Data Mining project:

Predicting Gender of Students Based on Exams Scores.

by:

Aya Hamrouni

Motaz Saidani

Noura hssini

Safa trabelsi

Zied Ferah

Outline :

Description of the dataset :

This dataset contains marks secured by high school students in the US. Our goal is to predict the gender of the students from their exam scores in certain fields.

Variables:

- **Race/ethnicity** : 5 unique races/ethnicities in this table
- **Parental level of education:** The education background of the parents, in what level they finished their studies ? (6 Unique values)
- **lunch** : Do students pay normally for their lunch ? or they have a free/reduced lunch ? (2 unique values)
- **test preparation course** : did the student prepared for the tests or not ? (2 unique values)

- **Math percentage** : the math exam score / 100
- **Reading score percentage** : the reading score / 100
- **Writing score percentage** : the writing score / 100
- **gender** : the gender of the student (Male or Female)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Loading Data:

```
!gdown --id 1uYvHLuPHMz7cH-i3cyugzTgP_5PKQ7IW
```

```
Downloading...
From: https://drive.google.com/uc?id=1uYvHLuPHMz7cH-i3cyugzTgP_5PKQ7IW
To: /content/Student Performance new.csv
100% 62.6k/62.6k [00:00<00:00, 48.5MB/s]
```

```
data = pd.read_csv('Student Performance new.csv')
```

```
print(data.shape)
data[:3]
```

```
(1000, 9)
```

	Unnamed: 0	race/ethnicity	parental level of education	lunch	test preparation course	math percentage	reading score percentage
0	0	group B	bachelor's degree	standard	none	0.72	0.72
1	1	group C	some college	standard	completed	0.69	0.90
2	2	group B	master's	standard	none	0.90	0.95

I- Data Preprocessing

Picking the relevant variables

```
df = data.iloc[:,-4:]
```

```
df.head()
```

	math percentage	reading score percentage	writing score percentage	gender
0	0.72	0.72	0.74	F
1	0.69	0.90	0.88	F
2	0.90	0.95	0.93	F
3	0.47	0.57	0.44	M
4	0.76	0.78	0.75	M

Checking for missing values

```
df.isna().sum()
```

```
math percentage             0
reading score percentage    0
writing score percentage    0
gender                      0
dtype: int64
```

Data types:

```
df.dtypes
```

```
math percentage             float64
reading score percentage    float64
writing score percentage    float64
gender                       object
dtype: object
```

Checking for the balance of target classes:

```
df.gender.value_counts()
```

```
F    518
M    482
Name: gender, dtype: int64
```

Renaming target classes:

F : female

M : male

```
df.gender = df.gender.map(dict(F='female', M='male'))
df.head()
```

	math percentage	reading score percentage	writing score percentage	gender
0	0.72	0.72	0.74	female
1	0.69	0.90	0.88	female
2	0.90	0.95	0.93	female
3	0.47	0.57	0.44	male
4	0.76	0.78	0.75	male

