Student Performance Analysis

Cet ensemble de données comprend une base des étudiants, d'un exemple de lycée au États-Unis qui comprends des résultats de trois examens et une variété de facteurs personnels, sociaux et économiques qui ont des effets d'interaction sur eux.

Table des matières

1. Importer les librairies

Entrée [1]:

```
#Importer les librairies
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import matplotlib as mpl
from sklearn import metrics
from sklearn.metrics import *
import matplotlib.gridspec as gridspec
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
%matplotlib inline
```

2. Lire le Dataset Student

Entrée [2]:

```
student = pd.read_csv(r"../Downloads/StudentsPerformance.csv")
studentregli =  pd.read_csv(r"../Downloads/StudentsPerformance.csv")
student.head()
```

Out[2]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

Entrée [3]:
```
student.describe()
```

Out[3]:

	math score	reading score	writing score
count	1000.00000	1000.000000	1000.000000
mean	66.08900	69.169000	68.054000
std	15.16308	14.600192	15.195657
min	0.00000	17.000000	10.000000
25%	57.00000	59.000000	57.750000
50%	66.00000	70.000000	69.000000
75%	77.00000	79.000000	79.000000
max	100.00000	100.000000	100.000000

La moyenne des scores de math est de 66 de l'oral 68 et de l'ecriture est de 69. le minimum est de 0 le maximum est de 100%

Entrée [4]:
```
student.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64 
 6   reading score                1000 non-null   int64 
 7   writing score                1000 non-null   int64 
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

Entrée [5]:
```
#Pas de variable manquante
student.isnull().sum()
```

Out[5]:
```
gender                         0
race/ethnicity                 0
parental level of education    0
lunch                          0
test preparation course        0
math score                     0
reading score                  0
writing score                  0
dtype: int64
```

Entrée [6]:
```
student.tail()
```

Out[6]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
995	female	group E	master's degree	standard	completed	88	99	95
996	male	group C	high school	free/reduced	none	62	55	55
997	female	group C	high school	free/reduced	completed	59	71	65
998	female	group D	some college	standard	completed	68	78	77
999	female	group D	some college	free/reduced	none	77	86	86

Entrée [7]:
```
student.shape
```

Out[7]:
```
(1000, 8)
```

Le jeu de données comporte 1000 observations et 8 colonnes qui représentent :

- Le Genre
- Le niveau d'éducation des parents
- Le test de preparation des cours
- Le score des math
- Le score de l'oral de lecture
- Le score de l'écriture

Statistiques descriptives

Entrée [8]:
```
student.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
math score	1000.0	66.089	15.163080	0.0	57.00	66.0	77.0	100.0
reading score	1000.0	69.169	14.600192	17.0	59.00	70.0	79.0	100.0
writing score	1000.0	68.054	15.195657	10.0	57.75	69.0	79.0	100.0

Classement des variables par Group

Entrée [9]:
```
#Groupe la moyenne des 'race/ethnicity' par 'parental level of education'
student.groupby(['race/ethnicity','parental level of education']).mean()
```

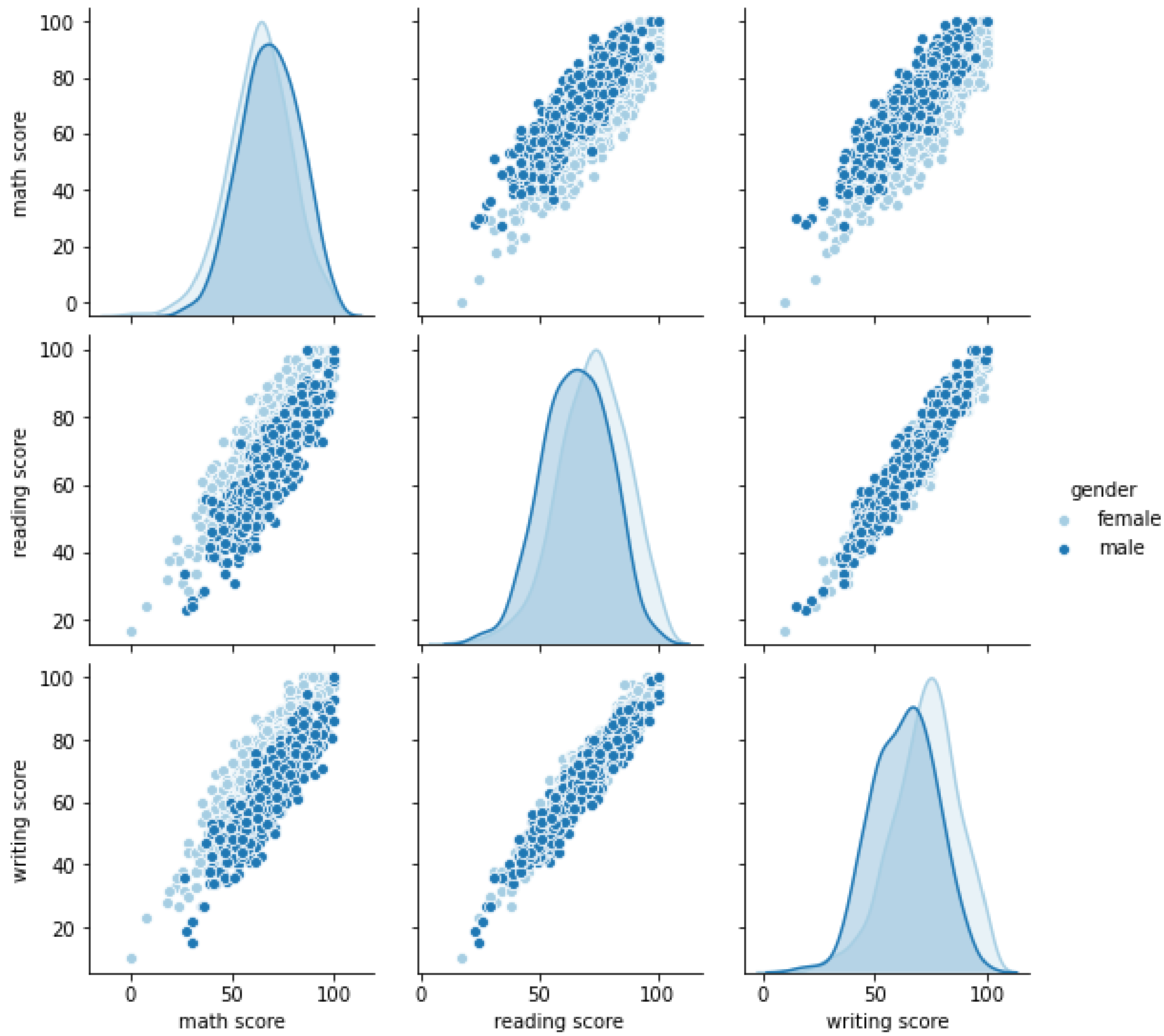
Out[9]:

race/ethnicity	parental level of education	math score	reading score	writing score
group A	associate's degree	61.000000	67.071429	63.571429
	bachelor's degree	67.166667	68.083333	68.333333
	high school	60.444444	62.888889	60.500000
	master's degree	57.666667	64.666667	67.666667
	some college	63.888889	65.777778	65.000000
	some high school	58.916667	62.083333	58.583333
group B	associate's degree	66.097561	69.585366	68.243902
	bachelor's degree	69.300000	72.950000	71.650000
	high school	59.791667	63.458333	61.250000
	master's degree	67.166667	80.166667	77.166667
	some college	63.189189	65.756757	64.189189
	some high school	61.815789	66.447368	64.605263
group C	associate's degree	66.730769	71.128205	70.269231
	bachelor's degree	68.150000	75.675000	75.900000
	high school	60.906250	64.421875	61.656250
	master's degree	67.052632	70.526316	69.526316
	some college	65.130435	69.420290	68.869565
	some high school	60.551020	65.632653	63.285714
group D	associate's degree	67.600000	70.540000	69.860000
	bachelor's degree	67.571429	70.142857	71.892857
	high school	62.863636	64.409091	63.159091
	master's degree	72.521739	77.173913	79.739130
	some college	68.731343	70.880597	71.701493
	some high school	66.760000	69.980000	69.100000
group E	associate's degree	74.897436	73.820513	73.205128
	bachelor's degree	76.555556	74.833333	75.388889
	high school	70.772727	70.318182	67.545455
	master's degree	74.625000	82.125000	80.500000
	some college	73.828571	72.628571	70.200000
	some high school	72.111111	69.555556	66.555556

3.Visualisation de la base student (EDA)

