This project aims to develop a predictive model for Master's program applications, leveraging historical data and machine learning techniques. By analyzing factors such as academic records, standardized test scores, recommendation letters, and personal statements, the model will provide a probabilistic assessment of an applicant's likelihood of acceptance.

Our dataset encompasses multiple crucial parameters that hold significance during the application process for Masters Programs.

```
The objective of this project is to develop a predictive model capable of estimating the
likelihood of admission into these Masters Programs. -This dataset was built with the purpose of
helping students in shortlisting universities with their profiles. The predicted output gives
them a fair idea about their chances for a particular university
```

In [37]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [38]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [39]:

```
df = pd.read_csv('Admission_Predict.csv')
df.head()
```

Out[39]:

	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

In [40]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

In [41]:

```
df.isnull().sum()
```

Out[41]:

```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [42]:

```
df.describe()
```

Out[42]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.724350
std	115.614301	11.473646	6.069514	1.143728	1.006869	0.898478	0.596317	0.498362	0.142609

min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.000000	0.340000
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.170000	0.000000	0.640000
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.000000	0.730000
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.062500	1.000000	0.830000
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000000	0.970000

In [43]:
```
df.drop('Serial No.',inplace = True,axis=1)
```

In [44]:
```
df.head()
```

Out[44]:

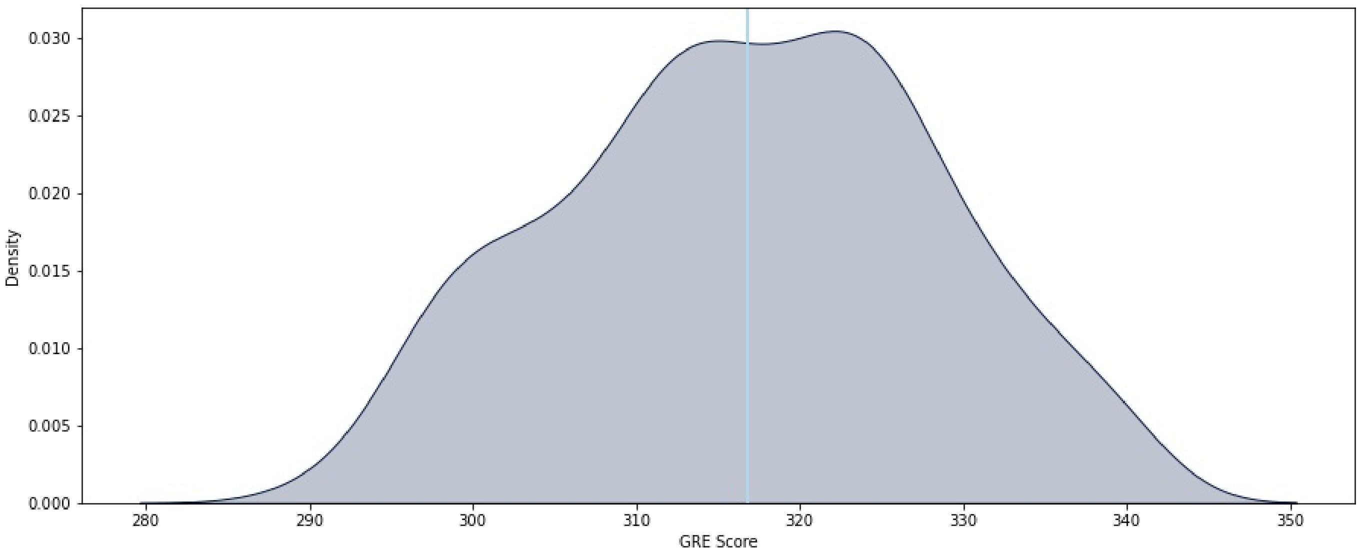
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

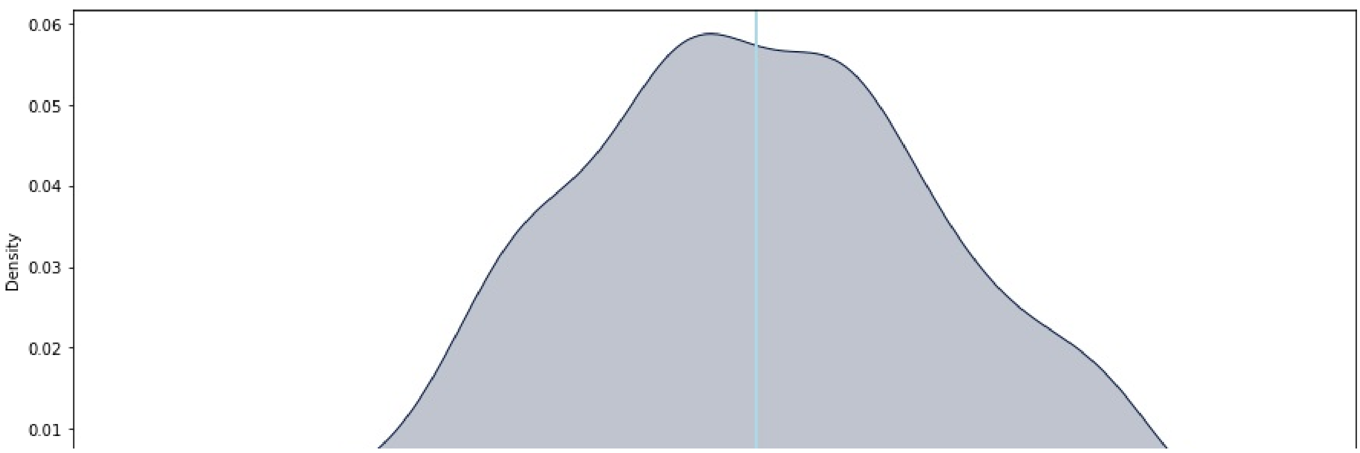
In [45]:
```
cols = df.columns
cols
```

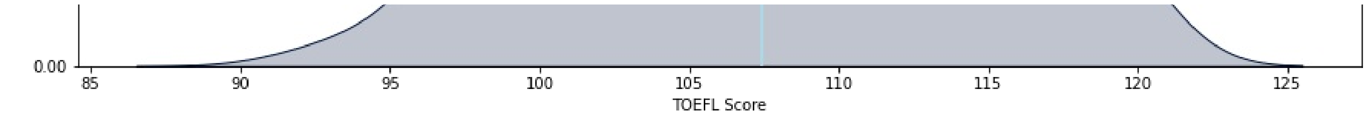
Out[45]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

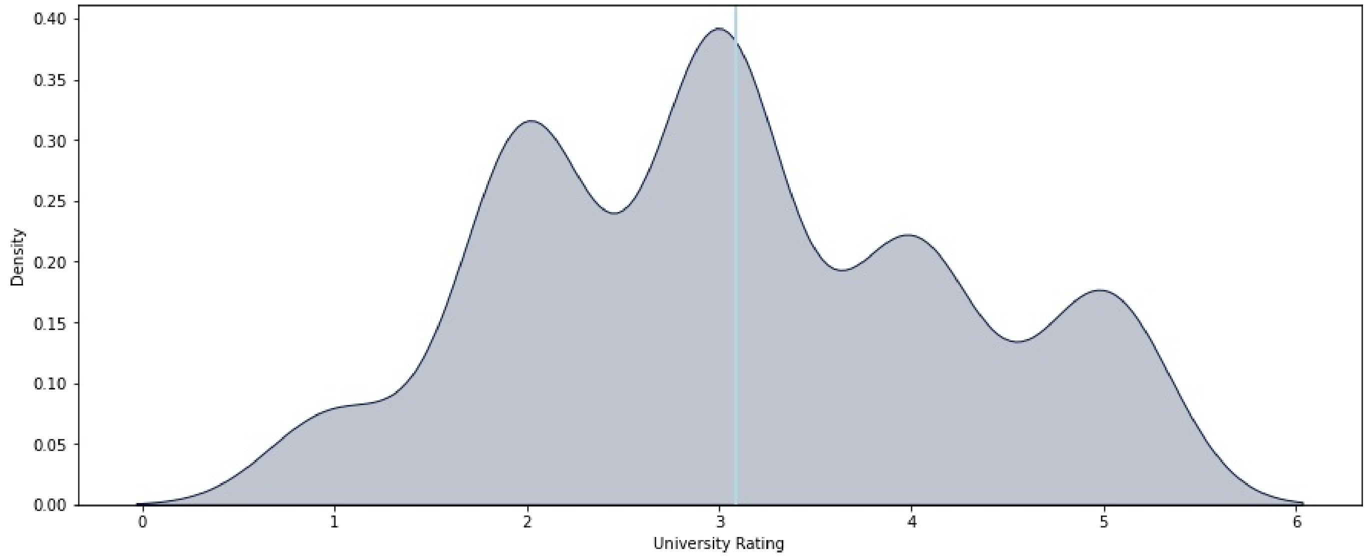
In [46]:
```
palette = ["#01153E","#ADD8E6","#136F63","#F72585","#FFBA08"]
```

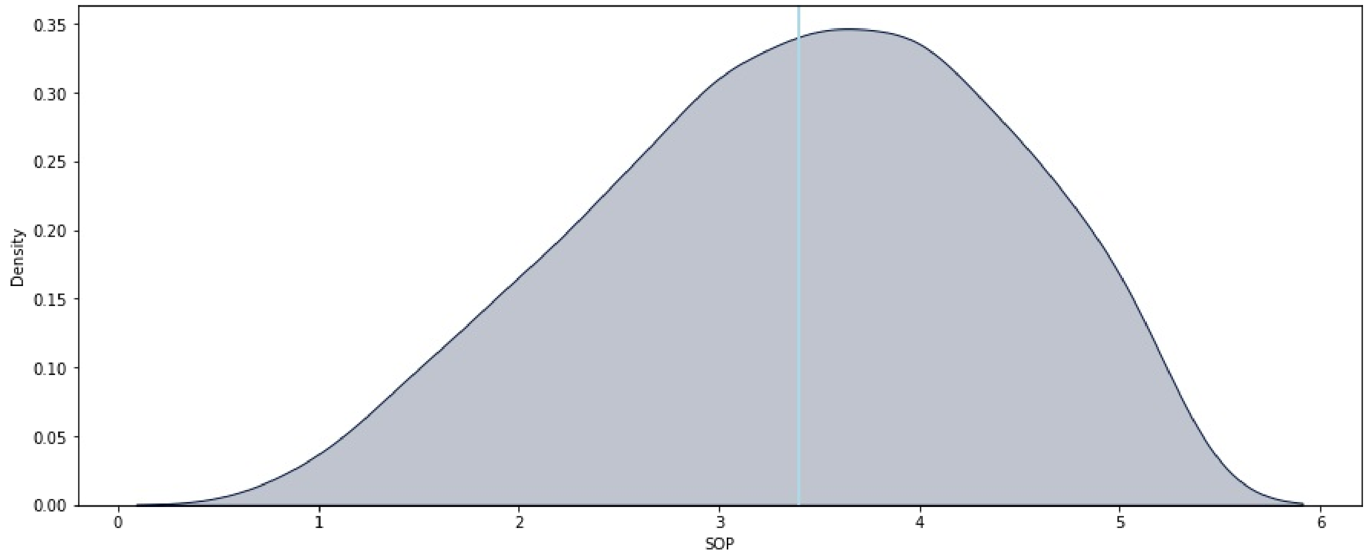
In [47]:
```
for n in cols :
  fig, ax = plt.subplots(1, 1, figsize=(15, 6))
  sns.kdeplot(df[n], color=palette[0], ax=ax, fill = True)
  ax.axvline(df[n].mean(), color=palette[1], linewidth=2)

