Pràctica 2: Neteja i validació de les dades

Xavi Medina Torregrosa

Contenido

1. Descripció del dataset

El dataset escollit per a realitzar la pràctica és el conjunt de dades de les persones del Titanic, que està disponible en https://www.kaggle.com/c/titanic/data per a practicar models predictius.

El dataset conté la següent informació:

- **PassengerId**: Identificador del passatger, valor auto numèric.
- **Survived**: Indica si el passatger va sobreviure o no, si el valor és 0 no va sobreviure, i amb el valor 1 si va sobreviure.
- **Pclass**: Classe del tiquet amb el que van embarcar, 1 per primer classe, 2 per segona classe i 3 per tercera classe.
- **Name**: Nom del passatger, amb el format 'cognom, títol nom'.
- **Sex**: Gènere del passatger, amb els valor 'male' i 'female'.
- **Age**: Edat del passatger.
- **SibSp**: Número de germans i esposes a bord.
- **Parch**: Número de pares a bord.
- **Ticket**: Número del tiquet.
- **Fare**: Tarifa que van pagar.
- **Cabin**: Número de la cabina.
- **Embarked**: Port en el que van embarcar, C = Cherbourg, Q = Queenstown, S = Southampton.

El dataset d'entrenament conté 891 registres, mentre que el de test conté 417.

2. Integració i selecció de les dades d'interès a analitzar

Per a realitzar aquesta pràctica, haurem de netejar tots dos datasets, l'única diferencia que tenen és que el de test no conté la variable Survived, que és la variable que s'ha de predir.

Fen-nos una idea de l'època en la que va passar l'incident, ens hem de fixar en les variables que impliquin una diferencia de classe entre els passatgers, segurament les dones, nens o famílies completes tenien prioritat a persones que estaven soles, la classe social, etc...

Naturalment, podem descartar directament l'anàlisi de PassengerId, ja que no aporta cap valor.

3. Neteja de les dades

Per començar, ajuntarem els dos datasets per a tenir més quantitat de registres:

```
# Combinar datasets
def combine_datasets(train, test):
    # Eliminem la variable survived del dataset d'entrenament
    train.drop('Survived', 1, inplace=True)

    # Combinem els datasets
    combined_ds = train.append(test)
    combined_ds.reset_index(inplace=True)
    combined_ds.drop('index', inplace=True, axis=1)
```

```
    return combined_ds
```

El primer que hauríem de fer és aprofitar que dins el nom tenim el títol social de la persona, per tal de crear una nova variable amb aquesta informació, això ens ajudarà a classificar millor als passatgers en funció de la seva classe social, per això creem un diccionari per mapejar els títols de cada persona per tal de normalitzar-los i agrupar-los en categories.

```
def get_titles(dataset):

    # Obtenim el títol del nom i creem una nova columna
    dataset['Title'] = dataset['Name'].map(lambda name:
name.split(',')[1].split('.')[0].strip())

    # Creem un diccionari per mapejar els titols a una categoria
social
    dictionary = {
        "Capt": "Oficial",
        "Col": "Oficial",
        "Major": "Oficial",
        "Jonkheer": "Reialesa",
        "Don": "Reialesa",
        "Sir": "Reialesa",
        "Dr": "Oficial",
        "Rev": "Oficial",
        "the Countess": "Reialesa",
        "Dona": "Reialesa",
        "Mme": "Mrs",
        "Mlle": "Miss",
        "Ms": "Mrs",
        "Mr": "Mr",
        "Mrs": "Mrs",
        "Miss": "Miss",
        "Master": "Master",
        "Lady": "Reialesa"

    }

    # Mapejem els valors per normalitzar-los
    dataset['Title'] = dataset.Title.map(dictionary)
```

Si utilitzem el mètode describe del dataset, podem veure que hi ha persones que no tenen l'edat informada, hi ha un altre en el dataset de test que no té el Fare informat, i altres amb l'Embarked.

```
def explore_dataset(dataset):
    print dataset.describe()
```

```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

             Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
```

Per a omplir les dades de l'edat que falten, el que farem serà agrupar els dades per Sexe, Classe i Títol, i llavors calcular la mitjana de l'edat de passatgers similars a ells, ja que per exemple, no té sentit que un passatger que sigui un Oficial del vaixell, influenciï l'edat d'un nen.

```
group_by_columns = ['Sex', 'Pclass', 'Title']
# Agrupem les dades per sexe, classe i títol
grouped_train = self.dataset.head(891).groupby(group_by_columns)
grouped_median_train = grouped_train.median()
grouped_test = self.dataset.iloc[891:].groupby(group_by_columns)
grouped_median_test = grouped_test.median()
```