Saving data:

```
df.to_csv(r'Student-Performance-clean.csv', index = False)
```

II- Data exploration

Descriptive statistics

```
df.describe()
```

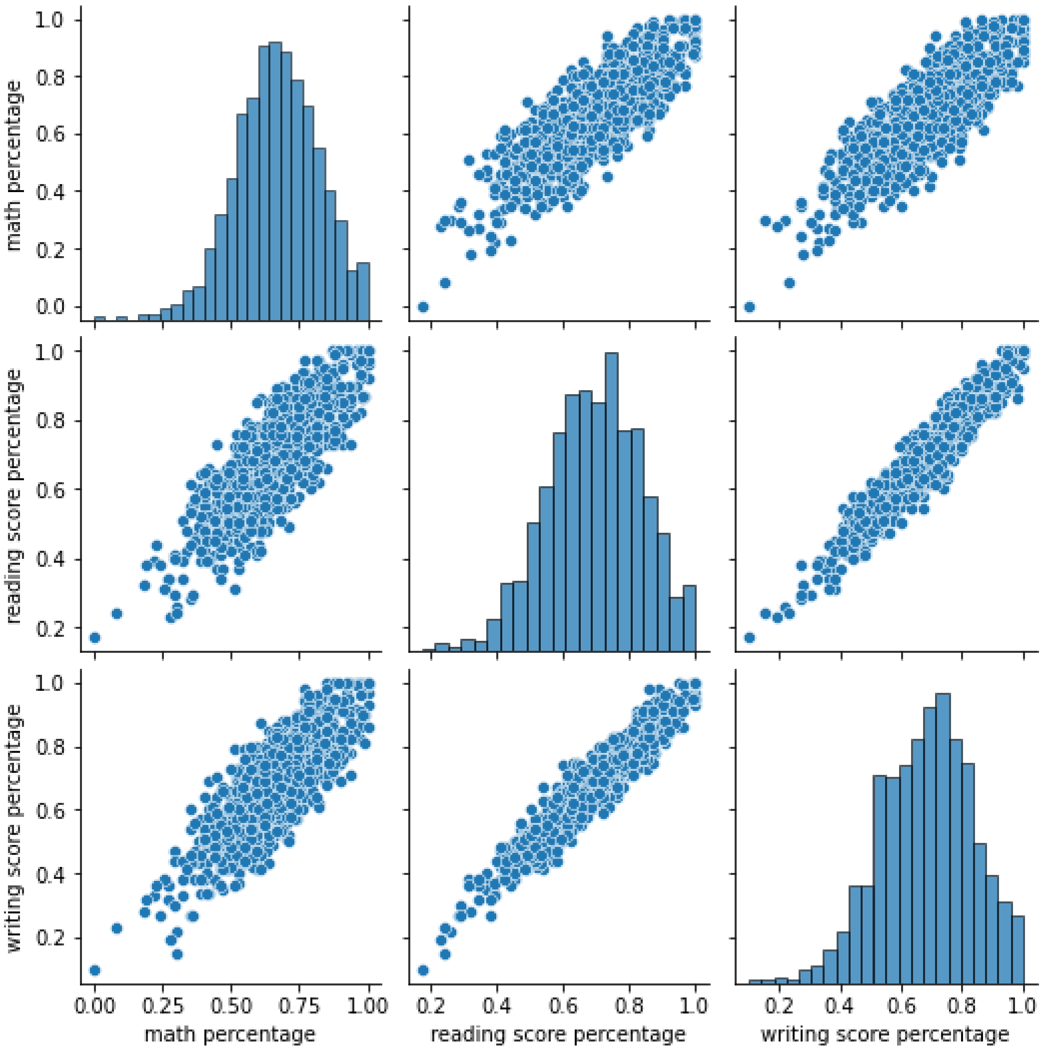
	math percentage	reading score percentage	writing score percentage
count	1000.000000	1000.000000	1000.000000
mean	0.660890	0.691690	0.680540
std	0.151631	0.146002	0.151957
min	0.000000	0.170000	0.100000
25%	0.570000	0.590000	0.577500
50%	0.660000	0.700000	0.690000
75%	0.770000	0.790000	0.790000
max	1.000000	1.000000	1.000000

```
df.corr()
```

	math percentage	reading score percentage	writing score percentage
math percentage	1.000000	0.817580	0.802642
reading score percentage	0.817580	1.000000	0.954598

Visualizing variables distributions & relationships:

```
sns.pairplot(df.drop(['gender'],axis=1))
plt.show()
```