  plt.show()
```

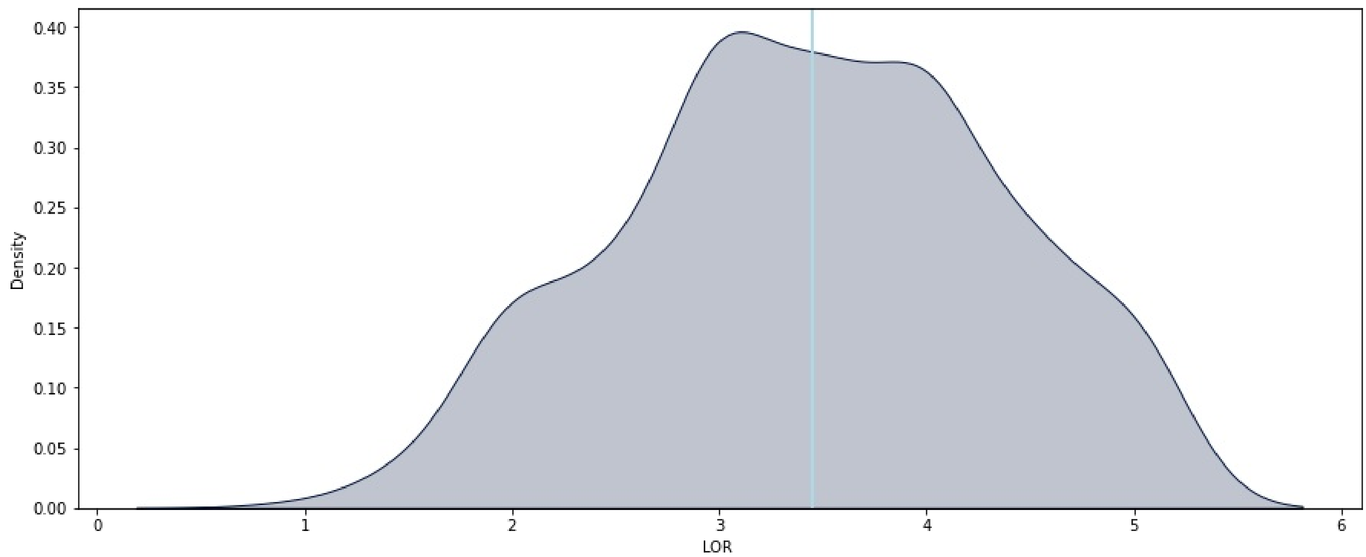


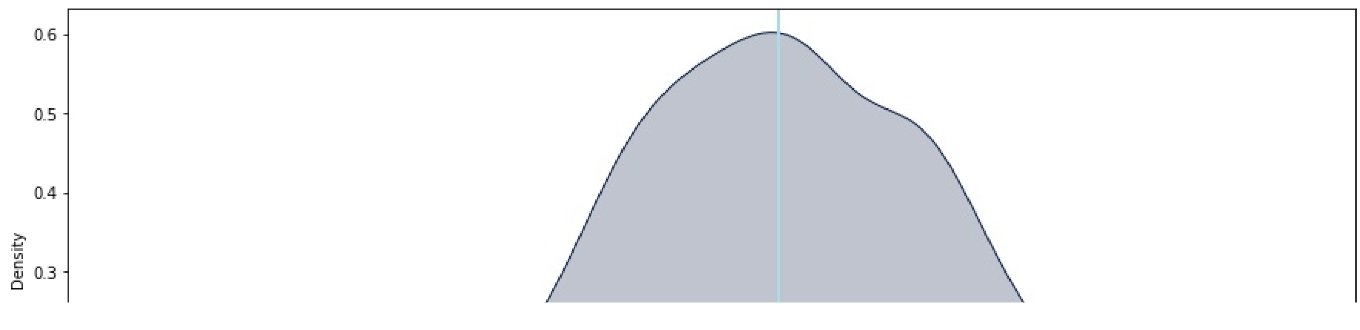


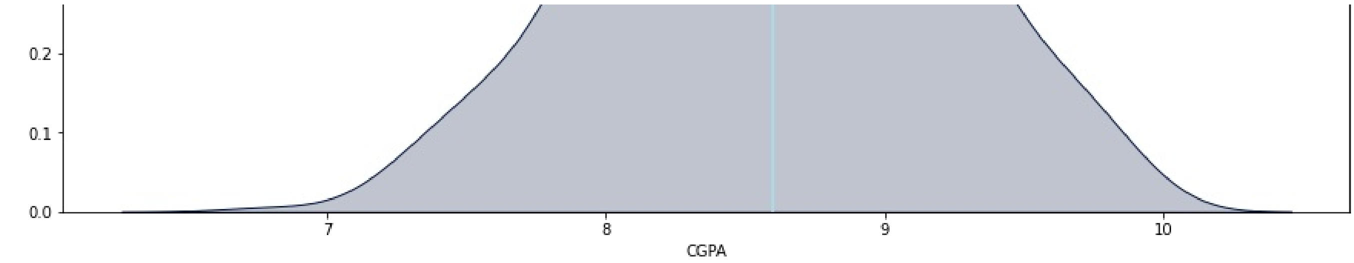

0.00
85
90
95
100
105
110
115
120
125
TOEFL Score

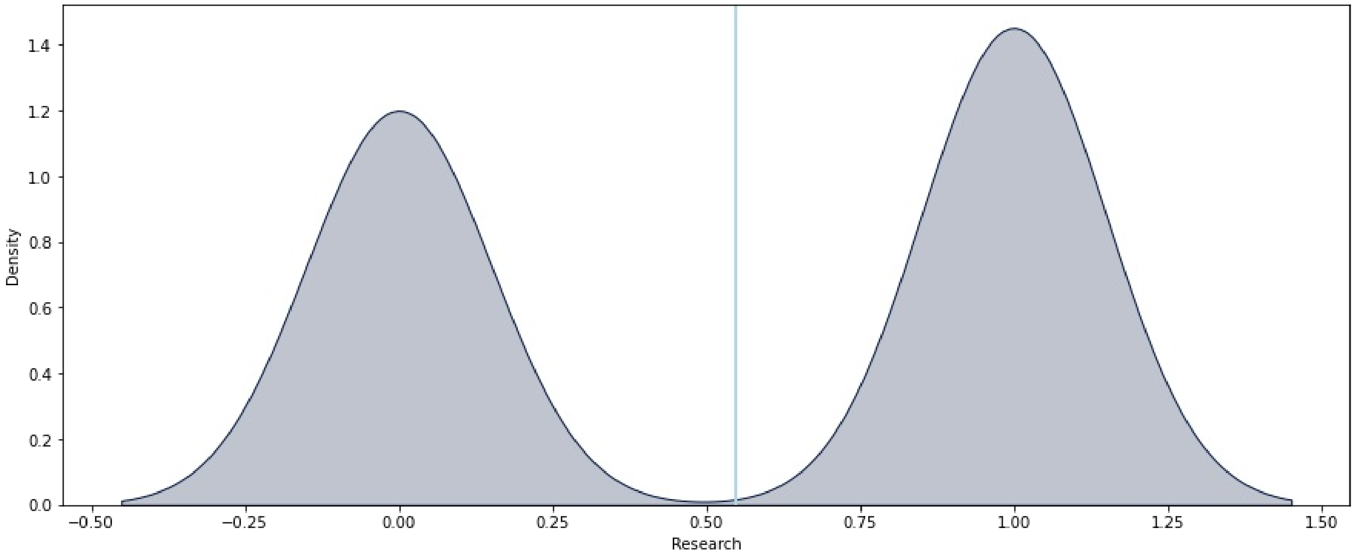

Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
University Rating

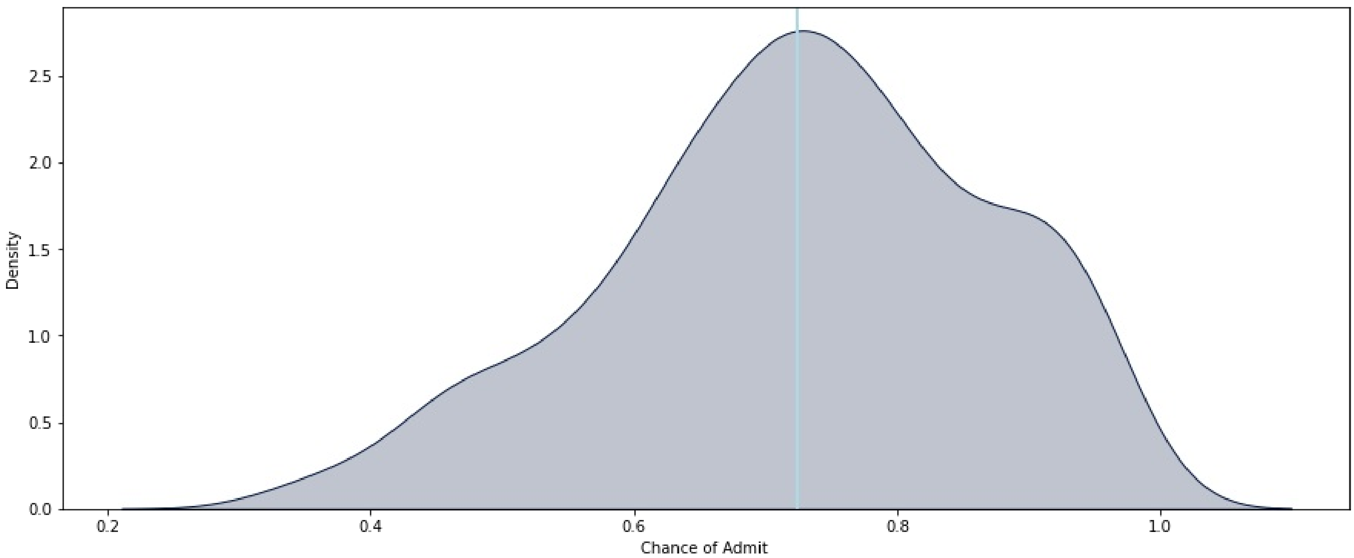

Density
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
SOP


Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
LOR


Density
0.6
0.5
0.4
0.3







In [48]:

```
from scipy.stats import skew

skew_data = pd.DataFrame(data = cols, columns=['Features'])
skewness = []
for n in cols :
  skewness.append(skew(df[n]))

skew_data['Skewness'] = skewness
skew_data
```

Out[48]:

	Features	Skewness
0	GRE Score	-0.062657
1	TOEFL Score	0.057001
2	University Rating	0.170617
3	SOP	-0.274726
4	LOR	-0.106590
5	CGPA	-0.065743
6	Research	-0.190863
7	Chance of Admit	-0.352121

The purpose of this code is to assess the skewness of the data distribution in each feature. Skewness is a measure of the asymmetry of a distribution. A skewness value of 0 indicates a perfectly symmetric distribution. As you see all features have skewness values close to 0, that's pretty good.

I will use value_counts() to return a count of unique values in that column, along with their respective frequencies. I'll use it for :

```
University Rating
Statement of Purpose
Letter of Recommendation Strength
```

In [49]:
```
ur = df['University Rating'].value_counts()
ur
```

Out[49]:
```
3    133
2    107
4     74
5     60
1     26
Name: University Rating, dtype: int64
```

In [50]:
```
sop = df['SOP'].value_counts()
```

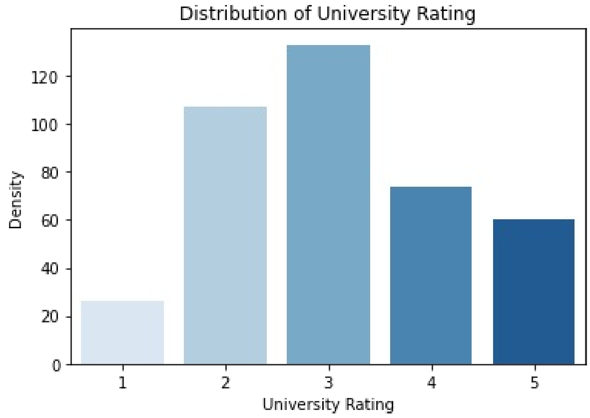
In [51]:
```
sop
```

Out[51]:
```
3.5    70
4.0    70
3.0    64
4.5    53
2.5    47
5.0    37
2.0    33
1.5    20
1.0     6
Name: SOP, dtype: int64
```

In [52]:
```
research = df['Research'].value_counts()
research
```

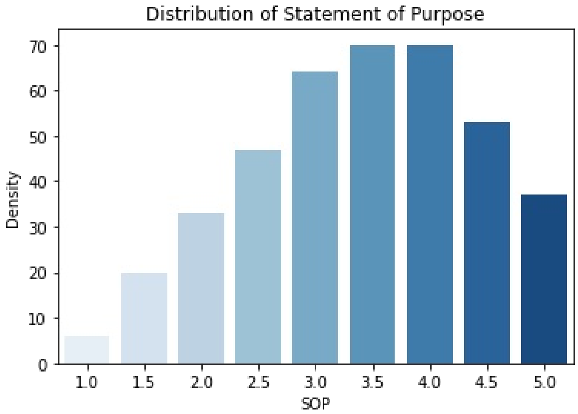
Out[52]:
```
1    219
0    181
Name: Research, dtype: int64
```

In [55]:
```
ax = sns.barplot(x= ur.index , y= ur.tolist(), data=df, palette= 'Blues')


plt.xlabel("University Rating")
plt.ylabel("Density")
plt.title("Distribution of University Rating")

plt.show()
```

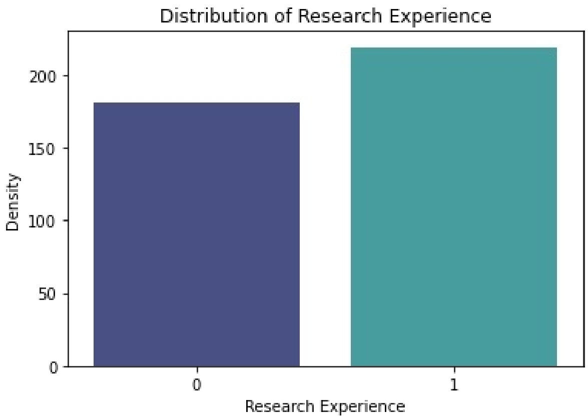


In [58]:
```
ax = sns.barplot(x= sop.index , y= sop.tolist(), data=df, palette= 'Blues')
```

```
plt.xlabel("SOP")
plt.ylabel("Density")
plt.title("Distribution of Statement of Purpose")

plt.show()
```

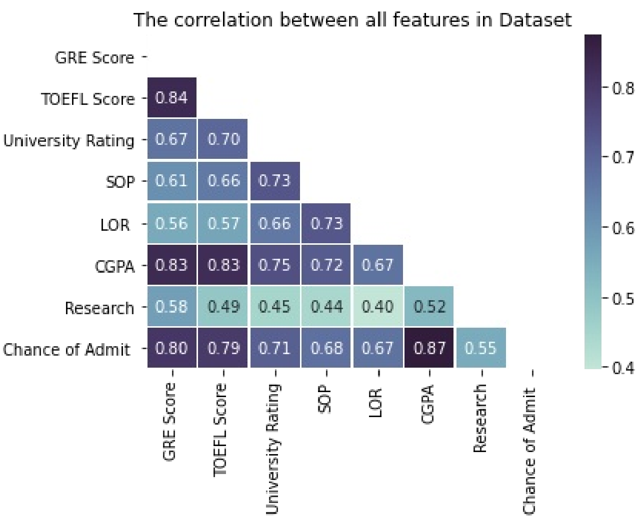


In [63]:
```
ax = sns.barplot(x= research.index , y= research.tolist(), data=df, palette= 'mako')


plt.xlabel("Research Experience")
plt.ylabel("Density")
plt.title("Distribution of Research Experience")

plt.show()
```



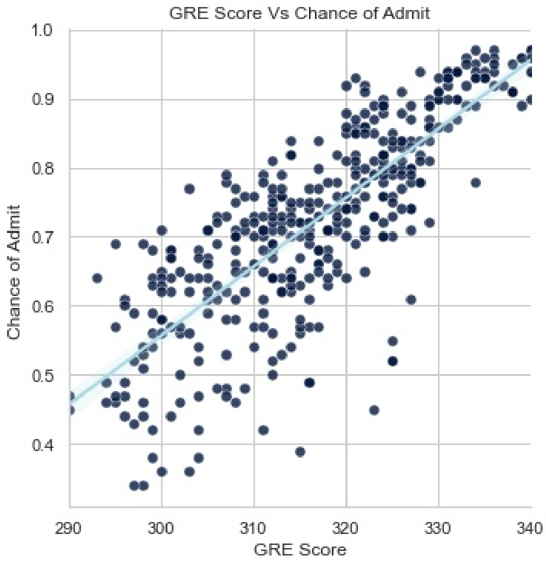
In [64]:
```
corr= df.corr()
```

In [65]:
```
dropSelf = np.zeros_like(corr)
dropSelf[np.triu_indices_from(dropSelf)] = True
sns.heatmap(corr, annot=True,linewidths=.5, fmt=".2f", cmap=sns.cubehelix_palette(start=.5, rot=-.5, as_cmap=True

plt.title("The correlation between all features in Dataset")
plt.show()
```



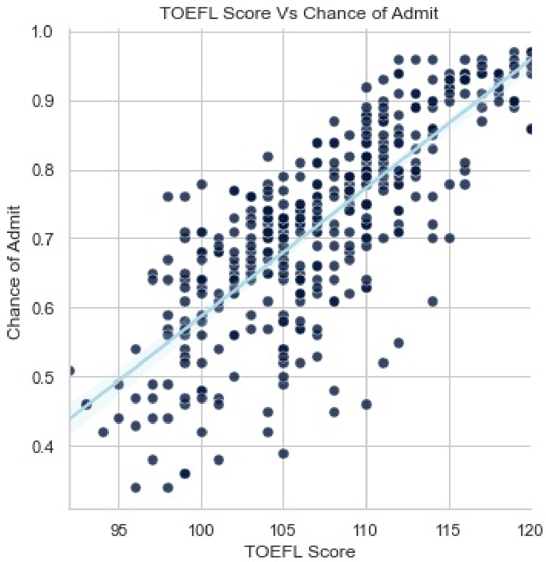
In [66]:

```
sns.set(style="whitegrid", color_codes=True)

sns.lmplot(x='GRE Score', y='Chance of Admit ', data= df,
          scatter_kws={'s': 50, 'linewidth': 0.5, 'color' : palette[0], 'edgecolor': 'w'},
          line_kws = {'color' : palette[1]})

plt.title("GRE Score Vs Chance of Admit")
plt.show()
```



In [68]:

```
sns.set(style="whitegrid", color_codes=True)

sns.lmplot(x='TOEFL Score', y='Chance of Admit ', data= df,
          scatter_kws={'s': 50, 'linewidth': 0.5, 'color' : palette[0], 'edgecolor': 'w'},
          line_kws = {'color' : palette[1]})

plt.title("TOEFL Score Vs Chance of Admit")
plt.show()
```

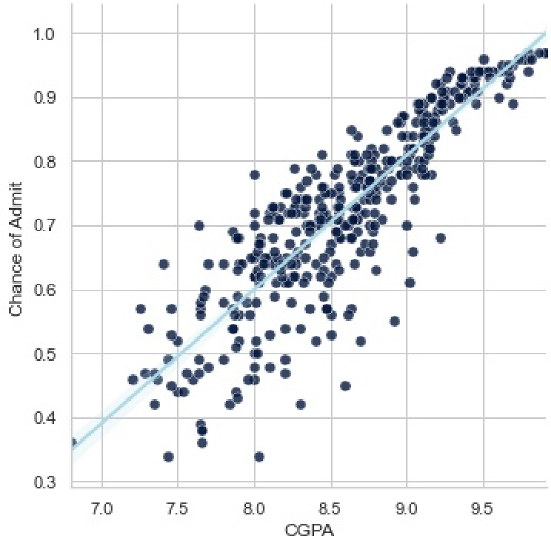


In [69]:

