Problem Statement

Jamboree has helped thousands to make it to top colleges abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort.

They recently launched a feature where students/learners can come to their website and check their probability of getting into the IVY league college. This feature estimates the chances of graduate admission from an Indian perspective.

In [1]:

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

In [2]:

```python
raw_data = pd.read_csv('Jamboree_Admission.csv')
```

In [3]:

```python
raw_data.head()
```

Out[3]:

	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 [4]:

```python
for i in raw_data.columns:
  print(f'Unique values in {i} feature are:')
  print(raw_data[i].unique())
  print("="*100)
```

```
Unique values in Serial No. feature are:
[   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
   19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
   37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
   55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
   73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
   91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
  109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
  127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
  145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
  163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
  181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
  199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
  217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
  235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
  253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270
  271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
  289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
  307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324
  325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342
  343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
  361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378
  379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396
```

```
 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396
 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414
 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432
 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450
 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468
 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486
 487 488 489 490 491 492 493 494 495 496 497 498 499 500]
=========================================================================================
===========
Unique values in GRE Score feature are:
[337 324 316 322 314 330 321 308 302 323 325 327 328 307 311 317 319 318
 303 312 334 336 340 298 295 310 300 338 331 320 299 304 313 332 326 329
 339 309 315 301 296 294 306 305 290 335 333 297 293]
=========================================================================================
===========
Unique values in TOEFL Score feature are:
[118 107 104 110 103 115 109 101 102 108 106 111 112 105 114 116 119 120
  98  93  99  97 117 113 100  95  96  94  92]
=========================================================================================
===========
Unique values in University Rating feature are:
[4 3 2 5 1]
=========================================================================================
===========
Unique values in SOP feature are:
[4.5 4.  3.  3.5 2.  5.  1.5 1.  2.5]
=========================================================================================
===========
Unique values in LOR  feature are:
[4.5 3.5 2.5 3.  4.  1.5 2.  5.  1. ]
=========================================================================================
===========
Unique values in CGPA feature are:
[9.65 8.87 8.   8.67 8.21 9.34 8.2  7.9  8.6  8.4  9.   9.1  8.3  8.7
 8.8  8.5  9.5  9.7  9.8  9.6  7.5  7.2  7.3  8.1  9.4  9.2  7.8  7.7
 9.3  8.85 7.4  7.6  6.8  8.92 9.02 8.64 9.22 9.16 9.64 9.76 9.45 9.04
 8.9  8.56 8.72 8.22 7.54 7.36 8.02 9.36 8.66 8.42 8.28 8.14 8.76 7.92
 7.66 8.03 7.88 7.84 8.96 9.24 8.88 8.46 8.12 8.25 8.47 9.05 8.78 9.18
 9.46 9.38 8.48 8.68 8.34 8.45 8.62 7.46 7.28 8.84 9.56 9.48 8.36 9.32
 8.71 9.35 8.65 9.28 8.77 8.16 9.08 9.12 9.15 9.44 9.92 9.11 8.26 9.43
 9.06 8.75 8.89 8.69 7.86 9.01 8.97 8.33 8.27 7.98 8.04 9.07 9.13 9.23
 8.32 8.98 8.94 9.53 8.52 8.43 8.54 9.91 9.87 7.65 7.89 9.14 9.66 9.78
 9.42 9.26 8.79 8.23 8.53 8.07 9.31 9.17 9.19 8.37 7.68 8.15 8.73 8.83
 8.57 9.68 8.09 8.17 7.64 8.01 7.95 8.49 7.87 7.97 8.18 8.55 8.74 8.13
 8.44 9.47 8.24 7.34 7.43 7.25 8.06 7.67 9.54 9.62 7.56 9.74 9.82 7.96
 7.45 7.94 8.35 7.42 8.95 9.86 7.23 7.79 9.25 9.67 8.86 7.57 7.21 9.27
 7.81 7.69]
=========================================================================================
===========
Unique values in Research feature are:
[1 0]
=========================================================================================
===========
Unique values in Chance of Admit  feature are:
[0.92 0.76 0.72 0.8  0.65 0.9  0.75 0.68 0.5  0.45 0.52 0.84 0.78 0.62
 0.61 0.54 0.66 0.63 0.64 0.7  0.94 0.95 0.97 0.44 0.46 0.74 0.91 0.88
 0.58 0.48 0.49 0.53 0.87 0.86 0.89 0.82 0.56 0.36 0.42 0.47 0.55 0.57
 0.96 0.93 0.38 0.34 0.79 0.71 0.69 0.59 0.85 0.77 0.81 0.83 0.67 0.73
 0.6  0.43 0.51 0.39 0.37]
=========================================================================================
===========
```

In [5]:

```
raw_data.drop_duplicates(inplace = True)
```

In [6]:

```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
```

```
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         500 non-null    int64  
 1   GRE Score          500 non-null    int64  
 2   TOEFL Score        500 non-null    int64  
 3   University Rating  500 non-null    int64  
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64  
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

In [7]:

```
raw_data.describe()
```

Out[7]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

In [8]:

```
raw_data['Serial No.'] = raw_data['Serial No.'].astype('category')
```

In [9]:

```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   Serial No.         500 non-null    category
 1   GRE Score          500 non-null    int64   
 2   TOEFL Score        500 non-null    int64   
 3   University Rating  500 non-null    int64   
 4   SOP                500 non-null    float64 
 5   LOR                500 non-null    float64 
 6   CGPA               500 non-null    float64 
 7   Research           500 non-null    int64   
 8   Chance of Admit    500 non-null    float64 
dtypes: category(1), float64(4), int64(4)
memory usage: 52.4 KB
```

In [10]:

```
raw_data.isna().sum()
```

Out[10]:

```
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
```

- **No Null values are present in the dataset.**

In [11]:

```
raw_data.columns
```

Out[11]:

```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [12]:

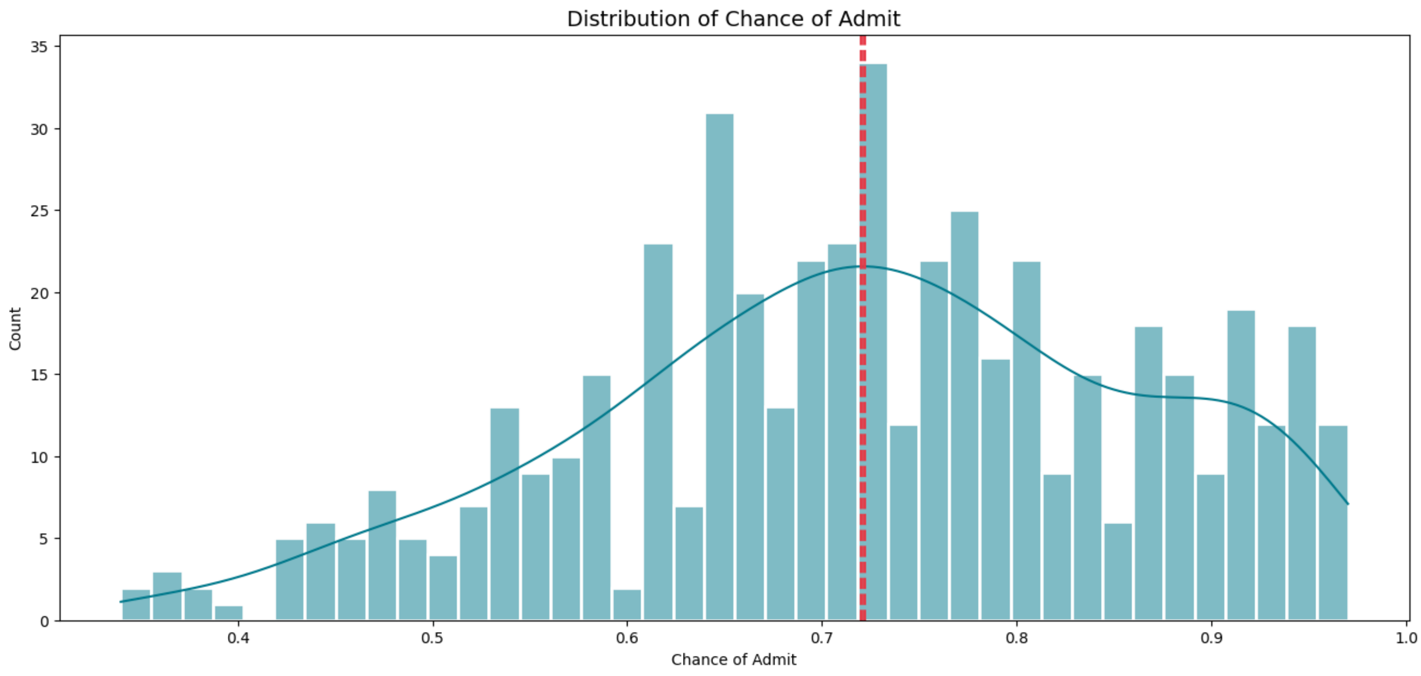
```
raw_data.columns = ['Serial_No','GRE_score','TOEFL_score','university_rating','SOP','LOR'
,'CGPA','Research','chance_of_admit']
```

Let's check for the outliers and distribution of the features

Chance of Admit Feature

In [13]:

```
plt.figure(figsize = (16,7))
sns.histplot(data = raw_data, x = 'chance_of_admit',kde = True,bins = 40,color='#00798C'
, edgecolor='white', linewidth=2)
mean_chance_of_admit = np.mean(raw_data['chance_of_admit'])
median_chance_of_admit = np.median(raw_data['chance_of_admit'])
plt.axvline(median_chance_of_admit, color='#E63946', linestyle='dashed', linewidth=2, lab
el=f'Median Chance of Admit: {median_chance_of_admit:.2f}')
plt.axvline(mean_chance_of_admit, color='#E63946', linestyle='dashed', linewidth=2, label
=f'Mean Chance of Admit: {mean_chance_of_admit:.2f}')
plt.xlabel('Chance of Admit')
plt.title('Distribution of Chance of Admit',fontsize = 14)
plt.show()
```



In [14]:

```
# Test of Normal distribution
'''
H0 : Data appears to be Normally Distributed
H1 : Data does not appear to be normally distributed
'''
from scipy.stats import shapiro
chance_of_admit_data = raw_data['chance_of_admit']
s_stat, p_val = shapiro(chance_of_admit_data)
alpha = 0.05
print(f'p value is : {p_val}')
if p_val > alpha:
  print("The data appears to be normally distributed (fail to reject the null hypothesis)
")
else:
  print("The data does not appear to be normally distributed (reject the null hypothesis)
")
```