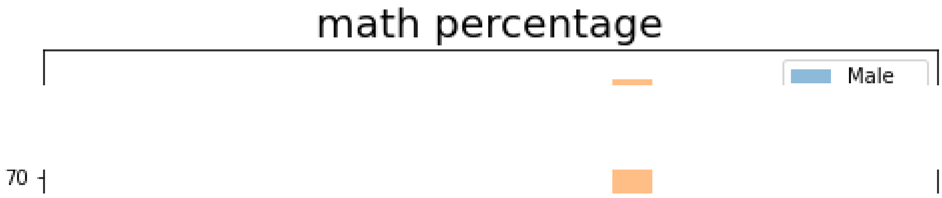
Comparing distributions of each exam scores by gender:

```
# Constants
mask_F = df.gender == 'female'
mask_M = df.gender == 'male'

female_obs = df[mask_F]
male_obs = df[mask_M]


## Plots
for i in df.columns:
  if i != 'gender':
    plt.figure(figsize=(8,6))
    plt.hist(male_obs[i]*100, bins=20, alpha=0.5, label="Male")
    plt.hist(female_obs[i]*100, bins=20, alpha=0.5, label="Female")
    plt.xlabel("Exam Score", size=14)
    plt.ylabel("Students count", size=14)
    plt.title(i, size=20)
```

```
        plt.legend(loc='upper right')
        plt.show()
        if i == 'writing score percentage':
          break
```



III- Data modeling

1) Splitting Data

```
from sklearn.model_selection import train_test_split

# Feature selection
X = df.iloc[:,:3]
y = df.iloc[:,-1]

# Splitting ( test set sample size = 20%, training set = 80%)
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=30)

print ('Train set:', X_train.shape, y_train.shape)
print ('Test set:', X_test.shape, y_test.shape)
```

```
Train set: (800, 3) (800,)
Test set: (200, 3) (200,)
```

```
## Is the sample representative ?

y_train.value_counts()
```

```
female    410
male      390
Name: gender, dtype: int64
```

2) Custom Functions for Evaluation

[ ] ↳ 3 cells hidden

3) Models:

a) *Logistic Regression*

Training

```
from sklearn.linear_model import LogisticRegression


logistic_model = LogisticRegression(C=10, solver='liblinear')
logistic_model.fit(X_train, y_train)

    LogisticRegression(C=10, solver='liblinear')
```

Evaluation

```
# yhat
yhat_logistic_test = logistic_model.predict(X_test)
yhat_prob_logistic_test = (logistic_model.predict_proba(X_test))[:,1]

# Confusion matrix
print('\n\n Logistic Regression Confusion Matrix:')
confusion_matrix_plot(logistic_model, y_test, yhat_logistic_test)

# Report
print('\n\n Report:')
logistic_eval = model_evaluation('Logistic Reg.', yhat_logistic_test)
logistic_eval
```

Logistic Regression Confusion Matrix:

Report:

	Test Accuracy	Precision	Recall	Jaccard-Score	F1-Score
Logistic Reg.	0.865	0.878505	0.87037	0.77686	0.865051

b) *K-Nearest Neighbor*

*We further split the training sets into train and validation sets to find the best **k**.*

```
X_knn_train, X_knn_val, y_knn_train, y_knn_val = train_test_split(X_train, y_train, test_s
print ('KNN Train Set:', X_knn_train.shape,  y_knn_train.shape)
print ('KNN Validation Set:', X_knn_val.shape,  y_knn_val.shape)
```

```
KNN Train Set: (640, 3) (640,)
KNN Validation Set: (160, 3) (160,)
```

*Finding the best **k***

```
from sklearn.neighbors import KNeighborsClassifier
```

```
## iterating using different 'K's
error_rate = []
for i in range(1,30):
 knn_trial = KNeighborsClassifier(n_neighbors=i)
 knn_trial.fit(X_knn_train,y_knn_train)
 knn_trial_yhat = knn_trial.predict(X_test)
 error_rate.append(np.mean(knn_trial_yhat != y_test))

## Visualizing the error
plt.figure(figsize=(10,6))
plt.plot(range(1,30),error_rate, linestyle='dashed', marker='X',markerfacecolor='red', mar
plt.xlabel('K')
plt.ylabel('Error Rate')
req_k_value = error_rate.index(min(error_rate))+1
plt.show()

print("Min-Error:",min(error_rate),"at K =",req_k_value)
```