Entrée [10]:
```
sns.pairplot(student,hue='gender',palette='Paired')
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x7ff97c56f850>



Entrée [11]:
```
student['total'] = (student['math score']+student['reading score']+studen
student.sample()
#course_gender = student.groupby(['gender','test preparation course']).me
#sns.factorplot(x='gender', y='total', hue='test preparation course', dat
```

Out[11]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	tota
873	male	group E	associate's degree	free/reduced	none	90	90	82	87.33333

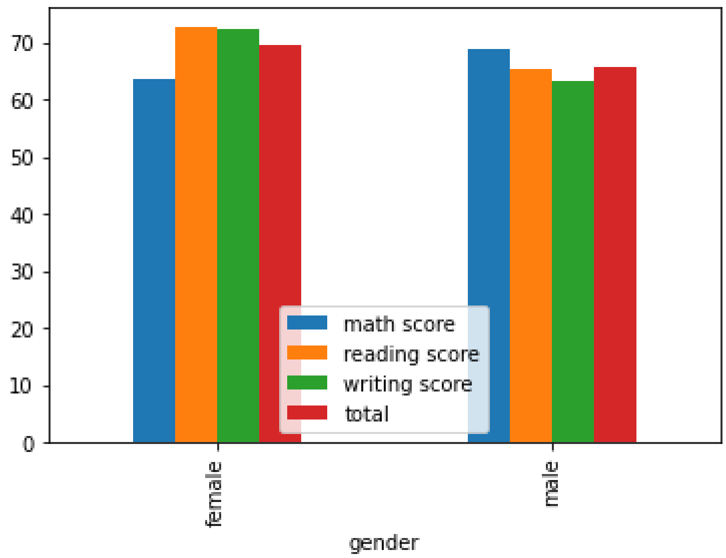
Classification des étudiants par sexe

Entrée [12]:
```
#Groupe par genre
student.groupby(['gender']).mean()
```

Out[12]:

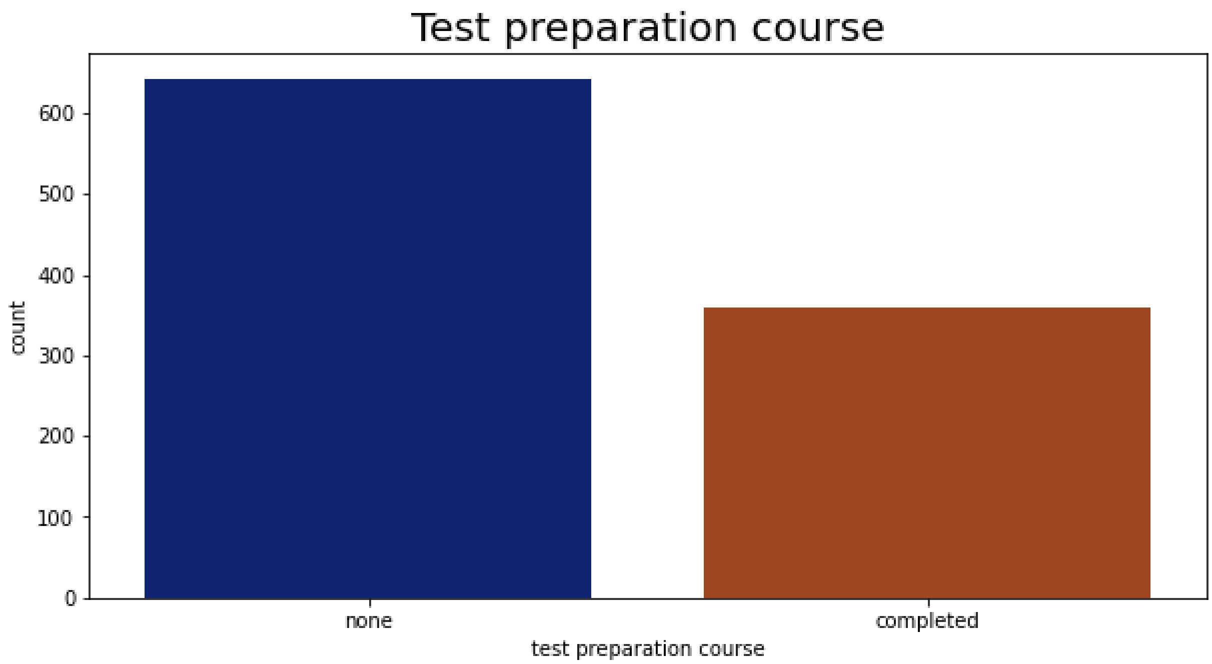
	math score	reading score	writing score	total
gender				
female	63.633205	72.608108	72.467181	69.569498
male	68.728216	65.473029	63.311203	65.837483

Entrée [13]:
```
#Genre par test preparation
student.groupby(['gender']).mean().plot.bar()
plt.show()
```



par Test Preparation Course

Entrée [14]:
```
plt.rcParams['figure.figsize'] = (10, 5)
sns.countplot(student['test preparation course'], palette = 'dark')
plt.title('Test preparation course',fontsize = 20)
plt.show()
```

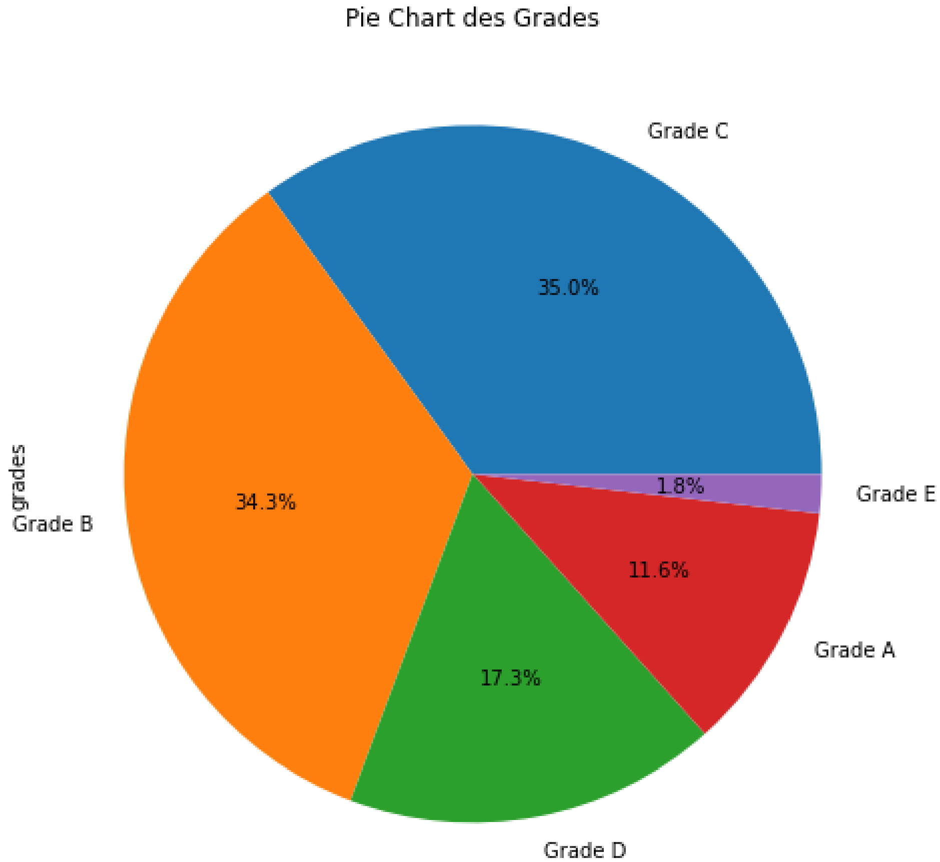


Visualisation des Grades par le total des scores

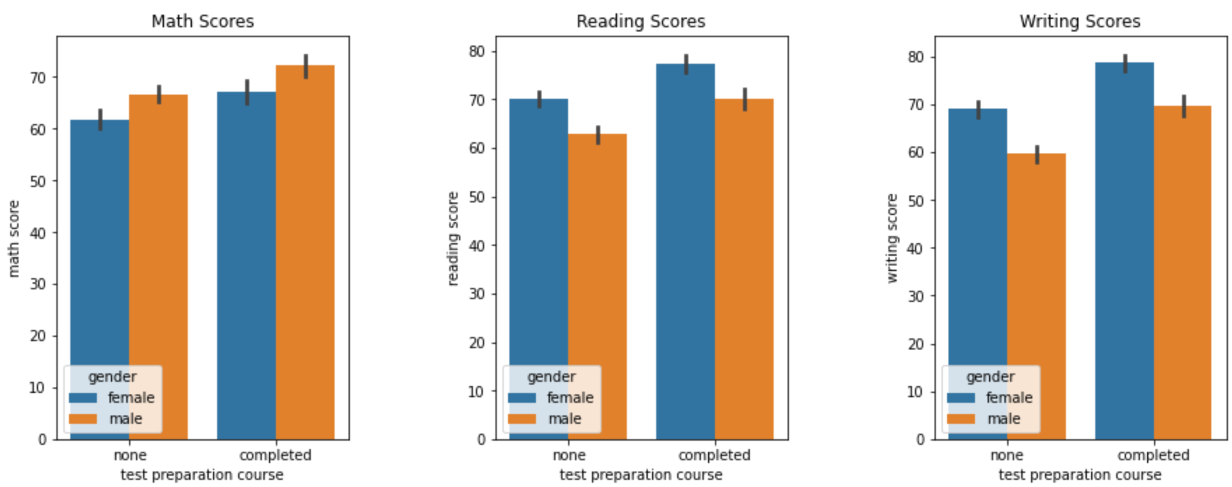
Entrée [15]:
```
#Crée une table Total des Grade de A à E
student['Total'] = student['math score']+student['reading score']+student
student['percentage']=student['Total']/300*100
```

Entrée [16]:
```
def determine_grade(Total):
    if Total >= 85 and Total <= 100: return 'Grade A'
    elif Total >= 70 and Total< 85:  return 'Grade B'
    elif Total >= 55 and Total < 70: return 'Grade C'
    elif Total >= 35 and Total < 55: return 'Grade D'
    elif Total >= 0 and Total < 35:  return 'Grade E'

