Part 1: Load the data

```
import pandas as pd

data_sina = pd.read_csv('student-por.csv', sep=';')
```

Part 2: Carryout some initial investigations:

```
# a. Check the names and types of columns
data_sina.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64 
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64 
 7   Fedu        649 non-null    int64 
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64 
 13  studytime   649 non-null    int64 
 14  failures    649 non-null    int64 
 15  schoolsup   649 non-null    object
 16  famsup      649 non-null    object
 17  paid        649 non-null    object
 18  activities  649 non-null    object
 19  nursery     649 non-null    object
 20  higher      649 non-null    object
 21  internet    649 non-null    object
 22  romantic    649 non-null    object
 23  famrel      649 non-null    int64 
 24  freetime    649 non-null    int64 
 25  goout       649 non-null    int64 
 26  Dalc        649 non-null    int64 
 27  Walc        649 non-null    int64 
 28  health      649 non-null    int64 
 29  absences    649 non-null    int64 
 30  G1          649 non-null    int64 
 31  G2          649 non-null    int64 
 32  G3          649 non-null    int64 
```

```
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
```

```
# b. Check the missing values.
data_sina.isnull().sum()
```

```
school        0
sex           0
age           0
address       0
famsize       0
Pstatus       0
Medu          0
Fedu          0
Mjob          0
Fjob          0
reason        0
guardian      0
traveltime    0
studytime     0
failures      0
schoolsup     0
famsup        0
paid          0
activities    0
nursery       0
higher        0
internet      0
romantic      0
famrel        0
freetime      0
goout         0
Dalc          0
Walc          0
health        0
absences      0
G1            0
G2            0
G3            0
dtype: int64
```

```
# c. Check the statistics of the numeric fields (mean, min, max,
median, count,..etc.)
data_sina.describe()
```

```
              age        Medu        Fedu  traveltime   studytime
failures  \
count  649.000000  649.000000  649.000000  649.000000  649.000000
649.000000
mean    16.744222    2.514638    2.306626    1.568567    1.930663
0.221880
```

```
std      1.218138    1.134552    1.099931    0.748660    0.829510
0.593235
min     15.000000    0.000000    0.000000    1.000000    1.000000
0.000000
25%     16.000000    2.000000    1.000000    1.000000    1.000000
0.000000
50%     17.000000    2.000000    2.000000    1.000000    2.000000
0.000000
75%     18.000000    4.000000    3.000000    2.000000    2.000000
0.000000
max     22.000000    4.000000    4.000000    4.000000    4.000000
3.000000

           famrel    freetime       goout        Dalc        Walc
health  \
count  649.000000  649.000000  649.000000  649.000000  649.000000
649.000000
mean     3.930663    3.180277    3.184900    1.502311    2.280431
3.536210
std      0.955717    1.051093    1.175766    0.924834    1.284380
1.446259
min      1.000000    1.000000    1.000000    1.000000    1.000000
1.000000
25%      4.000000    3.000000    2.000000    1.000000    1.000000
2.000000
50%      4.000000    3.000000    3.000000    1.000000    2.000000
4.000000
75%      5.000000    4.000000    4.000000    2.000000    3.000000
5.000000
max      5.000000    5.000000    5.000000    5.000000    5.000000
5.000000

          absences          G1          G2          G3
count  649.000000  649.000000  649.000000  649.000000
mean     3.659476   11.399076   11.570108   11.906009
std      4.640759    2.745265    2.913639    3.230656
min      0.000000    0.000000    0.000000    0.000000
25%      0.000000   10.000000   10.000000   10.000000
50%      2.000000   11.000000   11.000000   12.000000
75%      6.000000   13.000000   13.000000   14.000000
max     32.000000   19.000000   19.000000   19.000000
```

```
# d. Check the categorical values and their frequencies
data_sina.describe(include=['object'])
```

```
       school  sex address famsize Pstatus   Mjob   Fjob  reason
guardian  \
count     649  649     649     649     649    649    649     649
649
unique      2    2       2       2       2      5      5       4
```

```
3
top        GP    F       U     GT3       T  other  other  course
mother
freq      423  383     452     457     569    258    367     285
455

       schoolsup famsup paid activities nursery higher internet
romantic
count        649    649  649        649     649    649      649
649
unique         2      2    2          2       2      2        2
2
top           no    yes   no         no     yes    yes      yes
no
freq         581    398  610        334     521    580      498
410
```

```
# TODO e. in your written response write a paragraph explaining your
findings about each column
```

Part 3: Pre processing

```
# if G1 + G2 + G3 >= 35 then 1 else 0
data_sina['pass_sina'] = data_sina.apply(lambda x: 1 if x['G1'] +
x['G2'] + x['G3'] >= 35 else 0, axis=1)

# Drop the columns G1, G2, G3
data_sina.drop(['G1', 'G2', 'G3'], axis=1, inplace=True)

# Print out the total number of instances in each class
data_sina['pass_sina'].value_counts()
```

```
1    328
0    321
Name: pass_sina, dtype: int64
```

```
# a list to save the names of numeric columns
numeric_features_sina = []
cat_features_sina = []

for col in data_sina.columns:
    if data_sina[col].dtype == 'object':
        cat_features_sina.append(col)
    else:
        numeric_features_sina.append(col)

# TODO: note into your report and explain your findings in terms of
balanced and un-balanced

# check for any missing columns
if len(data_sina.columns) == len(numeric_features_sina) +
len(cat_features_sina):
```

```
    print('No missing columns')
else:
    print('Missing columns')
```

```
No missing columns
```

```
# prepare a column transformer to handle all the categorical values
and convert them into numeric values
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
# use the column transformer to encode the categorical values and
preserve the rest of the columns
transformer_sina = ColumnTransformer(transformers=[('encoder',
OneHotEncoder(), cat_features_sina)], remainder='passthrough')

# prepare a classifier decision tree model with criterion='entropy'
and max_depth=5, and check all possible parameters
from sklearn.tree import DecisionTreeClassifier
clf_sina = DecisionTreeClassifier(criterion='entropy', max_depth=5)

# build a pipeline that first step is the column transformer and the
second step is the classifier
from sklearn.pipeline import Pipeline
pipeline_sina = Pipeline(steps=[('preprocessor', transformer_sina),
('classifier', clf_sina)])

# split the data into training and testing sets with test_size=0.2 and
random_state=0
from sklearn.model_selection import train_test_split
X_sina = data_sina.drop(['pass_sina'], axis=1)
y_sina = data_sina['pass_sina']
X_train_sina, X_test_sina, y_train_sina, y_test_sina =
train_test_split(X_sina, y_sina, test_size=0.2, random_state=0)
```

Part 4: Build Classification Models

```
# fit the pipeline on the training data
pipeline_sina.fit(X_train_sina, y_train_sina)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('encoder',
OneHotEncoder(),
                                                  ['school', 'sex',
'address',
                                                   'famsize',
'Pstatus', 'Mjob',
                                                   'Fjob', 'reason',
'guardian',
                                                   'schoolsup',
'famsup',
                                                   'paid',
```

```
'activities',
                                                  'nursery',
'higher',
                                                  'internet',
'romantic'])])),
                ('classifier',
                 DecisionTreeClassifier(criterion='entropy',
max_depth=5))])
```

```
# cross validate the pipeline on the training data using 10 folds and
seed=98 and shuffle=True
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=98)
scores_sina = cross_val_score(pipeline_sina, X_train_sina,
y_train_sina, cv=cv, scoring='accuracy')

# print the 10-fold cross validation scores
# print the mean and standard deviation of the 10-fold cross
validation scores
print(scores_sina)
```

```
[0.68269231 0.66346154 0.66346154 0.69230769 0.73076923 0.63461538
 0.64423077 0.63461538 0.68269231 0.69230769]
```

```
# visualize the tree using Graphviz
from sklearn.tree import export_graphviz
import graphviz
dot_data = export_graphviz(clf_sina, out_file=None,
class_names=['fail', 'pass'], filled=True, rounded=True,
special_characters=True)

graph = graphviz.Source(dot_data)

graph
```

```
# save the tree as a png file
# TODO find whether this worked
graph.render('student_sina.jpg')
```

```
'student_sina.jpg.pdf'
```

```
# TODO In you written response describe the key findings for the first
level split i.e. which field it split on what are the samples

# print the accuracy for the model on the training set and the testing
set
print('Training accuracy: ', pipeline_sina.score(X_train_sina,
y_train_sina))
print('Testing accuracy: ', pipeline_sina.score(X_test_sina,
y_test_sina))
```

```
Training accuracy:  0.7610789980732178
Testing accuracy:  0.6846153846153846
```

