In [118…

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from time import time
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import confusion_matrix, roc_curve, accuracy_score, f1_score, roc
from astropy.table import Table
from sklearn.metrics import roc_auc_score

df = pd.read_csv('student-data.csv')
dfv = pd.read_csv('student-data.csv')
```

In [ ]:

In [119…

```
df
```

Out[119]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	rom
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	no	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	yes	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	yes	
3	GP	F	15	U	GT3	T	4	2	health	services	...	yes	
4	GP	F	16	U	GT3	T	3	3	other	other	...	no	
...	...	...	...	...	...	...	...	...	...	...	...	...	
390	MS	M	20	U	LE3	A	2	2	services	services	...	no	
391	MS	M	17	U	LE3	T	3	1	services	services	...	yes	
392	MS	M	21	R	GT3	T	1	1	other	other	...	no	
393	MS	M	18	R	LE3	T	3	2	services	other	...	yes	
394	MS	M	19	U	LE3	T	1	1	other	at_home	...	yes	

395 rows × 31 columns

In [120…

```
def numerical_data():
    df['school'] = df['school'].map({'GP': 0, 'MS': 1})
    df['sex'] = df['sex'].map({'M': 0, 'F': 1})
    df['address'] = df['address'].map({'U': 0, 'R': 1})
    df['famsize'] = df['famsize'].map({'LE3': 0, 'GT3': 1})
    df['Pstatus'] = df['Pstatus'].map({'T': 0, 'A': 1})
    df['Mjob'] = df['Mjob'].map({'teacher': 0, 'health': 1, 'services': 2, 'at_home':
    df['Fjob'] = df['Fjob'].map({'teacher': 0, 'health': 1, 'services': 2, 'at_home':
    df['reason'] = df['reason'].map({'home': 0, 'reputation': 1, 'course': 2, 'other':
    df['guardian'] = df['guardian'].map({'mother': 0, 'father': 1, 'other': 2})
    df['schoolsup'] = df['schoolsup'].map({'no': 0, 'yes': 1})
    df['famsup'] = df['famsup'].map({'no': 0, 'yes': 1})
```

```
    df['paid'] = df['paid'].map({'no': 0, 'yes': 1})
    df['activities'] = df['activities'].map({'no': 0, 'yes': 1})
    df['nursery'] = df['nursery'].map({'no': 0, 'yes': 1})
    df['higher'] = df['higher'].map({'no': 0, 'yes': 1})
    df['internet'] = df['internet'].map({'no': 0, 'yes': 1})
    df['romantic'] = df['romantic'].map({'no': 0, 'yes' : 1})
    df['passed'] = df['passed'].map({'no': 0, 'yes': 1})
    # reorder dataframe columns :
    col = df['passed']
    del df['passed']
    df['passed'] = col


# feature scaling will allow the algorithm to converge faster, large data will have sc
def feature_scaling(df):
    for i in df:
        col = df[i]
        # let's choose columns that have large values
        if(np.max(col)>6):
            Max = max(col)
            Min = min(col)
            mean = np.mean(col)
            col  = (col-mean)/(Max)
            df[i] = col
        elif(np.max(col)<6):
            col = (col-np.min(col))
            col /= np.max(col)
            df[i] = col
```

In [121…

```
numerical_data()
df
```

Out[121]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	romantic
0	0	1	18	0	1	1	4	4	3	0	...	0	0
1	0	1	17	0	1	0	1	1	3	4	...	1	0
2	0	1	15	0	0	0	1	1	3	4	...	1	0
3	0	1	15	0	1	0	4	2	1	2	...	1	1
4	0	1	16	0	1	0	3	3	4	4	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
390	1	0	20	0	0	1	2	2	2	2	...	0	0
391	1	0	17	0	0	0	3	1	2	2	...	1	0
392	1	0	21	1	1	0	1	1	4	4	...	0	0
393	1	0	18	1	0	0	3	2	2	4	...	1	0
394	1	0	19	0	0	0	1	1	4	3	...	1	0

395 rows × 31 columns

In [122…

```
# Let's scal our features
feature_scaling(df)
```

```
# Now we are ready for models training
df
```

Out[122]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	roma
0	0.0	1.0	0.059264	0.0	1.0	1.0	1.00	1.00	0.75	0.00	...	0.0	
1	0.0	1.0	0.013809	0.0	1.0	0.0	0.25	0.25	0.75	1.00	...	1.0	
2	0.0	1.0	-0.077100	0.0	0.0	0.0	0.25	0.25	0.75	1.00	...	1.0	
3	0.0	1.0	-0.077100	0.0	1.0	0.0	1.00	0.50	0.25	0.50	...	1.0	
4	0.0	1.0	-0.031646	0.0	1.0	0.0	0.75	0.75	1.00	1.00	...	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
390	1.0	0.0	0.150173	0.0	0.0	1.0	0.50	0.50	0.50	0.50	...	0.0	
391	1.0	0.0	0.013809	0.0	0.0	0.0	0.75	0.25	0.50	0.50	...	1.0	
392	1.0	0.0	0.195627	1.0	1.0	0.0	0.25	0.25	1.00	1.00	...	0.0	
393	1.0	0.0	0.059264	1.0	0.0	0.0	0.75	0.50	0.50	1.00	...	1.0	
394	1.0	0.0	0.104718	0.0	0.0	0.0	0.25	0.25	1.00	0.75	...	1.0	

395 rows × 31 columns

In [123…
```
df.shape
```

Out[123]:
```
(395, 31)
```

In [124…
```
df.dropna().shape # their is no null value "fortunately:)"
```

Out[124]:
```
(395, 31)
```

In [125…
```
df.columns
```

Out[125]:
```
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences', 'passed'],
      dtype='object')
```

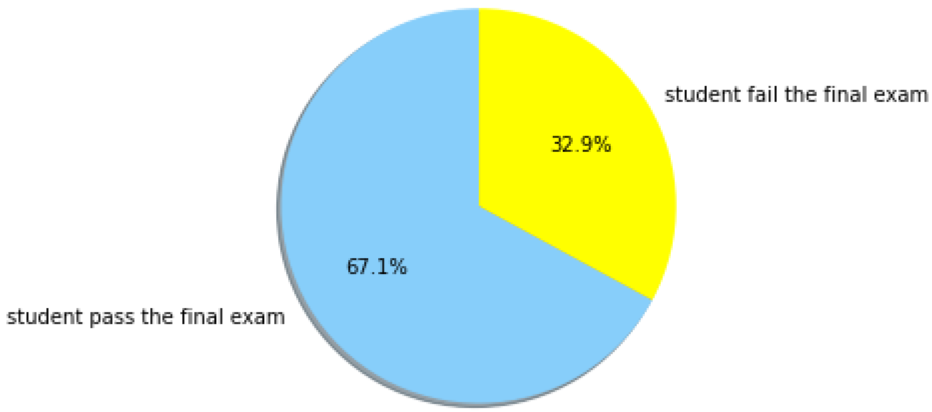
In [126…
```
features=['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences']
```

In [127…
```
#plot of student status
dfv['passed'].value_counts()
```

Out[127]:
```
yes    265
no     130
Name: passed, dtype: int64
```

In [128…

```
labels = 'student pass the final exam ', 'student fail the final exam'
sizes = [265, 130]
colors=['lightskyblue','yellow']
fig1, ax1 = plt.subplots()
ax1.pie(sizes,  labels=labels, autopct='%1.1f%%',colors=colors,
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

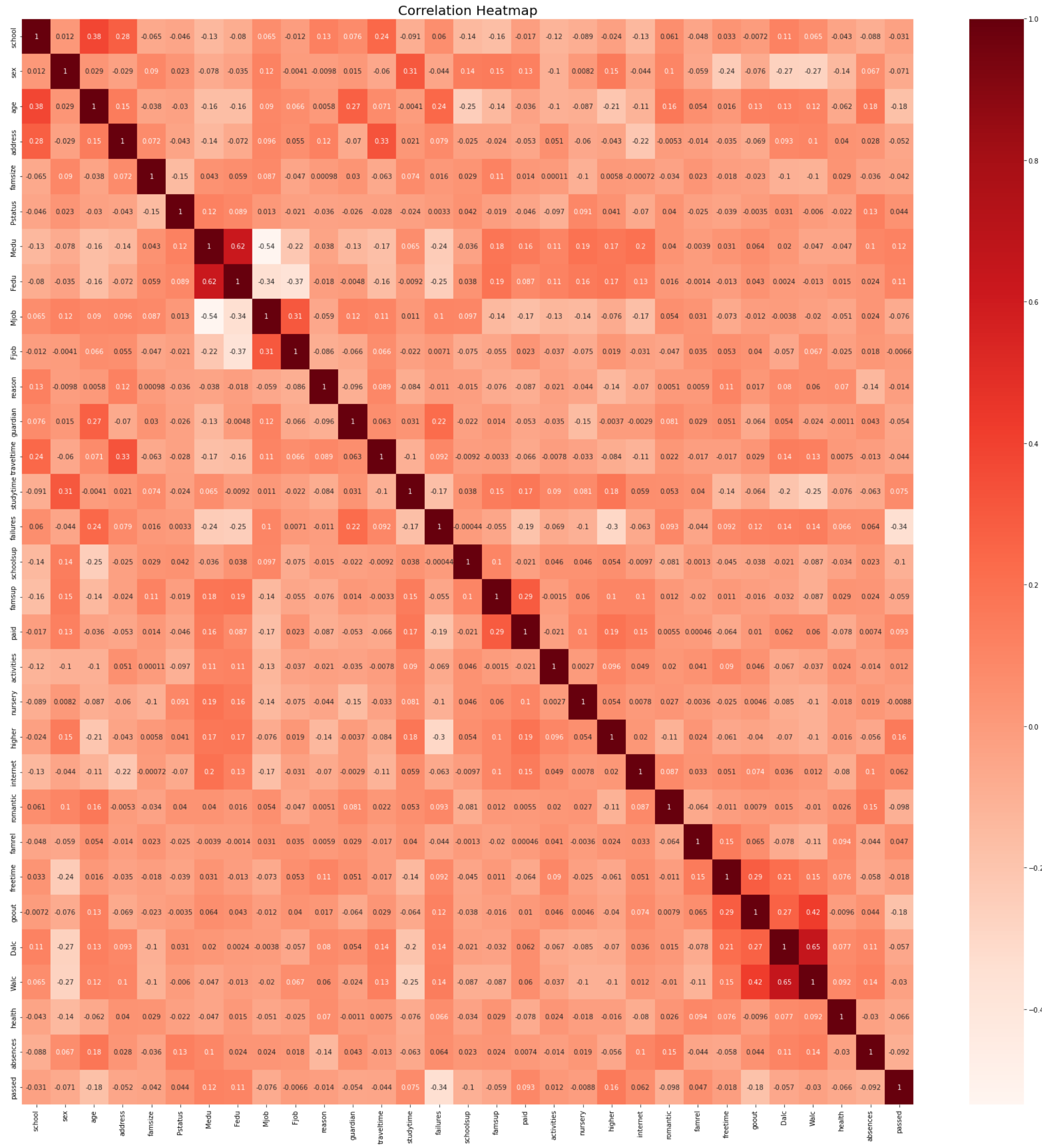


In [129…

```
# see correlation between variables through a correlation heatmap
corr = df.corr()
plt.figure(figsize=(30,30))
sns.heatmap(corr, annot=True, cmap="Reds")
plt.title('Correlation Heatmap', fontsize=20)
```

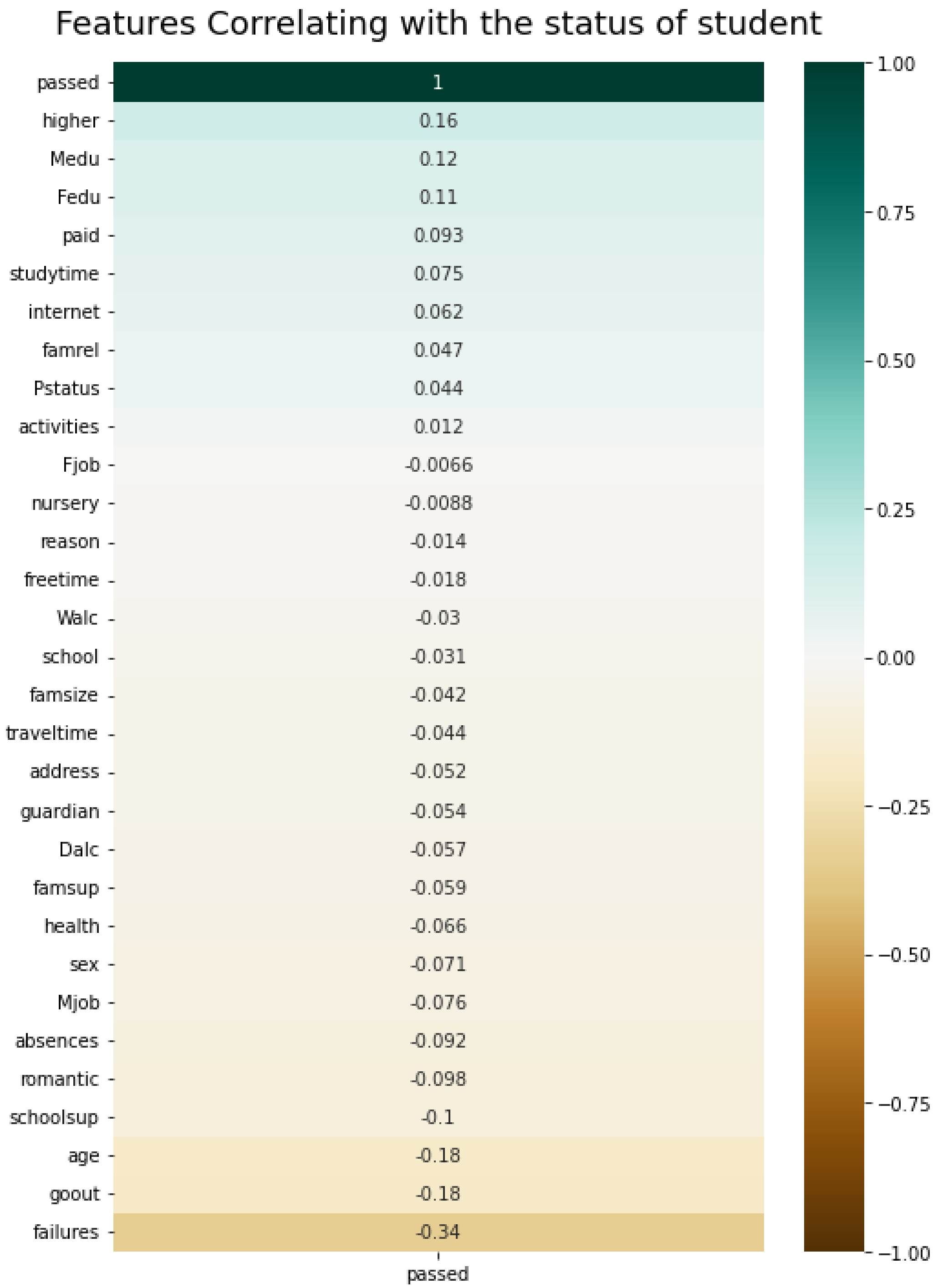
Out[129]:

```
Text(0.5, 1.0, 'Correlation Heatmap')
```



In [130…

```
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(df.corr()[['passed']].sort_values(by='passed', ascending=False),
heatmap.set_title('Features Correlating with the status of student', fontdict={'fontsi
```

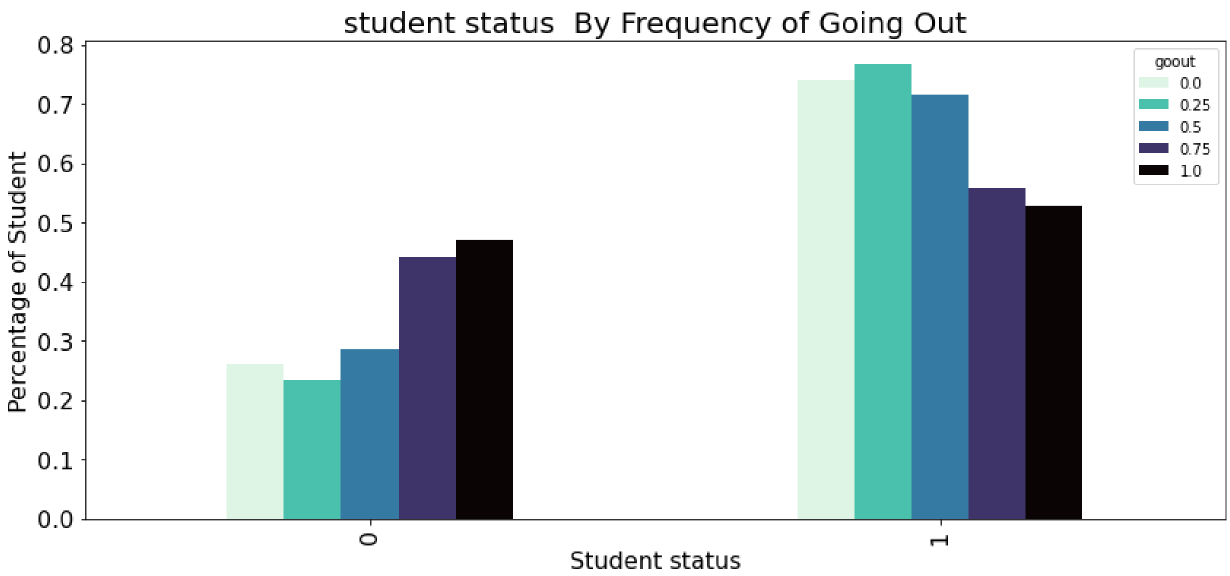


In [131…
```
df["goout"].unique()
```

Out[131]:
```
array([0.75, 0.5 , 0.25, 0.  , 1.  ])
```

In [132…
```
# going out
perc = (lambda col: col/col.sum())
index = [0,1]
out_tab = pd.crosstab(index=df.passed, columns=df.goout)
out_perc = out_tab.apply(perc).reindex(index)
out_perc.plot.bar(colormap="mako_r", fontsize=16, figsize=(14,6))
plt.title('student status  By Frequency of Going Out', fontsize=20)
plt.ylabel('Percentage of Student', fontsize=16)
plt.xlabel('Student status', fontsize=16)
```