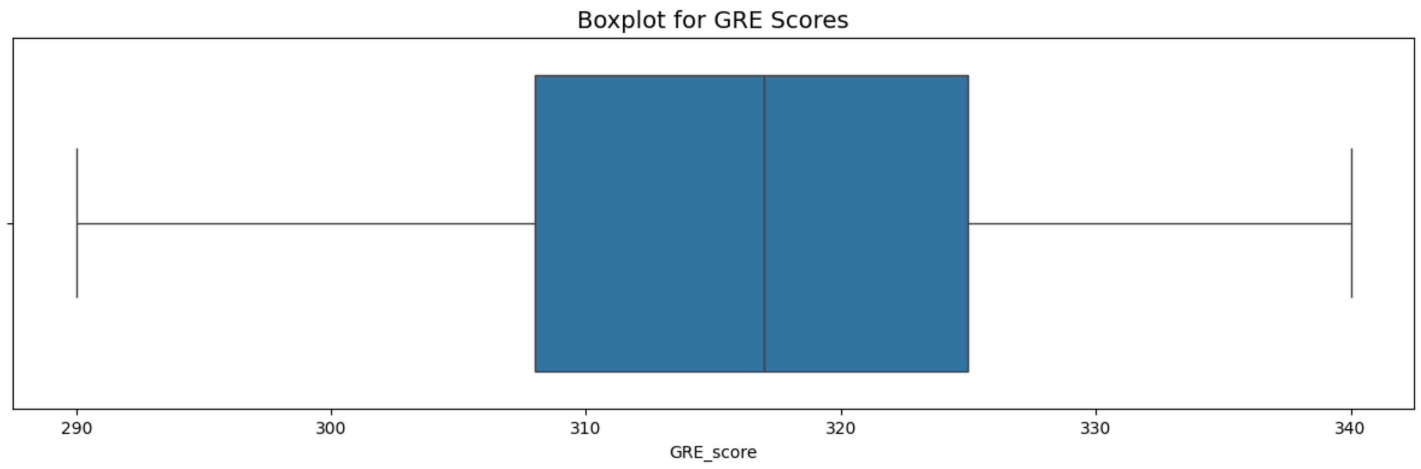
```
p value is : 2.654237050592201e-06
The data does not appear to be normally distributed (reject the null hypothesis)
```

- **Chance of Admit data is not normally distributed.**

GRE Scores

In [15]:

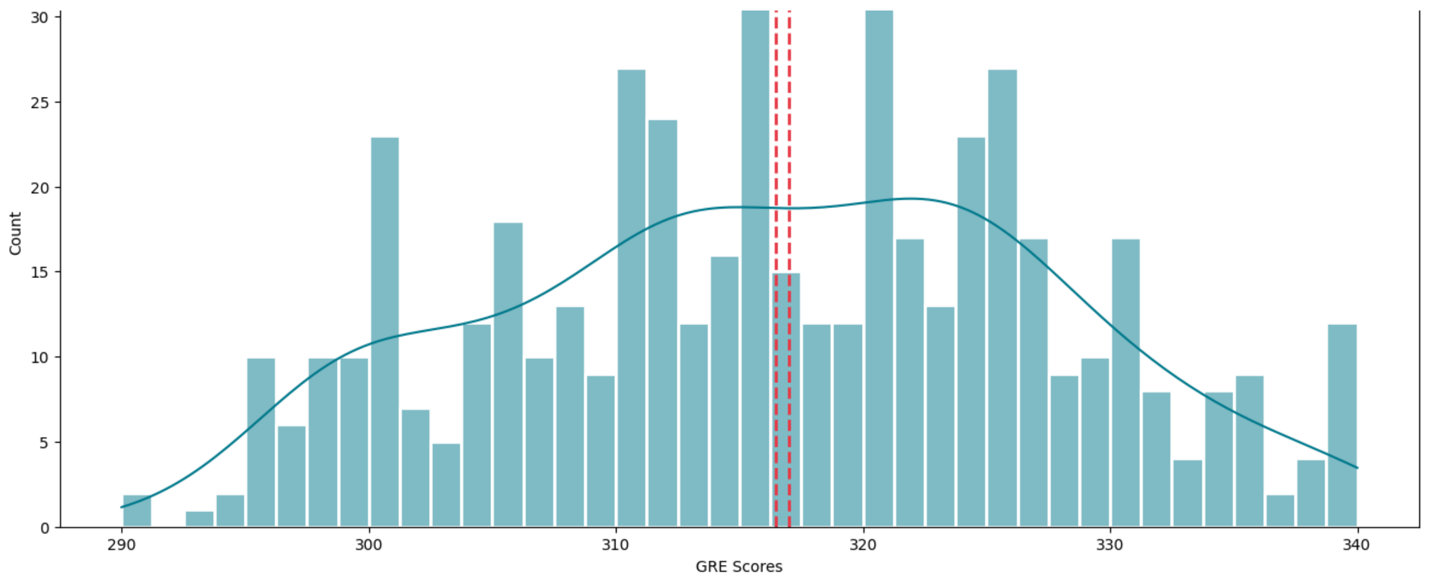
```
plt.figure(figsize = (15,4))
sns.boxplot(x = raw_data['GRE_score'])
plt.title('Boxplot for GRE Scores',fontsize = 14)
plt.show()
```



- **No outliers present in the dataset**

In [16]:

```
plt.figure(figsize = (16,7))
sns.histplot(data = raw_data, x = 'GRE_score',kde = True,bins = 40,color='#00798C', edge
color='white', linewidth=2)
mean_gre_score = np.mean(raw_data['GRE_score'])
median_gre_score = np.median(raw_data['GRE_score'])
plt.axvline(median_gre_score, color='#E63946', linestyle='dashed', linewidth=2, label=f'
Median GRE Score: {median_gre_score:.2f}')
plt.axvline(mean_gre_score, color='#E63946', linestyle='dashed', linewidth=2, label=f'Mea
n GRE Score: {mean_gre_score:.2f}')
plt.xlabel('GRE Scores')
plt.title('Distribution of GRE Scores',fontsize = 14)
plt.show()
```



- **Visually GRE scores are normally distributed**
- **Let's check it statistically by using Shapiro Wilk test**

In [17]:

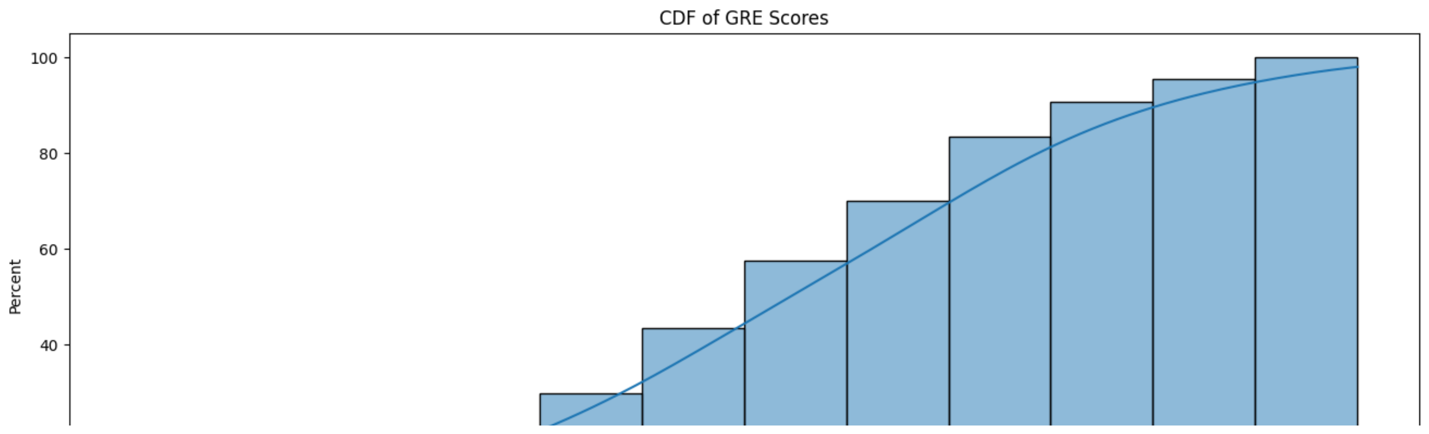
```
# Test of Normal distribution
'''
H0 : Data appears to be Normally Distributed
H1 : Data does not appear to be normally distributed
'''
from scipy.stats import shapiro
gre_score_data = raw_data['GRE_score']
s_stat, p_val = shapiro(gre_score_data)
alpha = 0.05
print(f'p value is : {p_val}')
if p_val > alpha:
  print("The data appears to be normally distributed (fail to reject the null hypothesis)
")
else:
  print("The data does not appear to be normally distributed (reject the null hypothesis)
")
```

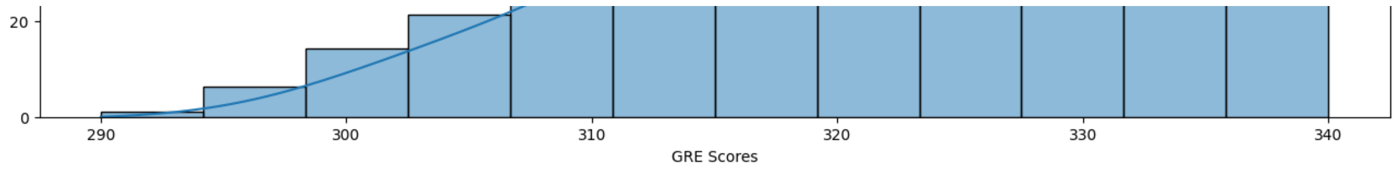
```
p value is : 8.212661487050354e-05
The data does not appear to be normally distributed (reject the null hypothesis)
```

- **Final verdict is GRE scores data is not normally distributed.**

In [18]:

```
plt.figure(figsize = (16,6))
sns.histplot(data = raw_data, x = 'GRE_score',kde = True , cumulative = True , stat = 'p
ercent')
plt.xlabel('GRE Scores')
plt.title('CDF of GRE Scores')
plt.show()
```



