In [1]:

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

In [73]:

```
train = pd.read_csv("./train.csv",sep = ",")
test = pd.read_csv("./test.csv",sep = ",")
```

In [74]:

```
#print(test.info())
```

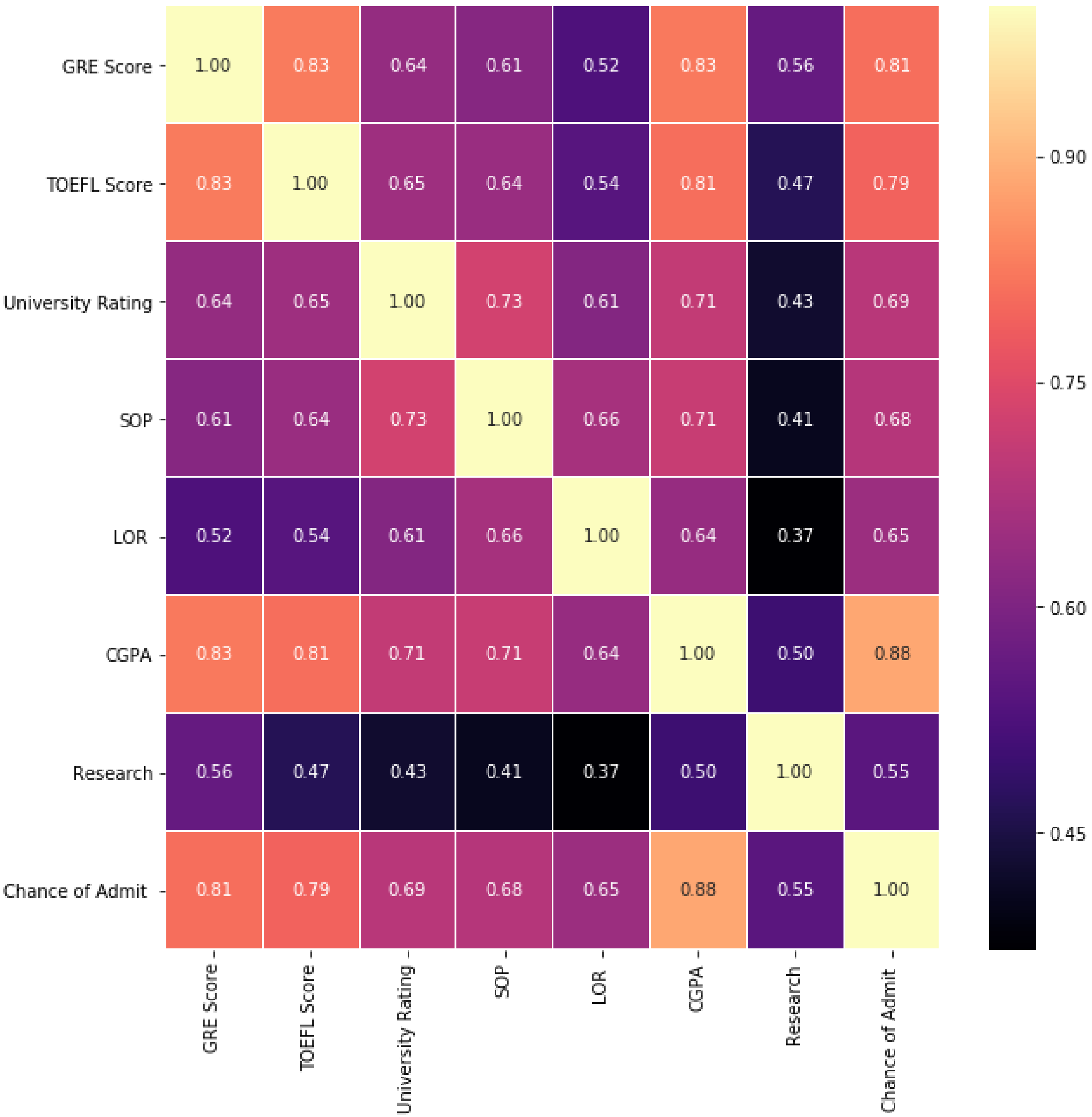
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
SerialNo             400 non-null int64
GREScore             400 non-null int64
TOEFLScore           400 non-null int64
UniversityRating     400 non-null int64
SOP                  400 non-null float64
LOR                  400 non-null float64
CGPA                 400 non-null float64
Research             400 non-null int64
Chanceofadmit        400 non-null float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
None
```

In [75]:

```
train.drop('SerialNo',axis=1,inplace=True)
test.drop('SerialNo',axis=1,inplace=True)
```

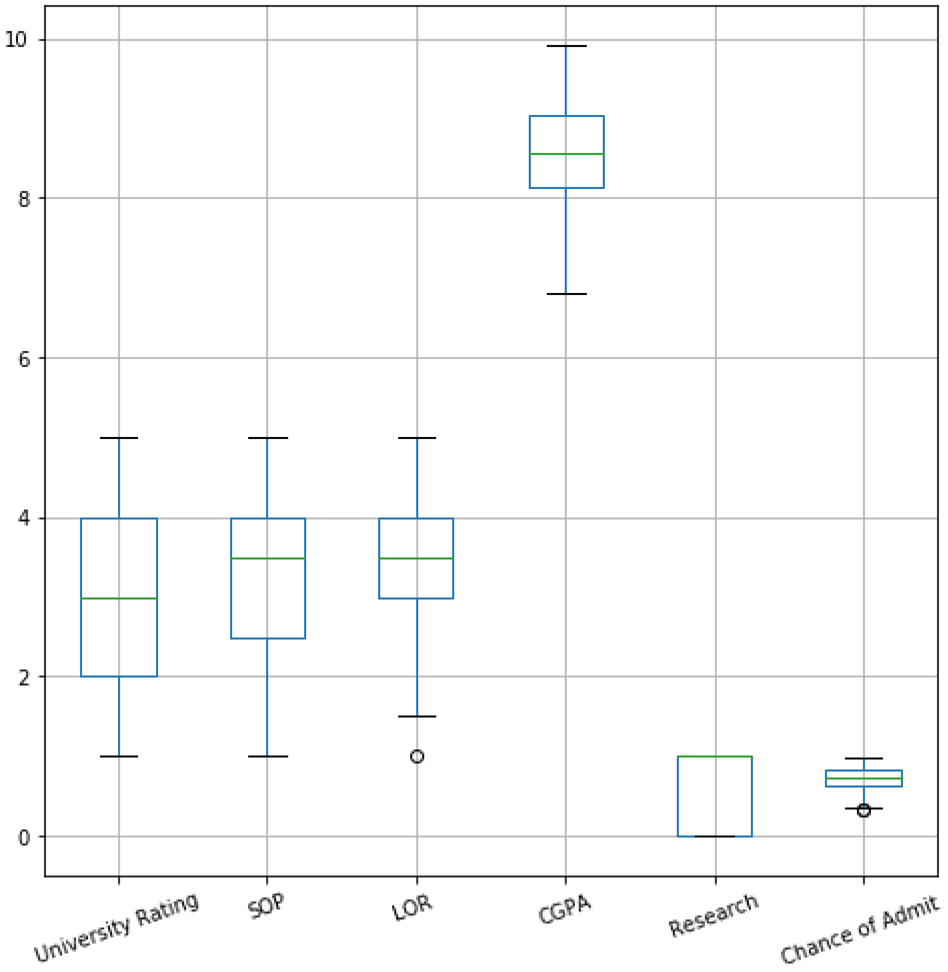
In [35]:

```
fig,ax = plt.subplots(figsize=(10, 10))
sns.heatmap(train.corr(), ax=ax, annot=True, linewidths=0.05, fmt= '.2f',cmap=
"magma")
plt.show()
```



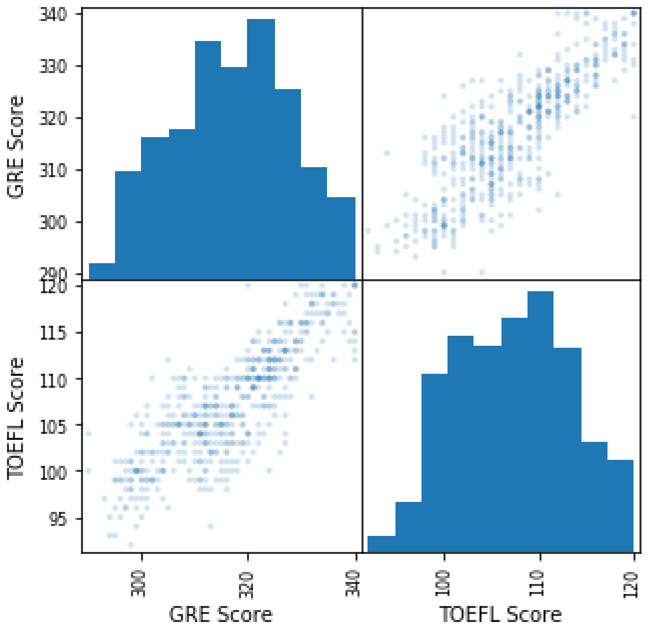
In [48]:
```
train1=train.drop(['TOEFLScore','GREScore'],axis=1)
#train1.boxplot(figsize=(8,8),rot=20)
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe6bd8f28>
```



In [42]:

```
plot = pd.plotting.scatter_matrix(train[['GREScore','TOEFLScore']], alpha = 0.
2, figsize=(5,5))
```



In [103]:

```
from sklearn.linear_model import LinearRegression
def rmse(y_true,y_pred):
    return np.sqrt(np.mean(np.square(y_true-y_pred)))
y_train = train['ChanceofAdmit']
X_train = train.drop(['ChanceofAdmit'],axis=1)
y_test = test['Chanceofadmit ']
X_test = test.drop(['Chanceofadmit '],axis=1)
# normalization
from sklearn.preprocessing import MinMaxScaler
scalerX = MinMaxScaler(feature_range=(0, 1))
X_train[X_train.columns] = scalerX.fit_transform(X_train[X_train.columns])
X_test[X_test.columns] = scalerX.transform(X_test[X_test.columns])
reg = LinearRegression().fit(X_train, y_train)
print("Train RMSE:", rmse(np.expm1(reg.predict(X_train)),np.expm1(y_train)))
print("Dev RMSE:", rmse(np.expm1(reg.predict(X_test)),np.expm1(y_test)))
```

```
Train RMSE: 0.11305655248136955
Dev RMSE: 0.11987948785272613
```

In [113]:

```
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
lr = LinearRegression()
lr.fit(X_train,y_train)
y_head_lr = lr.predict(X_test)
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("r_square score: ", r2_score(y_test,y_head_lr))

y_head_lr_train = lr.predict(X_train)
print("r_square score (train dataset): ", r2_score(y_train,y_head_lr_train))
print("MSE : ",mean_squared_error(y_test,y_head_lr))
print(lr.coef_)
print(lr.intercept_)
```

```
r_square score:  0.8027945404658843
r_square score (train dataset):  0.8219007395178417
MSE :  0.004000623838261953
[0.09292532 0.07778323 0.02376547 0.00634455 0.06743497 0.36936137
 0.02430748]
0.3482198690275547
```

In [117]:

```
import statsmodels.api as sm
def stepwise_selection(X, y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[new_col
umn]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.argmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, bes
t_pval))
         # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, wo
rst_pval))
        if not changed:
            break
    return included

result = stepwise_selection(X_train, y_train)

print('resulting features:')
print(result)
```

```
Add  CGPA                           with p-value 3.39654e-165
Add  GREScore                       with p-value 2.15892e-12
Add  LOR                            with p-value 4.82109e-08
Add  Research                       with p-value 0.000297132
Add  TOEFLScore                     with p-value 0.000505753
resulting features:
['CGPA', 'GREScore', 'LOR ', 'Research', 'TOEFLScore']
```

```
C:\Users\pc\Anaconda3\lib\site-packages\ipykernel_launcher.py:18: FutureWarni
ng: 'argmin' is deprecated, use 'idxmin' instead. The behavior of 'argmin'
will be corrected to return the positional minimum in the future.
Use 'series.values.argmin' to get the position of the minimum now.
```

In [131]:

```
train2=train.drop(['SOP','UniversityRating'],axis=1)
test2=test.drop(['SOP','UniversityRating'],axis=1)
```

In [ ]:

In [132]:

```
y_train2 = train2['ChanceofAdmit']
X_train2 = train2.drop(['ChanceofAdmit'],axis=1)
y_test2 = test2['Chanceofadmit ']
X_test2 = test2.drop(['Chanceofadmit '],axis=1)
from sklearn.preprocessing import MinMaxScaler
scalerX = MinMaxScaler(feature_range=(0, 1))
X_train2[X_train2.columns] = scalerX.fit_transform(X_train2[X_train2.columns])
X_test2[X_test2.columns] = scalerX.transform(X_test2[X_test2.columns])
reg = LinearRegression().fit(X_train2, y_train2)
print("Train RMSE:", rmse(np.expm1(reg.predict(X_train2)),np.expm1(y_train2)))
print("Dev RMSE:", rmse(np.expm1(reg.predict(X_test2)),np.expm1(y_test2)))
y_head_lr2 = reg.predict(X_test2)
print("r_square score: ", r2_score(y_test2,y_head_lr2))

y_head_lr_train2 = reg.predict(X_train2)
print("r_square score (train dataset): ", r2_score(y_train2,y_head_lr_train2))
print("MSE : ",mean_squared_error(y_test,y_head_lr2))
print(lr.coef_)
print(lr.intercept_)
```

```
Train RMSE: 0.11387067021485664
Dev RMSE: 0.12039845870299955
r_square score:  0.8024198485620984
r_square score (train dataset):  0.8206600544799475
MSE :  0.004008225054606728
[0.09292532 0.07778323 0.02376547 0.00634455 0.06743497 0.36936137
 0.02430748]
0.3482198690275547
```

In [134]:

```
import scipy
import statsmodels
from statsmodels.formula.api import ols

formula = 'ChanceofAdmit ~ C(TOEFLScore) + C(GREScore) + C(TOEFLScore):C(GRESc
ore)'
model = ols(formula, train).fit()
aov_table = statsmodels.stats.anova.anova_lm(model, typ=2)
print(aov_table)
```

```
C:\Users\pc\Anaconda3\lib\site-packages\statsmodels\base\model.py:1532: Value
Warning: covariance of constraints does not have full rank. The number of con
straints is 28, but rank is 5
  'rank is %d' % (J, J_), ValueWarning)
C:\Users\pc\Anaconda3\lib\site-packages\statsmodels\base\model.py:1532: Value
Warning: covariance of constraints does not have full rank. The number of con
straints is 48, but rank is 8
  'rank is %d' % (J, J_), ValueWarning)

                              sum_sq      df         F        PR(>F)
C(TOEFLScore)               1.489230    28.0  8.748696  1.959320e-07
C(GREScore)                 0.612450    48.0  2.098791  3.815772e-02
C(TOEFLScore):C(GREScore)  60.196089  1344.0  7.367305  3.714409e-39
Residual                    1.076053   177.0       NaN           NaN

C:\Users\pc\Anaconda3\lib\site-packages\statsmodels\base\model.py:1532: Value
Warning: covariance of constraints does not have full rank. The number of con
straints is 1344, but rank is 320
  'rank is %d' % (J, J_), ValueWarning)
```

In [139]:

```
y_train3 = train2['ChanceofAdmit']
X_train3 = train2.drop(['ChanceofAdmit','TOEFLScore'],axis=1)
y_test3 = test2['Chanceofadmit ']
X_test3 = test2.drop(['Chanceofadmit ','TOEFLScore'],axis=1)
scalerX = MinMaxScaler(feature_range=(0, 1))
X_train3[X_train3.columns] = scalerX.fit_transform(X_train3[X_train3.columns])
X_test3[X_test3.columns] = scalerX.transform(X_test3[X_test3.columns])
reg = LinearRegression().fit(X_train3, y_train3)
print("Train RMSE:", rmse(np.expm1(reg.predict(X_train3)),np.expm1(y_train3)))
print("Dev RMSE:", rmse(np.expm1(reg.predict(X_test3)),np.expm1(y_test3)))
y_head_lr3 = reg.predict(X_test3)
print("r_square score: ", r2_score(y_test3,y_head_lr3))

y_head_lr_train3 = reg.predict(X_train3)
print("r_square score (train dataset): ", r2_score(y_train3,y_head_lr_train3))
print("MSE : ",mean_squared_error(y_test,y_head_lr3))
print(lr.coef_)
print(lr.intercept_)
```

```
Train RMSE: 0.11553384473757276
Dev RMSE: 0.12186550689534695
r_square score:  0.7984015073271082
r_square score (train dataset):  0.8162106363984603
MSE :  0.0040897434454918
[0.09292532 0.07778323 0.02376547 0.00634455 0.06743497 0.36936137
 0.02430748]
0.3482198690275547
```