1. 원시 데이터 간편 분석

1-가. 데이터 확인 : df.head( ) 함수 이용

In [50]:
```python
import pandas as pd
import numpy as np

df = pd.read_excel("Admission_Predictions.xlsx")

df.head()
```

Out[50]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

1-나. 변수 설명

- Serial No. : 불필요 변수로서 제거 예정
- GRE Score : GRE 대학원 입학 시험 점수 (out of 340)
- TOEFL Score : 토플 시험 점수 (out of 120)
- University Rating : 지원 학생의 소속 대학교 평가점수 (out of 5)
- SOP : Statement of Purpose Strength (out of 5)
- LOR : Letter of Recommendation Strength (out of 5)
- CGPA : 지원 학생의 대학교 내신 성적 GPA (out of 10)
- Research : 리서치 경험 (either 0 or 1)
- Chance of Admit : 입학 가능성 (ranging from 0 to 1)

기타 데이터별 주석

- University Rating : 지원자가 소속된 대학교의 미국 내 순위로서 경쟁자들과 다른 변수가 동일한 경우 대학교 평가 점수의 기여도를 활용할 수 있겠으나 그나마 데이터 양이 적어서 정확도가 미흡
- CGPA : 데이터셋 제공자가 임의 조정하였으며 미 대학원에 지원하는 미국인과 인도인 대학생 기준임
- Research : 대학교 재학 중 리서치 경험 여부를 표시 (Binary component로서 특수 변수)
- Chance of Admit : 입학 승인 가능성 내지는 지원자의 "confidence"로 보는 견해도 있음

1-다. 데이터 정보 출력

In [51]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
Serial No.           500 non-null int64
GRE Score            500 non-null int64
TOEFL Score          500 non-null int64
University Rating    500 non-null int64
SOP                  500 non-null float64
LOR                  500 non-null float64
CGPA                 500 non-null float64
Research             500 non-null int64
Chance of Admit      500 non-null float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

1-라. 불필요 컬럼 제외

In [52]:
```python
df = df.drop(columns=['Serial No.'])

df.head()
```

Out[52]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

1-마. 데이터 전반적 통계적 특성 보기

In [53]:
```python
df.describe()
```

Out[53]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

2. 데이터 변수 간 상관관계

2-가. 변수 간 상관관계 측정

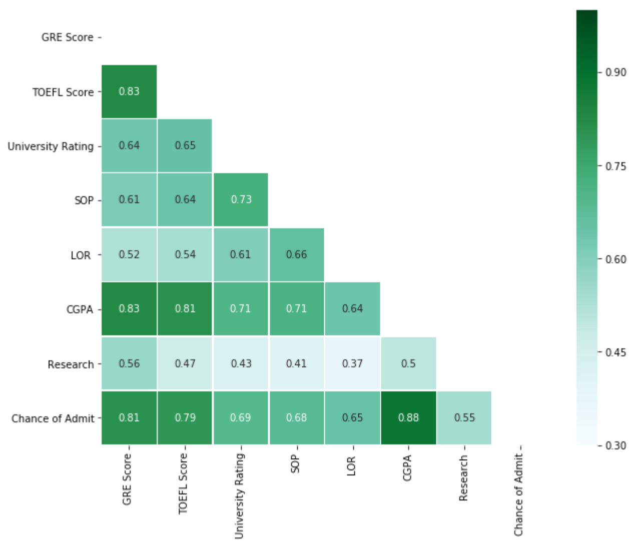
In [54]:
```
df.corr()
```

Out[54]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1.000000	0.827200	0.635376	0.613498	0.524679	0.825878	0.563398	0.810351
TOEFL Score	0.827200	1.000000	0.649799	0.644410	0.541563	0.810574	0.467012	0.792228
University Rating	0.635376	0.649799	1.000000	0.728024	0.608651	0.705254	0.427047	0.690132
SOP	0.613498	0.644410	0.728024	1.000000	0.663707	0.712154	0.408116	0.684137
LOR	0.524679	0.541563	0.608651	0.663707	1.000000	0.637469	0.372526	0.645365
CGPA	0.825878	0.810574	0.705254	0.712154	0.637469	1.000000	0.501311	0.882413
Research	0.563398	0.467012	0.427047	0.408116	0.372526	0.501311	1.000000	0.545871
Chance of Admit	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

2-나. 상관관계 히트맵

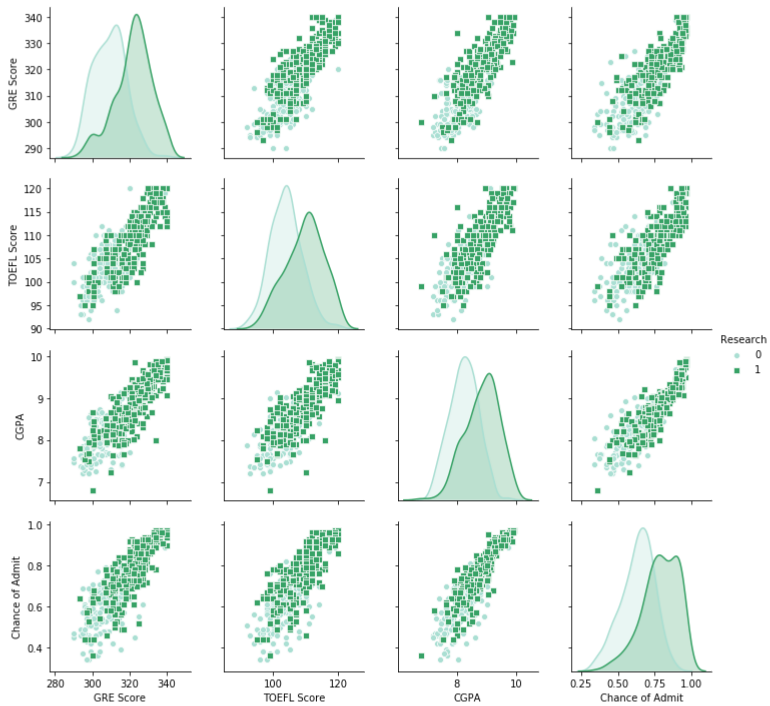
In [55]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))