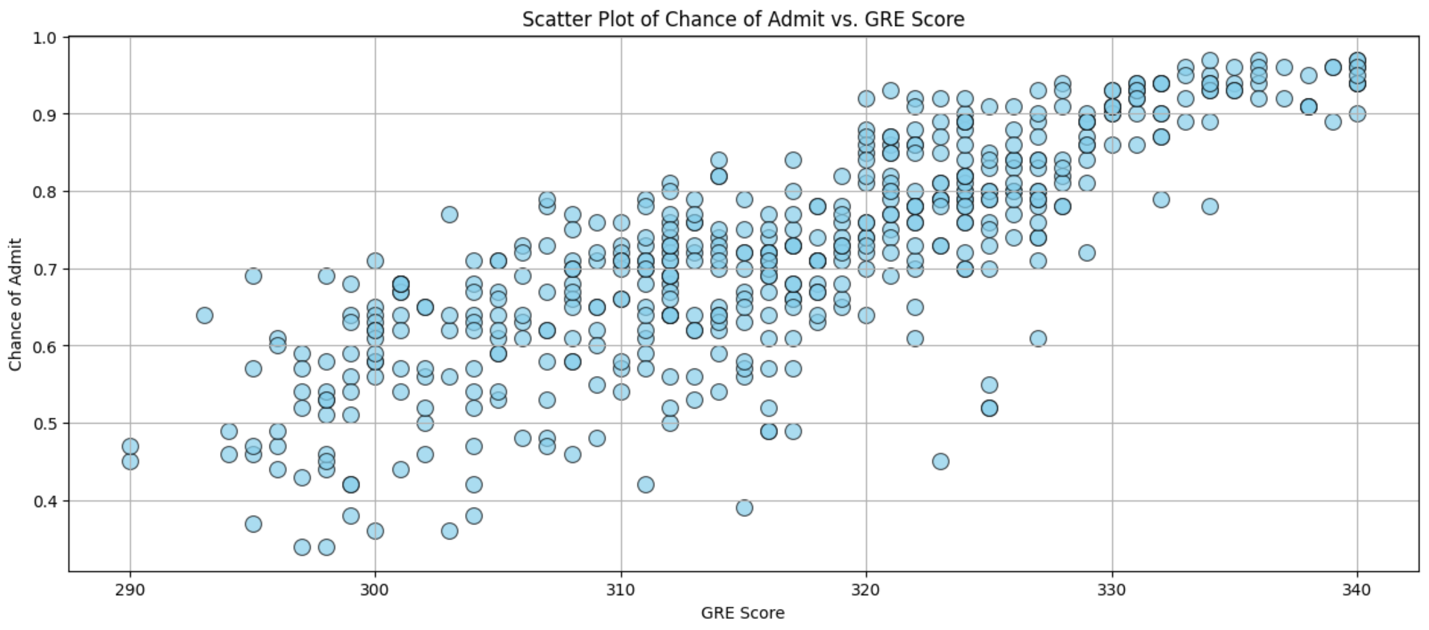


- **60% students have GRE score less than 320.**

Bivariate analysis on chance of admit and GRE scores

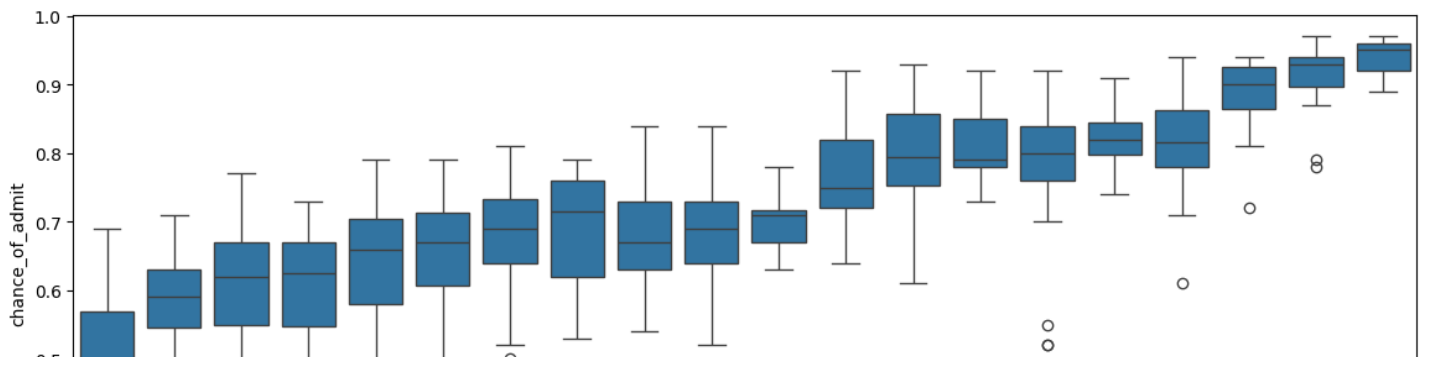
In [19]:

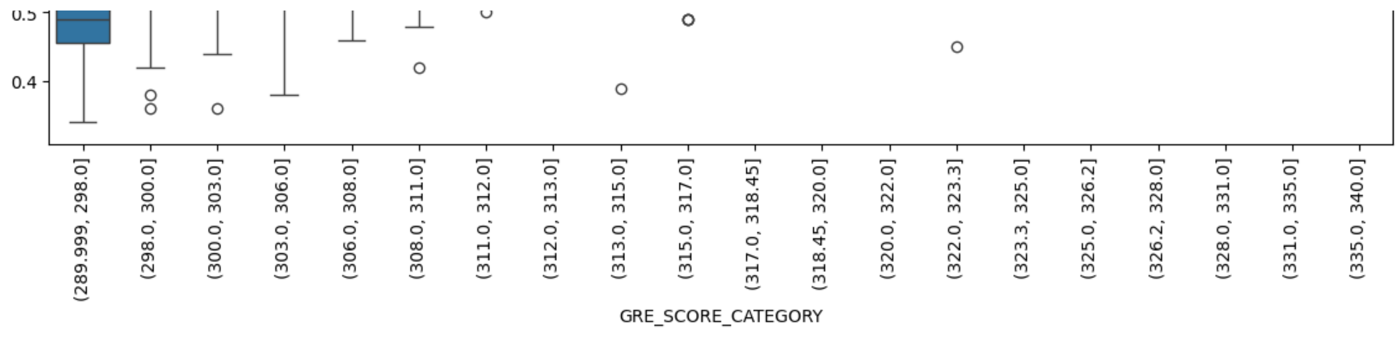
```
plt.figure(figsize = (15,6))
# Create a scatter plot
sns.scatterplot(x=raw_data['GRE_score'], y=raw_data['chance_of_admit'], s=100, color='sk
yblue', edgecolor='black', alpha=0.7)

# Add labels and title
plt.title('Scatter Plot of Chance of Admit vs. GRE Score')
plt.xlabel('GRE Score')
plt.ylabel('Chance of Admit')

# Add grid
plt.grid(True)

# Show the plot
plt.show()
```



In [20]:

```
raw_data["GRE_SCORE_CATEGORY"]=pd.qcut(raw_data["GRE_score"],20)
plt.figure(figsize=(14,5))
sns.boxplot(y = raw_data["chance_of_admit"], x = raw_data["GRE_SCORE_CATEGORY"])
plt.xticks(rotation = 90)
plt.show()
```





In [21]:

```
from scipy.stats import spearmanr,pearsonr
```

In [22]:

```
pearsonr(raw_data['GRE_score'],raw_data['chance_of_admit'])
```

Out[22]:

```
PearsonRResult(statistic=0.8103506354632598, pvalue=1.0884036862479007e-117)
```

In [23]:

```
spearmanr(raw_data['GRE_score'],raw_data['chance_of_admit'])
```

Out[23]:

```
SignificanceResult(statistic=0.8222011595365538, pvalue=5.734552105475668e-124)
```

- **From scatter plot and boxplot it can be said that GRE Socre and Chance of Admission are positively correlated.**
- **In both cases they show correlation of more than 80%.**

TOEFL Score and Chance of Admit

In [24]:

```
raw_data.columns
```

Out[24]:

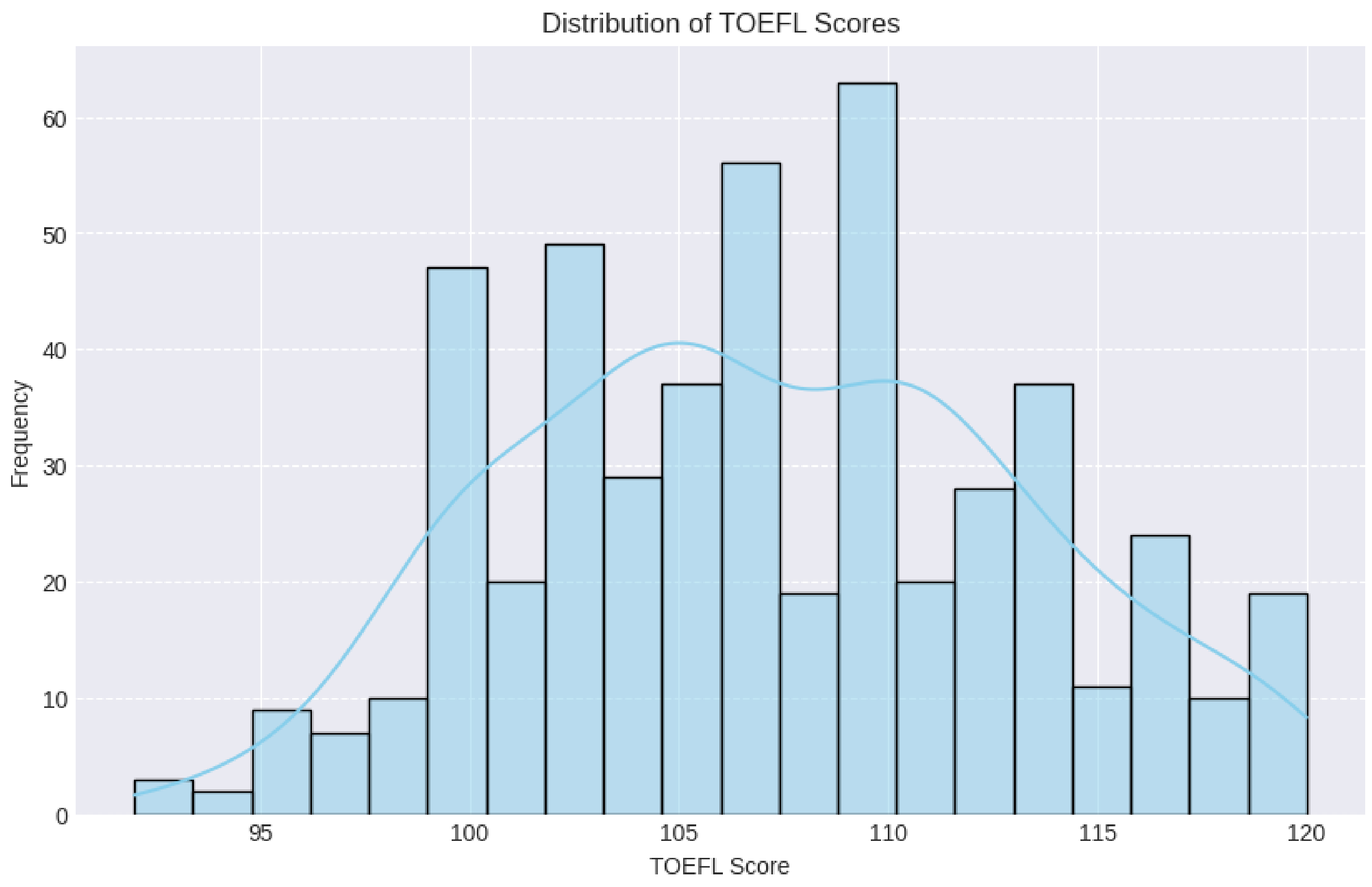
```
Index(['Serial_No', 'GRE_score', 'TOEFL_score', 'university_rating', 'SOP',
       'LOR', 'CGPA', 'Research', 'chance_of_admit', 'GRE_SCORE_CATEGORY'],
      dtype='object')
```