Training the model on the whole training set

```
k=18
knn_model = KNeighborsClassifier(n_neighbors = k)
knn_model.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=18)
```

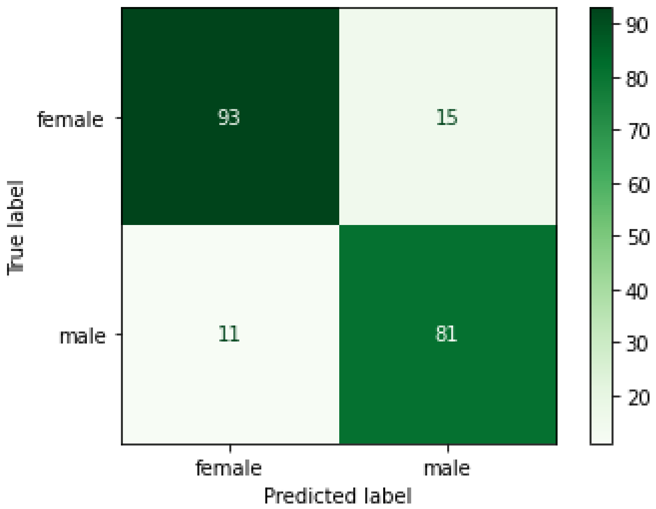
Evalution

```
# Constants
yhat_knn_test = knn_model.predict(X_test)
yhat_prob_knn_test = (knn_model.predict_proba(X_test))[:,1]

# Confusion Matrix
print('\n\n KNN Confusion Matrix')
confusion_matrix_plot(knn_model, y_test, yhat_knn_test)

# Report
print('\n\n Report:')
knn_eval = model_evaluation('KNN', yhat_knn_test)
knn_eval
```

KNN Confusion Matrix



Report:

	Test Accuracy	Precision	Recall	Jaccard-Score	F1-Score
KNN	0.87	0.894231	0.861111	0.781513	0.870157

c) *Decision Tree*

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

#Tree Visualization
import pydotplus
from IPython.display import Image, HTML
```

Training

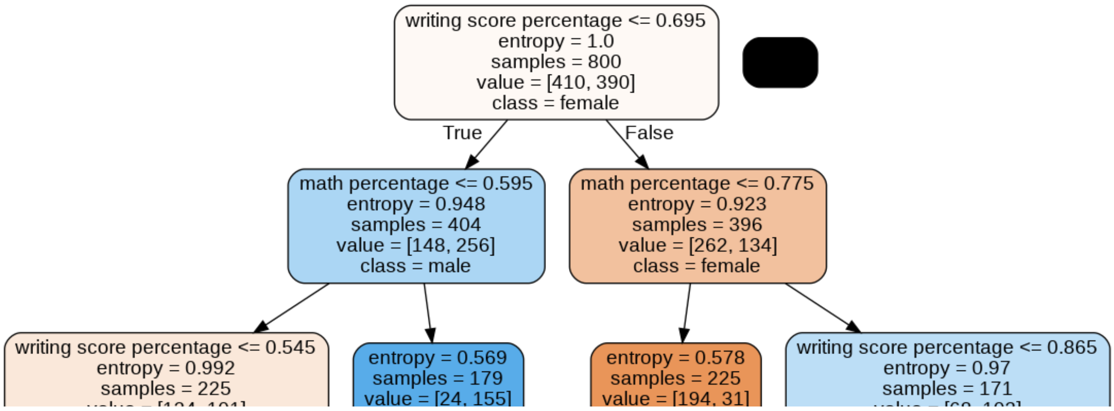
```
tree_model = DecisionTreeClassifier(criterion="entropy", max_depth=5, max_leaf_nodes=7, ra
tree_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5, max_leaf_nodes=7,
                       random_state=100)
```

```
#Creating Dot Data
dot_data = tree.export_graphviz(tree_model,
                                out_file=None,
                                feature_names=list(X.columns.values),
                                class_names=tree_model.classes_,
                                rounded=True,
                                filled=True)

#Creating Graph from DOT data
graph = pydotplus.graph_from_dot_data(dot_data)

# displaying graph
Image(graph.create_png())
```



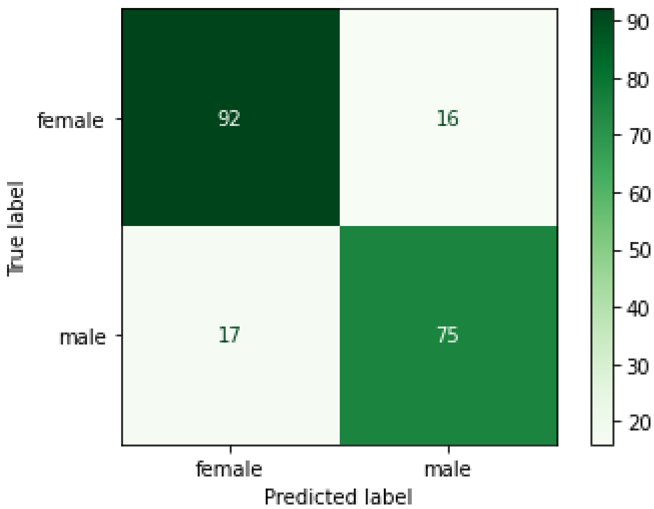
Evaluation


```
# Constants
yhat_tree_test = tree_model.predict(X_test)
yhat_prob_tree_test = (tree_model.predict_proba(X_test))[:,1]

# Confusion Matrix
print('\n\n Decision Tree Confusion Matrix')
confusion_matrix_plot(tree_model, y_test, yhat_tree_test)

# Report
print('\n\n Report:')
tree_eval = model_evaluation('Decision Tree',  yhat_tree_test)
tree_eval
```

```
Decision Tree Confusion Matrix
```




```
Report:
```

	Test Accuracy	Precision	Recall	Jaccard-Score	F1-Score
Decision Tree	0.835	0.844037	0.851852	0.736	0.834929

d) *Support Vector Machine*

```
from sklearn import svm
```

Training

```
svm_model = svm.SVC(random_state=100, probability=True, C=10)
svm_model.fit(X_train, y_train)
```

```
    SVC(C=10, probability=True, random_state=100)
```

Evaluation

```
# Constants
yhat_svm_test = svm_model.predict(X_test)
yhat_prob_svm_test = (svm_model.predict_proba(X_test))[:,1]

# Confusion Matrix
print('\n\n SVM Confusion Matrix')
confusion_matrix_plot(svm_model, y_test, yhat_svm_test)

# Report
print('\n\n Report:')
svm_eval = model_evaluation('SVM', yhat_svm_test)
svm_eval
```

SVM Confusion Matrix

Report:

	Test Accuracy	Precision	Recall	Jaccard-Score	F1-Score
SVM	0.875	0.895238	0.87037	0.789916	0.875122

IV- Final Report

1) Generating Evaluation table & ROC-AUC graph:

[ ] ↳ *7 cells hidden*

2) Displaying Final Report:

```
final_report()
```

Evaluation of All Algorithms

	Test Accuracy	Precision	Recall	Jaccard-Score	F1-Score
Logistic Reg.	86.50%	87.85%	87.04%	77.69%	86.51%
KNN	87.00%	89.42%	86.11%	78.15%	87.02%
Decision Tree	83.50%	84.40%	85.19%	73.60%	83.49%
SVM	87.50%	89.52%	87.04%	78.99%	87.51%