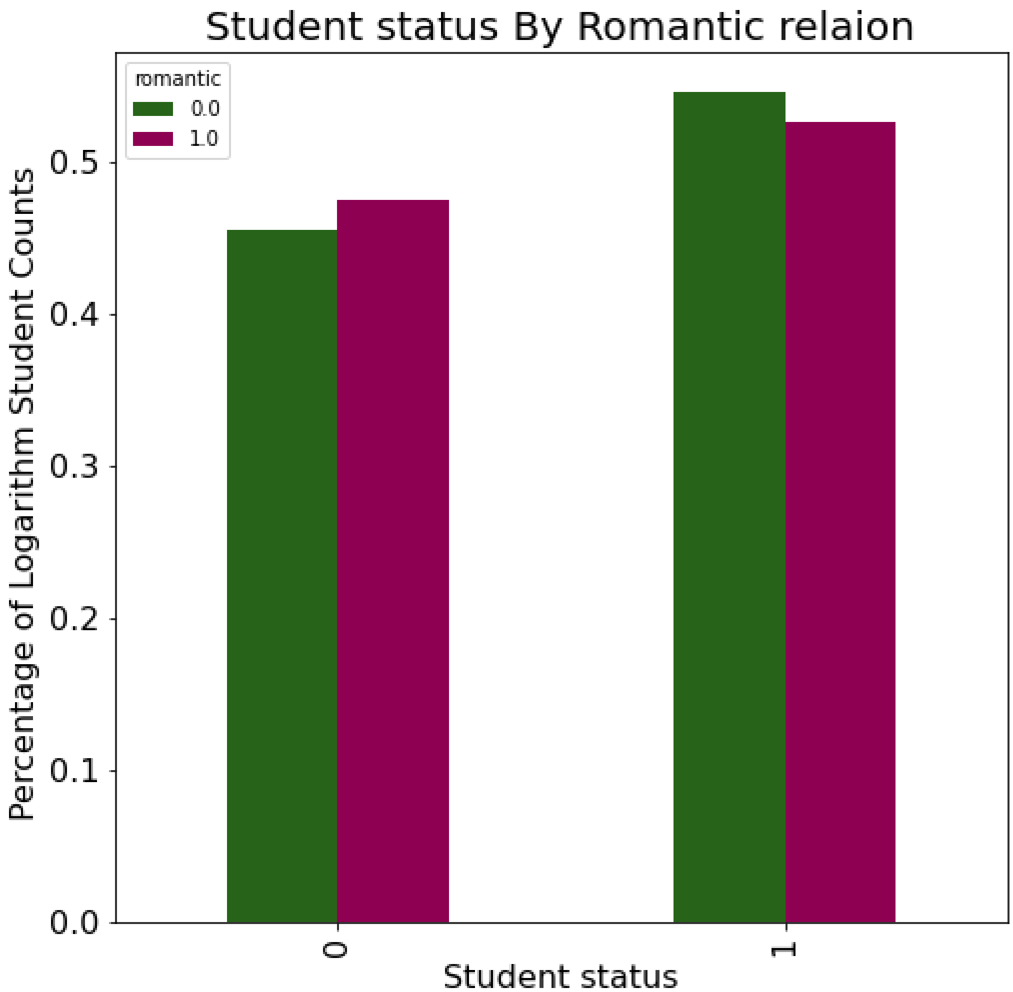
Out[132]: Text(0.5, 0, 'Student status')



In [133…

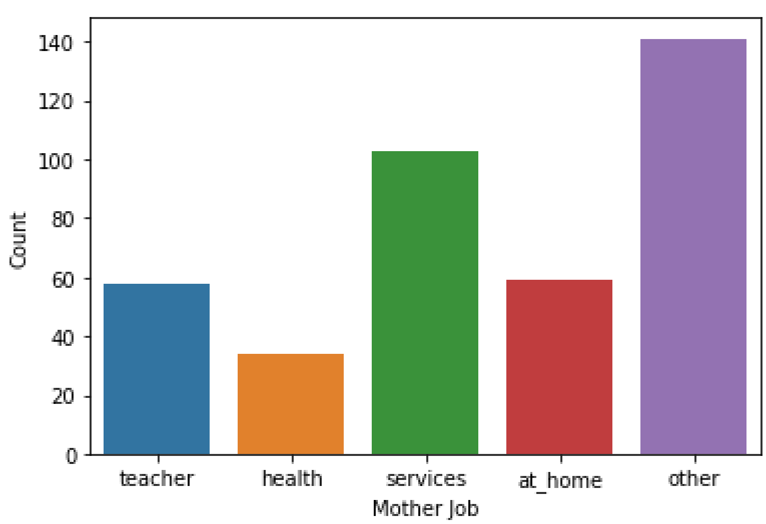
```
# romantic status
romance_tab1 = pd.crosstab(index=df.passed, columns=df.romantic)
romance_tab = np.log(romance_tab1)
romance_perc = romance_tab.apply(perc).reindex(index)
plt.figure()
romance_perc.plot.bar(colormap="PiYG_r", fontsize=16, figsize=(8,8))
plt.title('Student status By Romantic relaion', fontsize=20)
plt.ylabel('Percentage of Logarithm Student Counts ', fontsize=16)
plt.xlabel('Student status', fontsize=16)
plt.show()
# 0 in romantic mean no romantic relation
```

<Figure size 432x288 with 0 Axes>



In [134…

```
# 1) mother job
# Mjob distribution
f, fx = plt.subplots()
figure = sns.countplot(x = 'Mjob', data=dfv, order=['teacher','health','services','at_
fx = fx.set(ylabel="Count", xlabel="Mother Job")
figure.grid(False)
```

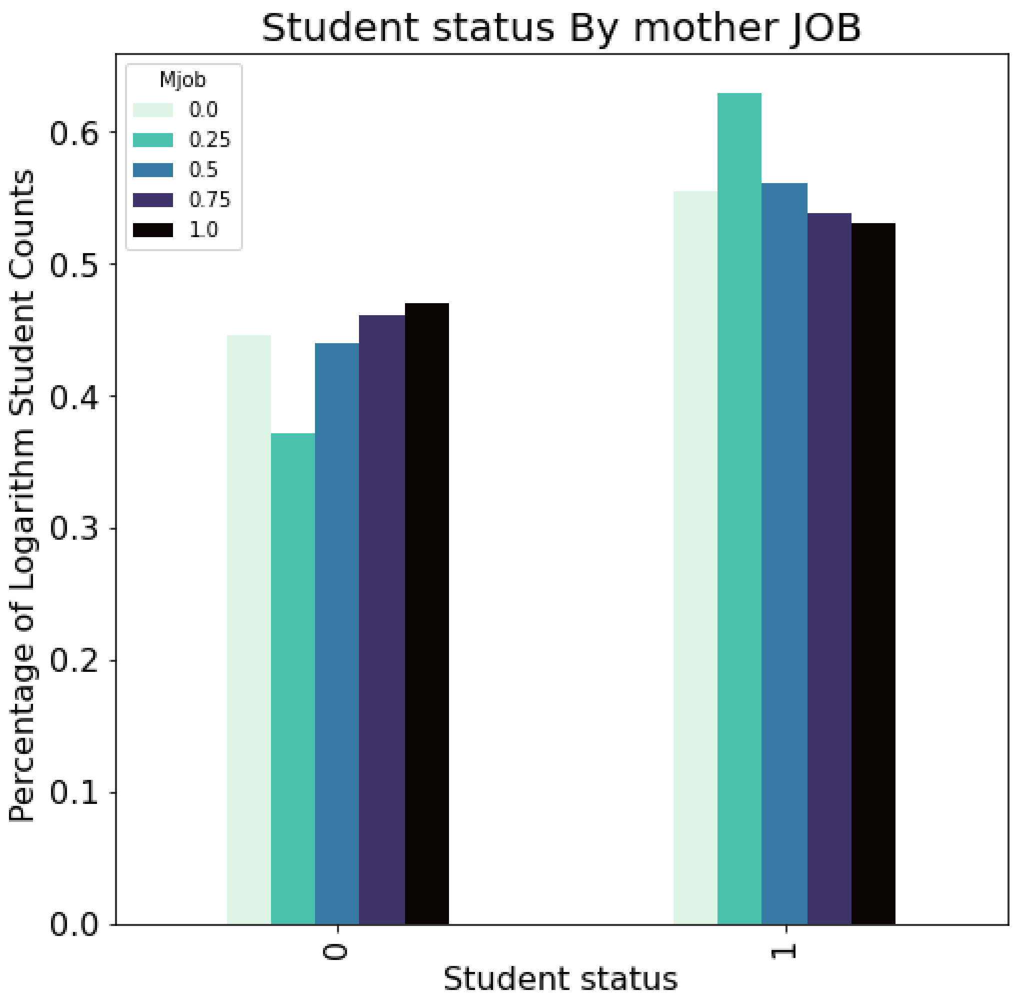


In [135…

```
mjob_tab1 = pd.crosstab(index=df.passed, columns=df.Mjob)
mjob_tab = np.log(mjob_tab1)
mjob_perc = mjob_tab.apply(perc).reindex(index)
```

```
plt.figure()
mjob_perc.plot.bar(colormap="mako_r", fontsize=16, figsize=(8,8))
plt.title('Student status By mother JOB', fontsize=20)
plt.ylabel('Percentage of Logarithm Student Counts ', fontsize=16)
plt.xlabel('Student status', fontsize=16)
plt.show()
#'teacher': 0, 'health': 1, 'services': 2, 'at_home': 3, 'other': 4
```

```
<Figure size 432x288 with 0 Axes>
```



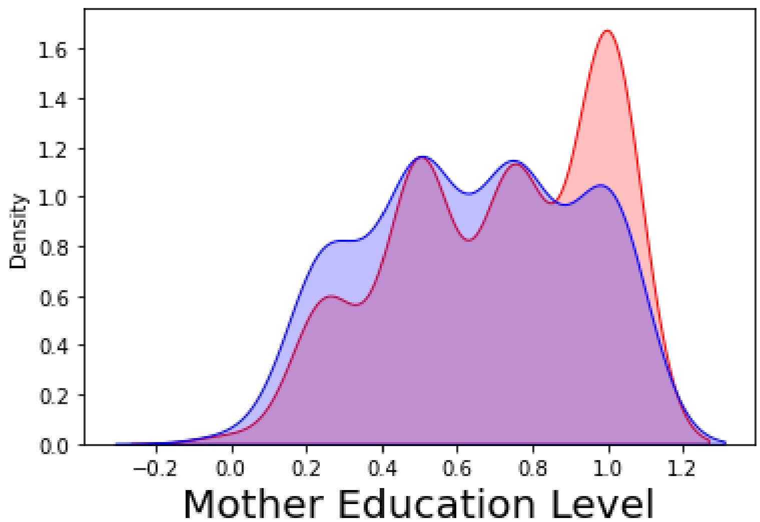
In [136...

```
#Mother education:
good = df.loc[df.passed==1]
poor=df.loc[df.passed==0]
good['good_student_mother_education'] = good.Medu
poor['poor_student_mother_education'] = poor.Medu
plt.figure(figsize=(6,4))
p=sns.kdeplot(good['good_student_mother_education'], shade=True, color="r")#good_stude
p=sns.kdeplot(poor['poor_student_mother_education'], shade=True, color="b")#poor_stude
plt.xlabel('Mother Education Level', fontsize=20)
```

```
C:\Users\sivas\AppData\Local\Temp\ipykernel_6968\3018233835.py:4: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  good['good_student_mother_education'] = good.Medu
C:\Users\sivas\AppData\Local\Temp\ipykernel_6968\3018233835.py:5: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  poor['poor_student_mother_education'] = poor.Medu
```

Out[136]:

```
Text(0.5, 0, 'Mother Education Level')
```

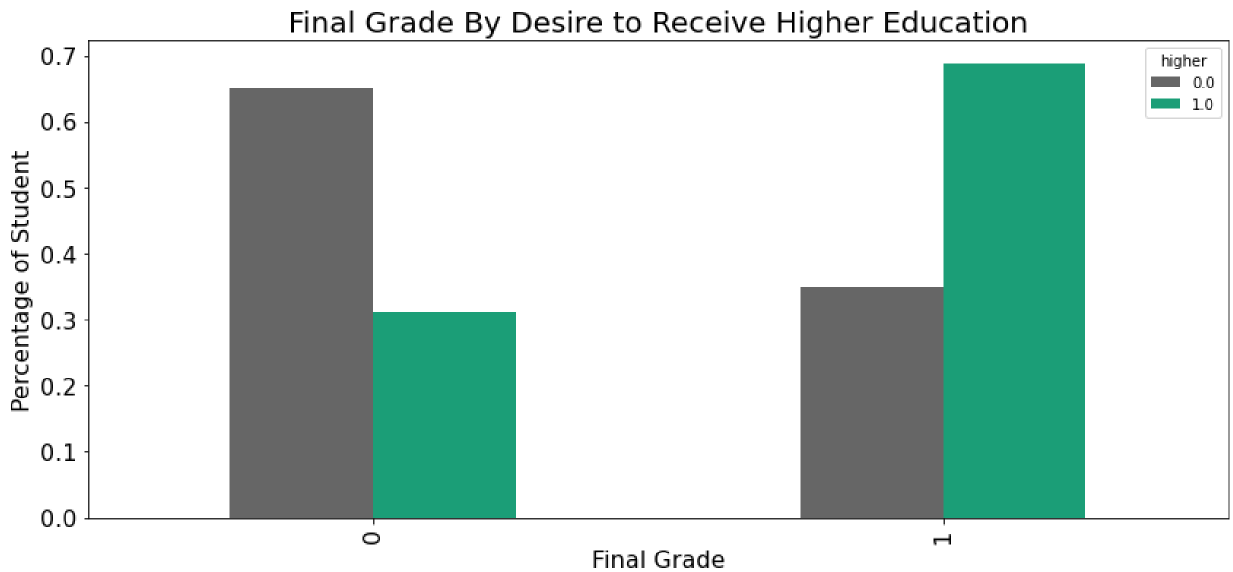


In [137…

```
higher_tab = pd.crosstab(index=df.passed, columns=df.higher)
higher_perc = higher_tab.apply(perc).reindex(index)
higher_perc.plot.bar(colormap="Dark2_r", figsize=(14,6), fontsize=16)
plt.title('Final Grade By Desire to Receive Higher Education', fontsize=20)
plt.xlabel('Final Grade', fontsize=16)
plt.ylabel('Percentage of Student', fontsize=16)
```

Out[137]:

```
Text(0, 0.5, 'Percentage of Student')
```

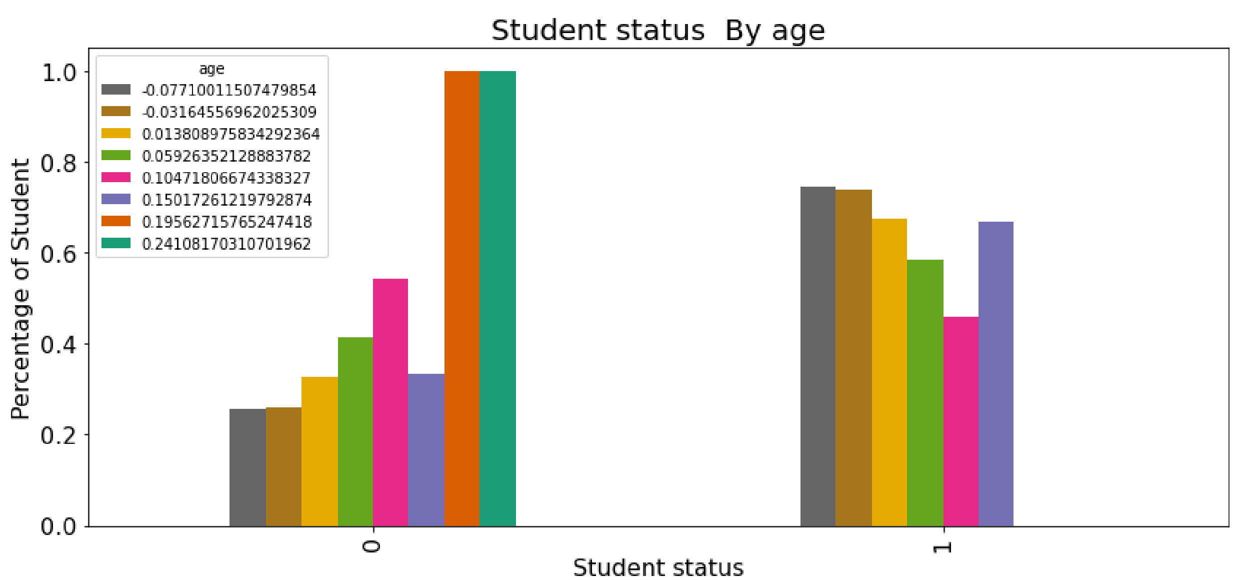


In [138…

```
#impact of age
higher_tab = pd.crosstab(index=df.passed, columns=df.age)
higher_perc = higher_tab.apply(perc).reindex(index)
higher_perc.plot.bar(colormap="Dark2_r", figsize=(14,6), fontsize=16)
plt.title('Student status  By age', fontsize=20)
plt.xlabel('Student status', fontsize=16)
plt.ylabel('Percentage of Student', fontsize=16)
```

