn = (FP_KNN) / float(TN_KNN + FP_KNN)
print(false_positive_rate_knn)
print(1 - specificity_knn)
```

```
0.03869407496977025
0.03869407496977029
```

In [142…
```
# precisión
precision_knn = (TP_KNN) / float(TP_KNN + FP_KNN)
print(precision_knn)
print(metrics.precision_score(y , y_pred, average='micro'))
```

```
0.9101123595505618
0.5089909481411645
```

Support Vector Machine

In [143…
```
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [144…
```
# por rendimiento, limito las muestras a 20000, pese a que puede afectar los resultados.
X = X[ :1000]
y = y[ :1000]
```

In [145…
```
# Clasificación mediante SVM
clf = svm.SVC()
clf.fit(X , y)
# Predicciones
y_pred = clf.predict(X)
```

In [146…
```
# Accuracy de modelo SVM
accuracy_svm = metrics.accuracy_score(y , y_pred)
```

In [147…
```
# Separación de la matriz de confianza
confusion_svm = metrics.confusion_matrix(y , y_pred)
print(confusion_svm)
#[fila, columna]
TP_SVM = confusion_svm[1, 1]
TN_SVM = confusion_svm[0, 0]
FP_SVM = confusion_svm[0, 1]
FN_SVM = confusion_svm[1, 0]
```

```
[[0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 1 0]]
```

In [148…
```
#classification accuracy
classification_accuracy_svm = (TP_SVM + TN_SVM) / float(TP_SVM + TN_SVM + FP_SVM + FN_SVM)
print(classification_accuracy_svm)
print(metrics.accuracy_score(y , y_pred))
```

```
nan
0.065
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/2487099221.py:2: RuntimeWarning: invalid value e
ncountered in true_divide
  classification_accuracy_svm = (TP_SVM + TN_SVM) / float(TP_SVM + TN_SVM + FP_SVM + FN_SVM)
```

In [149…
```
# classification error
classification_error_svm = (FP_SVM + FN_SVM) / float(TP_SVM + TN_SVM + FP_SVM + FN_SVM)
print(classification_error_svm)
print(1 - metrics.accuracy_score(y , y_pred))
```

```
nan
0.935
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/2179497558.py:2: RuntimeWarning: invalid value e
ncountered in true_divide
  classification_error_svm = (FP_SVM + FN_SVM) / float(TP_SVM + TN_SVM + FP_SVM + FN_SVM)
```

In [150…
```
# sensibilidad
sensitivity_svm = (TP_SVM) / float(FN_SVM + TP_SVM)
print(sensitivity_svm)
print(metrics.recall_score(y , y_pred, average='micro'))
```

```
nan
0.065
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/2944104583.py:2: RuntimeWarning: invalid value e
ncountered in true_divide
  sensitivity_svm = (TP_SVM) / float(FN_SVM + TP_SVM)
```

In [151…
```
# Especifidad
specificity_svm = (TN_SVM) / (TN_SVM + FP_SVM)
print(specificity_svm)
```

```
nan
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/1190349425.py:2: RuntimeWarning: invalid value e
ncountered in long_scalars
  specificity_svm = (TN_SVM) / (TN_SVM + FP_SVM)
```

In [152…
```
# Ratio de falsos positivos
false_positive_rate_svm = (FP_SVM) / float(TN_SVM + FP_SVM)
print(false_positive_rate_svm)
print(1 - specificity_svm)
```

```
nan
nan
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/2499976114.py:2: RuntimeWarning: invalid value e
ncountered in true_divide
  false_positive_rate_svm = (FP_SVM) / float(TN_SVM + FP_SVM)
```

In [153…
```
#precisión
precision_svm = (TP_SVM) / float(TP_SVM + FP_SVM)
print(precision_svm)
print(metrics.precision_score(y , y_pred, average='micro'))
```

```
nan
0.065
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/86456474.py:2: RuntimeWarning: invalid value enc
ountered in true_divide
  precision_svm = (TP_SVM) / float(TP_SVM + FP_SVM)
```

- Exercici 2

Compara els models de classificació utilitzant la precisió (accuracy), una matriu de confiança i d'altres mètriques més avançades.