```
sns.set(style="whitegrid", color_codes=True)

sns.lmplot(x='CGPA', y='Chance of Admit ', data= df,
          scatter_kws={'s': 50, 'linewidth': 0.5, 'color' : palette[0], 'edgecolor': 'w'},
          line_kws = {'color' : palette[1]})

plt.title("CGPA Vs Chance of Admit")
plt.show()
```

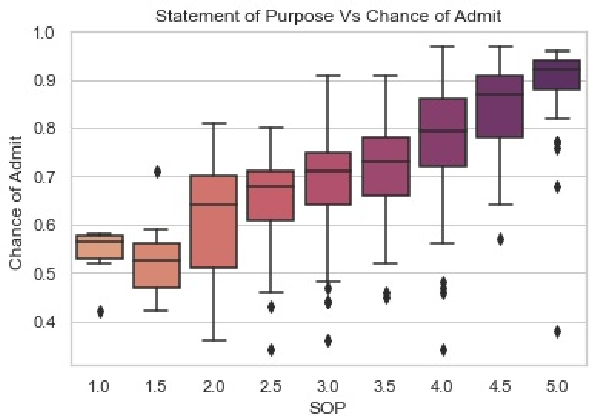
```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:238: RuntimeWarning: Glyph 9 missin
g from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:201: RuntimeWarning: Glyph 9 missin
g from current font.
  font.set_text(s, 0, flags=flags)
```

CGPA□Vs Chance of Admit



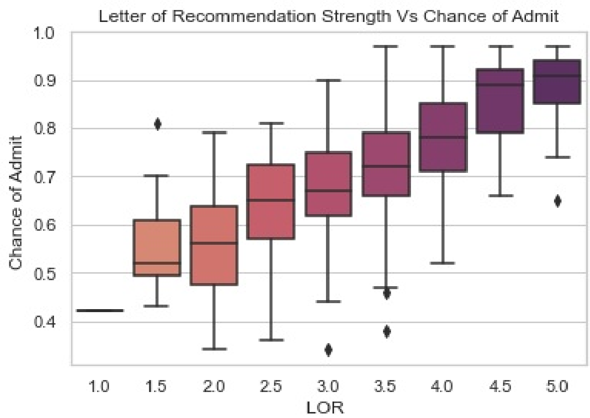
In [70]:

```python
sns.boxplot(x= 'SOP', y='Chance of Admit ', data= df, palette = 'flare')

plt.title("Statement of Purpose Vs Chance of Admit")
plt.show()
```



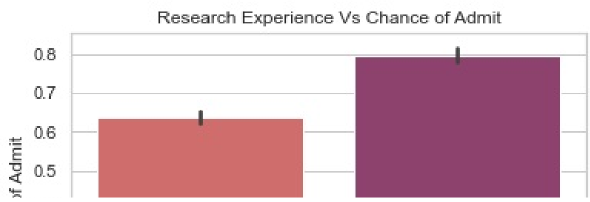
In [71]:

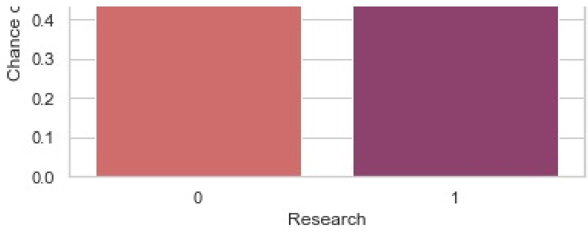
```python
sns.boxplot(x= 'LOR ', y='Chance of Admit ', data= df, palette = 'flare')

plt.title("Letter of Recommendation Strength Vs Chance of Admit")
plt.show()
```



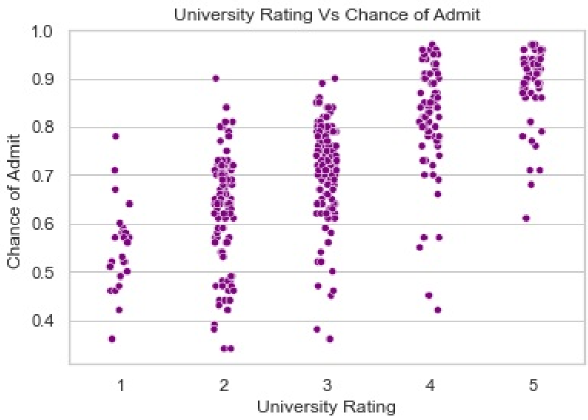
In [72]:

```python
sns.barplot(x= 'Research', y= 'Chance of Admit ', data= df, estimator=np.mean, palette ='flare')

plt.title("Research Experience Vs Chance of Admit")
plt.show()
```





In [73]:
```
sns.stripplot(x= 'University Rating', y='Chance of Admit ', data= df, jitter=True, dodge= True,
              color ='purple', edgecolor= 'white', linewidth= 0.5 )

plt.title("University Rating Vs Chance of Admit")
plt.show()
```



In [74]:
```
df.columns
```

Out[74]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [75]:
```
features = df[['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit ']]
# we create a new data frame which will include all the VIFs
# note that each variable has its own variance inflation factor as this measure is variable specific (not model s
vif = pd.DataFrame()

# here we make use of the variance_inflation_factor, which will basically output the respective VIFs
vif["VIF"] = [variance_inflation_factor(features.values, i) for i in range(features.shape[1])]
# Finally, I like to include names so it is easier to explore the result
vif["Features"] = features.columns
```

In [76]:
```
vif
```

Out[76]:

	VIF	Features
0	1607.928316	GRE Score
1	1373.804681	TOEFL Score
2	22.998812	University Rating
3	38.051007	SOP
4	39.774185	LOR
5	1333.886926	CGPA
6	3.211789	Research
7	108.476950	Chance of Admit

In [78]:
```
X = df.drop(['Chance of Admit '], axis=1)
y = df['Chance of Admit ']
```

In [79]:
```
X
```

Out[79]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1

2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0
...	...	...	...	...	...	...	...
395	324	110	3	3.5	3.5	9.04	1
396	325	107	3	3.0	3.5	9.11	1
397	330	116	4	5.0	4.5	9.45	1
398	312	103	3	3.5	4.0	8.78	0
399	333	117	4	5.0	4.0	9.66	1

400 rows × 7 columns

In [80]:

```
y
```

Out[80]:

```
0      0.92
1      0.76
2      0.72
3      0.80
4      0.65
       ...
395    0.82
396    0.84
397    0.91
398    0.67
399    0.95
Name: Chance of Admit , Length: 400, dtype: float64
```

In [81]:

```
# Import the scaling module
from sklearn.preprocessing import StandardScaler

# Create a scaler object
scaler = StandardScaler()
# Fit the inputs (calculate the mean and standard deviation feature-wise)
scaler.fit(X)
```

Out[81]:

```
StandardScaler()
```

In [82]:

```
inputs_scaled = scaler.transform(X)
```

In [83]:

```
from sklearn.model_selection import train_test_split
```

In [84]:

```
# Import the module for the split
from sklearn.model_selection import train_test_split

# Split the variables with an 80-20 split and some random state
# To have the same split as mine, use random_state = 365
X_train, X_test, y_train, y_test = train_test_split(inputs_scaled, y, test_size=0.2, random_state=365)
```

In [86]:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean_squared_error
```

In [89]:

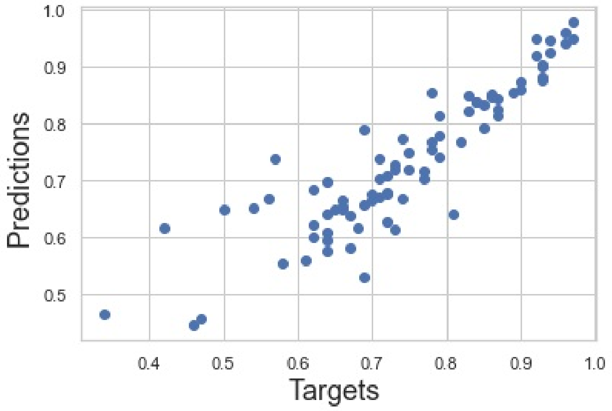
```
models = [['Linear Regression :', LinearRegression()],
          ['DecisionTree :',DecisionTreeRegressor()],
          ['RandomForest :',RandomForestRegressor()],
          ['KNeighbours :', KNeighborsRegressor(n_neighbors = 2)]]
```

In [90]:

```
model = models[0][1]
name =  models[0][0]

model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(name, (np.sqrt(mean_squared_error(y_test, predictions))))


# The simplest way to compare the targets (y_train) and the predictions (y_hat) is to plot them on a scatter plot
# The closer the points to the 45-degree line, the better the prediction
plt.scatter(y_test, predictions)
# Let's also name the axes
```

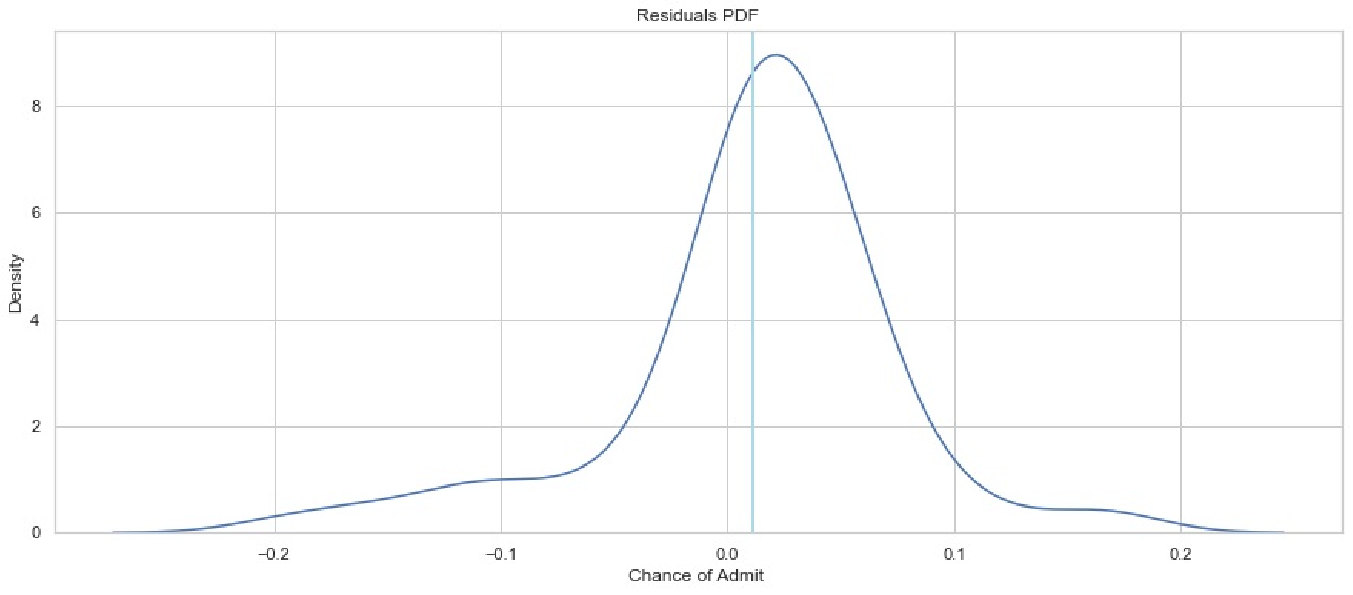
```python
plt.xlabel('Targets ',size=18)
plt.ylabel('Predictions ',size=18)
# Sometimes the plot will have different scales of the x-axis and the y-axis
# This is an issue as we won't be able to interpret the '45-degree line'
# We want the x-axis and the y-axis to be the same

plt.show()
```

Linear Regression : 0.061794978711531535



In [91]:
```python
# Another useful check of our model is a residual plot
# We can plot the PDF of the residuals and check for anomalies
fig, ax = plt.subplots(1, 1, figsize=(15, 6))
sns.kdeplot(y_test - predictions)
ax.axvline((y_test - predictions).mean(), color=palette[1], linewidth=2)


# Include a title
plt.title("Residuals PDF")

# In the best case scenario this plot should be normally distributed
# In our case we notice that there are a little residuals far away from the mean
```

Out[91]: Text(0.5, 1.0, 'Residuals PDF')



In [92]:
```python
model.score(X_test,y_test)
```

Out[92]: 0.7953537334455247

In [93]:
```python
df_reg = pd.DataFrame(predictions, columns=['Prediction'])
df_reg['Target'] = y_test
y_test = y_test.reset_index(drop=True)
df_reg['Target'] = y_test
df_reg['Residual'] = df_reg['Target'] - df_reg['Prediction']
df_reg['Difference%'] = np.absolute(df_reg['Residual']/df_reg['Target']*100)

# Sometimes it is useful to check these outputs manually
```

```python
# To see all rows, we use the relevant pandas syntax
pd.options.display.max_rows = 999
# Moreover, to make the dataset clear, we can display the result with only 2 digits after the dot
pd.set_option('display.float_format', lambda x: '%.2f' % x)
# Finally, we sort by difference in % and manually check the model
df_reg.sort_values(by=['Difference%'])
```

Out[93]:

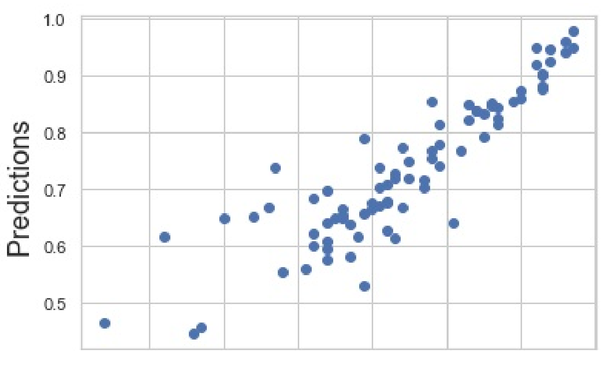
	Prediction	Target	Residual	Difference%
16	0.64	0.64	0.00	0.02
48	0.96	0.96	0.00	0.04
49	0.92	0.92	-0.00	0.05
17	0.75	0.75	0.00	0.10
30	0.84	0.84	0.00	0.11
59	0.62	0.62	-0.00	0.20
22	0.84	0.84	0.00	0.24
61	0.73	0.73	0.00	0.34
5	0.65	0.65	0.00	0.37
9	0.95	0.94	-0.01	0.71
44	0.66	0.66	-0.00	0.73
70	0.65	0.66	0.01	0.83
27	0.85	0.86	0.01	0.92
29	0.98	0.97	-0.01	1.00
2	0.82	0.83	0.01	1.03
38	0.70	0.71	0.01	1.13
64	0.77	0.78	0.01	1.53
35	0.72	0.73	0.01	1.56
21	0.71	0.72	0.01	1.56
10	0.78	0.79	0.01	1.58
19	0.85	0.86	0.01	1.62
31	0.92	0.94	0.02	1.71
34	0.94	0.96	0.02	1.84
54	0.65	0.66	0.01	2.02
63	0.95	0.97	0.02	2.03
39	0.83	0.85	0.02	2.21
24	0.85	0.83	-0.02	2.29
79	0.90	0.93	0.03	2.88
15	0.87	0.90	0.03	2.96
74	0.81	0.79	-0.02	2.98
68	0.84	0.87	0.03	3.01
47	0.95	0.92	-0.03	3.16
4	0.75	0.78	0.03	3.22
51	0.90	0.93	0.03	3.32
0	0.45	0.47	0.02	3.36
33	0.68	0.70	0.02	3.41
60	0.60	0.62	0.02	3.41
62	0.44	0.46	0.02	3.61
8	0.74	0.71	-0.03	3.84
36	0.85	0.89	0.04	3.97
32	0.72	0.75	0.03	4.02
67	0.77	0.74	-0.03	4.45
43	0.86	0.90	0.04	4.51
69	0.55	0.58	0.03	4.55
76	0.66	0.69	0.03	4.99
53	0.61	0.64	0.03	5.02
52	0.64	0.67	0.03	5.10
78	0.66	0.70	0.04	5.22
40	0.82	0.87	0.05	5.25

71	0.88	0.93	0.05	5.31
41	0.67	0.71	0.04	5.54
46	0.88	0.93	0.05	5.83
28	0.68	0.72	0.04	5.98
65	0.77	0.82	0.05	6.26
3	0.74	0.79	0.05	6.28
42	0.67	0.72	0.05	6.35
72	0.81	0.87	0.06	6.47
45	0.79	0.85	0.06	6.80
18	0.71	0.77	0.06	7.20
58	0.59	0.64	0.05	7.25
6	0.56	0.61	0.05	8.55
77	0.70	0.77	0.07	8.75
11	0.70	0.64	-0.06	8.96
66	0.62	0.68	0.06	9.52
37	0.85	0.78	-0.07	9.55
75	0.67	0.74	0.07	10.03
57	0.68	0.62	-0.06	10.23
20	0.57	0.64	0.07	10.38
13	0.63	0.72	0.09	12.92
55	0.58	0.67	0.09	13.37
56	0.79	0.69	-0.10	14.37
73	0.61	0.73	0.12	15.92
26	0.67	0.56	-0.11	18.90
14	0.65	0.54	-0.11	20.44
50	0.64	0.81	0.17	20.85
12	0.53	0.69	0.16	23.45
25	0.74	0.57	-0.17	29.30
1	0.65	0.50	-0.15	29.37
7	0.46	0.34	-0.12	36.39
23	0.61	0.42	-0.19	46.36

In [94]:

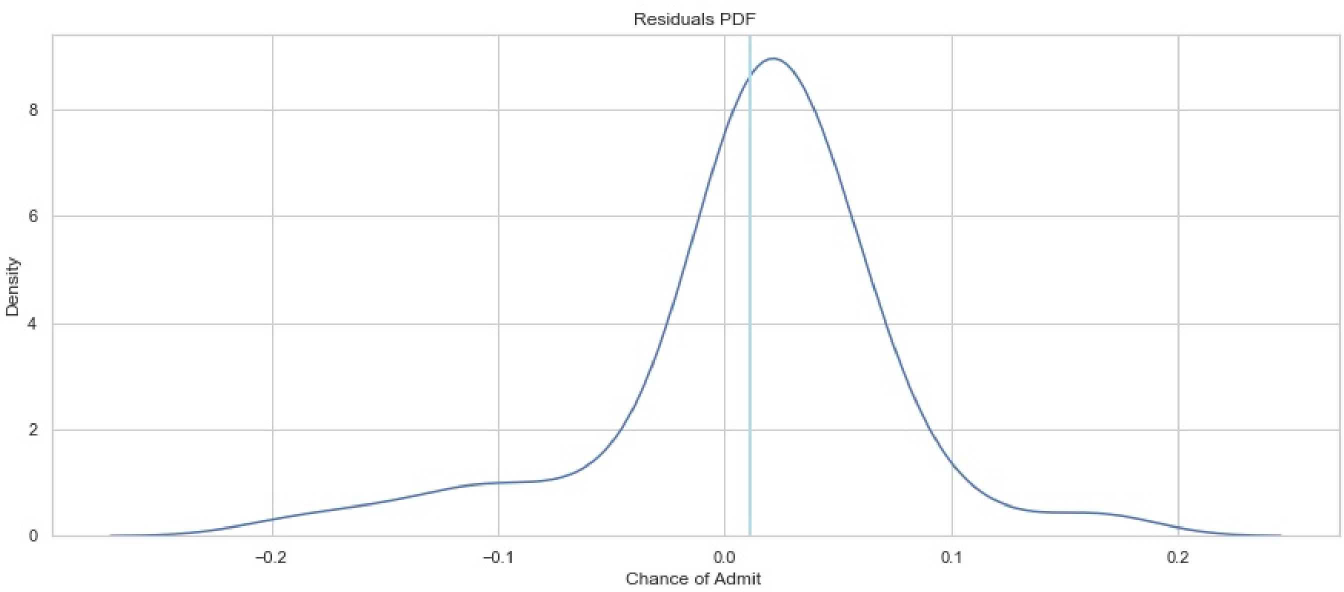
```
ame =  models[1][0]

model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(name, (np.sqrt(mean_squared_error(y_test, predictions))))


# The simplest way to compare the targets (y_train) and the predictions (y_hat) is to plot them on a scatter plot
# The closer the points to the 45-degree line, the better the prediction
plt.scatter(y_test, predictions)
# Let's also name the axes
plt.xlabel('Targets ',size=18)
plt.ylabel('Predictions ',size=18)
# Sometimes the plot will have different scales of the x-axis and the y-axis
# This is an issue as we won't be able to interpret the '45-degree line'
# We want the x-axis and the y-axis to be the same

plt.show()
```

```
Linear Regression : 0.061794978711531535
```



In [95]:
```python
# Another useful check of our model is a residual plot
# We can plot the PDF of the residuals and check for anomalies
fig, ax = plt.subplots(1, 1, figsize=(15, 6))
sns.kdeplot(y_test - predictions)
ax.axvline((y_test - predictions).mean(), color=palette[1], linewidth=2)


# Include a title
plt.title("Residuals PDF")

# In the best case scenario this plot should be normally distributed
# In our case we notice that there are a little residuals far away from the mean
# Given the definition of the residuals (y_test - predictions), negative values imply
# that predictions are much higher than
```

Out[95]: `Text(0.5, 1.0, 'Residuals PDF')`

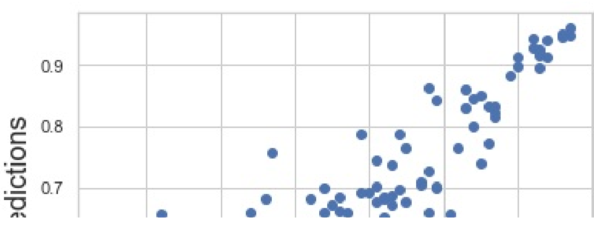


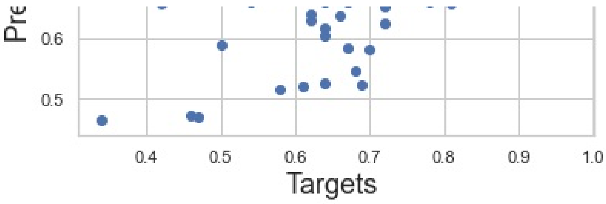
In [96]:
```python
model.score(X_test,y_test)
```

Out[96]: `0.7953537334455247`

In [97]:
```python
model = models[2][1]
name =  models[2][0]

model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(name, (np.sqrt(mean_squared_error(y_test, predictions))))


# The simplest way to compare the targets (y_train) and the predictions (y_hat) is to plot them on a scatter plot
# The closer the points to the 45-degree line, the better the prediction
plt.scatter(y_test, predictions)
# Let's also name the axes
plt.xlabel('Targets ',size=18)
plt.ylabel('Predictions ',size=18)
# Sometimes the plot will have different scales of the x-axis and the y-axis
# This is an issue as we won't be able to interpret the '45-degree line'
# We want the x-axis and the y-axis to be the same

plt.show()
```