Un cop agrupades les dades, calculem la mitjana en funció de les tres variables:

```
def fill_age_by_sex_and_class(self, row, grouped_median):
    if row['Sex'] == 'female' and row['Pclass'] == 1:
        if row['Title'] == 'Miss':
            return grouped_median.loc['female', 1, 'Miss']['Age']
        elif row['Title'] == 'Mrs':
            return grouped_median.loc['female', 1, 'Mrs']['Age']
        elif row['Title'] == 'Oficial':
            return grouped_median.loc['female', 1, 'Oficial']['Age']
        elif row['Title'] == 'Reialesa':
            return grouped_median.loc['female', 1, 'Reialesa']['Age']

    elif row['Sex'] == 'female' and row['Pclass'] == 2:
        if row['Title'] == 'Miss':
            return grouped_median.loc['female', 2, 'Miss']['Age']
        elif row['Title'] == 'Mrs':
            return grouped_median.loc['female', 2, 'Mrs']['Age']

    elif row['Sex'] == 'female' and row['Pclass'] == 3:
        if row['Title'] == 'Miss':
            return grouped_median.loc['female', 3, 'Miss']['Age']
        elif row['Title'] == 'Mrs':
            return grouped_median.loc['female', 3, 'Mrs']['Age']
```

```
    elif row['Sex'] == 'male' and row['Pclass'] == 1:
        if row['Title'] == 'Master':
            return grouped_median.loc['male', 1, 'Master']['Age']
        elif row['Title'] == 'Mr':
            return grouped_median.loc['male', 1, 'Mr']['Age']
        elif row['Title'] == 'Oficial':
            return grouped_median.loc['male', 1, 'Oficial']['Age']
        elif row['Title'] == 'Reialesa':
            return grouped_median.loc['male', 1, 'Reialesa']['Age']

    elif row['Sex'] == 'male' and row['Pclass'] == 2:
        if row['Title'] == 'Master':
            return grouped_median.loc['male', 2, 'Master']['Age']
        elif row['Title'] == 'Mr':
            return grouped_median.loc['male', 2, 'Mr']['Age']
        elif row['Title'] == 'Oficial':
            return grouped_median.loc['male', 2, 'Oficial']['Age']

    elif row['Sex'] == 'male' and row['Pclass'] == 3:
        if row['Title'] == 'Master':
            return grouped_median.loc['male', 3, 'Master']['Age']
        elif row['Title'] == 'Mr':
            return grouped_median.loc['male', 3, 'Mr']['Age']
```

Eliminem la columna Name, ja que el nom no ens aporta cap valor ara que tenim el títol, i normalitzem la columna Títol per a crear els dummies amb valors booleans per a cadascun.

```
def clean_names(self):
    # Eliminem la columna Name
    self.dataset.drop('Name', axis=1, inplace=True)

    # Creem una variable dummy dels títols
    titles_dummies = pd.get_dummies(self.dataset['Title'],
prefix='Title')
    # Combinem la nova columna amb el dataset
    self.dataset = pd.concat([self.dataset, titles_dummies], axis=1)

    # Eliminem l'antiga columna del títol
    self.dataset.drop('Title', axis=1, inplace=True)
```

També hi ha valors nuls a la variable Fare, per tant, omplim els valors amb el valor més típic:

```
def clean_fares(self):

self.dataset.head(891).Fare.fillna(self.dataset.head(891).Fare.mean(),
inplace=True)

self.dataset.iloc[891:].Fare.fillna(self.dataset.iloc[891:].Fare.mean(
), inplace=True)
```

Els valors nuls de Embarked, els omplim amb el valor més típic dintre del dataset, que és 'S', un cop tenim totes les files amb els valors informats a la variable Embarked, normalitzem la variable creant columnes dummy:

```
def clean_embarked(self):
    # Omplim els valors buits amb el valor més típic
```

```
    self.dataset.head(891).Embarked.fillna('S', inplace=True)
    self.dataset.iloc[891:].Embarked.fillna('S', inplace=True)
    # Creem les variables dummy per a normalitzar els valors
    embarked_dummies = pd.get_dummies(self.dataset['Embarked'],
prefix='Embarked')
    self.dataset
= pd.concat([self.dataset, embarked_dummies], axis=1)
    # Eliminem la columna de Embarked
    self.dataset
.drop('Embarked', axis=1, inplace=True)
```