Out[138]:

```
Text(0, 0.5, 'Percentage of Student')
```

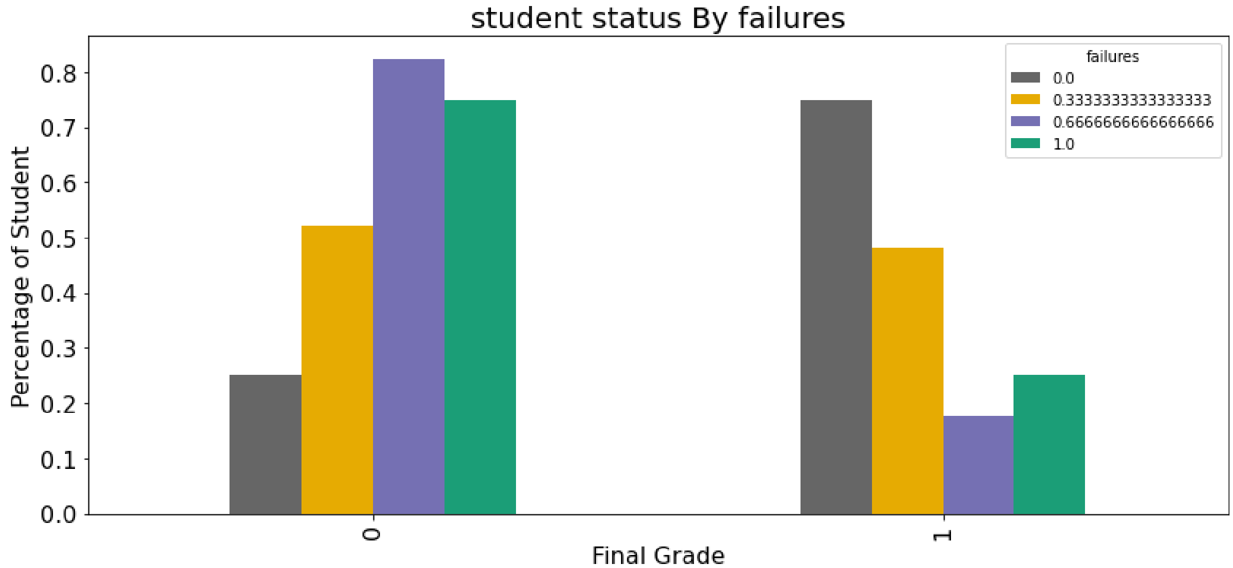


In [139…

```
fail_tab = pd.crosstab(index=df.passed, columns=df.failures)
fail_perc = fail_tab.apply(perc).reindex(index)
fail_perc.plot.bar(colormap="Dark2_r", figsize=(14,6), fontsize=16)
plt.title('student status By failures', fontsize=20)
plt.xlabel('Final Grade', fontsize=16)
plt.ylabel('Percentage of Student', fontsize=16)
```

Out[139]:

```
Text(0, 0.5, 'Percentage of Student')
```

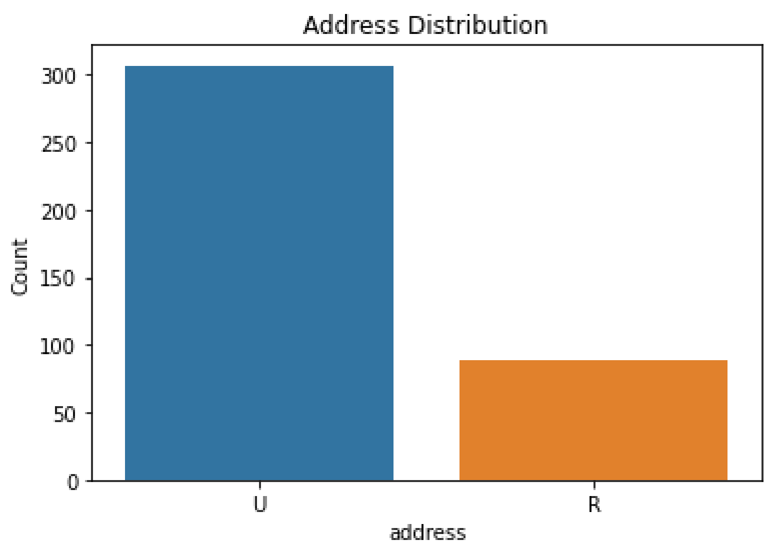


In [140…

```
#first let's see the destribution of students who live in urban or rural area
f, fx = plt.subplots()
figure = sns.countplot(x = 'address', data=dfv, order=['U','R'])
fx = fx.set(ylabel="Count", xlabel="address")
figure.grid(False)
plt.title('Address Distribution')
```

Out[140]:

```
Text(0.5, 1.0, 'Address Distribution')
```

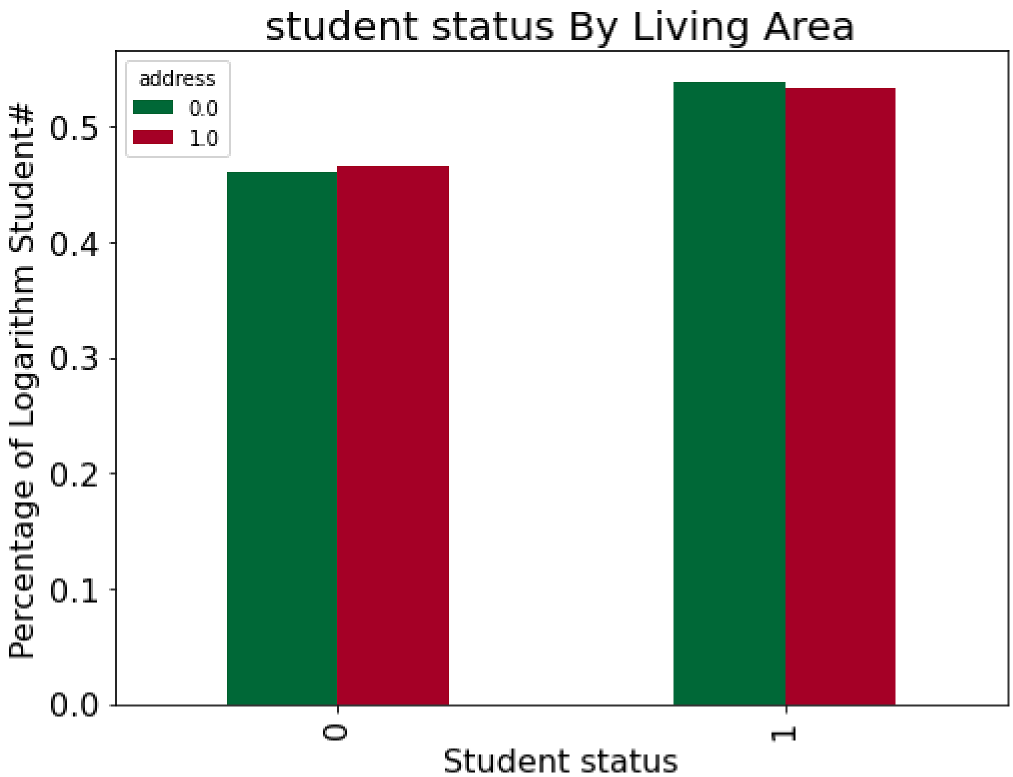


In [141…

```
ad_tab1 = pd.crosstab(index=df.passed, columns=df.address)
ad_tab = np.log(ad_tab1)
ad_perc = ad_tab.apply(perc).reindex(index)
ad_perc.plot.bar(colormap="RdYlGn_r", fontsize=16, figsize=(8,6))
plt.title('student status By Living Area', fontsize=20)
plt.ylabel('Percentage of Logarithm Student#', fontsize=16)
plt.xlabel('Student status', fontsize=16)
```

Out[141]:

```
Text(0.5, 0, 'Student status')
```

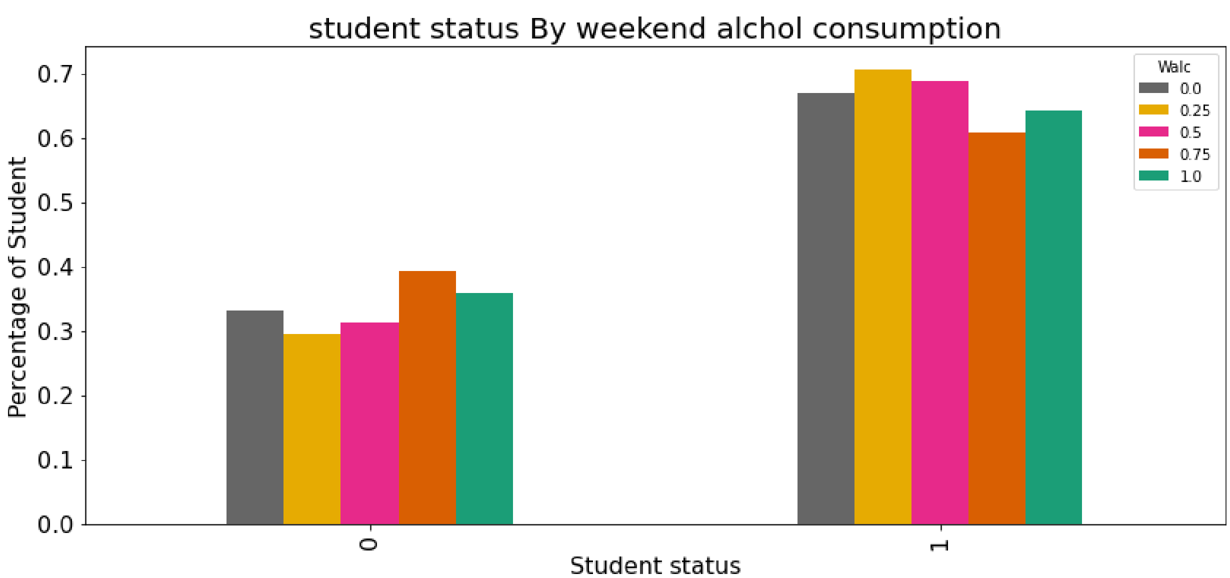


In [142…

```
#impact of weekend alcohol consumption in student performance
alc_tab = pd.crosstab(index=df.passed, columns=df.Walc)
alc_perc = alc_tab.apply(perc).reindex(index)
alc_perc.plot.bar(colormap="Dark2_r", figsize=(14,6), fontsize=16)
plt.title('student status By weekend alchol consumption', fontsize=20)
plt.xlabel('Student status', fontsize=16)
plt.ylabel('Percentage of Student', fontsize=16)
```

Out[142]:

```
Text(0, 0.5, 'Percentage of Student')
```



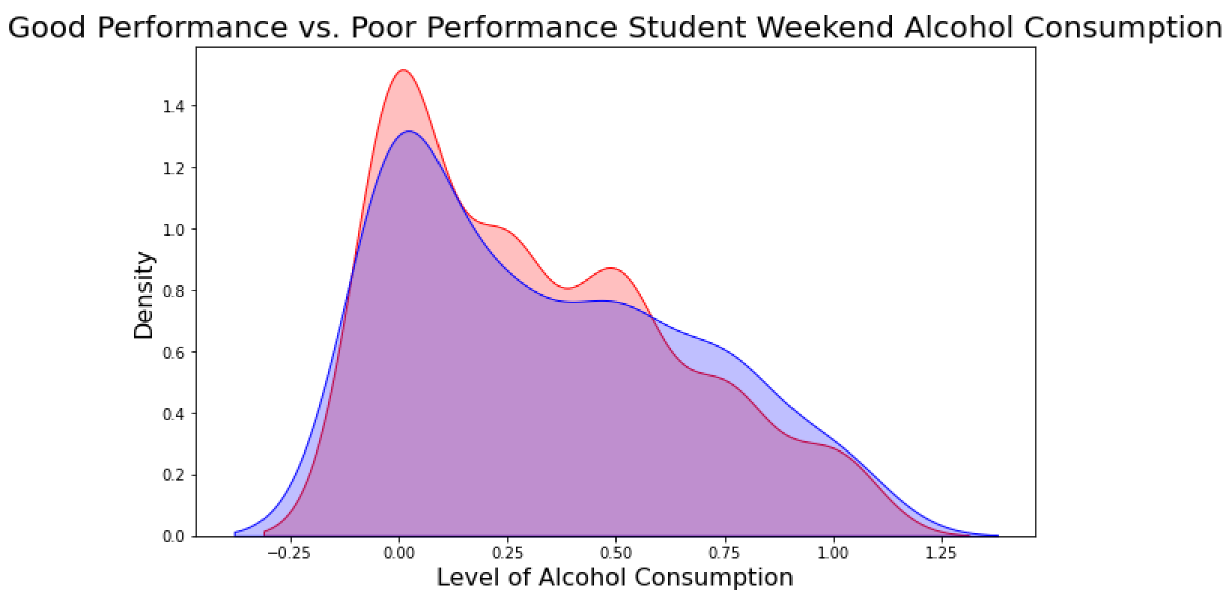
In [143…

```
# weekend alcohol consumption
# create good student dataframe
good = df.loc[df.passed == 1]
good['good_alcohol_usage']=good.Walc
# create poor student dataframe
poor = df.loc[df.passed == 0]
```

```
poor['poor_alcohol_usage']=poor.Walc
plt.figure(figsize=(10,6))
p1=sns.kdeplot(good['good_alcohol_usage'], shade=True, color="r")
p1=sns.kdeplot(poor['poor_alcohol_usage'], shade=True, color="b")
plt.title('Good Performance vs. Poor Performance Student Weekend Alcohol Consumption',
plt.ylabel('Density', fontsize=16)
plt.xlabel('Level of Alcohol Consumption', fontsize=16)
```

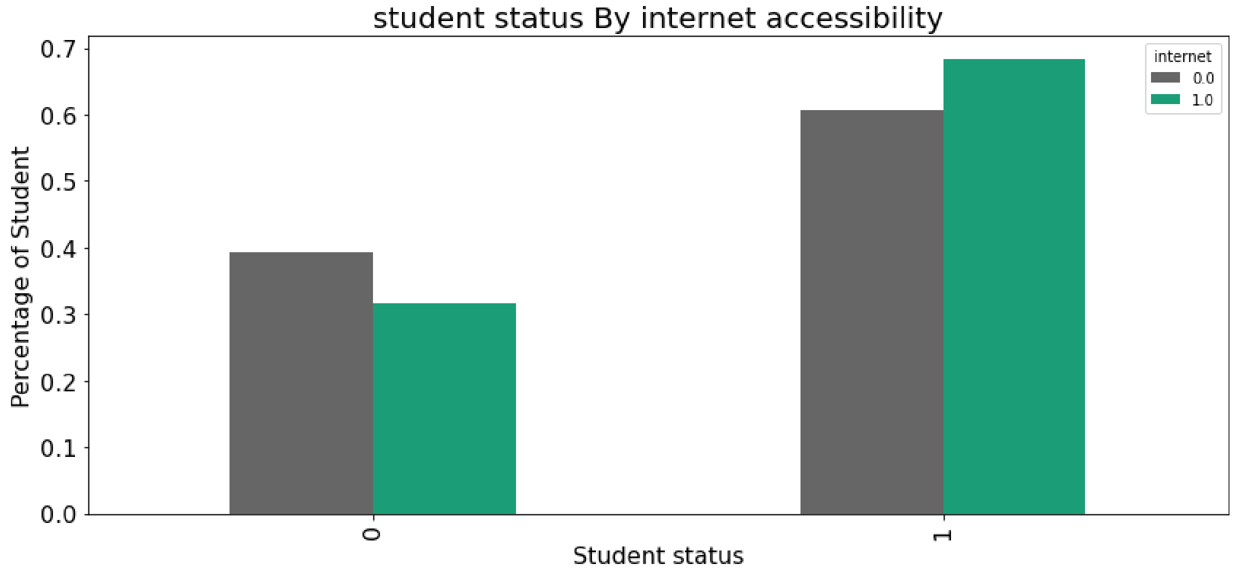
```
C:\Users\sivas\AppData\Local\Temp\ipykernel_6968\1621555142.py:4: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  good['good_alcohol_usage']=good.Walc
C:\Users\sivas\AppData\Local\Temp\ipykernel_6968\1621555142.py:7: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  poor['poor_alcohol_usage']=poor.Walc
```

Out[143]:
```
Text(0.5, 0, 'Level of Alcohol Consumption')
```



In [144…
```
alc_tab = pd.crosstab(index=df.passed, columns=df.internet)
alc_perc = alc_tab.apply(perc).reindex(index)
alc_perc.plot.bar(colormap="Dark2_r", figsize=(14,6), fontsize=16)
plt.title('student status By internet accessibility', fontsize=20)
plt.xlabel('Student status', fontsize=16)
plt.ylabel('Percentage of Student', fontsize=16)
```

Out[144]:
```
Text(0, 0.5, 'Percentage of Student')
```

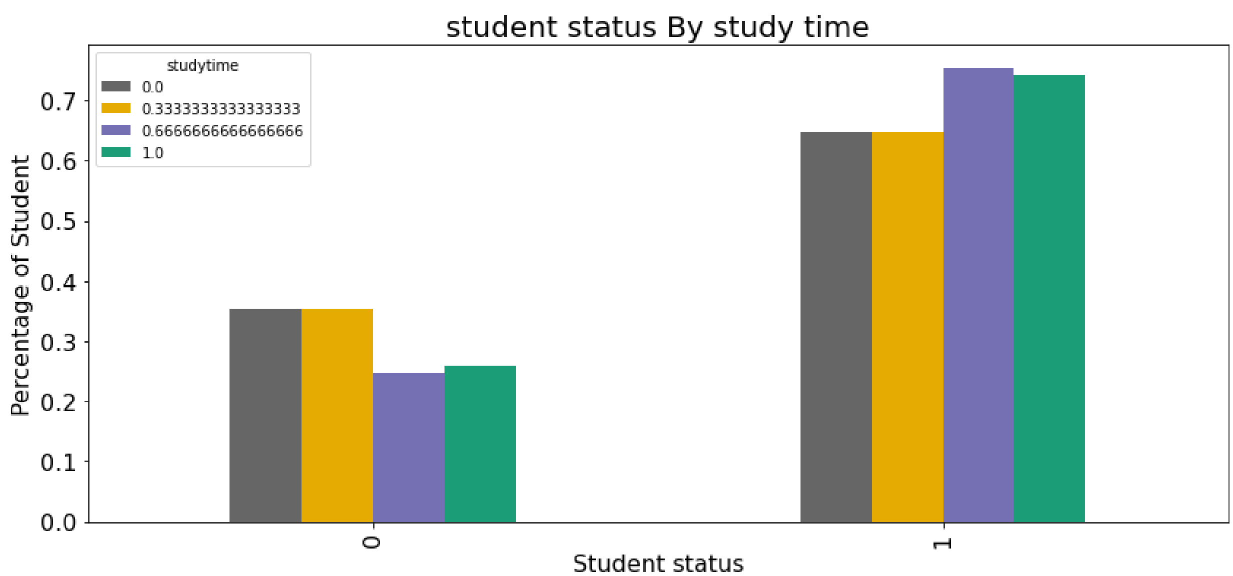


In [145…

```
stu_tab = pd.crosstab(index=df.passed, columns=df.studytime)
stu_perc = stu_tab.apply(perc).reindex(index)
stu_perc.plot.bar(colormap="Dark2_r", figsize=(14,6), fontsize=16)
plt.title('student status By study time', fontsize=20)
plt.xlabel('Student status', fontsize=16)
plt.ylabel('Percentage of Student', fontsize=16)
```