mask = np.zeros_like(df.corr())
mask[np.triu_indices_from(mask)] = True
ax = sns.heatmap(df.corr(), 0.3, 1, annot=True, linewidths=.5, cmap="BuGn", mask = mask)
```



2-다. "Research" 변수 분석 : Pairplot 함수 이용

In [56]:
```
plt.figure()

sns.pairplot(df, diag_kind='kde' , hue= "Research", vars=['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit'], palette="BuGn", markers=["o", "s"])
```

Out[56]:
```
<seaborn.axisgrid.PairGrid at 0x2dda39747c8>

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



리서치 유경험 학생들이 높은 시험 성적 경향이 높고, 입학 승인 가능성에서 다소 우위를 보이는 것으로 분석

3. 기계학습을 위한 데이터 전처리

3-가. 알고리즘 대입을 위한 변수 정하기

3-(1). 예측 분석용 데이터 테이블 주관적 가공

너무 많은 독립변수는 선형 회귀분석의 과적함을 불러일으킬 수 있으므로 상관계수가 0.8에 근접하거나 이상인 변수들만 (임의로) 채택함

In [57]:
```
Predictive_df = df[['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit']]

#Linear Regression 을 위한 dataframe 이 생성되었음을 확인
Predictive_df
```

Out[57]:

	GRE Score	TOEFL Score	CGPA	Chance of Admit
0	337	118	9.65	0.92
1	324	107	8.87	0.76
2	316	104	8.00	0.72
3	322	110	8.67	0.80
4	314	103	8.21	0.65
...	...	...	...	...
495	332	108	9.02	0.87
496	337	117	9.87	0.96
497	330	120	9.56	0.93
498	312	103	8.43	0.73
499	327	113	9.04	0.84

500 rows × 4 columns

3-(2). 변수 선택 방법으로 변수 정하기 - Forward Selection

주관적인 견해가 들어가지 않는 변수 선택 알고리즘을 통한 변수 선택하기

In [58]:
```python
import statsmodels.api as sm
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


before_selection = df.columns[:-1].tolist()

Y = df['Chance of Admit']
after_selection = []
criteria = 0.05

while len(before_selection ) > 0:
    remaining_selection = list(set(before_selection) - set(after_selection))
    p_value = pd.Series(index=remaining_selection)

    for col in remaining_selection:
        X = df[after_selection+[col]]
        X = sm.add_constant(X)
        model = sm.OLS(Y,X).fit()
        p_value[col] = model.pvalues[col]
    min_p_value = p_value.min()

    if min_p_value < criteria:
        after_selection.append(p_value.idxmin())

    else:
        break

after_selection
```

Out[58]: ['CGPA', 'GRE Score', 'LOR ', 'Research', 'TOEFL Score']

3-(3). 변수 선택 방법으로 변수 정하기 - Backward Selection

In [59]:
```python
considering_variables = df.columns[:-1].tolist()

Y = df['Chance of Admit']
after_selection = considering_variables
criteria = 0.05

while len(considering_variables) > 0:
    X = sm.add_constant(df[after_selection])
    p_value = sm.OLS(Y,X).fit().pvalues[1:]
    max_p_value = p_value.max()
    if max_p_value >= criteria:
        removed_variables = p_value.idxmax()
        after_selection.remove(removed_variables)

    else:
        break

after_selection
```

Out[59]: ['GRE Score', 'TOEFL Score', 'LOR ', 'CGPA', 'Research']

3-(4). 변수 선택 방법으로 변수 정하기 - Stepwise Selection

In [60]:
```python
considering_variables = df.columns[:-1].tolist()

Y = df['Chance of Admit']
after_selection = []
criteria = 0.05

while len(considering_variables) > 0:
    remaining_selection = list(set(considering_variables) - set(after_selection))
    p_value = pd.Series(index=remaining_selection)
    for col in remaining_selection:
        X = df[after_selection+[col]]
        X = sm.add_constant(X)
        model = sm.OLS(Y,X).fit()
        p_value[col] = model.pvalues[col]

    min_p_value = p_value.min()
    if min_p_value < criteria:
        after_selection.append(p_value.idxmin())

        while len(after_selection) > 0:
            after_X = df[after_selection]
            after_X = sm.add_constant(after_X)
            after_p_value = sm.OLS(Y,after_X).fit().pvalues[1:]
            max_p_value = after_p_value.max()
            if max_p_value >= criteria:
                removed_variables = after_p_value.idxmax()
                after_selection.remove(removed_variables)
            else:
                break

    else:
        break

after_selection
```

Out[60]: ['CGPA', 'GRE Score', 'LOR ', 'Research', 'TOEFL Score']

3-(5). 변수 선택 방법 요약

In [61]:
```python
import pandas as pd

selections = {'Forward Selection' : ['CGPA', 'GRE Score', 'LOR ', 'Research', 'TOEFL Score'],
              'Backward Selection' : ['GRE Score', 'TOEFL Score', 'LOR ', 'CGPA', 'Research'],
              'Stepwise Selection' : ['CGPA', 'GRE Score', 'LOR ', 'Research', 'TOEFL Score']
             }

SelectionMethod_df = pd.DataFrame(selections, columns = ['Forward Selection', 'Backward Selection', 'Stepwise Selection'],
                                                         index=['1st', '2nd', '3rd', '4th', '5th'])

SelectionMethod_df.style
```