In [154…
```
# comparativa de Accuracy de los tres modelos de clasificación
print("Decision Tree accuracy: %f" %accuracy_decision_tree)
print("KNN accuracy: %f" %accuracy_knn)
print("SVM accuracy: %f" %accuracy_svm)
```

```
Decision Tree accuracy: 0.902816
KNN accuracy: 0.508991
SVM accuracy: 0.065000
```

In [155…
```
print("Matriz del arbol de decisión: ")
print(matrix_decision_tree)
print("\nMatriz del modelo KNN: ")
print(confusion_knn)
print("\nMatriz del modelo SVM: ")
print(confusion_svm)
```

```
Matriz del arbol de decisión:
[[1041    1    0 ...    1    0    0]
 [   6  511    0 ...    0    0    0]
 [   2    1  474 ...    0    0    0]
 ...
 [   3    0    2 ...  853    1    0]
 [   4    0    3 ...    5  993    0]
 [   7    0    1 ...    6   14 1437]]

Matriz del modelo KNN:
[[795  32  19 ...  21  22  87]
 [ 75 324  12 ...   4   3  21]
 [ 61  20 275 ...   4   8  13]
 ...
 [ 61  11  16 ... 408  66  60]
 [ 94  11  18 ...  57 555  87]
 [152  37  30 ...  71  71 927]]

Matriz del modelo SVM:
[[0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 1 0]]
```

In [156…
```
# comparativa resultados Classification Accuracy de los tres modelos
print("Classificaction Accuracy Decision Tree: ", classification_accuracy_decision_tree)
print("Classificaction KNN: ", classification_accuracy_knn)
print("Classificaction SVM: ", classification_accuracy_svm)
```

```
Classificaction Accuracy Decision Tree:  0.9955099422706863
Classificaction KNN:  0.9127243066884176
Classificaction SVM:  nan
```

In [157…
```
# comparativa Classification error accuracy de los tres modelos
print("Classification error accuracy Decision Tree: " , classification_error_decision_tree)
print("Classification error accuracy KNN: " , classification_error_knn)
print("Classification error accuracy SVM: " ,  classification_error_svm)
```

```
Classification error accuracy Decision Tree:  0.004490057729313663
Classification error accuracy KNN:  0.08727569331158239
Classification error accuracy SVM:  nan
```

In [158…
```
# comparativa sensibilidad de los tres modelos
print("Sensitivity decision tree: " , sensitivity_decision_tree)
print("Sensitivity KNN: " ,sensitivity_knn)
print("Sensitivity SVM: " ,sensitivity_svm)
```

```
Sensitivity decision tree:  0.988394584139265
Sensitivity KNN:  0.8120300751879699
Sensitivity SVM:  nan
```

In [159…
```
# comparativa especifidad de los tres modelos
print("Specifitcity Decision Tree: " , specificity_decision_tree)
print("Specifitcity KNN: " , specificity_knn)
print("Specifitcity SVM: " , specificity_svm)
```

```
Specifitcity Decision Tree:  0.9990403071017274
Specifitcity KNN:  0.9613059250302297
Specifitcity SVM:  nan
```

In [160…
```
# comparativa falsos positivos de los tres modelos
print("False positive Decision Tree: " , false_positive_rate_decision_tree)
print("False positive KNN: " , false_positive_rate_knn)
print("False positive SVM: " , false_positive_rate_svm)
```

```
False positive Decision Tree:  0.0009596928982725527
False positive KNN:  0.03869407496977025
False positive SVM:  nan
```

In [161…
```
# comparativa precisión de los tres modelos
print("Precision Decision tree: " , precision_decision_tree)
print("Precision Decision KNN: " , precision_knn)
print("Precision Decision SVM: " , precision_svm)
```

```
Precision Decision tree:  0.998046875
Precision Decision KNN:  0.9101123595505618
Precision Decision SVM:  nan
```

- Exercici 3

Entrena'ls utilitzant els diferents paràmetres que admeten.