student['grades']=student['percentage'].apply(determine_grade)
```

Entrée [17]:
```
plt.figure(figsize=(15,8))
student['grades'].value_counts().plot.pie(autopct="%1.1f%%")
plt.title("Pie Chart des Grades")
plt.show()
```



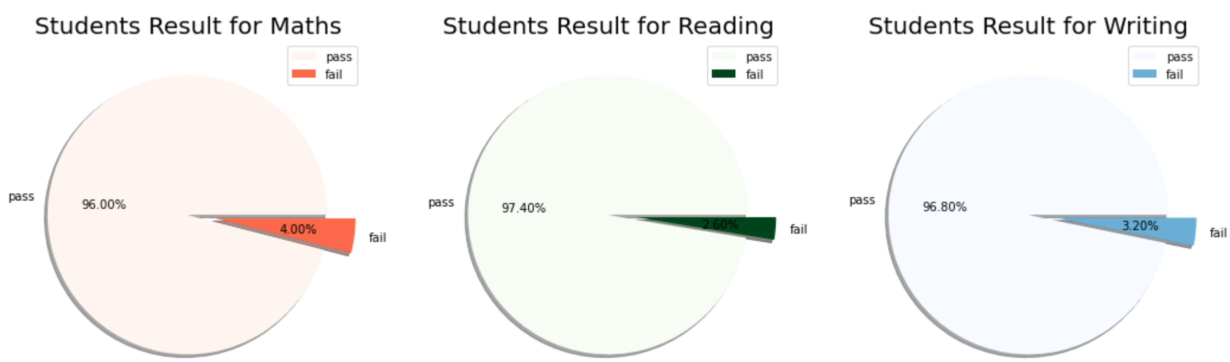
Entrée [18]:
```
plt.figure(figsize=(15,5))
plt.subplots_adjust(left=0.125, bottom=0.1, right=0.9, top=0.9,
                      wspace=0.5, hspace=0.2)
plt.subplot(131)
plt.title('Math Scores')
sns.barplot(hue="gender", y="math score", x="test preparation course", da
plt.subplot(132)
plt.title('Reading Scores')
sns.barplot(hue="gender", y="reading score", x="test preparation course",
plt.subplot(133)
plt.title('Writing Scores')
sns.barplot(hue="gender", y="writing score", x="test preparation course",
plt.show()
```



Classification des étudiants selon qu'ils ont réussi ou pas

Entrée [19]:
```
# Diagramme à secteurs pour représenter le ratio de réussite et d'échec e
passmarks = 40
plt.rcParams['figure.figsize'] = (18, 12)

# creating a new column pass_math, this column will tell us whether the s
student['pass_math'] = np.where(student['math score']< passmarks, 'Fail',
student['pass_reading'] = np.where(student['reading score']< passmarks, '
student['pass_writing'] = np.where(student['writing score']< passmarks, '


size = student['pass_math'].value_counts()
colors = plt.cm.Reds(np.linspace(0, 1, 3))
labels = "pass", "fail"
explode = [0, 0.2]
```

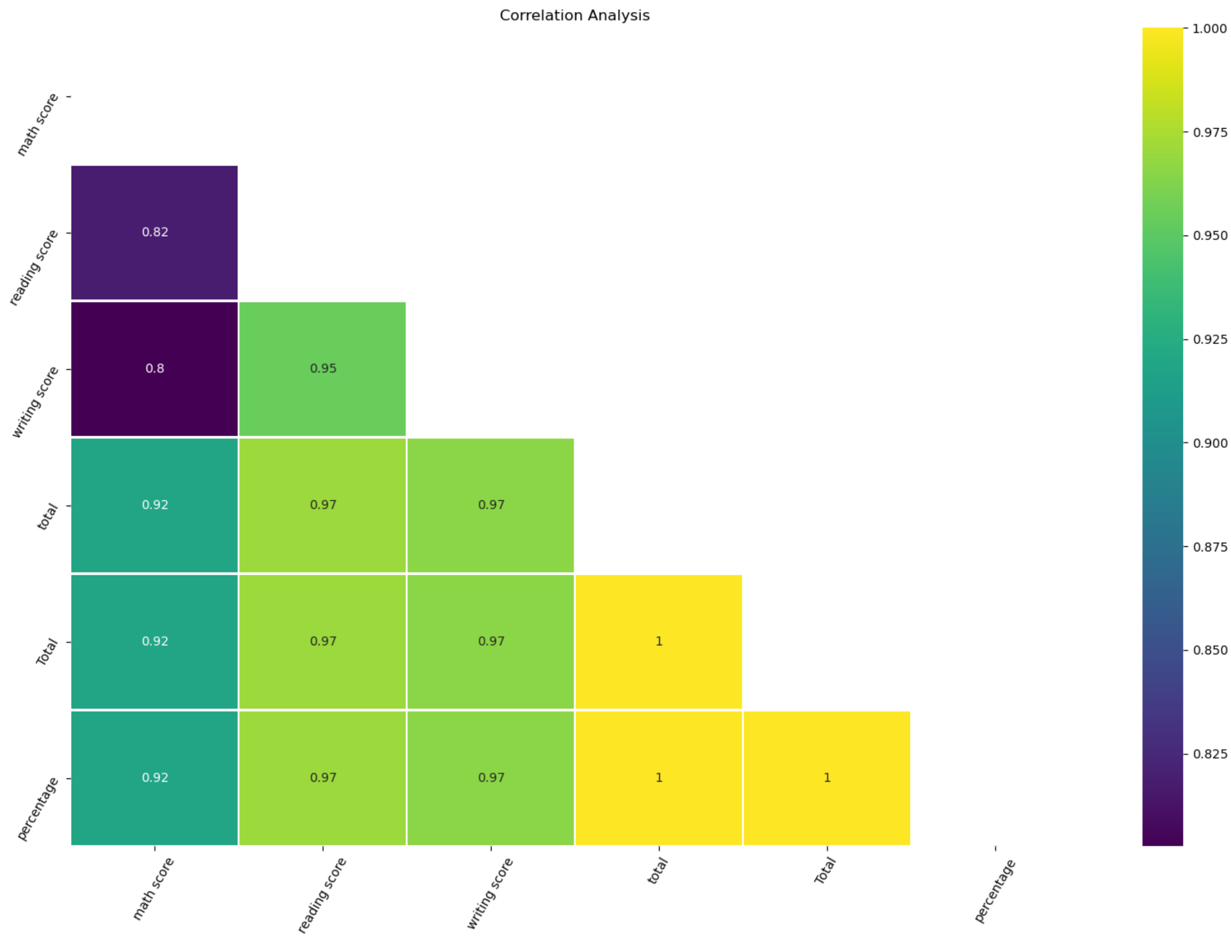
Entrée [20]:
```
plt.subplot(1, 3, 1)
plt.pie(size, colors = colors, labels = labels, autopct = '%.2f%%', explo
plt.title('Students Result for Maths', fontsize = 20)
plt.legend()

size = student['pass_reading'].value_counts()
colors = plt.cm.Greens(np.linspace(0, 1, 2))
labels = "pass", "fail"
explode = [0, 0.2]

plt.subplot(1, 3, 2)
plt.pie(size, colors = colors, labels = labels, autopct = '%.2f%%', explo
plt.title('Students Result for Reading', fontsize = 20)
plt.legend()

size = student['pass_writing'].value_counts()
colors = plt.cm.Blues(np.linspace(0, 1, 3))
labels = "pass", "fail"
explode = [0, 0.2]

plt.subplot(1, 3, 3)
plt.pie(size, colors = colors, labels = labels, autopct = '%.2f%%', explo
plt.title('Students Result for Writing', fontsize = 20)
plt.legend()

plt.show()
```



Correlation entre les Variables

Entrée [21]:
```
plt.figure(dpi=100)
plt.title('Correlation Analysis')
sns.heatmap(student.corr(),annot=True,lw=1,linecolor='white',cmap='viridi
plt.xticks(rotation=60)
plt.yticks(rotation = 60)
plt.show()
```

Entrée [22]:
```
corr = student.corr()
mask = np.triu(np.ones_like(corr,dtype = bool))

plt.figure(dpi=100)
plt.title('Correlation Analysis')
sns.heatmap(student.corr(),mask=mask,annot=True,lw=1,linecolor='white',cm
plt.xticks(rotation=60)
plt.yticks(rotation = 60)
plt.show()
```



4. Descente de Gradient from scratch

En se référant à la matrice de corrélation et au heatmap ci-dessus, il semblerait que la variable explicative 'math score' soit la plus corrélée au total des 3 scores, et donc serait aussi la plus corrélée à la moyenne des 3 scores. On va utiliser cette variable 'math score' comme variable explicative, avec les autres variables ordinales, pour tenter de prédire la moyenne des scores de chaque élève.

Entrée [23]:
```
# ajouter la colonne 'mean score'
studentregli['mean score'] = studentregli[['math score','reading score','
```

Entrée [24]:
```
# déclarer les variables catégorielles
studentregli[['gender',
              'race/ethnicity',
              'parental level of education',
              'lunch',
              'test preparation course']] = studentregli[['gender',
                                                          'race/ethnici
                                                          'parental lev
                                                          'lunch',
                                                          'test prepara
```

Entrée [25]:
```
studentregli['gender'] = studentregli['gender'].cat.codes
studentregli['race/ethnicity'] = studentregli['race/ethnicity'].cat.codes
studentregli['parental level of education'] = studentregli['parental leve
studentregli['lunch'] = studentregli['lunch'].cat.codes
studentregli['test preparation course'] = studentregli['test preparation
```

Entrée [26]:
```
# définir la variable à prédire et les variables explicatives
y = studentregli['mean score'].values.reshape(-1,1)
x = studentregli[['gender','race/ethnicity','parental level of education'
                  'lunch','test preparation course','math score']].value
```

Entrée [27]:
```
x.shape
```

Out[27]: (1000, 6)

Entrée [28]:
```
y.shape
```

Out[28]: (1000, 1)

Entrée [29]:
```
# ajouter une colonne de bias à la matrice des x_train
x_b = np.concatenate((np.ones((len(x),1)),x),axis=1)
```

Entrée [30]:
```
# préparer les jeux train et test
x_train, x_test, y_train, y_test = train_test_split(x_b, y, test_size=0.3
```

Entrée [31]:
```
x_train
```

Out[31]:
```
array([[ 1.,  0.,  2., ...,  1.,  1., 58.],
       [ 1.,  1.,  1., ...,  0.,  1., 61.],
       [ 1.,  1.,  4., ...,  1.,  1., 76.],
       ...,
       [ 1.,  0.,  2., ...,  1.,  0., 44.],
       [ 1.,  1.,  3., ...,  1.,  1., 73.],
       [ 1.,  1.,  1., ...,  1.,  0., 62.]])
```

Entrée [32]:
```
x_train.shape
```

Out[32]: (700, 7)

Entrée [33]:
```
y_train.shape
```

Out[33]: (700, 1)

Entrée [34]:
```
x_test.shape
```

Out[34]: (300, 7)

Entrée [35]:
```
# estimer l'intercept et les coefficients de régression par un calcul mat
```

Entrée [36]:
```
a = (np.linalg.inv(x_train.T@x_train))@(x_train.T@y_train)
```

Entrée [37]:
```
def predict(features, weights):

#    features - (1000, 7)
#    weights - (7, 1)
#    --> predictions - (1000,7)

    predictions = np.dot(features, weights)
    predictions = np.array(predictions).reshape(-1,1)

    return predictions
```

Entrée [38]:
```
y_test_pred = predict(x_test,a)
```

Entrée [39]:
```
df = pd.DataFrame({'True' : y_test.flatten(),  'Predicted' : y_test_pred.
df['error'] = df['True'] - df['Predicted']
df['error2'] = df['error']**2
df
```

Out[39]:

	True	Predicted	error	error2
0	69.333333	67.471583	1.861751	3.466116
1	77.333333	82.539221	-5.205888	27.101271
2	45.333333	42.462532	2.870802	8.241502
3	67.666667	72.499737	-4.833070	23.358569
4	74.333333	72.768770	1.564563	2.447857
...	...	...	...	...
295	70.666667	67.068548	3.598118	12.946455
296	74.333333	74.160424	0.172910	0.029898
297	77.000000	77.138696	-0.138696	0.019236
298	66.000000	68.846171	-2.846171	8.100688
299	54.666667	55.341709	-0.675043	0.455683

300 rows × 4 columns

Entrée [40]:
```
# MSE
df['error2'].mean()
```

Out[40]: 14.238556161073872

Entrée [41]:
```
# comparer avec la MSE de scikit-learn
from sklearn import metrics
metrics.mean_squared_error(y_test,y_test_pred)
```

Out[41]: 14.238556161073872

Entrée [42]:
```
# définir la fonction de coût
def cost_function(features, targets, weights):
    m = len(targets)
    J = np.sum((predict(features, weights) - targets) ** 2)/(2 * m)
    return J
```

Entrée [43]:
```
cost_function(x_train, y_train, a)
```

Out[43]: 6.009992903768611

Entrée [44]:
```
# initialiser les coefficients
init_weights = np.zeros((7,1))
init_weights = init_weights.reshape(-1,1)
```

Entrée [45]:
```
cost_function(x_train, y_train, init_weights)
```

Out[45]: 2404.346587301587

Entrée [46]:
```
# définir la fonction qui calcule le gradient

def feature_derivative(errors, features):
    derivative = 2 * np.dot(errors, features)
    return derivative
```

Entrée [47]:
```
# définir la fonction de descente de gradient et qui renvoie les nouveaux
# coût

def batch_gradient_descent(feature_matrix, output, initial_weights, step_
    converged = False
    weights = np.array(initial_weights) # make sure it's a numpy array
    for j in range(200000):
        # compute the predictions based on feature_matrix and weights usi
        predictions = predict(feature_matrix, weights)
        # compute the errors as predictions - output
        errors = predictions - output
        errors = errors.reshape(1,-1)
        gradient_sum_squares = 0 # initialize the gradient sum of squares
        # while we haven't reached the tolerance yet, update each feature
        for i in range(len(weights)): # loop over each weight
            # Recall that feature_matrix[:, i] is the feature column asso
            # compute the derivative for weight[i]:
            derivative = feature_derivative(errors, feature_matrix[:, i])
            # subtract the step size times the derivative from the curren
            weights[i]= weights[i] - (step_size * derivative)
            # add the squared value of the derivative to the gradient sum
            derivative_square = derivative * derivative
            gradient_sum_squares = derivative_square.sum()
        # compute the square-root of the gradient sum of squares to get t
        gradient_magnitude = np.sqrt(gradient_sum_squares)
        if gradient_magnitude < tolerance:
            converged = True
    return(weights)
```

Entrée [48]:
```
# calculer la descente de gradient

initial_weights = np.random.randn(7)
step_size = 8e-8
tolerance = 1e9
up_weights = batch_gradient_descent(x_train, y_train, initial_weights, st

print ('inital weights:', initial_weights, 'cost function:', cost_functio
print('')
print ('up_weights by BGD: ', up_weights, 'cost function:', cost_function
print('')
print ('weights by matrix calculus:', a, 'cost function:', cost_function(
```

```
inital weights: [-0.10177396 -0.85161756  1.10385209  0.20558433 -0.33423
095 -1.23546543
  0.86024016] cost function: 59.124595305725265

up_weights by BGD:  [ 8.28694644 -7.91553858 -0.20243202  0.10952762 -1.2
3858315 -1.3756928
  0.98111341] cost function: 6.927821976036074