In [25]:

```
# Set the style for the plot
plt.style.use('seaborn-darkgrid')

# Create the histogram plot
plt.figure(figsize=(10, 6))
sns.histplot(data=raw_data, x='TOEFL_score', bins=20, kde=True, color='skyblue')

# Add labels and title
plt.xlabel('TOEFL Score')
plt.ylabel('Frequency')
plt.title('Distribution of TOEFL Scores')

# Add grid lines
plt.grid(axis='y', linestyle='--')

# Show the plot
plt.show()
```

```
<ipython-input-25-a1ee5cdf5132>:2: MatplotlibDeprecationWarning: The seaborn styles shipp
ed by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shi
```

```
pped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternati
vely, directly use the seaborn API instead.
  plt.style.use('seaborn-darkgrid')
```



Conducting Shapiro Wilk Test for TOEFL score

In [26]:

```python
toefl_scores = raw_data['TOEFL_score']

# Perform the Shapiro-Wilk test
statistic, p_value = shapiro(toefl_scores)

# Print the test results
print(f"Shapiro-Wilk Test Statistic: {statistic}")
print(f"P-value: {p_value}")

# Interpret the results
alpha = 0.05  # significance level
if p_value > alpha:
    print("The data appears to be normally distributed (fail to reject H0)")
else:
    print("The data does not appear to be normally distributed (reject H0)")
```

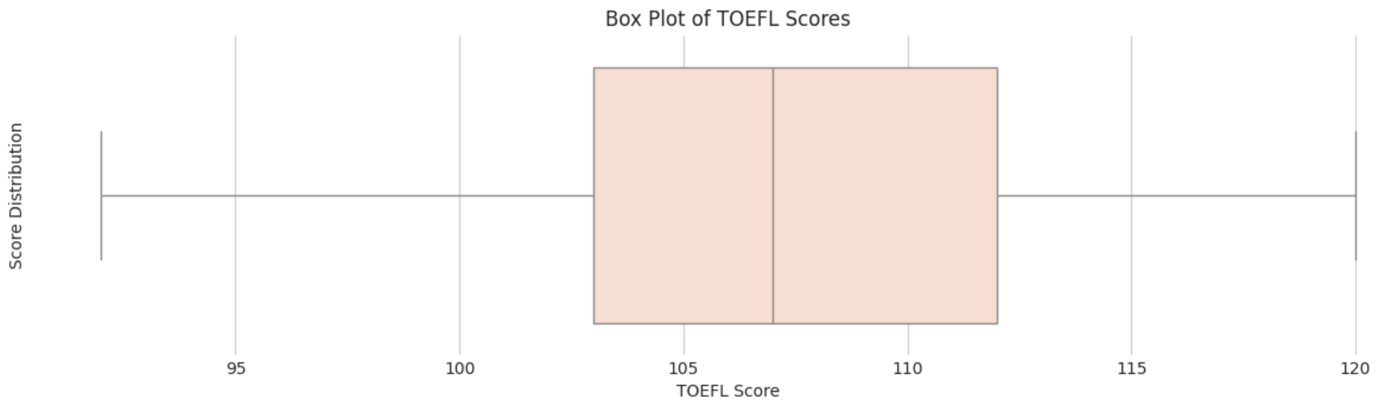
```
Shapiro-Wilk Test Statistic: 0.9858347177505493
P-value: 8.730924309929833e-05
The data does not appear to be normally distributed (reject H0)
```

- **The TOEFL scores data isn't Normally distributed.**

In [27]:

```python
sns.set_style("whitegrid")
sns.set_palette("Reds")

# Create the box plot
plt.figure(figsize=(15, 3.5))
sns.boxplot(data=raw_data, x='TOEFL_score')
```

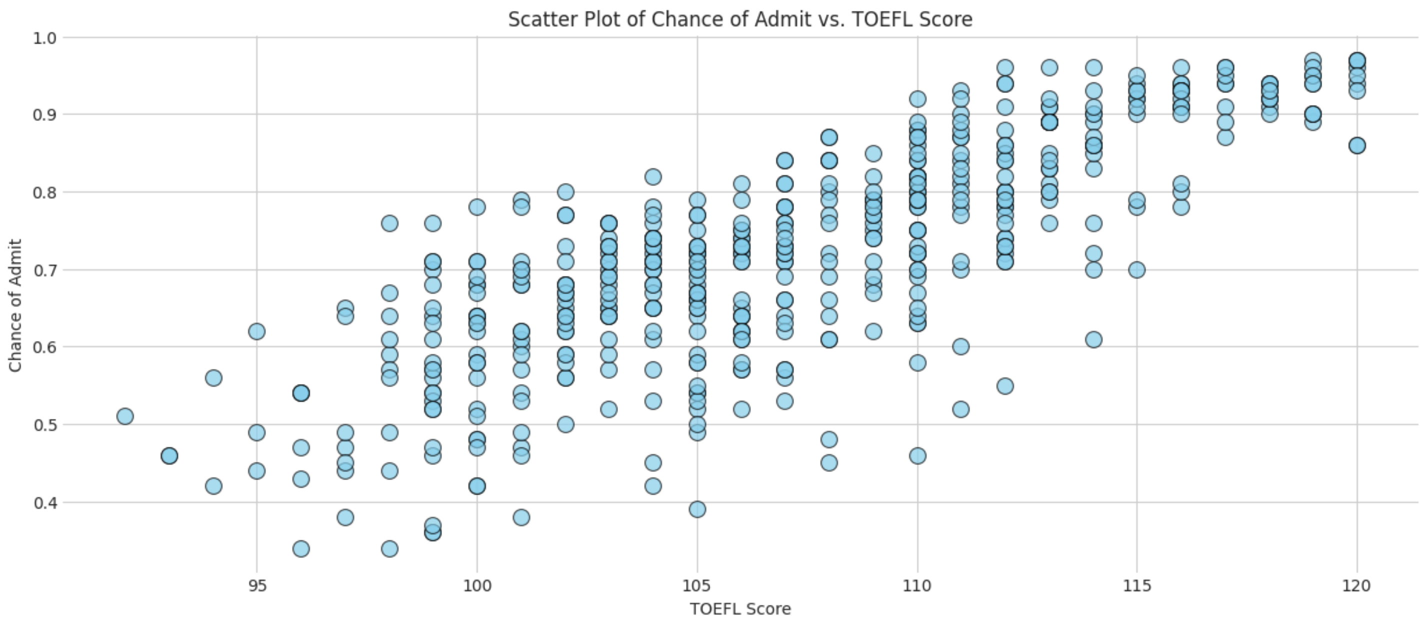
```
# Add labels and title
plt.xlabel('TOEFL Score')
plt.ylabel('Score Distribution')
plt.title('Box Plot of TOEFL Scores')

# Show the plot
plt.show()
```



- **No outliers present in the TOEFL data.**

In [28]:

```
plt.figure(figsize = (15,6))
# Create a scatter plot
sns.scatterplot(x=raw_data['TOEFL_score'], y=raw_data['chance_of_admit'], s=100, color='
skyblue', edgecolor='black', alpha=0.7)

# Add labels and title
plt.title('Scatter Plot of Chance of Admit vs. TOEFL Score')
plt.xlabel('TOEFL Score')
plt.ylabel('Chance of Admit')

# Add grid
plt.grid(True)

# Show the plot
plt.show()
```



- **From the scatterplot it is evident TOEFL score and chance of admit have low correlation.**

Checking correlation coefficients

In [29]:

```
pearsonr(raw_data['chance_of_admit'],raw_data['TOEFL_score'])
```

Out[29]:

```
PearsonRResult(statistic=0.792227614305083, pvalue=6.729926762330067e-109)
```

In [30]:

```
spearmanr(raw_data['chance_of_admit'],raw_data['TOEFL_score'])
```

Out[30]:

```
SignificanceResult(statistic=0.7936341632036854, pvalue=1.504956427966445e-109)
```

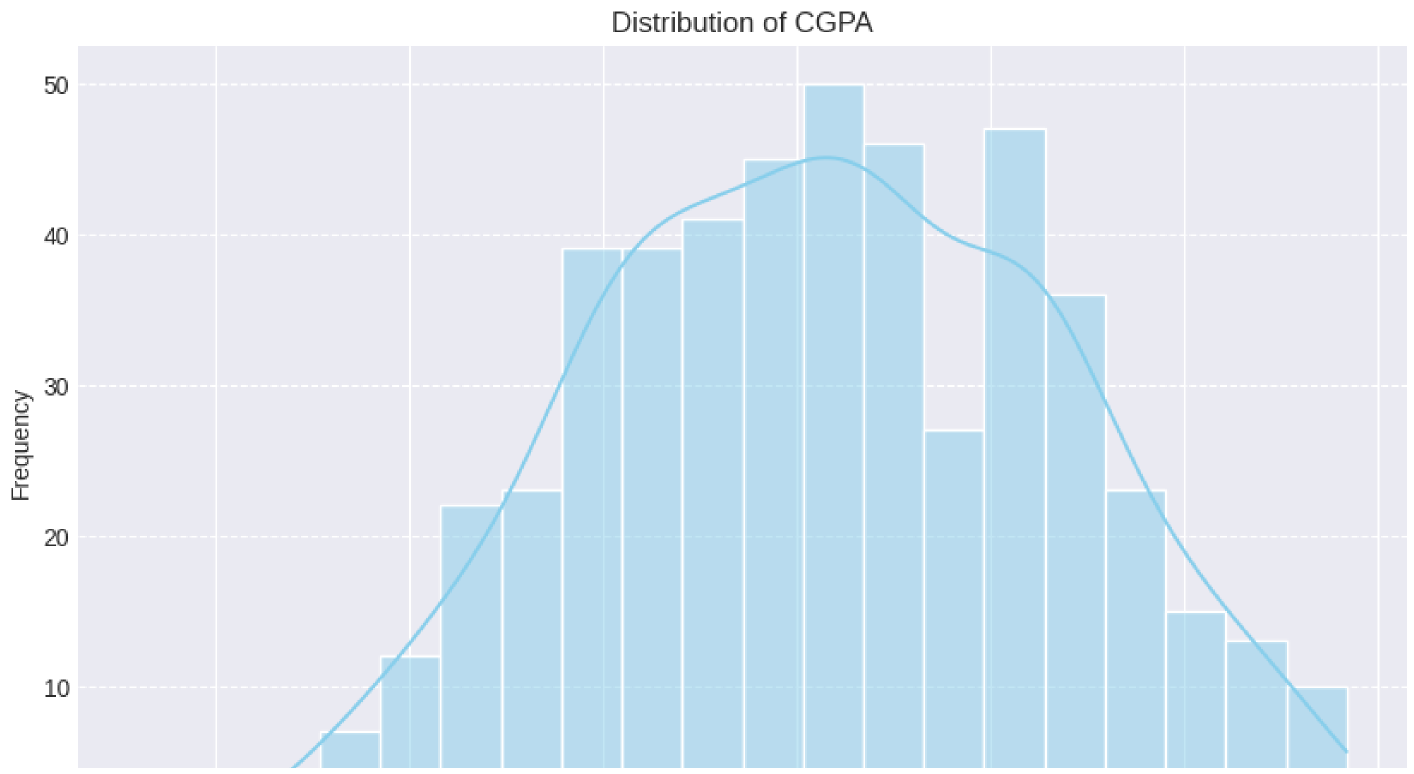
- **Chance of Admit and TOEFL score both have Strong positive correlation.**