Decison Tree

In [162…
```
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [163…
```
# creación del objeto modificado criterio y la profundidad
clf = DecisionTreeClassifier(criterion='gini', max_depth=5)
# Train del Decision Tree Classifer
clf = clf.fit(X ,y)
# prediccíones de test dataset
y_pred = clf.predict(X)
```

In [164…
```
y_pred
```

Out[164…
```
array([2135,   55,   55, ..., 2210, 2035, 1910])
```

In [165…
```
# Accuracy de modelo Decision Tree
trained_accuracy_decision_tree = metrics.accuracy_score(y, y_pred)
print(trained_accuracy_decision_tree)
```

```
0.09276169088087458
```

KNN

In [166…
```
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [167…
```
# KNN con parámetro de n_neighbors personalizado
classifier = KNeighborsClassifier(n_neighbors=9)
classifier.fit(X , y)
y_pred = classifier.predict(X)
```

In [168…
```
y_pred
```

Out[168…
```
array([2205,  945,  750, ..., 2215, 1950, 1920])
```

In [169…
```
# Accuracy de modelo KNN
trained_accuracy_knn = metrics.accuracy_score(y , y_pred)
print(accuracy_knn)
```

```
0.5089909481411645
```

SVM

In [170…
```
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [171…
```
# Limitación de muestras por motivos de rendiento y limitación de equipo
X = X[ :1000]
y = y[ :1000]
```

In [172…
```
# Clasificación mediante SVM modificando a Kernel Lineal
clf = svm.SVC(kernel='linear')
clf.fit(X , y)
# Predicciones
y_pred = clf.predict(X)
```

In [173…
```
y_pred
```

Out[173…
```
array([2150, 1000,  750, 1855, 2100, 1950,  915, 1725, 1010, 1615,  955,
       1455, 1435, 1730, 2045, 1115, 1550, 1505, 2200, 2210, 1540, 1700,
       1815, 1815, 1215, 1455, 1530, 2030,  950,  915, 1440, 1730, 1730,
       1850, 2115, 1235, 2100,  940,  655, 1125, 2225, 1835, 1845,  950,
       1850,  915, 1405, 1020, 1945, 1655,    5, 1510, 1825,   30, 1600,
       1720,   50, 1320,   25, 1810, 2010,   50, 1750,   25, 2255, 1845,
       2255, 2010, 1510, 1145, 1355, 1750, 1325, 1610,   50, 1645,   30,
       1850, 1320,    5, 1610, 2225, 1720, 1905, 2005, 1320, 1305, 1845,
       2225, 1640, 1235,   55, 2245,   55, 1725, 2225, 1825, 2255, 2015,
        745, 1620, 1630, 1325, 2155, 2245, 1510, 1345, 1510, 1720,   50,
       2335, 2000, 1430, 2025, 2030, 1945, 1720, 1320, 2225, 2350, 2335,
         50, 1850,   35, 2350, 1510, 1915, 1410, 1655, 2255, 2105, 1615,
       1055, 2235, 1640, 1755, 2355, 2115, 1125, 2255,  930, 1315, 1855,
       1425,   25, 2000, 1520, 2255, 1835,   45,  915, 1530, 2015, 1950,
       1750, 1320, 2245, 2255, 1645, 2115, 1900, 1120,   50, 1620, 2350,
       1840,   45, 1750, 2000, 1905, 1645,  955, 2315, 2255, 1145, 1400,
       1455, 1440, 2255, 1745, 1035, 2015, 1845, 2255, 1530, 1625, 2255,
       1215, 1920,    5, 1510,   45, 1905, 2235, 1330, 1530, 1045, 2010,