weights by matrix calculus: [[15.50006698]
 [-8.08162095]
 [-0.40498323]
 [-0.14910847]
 [-0.98711701]
 [-2.70044322]
 [ 0.90380672]] cost function: 6.009992903768611
```

5.Régression linéaire multiple

On utilise ici studentregli qui est une copie du dataframe student de départ

Entrée [49]:
```
#On essaie de faire une regression linéaire pour prédire la moyenne des n
```

Entrée [50]:
```
studentregli.dtypes
```

Out[50]:
```
gender                          int8
race/ethnicity                  int8
parental level of education     int8
lunch                           int8
test preparation course         int8
math score                     int64
reading score                  int64
writing score                  int64
mean score                   float64
dtype: object
```

Entrée [51]:
```
# On convertit les variables catégorielles en numériques
```

Entrée [52]:
```
mapGender = {'female':0,'male':1}
mapGroup = {'group C':3,'group D':4,'group B' :2,'group E':5,'group A':1}
mapLevel = {'some college':1,"associate's degree":2,"high school":3,
            'some high school':4,"bachelor's degree":5,"master's degree":
mapLunch = {"standard":0,"free/reduced":1}
mapPrepare = {'none':0,'completed':1}
```

Entrée [53]:
```
studentregli['gender'] = student['gender'].map(mapGender)
studentregli['race/ethnicity'] = student['race/ethnicity'].map(mapGroup)
studentregli['parental level of education'] = student['parental level of
studentregli['lunch'] = student['lunch'].map(mapLunch)
studentregli['test preparation course'] = student['test preparation cours
```

Entrée [54]:
```
studentregli.dtypes
```

Out[54]:
```
gender                          int64
race/ethnicity                  int64
parental level of education     int64
lunch                           int64
test preparation course         int64
math score                      int64
reading score                   int64
writing score                   int64
mean score                    float64
dtype: object
```

Entrée [55]:
```
#On sépare les features de la target pour la régression linéaire
```

Entrée [56]:
```
X = studentregli.drop(columns = ['mean score', 'writing score', 'reading
y = studentregli['mean score']
```

Création des datasets Train et Test

Entrée [57]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

Entrée [58]:
```
# instancier l'objet lm
lm = LinearRegression()
```

Entrée [59]:
```
# fitter le modèle sur le jeu train
lm.fit(X_train,y_train)
```

Out[59]:
```
LinearRegression()
```

Entrée [60]:
```
# afficher l'estimation de l'intercept et les coefficients par le modèle
print(lm.intercept_, lm.coef_)
```

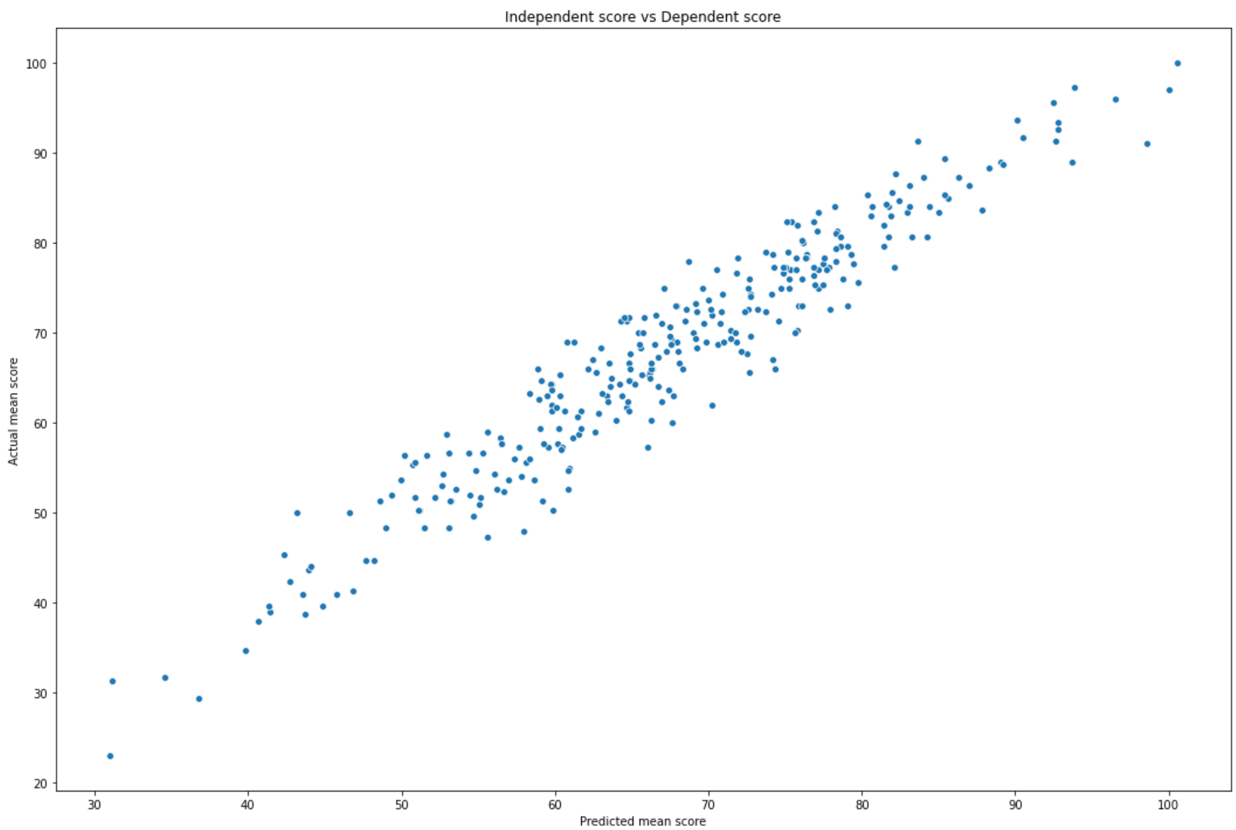
```
11.269166294994676 [-8.08141916 -0.38793342  0.13435224  0.9973349   2.69
584437  0.90558752]
```

Entrée [61]:
```
# prédire sur le jeu test
predictions = lm.predict(X_test)
```

On représente ici ce qu'on a prédit versus ce qu'on devrait avoir (les valeurs réelles)

Entrée [62]:
```
sns.scatterplot(y=y_test,x=predictions)
plt.title('Independent score vs Dependent score')
plt.xlabel('Predicted mean score')
plt.ylabel('Actual mean score')
```

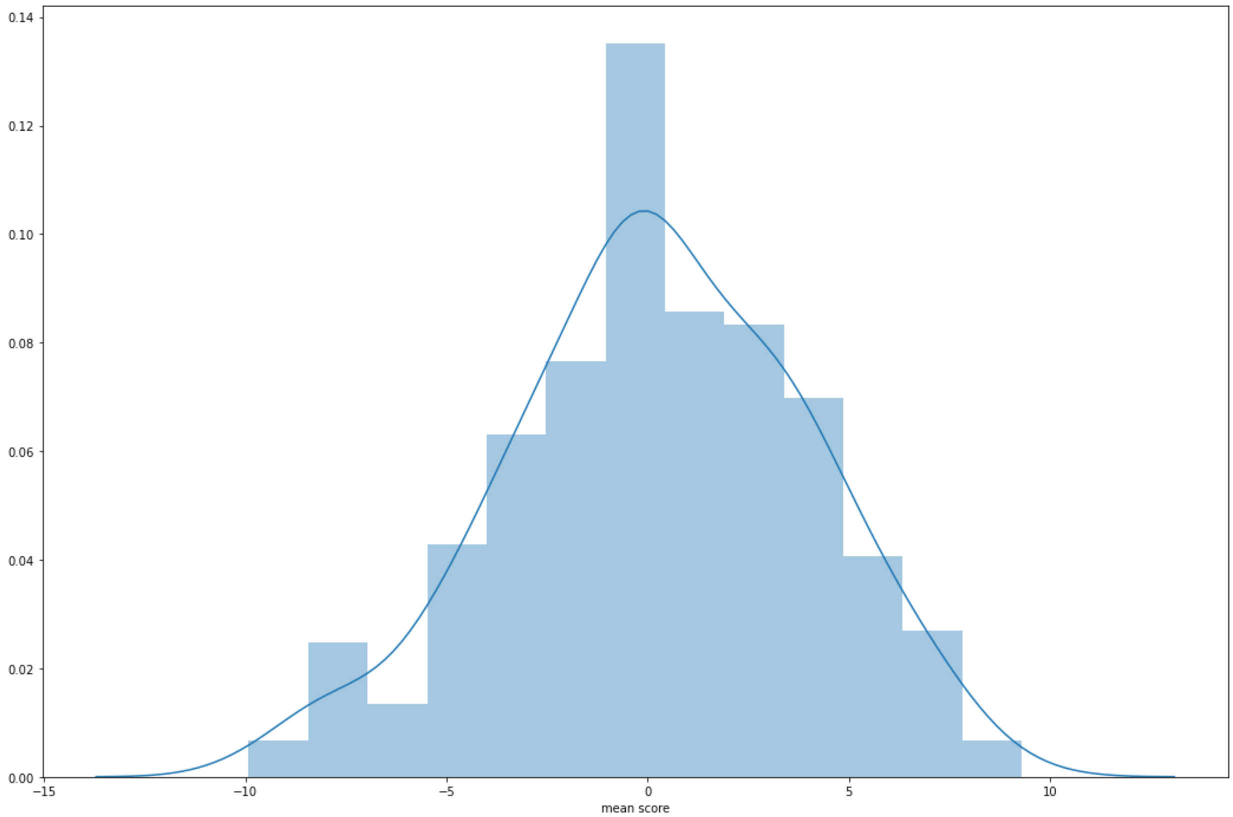
Out[62]:
```
Text(0, 0.5, 'Actual mean score')
```



Entrée [63]:
```
#On observe la répartition des erreurs et le taux d'erreur
```

Entrée [64]:
```
sns.distplot((y_test-predictions))
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff97c4fd670>



Hypothèse stochastique vérifiée: les erreurs du modèle suivent une loi normale centrée réduite autour de 0.

Entrée [65]:
```
#L'erreur absolue moyenne est une mesure des erreurs entre des observatic
```

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|$$

Entrée [66]:
```
MAE = metrics.mean_absolute_error(y_test,predictions)
MAE
```
Out[66]:
```
2.974385520434697
```

Entrée [67]:
```
#L'erreur quadratique moyenne
```

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

Entrée [68]:
```
MSE = metrics.mean_squared_error(y_test,predictions)
MSE
```
Out[68]:
```
13.96769043872347
```

Entrée [69]:
```
# root mean squared error (RMSE)
np.sqrt(metrics.mean_squared_error(y_test,predictions))
```
Out[69]:
```
3.7373373461227004
```

Entrée [70]:
```
y_pred = lm.predict(X_test)
print('Accuracy of linear regression on test set: {:.2f}'.
      format(lm.score(X_test, y_test)))
```
```
Accuracy of linear regression on test set: 0.92
```

6.Régression logistique

Entrée [71]:
```
# vérifier s'il y a un déséquilibre entre les différentes classes
studentregli.groupby(['gender']).gender.count()
```
Out[71]:
```
gender
0    518
1    482
Name: gender, dtype: int64
```

Entrée [72]:
```
studentregli.groupby(['lunch']).gender.count()
```

Out[72]:
```
lunch
0    645
1    355
Name: gender, dtype: int64
```

Entrée [73]:
```
studentregli.groupby(['test preparation course']).gender.count()
```

Out[73]:
```
test preparation course
0    642
1    358
Name: gender, dtype: int64
```

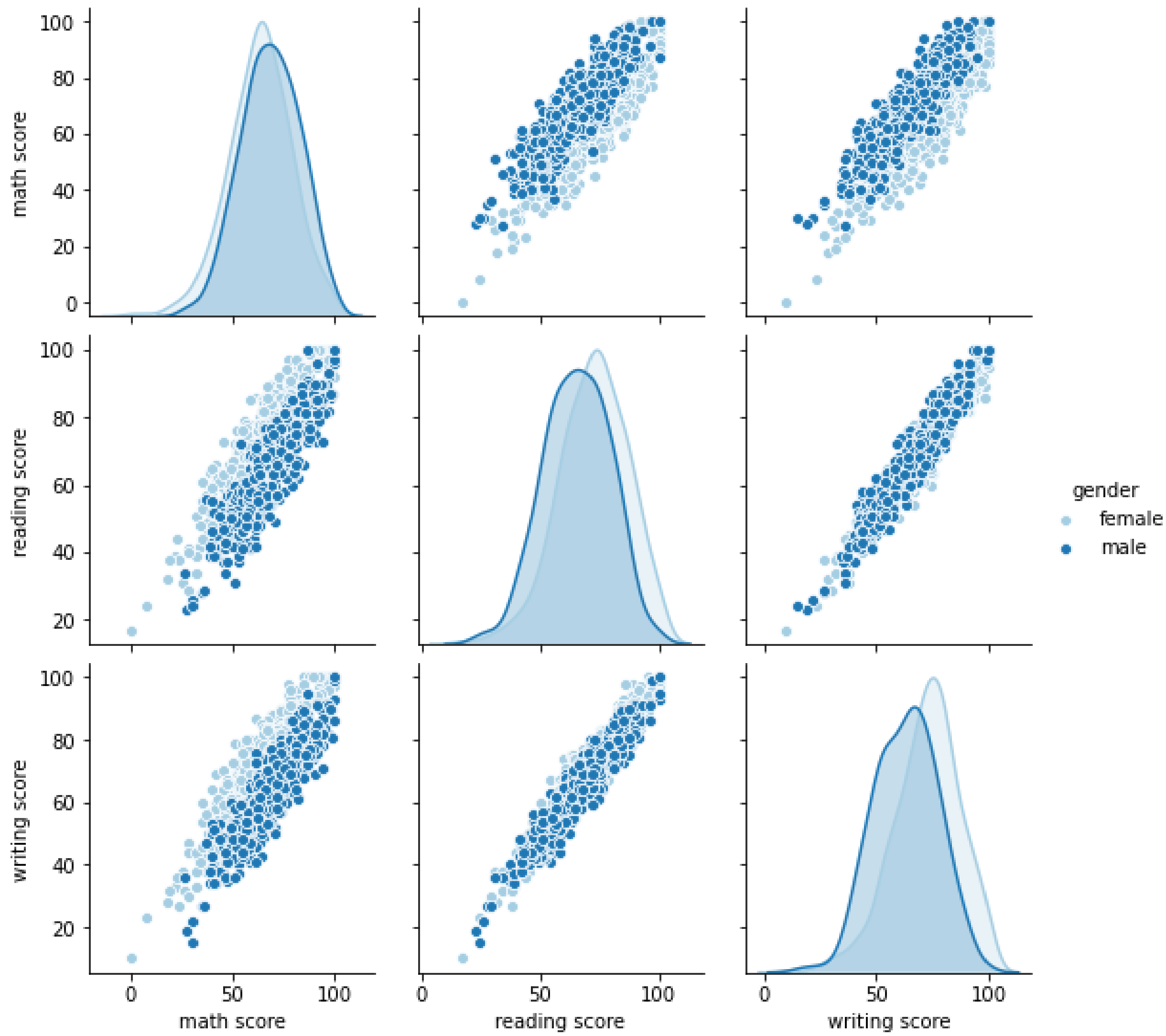
Entrée [74]:
```
colomns_drop = [ "race/ethnicity", "parental level of education", "lunch"
studentregli.drop(colomns_drop, axis=1, inplace=True)
```

Entrée [76]:
```
# la variable à prédire 'gender'
y = studentregli.gender.values
y.shape
```

Out[76]:
```
(1000,)
```

Entrée [92]:
```
# relation entre les 3 scores pour les 2 classes de 'gender'
sns.pairplot(student.drop(['total','Total','percentage'],axis=1),hue='gen
```

Out[92]: <seaborn.axisgrid.PairGrid at 0x7ff97f23eac0>



Entrée [77]:
```
X = studentregli.drop("gender", axis=1)
X.shape
```

Out[77]:
```
(1000, 3)
```

Entrée [78]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3
```

Entrée [79]:
```
print('Train:', x_train.shape, '\n Test:', x_test.shape)
```

```
Train: (700, 3)
 Test: (300, 3)
```

Entrée [80]:
```
LogisticReg = LogisticRegression(max_iter = 1000)
```

Entrée [81]:
```
LogisticReg.fit(x_train, y_train)
```

Out[81]:
```
LogisticRegression(max_iter=1000)
```

Entrée [82]:
```
acc_train = LogisticReg.score(x_train, y_train)
print("Precision du model :", (acc_train * 100).round())
```

```
Precision du model : 87.0
```

Entrée [83]:
```
y_pred = LogisticReg.predict(x_test)
```

Entrée [84]:
```
acc = LogisticReg.score(x_test, y_test)
print("Precision du model :", (acc * 100).round())
```

```
Precision du model : 90.0
```

Entrée [85]:
```
confusion = confusion_matrix(y_test, y_pred)
print(confusion)
```

```
[[147  11]
 [ 20 122]]
```

À partir de notre matrice de conclusion, nous pouvons voir que notre modèle a obtenu (147 + 122) 269 prédictions correctes et (20 + 11) 31 prédictions fausses.

Entrée [86]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       158
           1       0.92      0.86      0.89       142

    accuracy                           0.90       300
   macro avg       0.90      0.89      0.90       300
weighted avg       0.90      0.90      0.90       300
```

Interprétation: De notre rapport de classification, nous pouvons voir que notre modèle a un taux de précision de 90% et un taux de rappel de 90%, notre modèle à un taux de prediction correcte