Out[145]:

```
Text(0, 0.5, 'Percentage of Student')
```

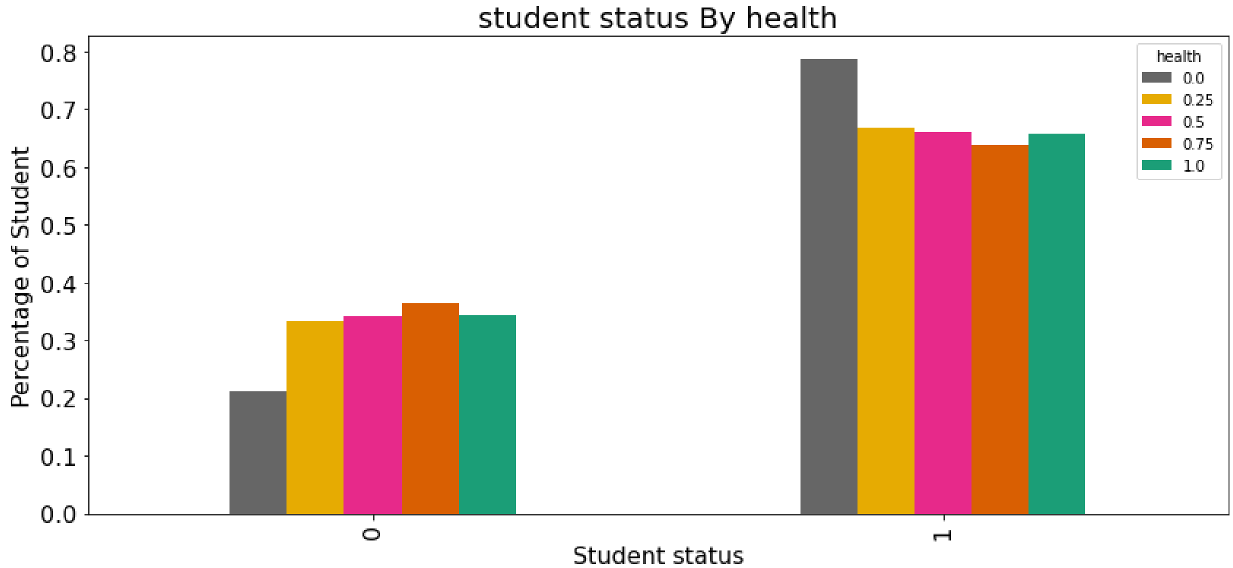


In [146…

```
he_tab = pd.crosstab(index=df.passed, columns=df.health)
he_perc = he_tab.apply(perc).reindex(index)
he_perc.plot.bar(colormap="Dark2_r", figsize=(14,6), fontsize=16)
plt.title('student status By health', fontsize=20)
plt.xlabel('Student status', fontsize=16)
plt.ylabel('Percentage of Student', fontsize=16)
```

Out[146]:

```
Text(0, 0.5, 'Percentage of Student')
```



In [147…
```
data = df.to_numpy()
n = data.shape[1]
x = data[:,0:n-1]
y = data[:,n-1]
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=0)
```

In [148…
```
logisticRegr = LogisticRegression(C=1)
```

In [149…
```
logisticRegr.fit(x_train,y_train)
```

Out[149]:
```
LogisticRegression(C=1)
```

In [150…
```
y_pred=logisticRegr.predict(x_test)
y_pred
```

Out[150]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
       1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0.,
       1., 0., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 1., 0., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1.,
       1., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1.])
```

In [151…
```
Sctest=logisticRegr.score(x_test,y_test)
Sctrain=logisticRegr.score(x_train,y_train)

print('#Accuracy test is: ',Sctest)
print('#Accuracy train is: ',Sctrain)


f1 = f1_score(y_test, y_pred, average='macro')

print('\n#f1 score is: ',f1)
```

```
#Accuracy test is:  0.6386554621848739
#Accuracy train is:  0.7463768115942029

#f1 score is:  0.5533734834598935
```

In [152…
```
#let's have a look at the accuracy of the model
```

```
Sctest=logisticRegr.score(x_test,y_test)
Sctrain=logisticRegr.score(x_train,y_train)

print('Accuracy test is: ',Sctest)
print('Accuracy train is: ',Sctrain)
```

```
Accuracy test is:  0.6386554621848739
Accuracy train is:  0.7463768115942029
```

In [153…

```
#now, we can get the confusion matrix with confusion_matrix():

confusion_matrix(y_test, y_pred)
```

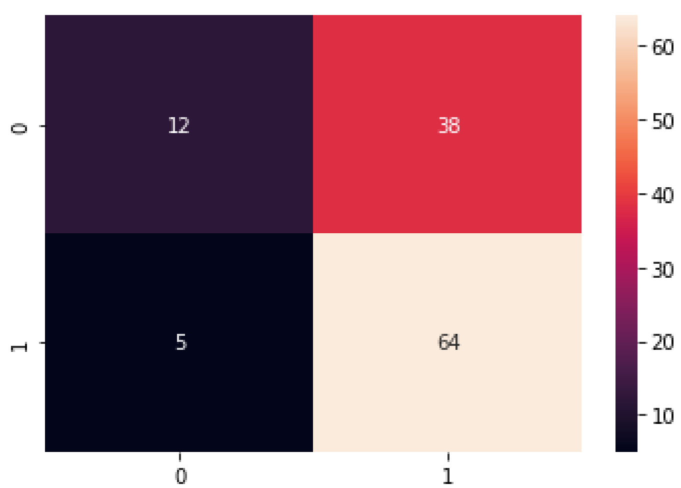
Out[153]:

```
array([[12, 38],
       [ 5, 64]], dtype=int64)
```

In [154…

```
#let's visualize the confusion matrix:
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm,annot=True)
```

Out[154]: <AxesSubplot:>



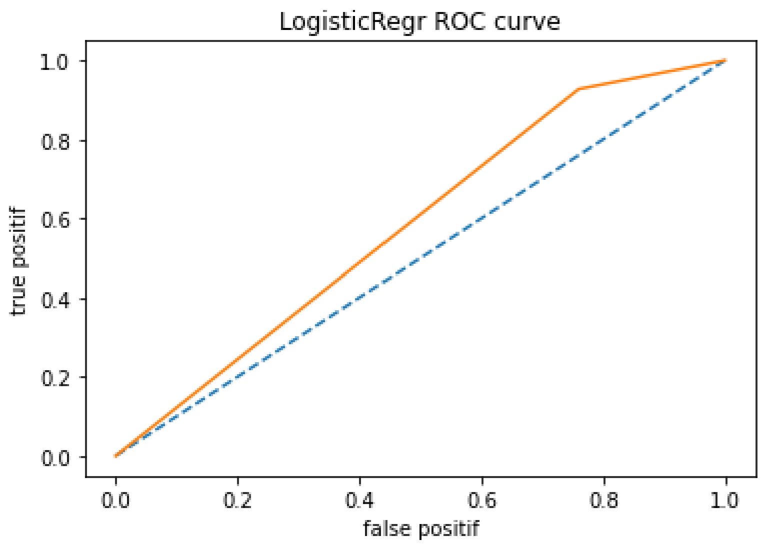
In [155…

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.71      0.24      0.36        50
         1.0       0.63      0.93      0.75        69

    accuracy                           0.64       119
   macro avg       0.67      0.58      0.55       119
weighted avg       0.66      0.64      0.58       119
```

In [156…

```
fpositif, tpositif, thresholds = roc_curve(y_test, y_pred)
plt.plot([0,1],[0,1],'--')
plt.plot(fpositif,tpositif, label='LogisticRegr')
plt.xlabel('false positif')
plt.ylabel('true positif')
plt.title('LogisticRegr ROC curve')
p=plt.show()
```

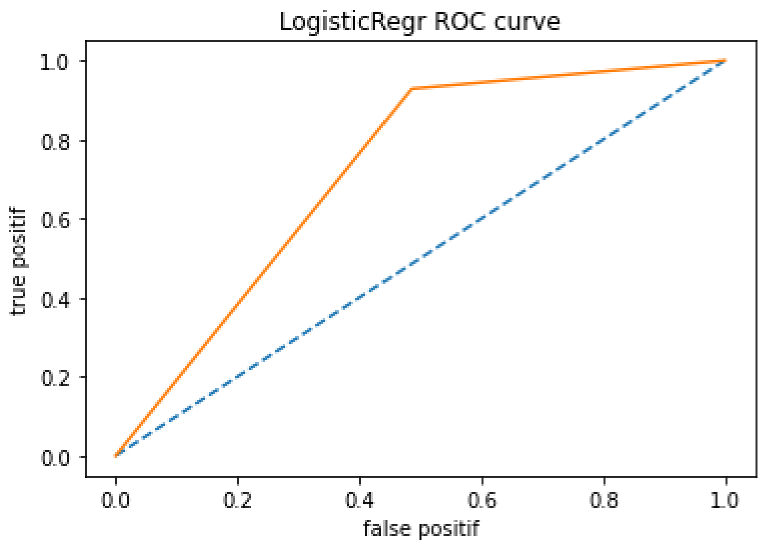


In [157…

```
max_iteration = 0
maxF1 = 0
maxAccuracy = 0
optimal_state = 0
import random
for k in range(max_iteration):
    print ('Iteration :'+str(k)+', Current accuracy: '+str(maxAccuracy)+ ', Current f1
    split_state = np.random.randint(1,100000000)-1
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=sp
    logisticRegr = LogisticRegression(C=1)
    logisticRegr.fit(x_train,y_train)
    y_pred=logisticRegr.predict(x_test)
    f1 = f1_score(y_test, y_pred, average='macro')
    accuracy = accuracy_score(y_test, y_pred)*100

    if (accuracy>maxAccuracy and f1>maxF1):
        maxF1 = f1
        maxAccuracy = accuracy
        optimal_state = split_state


optimal_state = 85491961
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=optima
logisticRegr = LogisticRegression(C=1)
logisticRegr.fit(x_train,y_train)
y_pred=logisticRegr.predict(x_test)
f1 = f1_score(y_test, y_pred, average='macro')
accuracy = accuracy_score(y_test, y_pred)*100
print('\n\n\n*Accuracy is: '+str(accuracy)+'\n*f1 score is: ',f1)

yt_lg,yp_lg = y_test,y_pred
#ploting the roc_curve

print ( '\n\n *the ROC curve: ')

fpositif, tpositif, thresholds = roc_curve(y_test, y_pred)
plt.plot([0,1],[0,1],'--')
plt.plot(fpositif,tpositif, label='LogisticRegr')
plt.xlabel('false positif')
plt.ylabel('true positif')
plt.title('LogisticRegr ROC curve')
```

```
p=plt.show()


#visualizig the confusion matrix:

print (' *the confusion matrix ')

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm,annot=True)
```

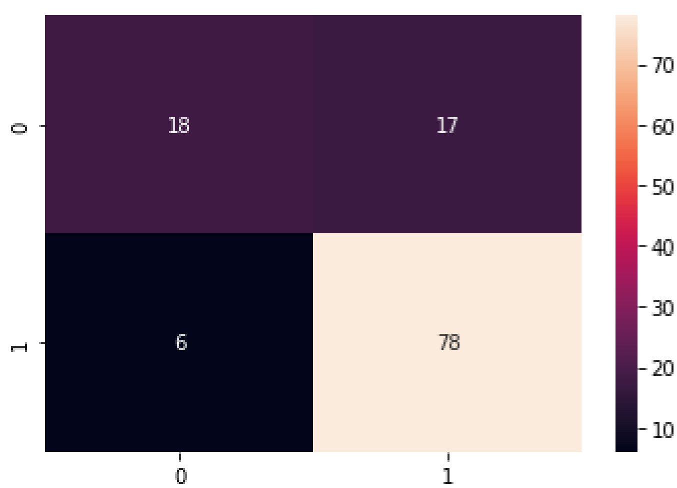
```
*Accuracy is: 80.67226890756302
*f1 score is:  0.7408389357068459


 *the ROC curve:
```




```
 *the confusion matrix
```

Out[157]:

```
<AxesSubplot:>
```



In [158…

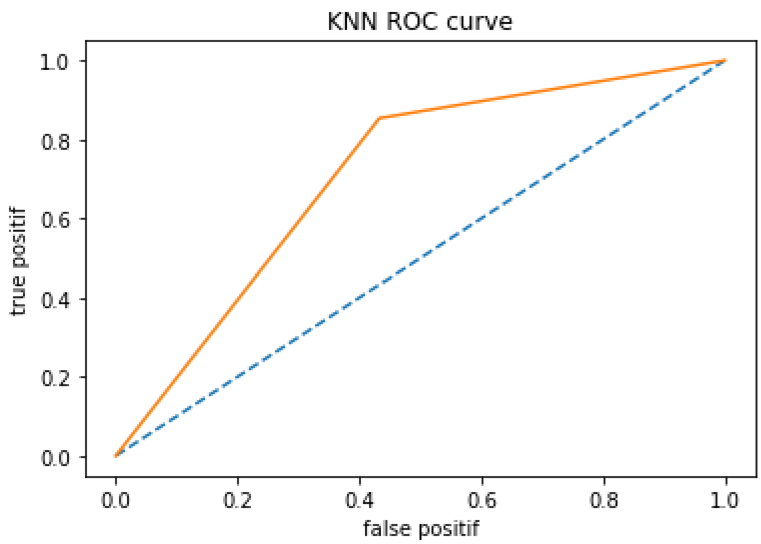
```
#define data
y=df.passed
target=["passed"]
x = df.drop(target,axis = 1 )
```

In [159…

```
max_iteration = 0
```

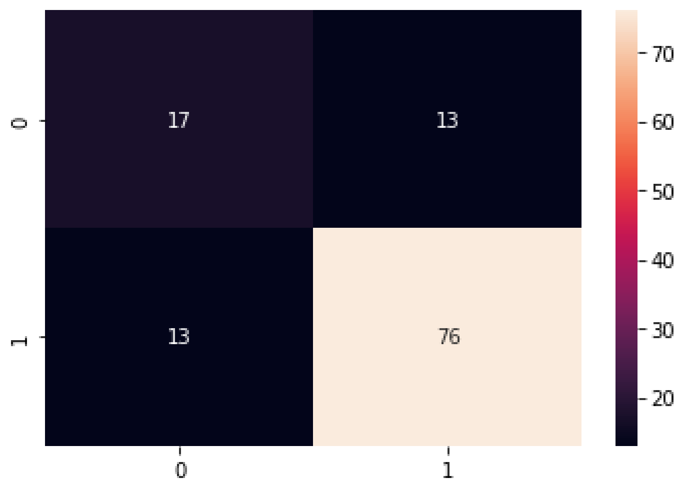
```
maxF1 = 0
maxAccuracy = 0
optimal_state = 0
for k in range(max_iteration):
    print ('Iteration :'+str(k)+', Current accuracy: '+str(maxAccuracy)+ ', Current f1
    split_state = np.random.randint(1,100000000)-1
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=sp
    KNN = KNeighborsClassifier()
    KNN.fit(x_train,y_train)
    y_pred=KNN.predict(x_test)
    f1 = f1_score(y_test, y_pred, average='macro')
    accuracy = accuracy_score(y_test, y_pred)*100

    if (accuracy>maxAccuracy and f1>maxF1):
        maxF1 = f1
        maxAccuracy = accuracy
        optimal_state = split_state

optimal_state = 71027464

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=optima
KNN= KNeighborsClassifier()
KNN.fit(x_train,y_train)
y_pred=KNN.predict(x_test)
f1 = f1_score(y_test, y_pred, average='macro')
accuracy = accuracy_score(y_test, y_pred)*100
print('\n\n\n*Accuracy is: '+str(accuracy)+'\n*f1 score is: ',f1)

print ('random_state is ',optimal_state)


#ploting the roc_curve

print ( '\n\n *the ROC curve: ')

fpositif, tpositif, thresholds = roc_curve(y_test, y_pred)
plt.plot([0,1],[0,1],'--')
plt.plot(fpositif,tpositif, label='knn')
plt.xlabel('false positif')
plt.ylabel('true positif')
plt.title('KNN ROC curve')
p=plt.show()

yt_knn,yp_knn= y_test,y_pred
#visualizig the confusion matrix:

print (' *the confusion matrix ')

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm,annot=True)
```

```
*Accuracy is: 78.15126050420169
*f1 score is:  0.7102996254681648
random_state is  71027464


 *the ROC curve:
```




```
 *the confusion matrix
```