       2105, 2240, 1350, 1400, 2115, 2255, 1645, 1815, 1320, 2140, 1045,
       2255, 1815, 1555, 2355, 2045, 1520, 1855, 1035, 1250, 1630, 2225,
       1925, 1110, 1600, 2040, 2350,   35, 1505, 2330, 1930, 1540, 2255,
       1330, 1955, 2255, 1615, 1510, 2040, 1835, 1615, 2025, 1235, 2040,
         35, 1520, 1240, 1745, 1045, 1845, 1130, 2000, 2205, 2255, 2115,
       1435, 1835, 1640, 2350, 2225, 1835, 2030, 1505, 2350, 1115, 2040,
       1105, 1530, 1615, 1845, 2245, 2115, 1915, 1950, 1545, 1825, 1815,
       1730, 2135,  915, 1315, 1825, 1930, 2035, 2105, 1345, 2255, 1725,
       1220, 1120, 1550, 1830, 1005, 1515, 1640, 2115, 1930, 1550, 1210,
       1405, 1550, 1645, 2255, 1240, 2255, 1530, 1930, 1340, 1110, 1825,
       1615, 1330, 2255, 1920, 1220,  930, 2040, 1045, 2255, 1615, 1520,
       1830, 2205, 2225, 1615, 2105, 1300, 2255, 1845, 1115, 1440, 1745,
       1915, 1625, 1925, 1605, 1820, 1325, 1205, 2115, 1530, 1440, 1230,
       1915, 1135, 1925, 1630, 1855, 2045, 1015, 1550, 1750, 2000, 1000,
       1315, 2245, 1655, 2245, 1040, 1405, 2040, 1145, 2045, 1450,  720,
       1230, 2100, 1530, 1835,  810, 1920,  820, 1615, 2225, 1640, 1050,
       2210, 1000, 1915, 1130, 2115, 1730, 1225, 2225, 1815, 1410,  830,
       1830, 2255, 1235,  925, 1620, 2015, 1545, 2205, 1930, 1135, 1535,
       1125, 1855, 1455, 2225, 1600, 1455, 1320,  750, 1730, 1330, 1115,
       1810,  725,  955, 2135,  955, 1920, 2005, 1540, 1720, 1230, 1320,
       2135, 1530, 1730, 1335, 1905, 2155, 1950, 1045, 1800,  755, 1550,
       1205, 1855, 2155, 1010, 1440, 2100, 2225, 1750, 2110, 1555, 1705,
       1230, 1400, 1825,  830, 1855, 2015,  805, 1135, 2255, 2025,  915,
       2225,  830, 2015, 1120, 1700, 1945, 1630, 1400, 1505, 1920, 2245,
       1440,  915, 2135, 1110, 1720, 1045, 2240, 1405, 1820, 1700, 1140,
       1630, 1400,  915, 2210, 1655, 1735, 1320, 2335, 1030, 2150, 2200,
       1645, 1140, 2125, 1855,  920, 1330, 1700, 2210, 2240,  730, 1040,
       1510, 2255, 2115, 1600, 1810, 2030, 1105, 1330,    5, 2035, 2250,
       1025, 1340, 1620, 1815, 2235, 2335,  815, 1825, 1530, 1935, 2255,
       1355, 1105,  805, 1550, 1845, 1425, 1000, 2040, 1825, 1200, 1615,
       2325, 1700, 2040, 1320, 2350, 1030,  925, 2255, 1105, 1655, 1410,
       1945, 2250, 1725, 2200,  915, 1845, 2250, 1445,  905, 1205, 1850,
       1600, 2150, 2045,  925, 2250, 1905, 1230, 1750, 2235, 1355,  815,
        940, 1700, 1030, 2115, 1630,  925, 2320, 1730, 1415, 1630, 2030,
       1630,  840, 1205, 1845, 1410, 2255, 1010, 1050, 2100, 1625, 1750,
       1620,  955, 2335, 1720, 1135, 1350, 2025, 2305, 2335,  955, 2110,
       1645, 1330, 1620, 1150, 1435, 2335, 1615, 1110, 1350, 1855,  825,
