Exercise - Ensemble

In this exercise, we will focus on underage drinking. The data set contains data about high school students. Each row represents a single student. The columns include the characteristics of deidentified students. This is a binary classification task: predict whether a student drinks alcohol or not (this is the **alc** column: 1=Yes, 0=No). This is an important prediction task to detect underage drinking and deploy intervention techniques.

Description of Variables

The description of variables are provided in "Alcohol - Data Dictionary.docx"

Goal

Use the **alcohol.csv** data set and build a model to predict **alc**.

Read and Prepare the Data

In [188]:
```
# Common imports

import pandas as pd
import numpy as np

np.random.seed(42)
```

Get the data

In [189]:
```
#We will predict the "price" value in the data set:

alcohol = pd.read_csv("alcohol.csv")
alcohol.head()
```

Out[189]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	health	absences
0	18	2	1	4	2	0	5	4	2	5	2
1	18	4	3	1	0	0	4	4	2	3	9
2	15	4	3	2	3	0	5	3	4	5	0
3	15	3	3	1	4	0	4	3	3	3	10
4	17	3	2	1	2	0	5	3	5	5	2

Split data (train/test)

In [190]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(alcohol, test_size=0.3)
```

Data Prep

In [215]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.preprocessing import FunctionTransformer
```

Separate the target variable

In [216]:
```
train_y = train['alc']
test_y = test['alc']

train_inputs = train.drop(['alc'], axis=1)
test_inputs = test.drop(['alc'], axis=1)
```

Feature Engineering: Derive a new column

Examples:

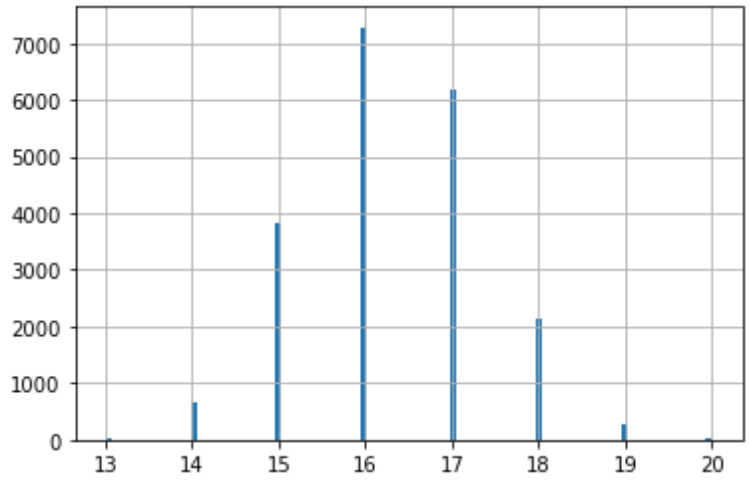
- Ratio of study time to travel time
- Student is younger than 18 or not
- Average of father's and mother's level of education
- (etc.)

In [217]:
```
train_inputs['age'].describe()
```

Out[217]:
```
count    20400.000000
mean        16.301765
std          1.047393
min         13.000000
25%         16.000000
50%         16.000000
75%         17.000000
max         20.000000
Name: age, dtype: float64
```

In [218]:
```
train_inputs['age'].hist(bins=100)
```

Out[218]: <AxesSubplot:>



In [219]:
```
def new_col(df):

    #Create a copy so that we don't overwrite the existing dataframe
    df1 = df.copy()

    # Use the formula, though fill in 0s when the value is 0/0 (because 0/0 g
    df1['ave'] = ((df1['Medu'] + df1['Fedu'])/2).fillna(0)

    # Replace the infinity values with 1 (because a value divided by 0 generat
    df1['ave'].replace(np.inf, 1, inplace=True)

    return df1[['ave']]
    # You can use this to check whether the calculation is made correctly:
    #return df1
```

In [220]:
```
new_col(train_set)
```

Out[220]:

	ave
12759	4.5
4374	1.5
8561	3.5
10697	4.0
19424	5.0
...	...
16850	1.5
6265	2.0
11284	0.0
860	2.0
15795	1.5

23800 rows × 1 columns

Identify the numeric, binary, and categorical columns

In [221]:
```
# Identify the numerical columns
numeric_columns = train_inputs.select_dtypes(include=[np.number]).columns.to_

# Identify the categorical columns
categorical_columns = train_inputs.select_dtypes('object').columns.to_list()
```

In [222]:
```
numeric_columns
```

Out[222]:
```
['age',
 'Medu',
 'Fedu',
 'traveltime',
 'studytime',
 'failures',
 'famrel',
 'freetime',
 'goout',
 'health',
 'absences']
```