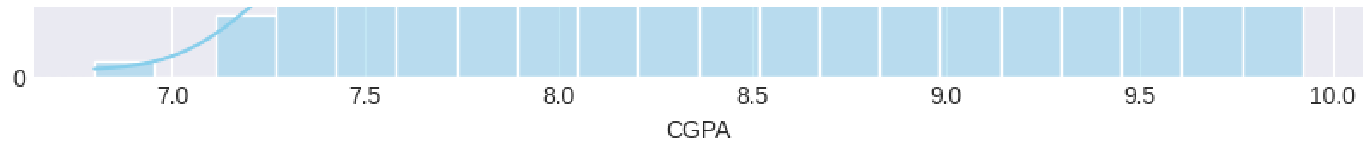
CGPA and chance of Admission

In [31]:

```
plt.style.use('seaborn-darkgrid')

# Create the histogram plot for CGPA
plt.figure(figsize=(10, 6))
sns.histplot(data=raw_data, x='CGPA', bins=20, kde=True, color='skyblue')

# Add labels and title
plt.xlabel('CGPA')
plt.ylabel('Frequency')
plt.title('Distribution of CGPA')

# Add grid lines
plt.grid(axis='y', linestyle='--')

# Show the plot
plt.show()
```

```
<ipython-input-31-b754dbf2e023>:1: MatplotlibDeprecationWarning: The seaborn styles shipp
ed by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shi
pped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternati
vely, directly use the seaborn API instead.
  plt.style.use('seaborn-darkgrid')
```





In [32]:

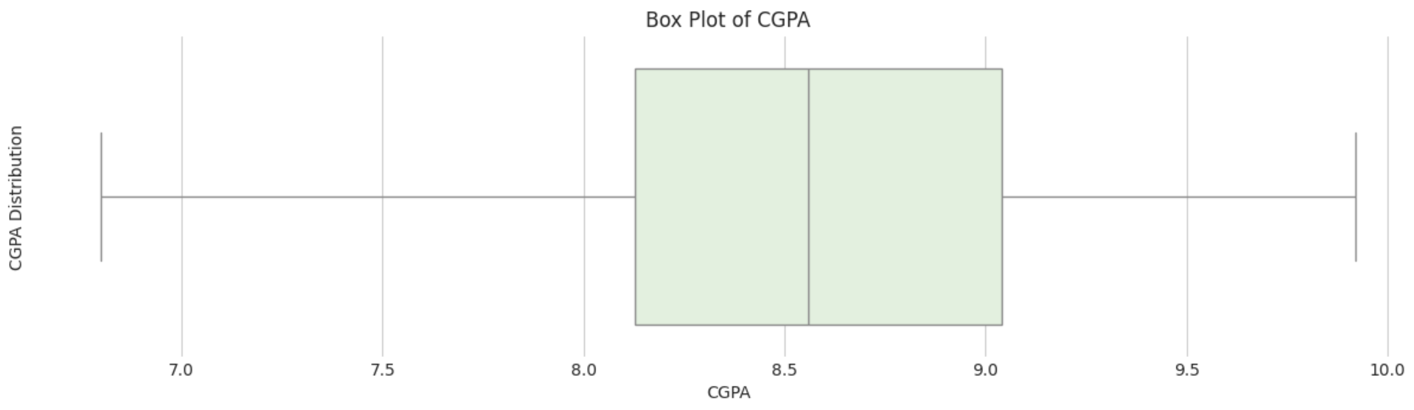
```
# Test of Normality for CGPA Using Shapiro Wilk Test
statistic, p_value = shapiro(raw_data['CGPA'])

# Print the test results
print(f"Shapiro-Wilk Test Statistic: {statistic}")
print(f"P-value: {p_value}")

# Interpret the results
alpha = 0.05  # significance level
if p_value > alpha:
    print("The data appears to be normally distributed (fail to reject H0)")
else:
    print("The data does not appear to be normally distributed (reject H0)")
```

```
Shapiro-Wilk Test Statistic: 0.9922108054161072
P-value: 0.010292120277881622
The data does not appear to be normally distributed (reject H0)
```

- **CGPA data is not Normally Distributed.**

In [33]:

```
# Set the style and color palette for box plot
sns.set_style("whitegrid")
sns.set_palette("Greens")

# Create the box plot for CGPA
plt.figure(figsize=(15, 3.5))
sns.boxplot(data=raw_data, x='CGPA')

# Add labels and title
plt.xlabel('CGPA')
plt.ylabel('CGPA Distribution')
plt.title('Box Plot of CGPA')

# Show the plot
plt.show()
```



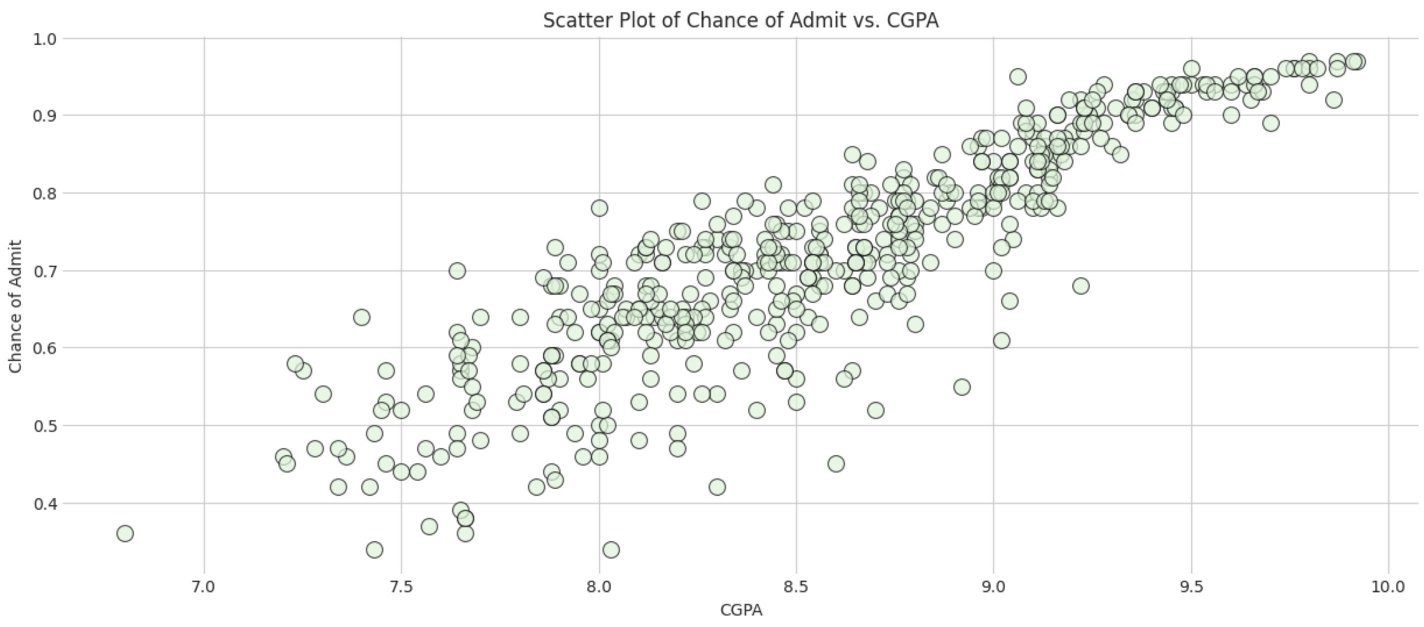
- **No outliers present in the CGPA data.**

In [34]:

```
plt.figure(figsize=(15, 6))
sns.scatterplot(x=raw_data['CGPA'], y=raw_data['chance_of_admit'], s=100, palette='virid
is', edgecolor='black', alpha=0.7)

# Add labels and title
```

```
plt.title('Scatter Plot of Chance of Admit vs. CGPA')
plt.xlabel('CGPA')
plt.ylabel('Chance of Admit')

# Add grid
plt.grid(True)

# Show the plot
plt.show()
```

```
<ipython-input-34-4f580f8b526a>:2: UserWarning: Ignoring `palette` because no `hue` varia
ble has been assigned.
  sns.scatterplot(x=raw_data['CGPA'], y=raw_data['chance_of_admit'], s=100, palette='viri
dis', edgecolor='black', alpha=0.7)
```



In [35]:

```
pearsonr(raw_data['chance_of_admit'],raw_data['CGPA'])
```

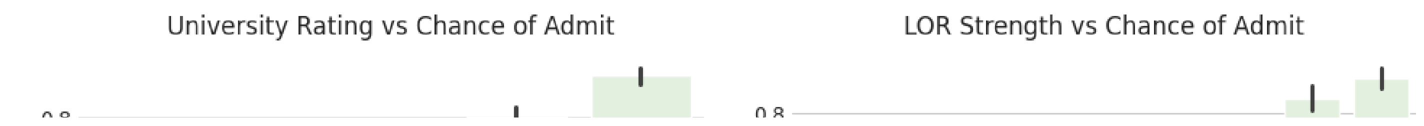
Out[35]:

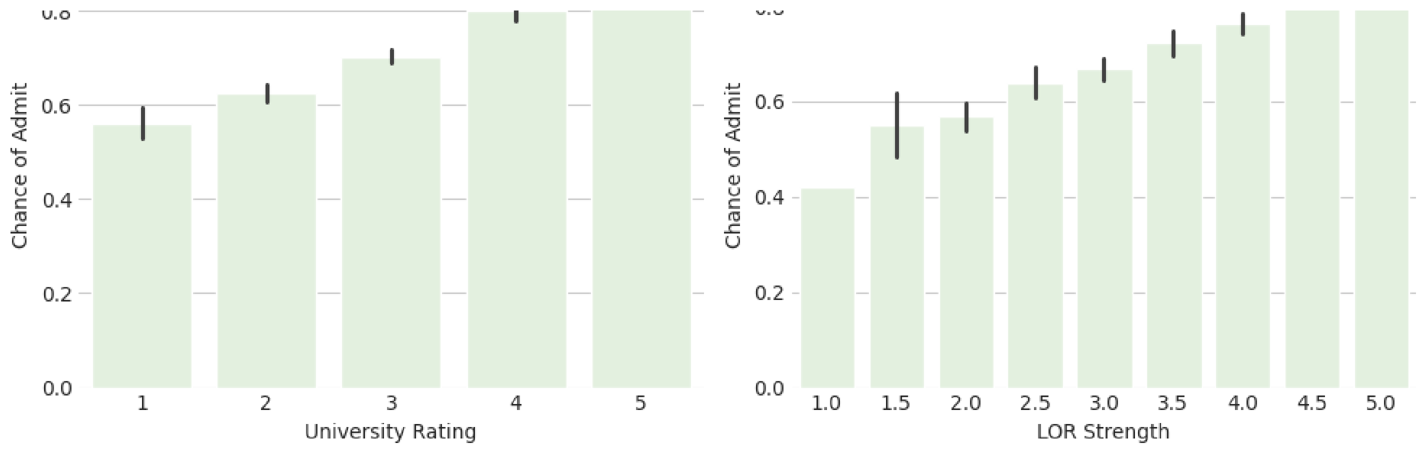
```
PearsonRResult(statistic=0.8824125749045744, pvalue=3.3965448587112374e-165)
```

- **CGPA data and Chance of Admit have Strong Positive Correlation.**

In [36]:

```
plt.show()

fig, axs = plt.subplots(1, 2, figsize=(10, 4))

sns.barplot(data=raw_data,x='university_rating',y='chance_of_admit',ax=axs[0])
axs[0].set_title('University Rating vs Chance of Admit')
axs[0].set_xlabel('University Rating')
axs[0].set_ylabel('Chance of Admit')

sns.barplot(data=raw_data,x='LOR',y='chance_of_admit',ax=axs[1])
axs[1].set_title('LOR Strength vs Chance of Admit')
axs[1].set_xlabel('LOR Strength')
axs[1].set_ylabel('Chance of Admit')


plt.tight_layout()
plt.show()
```





- **It can be observed that people with higher university rating and higher LOR rating have got high chance of admit.**

Research vs Chance of Admit

In [37]:

```
raw_data.columns
```

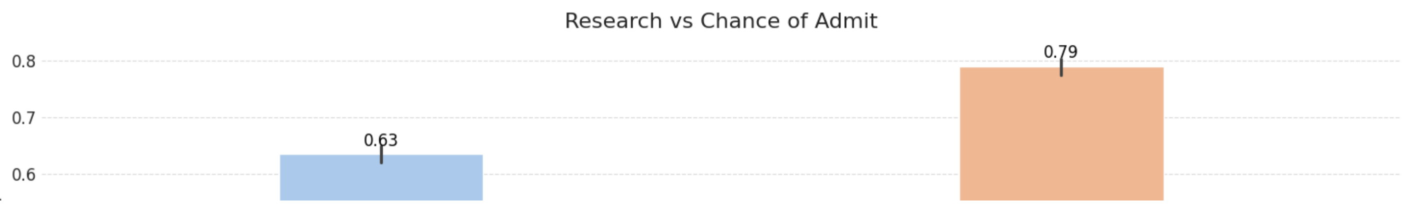
Out[37]:

```
Index(['Serial_No', 'GRE_score', 'TOEFL_score', 'university_rating', 'SOP',
       'LOR', 'CGPA', 'Research', 'chance_of_admit', 'GRE_SCORE_CATEGORY'],
      dtype='object')
```

In [38]:

```
fig, ax = plt.subplots(figsize=(15, 6))

sns.barplot(data=raw_data, x='Research', y='chance_of_admit', palette='pastel', ax=ax,wi
dth = 0.3)
ax.set_title('Research vs Chance of Admit', fontsize=16)
ax.set_xlabel('Research', fontsize=14)
ax.set_ylabel('Chance of Admit', fontsize=14)
ax.tick_params(axis='both', which='major', labelsize=12)

# Adding grid lines
ax.grid(axis='y', linestyle='--', alpha=0.7)

# Adding annotations (optional)
for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}', (p.get_x() + p.get_width() / 2., p.get_height()
),
                ha='center', va='center', fontsize=12, color='black', xytext=(0, 10),
                textcoords='offset points')

# Show plot
plt.tight_layout()
plt.show()
```

```
<ipython-input-38-d92fd771b484>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. A
ssign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=raw_data, x='Research', y='chance_of_admit', palette='pastel', ax=ax,w
idth = 0.3)
```



Chance of Admit
0.5
0.4
0.3
0.2
0.1
0.0
0
1
Research

Creating New Feature

In [39]:

```
raw_data.columns
```

Out[39]:

```
Index(['Serial_No', 'GRE_score', 'TOEFL_score', 'university_rating', 'SOP',
       'LOR', 'CGPA', 'Research', 'chance_of_admit', 'GRE_SCORE_CATEGORY'],
      dtype='object')
```

In [40]:

```
raw_data['TotalScore']= (raw_data['GRE_score']*10/340)+(raw_data['TOEFL_score']*10/120)
raw_data['SOP_LOR'] =raw_data['SOP']+raw_data['LOR']
```

In [41]:

```
# dropping the Unique counting variable
raw_data.drop(['Serial_No'],axis = 1,inplace = True)
```

In [42]:

```
raw_data
```

Out[42]:

	GRE_score	TOEFL_score	university_rating	SOP	LOR	CGPA	Research	chance_of_admit	GRE_SCORE_CATEGORY	Tot
0	337	118	4	4.5	4.5	9.65	1	0.92	(335.0, 340.0]	19
1	324	107	4	4.0	4.5	8.87	1	0.76	(323.3, 325.0]	18
2	316	104	3	3.0	3.5	8.00	1	0.72	(315.0, 317.0]	17
3	322	110	3	3.5	2.5	8.67	1	0.80	(320.0, 322.0]	18
4	314	103	2	2.0	3.0	8.21	0	0.65	(313.0, 315.0]	17
...	...	...	...	...	...	...	...	...	...	
495	332	108	5	4.5	4.0	9.02	1	0.87	(331.0, 335.0]	18
496	337	117	5	5.0	5.0	9.87	1	0.96	(335.0, 340.0]	19
497	330	120	5	4.5	5.0	9.56	1	0.93	(328.0, 331.0]	19
498	312	103	4	4.0	5.0	8.43	0	0.73	(311.0, 312.0]	17
499	327	113	4	4.5	4.5	9.04	0	0.84	(326.2, 328.0]	19

500 rows × 11 columns

Seperating Features and Target

In [43]:

```
X = raw_data.drop(columns = ['chance_of_admit','GRE_score','GRE_SCORE_CATEGORY','TOEFL_s
X = raw_data.drop(columns = ['chance_of_admit','GRE_score','GRE_SCORE_CATEGORY','TOEFL_s
core','SOP','LOR'])
y = raw_data['chance_of_admit']
```

Train Test Split

In [44]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=1)
```

Scaling the Data

In [45]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# Train data split
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)

# Test data split
X_test = pd.DataFrame(scaler.fit_transform(X_test), columns=X_test.columns)
```

Linear Regression

Checking the Assumptions

1. **Assumption : Linearity Check**

In [46]:

```
'''
Ho : There is no linear correlation between the two variables
Ha: There is linear correlation between the two variables
'''
from scipy.stats import pearsonr
alpha=0.05
cols = X_train.columns
for col in cols:
    print('*'*50)
    pstat,pvalue=pearsonr(X_train[col],y_train)
    print('Column Name:',col)
    if pvalue < alpha:
        print('P value:',pvalue)
        print('*'*50)
        print('Reject Null Hypothesis and we conclude that there\nis linear correlation
between the independent and\ndependent variable (Target)')
    else:
        print('Failed to Reject Null Hypothesis')
    print('*'*50)
print('*'*50)
```

```
**************************************************
Column Name: university_rating
P value: 8.472734355963263e-56
**************************************************
Reject Null Hypothesis and we conclude that there
is linear correlation between the independent and
dependent variable (Target)
**************************************************
**************************************************
Column Name: CGPA
P value: 2.291029286206592e-112
```

```
P value: [illegible]
**************************************************
Reject Null Hypothesis and we conclude that there
is linear correlation between the independent and
dependent variable (Target)
**************************************************
**************************************************
Column Name: Research
P value: 2.6471704101314467e-31
**************************************************
Reject Null Hypothesis and we conclude that there
is linear correlation between the independent and
dependent variable (Target)
**************************************************
**************************************************
Column Name: TotalScore
P value: 1.8528154090274337e-95
**************************************************
Reject Null Hypothesis and we conclude that there
is linear correlation between the independent and
dependent variable (Target)
**************************************************
**************************************************
Column Name: SOP_LOR
P value: 4.846256573430935e-53
**************************************************
Reject Null Hypothesis and we conclude that there
is linear correlation between the independent and
dependent variable (Target)
**************************************************
**************************************************
```

- **All the independent features have Linear Relationship with the Target variable.**

Assumption 2 : Multicolinearity

In [47]:

```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor


vif_data = pd.DataFrame()
vif_data["feature"] = X_train.columns


vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.s
hape[1])]


print('VIF Score',vif_data)
print('*'*50)
```

```
VIF Score              feature       VIF
0  university_rating  2.908534
1               CGPA  4.554536
2           Research  1.460118
3         TotalScore  3.921523
4            SOP_LOR  2.818255
**************************************************
```

- **VIF Score for all the Independent features is below 5,so we will have a good model**

In [48]:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

```
# Training the model
model.fit(X_train,y_train)
```