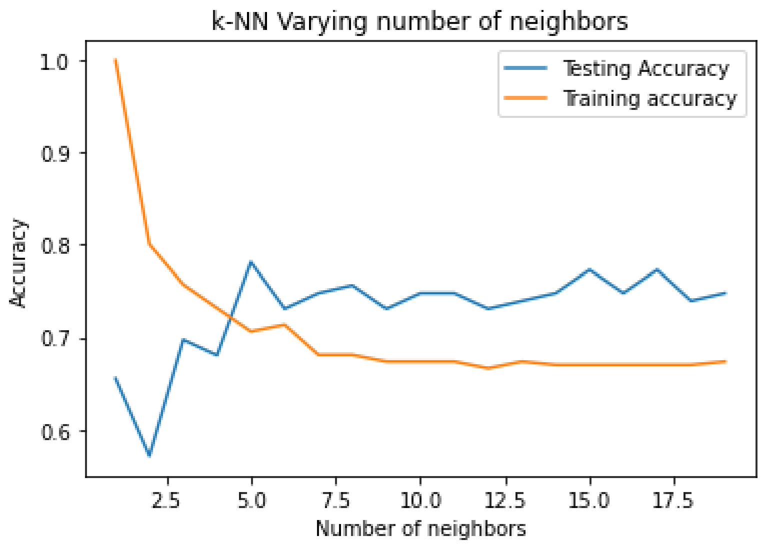
Out[159]: <AxesSubplot:>



In [160…

```
#Setup arrays to store training and test accuracies
neighbors= np.arange(1,20)
train_accuracy =np.empty(19)
test_accuracy = np.empty(19)

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(x_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(x_train, y_train)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(x_test, y_test)

#  Plotting the curv
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
```

```
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



In [161…

```
#In case of classifier like knn the parameter to be tuned is n_neighbors
param_grid = {'n_neighbors':np.arange(1,20)}
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(x_train,y_train)
#best score\n",
knn_cv.best_score_
```

Out[161]: 0.6449350649350649

In [162…

```
knn_cv.best_params_
```

Out[162]: {'n_neighbors': 19}

In [163…

```
param_grid = {'n_neighbors':np.arange(1,20)}
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(x_test,y_test)
#best score\n",
knn_cv.best_score_
```

Out[163]: 0.7728260869565217

In [164…

```
knn_cv.best_params_
```

Out[164]: {'n_neighbors': 13}

In [165…

```
param_grid = {'n_neighbors':np.arange(1,20)}
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(x,y)
#best score\n",
knn_cv.best_score_
```

Out[165]: 0.6734177215189873

In [166…

```
knn_cv.best_params_
```

Out[166]:

```
{'n_neighbors': 7}
```

In [167…

```
params = {"n_neighbors":[7,19] , "metric":["euclidean", "manhattan", "chebyshev"]}
acc = {}

for m in params["metric"]:
    acc[m] = []
    for k in params["n_neighbors"]:
        print("Model_{} metric: {}, n_neighbors: {}".format(i, m, k))
        i += 1
        t = time()
        knn = KNeighborsClassifier(n_neighbors=k, metric=m)
        knn.fit(x_train,y_train)
        pred = knn.predict(x_test)
        print("Time: ", time() - t)
        acc[m].append(accuracy_score(y_test, y_pred))
        print("Acc: ", acc[m][-1])
```

```
Model_18 metric: euclidean, n_neighbors: 7
Time:  0.0106048583984375
Acc:  0.7815126050420168
Model_19 metric: euclidean, n_neighbors: 19
Time:  0.0
Acc:  0.7815126050420168
Model_20 metric: manhattan, n_neighbors: 7
Time:  0.0
Acc:  0.7815126050420168
Model_21 metric: manhattan, n_neighbors: 19
Time:  0.0
Acc:  0.7815126050420168
Model_22 metric: chebyshev, n_neighbors: 7
Time:  0.0
Acc:  0.7815126050420168
Model_23 metric: chebyshev, n_neighbors: 19
Time:  0.0
Acc:  0.7815126050420168
```

In [168…

```
max_iteration = 0
maxF1 = 0
maxAccuracy = 0
optimal_state = 0
f1 = 0
accuracy = 0
True60 = False
for k in range(max_iteration):
    print ('Iteration :'+str(k)+', Current accuracy: '+str(maxAccuracy)+ ', Current f1
    split_state = np.random.randint(1,100000000)-1
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=sp
    KNN = KNeighborsClassifier(n_neighbors=7,metric='chebyshev')
    KNN.fit(x_train,y_train)
    y_pred=KNN.predict(x_test)
    f1 = f1_score(y_test, y_pred, average='macro')
    accuracy = accuracy_score(y_test, y_pred)*100

    if accuracy>maxAccuracy and f1>=0.5:
        maxF1 = f1
        maxAccuracy = accuracy
```

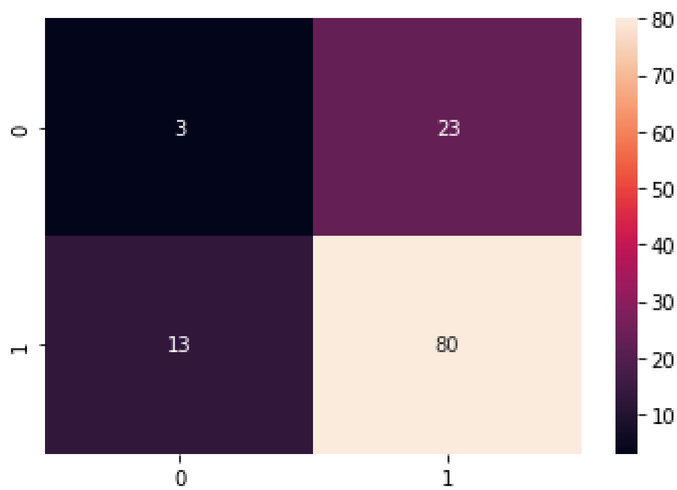
```
        optimal_state = split_state
        if maxAccuracy>79:
            break

optimal_state = 29300362
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=optima
KNN_f= KNeighborsClassifier(n_neighbors=7,metric='chebyshev')
KNN_f.fit(x_train,y_train)
y_pred=KNN_f.predict(x_test)
f1 = f1_score(y_test, y_pred, average='macro')
accuracy = accuracy_score(y_test, y_pred)*100
print('\n\n\n*Accuracy is: '+str(accuracy)+'\n*f1 score is: ',f1)

print ('random_state is ',optimal_state)

yt_knn,yp_knn= y_test,y_pred
```

```
*Accuracy is: 69.74789915966386
*f1 score is:  0.47959183673469385
random_state is  29300362
```

In [169…

```
ac = accuracy_score(yt_knn,yp_knn)
print('Accuracy is: ',ac)
cm= confusion_matrix(yt_knn,yp_knn)
sns.heatmap(cm,annot=True)
yt_knn,yp_knn = y_test,y_pred
```

```
Accuracy is:  0.6974789915966386
```



In [170…

```
print(classification_report(y_test,y_pred))
```

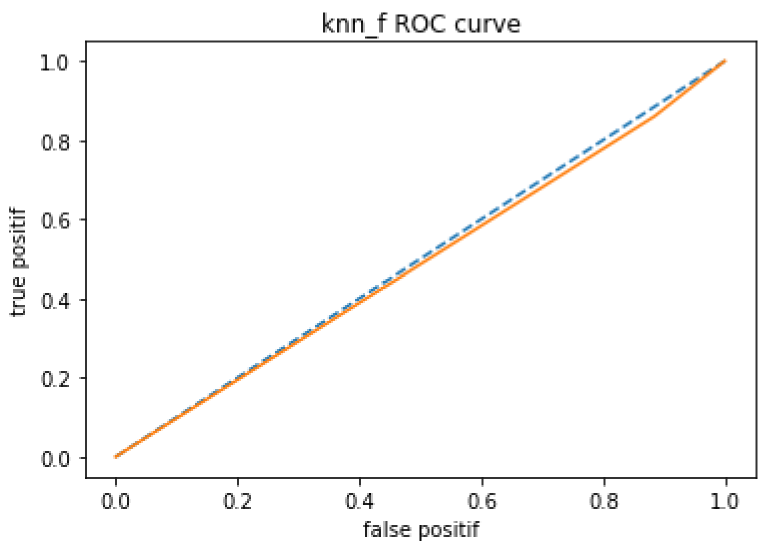
```
              precision    recall  f1-score   support

         0.0       0.19      0.12      0.14        26
         1.0       0.78      0.86      0.82        93

    accuracy                           0.70       119
   macro avg       0.48      0.49      0.48       119
weighted avg       0.65      0.70      0.67       119
```

In [171…

```
#ploting the roc_curve
```

```python
print ( ' the ROC curve: ')

fpositif, tpositif, thresholds = roc_curve(y_test, y_pred)
plt.plot([0,1],[0,1],'--')
plt.plot(fpositif,tpositif, label='final knn model')
plt.xlabel('false positif')
plt.ylabel('true positif')
plt.title('knn_f ROC curve')
p=plt.show()
```

```
the ROC curve:
```



In [172…

```python
# ------------------------------------------------------------------------------------------------
# Show results of every model

def showResults(accuracy, trainingTime, y_pred,model):

    print('------------------------------------------------Results :',model,'---------
    confusionMatrix = confusion_matrix(y_test, y_pred)
    print('\n The ROC curve is :\n')
    fig, _ = plt.subplots()
    fpr,tpr,thresholds=roc_curve(y_test,y_pred)
    plt.plot([0, 1],[0, 1],'--')
    plt.plot(fpr,tpr,label=model)
    plt.xlabel('false positive')
    plt.ylabel('false negative')
    plt.legend()
    fig.suptitle('ROC curve: '+str(model))
    plt.show()

    print('-----------------------------------------------')
    print('The model  accuracy:', round(accuracy),'%')
    print('-----------------------------------------------')
    print('The training time is: ',trainingTime)
    print('-----------------------------------------------')
    print('The f1 score is :',round(100*f1_score(y_test, y_pred, average='macro'))/100
    print('-----------------------------------------------')
    print('The roc_auc_score is :',round(100*roc_auc_score(y_test, y_pred))/100)
    print('-----------------------------------------------')
    print('The confusion matrix is :\n')
    ax = plt.axes()
    sns.heatmap(confusionMatrix,annot=True)
```

```
# ---------------------------------------------------------------------------------
# Hyperparameter Tuning :
# C, degree and gamma are the parameters that are used in SVM classffier 'svc(C=..,..)
# The following functions will return those values that minimize the error on (X_val,y
# So this (X_val,y_val) set will be used to get the optimal SVM parameters before eval


# Optimal C
def optimal_C_value():
    Ci = np.array(( 0.0001,0.001,0.01,0.05,0.1,4,10,40,100))
    minError = float('Inf')
    optimal_C = float('Inf')

    for c in Ci:
        clf = SVC(C=c,kernel='linear')
        clf.fit(X_train, y_train)
        predictions = clf.predict(X_val)
        error = np.mean(np.double(predictions != y_val))
        if error < minError:
            minError = error
            optimal_C = c
    return optimal_C


# Optimal C and the degree of the polynomial
def optimal_C_d_values():
    Ci = np.array(( 0.0001,0.001,0.01,0.05,0.1,4,10,40,100))
    Di = np.array(( 2, 5, 10, 15, 20, 25, 30))
    minError = float('Inf')
    optimal_C = float('Inf')
    optimal_d = float('Inf')

    for d in Di:
        for c in Ci:
            clf = SVC(C=c,kernel='poly', degree=d)
            clf.fit(X_train, y_train)
            predictions = clf.predict(X_val)
            error = np.mean(np.double(predictions != y_val))
            if error < minError:
                minError = error
                optimal_C = c
                optimal_d = d
    return optimal_C,optimal_d


# Optimal C and gamma
def optimal_C_gamma_values():
    Ci = np.array(( 0.0001,0.001,0.01,0.05,0.1,4,10,40,100))
    Gi = np.array(( 0.000001,0.00001,0.01,1,2,3,5,20,70,100,500,1000))
    minError = float('Inf')
    optimal_C = float('Inf')
    optimal_g = float('Inf')

    for g in Gi:
        for c in Ci:
            clf = SVC(C=c,kernel='rbf', gamma=g)
            clf.fit(X_train, y_train)
            predictions = clf.predict(X_val)
```

```
            error = np.mean(np.double(predictions != y_val))
            if error < minError:
                minError = error
                optimal_C = c
                optimal_g = g
    return optimal_C,optimal_g


# -----------------------------------------------------------------------------
# Compare the three kernels


def compare_kernels():
    X_train1,X_val1,X_test1,y_train1,y_val1,y_test1 = split(df,rest_size=0.4,test_size
    X_train2,X_val2,X_test2,y_train2,y_val2,y_test2 = split(df,rest_size=0.4,test_size
    X_train3,X_val3,X_test3,y_train3,y_val3,y_test3 = split(df,rest_size=0.4,test_size
    print('------------------------------------------------ Comparison ---------------
    print('\n')
    f11 = "{:.2f}".format(f1_score(y_test1, y_linear, average='macro'))
    f22 = "{:.2f}".format(f1_score(y_test2, y_poly, average='macro'))
    f33 = "{:.2f}".format(f1_score(y_test3, y_gauss, average='macro'))
    roc1 = "{:.2f}".format(roc_auc_score(y_test1, y_linear))
    roc2 = "{:.2f}".format(roc_auc_score(y_test2, y_poly))
    roc3 = "{:.2f}".format(roc_auc_score(y_test3, y_gauss))
    a1,a2 = confusion_matrix(y_test1, y_linear)[0],confusion_matrix(y_test1, y_linear)
    b1,b2 = confusion_matrix(y_test2, y_poly)[0],confusion_matrix(y_test2, y_poly)[1]
    c1,c2 = confusion_matrix(y_test3, y_gauss)[0],confusion_matrix(y_test3, y_gauss)[1
    data_rows = [('training time',time1, time2, time3),
                 ('','','',''),
                  ('accuracy %',linear_accuracy, poly_accuracy, gauss_accuracy),
                 ('','','',''),
                 ('confusion matrix',a1, b1, c1),
                ('',a2,b2,c2),
                 ('','','',''),
                ('f1 score',f11,f22,f33),
                 ('','','',''),
                ('roc_auc_score',roc1,roc2,roc3)]
    t = Table(rows=data_rows, names=('metric','Linear kernel', 'polynomial kernel', 'g
    print(t)
    print('\n\n')
    print('The Roc curves :\n')
    y_pred1 = y_linear
    y_pred2 = y_poly
    y_pred3 = y_gauss
    fig, _ = plt.subplots()
    fig.suptitle('Comparison of three ROC curves')
    fpr,tpr,thresholds=roc_curve(y_test1,y_pred1)
    plt.plot([0, 1],[0, 1],'--')
    plt.plot(fpr,tpr,label='Linear kernel :'+str(roc1))
    plt.xlabel('false positive')
    plt.ylabel('false negative')
    fpr,tpr,thresholds=roc_curve(y_test2,y_pred2)
    plt.plot(fpr,tpr,label='Polynomial kernel :'+str(roc2))
    fpr,tpr,thresholds=roc_curve(y_test3,y_pred3)
    plt.plot(fpr,tpr,label='Gaussian kernel :'+str(roc3))
    plt.legend()
    plt.show()


# -----------------------------------------------------------------------------
```

```python
# Print results of the choosen kernel

def best_kernel(kernel):
    X_train1,X_val1,X_test1,y_train1,y_val1,y_test1 = split(df,rest_size=0.4,test_size
    X_train2,X_val2,X_test2,y_train2,y_val2,y_test2 = split(df,rest_size=0.4,test_size
    X_train3,X_val3,X_test3,y_train3,y_val3,y_test3 = split(df,rest_size=0.4,test_size

    time = 0
    f1 = 0
    accuracy = 0
    rc = 0
    y = 0
    if kernel == 'linear kernel':
        time = time1
        f1 = "{:.2f}".format(f1_score(y_test1, y_linear, average='macro'))
        accuracy = round(100*linear_accuracy)/100
        rc = round(100*roc_auc_score(y_test1, y_linear))/100
        y_test = y_test1
        y = y_linear
    elif kernel == 'polynomial kernel':
        time = time2
        f1 = "{:.2f}".format(f1_score(y_test2, y_poly, average='macro'))
        accuracy = round(100*poly_accuracy)/100
        rc = round(100*roc_auc_score(y_test2, y_poly))/100
        y_test = y_test2
        y = y_poly
    else :
        time = time3
        f1 = "{:.2f}".format(f1_score(y_test3, y_gauss, average='macro'))
        accuracy = round(100*gauss_accuracy)/100
        rc = round(100*roc_auc_score(y_test3, y_gauss))/100
        y_test = y_test3
        y = y_gauss

    # used for comparing three classfiers(knn, logistic regression and svm)
    yt_svm,yp_svm = y_test, y

    print('The choosen kernel :',kernel)
    print('the training :',time)
    print('the accuracy :',round(accuracy),'%')
    print('the f1 score :',f1)
    print('The roc_auc_score is :',rc)
    print('----------------------------------------\nThe ROC curve :')
    fig, _ = plt.subplots()
    fpr,tpr,thresholds=roc_curve(y_test,y)
    plt.plot([0, 1],[0, 1],'--')
    plt.plot(fpr,tpr,label=kernel+': '+str(rc))
    plt.xlabel('false positive')
    plt.ylabel('false negative')
    plt.legend()
    plt.show()
    confusionMatrix = confusion_matrix(y_test, y)
    print('----------------------------------------\nThe confusion matrix is  :')
    ax = plt.axes()
    sns.heatmap(confusionMatrix,annot=True)
    ax.set_title('Confusion matrix of SVM '+str(kernel))
    return yt_svm,yp_svm

# ---------------------------------------------------------------------------------
# svm factor : factor affecting students performance, later on on this Ipython noteboo
```

```
# 1) factor as svm coefficients
def factors(array, K, max_or_min, df):

    n = array.shape[1]
    array = array.reshape(n,1)
    my_list = array.tolist()

    if max_or_min == 'max':
        temp = sorted(my_list)[-K:]
        res = []
        for ele in temp:
            res.append(my_list.index(ele))
        return(get_factors(res, df))


    elif max_or_min == 'min':
        temp = sorted(my_list, reverse=True)[-K:]
        temp = temp = np.array(temp).reshape(K,1)
        res = []
        for ele in temp:
            if ele<0:
                res.append(my_list.index(ele))
        return(get_factors(res, df))


    else:
        return


# 2) converts those factors to dataset columns name
def get_factors(index, df):
    f = []
    for i in index:
        f.append(df.columns[i])
    return f