In [223]:
```
categorical_columns
```

Out[223]:
```
['gender']
```

In [224]:
```
feat_eng_columns = ['Medu', 'Fedu']
```

Pipeline

In [225]:
```
numeric_transformer = Pipeline(steps=[
                ('imputer', SimpleImputer(strategy='median')),
                ('scaler', StandardScaler())])
```

In [226]:
```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value=0)),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])
```

In [227]:
```
# Create a pipeline for the transformed column here

my_new = Pipeline(steps=[('my_new', FunctionTransformer(new_col)),
                                ('scaler', StandardScaler())])
```

In [231]:
```
preprocessor = ColumnTransformer([
        ('num', numeric_transformer, numeric_columns),
        ('cat', categorical_transformer, categorical_columns),

        ('trans', my_new, feat_eng_columns)],
        remainder='drop')

#passtrough is an optional step. You don't have to use it.
```

Transform: fit_transform() for TRAIN

In [232]:
```
#Fit and transform the train data
train_x = preprocessor.fit_transform(train_inputs)

train_x
```

Out[232]:
```
array([[-0.28811749, -0.30630919, -0.82135848, ...,  1.        ,
         0.        , -0.56468129],
       [ 0.66665783, -0.93973112,  0.90150543, ...,  1.        ,
         0.        , -0.17173893],
       [-1.24289281,  0.32711275,  0.90150543, ...,  1.        ,
         0.        ,  0.6141458 ],
       ...,
       [ 0.66665783, -2.20657499, -2.54422238, ...,  1.        ,
         0.        , -2.52939313],
       [ 1.62143314, -0.30630919, -1.68279043, ...,  0.        ,
         1.        , -0.95762366],
       [ 1.62143314, -0.30630919, -2.54422238, ...,  0.        ,
         1.        , -1.35056603]])
```

In [233]:
```
train_x.shape
```

Out[233]: (20400, 14)

Tranform: transform() for TEST

In [234]:
```
# Transform the test data
test_x = preprocessor.transform(test_inputs)

test_x
```

Out[234]:
```
array([[-1.24289281,  0.32711275,  1.76293738, ...,  1.        ,
         0.        ,  1.00708817],
       [-1.24289281, -0.30630919,  0.04007348, ...,  0.        ,
         1.        , -0.17173893],
       [-0.28811749,  0.32711275,  0.04007348, ...,  0.        ,
         1.        ,  0.22120344],
       ...,
       [-0.28811749, -1.57315306,  0.04007348, ...,  0.        ,
         1.        , -0.95762366],
       [ 0.66665783, -2.20657499, -1.68279043, ...,  0.        ,
         1.        , -2.13645076],
       [ 1.62143314, -0.30630919, -1.68279043, ...,  0.        ,
         1.        , -0.95762366]])
```

In [235]:
```
test_x.shape
```

Out[235]: (13600, 14)

Calculate the Baseline

In [236]:
```
from sklearn.dummy import DummyClassifier

dummy_clf = DummyClassifier(strategy="most_frequent")

dummy_clf.fit(train_x, train_y)
```

Out[236]:
```
DummyClassifier(strategy='most_frequent')
```

In [237]:
```
from sklearn.metrics import accuracy_score
```

In [238]:
```
#Baseline Train Accuracy
dummy_train_pred = dummy_clf.predict(train_x)

baseline_train_acc = accuracy_score(train_y, dummy_train_pred)

print('Baseline Train Accuracy: {}' .format(baseline_train_acc))
```

```
Baseline Train Accuracy: 0.5240686274509804
```

In [239]:
```
#Baseline Test Accuracy
dummy_test_pred = dummy_clf.predict(test_x)

baseline_test_acc = accuracy_score(test_y, dummy_test_pred)

print('Baseline Test Accuracy: {}' .format(baseline_test_acc))
```

```
Baseline Test Accuracy: 0.5195588235294117
```

Train a voting classifier

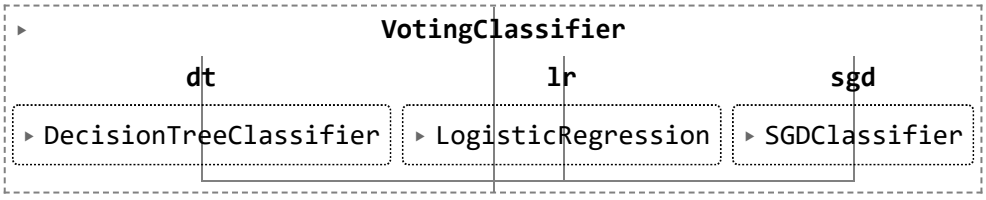
In [240]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier


dtree_clf = DecisionTreeClassifier(max_depth=30)
log_clf = LogisticRegression(multi_class='multinomial', solver = 'lbfgs', C=1
sgd_clf = SGDClassifier(max_iter=10000, tol=1e-3)

voting_clf = VotingClassifier(
            estimators=[('dt', dtree_clf),
                        ('lr', log_clf),
                        ('sgd', sgd_clf)],
            voting='hard')

voting_clf.fit(train_x, train_y)
```

Out[240]:



In [241]:
```
from sklearn.metrics import accuracy_score
```

In [242]:
```
#Train accuracy

train_y_pred = voting_clf.predict(train_x)

train_acc = accuracy_score(train_y, train_y_pred)

print('Train acc: {}' .format(train_acc))
```

```
Train acc: 0.847843137254902
```

In [243]:
```
#Test accuracy

test_y_pred = voting_clf.predict(test_x)

test_acc = accuracy_score(test_y, test_y_pred)

print('Test acc: {}' .format(test_acc))
```

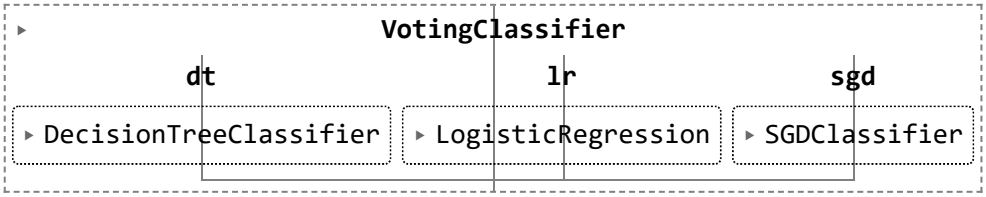
```
Test acc: 0.8191911764705883
```

In [251]:
```
dtree_clf = DecisionTreeClassifier(max_depth=20)
log_clf = LogisticRegression(multi_class='multinomial', solver = 'lbfgs', C=1
sgd_clf = SGDClassifier(max_iter=10000, tol=1e-3)

voting_clf = VotingClassifier(
            estimators=[('dt', dtree_clf),
                        ('lr', log_clf),
                        ('sgd', sgd_clf)],
            voting='hard',
            weights=[0.5, 0.25, 0.25])

voting_clf.fit(train_x, train_y)
```

Out[251]:



In [252]:
```
#Train accuracy

train_y_pred = voting_clf.predict(train_x)

train_acc = accuracy_score(train_y, train_y_pred)

print('Train acc: {}' .format(train_acc))
```

```
Train acc: 0.908921568627451
```

In [253]:
```
#Test accuracy

test_y_pred = voting_clf.predict(test_x)

test_acc = accuracy_score(test_y, test_y_pred)

print('Test acc: {}' .format(test_acc))
```

```
Test acc: 0.7868382352941177
```

Train a bagging classifier

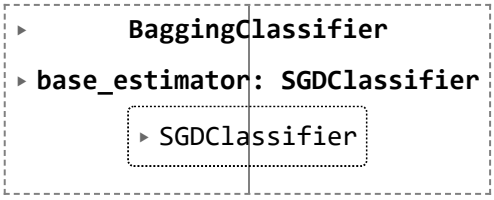
In [254]:
```
from sklearn.ensemble import BaggingClassifier


#If you want to do pasting, change "bootstrap=False"
#n_jobs=-1 means use all CPU cores
#bagging automatically performs soft voting

bag_clf = BaggingClassifier(
            SGDClassifier(), n_estimators=100,
            max_samples=1000, bootstrap=True, n_jobs=-1)

bag_clf.fit(train_x, train_y)
```

Out[254]:



In [255]:
```

train_y_pred = bag_clf.predict(train_x)

train_acc = accuracy_score(train_y, train_y_pred)

print('Train acc: {}' .format(train_acc))
```

```
Train acc: 0.8215686274509804
```

In [256]:
```
#Test accuracy

test_y_pred = bag_clf.predict(test_x)

test_acc = accuracy_score(test_y, test_y_pred)

print('Test acc: {}' .format(test_acc))
```

```
Test acc: 0.8196323529411764
```

Train a random forest classifier

In [257]:
```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=1000, max_depth=15, n_jobs=-1)

rnd_clf.fit(train_x, train_y)
```

Out[257]:
```
RandomForestClassifier(max_depth=15, n_estimators=1000, n_jobs=-1)
```

In [258]:
```
#Train accuracy

train_y_pred = rnd_clf.predict(train_x)