Out[48]:

```
▾ LinearRegression
LinearRegression()
```

In [49]:

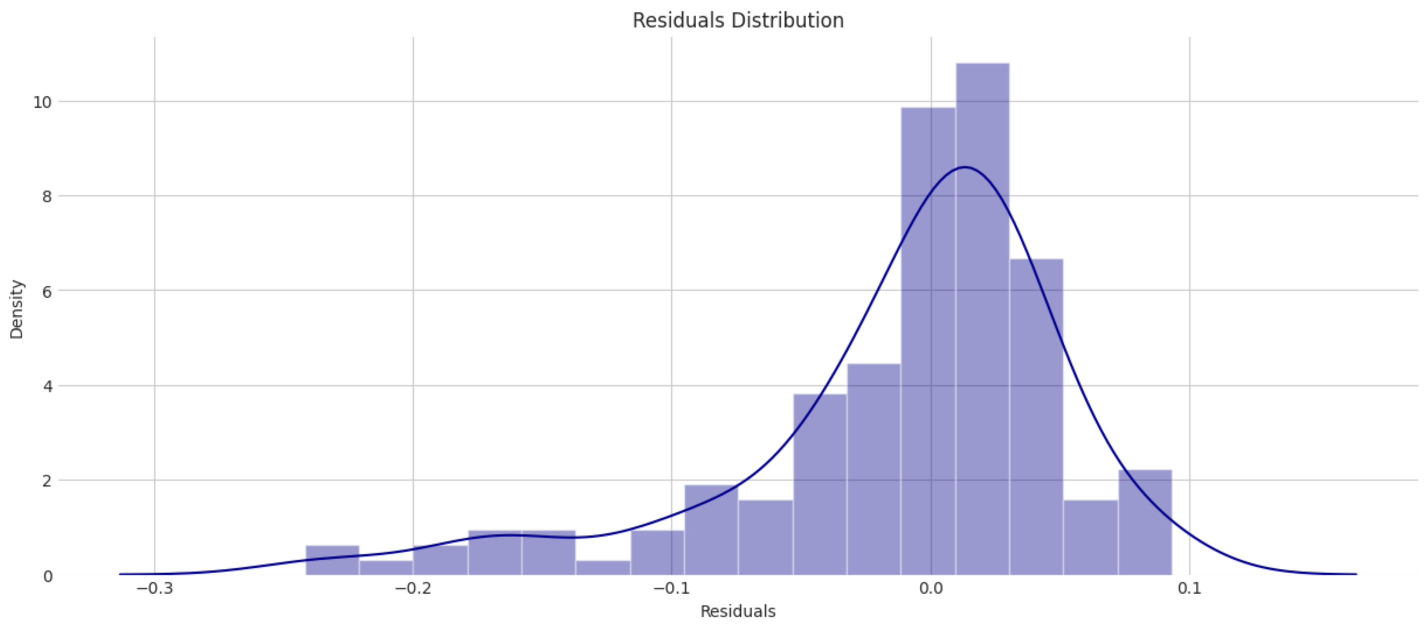
```
predictions_train = model.predict(X_train)
predictions_test = model.predict(X_test)
```

Assumption 3: Normality of Residuals

In [50]:

```
plt.figure(figsize = (15,6))
# Calculate residuals using predictions_test
residuals = y_test - predictions_test

# Plotting residuals distribution
sns.distplot(residuals,color = 'darkblue')
plt.title('Residuals Distribution')
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```

```
<ipython-input-50-0ba1a1199718>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(residuals,color = 'darkblue')
```



In [51]:

```
'''
H0 : Residuals are Normally Distributed
H1 : Residuals are not Normally Distributed
'''
# Shapiro-Wilk test
```

```
stat, p_value = shapiro(residuals)
print(f"Shapiro-Wilk Test Statistic: {stat}, p-value: {p_value}")
```

```
Shapiro-Wilk Test Statistic: 0.8696649074554443, p-value: 3.543590976207156e-10
```

p-value is less than alpha which infers that the Residuals are not Normally distributed.

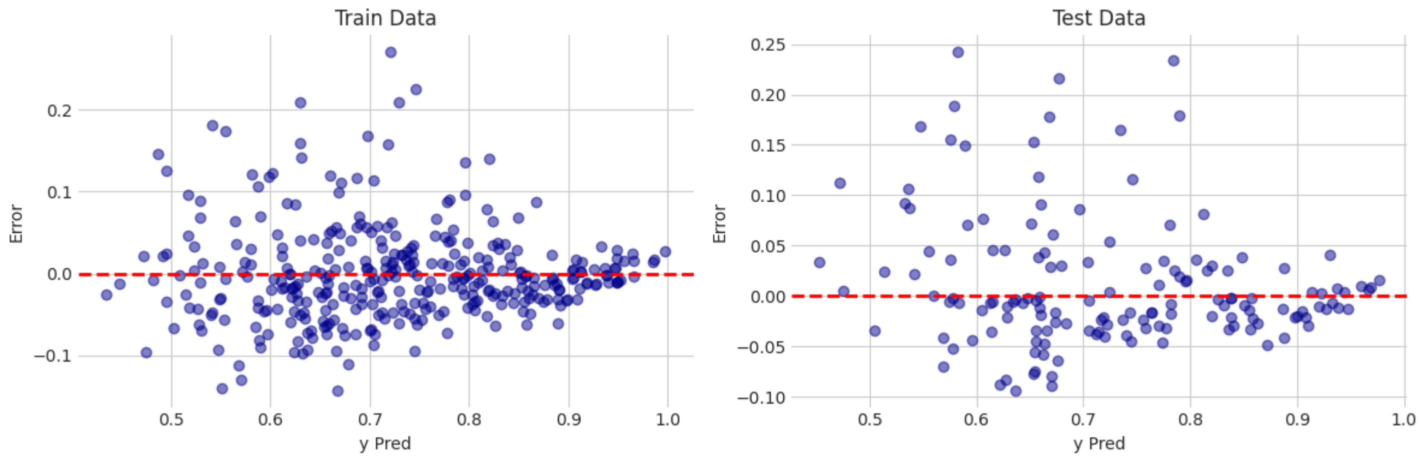
Assumption 4 : No Homoscedasticity

In [52]:

```
residuals_train = predictions_train - y_train
residuals_test = predictions_test - y_test
```

In [53]:

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))

# Plotting the first graph
axes[0].scatter(predictions_train, residuals_train, alpha=0.5,color = 'darkblue')
axes[0].axhline(0, color='red', linestyle='--', linewidth=2)
axes[0].set_title('Train Data')
axes[0].set_xlabel('y Pred')
axes[0].set_ylabel('Error')
axes[0].grid(True)

# Plotting the second graph
axes[1].scatter(predictions_test, residuals_test, alpha=0.5,color = 'darkblue')
axes[1].axhline(0, color='red', linestyle='--', linewidth=2)
axes[1].set_title('Test Data')
axes[1].set_xlabel('y Pred')
axes[1].set_ylabel('Error')
axes[1].grid(True)

# Adjust layout
plt.tight_layout()
plt.show()
```



Calculating Metrices

In [54]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [55]:

```
print("::---------Test Data---------::")
mae = mean_absolute_error(y_test, predictions_test)
mse = mean_squared_error(y_test, predictions_test)
r2_test = r2_score(y_test, predictions_test)

print('Mean Absolute Error:',mae)
print('Mean Squared Error:',mse)
print('R2 Score:',r2_test)
print('*'*50)
```

```
::---------Test Data---------::
Mean Absolute Error: 0.044355863120596
Mean Squared Error: 0.004295629681714218
R2 Score: 0.8081899059053868
**************************************************
```

In [56]:

```
def AdjustedR2score(R2,n,d):
    return 1-(((1-R2)*(n-1))/(n-d-1))
```

In [57]:

```
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,predictions_test),len(X),X.s
hape[1]))
```

```
Adjusted R2 score : 0.8062485081918785
```

- **Model Accuracy is 80.6%.**

In [57]:

```

```

2. Polynomial Regression

Train and Test Split

In [58]:

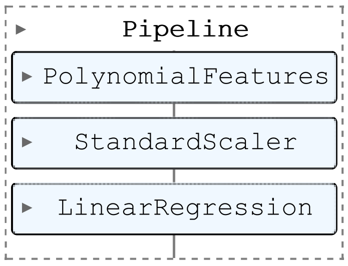
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=42)
```

Training the model

In [59]:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

degree = 2
model = make_pipeline(PolynomialFeatures(degree),StandardScaler(),LinearRegression())

model.fit(X_train,y_train)
```

Out[59]:



In [60]:

```
# Predicting the Values

predictions_train = model.predict(X_train)
```

```
predictions_test = model.predict(X_test)
```

In [61]:

```
print("::---------Test Data---------::")
mae = mean_absolute_error(y_test,predictions_test)
mse = mean_absolute_error(y_test,predictions_test)
r2_test = mean_absolute_error(y_test,predictions_test)

print('Mean Absolute Error:',mae)
print('Mean Squared Error:',mse)
print('R2 Score:',r2_test)
print('*'*50)
```

```
::---------Test Data---------::
Mean Absolute Error: 0.04202726790405688
Mean Squared Error: 0.04202726790405688
R2 Score: 0.04202726790405688
**************************************************
```

- **Polynomial Regression of Degree 2 is not useful in this case as it is giving accuracy of only 4%**

Regularisation

L2 Regularization

In [62]:

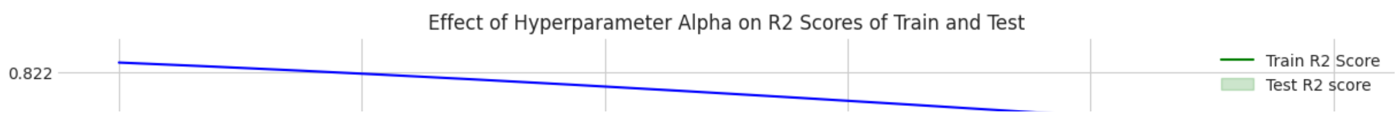
```
from sklearn.linear_model import Ridge   # L2 regualrization
from sklearn.linear_model import Lasso   # L1 regualrization
```

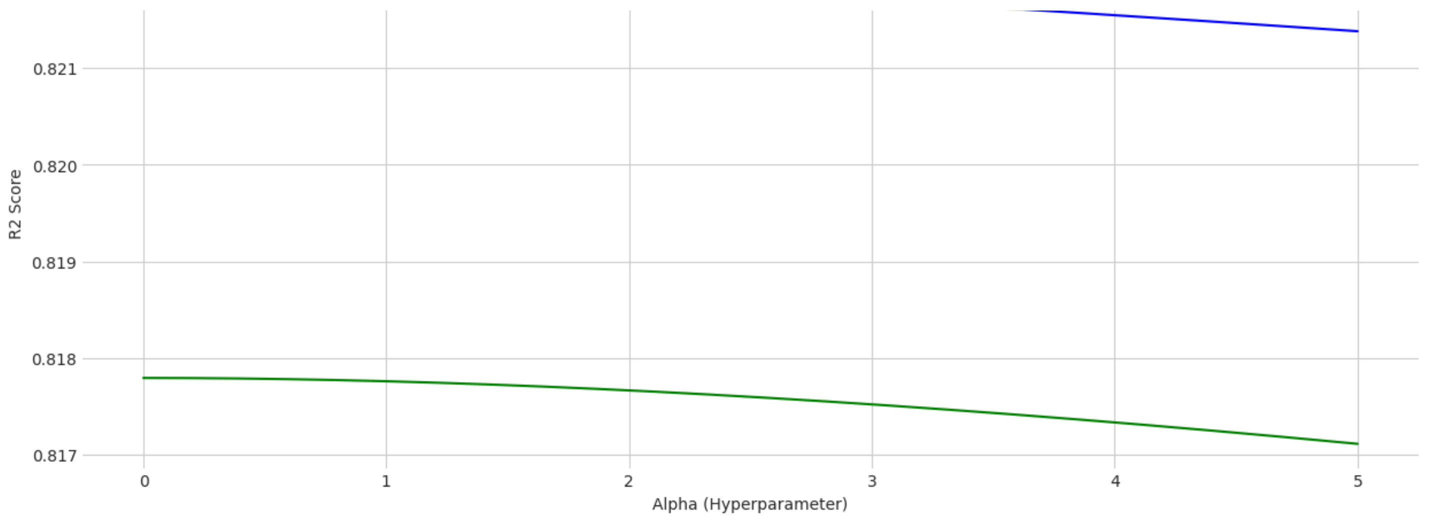
In [63]:

```
## Hyperparameter Tuning : for appropriate lambda value :

train_R2_score = []
test_R2_score = []
lambdas = []
train_test_difference_Of_R2 =  []
lambda_ = 0
while lambda_ <= 5:
    lambdas.append(lambda_)
    RidgeModel = Ridge(lambda_)
    RidgeModel.fit(X_train,y_train)
    trainR2 = RidgeModel.score(X_train,y_train)
    testR2 = RidgeModel.score(X_test,y_test)
    train_R2_score.append(trainR2)
    test_R2_score.append(testR2)

    lambda_ += 0.01
```

In [64]:

```
plt.figure(figsize=(15, 6))
sns.lineplot(x=lambdas, y=train_R2_score, color='green')  # Using a specific shade of gr
een
sns.lineplot(x=lambdas, y=test_R2_score, color='blue')    # Using a specific shade of bl
ue
plt.legend(['Train R2 Score', 'Test R2 score'])
plt.title("Effect of Hyperparameter Alpha on R2 Scores of Train and Test")
plt.xlabel("Alpha (Hyperparameter)")
plt.ylabel("R2 Score")
plt.show()
```





In [65]:

```
RidgeModel = Ridge(alpha = 0.1)
RidgeModel.fit(X_train,y_train)
trainR2 = RidgeModel.score(X_train,y_train)
testR2 = RidgeModel.score(X_test,y_test)
```

In [66]:

```
trainR2,testR2
```

Out[66]:

```
(0.8177926840425397, 0.822088925726489)
```

In [67]:

```
RidgeModel.coef_
```

Out[67]:

```
array([0.00153708, 0.11619005, 0.02729931, 0.0493736 , 0.00971166])
```

In [68]:

```
y_pred = RidgeModel.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred))  # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_test,y_pred)  ) # MAE
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))
# adjusted R2 score
```

```
MSE: 0.003638281468893302
RMSE: 0.060318168646712925
MAE : 0.04248517639513057
r2_score: 0.822088925726489
Adjusted R2 score : 0.820288206351251
```

- **Model Accuracy was 80.6% earlier but now it has increased to 82% after applying Ridge Regularization.**

Lasso Regularization

In [69]:

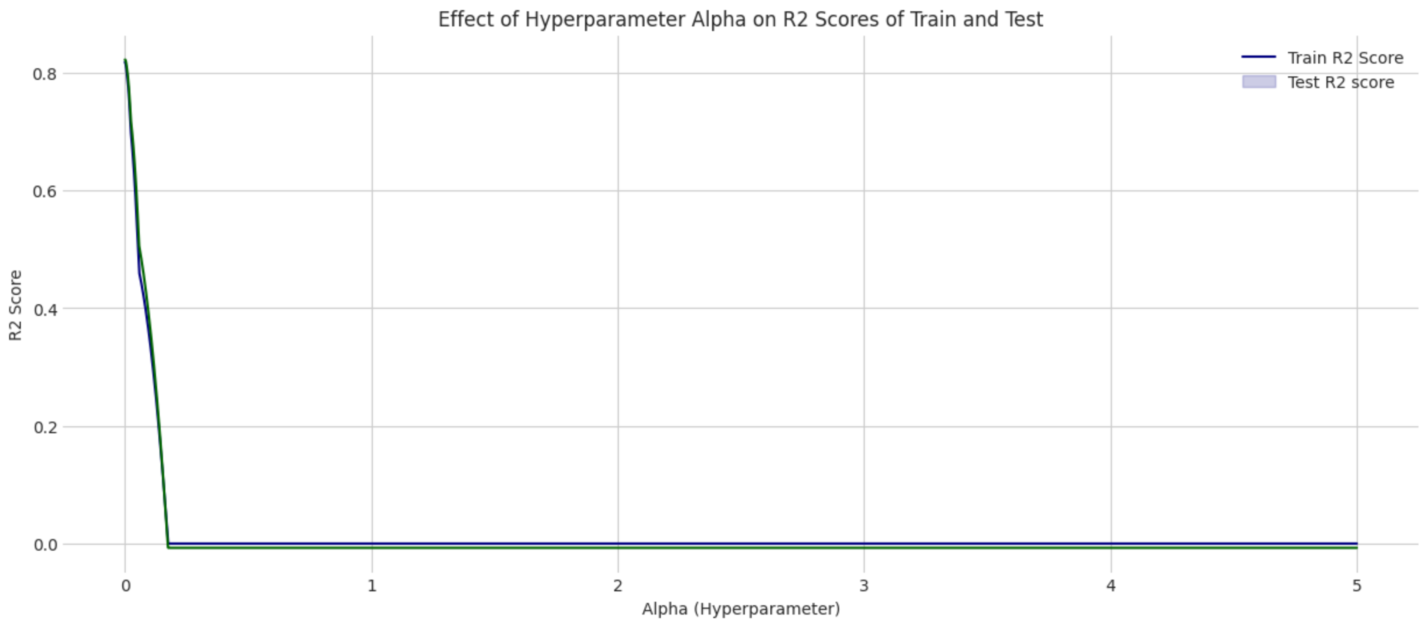
```
## Hyperparameter Tuning : for appropriate lambda value :

train_R2_score = []
test_R2_score = []
lambdas = []
```

```
train_test_difference_Of_R2 =  []
lambda_  = 0
while lambda_  <= 5:
    lambdas.append(lambda_)
    LassoModel = Lasso(alpha=lambda_)
    LassoModel.fit(X_train , y_train)
    trainR2 = LassoModel.score(X_train,y_train)
    testR2 = LassoModel.score(X_test,y_test)
    train_R2_score.append(trainR2)
    test_R2_score.append(testR2)

    lambda_ += 0.001
```

```
<ipython-input-69-e7ae1d118a4e>:11: UserWarning: With alpha=0, this algorithm does not co
nverge well. You are advised to use the LinearRegression estimator
  LassoModel.fit(X_train , y_train)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_descent.py:631:
UserWarning: Coordinate descent with no regularization may lead to unexpected results and
is discouraged.
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_descent.py:631:
ConvergenceWarning: Objective did not converge. You might want to increase the number of
iterations, check the scale of the features or consider increasing regularisation. Dualit
y gap: 7.182e-01, tolerance: 7.884e-04 Linear regression models with null weight for the
l1 regularization term are more efficiently fitted using one of the solvers implemented i
n sklearn.linear_model.Ridge/RidgeCV instead.
  model = cd_fast.enet_coordinate_descent(
```

In [70]:

```
plt.figure(figsize=(15, 6))
sns.lineplot(x = lambdas, y = train_R2_score, color='navy')   # Using a dark shade of blu
e
sns.lineplot(x = lambdas, y = test_R2_score, color='darkgreen')   # Using a dark shade of
green
plt.legend(['Train R2 Score', 'Test R2 score'])
plt.title("Effect of Hyperparameter Alpha on R2 Scores of Train and Test")
plt.xlabel("Alpha (Hyperparameter)")
plt.ylabel("R2 Score")
plt.show()
```



In [71]:

```
LassoModel = Lasso(alpha=0.001)
LassoModel.fit(X_train , y_train)
trainR2 = LassoModel.score(X_train,y_train)
testR2 = LassoModel.score(X_test,y_test)
```

In [72]:

```
trainR2,testR2
```

Out[72]:

```
(0.8174848383717302, 0.8222633383023912)
```

In [73]:

```
y_pred = LassoModel.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_test,y_pred) ) # MAE
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))
# adjusted R2 score
```

```
MSE: 0.0036347147317161003
RMSE: 0.06028859537023649
MAE : 0.04239017050992718
r2_score: 0.8222633383023912
Adjusted R2 score : 0.8204643842366259
```

- **Model Accuracy was 80.6% earlier but now it has increased to 82% after applying Lasso Regularization.**

Recommendations

1. **The model highlights that CGPA is the most significant predictor for admission chances to Ivy League colleges, followed by Total Score (GRE + TOEFL), indicating the weight of academic performance in the prediction process.**
2. **Applying Ridge and Lasso Regularization techniques has improved the model's accuracy, with Ridge Regularization achieving the highest accuracy of 82%.**
3. **Emphasize the importance of maintaining a high CGPA for aspiring Ivy League applicants, as it heavily influences their chance of admission.**
4. **Encourage applicants to focus not only on academics but also on aspects like Statement of Purpose (SOP), Letters of Recommendation (LOR), and Research Experience, which contribute significantly to the evaluation process.**
5. **Use the model predictions to strategically enhance areas that contribute most to admission chances, such as improving CGPA or strengthening SOP and LOR.**
6. **Provide valuable insights to admissions committees regarding the relative importance of different factors in evaluating applicants, aiding in more informed decision-making.**