       2240, 1815, 1615, 2220, 1105, 2040, 1835, 1520, 2145, 2000,  855,
       1230,   25, 1105,  955, 1825, 2335, 2030, 1520,   15, 1300, 2255,
        950, 1730, 1320, 2000, 2335, 1540, 2110, 1825, 1455, 2255,  955,
       1645, 2335, 1305, 1505,  855, 1045, 1720, 2110, 1020, 2140, 2300,
        830, 2000, 1550, 1300, 2335, 1305, 1025, 2350, 1555, 1240, 1020,
       2040, 1135, 1850, 2210, 1055, 1110, 1845, 2255, 2000,  725,  910,
       1750, 1405, 2040, 1220, 1655, 2300,  955, 2010, 1410, 1410, 2145,
       1325,  855, 1145, 2255, 1615, 1845, 1655, 1435,  950, 1500, 1235,
       2010,   30, 1155, 2225, 1855, 1550,  800, 1745, 1530,  830, 1050,
       1340, 1945, 2030, 1125, 1725, 1235, 1140, 1010, 1625, 1315, 1440,
       1505, 2115, 2205, 1245, 1750, 1055, 1825, 1630, 2235, 1835, 1510,
       1505,  855, 1410, 2245, 1730, 1300, 2200, 1730, 2000, 1550, 1240,
       2205, 1035, 1625, 1855, 1325, 2105, 2255,  910, 1630, 2000, 2220,
        955, 1320, 2140, 1110, 2040, 1625, 2255, 1400, 2115, 1505, 1250,
       1655, 1825, 2245, 1855, 1315, 1035, 1645, 2135,  945, 1855, 1530,
       2210, 1950, 1715, 1825, 1215,  915, 1915, 2045, 1320, 1835, 2025,
       1815,   30, 1615, 2115, 2210, 1845, 1400, 2000, 1700, 1845,  755,
       2325, 2115, 1355, 2300, 1920, 1540, 2105, 1410, 1655, 2155,  830,
       1400, 1815, 1405, 1615, 1110, 1325, 1855, 2100, 1455, 1800,  855,
       2155, 1645, 1340, 1950, 1925, 2255, 1600, 2255, 1300, 1920, 2020,
       2225, 1750, 1350, 1550, 2110, 1250, 1745, 2020, 2045, 1405, 1950,
       1405, 2045, 1750, 1435, 1800, 1825, 2040, 1000, 1550, 1225, 1910,
       2105, 1330,  820, 1925, 1610, 1550, 1715, 1045, 2205, 1350, 1830,
       2045, 1355, 1710, 1535, 2255, 2245, 2045, 1835, 1955, 2040, 1645,
       2320,  955, 1615, 1835, 1255, 1100, 1625, 2255, 2325, 1730, 1310,
       1915, 2045, 1455, 2115, 1810, 1510, 1100,  810,  840, 1750, 1445,
       1135, 2020,  750, 1800, 1535, 2220, 1825, 2100, 1655, 1855, 1450,
       1415, 1530, 1825, 1625, 1355, 2255, 1055, 2245, 1730, 1300, 1925,
       2210, 2025, 1615, 1330, 1435, 2220,  955, 2115, 1545, 2115, 1735,
       1315, 2005, 1320,  950, 1440, 1000, 1250, 2005, 2255, 1645, 1905,
       1405, 2135, 1615, 1500, 2100, 1105, 1305, 2000, 2135, 1645, 1815,
       1235, 2000, 2255, 1320, 1605, 2155, 2045, 1745, 1110, 1400, 1130,
       1645, 2135, 1800, 1105, 1955, 1340, 1750, 1435, 2225, 1205, 1035,
       2115, 1315, 1530, 1830, 1950, 1000, 1225, 2150, 1655,  915, 2255,
       1520, 1845, 1220, 2115, 1550, 1745, 1420, 2320, 1905, 1150, 2245,
         15, 2000, 1710, 1600, 1405, 1605, 1810, 1855, 1125,  825])
```

In [174…
```
# Accuracy de modelo SVM
trained_accuracy_svm = metrics.accuracy_score(y , y_pred)
print(trained_accuracy_svm)
```

```
0.462
```

In [175…
```
print("Accuracy sin modificación de parámetros")
print("Accuracy Decision Tree: " , accuracy_decision_tree)
print("Accuracy KNN: " , accuracy_knn)
print("Accuracy SVM: " , accuracy_svm)
```

```
Accuracy sin modificación de parámetros
Accuracy Decision Tree:  0.9028156630526322
Accuracy KNN:  0.5089909481411645
Accuracy SVM:  0.065
```

In [176…
```
print("Accuracy con parámetros modificados")
print("Decision tree con parámetros: " ,trained_accuracy_decision_tree)
print("KNN con parámetros: " , trained_accuracy_knn)
print("SVM con parámetros: " , trained_accuracy_svm)
```

```
Accuracy con parámetros modificados
Decision tree con parámetros:  0.09276169088087458
KNN con parámetros:  0.4555563794683291
SVM con parámetros:  0.462
```

- Exercici 4

Compara el seu rendiment utilitzant l'aproximació traint/test o cross-validation.

Decision Tree Model

In [177…
```
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [178…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [179…
```
# creación del objeto
clf = DecisionTreeClassifier()
# Train del Decision Tree Classifer
clf = clf.fit(X_train,y_train)
# prediccíones de test dataset
y_pred = clf.predict(X_test)
```

In [180…
```
# Accuracy de modelo Decision Tree
tt_accuracy_decision_tree = metrics.accuracy_score(y_test, y_pred)
```

In [181…
```
# Matriz de confianza
tt_matrix_decision_tree = metrics.confusion_matrix(y_test, y_pred)
```

In [182…
```
# Separación de la matriz de confianza
tt_confusion_decision_tree = metrics.confusion_matrix(y_test, y_pred)
print(tt_confusion_decision_tree)
#[fila, columna]
TP_DT = confusion_decision_tree[1, 1]
TN_DT = confusion_decision_tree[0, 0]
FP_DT = confusion_decision_tree[0, 1]
FN_DT = confusion_decision_tree[1, 0]
```

```
[[112  25   7 ...  27  17  50]
 [ 19  74   6 ...   3   1  24]
 [ 13   5  50 ...   7   4   5]
 ...
 [ 14   2   7 ...  74  26  24]
 [ 19  12   7 ...  27 121  40]
 [ 41   8  10 ...  33  41 201]]
```

In [183…
```
#classification accuracy
tt_classification_accuracy_decision_tree = (TP_DT + TN_DT) / float(TP_DT + TN_DT + FP_DT + FN_DT)
print(tt_classification_accuracy_decision_tree)
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.9955099422706863
0.25192221113867286
```

In [184…
```
#classification error
tt_classification_error_decision_tree = (FP_DT + FN_DT) / float(TP_DT + TN_DT + FP_DT + FN_DT)
print(tt_classification_error_decision_tree)
print(1 - metrics.accuracy_score(y_test, y_pred))
```

```
0.004490057729313663
0.7480777888613271
```

In [185…
```
# sensibilidad
tt_sensitivity_decision_tree = (TP_DT) / float(FN_DT + TP_DT)
print(tt_sensitivity_decision_tree)
print(metrics.recall_score(y_test, y_pred, average='micro'))
```

```
0.988394584139265
0.25192221113867286
```

In [186…
```
#especifidad
tt_specificity_decision_tree = (TN_DT) / (TN_DT + FP_DT)
print(specificity_decision_tree)
```

```
0.9990403071017274
```

In [187…
```
# ratio de falsos positivos
tt_false_positive_rate_decision_tree = (FP_DT) / float(TN_DT + FP_DT)
print(tt_false_positive_rate_decision_tree)
print(1 - specificity_decision_tree)
```

```
0.0009596928982725527
0.0009596928982725794
```

In [188…
```
# precisión
tt_precision_decision_tree = (TP_DT) / float(TP_DT + FP_DT)
print(tt_precision_decision_tree)
print(metrics.precision_score(y_test, y_pred, average='micro'))
```

```
0.998046875
0.25192221113867286
```

KNN

In [189…
```
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [190…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [191…
```
classifier = KNeighborsClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

In [192…
```
# Accuracy de modelo KNN
tt_accuracy_knn = metrics.accuracy_score(y_test, y_pred)
```

In [193…

```python
# Separación de la matriz de confianza
tt_confusion_knn = metrics.confusion_matrix(y_test, y_pred)
print(tt_confusion_knn)
#[fila, columna]
TP_KNN = confusion_knn[1, 1]
TN_KNN = confusion_knn[0, 0]
FP_KNN = confusion_knn[0, 1]
FN_KNN = confusion_knn[1, 0]
```

```
[[170  10   7 ...  10  17  50]
 [ 40  62   5 ...   2   3  19]
 [ 21  10  50 ...   1   5   9]
 ...
 [ 37   3   5 ...  66  25  22]
 [ 49   6   7 ...  29  95  40]
 [ 88   9  13 ...  29  22 190]]
```

In [194…

```python
#classification accuracy
tt_classification_accuracy_knn =(TP_KNN + TN_KNN) / float(TP_KNN + TN_KNN + FP_KNN + FN_KNN)
print(metrics.accuracy_score(y_test, y_pred))
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.2909402282818831
0.2909402282818831
```

In [195…

```python
# Classification error
tt_classification_error_knn = (FP_KNN + FN_KNN) / float(TP_KNN + TN_KNN + FP_KNN + FN_KNN)
print(tt_classification_error_knn)
print(1 - metrics.accuracy_score(y_test, y_pred))
```

```
0.08727569331158239
0.7090597717181168
```

In [196…

```python
# sensibilidad
tt_trained_sensitivity_knn = (TP_KNN) / float(FN_KNN + TP_KNN)
print(tt_trained_sensitivity_knn)
print(metrics.recall_score(y_test, y_pred, average='micro'))
```

```
0.8120300751879699
0.2909402282818831
```

In [197…

```python
# Especificidad
tt_trained_specificity_knn = TN_KNN / (TN_KNN + FP_KNN)
print(tt_trained_specificity_knn)
```

```
0.9613059250302297
```

In [198…

```python
# Ratio falsos positivos
tt_trained_false_positive_rate_knn = (FP_KNN) / float(TN_KNN + FP_KNN)
print(tt_trained_false_positive_rate_knn)
print(1 - tt_trained_specificity_knn)
```

```
0.03869407496977025
0.03869407496977029
```

In [199…

```python
# Precisión
tt_trained_precision_knn = (TP_KNN) / float(TP_KNN + FP_KNN)
print(tt_trained_precision_knn)
print(metrics.precision_score(y_test, y_pred, average='micro'))
```

```
0.9101123595505618
0.2909402282818831
```

Support vector Machine

In [200…

```python
feature_cols = ['CRSDepTime', 'DepTime', 'ArrTime',]
X = wn[feature_cols] # Caracteristicas
y = wn.CRSArrTime # Eje objetivo
```

In [201…

```python
X = X[ :1000]
y = y[ :1000]
```

In [202…

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [203…

```python
#Create a svm Classifier
clf = svm.SVC(kernel='linear')
#Train the model using the training sets
clf.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

In [204…

```python
# Accuracy de modelo SVM
c = metrics.accuracy_score(y_test, y_pred)
```

In [205…

```python
# Separación de la matriz de confianza
tt_confusion_svm = metrics.confusion_matrix(y_test, y_pred)
print(tt_confusion_svm)
#[fila, columna]
TP_SVM = confusion_svm[1, 1]
TN_SVM = confusion_svm[0, 0]
FP_SVM = confusion_svm[0, 1]
FN_SVM = confusion_svm[1, 0]
```

```
[[0 0 1 ... 0 0 1]
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]]
```

In [206…

```python
#classification accuracy
tt_classification_accuracy_svm = (TP_SVM + TN_SVM) / float(TP_SVM + TN_SVM + FP_SVM + FN_SVM)
print(tt_classification_accuracy_svm)
print(metrics.accuracy_score(y_test, y_pred))
```