train_acc = accuracy_score(train_y, train_y_pred)

print('Train acc: {}' .format(train_acc))
```

```
Train acc: 0.9312745098039216
```

In [259]:
```
#Test accuracy

test_y_pred = rnd_clf.predict(test_x)

test_acc = accuracy_score(test_y, test_y_pred)

print('Test acc: {}' .format(test_acc))
```

```
Test acc: 0.8152205882352941
```

In [260]:
```
rnd_clf.feature_importances_
```

Out[260]:
```
array([0.08939691, 0.14837393, 0.05420347, 0.08227138, 0.15398757,
       0.00337793, 0.05872083, 0.03952672, 0.05564999, 0.05959961,
       0.087997  , 0.02044607, 0.02047596, 0.12597265])
```

In [261]:
```
np.round(rnd_clf.feature_importances_,2)
```

Out[261]:
```
array([0.09, 0.15, 0.05, 0.08, 0.15, 0.  , 0.06, 0.04, 0.06, 0.06, 0.09,
       0.02, 0.02, 0.13])
```

In [262]:
```
from sklearn.ensemble import ExtraTreesClassifier

ext_clf = ExtraTreesClassifier(n_estimators=1000, max_depth=10, n_jobs=-1)

ext_clf.fit(train_x, train_y)
```

Out[262]:
```
ExtraTreesClassifier(max_depth=10, n_estimators=1000, n_jobs=-1)
```

In [263]:
```
#Train accuracy

train_y_pred = ext_clf.predict(train_x)

train_acc = accuracy_score(train_y, train_y_pred)

print('Train acc: {}' .format(train_acc))
```

```
Train acc: 0.8111274509803922
```

In [264]:
```
#Test accuracy

test_y_pred = ext_clf.predict(test_x)

test_acc = accuracy_score(test_y, test_y_pred)

print('Test acc: {}' .format(test_acc))
```

```
Test acc: 0.7918382352941177
```

Train an adaboost classifier

In [265]:
```
from sklearn.ensemble import AdaBoostClassifier


ada_clf = AdaBoostClassifier(
            DecisionTreeClassifier(max_depth=10), n_estimators=1000,
            learning_rate=0.2)


ada_clf.fit(train_x, train_y)
```

Out[265]:
```
AdaBoostClassifier
base_estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

In [266]:
```
#Train accuracy

train_y_pred = ada_clf.predict(train_x)

train_acc = accuracy_score(train_y, train_y_pred)

print('Train acc: {}' .format(train_acc))
```

```
Train acc: 0.9844607843137255
```

In [267]:
```
#Test accuracy

test_y_pred = ada_clf.predict(test_x)

test_acc = accuracy_score(test_y, test_y_pred)

print('Test acc: {}' .format(test_acc))
```

```
Test acc: 0.7879411764705883
```

Train a gradient boosting classifier

In [268]:
```
#Use GradientBoosting

from sklearn.ensemble import GradientBoostingClassifier

gbclf = GradientBoostingClassifier(max_depth=2, n_estimators=100, learning_ra

gbclf.fit(train_x, train_y)
```

Out[268]:
```
GradientBoostingClassifier
GradientBoostingClassifier(max_depth=2)
```

In [269]:
```
#Train accuracy

train_y_pred = gbclf.predict(train_x)

train_acc = accuracy_score(train_y, train_y_pred)

print('Train acc: {}' .format(train_acc))
```

```
Train acc: 0.8192647058823529
```

In [270]:
```
#Test accuracy

test_y_pred = gbclf.predict(test_x)

test_acc = accuracy_score(test_y, test_y_pred)

print('Test acc: {}' .format(test_acc))
```

```
Test acc: 0.8119852941176471
```

In [271]:
```
#Train on 75% of the sample only

gbclf = GradientBoostingClassifier(max_depth=2, n_estimators=100,
                                   learning_rate=0.1, subsample=0.75)

gbclf.fit(train_x, train_y)
```

Out[271]:
```
GradientBoostingClassifier

GradientBoostingClassifier(max_depth=2, subsample=0.75)
```

In [272]:
```
#Train accuracy

train_y_pred = gbclf.predict(train_x)

train_acc = accuracy_score(train_y, train_y_pred)

print('Train acc: {}' .format(train_acc))
```

```
Train acc: 0.8188235294117647
```

In [273]:
```
#Test accuracy

test_y_pred = gbclf.predict(test_x)

test_acc = accuracy_score(test_y, test_y_pred)

print('Test acc: {}' .format(test_acc))
```

```
Test acc: 0.8120588235294117
```

In [ ]:
```

```