Out[61]:

	Forward Selection	Backward Selection	Stepwise Selection
1st	CGPA	GRE Score	CGPA
2nd	GRE Score	TOEFL Score	GRE Score
3rd	LOR	LOR	LOR
4th	Research	CGPA	Research
5th	TOEFL Score	Research	TOEFL Score

결론적으로 university rating 및 SOP를 제외한 변수들 생존하였고 주관적 평가 요소인 LOR 변수는 제외 후 분석 예정

3-나. Training & Test Dataset Splitting

In [62]:
```python
from sklearn.model_selection import train_test_split

X = Predictive_df.iloc[:, 0:3]
Y = Predictive_df.iloc[:, 3]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= 0.30, random_state=123123)
```

4. 데이터 분석 (Predictive/Estimation Analysis)

4-(1) Linear Regression Without Scaling

In [63]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error as mse
import math

LR_reg = LinearRegression()
LR_reg.fit(X_train, Y_train)

Y_pred = LR_reg.predict(X_test)


LR_rScore  = r2_score(Y_test, Y_pred)
LR_rScore_r = round(LR_rScore, 6)
print("R square score for Linear Regression is : " + str(LR_rScore_r))

LR_error = mse(Y_test, Y_pred)
LR_error_r = round(LR_error, 6)
print("Mean Squared Error for Linear Regression is :" + str(LR_error_r))

LR_sq_error = math.sqrt(LR_error_r)
LR_sq_error_r = round(LR_sq_error, 6)
print("Root Mean Squared Error for Linear Regression is :" + str(LR_sq_error_r))

LR_Y_pred = Y_pred
LinRegChart = pd.DataFrame()
LinRegChart['예측값'] = np.round(Y_pred, 4)
LinRegChart['실제값'] = np.array(Y_test)
LinRegChart.head(10)
```

```
R square score for Linear Regression is : 0.846467
Mean Squared Error for Linear Regression is :0.00318
Root Mean Squared Error for Linear Regression is :0.056391
```

Out[63]:

	예측값	실제값
0	0.7510	0.77
1	0.5602	0.59
2	0.7525	0.77
3	0.6261	0.64
4	0.8367	0.82
5	0.8928	0.93
6	0.7365	0.84
7	0.6367	0.65
8	0.7540	0.71
9	0.8215	0.87

4-(2) Decision Tree Regression Without Scaling

In [64]:
```
from sklearn.tree import DecisionTreeRegressor

DT_reg = DecisionTreeRegressor(random_state = 8282)
DT_reg.fit(X_train, Y_train)

Y_pred = DT_reg.predict(X_test)


DT_rScore = r2_score(Y_test, Y_pred)
DT_rScore_r = round(DT_rScore, 6)
print("R square score for Decision Tree Regression is : " + str(DT_rScore_r))


DT_error = mse(Y_test, Y_pred)
DT_error_r = round(DT_error, 6)
print("Mean Squared Error for Decision Tree Regression is :" + str(DT_error_r))


DT_sq_error = math.sqrt(DT_error_r)
DT_sq_error_r = round(DT_sq_error, 6)
print("Root Mean Squared Error for Decision Tree Regression is :" + str(DT_sq_error_r))

DTChart = pd.DataFrame()
DTChart['예측값'] = np.round(Y_pred, 4)
DTChart['실제값'] = np.array(Y_test)
DTChart.head(10)
```

```
R square score for Decision Tree Regression is : 0.678144
Mean Squared Error for Decision Tree Regression is :0.006665
Root Mean Squared Error for Decision Tree Regression is :0.081639
```

Out[64]:

	예측값	실제값
0	0.81	0.77
1	0.63	0.59
2	0.74	0.77
3	0.73	0.64
4	0.84	0.82
5	0.92	0.93
6	0.71	0.84
7	0.62	0.65
8	0.78	0.71
9	0.86	0.87

4-(3) Random Forest Regression Without Scaling

In [65]:
```
from sklearn.ensemble import RandomForestRegressor
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


RFR_reg = RandomForestRegressor(random_state = 8282)
RFR_reg.fit(X_train, Y_train)

Y_pred = RFR_reg.predict(X_test)


RFR_rScore = r2_score(Y_test, Y_pred)
RFR_rScore_r = round(RFR_rScore, 6)
print("R square score for Random Forest Regression is : " + str(RFR_rScore_r))


RFR_error = mse(Y_test, Y_pred)
RFR_error_r = round(RFR_error, 6)
print("Mean Squared Error for Random Forest Regression is :" + str(RFR_error_r))


RFR_sq_error = math.sqrt(RFR_error_r)
RFR_sq_error_r = round(RFR_sq_error, 6)
print("Root Mean Squared Error for Random Forest Regression is :" + str(RFR_sq_error_r))

RFRChart = pd.DataFrame()
RFRChart['예측값'] = np.round(Y_pred, 4)
RFRChart['실제값'] = np.array(Y_test)
RFRChart.head(10)
```

```
R square score for Random Forest Regression is : 0.805673
Mean Squared Error for Random Forest Regression is :0.004024
Root Mean Squared Error for Random Forest Regression is :0.063435
```

Out[65]:

	예측값	실제값
0	0.720	0.77
1	0.642	0.59
2	0.723	0.77
3	0.663	0.64
4	0.826	0.82
5	0.911	0.93
6	0.690	0.84
7	0.618	0.65
8	0.735	0.71
9	0.801	0.87

4-더미 Scaling Data With MinMaxScaler

In [66]:
```python
from sklearn.preprocessing import MinMaxScaler

# we do scaling after Training & Testing data split because data leakage might happen

scaler = MinMaxScaler(feature_range = (0, 1))
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