```
nan
0.12
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/2008478641.py:2: RuntimeWarning: invalid value encountered in true_divide
  tt_classification_accuracy_svm = (TP_SVM + TN_SVM) / float(TP_SVM + TN_SVM + FP_SVM + FN_SVM)
```

In [207…

```python
# classification error
tt_classification_error_svm = (FP_SVM + FN_SVM) / float(TP_SVM + TN_SVM + FP_SVM + FN_SVM)
print(tt_classification_error_svm)
print(1 - metrics.accuracy_score(y_test, y_pred))
```

```
nan
0.88
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/385855457.py:2: RuntimeWarning: invalid value encountered in true_divide
  tt_classification_error_svm = (FP_SVM + FN_SVM) / float(TP_SVM + TN_SVM + FP_SVM + FN_SVM)
```

In [208…

```python
# sensibilidad
tt_sensitivity_svm = (TP_SVM) / float(FN_SVM + TP_SVM)
print(tt_sensitivity_svm)
print(metrics.recall_score(y_test, y_pred, average='micro'))
```

```
nan
0.12
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/1600740172.py:2: RuntimeWarning: invalid value encountered in true_divide
  tt_sensitivity_svm = (TP_SVM) / float(FN_SVM + TP_SVM)
```

In [209…

```python
# Especificidad
tt_specificity_svm = (TN_SVM) / (TN_SVM + FP_SVM)
print(tt_specificity_svm)
```

```
nan
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/4151203144.py:2: RuntimeWarning: invalid value encountered in long_scalars
  tt_specificity_svm = (TN_SVM) / (TN_SVM + FP_SVM)
```

In [210…

```python
# Ratio falsos positivos
tt_false_positive_rate_svm = (FP_SVM) / float(TN_SVM + FP_SVM)
print(tt_false_positive_rate_svm)
print(1 - tt_specificity_svm)
```

```
nan
nan
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/1009052086.py:2: RuntimeWarning: invalid value encountered in true_divide
  tt_false_positive_rate_svm = (FP_SVM) / float(TN_SVM + FP_SVM)
```

In [211…

```python
# precisión
tt_precision_svm = (TP_SVM) / float(TP_SVM + FP_SVM)
print(tt_precision_svm)
print(metrics.precision_score(y_test, y_pred, average='micro'))
```

```
nan
0.12
/var/folders/s5/_b24t8m574q3h2svhkm3k6500000gn/T/ipykernel_804/1451187741.py:2: RuntimeWarning: invalid value encountered in true_divide
  tt_precision_svm = (TP_SVM) / float(TP_SVM + FP_SVM)
```

In [212…

```python
print("Accuracy de los tres modelos Decision Tree")
print("Accuracy no entrenada: " , accuracy_decision_tree)
print("Accuracy entrenada: " , trained_accuracy_decision_tree)
print("Train Test accuracy: " , tt_accuracy_decision_tree)
```

```
Accuracy de los tres modelos Decision Tree
Accuracy no entrenada:  0.9028156630526322
Accuracy entrenada:  0.09276169088087458
Train Test accuracy:  0.25192221113867286
```

In [213…

```python
print("Accuracy de los tres modeos KNN")
print("Decision tree con parámetros: " ,trained_accuracy_knn)
print("KNN con parámetros: " , trained_accuracy_knn)
print("SVM con parámetros: " , tt_accuracy_knn)
```

```
Accuracy de los tres modeos KNN
Decision tree con parámetros:  0.4555563794683291
KNN con parámetros:  0.4555563794683291
SVM con parámetros:  0.2909402282818831
```

In [215…

```python
print("Accuracy de los tres modeos KNN")
print("Accuracy sin parámetros: " ,classification_accuracy_svm)
print("Accuracy con parámetros " , trained_accuracy_svm)
print("Accuracy con train/test: " , tt_classification_accuracy_svm)
```

```
Accuracy de los tres modeos KNN
Accuracy sin parámetros:  nan
Accuracy con parámetros  0.462
Accuracy con train/test:  nan
```

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