<u>Classification report</u>

First for all:

a) Preprocessing technique:

1- As I saw the first view of table I have 6 categorical features

as I use classification techniques they are

Dealing with numerical data so I had to converting them:

```
encoding = LabelEncoder()
seg['Gender'] = encoding.fit_transform(seg['Gender'])

seg['Ever_Married'] = encoding.fit_transform(seg['Ever_Married'])

seg['Graduated'] = encoding.fit_transform(seg['Graduated'])

seg['Profession'] = encoding.fit_transform(seg['Profession'])

seg['Spending_Score'] = encoding.fit_transform(seg['Spending_Score'])

seg['Var_1'] = encoding.fit_transform(seg['Var_1'])
```

in feature I found 3 parameters only(Gender, Ever_Married, Graduated, Profession, Spending_Score, Var_1) when they are converted the scene will be (0 ,1 ,2,4,etc…)

so, I decided to :

```
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
x1 = np.array(seg['Ever_Married'], dtype=int64)
seg['Ever_Married'] = imp.fit_transform(x1.reshape(-1, 1))

x2 = np.array(seg['Graduated'], dtype=int64)
seg['Graduated'] = imp.fit_transform(x2.reshape(-1, 1))

x3 = np.array(seg['Profession'], dtype=int64)
seg['Profession'] = imp.fit_transform(x3.reshape(-1, 1))

x4 = np.array(seg['Work_Experience'], dtype=int64)
seg['Work_Experience'] = imp.fit_transform(x4.reshape(-1, 1))

x5 = np.array(seg['Family_Size'], dtype=int64)
seg['Family_Size'] = imp.fit_transform(x5.reshape(-1, 1))

x6 = np.array(seg['Var_1'], dtype=int64)
seg['Var_1'] = imp.fit_transform(x6.reshape(-1, 1))
```

I found most of columns has nan values so, I had to remove them with any strategy in simple imputer the , I chooses the most_frequent parameter

Because it almost make the model more fitting for data to use.

```
 # # preprocessing MinMaxscaler
scale = MinMaxScaler(copy=True, feature_range=(0, 1))
a = np.array(seg['Age'], dtype=int64)
seg['Age'] = scale.fit_transform(a.reshape(-1, 1))

b = np.array(seg['Family_Size'], dtype=int64)
seg['Family_Size'] = scale.fit_transform(b.reshape(-1, 1))
```

Here I used minmax normalizer almost the same perprocessing technique

In regression phase , but here I chooses the most high range data and scaling it to be more usable in prediction

b) Analysis

1- when I applied the analysis found that the most features affects the model predictions are

Gender Ever_Married Graduated Profession Spending_Score Var_1

These are the categorical parameters that I had to:

1st in some model they are dealing with categorical data an some are numerical data so I had to use both

```
 # Check on the nan cells
print(pre.columns[pre.isnull().any()].tolist())
print(pre.isnull().any())
```

2nd I wanted to check on the nan values in all features

And the output is:

ID False

Gender False

Ever_Married True

Age False

Graduated True

Profession True

Work_Experience True

Spending_Score   False

Family_Size   True

Var_1   True

Segmentation   False

dtype: bool

ID   False

Gender   False

Ever_Married   False

Age   False

Graduated   False

Profession   False

Work_Experience   False

Spending_Score   False

Family_Size   False

Var_1   False

Segmentation   False

Any value are true it contains nan values so , they must be removed or scale

```
from sklearn.feature_selection import SelectFromModel
select2 = SelectFromModel(RandomForestRegressor())
Selected = select2.fit_transform(x, y)
print(Selected.shape)
print(select2.get_support())
```

here I got some help from sklearn library by featuring by model here I pick up the best features I use for prediction as you see I used ensemble model that I used it (explained later)
but in my final result I get numbers and I wanted to see and visualize so:
[True False False True False False True False False False]
When applied selection from model the out put parameters are Boolean true values are good feature for applying predictions , but when I used it I found high overfitting for them so , worked on all features.