```
# Use the model to predict the test data and printout the accuracy,
precision and recall scores and the confusion matrix
from sklearn.metrics import accuracy_score, precision_score,
recall_score, confusion_matrix
y_pred_sina = pipeline_sina.predict(X_test_sina)
print('Accuracy: ', accuracy_score(y_test_sina, y_pred_sina))
print('Precision: ', precision_score(y_test_sina, y_pred_sina))
print('Recall: ', recall_score(y_test_sina, y_pred_sina))
print('Confusion matrix: ', confusion_matrix(y_test_sina,
y_pred_sina))
```

```
Accuracy:  0.6846153846153846
Precision:  0.6559139784946236
Recall:  0.8714285714285714
Confusion matrix:  [[28 32]
 [ 9 61]]
```

Part 5: Fine tune the model

```
from sklearn.model_selection import RandomizedSearchCV

# Using Randomized grid search fine tune your model using the
following set of parameters
parameters = {
    'classifier__min_samples_split': range(10, 300, 20),
    'classifier__max_depth': range(1, 30, 2),
    'classifier__min_samples_leaf': range(1, 15, 3)
}

rndGrid = RandomizedSearchCV(pipeline_sina, cv=5, scoring='accuracy',
param_distributions=parameters,
                    n_iter=7, refit=True, verbose=3)

# fit the model on the training data
rndGrid.fit(X_train_sina, y_train_sina)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
[CV 1/5] END classifier__max_depth=27, classifier__min_samples_leaf=1,
classifier__min_samples_split=10;, score=0.615 total time=   0.0s
[CV 2/5] END classifier__max_depth=27, classifier__min_samples_leaf=1,
classifier__min_samples_split=10;, score=0.596 total time=   0.0s
[CV 3/5] END classifier__max_depth=27, classifier__min_samples_leaf=1,
classifier__min_samples_split=10;, score=0.663 total time=   0.0s
[CV 4/5] END classifier__max_depth=27, classifier__min_samples_leaf=1,
classifier__min_samples_split=10;, score=0.577 total time=   0.0s
[CV 5/5] END classifier__max_depth=27, classifier__min_samples_leaf=1,
classifier__min_samples_split=10;, score=0.718 total time=   0.0s
[CV 1/5] END classifier__max_depth=5, classifier__min_samples_leaf=7,
classifier__min_samples_split=270;, score=0.692 total time=   0.0s
[CV 2/5] END classifier__max_depth=5, classifier__min_samples_leaf=7,
classifier__min_samples_split=270;, score=0.712 total time=   0.0s
[CV 3/5] END classifier__max_depth=5, classifier__min_samples_leaf=7,
classifier__min_samples_split=270;, score=0.712 total time=   0.0s
[CV 4/5] END classifier__max_depth=5, classifier__min_samples_leaf=7,
classifier__min_samples_split=270;, score=0.635 total time=   0.0s
[CV 5/5] END classifier__max_depth=5, classifier__min_samples_leaf=7,
classifier__min_samples_split=270;, score=0.670 total time=   0.0s
[CV 1/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=90;, score=0.692 total time=   0.0s
[CV 2/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=90;, score=0.712 total time=   0.0s
[CV 3/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=90;, score=0.712 total time=   0.0s
[CV 4/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=90;, score=0.635 total time=   0.0s
[CV 5/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=90;, score=0.670 total time=   0.0s
[CV 1/5] END classifier__max_depth=13,
classifier__min_samples_leaf=10, classifier__min_samples_split=30;,
score=0.673 total time=   0.0s
[CV 2/5] END classifier__max_depth=13,
classifier__min_samples_leaf=10, classifier__min_samples_split=30;,
score=0.702 total time=   0.0s
[CV 3/5] END classifier__max_depth=13,
classifier__min_samples_leaf=10, classifier__min_samples_split=30;,
score=0.721 total time=   0.0s
[CV 4/5] END classifier__max_depth=13,
classifier__min_samples_leaf=10, classifier__min_samples_split=30;,
score=0.692 total time=   0.0s
[CV 5/5] END classifier__max_depth=13,
classifier__min_samples_leaf=10, classifier__min_samples_split=30;,
score=0.718 total time=   0.0s
[CV 1/5] END classifier__max_depth=15,
classifier__min_samples_leaf=10, classifier__min_samples_split=170;,
score=0.663 total time=   0.0s
[CV 2/5] END classifier__max_depth=15,
```

```
classifier__min_samples_leaf=10, classifier__min_samples_split=170;,
score=0.712 total time=   0.0s
[CV 3/5] END classifier__max_depth=15,
classifier__min_samples_leaf=10, classifier__min_samples_split=170;,
score=0.683 total time=   0.0s
[CV 4/5] END classifier__max_depth=15,
classifier__min_samples_leaf=10, classifier__min_samples_split=170;,
score=0.635 total time=   0.0s
[CV 5/5] END classifier__max_depth=15,
classifier__min_samples_leaf=10, classifier__min_samples_split=170;,
score=0.631 total time=   0.0s
[CV 1/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=150;, score=0.692 total time=   0.0s