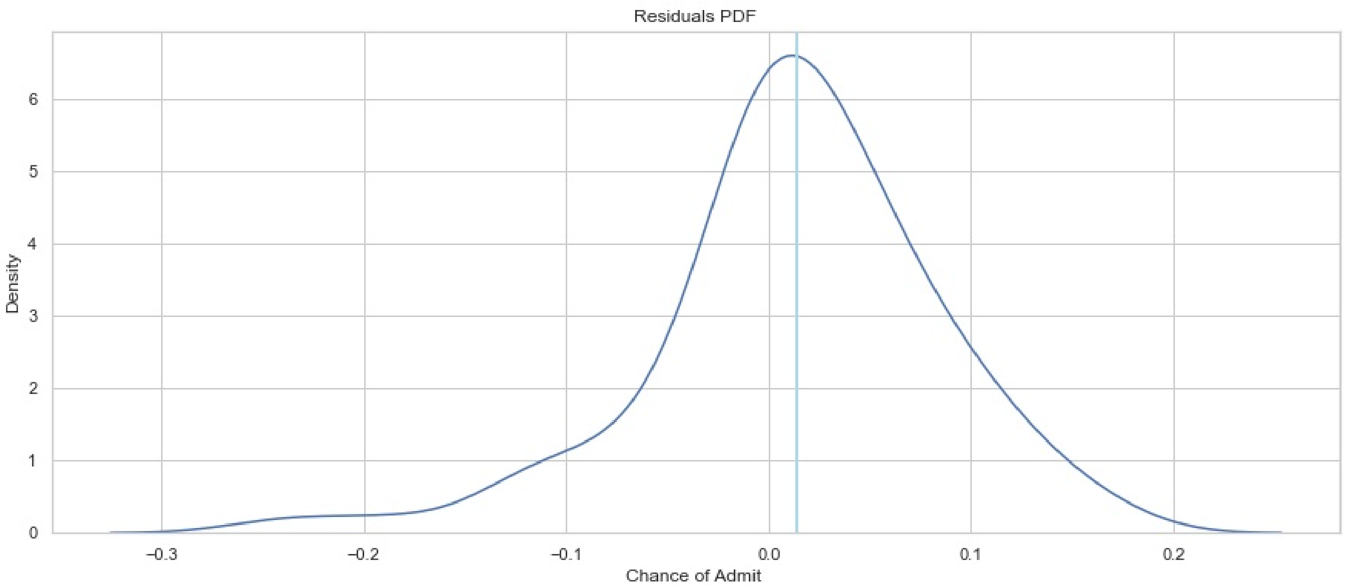
```
RandomForest : 0.07088587130028098
```





In [98]:
```
# Another useful check of our model is a residual plot
# We can plot the PDF of the residuals and check for anomalies
fig, ax = plt.subplots(1, 1, figsize=(15, 6))
sns.kdeplot(y_test - predictions)
ax.axvline((y_test - predictions).mean(), color=palette[1], linewidth=2)


# Include a title
plt.title("Residuals PDF")
```

Out[98]: `Text(0.5, 1.0, 'Residuals PDF')`

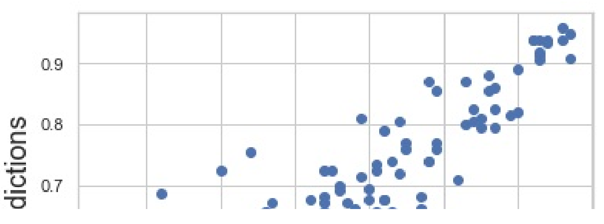


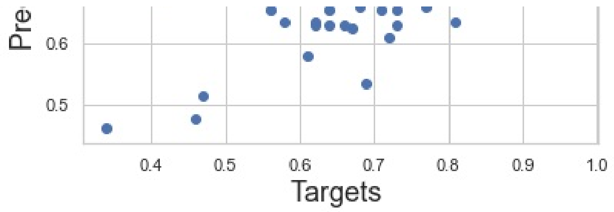
In [99]:
```
model.score(X_test,y_test)
```

Out[99]: `0.7307121146527218`

In [100…
```
model = models[3][1]
name =  models[3][0]

model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(name, (np.sqrt(mean_squared_error(y_test, predictions))))


# The simplest way to compare the targets (y_train) and the predictions (y_hat) is to plot them on a scatter plot
# The closer the points to the 45-degree line, the better the prediction
plt.scatter(y_test, predictions)
# Let's also name the axes
plt.xlabel('Targets ',size=18)
plt.ylabel('Predictions ',size=18)
# Sometimes the plot will have different scales of the x-axis and the y-axis
# This is an issue as we won't be able to interpret the '45-degree line'
# We want the x-axis and the y-axis to be the same

plt.show()
```

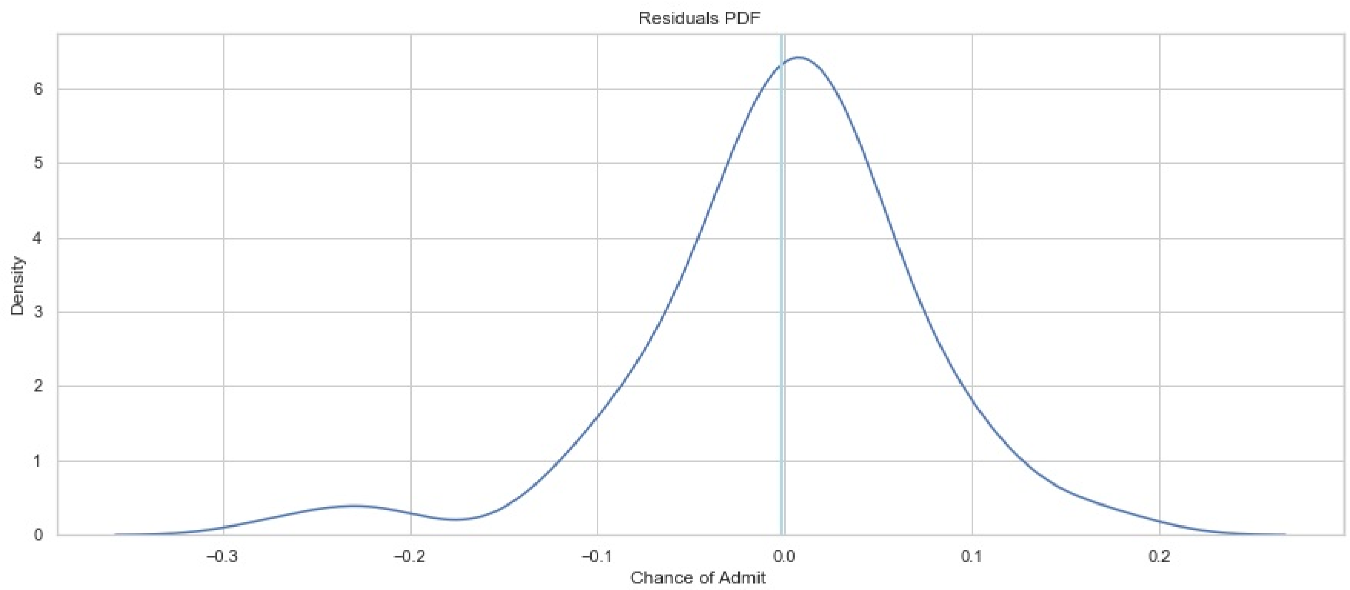
```
KNeighbours : 0.07347618661852288
```





In [101…
```
# Another useful check of our model is a residual plot
# We can plot the PDF of the residuals and check for anomalies
fig, ax = plt.subplots(1, 1, figsize=(15, 6))
sns.kdeplot(y_test - predictions)
ax.axvline((y_test - predictions).mean(), color=palette[1], linewidth=2)


# Include a title
plt.title("Residuals PDF")

# In the best case scenario this plot should be normally distributed
# In our case we notice that there are a little residuals far away from the mean
# Given the definition of the residuals (y_test - predictions), negative values imply
# that predictions are much higher than y
```

Out[101… Text(0.5, 1.0, 'Residuals PDF')



In [102…
```
model.score(X_test,y_test)
```

Out[102… 0.7106718639440972

The best model is K Neighbors

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js