4-(4) Linear Regression With Scaling

In [67]:
```python
LR_reg_o = LinearRegression()
LR_reg_o.fit(X_train, Y_train)

Y_pred = LR_reg_o.predict(X_test)


LR_rScore  = r2_score(Y_test, Y_pred)
LR_rScore_r_o = round(LR_rScore, 6)
print("R square score for Linear Regression is : " + str(LR_rScore_r_o))


LR_error = mse(Y_test, Y_pred)
LR_error_r_o = round(LR_error, 6)
print("Mean Squared Error for Linear Regression is :" + str(LR_error_r_o))

LR_sq_error = math.sqrt(LR_error_r_o)
LR_sq_error_r_o = round(LR_sq_error, 6)
print("Root Mean Squared Error for Linear Regression is :" + str(LR_sq_error_r_o))


LinRegChart = pd.DataFrame()
LinRegChart['예측값'] = np.round(Y_pred, 4)
LinRegChart['실제값'] = np.array(Y_test)
LinRegChart.head(10)
```

```
R square score for Linear Regression is : 0.835491
Mean Squared Error for Linear Regression is :0.003407
Root Mean Squared Error for Linear Regression is :0.05837
```

Out[67]:

	예측값	실제값
0	0.7579	0.77
1	0.5793	0.59
2	0.7604	0.77
3	0.6506	0.64
4	0.8394	0.82
5	0.8937	0.93
6	0.7455	0.84
7	0.6480	0.65
8	0.7653	0.71
9	0.8267	0.87

4-(5) Decision Tree Regression With Scaling

In [68]:
```
DT_reg = DecisionTreeRegressor(random_state = 8282)
DT_reg.fit(X_train, Y_train)

Y_pred = DT_reg.predict(X_test)


DT_rScore = r2_score(Y_test, Y_pred)
DT_rScore_r_o = round(DT_rScore, 6)
print("R square score for Decision Tree Regression is : " + str(DT_rScore_r_o))


DT_error = mse(Y_test, Y_pred)
DT_error_r_o = round(DT_error, 6)
print("Mean Squared Error for Decision Tree Regression is :" + str(DT_error_r_o))


DT_sq_error = math.sqrt(DT_error_r_o)
DT_sq_error_r_o = round(DT_sq_error, 6)
print("Root Mean Squared Error for Decision Tree Regression is :" + str(DT_sq_error_r_o))

DTChart = pd.DataFrame()
DTChart['예측값'] = np.round(Y_pred, 4)
DTChart['실제값'] = np.array(Y_test)
DTChart.head(10)
```

```
R square score for Decision Tree Regression is : 0.531477
Mean Squared Error for Decision Tree Regression is :0.009703
Root Mean Squared Error for Decision Tree Regression is :0.098504
```

Out[68]:

	예측값	실제값
0	0.55	0.77
1	0.34	0.59
2	0.85	0.77
3	0.67	0.64
4	0.88	0.82
5	0.91	0.93
6	0.63	0.84
7	0.62	0.65
8	0.79	0.71
9	0.78	0.87

4-(6) Random Forest Regression With Scaling

In [69]:
```
RFR_reg = RandomForestRegressor(random_state = 8282)
RFR_reg.fit(X_train, Y_train)

Y_pred = RFR_reg.predict(X_test)


RFR_rScore = r2_score(Y_test, Y_pred)
RFR_rScore_r_o = round(RFR_rScore, 6)
print("R square score for Random Forest Regression is : " + str(RFR_rScore_r_o))


RFR_error = mse(Y_test, Y_pred)
RFR_error_r_o = round(RFR_error, 6)
print("Mean Squared Error for Random Forest Regression is :" + str(RFR_error_r_o))


RFR_sq_error = math.sqrt(RFR_error_r_o)
RFR_sq_error_r_o = round(RFR_sq_error, 6)
print("Root Mean Squared Error for Random Forest Regression is :" + str(RFR_sq_error_r_o))

RFRChart = pd.DataFrame()
RFRChart['예측값'] = np.round(Y_pred, 4)
RFRChart['실제값'] = np.array(Y_test)
RFRChart.head(10)
```

```
R square score for Random Forest Regression is : 0.758994
Mean Squared Error for Random Forest Regression is :0.004991
Root Mean Squared Error for Random Forest Regression is :0.070647
```

Out[69]:

	예측값	실제값
0	0.739	0.77
1	0.544	0.59
2	0.766	0.77
3	0.643	0.64
4	0.871	0.82
5	0.910	0.93
6	0.707	0.84
7	0.579	0.65
8	0.681	0.71
9	0.804	0.87

4-가. 분석 결과 비교표 및 그래프

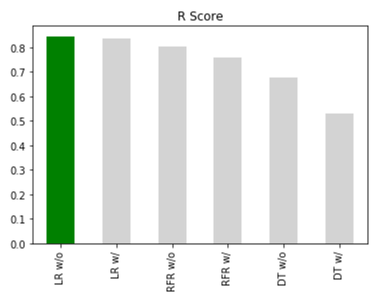
In [70]:
```
data = {
        'Linear Regression w/o scaling': [LR_rScore_r, LR_error_r , LR_sq_error_r],
        'Linear Regression w/ scaling' : [LR_rScore_r_o, LR_error_r_o, LR_sq_error_r_o],
        'Random Forest w/o scaling' : [RFR_rScore_r, RFR_error_r, RFR_sq_error_r],
        'Random Forest w/ scaling' : [RFR_rScore_r_o, RFR_error_r_o, RFR_sq_error_r_o],
        'Decision Tree w/o scaling' : [DT_rScore_r, DT_error_r, DT_sq_error_r],
        'Decision Tree w/ scaling' : [DT_rScore_r_o, DT_error_r_o, DT_sq_error_r_o]
         }

summary_df = pd.DataFrame(data, columns = ['Linear Regression w/o scaling',
                                           'Linear Regression w/ scaling',
                                           'Random Forest w/o scaling',
                                            'Random Forest w/ scaling',
                                           'Decision Tree w/o scaling',
                                           'Decision Tree w/ scaling'], index = ['R Score', 'MSE', 'R_MSE'])

summary_df.style
```