c) Classification models:

```
d) #
   =================================ENSEMLE=================================
   ===========
   # Gradient Boosting Classifier
   model1 = GradientBoostingClassifier(learning_rate=0.04)
   # ---------------------------------------------------------------
   # Random forest classifier
   model2 = RandomForestClassifier()

   #
   =================================NORMAL==================================
   ===========
   # Logistic regression
   model3 = LogisticRegression()
   # -----------------------------------------------------------------
   # KNN
   model5 = KNeighborsClassifier()
   # -----------------------------------------------------------------
   # Decision tree
   model6 = DecisionTreeClassifier(max_depth=10)
   # -----------------------------------------------------------------
   # naive bayes
   model7 = GaussianNB()
   model8 = BernoulliNB()
   # -----------------------------------------------------------------
   # LDA
   model9 = LinearDiscriminantAnalysis()
   # -----------------------------------------------------------------
   # QDA
   model10 = QuadraticDiscriminantAnalysis()
   # ====================================
   model1.fit(x_train, y_train)
   # sorted(model11.cv_results_.keys())
   y_pred = model1.predict(x_train)
   print('my predictions:', y_pred)
   print('accuracy score is:', accuracy_score(y_train, y_pred))
```

Every model I use has it’s own parameters for prediction so

Lets get started:

Every model has it’s score

1st logistic regression

accuracy score is: 0.25590917550716547

2nd KNeighborsClassifier

accuracy score is: 0.543644146659222

3rd DecisionTreeClassifier(max_depth=10)

accuracy score is: 0.629071282337614

4th linear discremenet analysis

accuracy score is: 0.4524474222966685

5th QuadraticDiscriminantAnalysis

accuracy score is: 0.25032570258700915

most of these values are almost having low score over or under fitting

I applied changes in parameters for this to in ensemble mode all I have to do just playing with the learning_rate starts from(0.05 to 1)

The values I set in:

Gradient Boosting Classifier

model1 = GradientBoostingClassifier(learning_rate=0.04)

gives me high accuracy for this model on Kaggle.

And you will see this in confusion matrix later.

e) Different between models:
   1st logistic regression
   accuracy score is: 0.25590917550716547
   2nd KNeighborsClassifier
   accuracy score is: 0.543644146659222
   3rd DecisionTreeClassifier(max_depth=10)
   accuracy score is: 0.629071282337614
   4th linear discremenet analysis
   accuracy score is: 0.4524474222966685

5th QuadraticDiscriminantAnalysis

accuracy score is: 0.25032570258700915

6th GaussianNB

accuracy score is: 0.25590917550716547

7th BernoulliNB()

accuracy score is: 0.42024939512376697

ensembles are the most useable models the are satisfied to me for both regression and classification

I used :

1- random forest regressor and classifier

2- Gradient boosting regressor and classifier

f) For the features I used them all I discareded nothing they are all Important with removing one of them they affected badly on score I just used to scale them according to the type of model that I work on it for now.

g) The size are default :

```
x_train, x_test, y_train, y_test = train_test_split(X, Y,
shuffle=False, random_state=33)
```

using default parameters of splitting data like I said in in regression

h) Her I used support of something:

```
i) eclf = VotingClassifier(
       estimators=[('1', model1), ('2', model2), ('3', model3), ('5',
   model5), ('6', model6),
                   ('7', model7), ('8', model8), ('9', model9), ('10',
   model10)], voting='hard')
   f, axarr = plt.subplots(2, 2, sharex="col", sharey="row", figsize=(10,
   8))
   for clf, label in zip(
           product([0, 1], [0, 1]),[model1, model2, model3, model5,
   model6, model7, model8, model9, model10, eclf],
                             ['GradientBoostingClassifier',
   'RandomForestClassifier','LogisticRegression',
                              'KNN', 'DecisionTreeClassifier', 'GaussianNB',
                              'BernoulliNB', 'LinearDiscriminantAnalysis',
   'QuadraticDiscriminantAnalysis']):
       scores = cross_val_score(clf, X, Y, scoring='accuracy')
       print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(),
   scores.std(), label))

   # decision boundray
```

```
    DecisionBoundaryDisplay.from_estimator(
        clf, X, alpha=0.4, ax=axarr[idx[0], idx[1]],
response_method="predict"
    )
    axarr[idx[0], idx[1]].scatter(X[:, 0], X[:, 1], c=9, s=20,
edgecolor="k")
    axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```

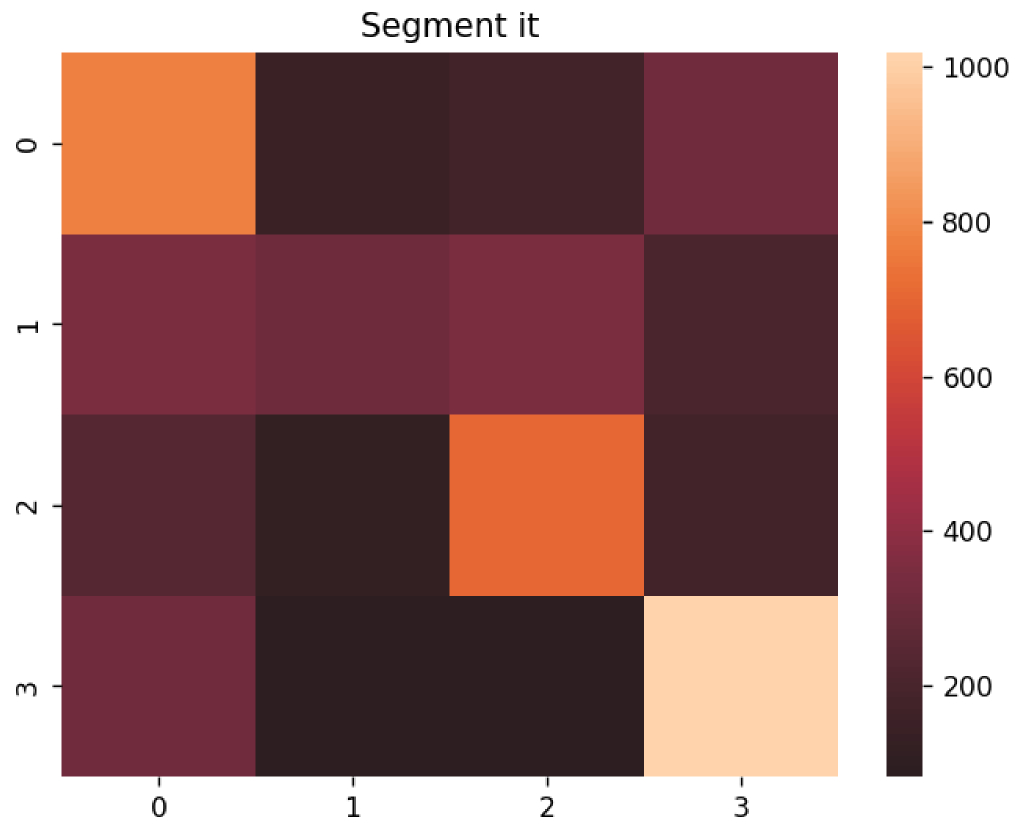
voting classifier led me to the numerous of models that I use and the x & y data for model and it gets me the rank of them from the best to lowest rank score model that best fit the data in them.

Accuracy: 0.48 (+/- 0.01) [GradientBoostingClassifier]
Accuracy: 0.44 (+/- 0.01) [RandomForestClassifier]
Accuracy: 0.25 (+/- 0.01) [LogisticRegression]
Accuracy: 0.31 (+/- 0.01) [KNN]
Accuracy: 0.43 (+/- 0.01) [DecisionTreeClassifier]
Accuracy: 0.25 (+/- 0.00) [GaussianNB]
Accuracy: 0.42 (+/- 0.01) [BernoulliNB]
Accuracy: 0.45 (+/- 0.01) [LinearDiscriminantAnalysis]
Accuracy: 0.25 (+/- 0.01) [QuadraticDiscriminantAnalysis]

They trying to help me for choosing the most good model to use

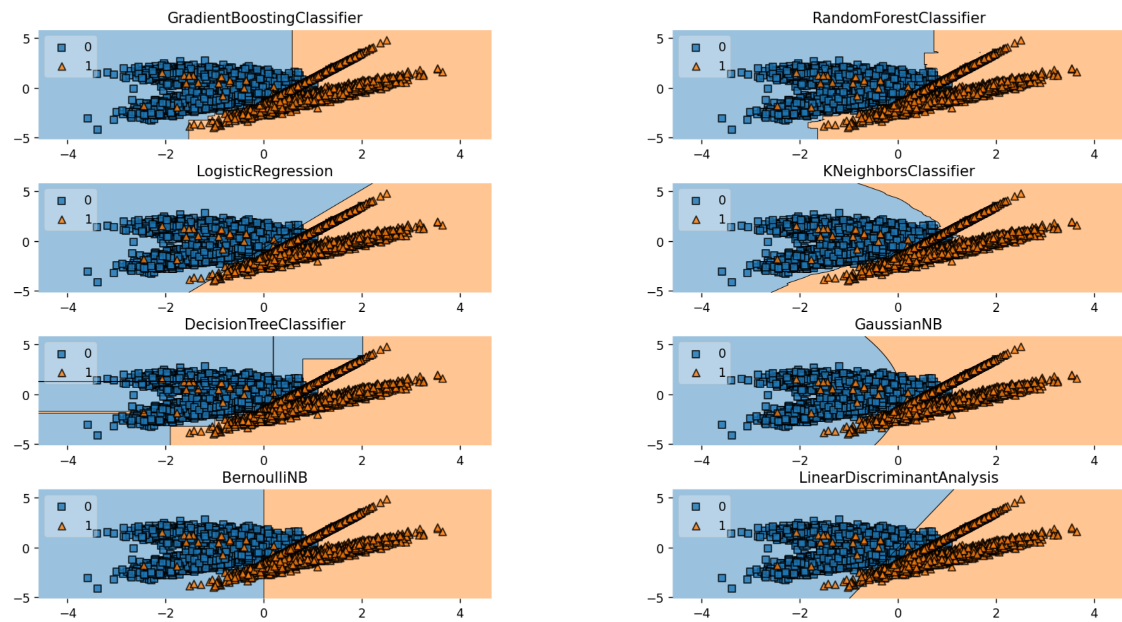
The other improvement are in the preprocessing that I did

j) the confusion matrix of all models I used in it:



k) My final conclusion is that I did suffered a lot for the first time
In scaling data as I did in regression:
Model contains nan. Values the means to optimize the
And remove these values as I said upper
Most of the data are categorized and that make modelling them easy to me by using label and one_hot encoder for them
Most of model deals with numerical data than categorical one

Some ensemble model deals with data contains nan values and that seems perfect to me like (hist gradient boosting classifier) by the way I so it is type for code errors ,compilers advice me to use it as my data contains nan values.



Their it is my decision regions for every model I used encluding ensemble :

```
# decision boundray
from sklearn.datasets import make_classification
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt

X, Y = make_classification(n_samples=7165, n_features=2, n_informative=2,
n_redundant=0, n_classes=2)
import matplotlib.gridspec as gridspec

gs = gridspec.GridSpec(4, 2)

fig = plt.figure(figsize=(20, 10))
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)

labels = ['GradientBoostingClassifier', 'RandomForestClassifier',
```

```
'LogisticRegression', 'KNeighborsClassifier',
          'DecisionTreeClassifier', 'GaussianNB', 'BernoulliNB',
'LinearDiscriminantAnalysis']
for clf, lab, grd in zip([model1, model2, model3, model4, model5, model6,
model7, model8],
                         labels,
                         [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1), (3,
0), (3, 1)]):
    clf.fit(X, Y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=Y, clf=clf, legend=2)
    plt.title(lab)

plt.show()
```

This is the code for using sklearn.dataset classification mode to use it But , when I search for how to make decision boundry I found that it almost working on SVM only for numrious features I could use it by votting classifier but it needs various mathimatical equation that doesn't fit my data in it ,

```
eclf = VotingClassifier(
    estimators=[('1', model1), ('2', model2), ('3', model3), ('5', model5),
('6', model6),
                ('7', model7), ('8', model8), ('9', model9), ('10',
model10)], voting='hard')
for clf, label in zip([model1, model2, model3, model5, model6, model7,
model8, model9, model10, eclf],
                      ['GradientBoostingClassifier',
'RandomForestClassifier','LogisticRegression',
                       'KNN', 'DecisionTreeClassifier', 'GaussianNB',
                       'BernoulliNB', 'LinearDiscriminantAnalysis',
'QuadraticDiscriminantAnalysis']):
```

this is mostly the same code for printing a grid of a boundrey decision of multimodels but it needs as I said.

```
X, Y = make_classification(n_samples=7165, n_features=2, n_informative=2,
n_redundant=0, n_classes=2)
```

These parameters show that the decision grid works only on binary elements (two features seleceted only) so I decied to make it the same number of sample I used in compition and unfortunately very exhustive to change every feature in table and seeing what you 've got in it.