[CV 2/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=150;, score=0.712 total time=   0.0s
[CV 3/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=150;, score=0.712 total time=   0.0s
[CV 4/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=150;, score=0.635 total time=   0.0s
[CV 5/5] END classifier__max_depth=3, classifier__min_samples_leaf=10,
classifier__min_samples_split=150;, score=0.670 total time=   0.0s
[CV 1/5] END classifier__max_depth=21,
classifier__min_samples_leaf=10, classifier__min_samples_split=270;,
score=0.692 total time=   0.0s
[CV 2/5] END classifier__max_depth=21,
classifier__min_samples_leaf=10, classifier__min_samples_split=270;,
score=0.712 total time=   0.0s
[CV 3/5] END classifier__max_depth=21,
classifier__min_samples_leaf=10, classifier__min_samples_split=270;,
score=0.712 total time=   0.0s
[CV 4/5] END classifier__max_depth=21,
classifier__min_samples_leaf=10, classifier__min_samples_split=270;,
score=0.635 total time=   0.0s
[CV 5/5] END classifier__max_depth=21,
classifier__min_samples_leaf=10, classifier__min_samples_split=270;,
score=0.670 total time=   0.0s

RandomizedSearchCV(cv=5,
                   estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(remainder='passthrough',

transformers=[('encoder',

OneHotEncoder(),

['school',

'sex',
```

```
'address',

'famsize',

'Pstatus',

'Mjob',

'Fjob',

'reason',

'guardian',

'schoolsup',

'famsup',

'paid',

'activities',

'nursery',

'higher',

'internet',

'romantic'])])),
                                                ('classifier',

DecisionTreeClassifier(criterion='entropy',

max_depth=5))]),
                   n_iter=7,
                   param_distributions={'classifier__max_depth':
range(1, 30, 2),

'classifier__min_samples_leaf': range(1, 15, 3),

'classifier__min_samples_split': range(10, 300, 20)},
                   scoring='accuracy', verbose=3)

# print the best parameters
print(rndGrid.best_params_)

{'classifier__min_samples_split': 30, 'classifier__min_samples_leaf':
10, 'classifier__max_depth': 13}
```

```
# print the best score
print(rndGrid.best_score_)
```

```
0.7013816280806572
```

```
# print the best estimator
print(rndGrid.best_estimator_)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('encoder',
OneHotEncoder(),
                                                  ['school', 'sex',
'address',
                                                   'famsize',
'Pstatus', 'Mjob',
                                                   'Fjob', 'reason',
'guardian',
                                                   'schoolsup',
'famsup',
                                                   'paid',
'activities',
                                                   'nursery',
'higher',
                                                   'internet',
'romantic'])])),
                ('classifier',
                 DecisionTreeClassifier(criterion='entropy',
max_depth=13,
                                        min_samples_leaf=10,
                                        min_samples_split=30))])
```

```
# fit the best estimator on the test data
rndGrid.best_estimator_.fit(X_test_sina, y_test_sina)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('encoder',
OneHotEncoder(),
                                                  ['school', 'sex',
'address',
                                                   'famsize',
'Pstatus', 'Mjob',
                                                   'Fjob', 'reason',
'guardian',
                                                   'schoolsup',
'famsup',
                                                   'paid',
'activities',
                                                   'nursery',
'higher',
```

```
                                                        'internet',
'romantic'])])),
                ('classifier',
                 DecisionTreeClassifier(criterion='entropy',
max_depth=13,
                                        min_samples_leaf=10,
                                        min_samples_split=30))])

# print the precision, recall and accuracy scores
y_pred_sina = rndGrid.best_estimator_.predict(X_test_sina)
print('Accuracy: ', accuracy_score(y_test_sina, y_pred_sina))
print('Precision: ', precision_score(y_test_sina, y_pred_sina))
print('Recall: ', recall_score(y_test_sina, y_pred_sina))

Accuracy:  0.7846153846153846
Precision:  0.8088235294117647
Recall:  0.7857142857142857

# TODO compare this with the previous model and write your findings in
the report

# save the model as a pickle file
import pickle
pickle.dump(rndGrid.best_estimator_, open('student_sina.pkl', 'wb'))
```