Out[70]:

	Linear Regression w/o scaling	Linear Regression w/ scaling	Random Forest w/o scaling	Random Forest w/ scaling	Decision Tree w/o scaling	Decision Tree w/ scaling
R Score	0.846467	0.835491	0.805673	0.758994	0.678144	0.531477
MSE	0.00318	0.003407	0.004024	0.004991	0.006665	0.009703
R_MSE	0.056391	0.05837	0.063435	0.070647	0.081639	0.098504

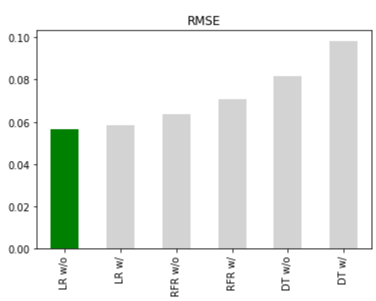
In [71]:
```
RegressBar = pd.DataFrame(
    {'R Score': [LR_rScore_r, LR_rScore_r_o, RFR_rScore_r, RFR_rScore_r_o, DT_rScore_r, DT_rScore_r_o]},
    index = ['LR w/o', 'LR w/', 'RFR w/o', 'RFR w/', 'DT w/o', 'DT w/'])

RegressBar['R Score'].plot(kind="bar", color =['green', 'lightgrey', 'lightgrey', 'lightgrey', 'lightgrey', 'lightgrey'])

plt.title("R Score")
```

Out[71]: `Text(0.5, 1.0, 'R Score')`



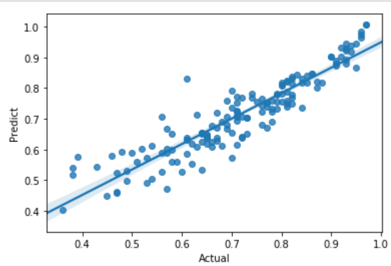
In [72]:
```
RegressBar = pd.DataFrame(
    {'RMSE': [LR_sq_error_r, LR_sq_error_r_o, RFR_sq_error_r, RFR_sq_error_r_o, DT_sq_error_r, DT_sq_error_r_o]},
    index = ['LR w/o', 'LR w/', 'RFR w/o', 'RFR w/', 'DT w/o', 'DT w/'])

RegressBar['RMSE'].plot(kind="bar", color =['green', 'lightgrey', 'lightgrey', 'lightgrey', 'lightgrey', 'lightgrey'])

plt.title("RMSE")
```

Out[72]: `Text(0.5, 1.0, 'RMSE')`



4-마. Residual Plot of Linear Regression w/o scaler

In [73]:
```
x, y = pd.Series(Y_test, name="Actual"), pd.Series(LR_Y_pred, name="Predict")
ax = sns.regplot(x=x, y=y)
```



4-바. 합격 여부 예측하기 (가상 점수 입력)

In [74]:
```
a = int(input ("Enter your GRE score (out of 340) : "))

b = int(input ("Enter your TOEFL score (out of 120) : "))

c = float(input ("Enter your College GPA score (out of 10 decimal place) : "))


print("Your likelihood for graduate school admission is : " )
print(LR_reg.predict([[a, b, c]]))
```

```
Enter your GRE score (out of 340) : 330
Enter your TOEFL score (out of 120) : 110
Enter your College GPA score (out of 10 decimal place) : 9
Your likelihood for graduate school admission is :
[0.82099384]
```

5. 데이터 분류 (Classification Method)

5-가. 데이터 칼럼 추가 (합격 여부 변수)

In [75]:
```
def f(row):
    if row['Chance of Admit'] >= 0.8:
        val = 1
    else:
        val = 0
    return val


df['Admit Condition'] = df.apply(f, axis=1)

df


# In[]:


df['Admit Condition'].value_counts()
```

Out[75]:
```
0    345
1    155
Name: Admit Condition, dtype: int64
```

차후에 진행할 분류 알고리즘을 위해 Chance of Admit의 합격 승인 가능성을 확률이 80% 이상으로 의제하였고, 이 경우 총 155명의 지원자가 합격할 것으로 예측함

5-나. 분류용 데이터 테이블 만들기

In [76]:
```
Classify_df = df[['GRE Score', 'TOEFL Score', 'CGPA', 'Admit Condition']]


Classify_df
```

Out[76]:

	GRE Score	TOEFL Score	CGPA	Admit Condition
0	337	118	9.65	1
1	324	107	8.87	0
2	316	104	8.00	0
3	322	110	8.67	1
4	314	103	8.21	0
...	...	...	...	...
495	332	108	9.02	1
496	337	117	9.87	1
497	330	120	9.56	1
498	312	103	8.43	0
499	327	113	9.04	1

500 rows × 4 columns

5-다. Training & Test Data Set Splitting

In [77]:
```
X = Classify_df.iloc[:, 0:3]
Y = Classify_df.iloc[:, 3]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= 0.20, random_state=123123)
```

5-라. Scaling Data (Unnecessary)

As for most of the Classification algorithms, there are no significance benefit in scaling & enhancing data. *Except for K-NN method with is extremely sensitive to magnitudes of the provided features*

5-(1) Logistic Regression

In [78]:

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)


Log_Score = accuracy_score(Y_test, Y_pred)
print("Accuracy : " + str(Log_Score))

plt.figure(figsize=(8,6))

cm = confusion_matrix(Y_test, Y_pred)
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred).ravel()
cm = [[tp,fp],[fn,tn]]

LR_recall = tp / (tp + fn)
LR_spec = tn / (tn + fp)
LR_prec = tp / (tp + fn)
LR_accu = (tp + tn)/ (tp + tn + fp + fn)
LR_f1 = (2 * tp) / (2 * tp + fp + fn)
LR_tn = tn
LR_tp = tp
LR_fn = fn
LR_fp = fp

sns.heatmap(cm, annot = True, fmt = "d", cmap="BuGn")
plt.xticks([0.5, 1.5], labels=[1,0])
plt.yticks([0.5, 1.5], labels=[1,0])
plt.title('Logistic Regressor Confusion Matrix')
plt.xlabel('Actual')
plt.ylabel('Predicted')

report = classification_report(Y_test, Y_pred)
print(report)

LRChart = pd.DataFrame()
LRChart['예측값'] = np.round(Y_pred)
LRChart['실제값'] = np.array(Y_test)
LRChart.loc[::1]
```

```
Accuracy : 0.81
              precision    recall  f1-score   support

           0       0.81      0.94      0.87        66
           1       0.83      0.56      0.67        34

    accuracy                           0.81       100
   macro avg       0.82      0.75      0.77       100
weighted avg       0.81      0.81      0.80       100
```

Out[78]:

	예측값	실제값
0	0	0
1	0	0
2	0	0
3	0	0
4	0	1
...	...	...
95	0	1
96	0	0
97	1	1
98	0	1
99	0	0

100 rows × 2 columns

5-(2) Decision Tree Classifier

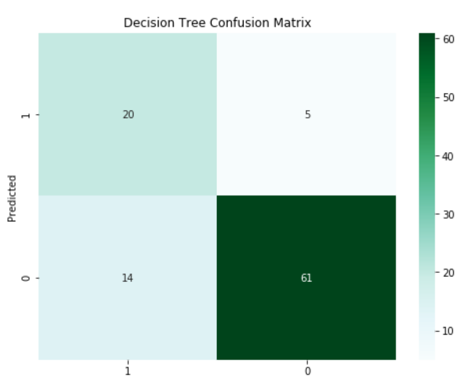
In [79]:
```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()
tree.fit(X_train, Y_train)
Y_pred = tree.predict(X_test)

tree_score = accuracy_score(Y_test, Y_pred)
print("Accuracy : " + str(tree_score))


plt.figure(figsize=(8,6))

cm = confusion_matrix(Y_test, Y_pred)
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred).ravel()
cm = [[tp,fp],[fn,tn]]

DT_recall = tp / (tp + fn)
DT_spec = tn / (tn + fp)
DT_prec = tp / (tp + fn)
DT_accu = (tp + tn)/ (tp + tn + fp + fn)
DT_f1 = (2 * tp) / (2 * tp + fp + fn)
DT_tn = tn
DT_tp = tp
DT_fn = fn
DT_fp = fp

sns.heatmap(cm, annot = True, fmt = "d", cmap="BuGn")
plt.xticks([0.5, 1.5], labels=[1,0])
plt.yticks([0.5, 1.5], labels=[1,0])
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Actual')
plt.ylabel('Predicted')

report = classification_report(Y_test, Y_pred)
print(report)

DTChart = pd.DataFrame()
DTChart['예측값'] = np.round(Y_pred)
DTChart['실제값'] = np.array(Y_test)
DTChart.loc[::1]
```

```
Accuracy : 0.81
              precision    recall  f1-score   support

           0       0.81      0.92      0.87        66
           1       0.80      0.59      0.68        34

    accuracy                           0.81       100
   macro avg       0.81      0.76      0.77       100
weighted avg       0.81      0.81      0.80       100
```

Out[79]:

	예측값	실제값
0	0	0
1	0	0
2	0	0
3	0	0
4	1	1
...	...	...
95	0	1
96	1	0
97	1	1
98	1	1
99	0	0

100 rows × 2 columns



5-(3). Random Forest Classifier

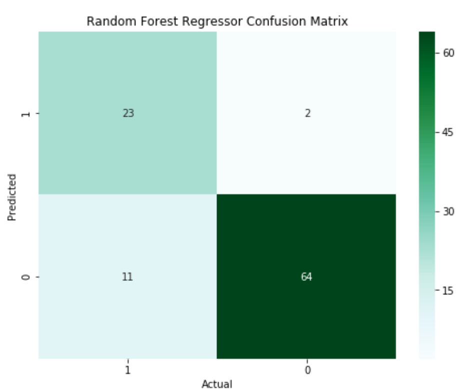
In [80]:
```
from sklearn.ensemble import RandomForestClassifier


Forest_reg = RandomForestClassifier(n_estimators=500, random_state=123123)
Forest_reg.fit(X_train, Y_train)
Y_pred = Forest_reg.predict(X_test)



Forest_Score = accuracy_score(Y_test, Y_pred.round())
Forest_Score_r = round(Forest_Score, 6)
print("Accuracy : " + str(Forest_Score_r))


plt.figure(figsize=(8,6))


cm = confusion_matrix(Y_test, Y_pred.round())
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred.round()).ravel()
cm = [[tp,fp],[fn,tn]]

RF_recall = tp / (tp + fn)
RF_spec = tn / (tn + fp)
RF_prec = tp / (tp + fn)
RF_accu = (tp + tn)/ (tp + tn + fp + fn)
RF_f1 = (2 * tp) / (2 * tp + fp + fn)
RF_tn = tn
RF_tp = tp
RF_fn = fn
RF_fp = fp

sns.heatmap(cm, annot = True, fmt = "d", cmap="BuGn")
plt.xticks([0.5, 1.5], labels=[1,0])
plt.yticks([0.5, 1.5], labels=[1,0])
plt.title('Random Forest Regressor Confusion Matrix')
plt.xlabel('Actual')
plt.ylabel('Predicted')

report = classification_report(Y_test, Y_pred.round())
print(report)

RFChart = pd.DataFrame()
RFChart['예측값'] = np.round(Y_pred)
RFChart['실제값'] = np.array(Y_test)
RFChart.loc[::1]
```

```
Accuracy : 0.87
              precision    recall  f1-score   support

           0       0.85      0.97      0.91        66
           1       0.92      0.68      0.78        34

    accuracy                           0.87       100
   macro avg       0.89      0.82      0.84       100
weighted avg       0.88      0.87      0.86       100
```

Out[80]:

	예측값	실제값
0	0	0
1	0	0
2	0	0
3	0	0
4	1	1
...	...	...
95	0	1
96	0	0
97	1	1
98	0	1
99	0	0

100 rows × 2 columns



5-(4). Naïve Bayes Classifier

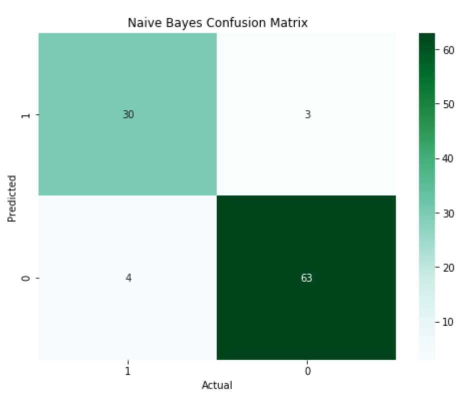
In [81]:
```python
from sklearn.naive_bayes import GaussianNB

NB_classifier = GaussianNB()
NB_classifier.fit(X_train, Y_train)
Y_pred = NB_classifier.predict(X_test)

NB_score = accuracy_score(Y_test, Y_pred)
print("Accuracy : " + str(NB_score))

plt.figure(figsize=(8,6))

cm = confusion_matrix(Y_test, Y_pred)
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred).ravel()
cm = [[tp,fp],[fn,tn]]

NB_recall = tp / (tp + fn)
NB_spec = tn / (tn + fp)
NB_prec = tp / (tp + fn)
NB_accu = (tp + tn)/ (tp + tn + fp + fn)
NB_f1 = (2 * tp) / (2 * tp + fp + fn)
NB_tn = tn
NB_tp = tp
NB_fn = fn
NB_fp = fp


sns.heatmap(cm, annot = True, fmt = "d", cmap="BuGn")
plt.xticks([0.5, 1.5], labels=[1,0])
plt.yticks([0.5, 1.5], labels=[1,0])
plt.title('Naive Bayes Confusion Matrix')
plt.xlabel('Actual')
plt.ylabel('Predicted')

report = classification_report(Y_test, Y_pred)
print(report)

NBChart = pd.DataFrame()
NBChart['예측값'] = np.round(Y_pred)
NBChart['실제값'] = np.array(Y_test)
NBChart.loc[::1]
```

```
Accuracy : 0.93
              precision    recall  f1-score   support

           0       0.94      0.95      0.95        66
           1       0.91      0.88      0.90        34

    accuracy                           0.93       100
   macro avg       0.92      0.92      0.92       100
weighted avg       0.93      0.93      0.93       100
```

Out[81]:

	예측값	실제값
0	0	0
1	0	0
2	0	0
3	0	0
4	1	1
...	...	...
95	1	1
96	0	0
97	1	1
98	0	1
99	0	0

100 rows × 2 columns



5-(5). Support Vector Machine

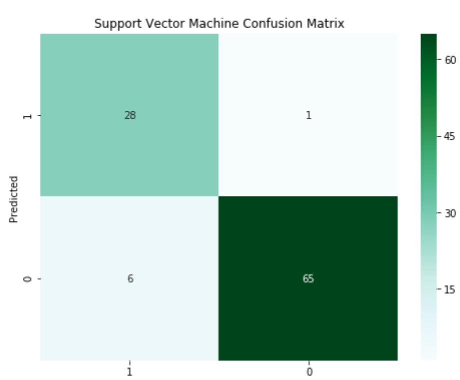
In [82]:
```
from sklearn.svm import SVC

SVM_classifier = SVC(kernel = 'linear')
SVM_classifier.fit(X_train, Y_train)
Y_pred = SVM_classifier.predict(X_test)

SVM_score = accuracy_score(Y_test, Y_pred)
print("Accuracy : " + str(SVM_score))


plt.figure(figsize=(8,6))

cm = confusion_matrix(Y_test, Y_pred)
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred).ravel()
cm = [[tp,fp],[fn,tn]]

SVM_recall = tp / (tp + fn)
SVM_spec = tn / (tn + fp)
SVM_prec = tp / (tp + fn)
SVM_accu = (tp + tn)/ (tp + tn + fp + fn)
SVM_f1 = (2 * tp) / (2 * tp + fp + fn)
SVM_tn = tn
SVM_tp = tp
SVM_fn = fn
SVM_fp = fp



sns.heatmap(cm, annot = True, fmt = "d", cmap="BuGn")
plt.xticks([0.5, 1.5], labels=[1,0])
plt.yticks([0.5, 1.5], labels=[1,0])
plt.title('Support Vector Machine Confusion Matrix')
plt.xlabel('Actual')
plt.ylabel('Predicted')

report = classification_report(Y_test, Y_pred)
print(report)

SVMChart = pd.DataFrame()
SVMChart['예측값'] = np.round(Y_pred)
SVMChart['실제값'] = np.array(Y_test)
SVMChart.loc[::1]
```

```
Accuracy : 0.93
              precision    recall  f1-score   support

           0       0.92      0.98      0.95        66
           1       0.97      0.82      0.89        34

    accuracy                           0.93       100
   macro avg       0.94      0.90      0.92       100
weighted avg       0.93      0.93      0.93       100
```

Out[82]:

	예측값	실제값
0	0	0
1	0	0
2	0	0
3	0	0
4	1	1
...	...	...
95	0	1
96	0	0
97	1	1
98	0	1
99	0	0

100 rows × 2 columns



5-마. 분류 결과 비교표 및 그래프

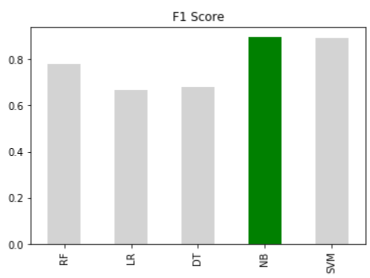
In [83]:
```python
class_data = {
        'Random Forest': [RF_tn, RF_tp, RF_fn, RF_fp, RF_recall, RF_spec, RF_prec, RF_accu, RF_f1],
        'Log Regression' : [LR_tn, LR_tp, LR_fn, LR_fp, LR_recall, LR_spec, LR_prec, LR_accu, LR_f1],
        'Decision Tree' : [DT_tn, DT_tp, DT_fn, DT_fp, DT_recall, DT_spec, DT_prec, DT_accu, DT_f1],
        'Naive Bayes' : [NB_tn, NB_tp, NB_fn, NB_fp, NB_recall, NB_spec, NB_prec, NB_accu, NB_f1],
        'SVM' : [SVM_tn, SVM_tp, SVM_fn, SVM_fp, SVM_recall, SVM_spec, SVM_prec, SVM_accu, SVM_f1]
         }

class_summary_df = pd.DataFrame(class_data, columns = ['Random Forest',
                                           'Log Regression',
                                           'Decision Tree',
                                            'Naive Bayes',
                                           'SVM'], index = ['TN', 'TP', 'FN', 'FP', 'Recall', 'Specificity', 'Precision', 'Accuracy', 'F1 score'])

class_summary_df.style
```

Out[83]:

	Random Forest	Log Regression	Decision Tree	Naive Bayes	SVM
TN	64	62	61	63	65
TP	23	19	20	30	28
FN	11	15	14	4	6
FP	2	4	5	3	1
Recall	0.676471	0.558824	0.588235	0.882353	0.823529
Specificity	0.969697	0.939394	0.924242	0.954545	0.984848
Precision	0.676471	0.558824	0.588235	0.882353	0.823529
Accuracy	0.87	0.81	0.81	0.93	0.93
F1 score	0.779661	0.666667	0.677966	0.895522	0.888889

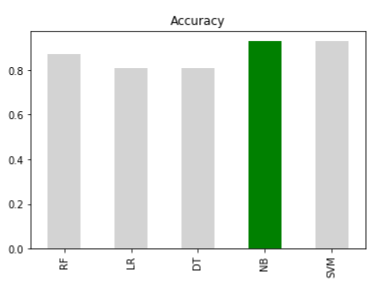
In [84]:
```python
ClassBar = pd.DataFrame(
    {'F1 Score': [RF_f1, LR_f1, DT_f1, NB_f1, SVM_f1]},
    index = ['RF', 'LR', 'DT', 'NB', 'SVM'])

ClassBar['F1 Score'].plot(kind="bar", color =['lightgrey', 'lightgrey', 'lightgrey', 'green', 'lightgrey', 'lightgrey'])

plt.title("F1 Score")
```

Out[84]: Text(0.5, 1.0, 'F1 Score')



In [85]:
```python
ClassBar = pd.DataFrame(
    {'Accuracy': [RF_accu, LR_accu, DT_accu, NB_accu, SVM_accu]},
    index = ['RF', 'LR', 'DT', 'NB', 'SVM'])

ClassBar['Accuracy'].plot(kind="bar", color =['lightgrey', 'lightgrey', 'lightgrey', 'green', 'lightgrey', 'lightgrey'])

plt.title("Accuracy")
```

Out[85]: Text(0.5, 1.0, 'Accuracy')



5-바. 합격 여부 예측하기 (가상 점수 입력)

In [86]:
```
aa = int(input ("Enter your GRE score (out of 340) : "))

bb = int(input ("Enter your TOEFL score (out of 120) : "))

cc = float(input ("Enter your College GPA score (out of 10 decimal place) : "))



if NB_classifier.predict([[aa, bb, cc]]) == 1:
    print("Congrats! You are admitted!")
else:
    print("We regret to inform you that your application is no longer under consideration")


print(NB_classifier.predict([[aa, bb, cc]]))
```

```
Enter your GRE score (out of 340) : 330
Enter your TOEFL score (out of 120) : 110
Enter your College GPA score (out of 10 decimal place) : 9
Congrats! You are admitted!
[1]
```

In [87]:
```
aa = int(input ("Enter your GRE score (out of 340) : "))

bb = int(input ("Enter your TOEFL score (out of 120) : "))

cc = float(input ("Enter your College GPA score (out of 10 decimal place) : "))



if NB_classifier.predict([[aa, bb, cc]]) == 1:
    print("Congrats! You are admitted!")
else:
    print("We regret to inform you that your application is no longer under consideration")


print(NB_classifier.predict([[aa, bb, cc]]))
```

```
Enter your GRE score (out of 340) : 200
Enter your TOEFL score (out of 120) : 90
Enter your College GPA score (out of 10 decimal place) : 5
We regret to inform you that your application is no longer under consideration
[0]
```

In [ ]:
```

```