# 3) Convert column names to understandable string

columns_name = {'famsize': 'family size', 'Pstatus': "parent's cohabitation status ",
                'Fedu': "father's education", 'Mjob': "mother's job", 'Fjob': "father'
                'reason': 'reason to choose this school ','schoolsup': 'extra educatic
                'paid': 'extra paid classes within the course subject', 'higher': 'war
                'romantic': 'with a romantic relationship ', 'famrel': 'quality of fam
                'Dalc': 'workday alcohol consumption', 'Walc': 'weekend alcohol consum


def column_to_string(fcts,max_or_min):

    if max_or_min == 'max':
        print('-----------------------------------------------------------------------
        print('Factors helping students succeed :')
    else:
        print('-----------------------------------------------------------------------
        print('-----------------------------------------------------------------------
        print('Factors leading students to failure')

    for fct in fcts:
```

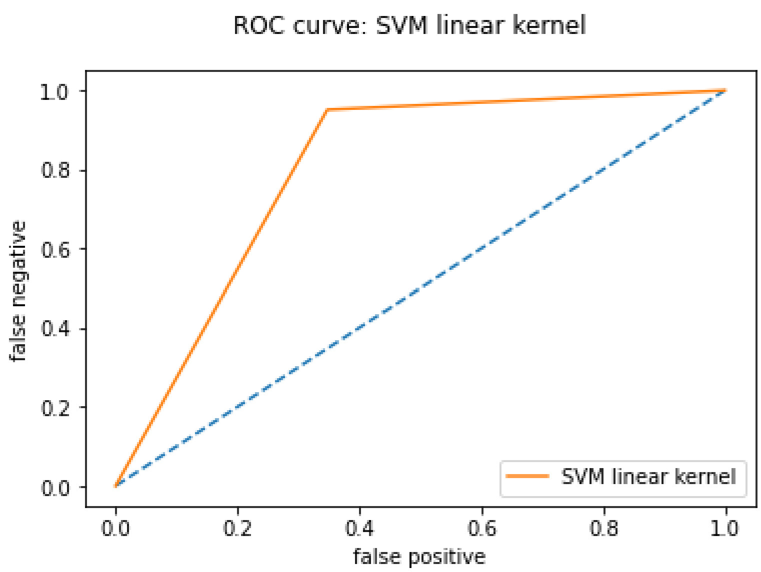
```
        if fct in columns_name:
            print(columns_name[fct])
        else:
            print(fct)



# -----------------------------------------------------------------------------
# Splitting the data for SVM
# Here We will split data into test set, cross validation (X_val, y_val) set and trair
# The cross validation (X_val, y_val) is used for choosing the optimal value for svm p

def split(df,rest_size,test_size,randomState):
    data = df.to_numpy()
    n = data.shape[1]
    x = data[:,0:n-1]
    y = data[:,n-1]
    if(randomState):
        X_train,X_rest,y_train,y_rest = train_test_split(x,y,test_size=rest_size,rando
        X_val,X_test,y_val,y_test = train_test_split(X_rest,y_rest,test_size=test_size
    else:
        X_train,X_rest,y_train,y_rest = train_test_split(x,y,test_size=rest_size,rando
        X_val,X_test,y_val,y_test = train_test_split(X_rest,y_rest,test_size=test_size

    return X_train,X_val,X_test,y_train,y_val,y_test
```

In [173…

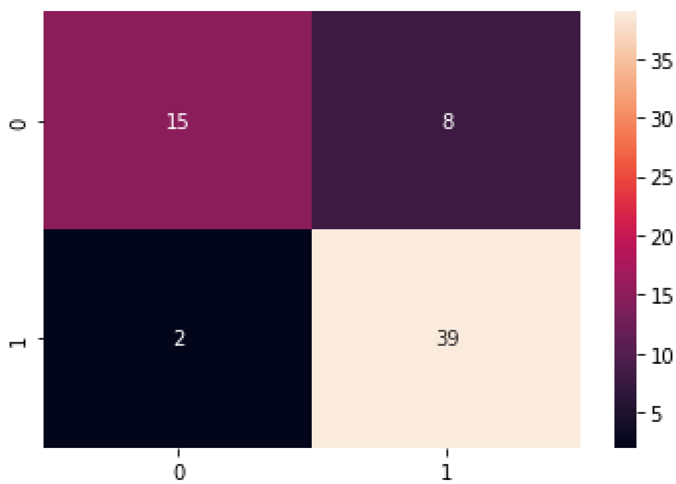
```
##################################################### Linear kernel #################
optimal_split_state1 = 0
maxAccuracy = 0
maxF1 = 0

# We already tune parameters, we do not need to loop over all the hyperparamters agair
# if you want to do so just set max_iteration to 2000 for example
# and remove the line 'optimal_split_state = 388628375' at the bottom of this cell.

max_iteration = 0
if max_iteration != 0:
    print ('---------------------------------------Hyperparameters tunning starts----

for k in range(max_iteration):
    print ('Iteration :'+str(k)+', Current accuracy: '+str(maxAccuracy)+' Current f1 '
    # Let's get the optimal C value for the linear kernal
    split_state = np.random.randint(1,1000000000)-1
    X_train,X_val,X_test,y_train,y_val,y_test = split(df,rest_size=0.4,test_size=0.4,r
    optimal_C = optimal_C_value()


    # Now let's use the optimal C value
    linear_clf = SVC(C=optimal_C,kernel='linear')

    # Let's train the model with the optimal C value and calculate the training time
    tic = time()
    linear_clf.fit(X_train, y_train)
    toc = time()
    time1 = str(round(1000*(toc-tic))) + "ms"
    y_linear = linear_clf.predict(X_test)
    linear_f1 = f1_score(y_test, y_linear, average='macro')
    linear_accuracy = accuracy_score(y_test, y_linear)*100
    if linear_accuracy>maxAccuracy and linear_f1>maxF1:
        maxAccuracy = linear_accuracy
```

```
        maxF1 = linear_f1
        optimal_split_state1 = split_state
    if maxAccuracy>86 and maxF1>80:
        break;

# We've already tuned our hyperparameters, we will not repeat that again as it takes s
# The optimal split state for linear kernel is 388628375
# Let's try that split state
optimal_split_state1 = 388628375
X_train,X_val,X_test,y_train,y_val,y_test = split(df,rest_size=0.4,test_size=0.4,rando
optimal_C = optimal_C_value()


# Now let's use the optimal C value
linear_clf = SVC(C=optimal_C,kernel='linear')

# Let's train the model with the optimal C value and calculate the training time
tic = time()
linear_clf.fit(X_train, y_train)
toc = time()
time1 = str(round(1000*(toc-tic))) + "ms"
y_linear = linear_clf.predict(X_test)
linear_accuracy = accuracy_score(y_test, y_linear)*100
if max_iteration != 0:
    print('\n\n\n                                ---------------------------process ended'
         '-------------------------------------                              \n\n\n')

# Let's show the resuls
showResults(linear_accuracy, time1, y_linear,'SVM linear kernel')
```

```
------------------------------------------------Results : SVM linear kernel ---------
----------------------------------------

 The ROC curve is :
```



```
---------------------------------------------
The model  accuracy: 84 %
---------------------------------------------
The training time is:  16ms
---------------------------------------------
The f1 score is : 0.82
---------------------------------------------
The roc_auc_score is : 0.8
---------------------------------------------
The confusion matrix is :
```

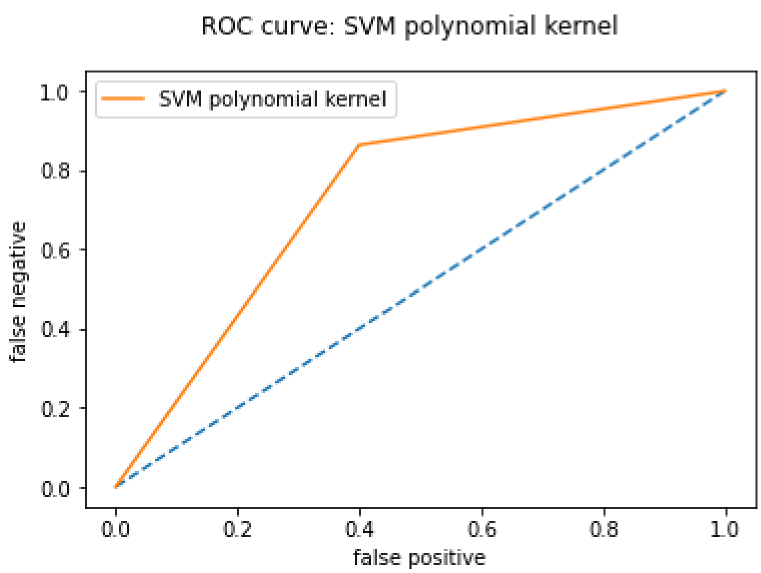


In [174…

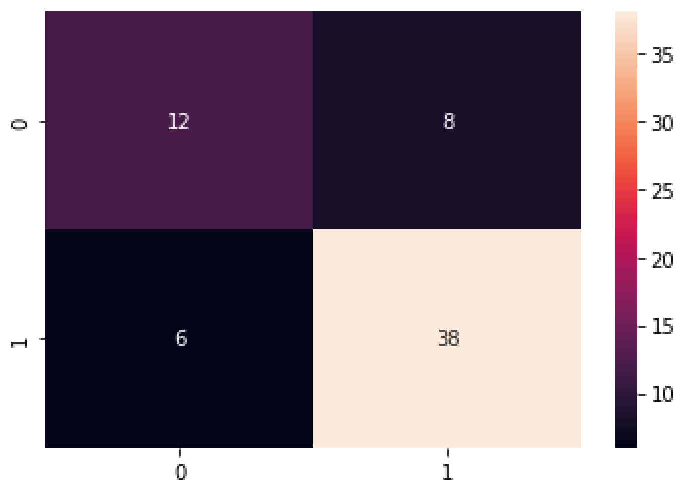
```
####################################################### Polynomial kernel ############
optimal_split_state2 = 0
maxAccuracy = 0
maxF1 = 0


# We already tune parameters, we do not need to loop over all the hyperparamters agair
# if you want to do so just set max_iteration to 500 for example
# and remove the line 'optimal_split_state2 = 7070621' at the bottom of this cell.

max_iteration = 0
if max_iteration != 0:
    print ('----------------------------------------Hyperparameters tunning starts----
for k in range(max_iteration):
    print ('Iteration :'+str(k)+', Current accuracy: '+str(maxAccuracy)+', Current f1

    split_state = np.random.randint(1,100000000)-1
    X_train,X_val,X_test,y_train,y_val,y_test = split(df,rest_size=0.4,test_size=0.4,r

    # Let's get the optimal C and the degree value for the polynomial kernal
    optimal_C, optimal_d = optimal_C_d_values()

    # Now let's use the optimal c value and the optimal degree value
    poly_clf = SVC(C=optimal_C,kernel='poly', degree=optimal_d)

    # Let's train the model with the optimal C value
    poly_clf.fit(X_train, y_train)
    y_poly = poly_clf.predict(X_test)
    poly_f1 = f1_score(y_test, y_poly, average='macro')
    poly_accuracy = accuracy_score(y_test, y_poly)*100

    if poly_accuracy>maxAccuracy and poly_f1>maxF1:
```

```
        maxAccuracy = poly_accuracy
        maxF1 = poly_f1
        optimal_split_state2 = split_state

# We've already tuned our hyperparameters, we will not repeat that again as it takes s
# The optimal split state for polynomial kernel is 7070621
# Let's try that split state
optimal_split_state2 = 7070621

X_train,X_val,X_test,y_train,y_val,y_test = split(df,rest_size=0.4,test_size=0.4,rando

optimal_C, optimal_d = optimal_C_d_values()


# Now let's use the optimal C value
poly_clf = SVC(C=optimal_C,kernel='poly', degree=optimal_d)

# Let's train the model and calculate the training time
tic = time()
poly_clf.fit(X_train, y_train)
toc = time()
time2 = str(round(1000*(toc-tic))) + "ms"
y_poly = poly_clf.predict(X_test)
poly_accuracy = accuracy_score(y_test, y_poly)*100
if max_iteration != 0:
    print('\n\n\n                            ---------------------------process ended'
         '-------------------------------------                            \n\n\n')

# Let's show the resuls
showResults(poly_accuracy, time2, y_poly,'SVM polynomial kernel')
```

```
------------------------------------------------Results : SVM polynomial kernel -----
--------------------------------------------

 The ROC curve is :
```



```
---------------------------------------------
The model  accuracy: 78 %
---------------------------------------------
The training time is:  16ms
---------------------------------------------
The f1 score is : 0.74
---------------------------------------------
The roc_auc_score is : 0.73
---------------------------------------------
The confusion matrix is :
```

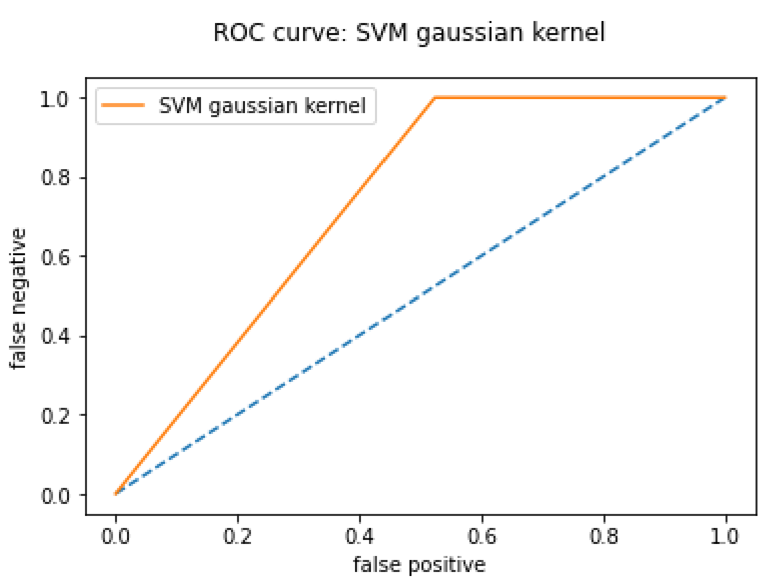


In [175…

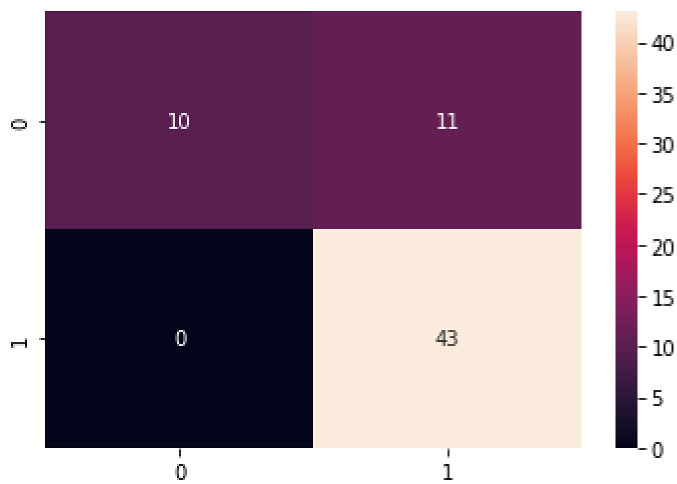
```
######################################################## Gaussian kernel ###############
optimal_split_state3 = 0
maxAccuracy = 0
maxF1 = 0


# We already tune parameters, we do not need to loop over all the hyperparamters agai
# if you want to do so just set max_iteration to 500 for example
# and remove the line 'optimal_split_state3 = 93895097' at the bottom of this cell.

max_iteration = 0
if max_iteration != 0:
    print ('---------------------------------------------Hyperparameters tunning sta
    '---------------------------------------------\n\n')
for k in range(max_iteration):
    print ('Iteration :'+str(k)+', Current accuracy: '+str(maxAccuracy)+', Current f1

    split_state = np.random.randint(1,100000000)-1
    X_train,X_val,X_test,y_train,y_val,y_test = split(df,rest_size=0.4,test_size=0.4,

    # Let's get the optimal C and the degree value for the polynomial kernal
    optimal_C, optimal_gamma = optimal_C_gamma_values()

    # Now let's use the optimal c value and the optimal degree value
    gauss_clf = SVC(C=optimal_C,kernel='rbf',gamma=optimal_gamma)

    # Let's train the model with the optimal C value
    gauss_clf.fit(X_train, y_train)
    y_gauss = gauss_clf.predict(X_test)
    gauss_f1 = f1_score(y_test, y_gauss, average='macro')
    gauss_accuracy = accuracy_score(y_test, y_gauss)*100
```

```
    if gauss_accuracy>maxAccuracy and gauss_f1>maxF1:
        maxAccuracy = gauss_accuracy
        maxF1 = gauss_f1
        optimal_split_state3 = split_state

# We've already tuned our hyperparameters, we will not repeat that again as it takes s
# The optimal split state for polynomial kernel is 93895097
# Let's try that split state
optimal_split_state3 = 93895097

X_train,X_val,X_test,y_train,y_val,y_test = split(df,rest_size=0.4,test_size=0.4,rando

optimal_C, optimal_gamma = optimal_C_gamma_values()


# Now let's use the optimal C value
gauss_clf = SVC(C=optimal_C,kernel='rbf',gamma=optimal_gamma)

# Let's train the model and calculate the training time
tic = time()
gauss_clf.fit(X_train, y_train)
toc = time()
time3 = str(round(1000*(toc-tic))) + "ms"
y_gauss = gauss_clf.predict(X_test)
gauss_accuracy = (accuracy_score(y_test, y_gauss)*100)

if max_iteration != 0:
    print('\n\n\n                                  ---------------------------process ended'
         '------------------------------------                                  \n\n\n')

# Let's show the resuls
showResults(gauss_accuracy, time3, y_gauss,'SVM gaussian kernel')
```

```
------------------------------------------------Results : SVM gaussian kernel -------
------------------------------------------