Ara netegem la variable Cabin, que també té valors nuls, en aquest cas, omplirem els valors nuls amb una lletra 'U' (Unknown), ja que per la resta de valors, només ens quedarem amb la lletra, i tornarem a crear variables dummy, per a que sigui molt més fàcil de classificar per al model posterior.

```
def clean_cabin(self):
    # Omplim els valors buits amb Unknown
    self.dataset.Cabin.fillna('U', inplace=True)
    # Modifiquem els valors de la cabina per a quedar-nos amb la
lletra
    self.dataset['Cabin'] = self.dataset['Cabin'].map(lambda c: c[0])

    # Fem un dummy encoding de la variable
    cabin_dummies = pd.get_dummies(self.dataset['Cabin'],
prefix='Cabin')
    self.dataset
= pd.concat([self.dataset, cabin_dummies], axis=1)
    # Eliminem la columna original
    self.dataset
.drop('Cabin', axis=1, inplace=True)
```

Normalitzem els valors de la variable Sex:

```
def clean_sex(self):
    # Normalitzem els valors a numeric per a que sigui més fàcil
treballar
    self.dataset['Sex'] = self.dataset['Sex'].map({'male': 1,
'female': 0})
```

Tornem a crear variables dummy per a la columna Pclass:

```
def clean_pclass(self):
    # Creem les variables dummy per Pclass
    pclass_dummies = pd.get_dummies(self.dataset['Pclass'],
prefix="Pclass")
    self.dataset
= pd.concat([self.dataset, pclass_dummies], axis=1)
```

Extraiem ara els prefixos dels tiquets:

```
def extract_ticket_number(self, ticket):
    ticket = ticket.replace('.', '')
    ticket = ticket.replace('/', '')
    ticket = ticket.split()
    ticket = map(lambda t: t.strip(), ticket)
```

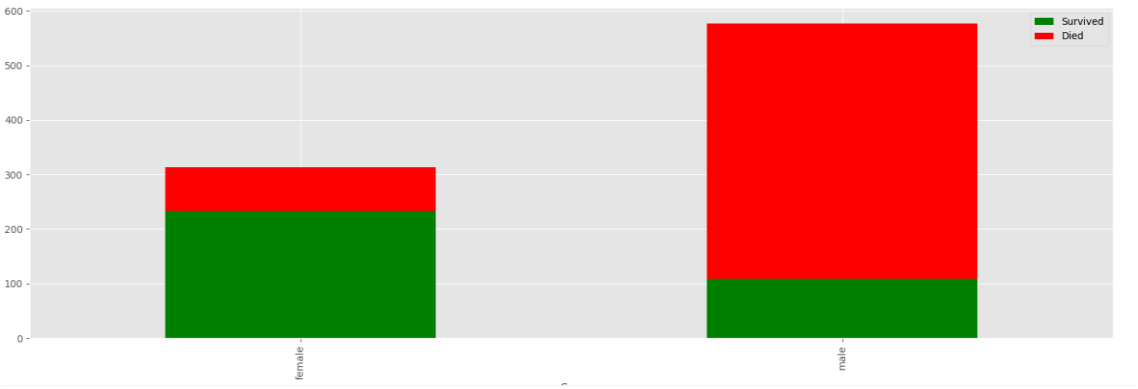
```
    ticket = filter(lambda t: not t.isdigit(), ticket)
    if len(ticket) > 0:
        return ticket[0]
    else:
        return 'U'

def clean_ticket(self):
    # Extraiem els prefixos dels tiquets, si no en té, possem 'U' per
Unknwon
    self.dataset['Ticket'] =
self.dataset['Ticket'].map(self.extract_ticket_number)
    # Creem les variables dummy per a les numeracions dels tiquets
    tickets_dummies = pd.get_dummies(self.dataset['Ticket'],
prefix='Ticket')
    self.dataset
= pd.concat([self.dataset, tickets_dummies], axis=1)
    # Eliminem la columna Ticket
    self.dataset
.drop('Ticket', inplace=True, axis=1)
```

Per últim, com tenim dues variables que tenen en compte els familiars del passatgers, podem combinar les variables en una sola que indiqui el tamany de la família a bord:

```
def clean_family(self):
    # Creem una nova columna, que combina el nombre de germans +
esposes + fills, així tenim una variable amb
    #  el tamany de la familia
    self.dataset['Family'] = self.dataset['Parch'] +
self.dataset['SibSp'] + 1
```

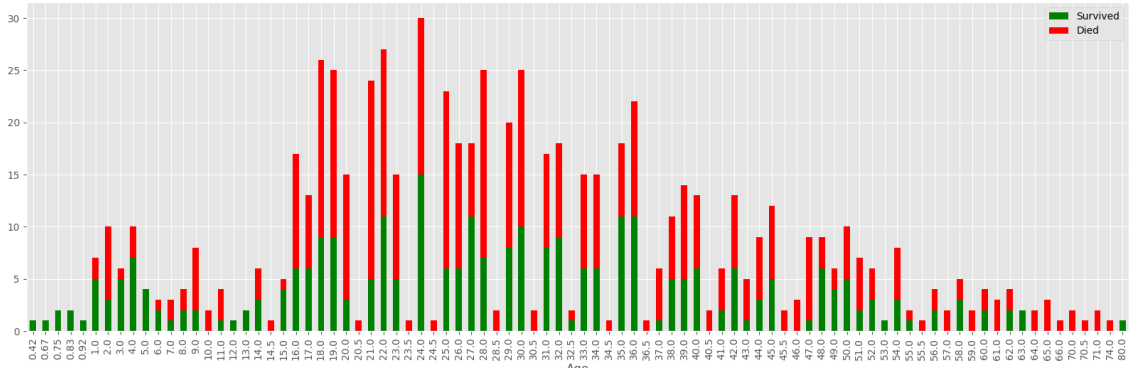
4. Anàlisi de les dades

Podem observar la relació de les variables amb la possibilitat de que sobrevisqui o no el passatger, per exemple, per el sexe, podem observar el següent:

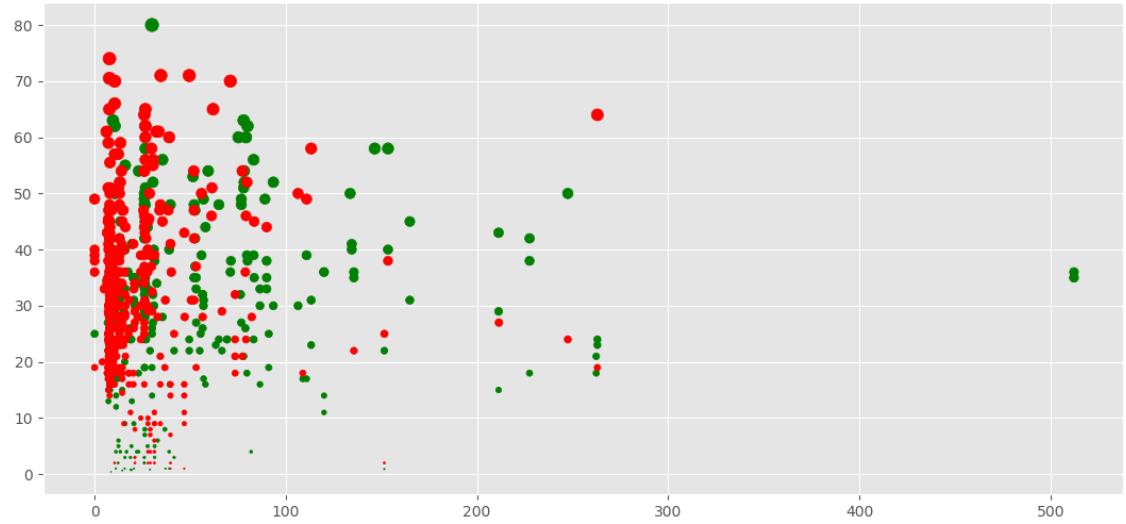


Com podem veure, la prioritat a l'hora de evacuar el vaixell, sembla que les dones tenien prioritat, per això les seves possibilitats de sobreviure són molt més altes que les dels homes.

Podem observar ara per edat, on podem observar que la majoria dels passatgers, es troben entre els 16 i 40 anys, i no sembla haver-hi una gran relació entre una edat concreta i les possibilitats de sobreviure.



També podem comprovar varies variables amb la possibilitat de que sobrevisquin, per exemple, si observem l'edat i el preu del tiquet:



Podem veure que la majoria ha pagat tarifes d'entre 0 i 150, mentre que hi ha alguns valors extrems per sobre dels 500. A més, els que han pagat tant, han sobreviscut, el que hem fa pensar que segurament no són dades errònies, si no que serien les passatgers més 'premium' del vaixell, i els van evacuar dels primers.

Podem realitzar el test d'Anderson per veure si els valors tenen una distribució normal, per a les variables principals que tenim:

```
# Test Anderson
DataExploration.anderson_darling_test(combined_ds['Age'])
DataExploration.anderson_darling_test(combined_ds['Fare'])
DataExploration.anderson_darling_test(combined_ds['Sex'])
DataExploration.anderson_darling_test(combined_ds['Family'])
DataExploration.anderson_darling_test(combined_ds['Pclass'])
```

```
AndersonResult(statistic=15.867629199936346, critical_values=array([
0.574,  0.654,  0.785,  0.915,  1.089]), significance_level=array([
15. ,  10. ,   5. ,   2.5,   1. ]))
AndersonResult(statistic=186.88981521417054, critical_values=array([
0.574,  0.654,  0.785,  0.915,  1.089]), significance_level=array([
15. ,  10. ,   5. ,   2.5,   1. ]))
AndersonResult(statistic=261.1420397725326, critical_values=array([
0.574,  0.654,  0.785,  0.915,  1.089]), significance_level=array([
15. ,  10. ,   5. ,   2.5,   1. ]))
AndersonResult(statistic=171.91379249725696, critical_values=array([
0.574,  0.654,  0.785,  0.915,  1.089]), significance_level=array([
15. ,  10. ,   5. ,   2.5,   1. ]))
AndersonResult(statistic=157.92824415689665, critical_values=array([
0.574,  0.654,  0.785,  0.915,  1.089]), significance_level=array([
15. ,  10. ,   5. ,   2.5,   1. ]))
```

Com podem veure els valors dels estadístics per a totes les variables, és molt més gran que els valors crítics, per tant, podem refusar la hipòtesi de que aquests valors segueixen una distribució normal.

Per últim podem fer un test de Fligner per veure la homogeneïtat entre el preu que van pagar i la classe:

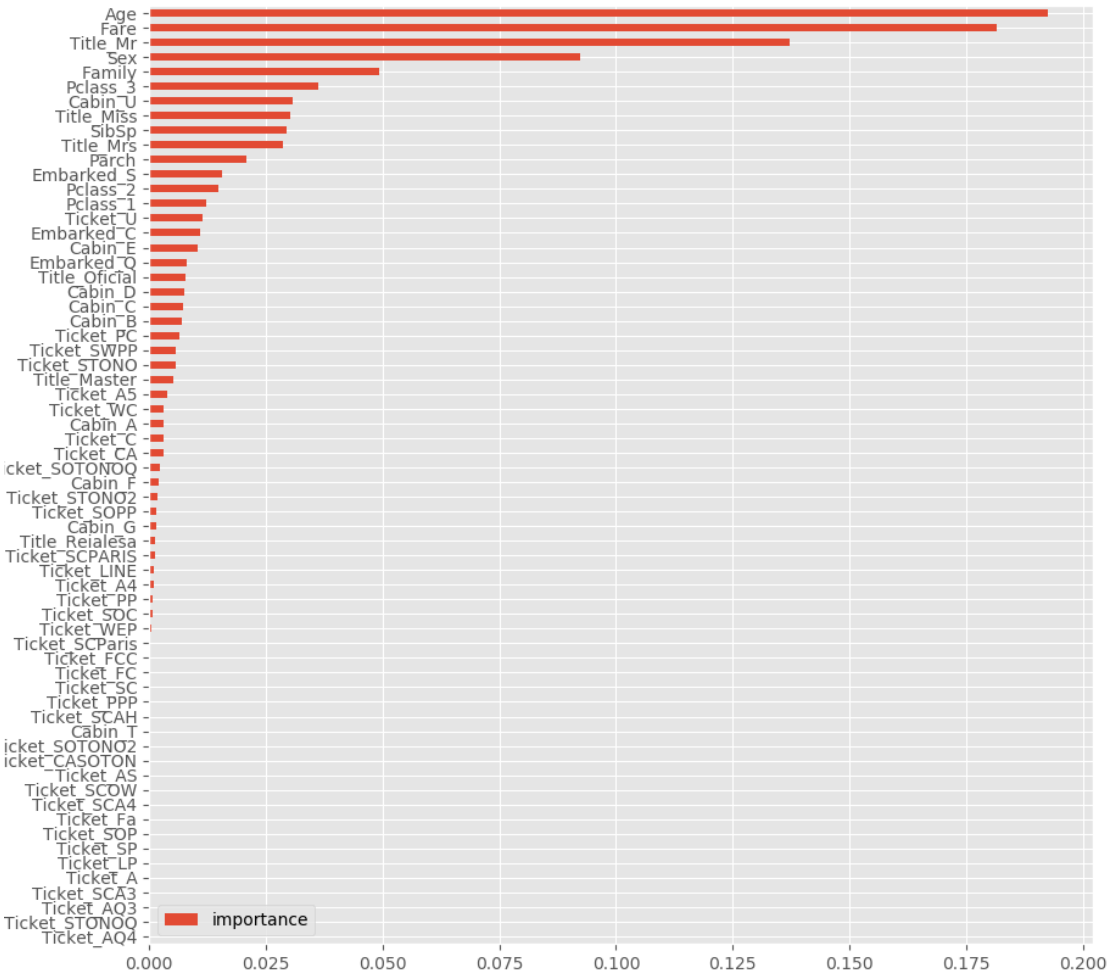
```
# Test Fligner
DataExploration.fligner_test(combined_ds['Fare'],
combined_ds['Pclass'])
```

FlignerResult(statistic=1408.8196319876702, pvalue=2.5462940558192438e-308)

Com el p-value és superior a 0.05, podem acceptar la hipòtesis de que les variàncies del Fare i Pclass són homogènies.

5. Representació dels resultats a partir de taules i gràfiques

Un cop tenim totes les dades netes i normalitzades, podem obtenir un gràfic de quina importància té cada variable sobre la que volem predir:



En aquest gràfic, com podem veure, l'edat, el preu, el sexe i el títol de Mr. són les variables que més pes tenen en aquest model.

6. Resolució del problema

Ara ja podem resoldre el problema, que consistia en predir si un passatger sobreviuria o no, per això, he fet servir un RandomForestClassifier, ja que aquest problema el vaig resoldre mesos enrere, i els DecisionTrees no donaven un accuracy suficientment alt com per a ser fiables.

```
def score_model(dataset):
    train, test, targets = recover_train_test_target(dataset)

    randomForestClassifier = RandomForestClassifier(n_estimators=50,
max_features='sqrt')
    randomForestClassifier = randomForestClassifier.fit(train,
targets)

    DataExploration.show_variable_relation_with_survival(train,
randomForestClassifier)

    model = SelectFromModel(randomForestClassifier, prefit=True)
    train_reduced = model.transform(train)
```

```
    test_reduced = model.transform(test)

    parameters = {'bootstrap': False, 'min_samples_leaf': 3,
'n_estimators': 50,
                  'min_samples_split': 10, 'max_features': 'sqrt',
'max_depth': 6}

    model = RandomForestClassifier(**parameters)
    model.fit(train, targets)

    print 'Score: ', compute_score(model, train, targets,
scoring='accuracy')

    output = model.predict(test).astype(int)
    df_output = pd.DataFrame()
    aux = pd.read_csv('test.csv')
    df_output['PassengerId'] = aux['PassengerId']
    df_output['Survived'] = output
    df_output[['PassengerId', 'Survived']].to_csv('solution.csv',
index=False)


def recover_train_test_target(dataset):

    train_original = pd.read_csv('train.csv')

    targets = train_original.Survived
    train = dataset.head(891)
    test = dataset.iloc[891:]

    return train, test, targets

def compute_score(clf, X, y, scoring='accuracy'):
    xval = cross_val_score(clf, X, y, cv = 5, scoring=scoring)
    return np.mean(xval)
```

Al calcular l'score d'aquest model amb el dataset de training, contra el de test, el resultat és el següent:

Score: 0.829431194626

7. Codi

El codi es pot trobar a https://github.com/XaviMedina/UOCDataCleaning