 The ROC curve is :
```



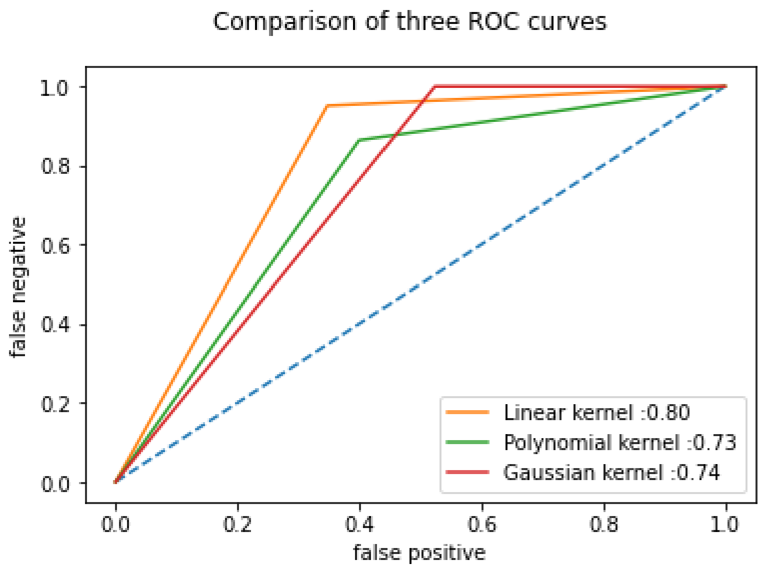
```
----------------------------------------------
The model  accuracy: 83 %
----------------------------------------------
The training time is:  8ms
----------------------------------------------
The f1 score is : 0.77
----------------------------------------------
The roc_auc_score is : 0.74
----------------------------------------------
The confusion matrix is :
```



In [176…

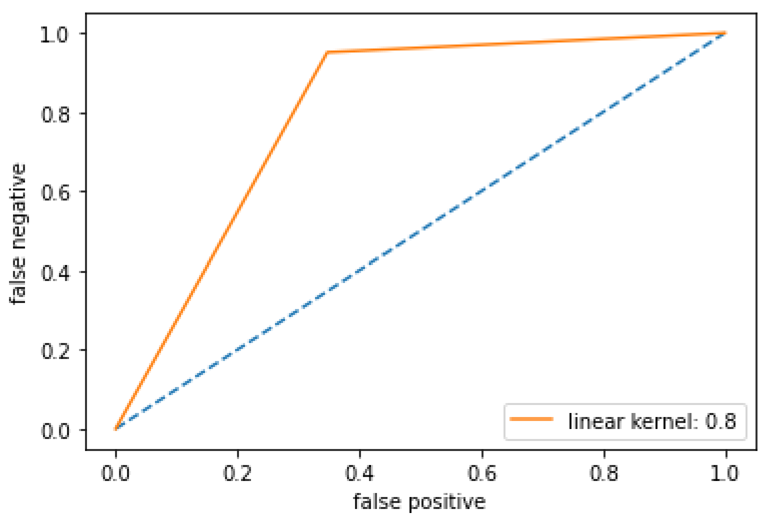
```python
compare_kernels()
```

```
------------------------------------------------ Comparison -------------------------
----------------------------


     metric       Linear kernel polynomial kernel gaussian kernel
---------------- ------------- ----------------- ---------------
   training time          16ms              16ms             8ms

      accuracy %        84.375            78.125         82.8125

confusion matrix       [15  8]           [12  8]         [10 11]
                       [ 2 39]           [ 6 38]         [ 0 43]

        f1 score          0.82              0.74            0.77

   roc_auc_score          0.80              0.73            0.74



The Roc curves :
```

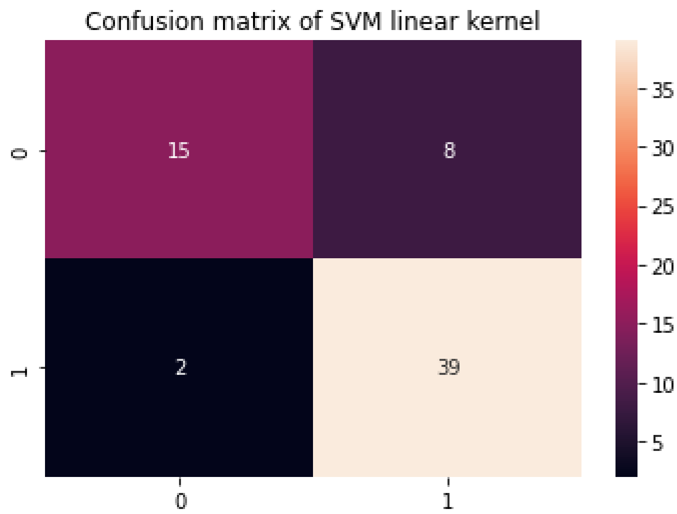


In [177…
```python
yt_svm,yp_svm = best_kernel('linear kernel')
```

```
The choosen kernel : linear kernel
the training : 16ms
the accuracy : 84 %
the f1 score : 0.82
The roc_auc_score is : 0.8
----------------------------------------
The ROC curve :
```




```
----------------------------------------
The confusion matrix is  :
```



In [178…

```
# Get svm parameters
coefs = linear_clf.coef_

# factors helping students to succeed
column_to_string(factors(coefs, 5, 'max', df),'max')

# factors leading students to failure
column_to_string(factors(coefs, 5, 'min', df), 'min')
```

```
--------------------------------------------------------------------------------
Factors helping students succeed :
father's education
guardian
wants to take higher education
studytime
father's job
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Factors leading students to failure
age
health
going out with friends
absences
failures
```

In [179…

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from time import time
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import confusion_matrix, roc_curve, accuracy_score, f1_score, roc
from astropy.table import Table
from sklearn.metrics import roc_auc_score

df = pd.read_csv('student-data.csv')
dfv = pd.read_csv('student-data.csv')
```

In [180…

```
def numerical_data():
```

```
    df['school'] = df['school'].map({'GP': 0, 'MS': 1})
    df['sex'] = df['sex'].map({'M': 0, 'F': 1})
    df['address'] = df['address'].map({'U': 0, 'R': 1})
    df['famsize'] = df['famsize'].map({'LE3': 0, 'GT3': 1})
    df['Pstatus'] = df['Pstatus'].map({'T': 0, 'A': 1})
    df['Mjob'] = df['Mjob'].map({'teacher': 0, 'health': 1, 'services': 2, 'at_home':
    df['Fjob'] = df['Fjob'].map({'teacher': 0, 'health': 1, 'services': 2, 'at_home':
    df['reason'] = df['reason'].map({'home': 0, 'reputation': 1, 'course': 2, 'other':
    df['guardian'] = df['guardian'].map({'mother': 0, 'father': 1, 'other': 2})
    df['schoolsup'] = df['schoolsup'].map({'no': 0, 'yes': 1})
    df['famsup'] = df['famsup'].map({'no': 0, 'yes': 1})
    df['paid'] = df['paid'].map({'no': 0, 'yes': 1})
    df['activities'] = df['activities'].map({'no': 0, 'yes': 1})
    df['nursery'] = df['nursery'].map({'no': 0, 'yes': 1})
    df['higher'] = df['higher'].map({'no': 0, 'yes': 1})
    df['internet'] = df['internet'].map({'no': 0, 'yes': 1})
    df['romantic'] = df['romantic'].map({'no': 0, 'yes' : 1})
    df['passed'] = df['passed'].map({'no': 0, 'yes': 1})
    # reorder dataframe columns :
    col = df['passed']
    del df['passed']
    df['passed'] = col


# feature scaling will allow the algorithm to converge faster, large data will have sc
def feature_scaling(df):
    for i in df:
        col = df[i]
        # let's choose columns that have large values
        if(np.max(col)>6):
            Max = max(col)
            Min = min(col)
            mean = np.mean(col)
            col  = (col-mean)/(Max)
            df[i] = col
        elif(np.max(col)<6):
            col = (col-np.min(col))
            col /= np.max(col)
            df[i] = col
```

In [181…

```
numerical_data()
df
```

Out[181]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	romantic
0	0	1	18	0	1	1	4	4	3	0	...	0	0
1	0	1	17	0	1	0	1	1	3	4	...	1	0
2	0	1	15	0	0	0	1	1	3	4	...	1	0
3	0	1	15	0	1	0	4	2	1	2	...	1	1
4	0	1	16	0	1	0	3	3	4	4	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
390	1	0	20	0	0	1	2	2	2	2	...	0	0
391	1	0	17	0	0	0	3	1	2	2	...	1	0
392	1	0	21	1	1	0	1	1	4	4	...	0	0
393	1	0	18	1	0	0	3	2	2	4	...	1	0
394	1	0	19	0	0	0	1	1	4	3	...	1	0

395 rows × 31 columns

In [182...

```
# Let's scal our features
feature_scaling(df)

# Now we are ready for models training
df
```

Out[182]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	roma
0	0.0	1.0	0.059264	0.0	1.0	1.0	1.00	1.00	0.75	0.00	...	0.0	
1	0.0	1.0	0.013809	0.0	1.0	0.0	0.25	0.25	0.75	1.00	...	1.0	
2	0.0	1.0	-0.077100	0.0	0.0	0.0	0.25	0.25	0.75	1.00	...	1.0	
3	0.0	1.0	-0.077100	0.0	1.0	0.0	1.00	0.50	0.25	0.50	...	1.0	
4	0.0	1.0	-0.031646	0.0	1.0	0.0	0.75	0.75	1.00	1.00	...	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
390	1.0	0.0	0.150173	0.0	0.0	1.0	0.50	0.50	0.50	0.50	...	0.0	
391	1.0	0.0	0.013809	0.0	0.0	0.0	0.75	0.25	0.50	0.50	...	1.0	
392	1.0	0.0	0.195627	1.0	1.0	0.0	0.25	0.25	1.00	1.00	...	0.0	
393	1.0	0.0	0.059264	1.0	0.0	0.0	0.75	0.50	0.50	1.00	...	1.0	
394	1.0	0.0	0.104718	0.0	0.0	0.0	0.25	0.25	1.00	0.75	...	1.0	

395 rows × 31 columns

In [183...

```
df.dropna().shape
```

Out[183]: (395, 31)

In [184…
```
df.columns
```

Out[184]:
```
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences', 'passed'],
      dtype='object')
```

In [185…
```
features=['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences']
```

In [186…
```
X=df.drop('passed',axis='columns')
y=df['passed']
```

In [187…
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=5)
```

In [188…
```
import tensorflow as tf
from tensorflow import keras

model=keras.Sequential([
    keras.layers.Dense(25,input_shape=(30,),activation='relu'),
    keras.layers.Dense(20,activation='relu'),
    keras.layers.Dense(1,activation='sigmoid'),
])

model.compile(optimizer='RMSProp',
             loss='binary_crossentropy',
             metrics=['accuracy'])

model.fit(X_train,y_train,epochs=180)
```

```
Epoch 1/180
10/10 [==============================] - 1s 2ms/step - loss: 0.6522 - accuracy: 0.667
7
Epoch 2/180
10/10 [==============================] - 0s 1ms/step - loss: 0.6403 - accuracy: 0.670
9
Epoch 3/180
10/10 [==============================] - 0s 2ms/step - loss: 0.6317 - accuracy: 0.670
9
Epoch 4/180
10/10 [==============================] - 0s 2ms/step - loss: 0.6280 - accuracy: 0.670
9
Epoch 5/180
10/10 [==============================] - 0s 2ms/step - loss: 0.6215 - accuracy: 0.670
9
Epoch 6/180
10/10 [==============================] - 0s 2ms/step - loss: 0.6163 - accuracy: 0.670
9
Epoch 7/180
10/10 [==============================] - 0s 2ms/step - loss: 0.6125 - accuracy: 0.674
1
Epoch 8/180
10/10 [==============================] - 0s 2ms/step - loss: 0.6082 - accuracy: 0.674
1
Epoch 9/180
10/10 [==============================] - 0s 2ms/step - loss: 0.6034 - accuracy: 0.677
2
Epoch 10/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5987 - accuracy: 0.680
4
Epoch 11/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5940 - accuracy: 0.683
5
Epoch 12/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5875 - accuracy: 0.708
9
Epoch 13/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5869 - accuracy: 0.693
0
Epoch 14/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5797 - accuracy: 0.712
0
Epoch 15/180
10/10 [==============================] - 0s 1ms/step - loss: 0.5770 - accuracy: 0.715
2
Epoch 16/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5727 - accuracy: 0.715
2
Epoch 17/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5696 - accuracy: 0.718
4
Epoch 18/180
10/10 [==============================] - 0s 1ms/step - loss: 0.5630 - accuracy: 0.727
8
Epoch 19/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5587 - accuracy: 0.724
7
Epoch 20/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5560 - accuracy: 0.731
0
```

```
Epoch 21/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5516 - accuracy: 0.721
5
Epoch 22/180
10/10 [==============================] - 0s 1ms/step - loss: 0.5485 - accuracy: 0.734
2
Epoch 23/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5431 - accuracy: 0.727
8
Epoch 24/180
10/10 [==============================] - 0s 1ms/step - loss: 0.5426 - accuracy: 0.727
8
Epoch 25/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5383 - accuracy: 0.746
8
Epoch 26/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5318 - accuracy: 0.737
3
Epoch 27/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5301 - accuracy: 0.750
0
Epoch 28/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5265 - accuracy: 0.746
8
Epoch 29/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5241 - accuracy: 0.769
0
Epoch 30/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5183 - accuracy: 0.765
8
Epoch 31/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5178 - accuracy: 0.775
3
Epoch 32/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5129 - accuracy: 0.765
8
Epoch 33/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5102 - accuracy: 0.769
0
Epoch 34/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5086 - accuracy: 0.762
7
Epoch 35/180
10/10 [==============================] - 0s 1ms/step - loss: 0.5040 - accuracy: 0.778
5
Epoch 36/180
10/10 [==============================] - 0s 2ms/step - loss: 0.5042 - accuracy: 0.772
2
Epoch 37/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4964 - accuracy: 0.781
6
Epoch 38/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4969 - accuracy: 0.775
3
Epoch 39/180
10/10 [==============================] - 0s 3ms/step - loss: 0.4943 - accuracy: 0.769
0
Epoch 40/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4906 - accuracy: 0.765
8
```

```
Epoch 41/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4904 - accuracy: 0.772
2
Epoch 42/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4847 - accuracy: 0.788
0
Epoch 43/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4809 - accuracy: 0.775
3
Epoch 44/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4806 - accuracy: 0.778
5
Epoch 45/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4780 - accuracy: 0.775
3
Epoch 46/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4785 - accuracy: 0.781
6
Epoch 47/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4699 - accuracy: 0.788
0
Epoch 48/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4698 - accuracy: 0.765
8
Epoch 49/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4709 - accuracy: 0.778
5
Epoch 50/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4698 - accuracy: 0.800
6
Epoch 51/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4631 - accuracy: 0.788
0
Epoch 52/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4604 - accuracy: 0.794
3
Epoch 53/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4582 - accuracy: 0.791
1
Epoch 54/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4562 - accuracy: 0.791
1
Epoch 55/180
10/10 [==============================] - 0s 3ms/step - loss: 0.4526 - accuracy: 0.791
1
Epoch 56/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4527 - accuracy: 0.794
3
Epoch 57/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4514 - accuracy: 0.788
0
Epoch 58/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4448 - accuracy: 0.784
8
Epoch 59/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4440 - accuracy: 0.800
6
Epoch 60/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4408 - accuracy: 0.803
8
```

```
Epoch 61/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4389 - accuracy: 0.807
0
Epoch 62/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4380 - accuracy: 0.800
6
Epoch 63/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4360 - accuracy: 0.797
5
Epoch 64/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4329 - accuracy: 0.800
6
Epoch 65/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4323 - accuracy: 0.797
5
Epoch 66/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4286 - accuracy: 0.800
6
Epoch 67/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4240 - accuracy: 0.797
5
Epoch 68/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4261 - accuracy: 0.797
5
Epoch 69/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4200 - accuracy: 0.816
5
Epoch 70/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4167 - accuracy: 0.813
3
Epoch 71/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4146 - accuracy: 0.816
5
Epoch 72/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4154 - accuracy: 0.813
3
Epoch 73/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4096 - accuracy: 0.810
1
Epoch 74/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4081 - accuracy: 0.813
3
Epoch 75/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4057 - accuracy: 0.835
4
Epoch 76/180
10/10 [==============================] - 0s 2ms/step - loss: 0.4039 - accuracy: 0.829
1
Epoch 77/180
10/10 [==============================] - 0s 1ms/step - loss: 0.4033 - accuracy: 0.819
6
Epoch 78/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3993 - accuracy: 0.835
4
Epoch 79/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3925 - accuracy: 0.816
5
Epoch 80/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3993 - accuracy: 0.819
6
```

```
Epoch 81/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3957 - accuracy: 0.819
6
Epoch 82/180
10/10 [==============================] - 0s 943us/step - loss: 0.3900 - accuracy: 0.8
259
Epoch 83/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3885 - accuracy: 0.832
3
Epoch 84/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3856 - accuracy: 0.838
6
Epoch 85/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3826 - accuracy: 0.841
8
Epoch 86/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3818 - accuracy: 0.832
3
Epoch 87/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3825 - accuracy: 0.835
4
Epoch 88/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3776 - accuracy: 0.844
9
Epoch 89/180
10/10 [==============================] - 0s 961us/step - loss: 0.3755 - accuracy: 0.8
418
Epoch 90/180
10/10 [==============================] - 0s 925us/step - loss: 0.3708 - accuracy: 0.8
386
Epoch 91/180
10/10 [==============================] - 0s 973us/step - loss: 0.3681 - accuracy: 0.8
576
Epoch 92/180
10/10 [==============================] - 0s 943us/step - loss: 0.3738 - accuracy: 0.8
449
Epoch 93/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3641 - accuracy: 0.844
9
Epoch 94/180
10/10 [==============================] - 0s 941us/step - loss: 0.3646 - accuracy: 0.8
449
Epoch 95/180
10/10 [==============================] - 0s 892us/step - loss: 0.3635 - accuracy: 0.8
576
Epoch 96/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3587 - accuracy: 0.867
1
Epoch 97/180
10/10 [==============================] - 0s 926us/step - loss: 0.3574 - accuracy: 0.8
608
Epoch 98/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3514 - accuracy: 0.857
6
Epoch 99/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3534 - accuracy: 0.860
8
Epoch 100/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3498 - accuracy: 0.873
4
```

```
Epoch 101/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3485 - accuracy: 0.854
4
Epoch 102/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3471 - accuracy: 0.870
3
Epoch 103/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3420 - accuracy: 0.867
1
Epoch 104/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3364 - accuracy: 0.876
6
Epoch 105/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3413 - accuracy: 0.867
1
Epoch 106/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3377 - accuracy: 0.863
9
Epoch 107/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3355 - accuracy: 0.873
4
Epoch 108/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3316 - accuracy: 0.876
6
Epoch 109/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3267 - accuracy: 0.873
4
Epoch 110/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3263 - accuracy: 0.873
4
Epoch 111/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3292 - accuracy: 0.882
9
Epoch 112/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3248 - accuracy: 0.889
2
Epoch 113/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3205 - accuracy: 0.863
9
Epoch 114/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3196 - accuracy: 0.876
6
Epoch 115/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3165 - accuracy: 0.886
1
Epoch 116/180
10/10 [==============================] - 0s 2ms/step - loss: 0.3171 - accuracy: 0.879
7
Epoch 117/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3135 - accuracy: 0.895
6
Epoch 118/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3135 - accuracy: 0.886
1
Epoch 119/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3039 - accuracy: 0.895
6
Epoch 120/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3064 - accuracy: 0.898
7
```

```
Epoch 121/180
10/10 [==============================] - 0s 1ms/step - loss: 0.3073 - accuracy: 0.882
9
Epoch 122/180
10/10 [==============================] - 0s 969us/step - loss: 0.3041 - accuracy: 0.8
987
Epoch 123/180
10/10 [==============================] - 0s 972us/step - loss: 0.2992 - accuracy: 0.8
892
Epoch 124/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2958 - accuracy: 0.901
9
Epoch 125/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2975 - accuracy: 0.886
1
Epoch 126/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2884 - accuracy: 0.901
9
Epoch 127/180
10/10 [==============================] - 0s 974us/step - loss: 0.2940 - accuracy: 0.8
987
Epoch 128/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2891 - accuracy: 0.905
1
Epoch 129/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2879 - accuracy: 0.892
4
Epoch 130/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2869 - accuracy: 0.908
2
Epoch 131/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2840 - accuracy: 0.905
1
Epoch 132/180
10/10 [==============================] - 0s 963us/step - loss: 0.2783 - accuracy: 0.9
082
Epoch 133/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2837 - accuracy: 0.901
9
Epoch 134/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2786 - accuracy: 0.901
9
Epoch 135/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2751 - accuracy: 0.905
1
Epoch 136/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2707 - accuracy: 0.911
4
Epoch 137/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2757 - accuracy: 0.905
1
Epoch 138/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2681 - accuracy: 0.911
4
Epoch 139/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2635 - accuracy: 0.914
6
Epoch 140/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2618 - accuracy: 0.911
4
```

```
Epoch 141/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2627 - accuracy: 0.905
1
Epoch 142/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2640 - accuracy: 0.911
4
Epoch 143/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2586 - accuracy: 0.908
2
Epoch 144/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2560 - accuracy: 0.911
4
Epoch 145/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2575 - accuracy: 0.905
1
Epoch 146/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2536 - accuracy: 0.911
4
Epoch 147/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2496 - accuracy: 0.908
2
Epoch 148/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2538 - accuracy: 0.911
4
Epoch 149/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2464 - accuracy: 0.917
7
Epoch 150/180
10/10 [==============================] - 0s 991us/step - loss: 0.2472 - accuracy: 0.9
177
Epoch 151/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2413 - accuracy: 0.917
7
Epoch 152/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2425 - accuracy: 0.920
9
Epoch 153/180
10/10 [==============================] - 0s 843us/step - loss: 0.2434 - accuracy: 0.9
146
Epoch 154/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2380 - accuracy: 0.920
9
Epoch 155/180
10/10 [==============================] - 0s 870us/step - loss: 0.2385 - accuracy: 0.9
146
Epoch 156/180
10/10 [==============================] - 0s 931us/step - loss: 0.2347 - accuracy: 0.9
177
Epoch 157/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2349 - accuracy: 0.917
7
Epoch 158/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2278 - accuracy: 0.920
9
Epoch 159/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2311 - accuracy: 0.924
1
Epoch 160/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2276 - accuracy: 0.914
6
```

```
Epoch 161/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2266 - accuracy: 0.927
2
Epoch 162/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2205 - accuracy: 0.927
2
Epoch 163/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2200 - accuracy: 0.920
9
Epoch 164/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2217 - accuracy: 0.924
1
Epoch 165/180
10/10 [==============================] - 0s 1ms/step - loss: 0.2152 - accuracy: 0.927
2
Epoch 166/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2132 - accuracy: 0.930
4
Epoch 167/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2150 - accuracy: 0.917
7
Epoch 168/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2119 - accuracy: 0.924
1
Epoch 169/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2039 - accuracy: 0.936
7
Epoch 170/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2075 - accuracy: 0.927
2
Epoch 171/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2009 - accuracy: 0.933
5
Epoch 172/180
10/10 [==============================] - 0s 3ms/step - loss: 0.2053 - accuracy: 0.939
9
Epoch 173/180
10/10 [==============================] - 0s 3ms/step - loss: 0.2038 - accuracy: 0.930
4
Epoch 174/180
10/10 [==============================] - 0s 2ms/step - loss: 0.2023 - accuracy: 0.936
7
Epoch 175/180
10/10 [==============================] - 0s 2ms/step - loss: 0.1989 - accuracy: 0.936
7
Epoch 176/180
10/10 [==============================] - 0s 2ms/step - loss: 0.1975 - accuracy: 0.933
5
Epoch 177/180
10/10 [==============================] - 0s 2ms/step - loss: 0.1959 - accuracy: 0.933
5
Epoch 178/180
10/10 [==============================] - 0s 3ms/step - loss: 0.1911 - accuracy: 0.936
7
Epoch 179/180
10/10 [==============================] - 0s 2ms/step - loss: 0.1907 - accuracy: 0.939
9
Epoch 180/180
10/10 [==============================] - 0s 2ms/step - loss: 0.1881 - accuracy: 0.933
5
```

Out[188]: <keras.callbacks.History at 0x1f006b474f0>

In [189…
```
model.evaluate(X_test,y_test)
```

```
3/3 [==============================] - 0s 2ms/step - loss: 1.0989 - accuracy: 0.5949
```

Out[189]:
```
[1.0988645553588867, 0.594936728477478]
```

In [190…
```
yp=model.predict(X_test)
yp[:5]
```

Out[190]:
```
array([[0.6653685 ],
       [0.07507494],
       [0.92375535],
       [0.8437072 ],
       [0.9589287 ]], dtype=float32)
```

In [191…
```
y_test[:10]
```

Out[191]:
```
306    1.0
343    0.0
117    1.0
50     1.0
316    0.0
279    1.0
394    0.0
354    1.0
123    1.0
357    1.0
Name: passed, dtype: float64
```

In [192…
```
y_pred=[]
for element in yp:
    if element>0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

In [193…
```
y_pred[:10]
```

Out[193]:
```
[1, 0, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [194…
```
from sklearn.metrics import confusion_matrix,classification_report

print(classification_report(y_test,y_pred))
yt_ann=y_test
yp_ann=y_pred
```

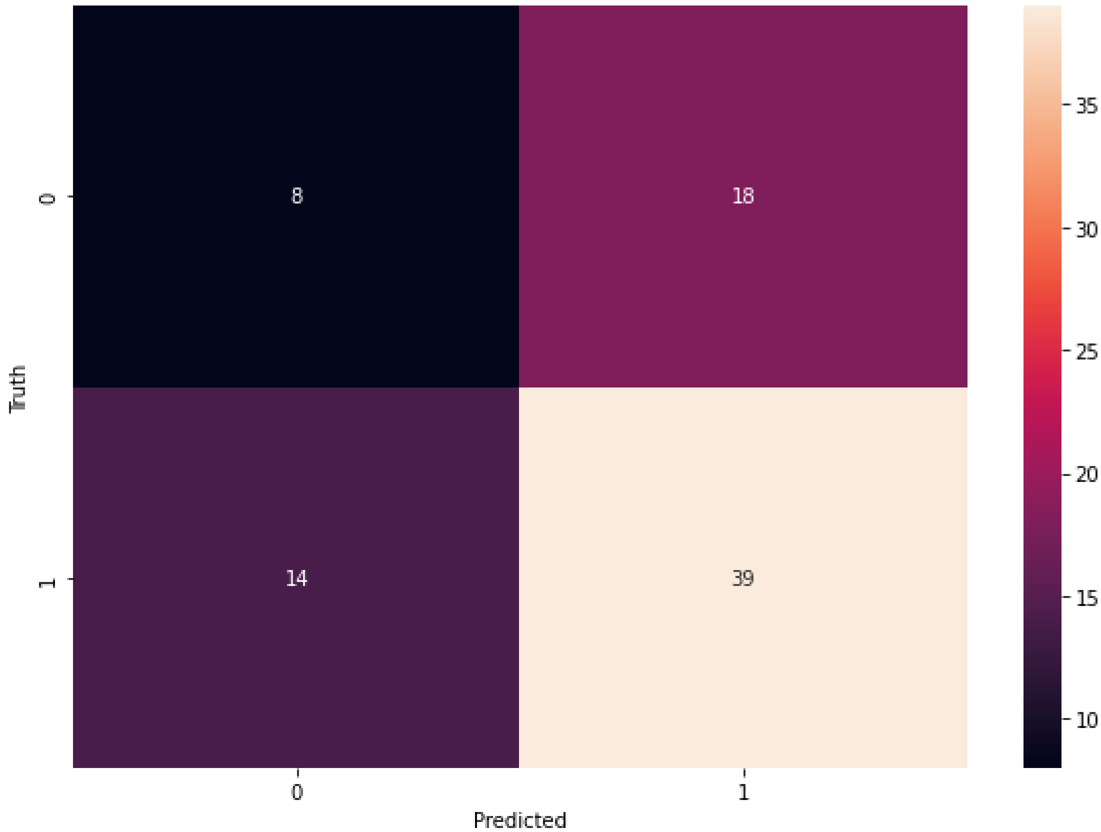
```
              precision    recall  f1-score   support

         0.0       0.36      0.31      0.33        26
         1.0       0.68      0.74      0.71        53

    accuracy                           0.59        79
   macro avg       0.52      0.52      0.52        79
weighted avg       0.58      0.59      0.59        79
```

In [195…
```
import seaborn as sn
```

```
cm=tf.math.confusion_matrix(labels=y_test,predictions=y_pred)
plt.figure(figsize=(10,7))
sn.heatmap(cm,annot=True,fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[195]:

```
Text(69.0, 0.5, 'Truth')
```



In [196…

```
round((6+38)/(6+38+15+38),2)*100
```

Out[196]:

```
45.0
```

In [ ]:

In [197…

```
# Function to compare the three classifiers (Logistic regression, KNN and SVM) perforn

def compare_lg_knn_svm(yt_knn,yp_knn,yt_lg,yp_lg,yt_svm,yp_svm,yt_ann,yp_ann):
    #F1 score
    f1_lg = round(f1_score(yt_lg, yp_lg, average='macro')*100)
    f1_knn = round(f1_score(yt_knn, yp_knn, average='macro')*100)
    f1_svm = round(f1_score(yt_svm, yp_svm, average='macro')*100)
    f1_ann = round(f1_score(yt_ann, yp_ann, average='macro')*100)

    #Accuracy score
    acc_lg = round(accuracy_score(yt_lg, yp_lg)*100)
    acc_knn = round(accuracy_score(yt_knn, yp_knn)*100)
    acc_svm = round(accuracy_score(yt_svm, yp_svm)*100)
    acc_ann = round(accuracy_score(yt_ann, yp_ann)*100)

    #Confusion matrix
    conf_lg = confusion_matrix(yt_lg, yp_lg)
    conf_knn = confusion_matrix(yt_knn, yp_knn)
```

```
    conf_svm = confusion_matrix(yt_svm, yp_svm)
    conf_ann = confusion_matrix(yt_ann, yp_ann)

    #ROC score
    roc_c_lg = round(roc_auc_score(yt_lg, yp_lg)*100)
    roc_c_knn = round(roc_auc_score(yt_knn, yp_knn)*100)
    roc_c_svm = round(roc_auc_score(yt_svm, yp_svm)*100)
    roc_c_ann = round(roc_auc_score(yt_ann, yp_ann)*100)

    #ROC curve thresholds
    roc_knn = roc_curve(yt_knn,yp_knn)
    roc_lg = roc_curve(yt_lg,yp_lg)
    roc_svm = roc_curve(yt_svm,yp_svm)
    roc_ann = roc_curve(yt_ann,yp_ann)

    # Table of metrics
    print('-----------------------------Table of metrics--------------------------------
    data_rows = [('f1 score',f1_lg,f1_knn,f1_svm,f1_ann),
                  ('','','','',''),
                   ('accuracy %',acc_lg,acc_knn,acc_svm,acc_ann),
                  ('','','','',''),
                  ('confusion matrix',conf_lg[0], conf_knn[0], conf_svm[0],conf_ann[0])
                 ('',conf_lg[1], conf_knn[1], conf_svm[1],conf_ann[1]),
                  ('','','','',''),
                 ('ROC score',roc_c_lg,roc_c_knn,roc_c_svm,roc_c_ann),
                 ('','','','','')]
    t = Table(rows=data_rows, names=('metric','Logistic regression', 'KNN', 'SVM','ANN
    print(t)

    #Plot ROC curve
    print('\n\n-----------------------------ROC curves---------------------------------
    fig, _ = plt.subplots()
    fig.suptitle('Comparison of three ROC curves')
    fpr,tpr,thresholds=roc_lg
    plt.plot([0, 1],[0, 1],'--')
    plt.plot(fpr,tpr,label='Logistic regression :'+str(roc_c_lg))
    plt.xlabel('false positive')
    plt.ylabel('false negative')
    fpr,tpr,thresholds=roc_knn
    plt.plot(fpr,tpr,label='KNN :'+str(roc_c_knn))
    fpr,tpr,thresholds=roc_svm
    plt.plot(fpr,tpr,label='SVM :'+str(roc_c_svm))
    fpr,tpr,thresholds=roc_ann
    plt.plot(fpr,tpr,label='ANN :'+str(roc_c_ann))
    plt.legend()
    plt.show()


    # Maximum metrics
    print('-----------------------------Max of metrics---------------------------------
    data_rows = [('max f1 score',algo_with_max_metric(f1_lg,f1_knn,f1_svm,f1_ann)),
                  ('','','',''),
                   ('max accuracy %',algo_with_max_metric(acc_lg,acc_knn,acc_svm,acc_ar
                  ('','','',''),
                 ('max ROC score',algo_with_max_metric(roc_c_lg,roc_c_knn,roc_c_svm,roc
    t = Table(rows=data_rows, names=('metric','Learning algorithm winnig'))
    print(t)

# Function returning name of winnig algorithm based on a single metric
def algo_with_max_metric(a,b,c,d):
```

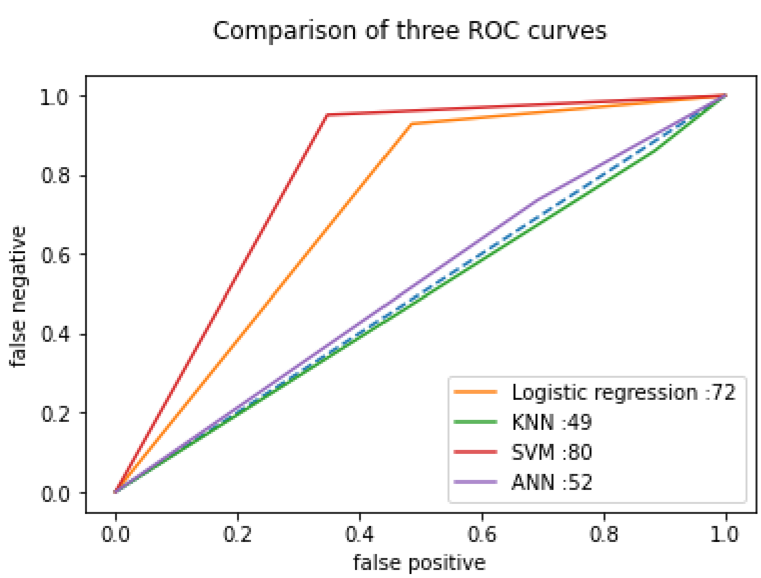
```python
    max_metric = max(a,b,c)
    if max_metric == a:
        return 'Logistic regression'
    elif max_metric == b:
        return 'KNN'
    elif max_metric == c:
        return 'SVM'
    else:
        return 'ANN'
```

In [ ]:

In [198…

```python
compare_lg_knn_svm(yt_knn,yp_knn,yt_lg,yp_lg,yt_svm,yp_svm,yt_ann,yp_ann)
```

```
----------------------------Table of metrics-------------------------------------


     metric       Logistic regression    KNN      SVM      ANN
----------------  -------------------  -------  -------  -------
        f1 score                   74       48       82       52

      accuracy %                   81       70       84       59

confusion matrix             [18 17]  [ 3 23]  [15  8]  [ 8 18]
                             [ 6 78]  [13 80]  [ 2 39]  [14 39]

       ROC score                   72       49       80       52



----------------------------ROC curves-------------------------------------
```



```
-----------------------------Max of metrics--------------------------------------


    metric      Learning algorithm winnig
-------------- -------------------------
  max f1 score                       SVM

max accuracy %                       SVM

 max ROC score                       SVM
```

In [ ]: