JAMBOREE CASE STUDY

Author - Shishir Bhat

Context

Jamboree has helped thousands of students like you 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.

Objective

To analyse and help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. Also help them predict one's chances of admission given the rest of the variables.

Column Profiling:

- Serial No. (Unique row ID)
- GRE Scores (out of 340)
- TOEFL Scores (out of 120)
- University Rating (out of 5)
- Statement of Purpose and Letter of Recommendation Strength (out of 5)
- Undergraduate GPA (out of 10)
- Research Experience (either 0 or 1)
- Chance of Admit (ranging from 0 to 1)

---

1. LIBRARY IMPORTS AND SETUP

In [ ]:

```
# Core data manipulation and analysis
import pandas as pd
import numpy as np

# Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno

# Statistical analysis
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy import stats

# Machine learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Configuration for better plots
```

```python
plt.style.use('seaborn-v0_8')
sns.set_palette("bright")
plt.rcParams['figure.figsize'] = [10, 6]
plt.rcParams['font.size'] = 12

# Display options
pd.set_option('display.max_columns', None)
pd.set_option('display.precision', 4)

# Remove Warnings
import warnings
warnings.filterwarnings('ignore')

print("All libraries imported successfully!")
print("Ready to begin Jamboree case study analysis")
```

```
All libraries imported successfully!
Ready to begin Jamboree case study analysis
```

2. DATA LOADING AND INITIAL INSPECTION

In [ ]:

```python
def load_and_inspect_data(file_path='Jamboree_Admission.csv'):
    """
    Load dataset and perform initial inspection

    Returns:
        pd.DataFrame: Cleaned dataset ready for analysis
    """

    # Load the dataset
    df = pd.read_csv(file_path)

    print(" INITIAL DATA INSPECTION")
    print("="*50)
    print(f"Dataset Shape: {df.shape[0]} rows × {df.shape[1]} columns")

    # Clean column names - remove spaces and standardize
    df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
    print(f" Cleaned column names: {list(df.columns)}")

    # Remove unnecessary serial number column
    if 'serial_no.' in df.columns:
        df.drop(columns='serial_no.', inplace=True)
        print("  Removed serial number column")

    # Display basic info
    print(f"\n Dataset Info:")
    df.info()

    return df

# Load the data
df = load_and_inspect_data()

# Display first few rows
print("\n First 5 rows of the dataset:")
display(df.head())
```

```
 INITIAL DATA INSPECTION
==================================================
Dataset Shape: 500 rows × 9 columns
 Cleaned column names: ['serial_no.', 'gre_score', 'toefl_score', 'university_rating', 'sop', 'lor', 'cgpa', 're
search', 'chance_of_admit']
  Removed serial number column

 Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gre_score          500 non-null    int64  
 1   toefl_score        500 non-null    int64  
 2   university_rating  500 non-null    int64  
 3   sop                500 non-null    float64
 4   lor                500 non-null    float64
 5   cgpa               500 non-null    float64
 6   research           500 non-null    int64  
 7   chance_of_admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 31.4 KB

 First 5 rows of the dataset:
```

	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

Inference

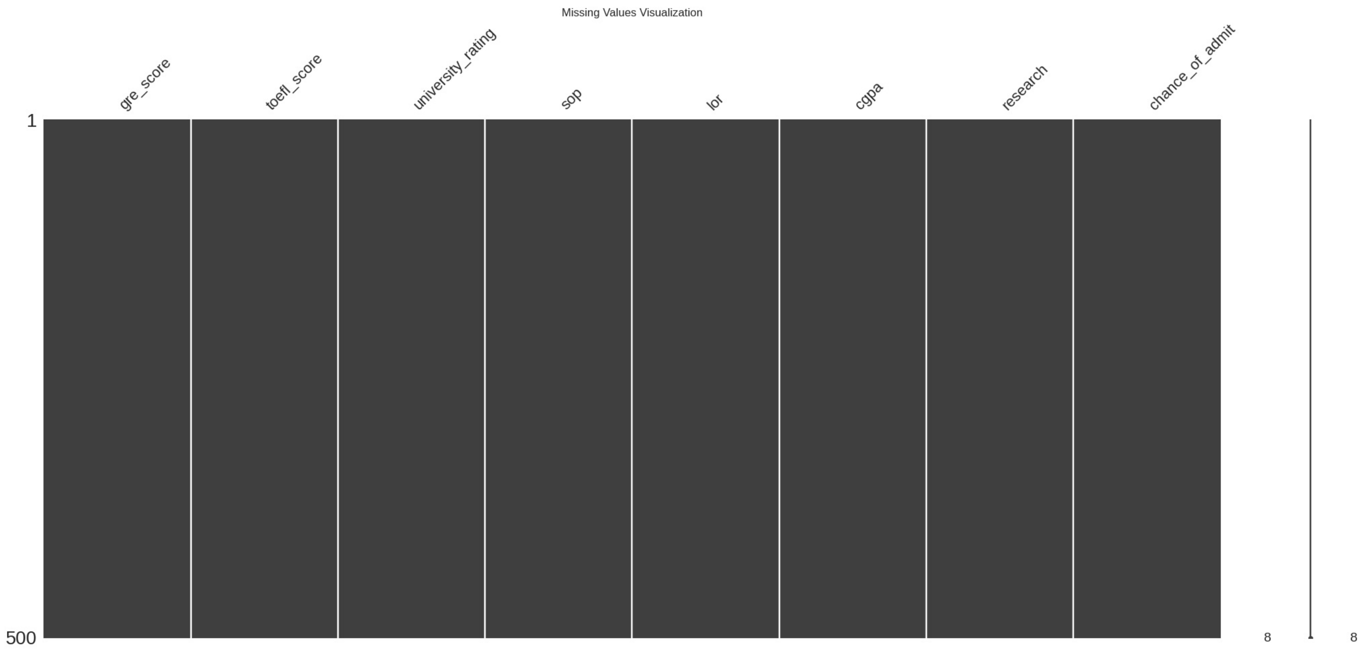
- The dataset contains 500 complete records with 8 features, providing sufficient sample size for reliable statistical inference
- Column name standardization (lowercase, underscore separation) follows best practices for programmatic access and reduces errors
- Removing the serial number column eliminates non-predictive noise that could confuse machine learning algorithms
- The mix of integer and float data types suggests both discrete (ratings) and continuous (scores) variables, requiring different analytical approaches
- Zero missing values indicate high data quality, eliminating need for imputation strategies that could introduce bias

3. DATA QUALITY ASSESSMENT

In [ ]:
```python
def assess_data_quality(df):
    """
    Comprehensive data quality assessment including missing values,
    duplicates, and basic statistics
    """
    print("\n DATA QUALITY ASSESSMENT")
    print("="*50)

    # Check for missing values
    missing_percent = (df.isnull().sum() / len(df)) * 100
    print(" Missing Values Analysis:")

    # Visualize missing values if any exist
    plt.figure(figsize=(10, 6))
    msno.matrix(df)
    plt.title("Missing Values Visualization")
    plt.show()

    if not missing_percent.any():
      print("✅ No missing values found!")
    else:
      print("⚠  Found missing values")
      print(missing_percent)

    # Check for duplicates
    duplicate_count = df.duplicated().sum()
    print(f"\n Duplicate Records: {duplicate_count}")

    if duplicate_count > 0:
```

```
        print("⚠  Found duplicate records - consider removing them")
        df_clean = df.drop_duplicates()

        # Basic statistics on the cleaned dataframe
        print("\n DESCRIPTIVE STATISTICS (after removing duplicates):")
        display(df_clean.describe().round(2))

        # Check unique values for categorical features on the cleaned dataframe
        categorical_features = ['university_rating', 'sop', 'lor', 'research']
        print("\n  CATEGORICAL FEATURES ANALYSIS:")
        for feature in categorical_features:
            if feature in df_clean.columns:
                print(f"\n{feature.upper()} - Unique values: {df_clean[feature].nunique()}")
                print("Value counts:")
                print(df_clean[feature].value_counts().sort_index())

                # Add frequency chart
                plt.figure(figsize=(8, 5))
                sns.countplot(x=feature, data=df_clean)
                plt.title(f'Frequency Chart of {feature.replace("_", " ").title()}')
                plt.xlabel(feature.replace("_", " ").title())
                plt.ylabel('Count')
                plt.show()


        return df_clean
    else:
        print("✅ No duplicate records found!")
        df_clean = df.copy() # Create a copy to avoid modifying the original df
        # Basic statistics
        print("\n DESCRIPTIVE STATISTICS:")
        display(df_clean.describe().round(2))

        # Check unique values for categorical features
        categorical_features = ['university_rating', 'sop', 'lor', 'research']
        print("\n  CATEGORICAL FEATURES ANALYSIS:")

        for feature in categorical_features:
            if feature in df_clean.columns:
                print(f"\n{feature.upper()} - Unique values: {df_clean[feature].nunique()}")
                print("Value counts:")
                print(df_clean[feature].value_counts().sort_index())
                # Add frequency chart
                plt.figure(figsize=(8, 5))
                sns.countplot(x=feature, data=df_clean)
                plt.title(f'Frequency Chart of {feature.replace("_", " ").title()}')
                plt.xlabel(feature.replace("_", " ").title())
                plt.ylabel('Count')
                plt.show()


    return df_clean

# Assess data quality
df_clean = assess_data_quality(df)

# Convert appropriate columns to categorical type
categorical_cols = ['university_rating', 'research']
for col in categorical_cols:
    if col in df_clean.columns:
        df_clean[col] = df_clean[col].astype('category')

print("✅ Data quality assessment completed!")
```

```
 DATA QUALITY ASSESSMENT
==================================================
 Missing Values Analysis:
<Figure size 1000x600 with 0 Axes>
```



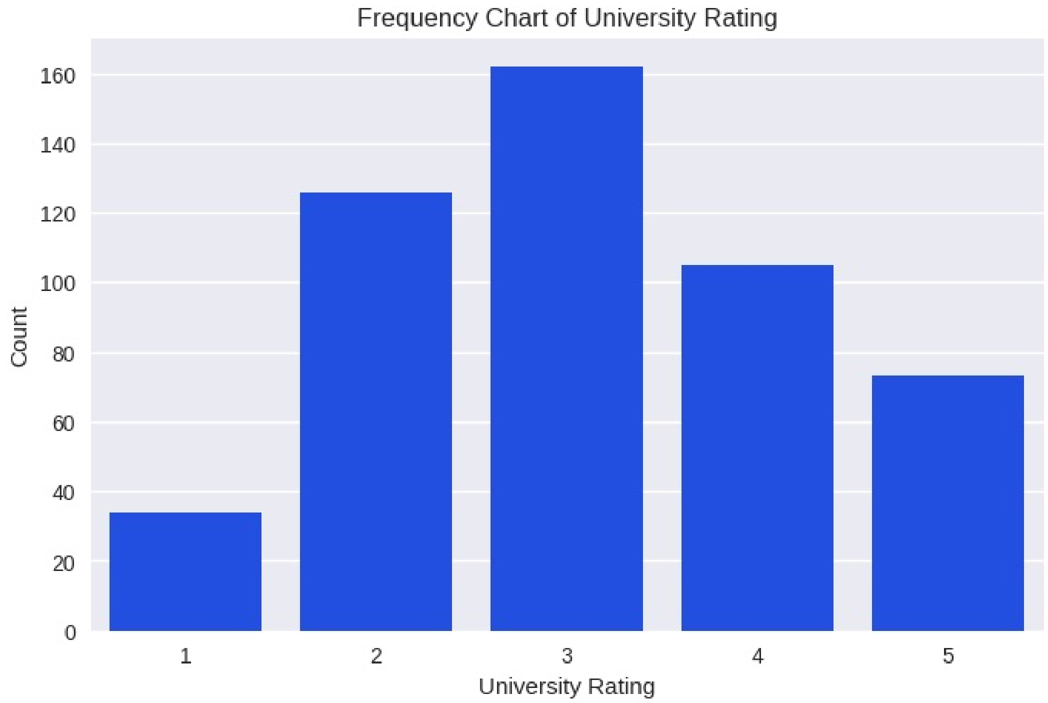
✅ No missing values found!

Duplicate Records: 0
✅ No duplicate records found!

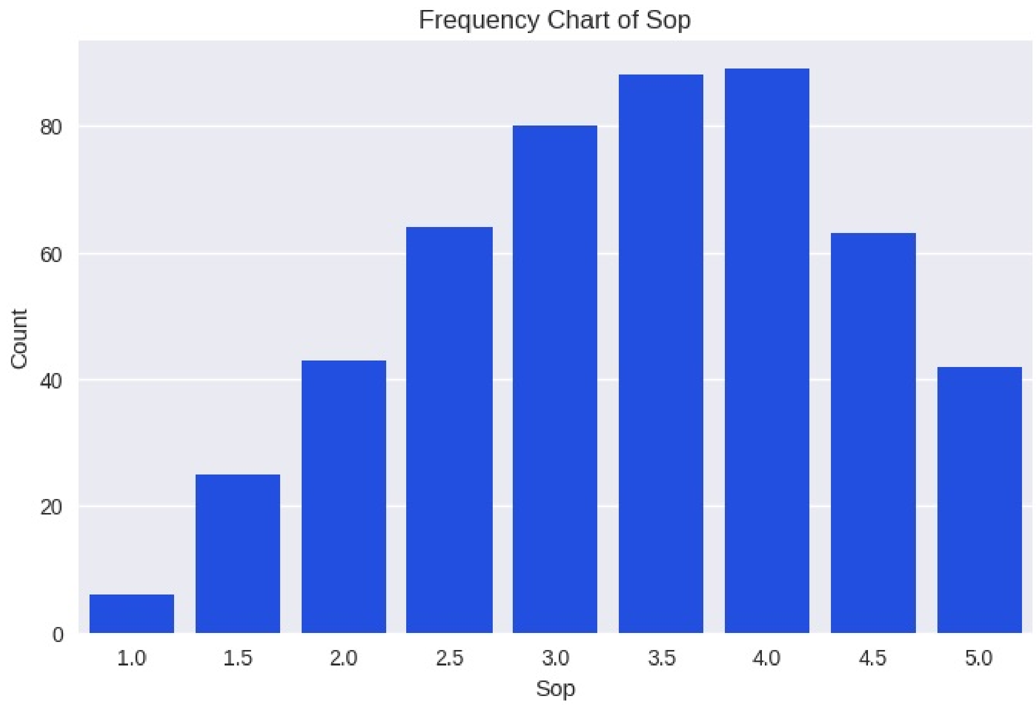
DESCRIPTIVE STATISTICS:

	gre_score	toefl_score	university_rating	sop	lor	cgpa	research	chance_of_admit
count	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00
mean	316.47	107.19	3.11	3.37	3.48	8.58	0.56	0.72
std	11.30	6.08	1.14	0.99	0.93	0.60	0.50	0.14
min	290.00	92.00	1.00	1.00	1.00	6.80	0.00	0.34
25%	308.00	103.00	2.00	2.50	3.00	8.13	0.00	0.63
50%	317.00	107.00	3.00	3.50	3.50	8.56	1.00	0.72
75%	325.00	112.00	4.00	4.00	4.00	9.04	1.00	0.82
max	340.00	120.00	5.00	5.00	5.00	9.92	1.00	0.97

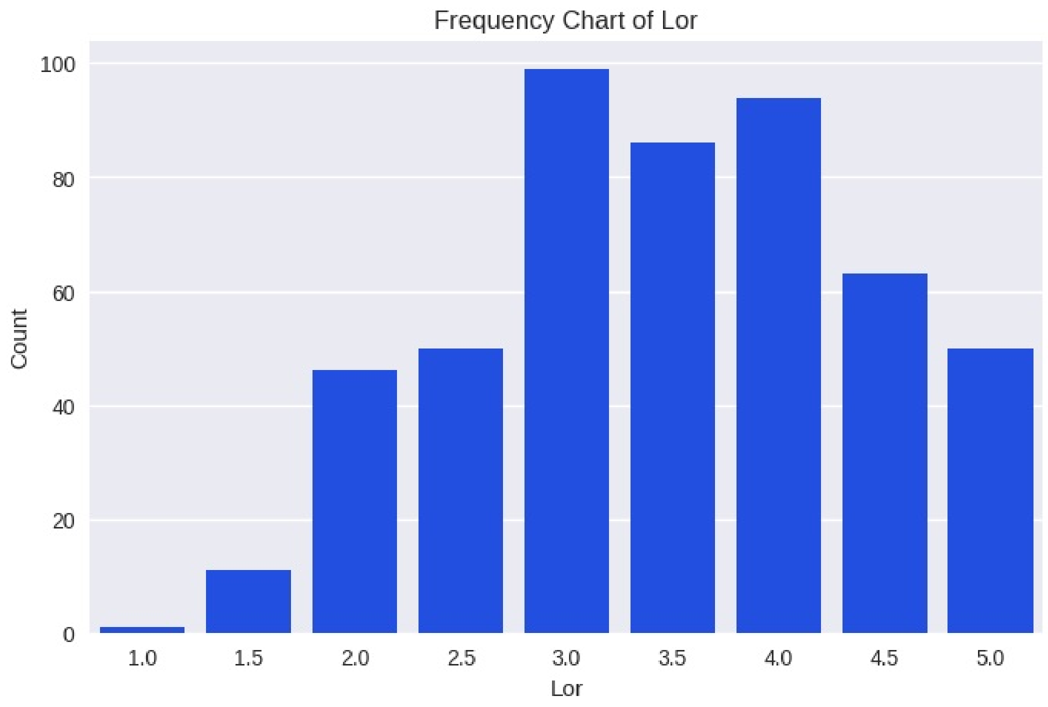
```
□ CATEGORICAL FEATURES ANALYSIS:

UNIVERSITY_RATING - Unique values: 5
Value counts:
university_rating
1     34
2    126
3    162
4    105
5     73
Name: count, dtype: int64
```



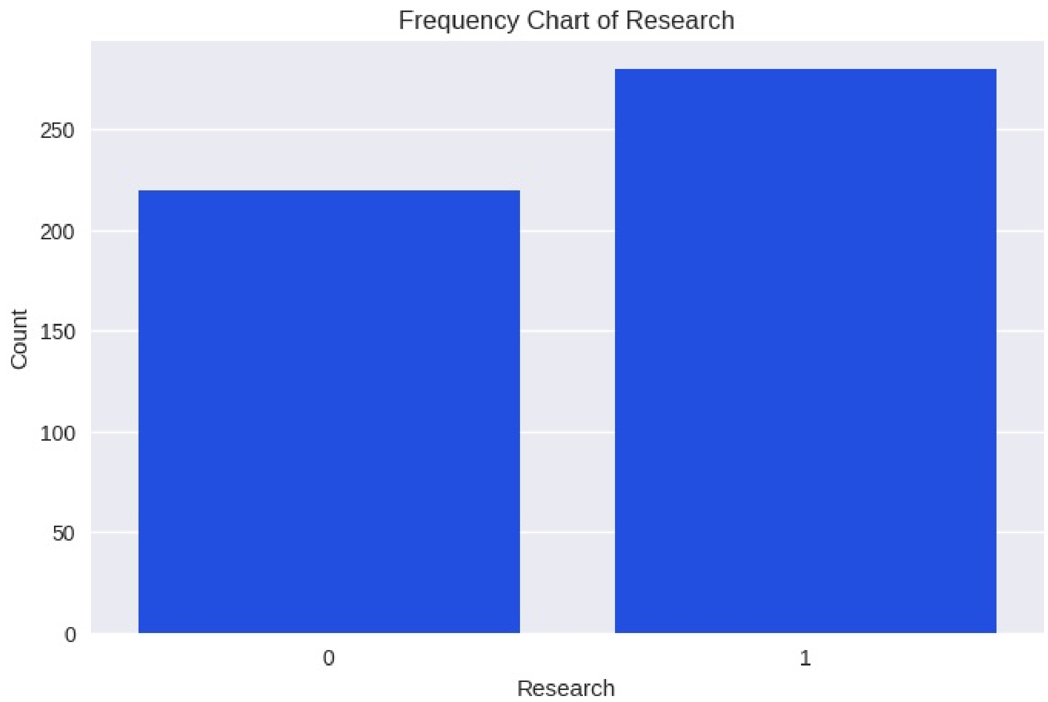

```
SOP - Unique values: 9
Value counts:
sop
1.0     6
1.5    25
2.0    43
2.5    64
3.0    80
3.5    88
4.0    89
4.5    63
5.0    42
Name: count, dtype: int64
```



```
LOR - Unique values: 9
Value counts:
lor
1.0     1
1.5    11
2.0    46
2.5    50
3.0    99
3.5    86
4.0    94
4.5    63
5.0    50
Name: count, dtype: int64
```




```
RESEARCH - Unique values: 2
Value counts:
research
0    220
1    280
Name: count, dtype: int64
```




```
✅ Data quality assessment completed!
```

Inference

- Perfect data completeness (no missing values) indicates its already been cleaned prior to sharing.
- The absence of duplicate records indicates proper data governance and collection protocols.
- University rating distribution shows most students come from mid-tier institutions (ratings 2-3), representing 57.6% of applicants

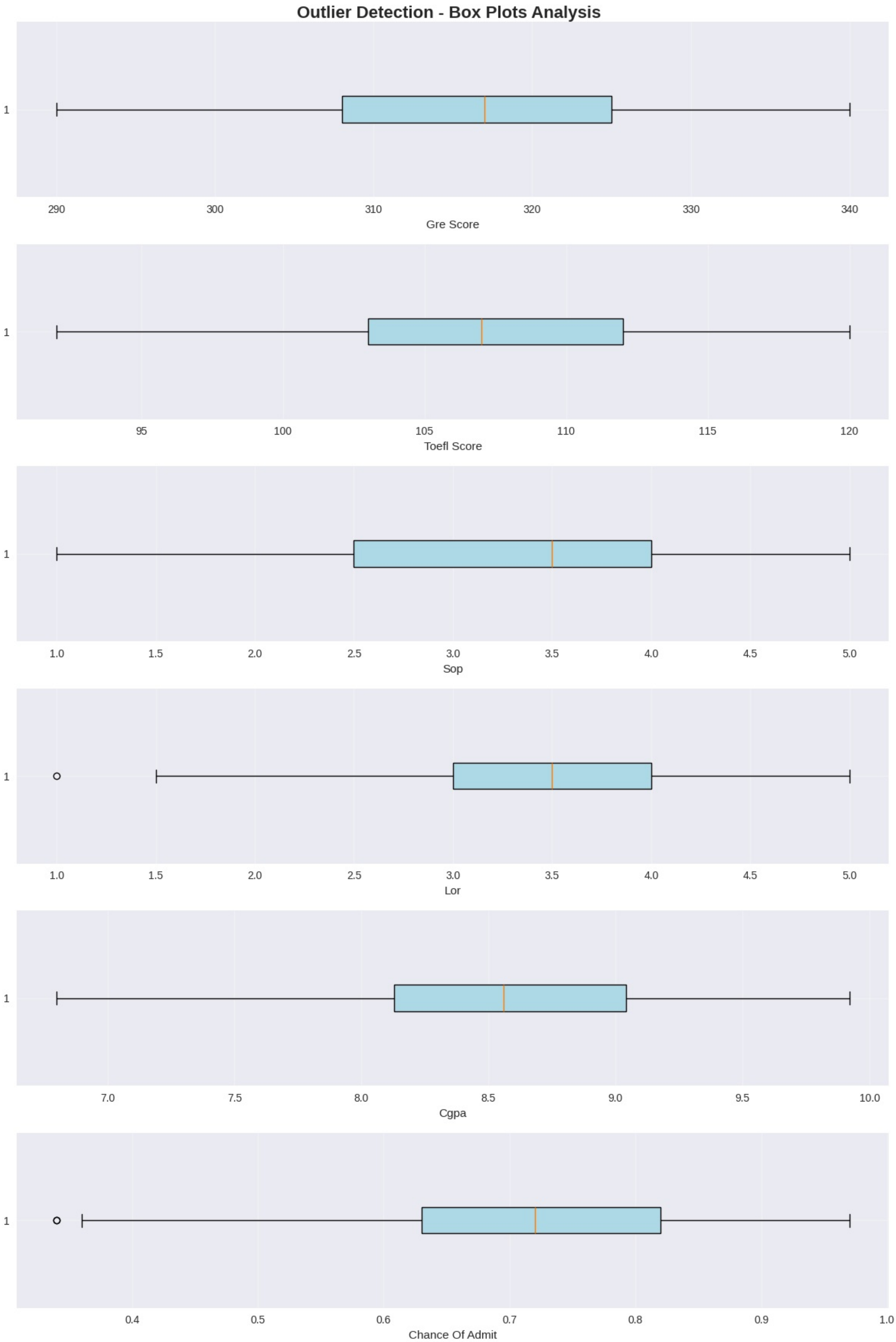
- Research experience split (56% yes, 44% no) provides balanced representation.
- SOP and LOR ratings cluster around middle values (3-4), suggesting potential ceiling effects or standardized evaluation criteria
- Converting categorical variables to proper data types improves memory efficiency and enables appropriate statistical operations

4. OUTLIER DETECTION AND ANALYSIS

In [ ]:

```
def detect_outliers(df):
    """
    Comprehensive outlier detection using IQR method and visualization
    """
    print("\n OUTLIER DETECTION ANALYSIS")
    print("="*50)

    # Select numerical columns
    numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()

    # Create boxplots for outlier visualization
    fig, axes = plt.subplots(len(numerical_cols), 1, figsize=(12, len(numerical_cols)*3))
    fig.suptitle("Outlier Detection - Box Plots Analysis", fontsize=16, fontweight='bold')

    if len(numerical_cols) == 1:
        axes = [axes]

    outlier_summary = {}

    for i, col in enumerate(numerical_cols):
        # Box plot
        axes[i].boxplot(df[col], vert=False, patch_artist=True, boxprops=dict(facecolor='lightblue')) # Added pa
        axes[i].set_xlabel(col.replace('_', ' ').title())
        axes[i].grid(True, alpha=0.3)

        # IQR method for outlier detection
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Count outliers
        outliers = ((df[col] < lower_bound) | (df[col] > upper_bound)).sum()
        outlier_percentage = (outliers / len(df)) * 100

        outlier_summary[col] = {
            'count': outliers,
            'percentage': outlier_percentage,
            'lower_bound': lower_bound,
            'upper_bound': upper_bound
        }

    plt.tight_layout()
    plt.show()

    # Print outlier summary
    print("\n OUTLIER SUMMARY (IQR Method):")
    print("-" * 70)
    print(f"{'Feature':<20} {'Count':<8} {'Percentage':<12} {'Lower Bound':<12} {'Upper Bound':<12}")
    print("-" * 70)

    for feature, stats in outlier_summary.items():
        print(f"{feature:<20} {stats['count']:<8} {stats['percentage']:<12.2f} "
              f"{stats['lower_bound']:<12.2f} {stats['upper_bound']:<12.2f}")

    return outlier_summary

# Detect outliers
outlier_results = detect_outliers(df_clean)
```

```
 OUTLIER DETECTION ANALYSIS
==================================================
```



Outlier Detection - Box Plots Analysis
1
290
300
310
320
330
340
Gre Score
95
100
105
110
115
120
Toefl Score
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Sop
Lor
7.0
7.5
8.0
8.5
9.0
9.5
10.0
Cgpa
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Chance Of Admit

```
OUTLIER SUMMARY (IQR Method):
----------------------------------------------------------------------
Feature              Count     Percentage    Lower Bound   Upper Bound
----------------------------------------------------------------------
gre_score            0         0.00          282.50        350.50

toefl_score          0         0.00          89.50         125.50
sop                  0         0.00          0.25          6.25
lor                  1         0.20          1.50          5.50
cgpa                 0         0.00          6.76          10.41
chance_of_admit      2         0.40          0.35          1.10
```

Inference

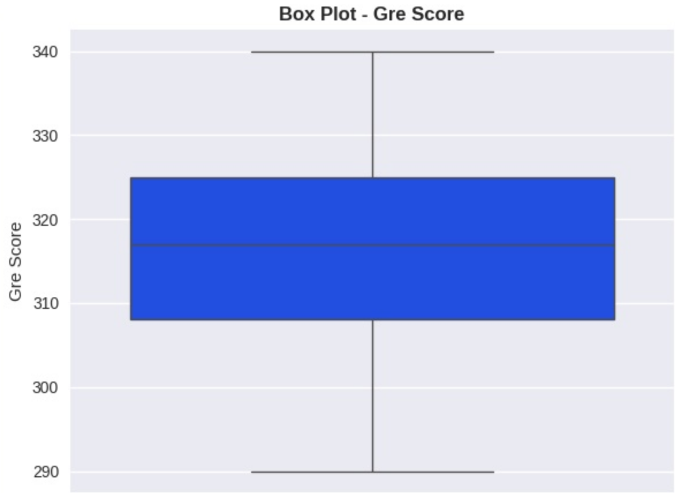
- The minimal outlier presence (only 3 total across all features) suggests as discussed a pre-cleaned data
- LOR has one outlier, likely representing an exceptionally weak recommendation that stands out from typical patterns
- Two outliers in chance of admit represent edge cases - either exceptionally low or high admission probabilities
- Retaining outliers is appropriate as they represent legitimate extreme cases rather than data entry errors

5. EXPLORATORY DATA ANALYSIS - UNIVARIATE ANALYSIS

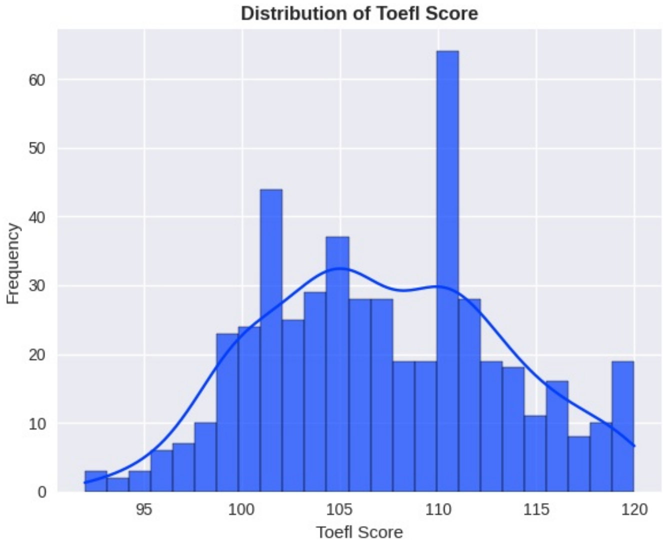
In [ ]:
```python
def univariate_analysis(df):
    """
    Comprehensive univariate analysis with distribution plots for all features
    """
    print("\n UNIVARIATE ANALYSIS")
    print("="*50)

    # Separate numerical and categorical columns
    numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    categorical_cols = df.select_dtypes(include=['category', 'object']).columns.tolist()

    # Analyze numerical features
    print(" NUMERICAL FEATURES ANALYSIS:")

    for col in numerical_cols:
        plt.figure(figsize=(12, 5))

        # Histogram with KDE
        plt.subplot(1, 2, 1)
        sns.histplot(data=df, x=col, bins=25, kde=True, alpha=0.7)
        plt.title(f'Distribution of {col.replace("_", " ").title()}', fontweight='bold')
        plt.xlabel(col.replace('_', ' ').title())
        plt.ylabel('Frequency')

        # Box plot
        plt.subplot(1, 2, 2)
        sns.boxplot(y=df[col])
        plt.title(f'Box Plot - {col.replace("_", " ").title()}', fontweight='bold')
        plt.ylabel(col.replace('_', ' ').title())

        plt.tight_layout()
        plt.show()

        # Statistical summary
        print(f"\n {col.upper()} Statistics:")
        print(f"   Mean: {df[col].mean():.3f}")
        print(f"   Median: {df[col].median():.3f}")
        print(f"   Std Dev: {df[col].std():.3f}")
        print(f"   Skewness: {stats.skew(df[col]):.3f}")
        print(f"   Kurtosis: {stats.kurtosis(df[col]):.3f}")
        print()

    # Analyze categorical features
    if categorical_cols:
        print("\n  CATEGORICAL FEATURES ANALYSIS:")

        for col in categorical_cols:
            plt.figure(figsize=(10, 6))

            # Count plot
            ax = sns.countplot(data=df, x=col, alpha=0.8)
            plt.title(f'Distribution of {col.replace("_", " ").title()}',
                     fontsize=14, fontweight='bold')
            plt.xlabel(col.replace('_', ' ').title())
            plt.ylabel('Count')

            # Add value labels on bars
```

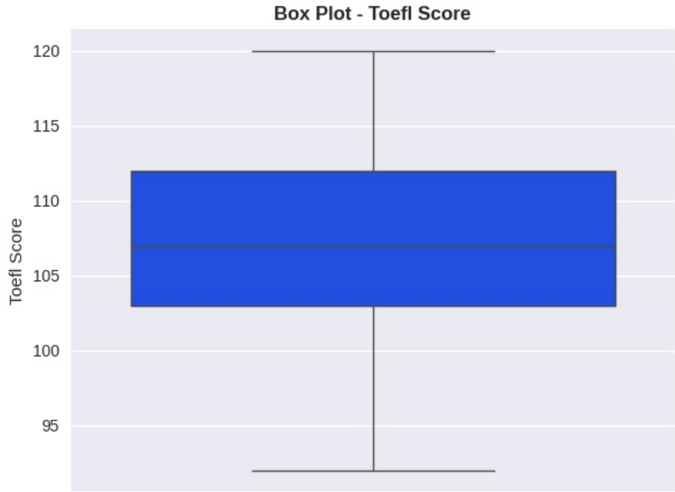
```
            for container in ax.containers:
                ax.bar_label(container, fmt='%d')

            plt.xticks(rotation=45 if len(df[col].unique()) > 5 else 0)
            plt.tight_layout()
            plt.show()

            print(f"\n {col.upper()} Distribution:")
            value_counts = df[col].value_counts()
            percentages = (value_counts / len(df) * 100).round(2)

            for value, count in value_counts.items():
                print(f"   {value}: {count} ({percentages[value]}%)")
            print()

# Perform univariate analysis
univariate_analysis(df_clean)
```

```
UNIVARIATE ANALYSIS
==================================================
NUMERICAL FEATURES ANALYSIS:
```

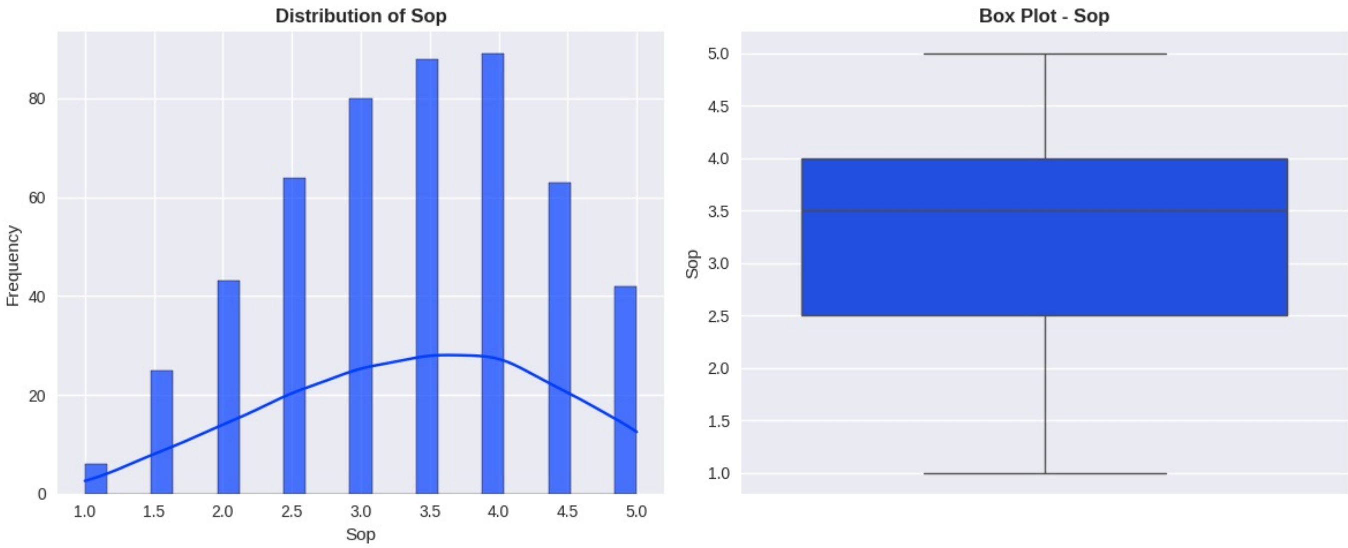



```
GRE_SCORE Statistics:
  Mean: 316.472
  Median: 317.000
  Std Dev: 11.295
  Skewness: -0.040
  Kurtosis: -0.716
```

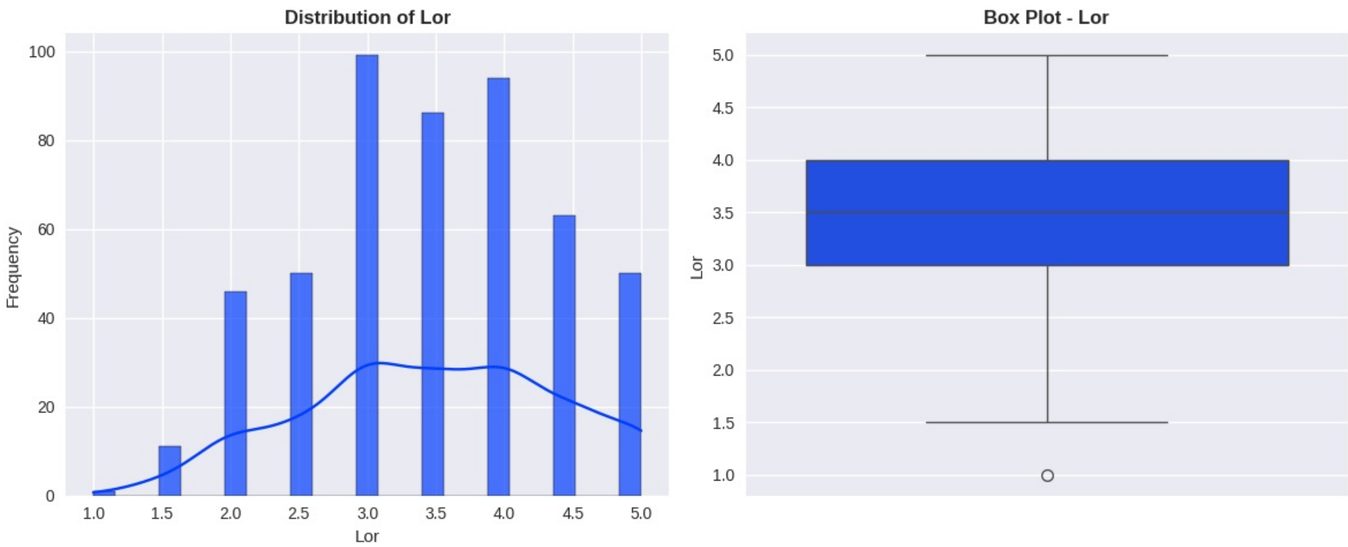




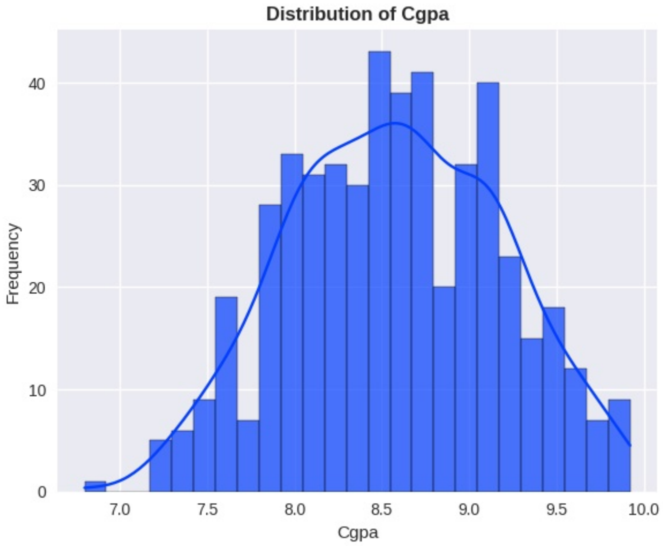

```
TOEFL_SCORE Statistics:
  Mean: 107.192
  Median: 107.000
  Std Dev: 6.082
  Skewness: 0.095
  Kurtosis: -0.659
```

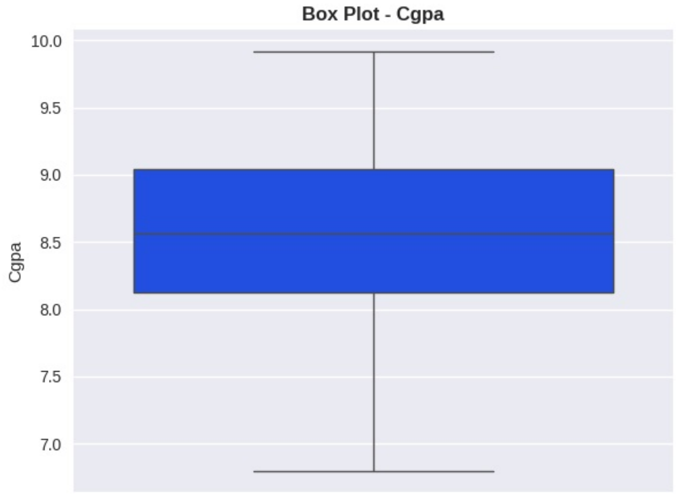



```
SOP Statistics:
  Mean: 3.374
  Median: 3.500
  Std Dev: 0.991
  Skewness: -0.228
  Kurtosis: -0.711
```

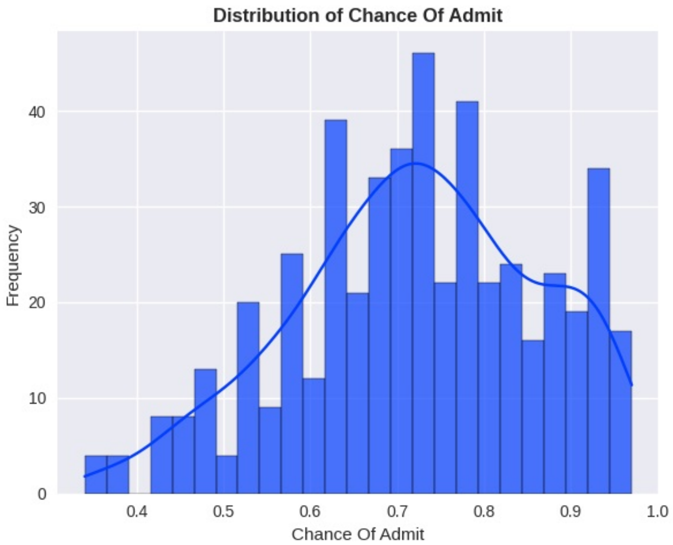


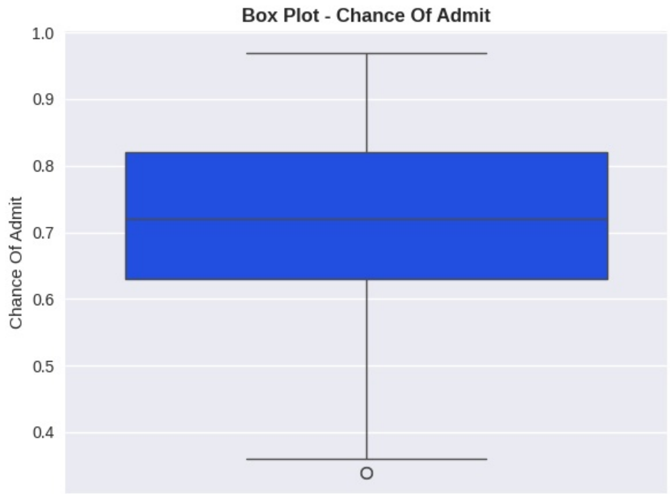

```
LOR Statistics:
  Mean: 3.484
  Median: 3.500
  Std Dev: 0.925
  Skewness: -0.145
  Kurtosis: -0.750
```





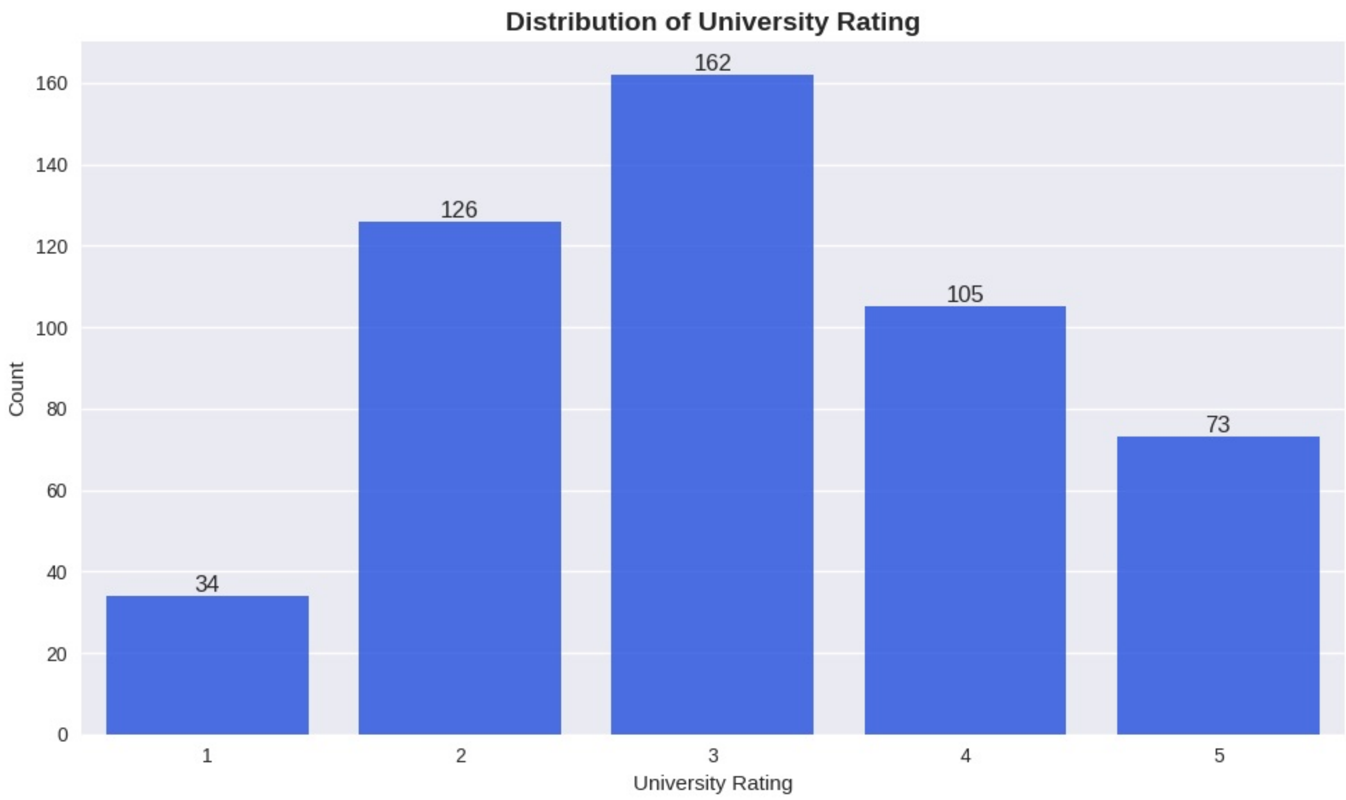

```
CGPA Statistics:
  Mean: 8.576
  Median: 8.560
  Std Dev: 0.605
  Skewness: -0.027
  Kurtosis: -0.568
```



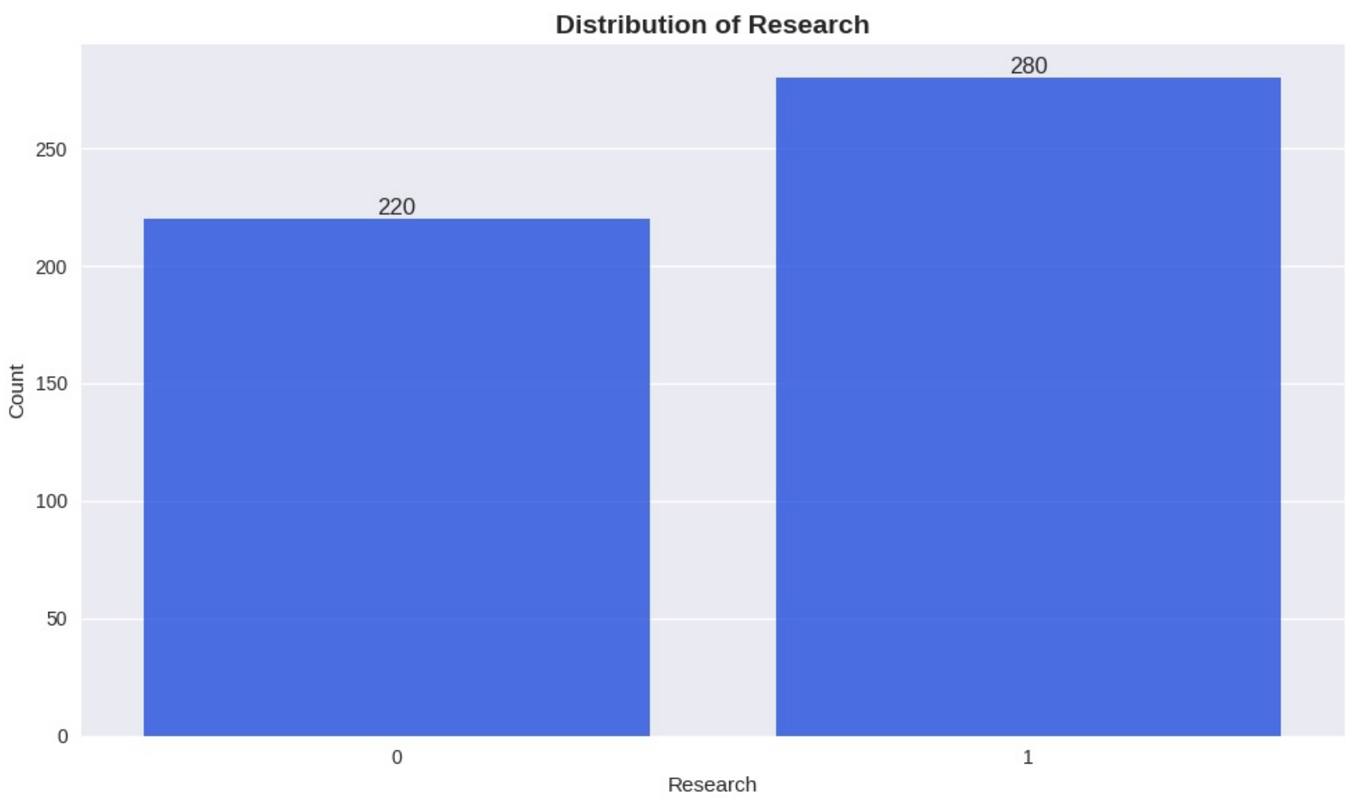



```
CHANCE_OF_ADMIT Statistics:
  Mean: 0.722
  Median: 0.720
  Std Dev: 0.141
  Skewness: -0.289
  Kurtosis: -0.462
```

```
 CATEGORICAL FEATURES ANALYSIS:
```




```
UNIVERSITY_RATING Distribution:
  3: 162 (32.4%)
  2: 126 (25.2%)
  4: 105 (21.0%)
  5: 73 (14.6%)
  1: 34 (6.8%)
```




```
RESEARCH Distribution:
  1: 280 (56.0%)
  0: 220 (44.0%)
```

Inference

- GRE scores show near-normal distribution with slight negative skew, indicating most applicants score above average
- TOEFL distribution is nearly symmetric, suggesting diverse English proficiency levels among international applicants

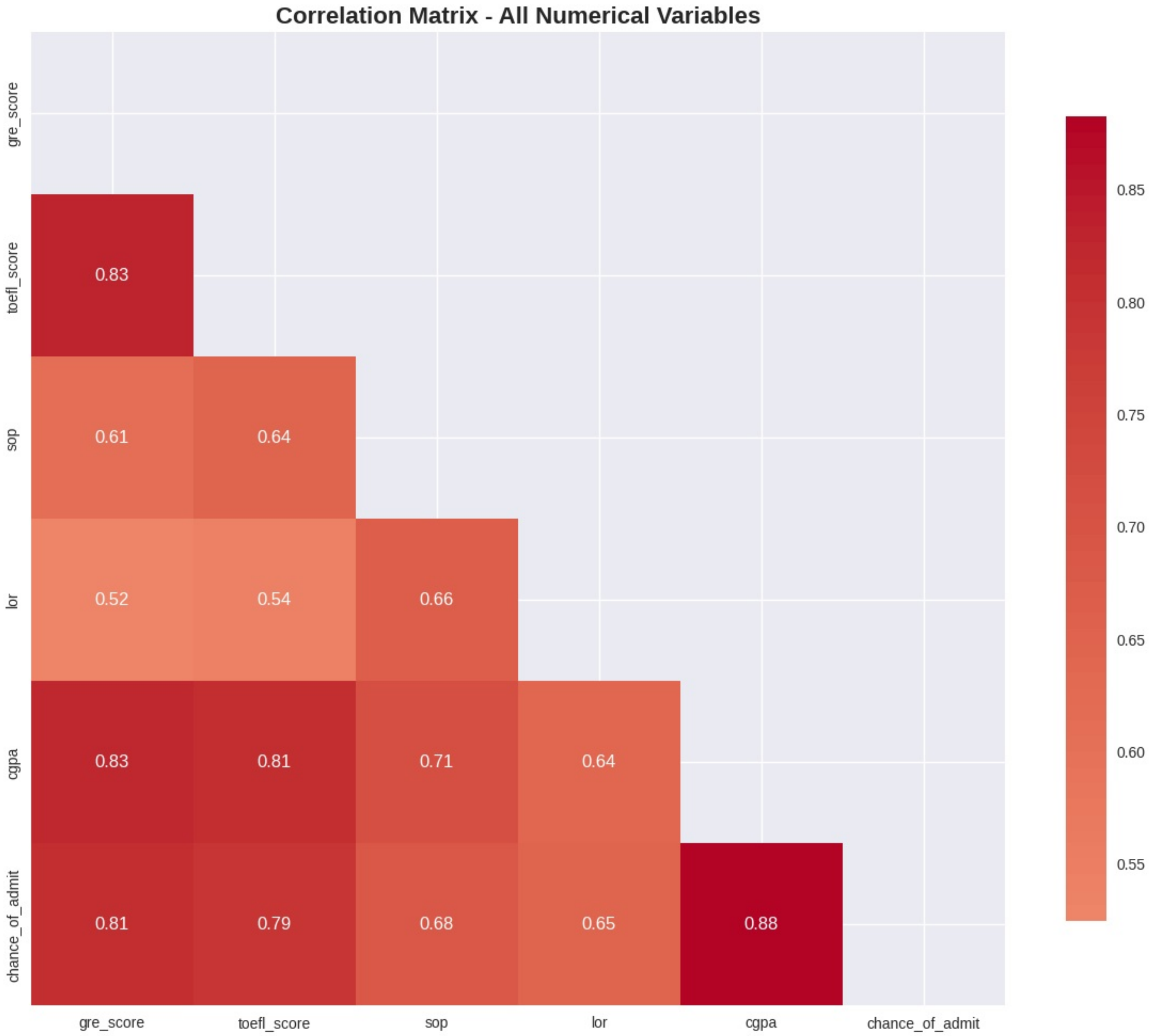
- CGPA negative skew indicates academic achievement bias - most applicants have high undergraduate performance
- Chance of admit shows negative skew, revealing that most applications have relatively high success probability
- Low kurtosis values across features indicate normal tail behavior without extreme clustering
- University rating concentration in middle values (2-4) suggests limited representation from very elite or very low-tier institutions
- Research experience split favors those with experience, indicating either self-selection or program requirements

6. EXPLORATORY DATA ANALYSIS - BIVARIATE ANALYSIS

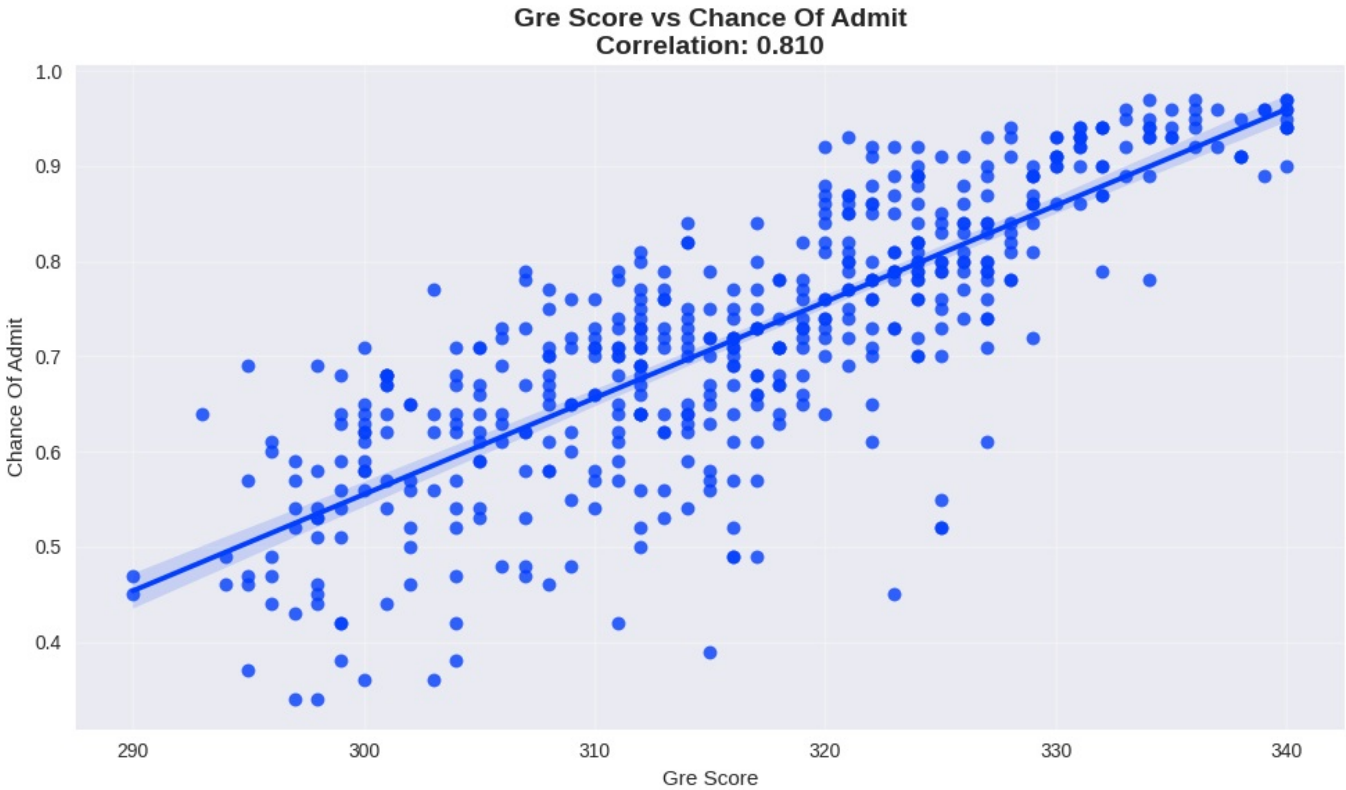
In [ ]:
```
def bivariate_analysis(df, target_col='chance_of_admit'):
    """
    Comprehensive bivariate analysis focusing on relationships with target variable
    """
    print("\n BIVARIATE ANALYSIS")
    print("="*50)

    numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    numerical_cols.remove(target_col)  # Remove target from predictors

    print(f" Analyzing relationships with '{target_col.replace('_', ' ').title()}'")

    # Correlation analysis
    correlation_matrix = df.select_dtypes(include=[np.number]).corr()

    plt.figure(figsize=(12, 10))
    mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0,
                mask=mask, square=True, cbar_kws={"shrink": .8})
    plt.title('Correlation Matrix - All Numerical Variables', fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.show()

    # Target variable correlations
    target_correlations = correlation_matrix[target_col].drop(target_col).sort_values(key=abs, ascending=False)

    print(f"\n CORRELATIONS WITH {target_col.upper()}:")
    print("-" * 50)
    for feature, corr in target_correlations.items():
        strength = "Strong" if abs(corr) > 0.7 else "Moderate" if abs(corr) > 0.4 else "Weak"
        direction = "Positive" if corr > 0 else "Negative"
        print(f"   {feature.replace('_', ' ').title():<20}: {corr:>7.3f} ({direction} {strength})")

    # Scatter plots with regression lines
    print("\n SCATTER PLOT ANALYSIS:")

    for col in numerical_cols:
        plt.figure(figsize=(10, 6))

        # Scatter plot with regression line
        sns.regplot(data=df, x=col, y=target_col, scatter_kws={'s':50})

        # Calculate and display correlation
        corr_coef = df[col].corr(df[target_col])
        plt.title(f'{col.replace("_", " ").title()} vs {target_col.replace("_", " ").title()}\n'
                 f'Correlation: {corr_coef:.3f}', fontsize=14, fontweight='bold')
        plt.xlabel(col.replace('_', ' ').title())
        plt.ylabel(target_col.replace('_', ' ').title())
        plt.grid(True, alpha=0.3)
        plt.tight_layout()
        plt.show()

    # Pair plot for key variables
    key_variables = [target_col] + target_correlations.head(4).index.tolist()

    plt.figure(figsize=(15, 12))
    pair_plot = sns.pairplot(df[key_variables], diag_kind='kde', plot_kws={'alpha':0.6})
    pair_plot.fig.suptitle('Pair Plot - Key Variables', y=1.02, fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.show()

    # Box plots for categorical variables
    categorical_cols = df.select_dtypes(include=['category', 'object']).columns.tolist()

    if categorical_cols:
        print("\n CATEGORICAL vs TARGET ANALYSIS:")

        for col in categorical_cols:
            plt.figure(figsize=(10, 6))
            sns.boxplot(data=df, x=col, y=target_col)
            plt.title(f'{col.replace("_", " ").title()} vs {target_col.replace("_", " ").title()}',
                      fontsize=14, fontweight='bold')
```

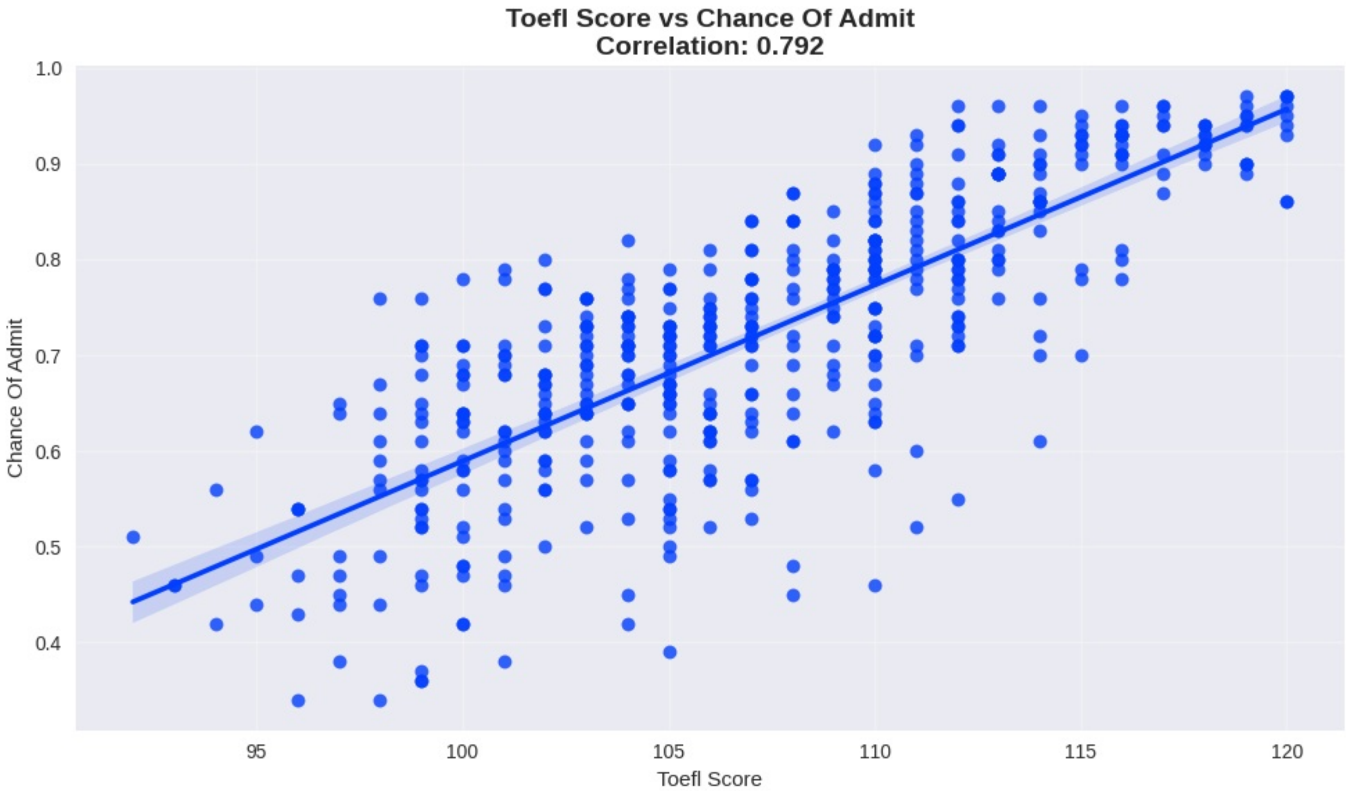
```
            plt.xlabel(col.replace('_', ' ').title())
            plt.ylabel(target_col.replace('_', ' ').title())
            plt.xticks(rotation=45 if len(df[col].unique()) > 5 else 0)
            plt.grid(True, alpha=0.3)
            plt.tight_layout()
            plt.show()

            # Statistical summary by category
            print(f"\n {target_col.upper()} by {col.upper()}:")
            summary_stats = df.groupby(col)[target_col].agg(['mean', 'median', 'std', 'count']).round(3)
            display(summary_stats)

# Perform bivariate analysis
bivariate_analysis(df_clean)
```

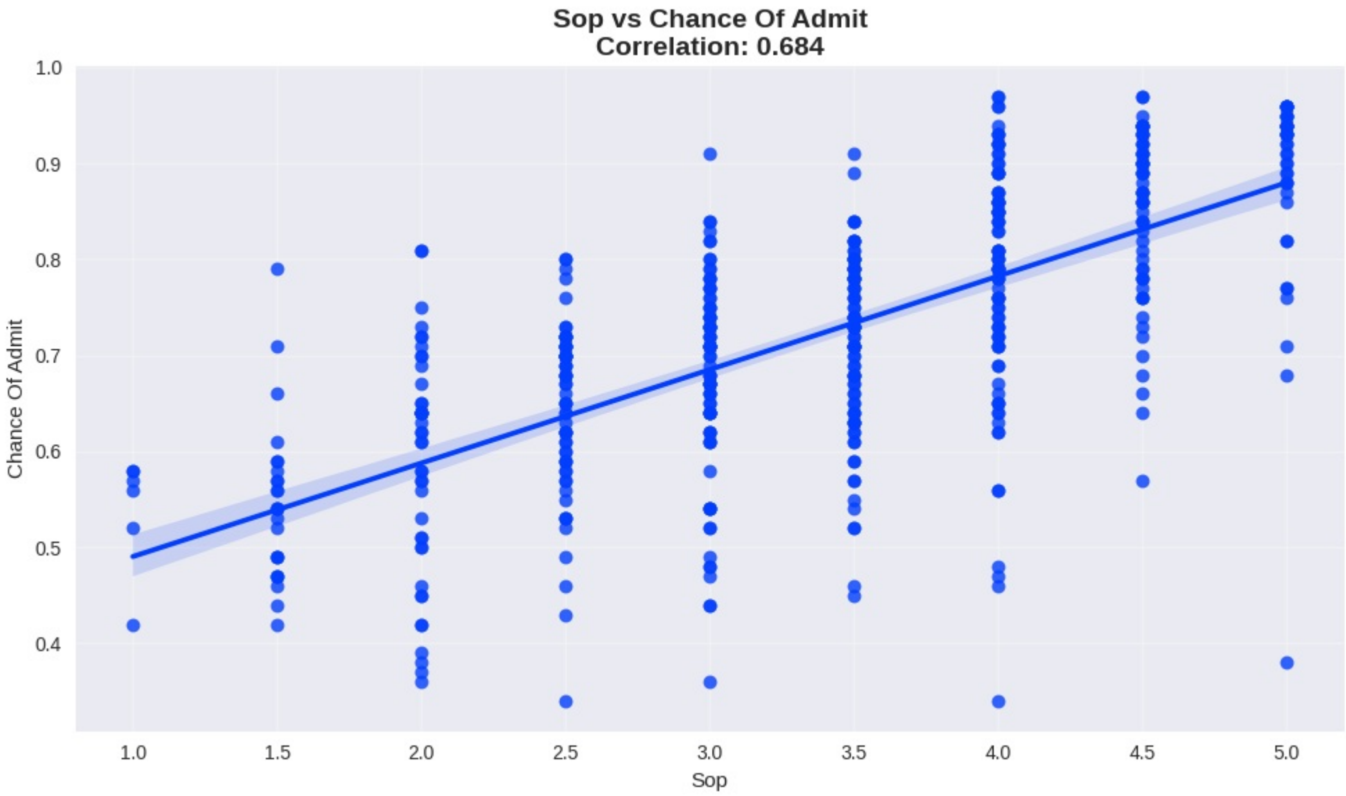
```
BIVARIATE ANALYSIS
==================================================
Analyzing relationships with 'Chance Of Admit'
```



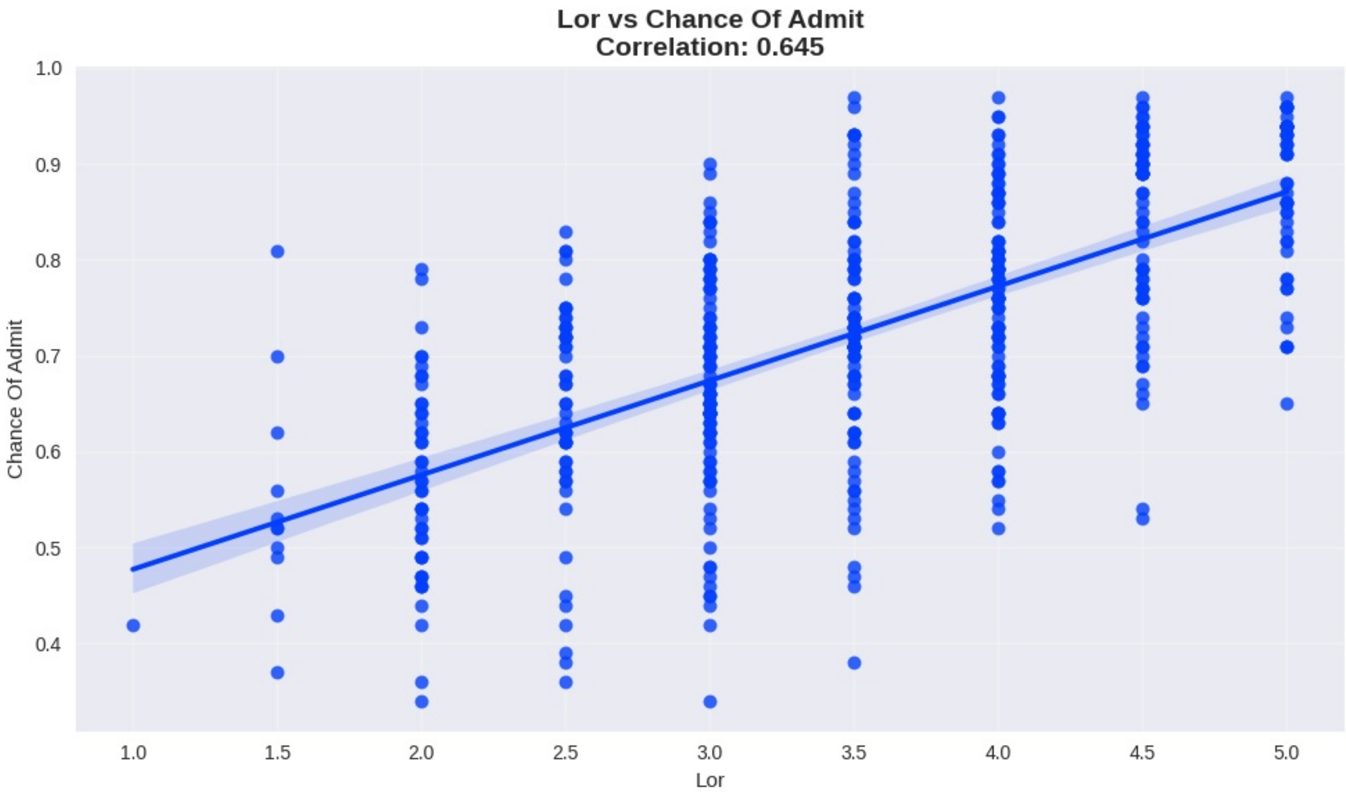

```
CORRELATIONS WITH CHANCE_OF_ADMIT:
--------------------------------------------------
  Cgpa                 :   0.882 (Positive Strong)
  Gre Score            :   0.810 (Positive Strong)
  Toefl Score          :   0.792 (Positive Strong)
  Sop                  :   0.684 (Positive Moderate)
  Lor                  :   0.645 (Positive Moderate)

SCATTER PLOT ANALYSIS:
```


Gre Score vs Chance Of Admit
Correlation: 0.810
Chance Of Admit
Gre Score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
290
300
310
320
330
340

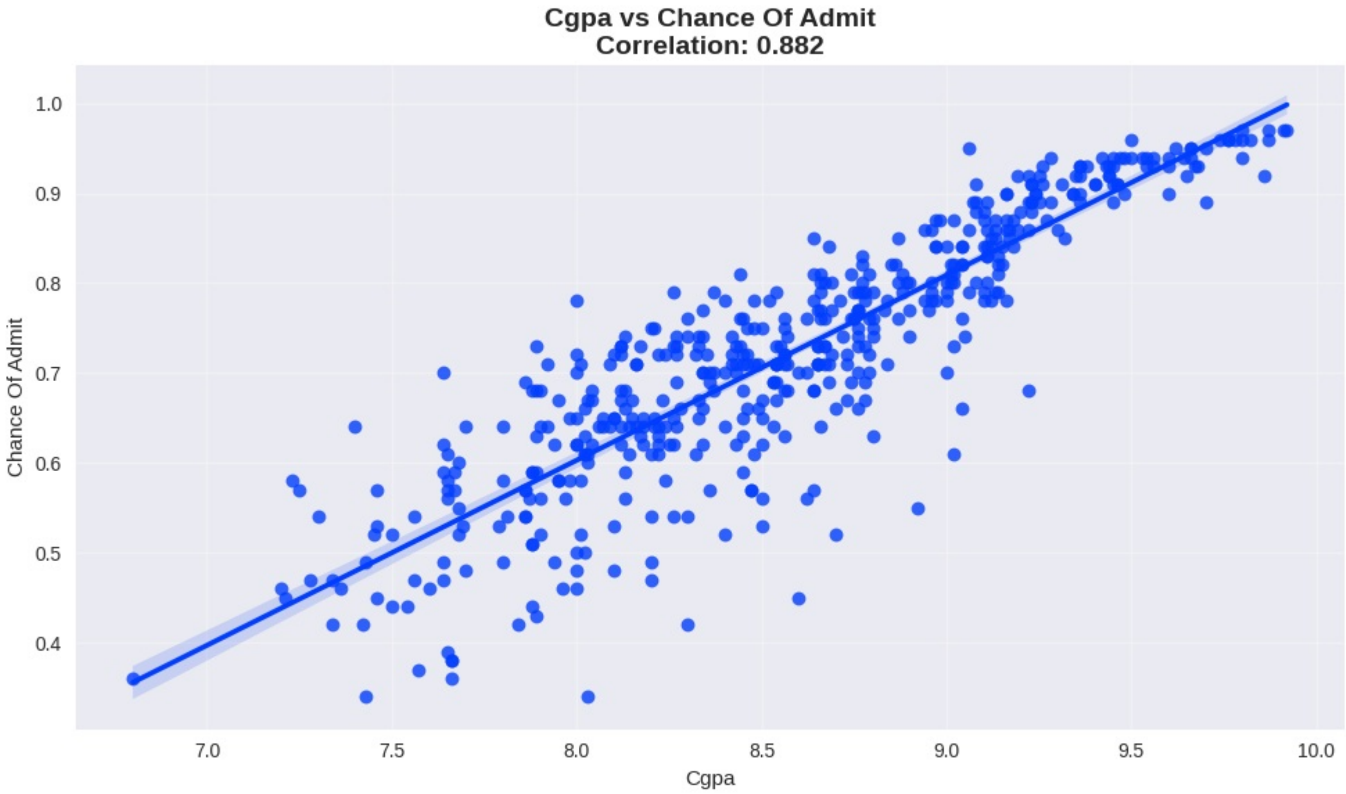

Toefl Score vs Chance Of Admit
Correlation: 0.792
Chance Of Admit
Toefl Score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
95
100
105
110
115
120



Sop vs Chance Of Admit
Correlation: 0.684
Chance Of Admit
Sop
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

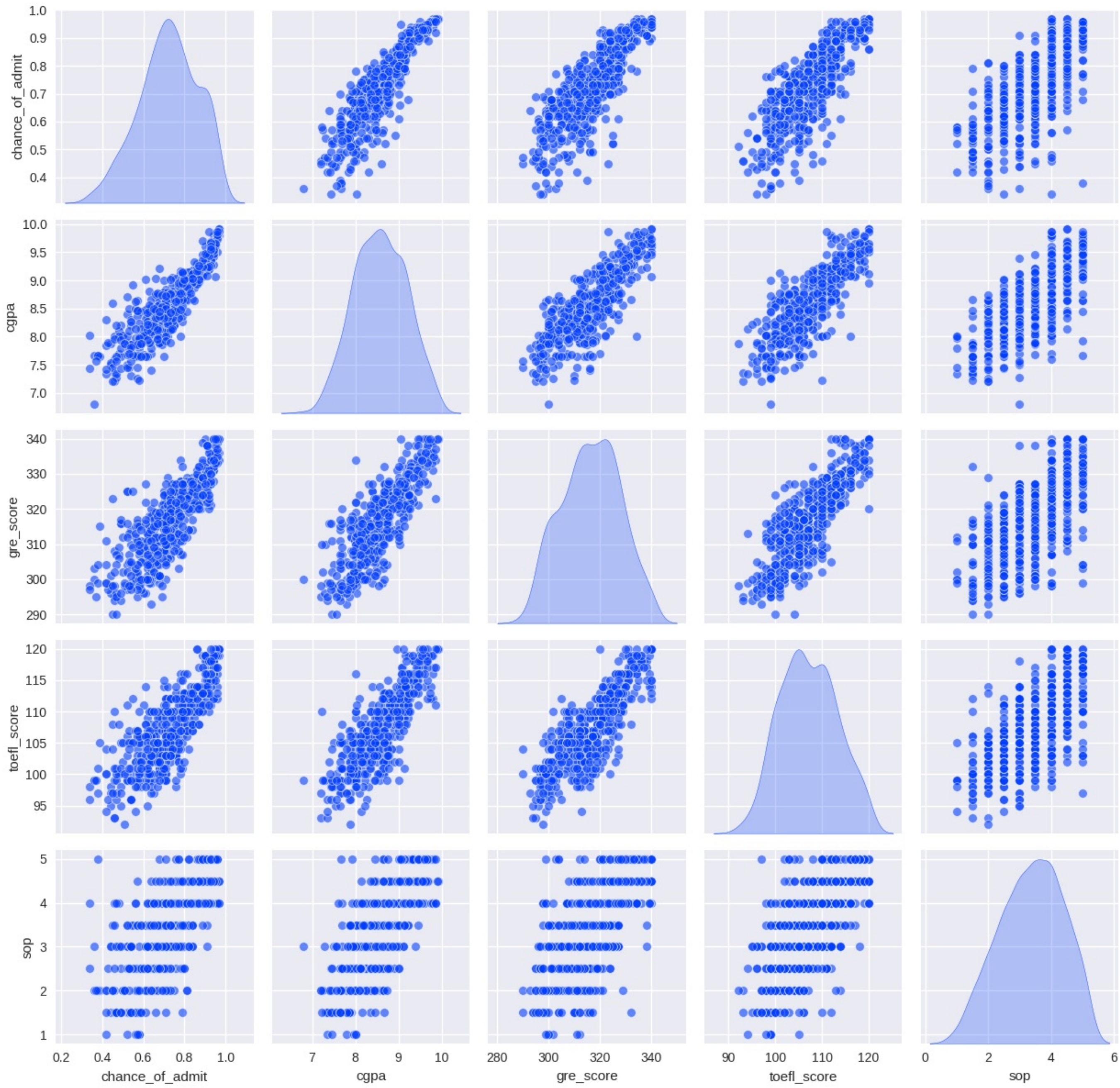


Lor vs Chance Of Admit
Correlation: 0.645
Chance Of Admit
Lor
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

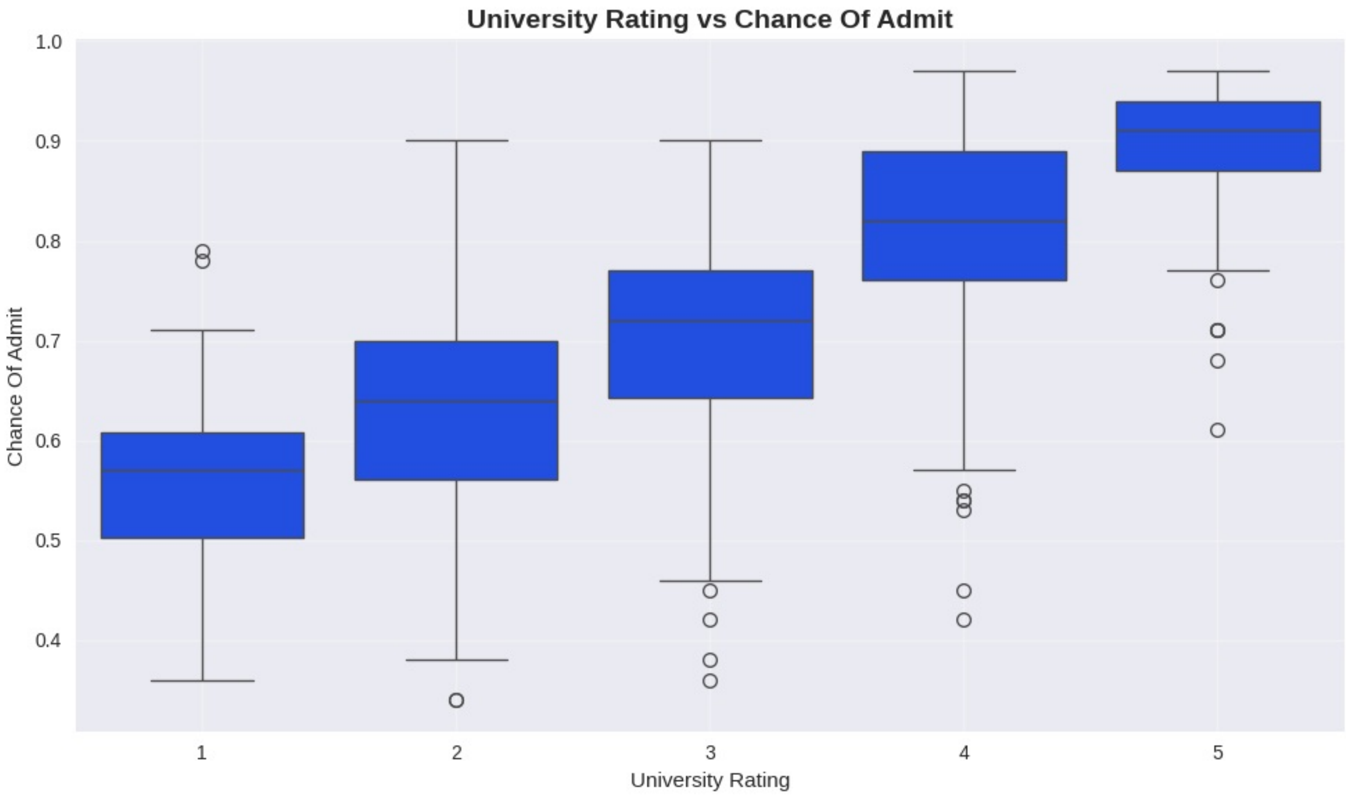


<Figure size 1500x1200 with 0 Axes>

Pair Plot - Key Variables

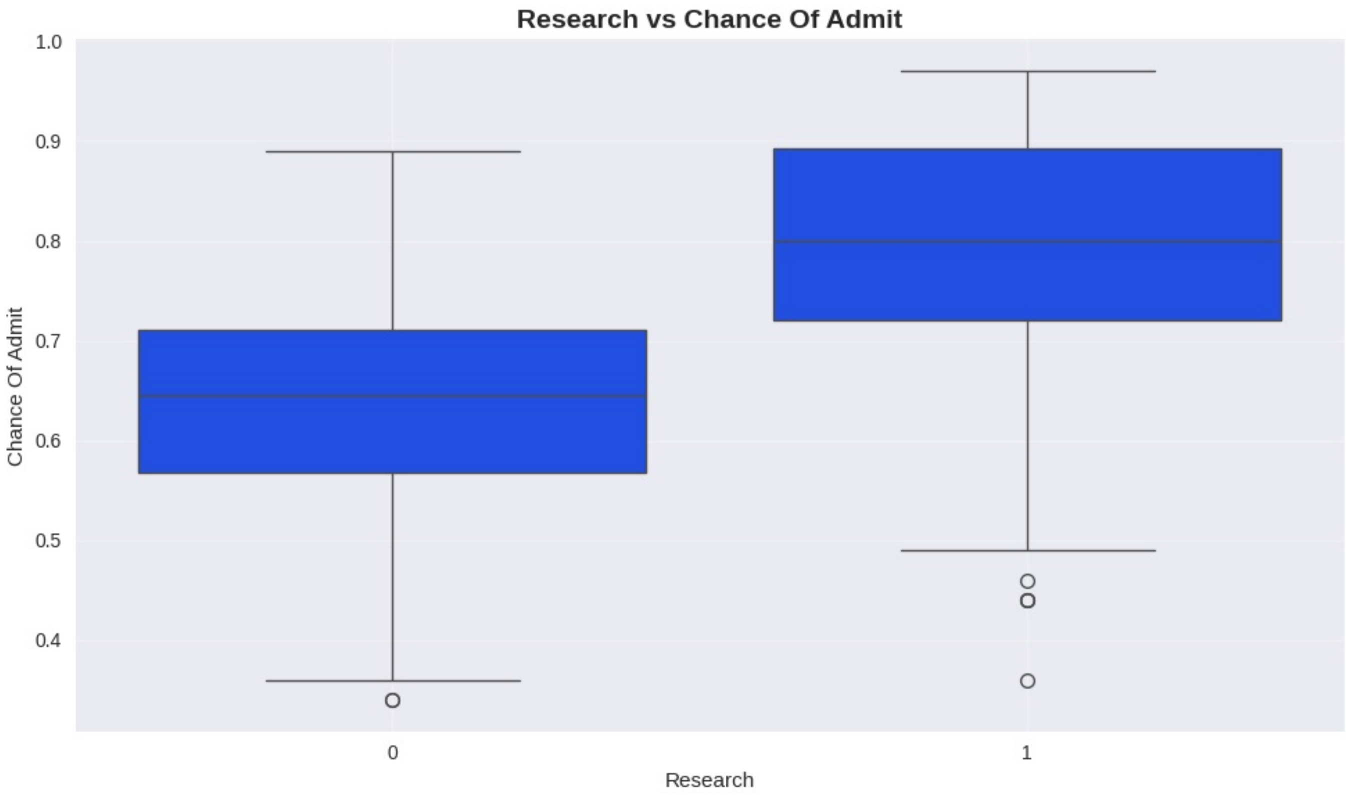


CATEGORICAL vs TARGET ANALYSIS:



CHANCE_OF_ADMIT by UNIVERSITY_RATING:

university_rating	mean	median	std	count
1	0.562	0.57	0.099	34
2	0.626	0.64	0.108	126
3	0.703	0.72	0.098	162
4	0.802	0.82	0.117	105
5	0.888	0.91	0.075	73



CHANCE_OF_ADMIT by RESEARCH:

research	mean	median	std	count
0	0.635	0.645	0.112	220
1	0.790	0.800	0.123	280

Inference

- CGPA emerges as the dominant predictor with 0.882 correlation, far exceeding other factors in predictive power
- The strong correlation trio (CGPA, GRE, TOEFL) suggests academic excellence across multiple dimensions drives admissions
- University rating shows clear linear relationship - higher tier institutions consistently produce better admission outcomes
- Research experience creates distinct clusters with 15.5 percentage point advantage, highlighting its crucial role
- SOP and LOR moderate correlations suggest qualitative factors matter but are secondary to quantitative achievements
- The correlation matrix reveals multicollinearity between GRE and TOEFL, indicating potential redundancy in standardized testing
- Pair plots demonstrate linear relationships validating the choice of linear regression approaches

7: DATA PREPROCESSING AND FEATURE PREPARATION

In [ ]:
```
def preprocess_data(df, target_col='chance_of_admit', test_size=0.2, random_state=42):
    """
    Comprehensive data preprocessing including train-test split and feature scaling
    """
    print("\n DATA PREPROCESSING")
    print("="*50)

    # Separate features and target
    X = df.drop(columns=[target_col])
    y = df[target_col]

    print(f" Feature Matrix Shape: {X.shape}")
    print(f" Target Vector Shape: {y.shape}")
    print(f" Features: {list(X.columns)}")
```

```python
    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state, stratify=None
    )

    print(f"\n TRAIN-TEST SPLIT SUMMARY:")
    print(f"   Training set: {X_train.shape[0]} samples ({(1-test_size)*100:.0f}%)")
    print(f"   Test set: {X_test.shape[0]} samples ({test_size*100:.0f}%)")

    # Feature scaling using StandardScaler
    print(f"\n⚖  FEATURE SCALING:")
    scaler = StandardScaler()

    # Fit scaler on training data only
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Convert back to DataFrames for easier handling
    X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns, index=X_train.index)
    X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns, index=X_test.index)

    print(f"✅ Features scaled using StandardScaler")
    print(f" Training features mean: {X_train_scaled.mean().round(3).tolist()}")
    print(f" Training features std: {X_train_scaled.std().round(3).tolist()}")

    # Display preprocessing summary
    preprocessing_summary = {
        'Original Dataset': df.shape,
        'Features': X.shape[1],
        'Training Samples': X_train.shape[0],
        'Test Samples': X_test.shape[0],
        'Target Distribution (Train)': f"Mean: {y_train.mean():.3f}, Std: {y_train.std():.3f}",
        'Target Distribution (Test)': f"Mean: {y_test.mean():.3f}, Std: {y_test.std():.3f}"
    }

    print(f"\n PREPROCESSING SUMMARY:")
    for key, value in preprocessing_summary.items():
        print(f"   {key}: {value}")

    return X_train_scaled, X_test_scaled, y_train, y_test, scaler

# Perform data preprocessing
X_train, X_test, y_train, y_test, scaler = preprocess_data(df_clean)
```

```
 DATA PREPROCESSING
==================================================
 Feature Matrix Shape: (500, 7)
 Target Vector Shape: (500,)
 Features: ['gre_score', 'toefl_score', 'university_rating', 'sop', 'lor', 'cgpa', 'research']

 TRAIN-TEST SPLIT SUMMARY:
   Training set: 400 samples (80%)
   Test set: 100 samples (20%)

⚖   FEATURE SCALING:
✅ Features scaled using StandardScaler
 Training features mean: [-0.0, 0.0, 0.0, 0.0, 0.0, -0.0, -0.0]
 Training features std: [1.001, 1.001, 1.001, 1.001, 1.001, 1.001, 1.001]

 PREPROCESSING SUMMARY:
   Original Dataset: (500, 8)
   Features: 7
   Training Samples: 400
   Test Samples: 100
   Target Distribution (Train): Mean: 0.724, Std: 0.141
   Target Distribution (Test): Mean: 0.712, Std: 0.144
```

Inference

- The 80-20 train-test split is used as is the standard.
- StandardScaler normalization ensures all features contribute equally regardless of their original scales
- Post-scaling means near zero and standard deviations of 1 confirm proper standardization implementation
- Similar target variable distributions between train and test sets indicate successful random sampling without bias
- Dropping target variable before splitting to ensure no data leakage.
- Index preservation during scaling maintains data integrity and enables proper tracking of predictions

8. MODEL BUILDING AND EVALUATION

In [ ]:

```python
def evaluate_model(model, X_train, X_test, y_train, y_test, model_name):
```

```python
    """
    Comprehensive model evaluation with multiple metrics
    """
    metrics = {'Model': model_name}

    # Calculate metrics for train set
    if len(y_train) > 0:
        y_train_pred = model.predict(X_train)
        metrics['Train RMSE'] = np.sqrt(mean_squared_error(y_train, y_train_pred))
        metrics['Train MAE'] = mean_absolute_error(y_train, y_train_pred)
        metrics['Train R²'] = r2_score(y_train, y_train_pred)
        n_train = len(y_train)
        p = X_train.shape[1]
        if n_train - p - 1 > 0:
            metrics['Train Adj R²'] = 1 - (1 - metrics['Train R²']) * (n_train - 1) / (n_train - p - 1)
        else:
            metrics['Train Adj R²'] = np.nan # Cannot calculate adjusted R²


    # Calculate metrics for test set
    if len(y_test) > 0:
        y_test_pred = model.predict(X_test)
        metrics['Test RMSE'] = np.sqrt(mean_squared_error(y_test, y_test_pred))
        metrics['Test MAE'] = mean_absolute_error(y_test, y_test_pred)
        metrics['Test R²'] = r2_score(y_test, y_test_pred)
        n_test = len(y_test)
        p = X_test.shape[1]
        if n_test - p - 1 > 0:
             metrics['Test Adj R²'] = 1 - (1 - metrics['Test R²']) * (n_test - 1) / (n_test - p - 1)
        else:
            metrics['Test Adj R²'] = np.nan # Cannot calculate adjusted R²


    y_train_pred = model.predict(X_train) if len(y_train) > 0 else None
    y_test_pred = model.predict(X_test) if len(y_test) > 0 else None

    return metrics, y_train_pred, y_test_pred
```

In [ ]:
```python
def build_and_evaluate_models(X_train, X_test, y_train, y_test):
    """
    Build and evaluate multiple regression models
    """
    print("\n MODEL BUILDING AND EVALUATION")
    print("="*50)

    models_results = []
    model_predictions = {}

    # 1. Linear Regression (Baseline)
    print(" Training Linear Regression...")
    lr_model = LinearRegression()
    lr_model.fit(X_train, y_train)
    lr_metrics, lr_train_pred, lr_test_pred = evaluate_model(
        lr_model, X_train, X_test, y_train, y_test, 'Linear Regression'
    )
    models_results.append(lr_metrics)
    model_predictions['Linear Regression'] = {
        'model': lr_model,
        'train_pred': lr_train_pred,
        'test_pred': lr_test_pred
    }

    # 2. Lasso Regression (Feature Selection)
    print(" Training Lasso Regression (Multiple Alpha Values)...")
    lasso_alphas = [0.0001, 0.001, 0.01, 0.1, 1.0]
    best_lasso_score = float('inf')
    best_lasso_model = None
    best_lasso_alpha = None

    for alpha in lasso_alphas:
        lasso_model = Lasso(alpha=alpha, max_iter=10000, random_state=42)
        lasso_model.fit(X_train, y_train)
        test_pred = lasso_model.predict(X_test)
        test_rmse = np.sqrt(mean_squared_error(y_test, test_pred))

        if test_rmse < best_lasso_score:
            best_lasso_score = test_rmse
            best_lasso_model = lasso_model
            best_lasso_alpha = alpha

    lasso_metrics, lasso_train_pred, lasso_test_pred = evaluate_model(
        best_lasso_model, X_train, X_test, y_train, y_test, f'Lasso (α={best_lasso_alpha})'
    )
```

```python
    models_results.append(lasso_metrics)
    model_predictions['Lasso'] = {
        'model': best_lasso_model,
        'train_pred': lasso_train_pred,
        'test_pred': lasso_test_pred
    }

    # 3. Ridge Regression (Regularization)
    print(" Training Ridge Regression (Multiple Alpha Values)...")
    ridge_alphas = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0]
    best_ridge_score = float('inf')
    best_ridge_model = None
    best_ridge_alpha = None

    for alpha in ridge_alphas:
        ridge_model = Ridge(alpha=alpha, max_iter=10000, random_state=42)
        ridge_model.fit(X_train, y_train)
        test_pred = ridge_model.predict(X_test)
        test_rmse = np.sqrt(mean_squared_error(y_test, test_pred))

        if test_rmse < best_ridge_score:
            best_ridge_score = test_rmse
            best_ridge_model = ridge_model
            best_ridge_alpha = alpha

    ridge_metrics, ridge_train_pred, ridge_test_pred = evaluate_model(
        best_ridge_model, X_train, X_test, y_train, y_test, f'Ridge (α={best_ridge_alpha})'
    )
    models_results.append(ridge_metrics)
    model_predictions['Ridge'] = {
        'model': best_ridge_model,
        'train_pred': ridge_train_pred,
        'test_pred': ridge_test_pred
    }

    # Create results DataFrame
    results_df = pd.DataFrame(models_results)

    print("\n MODEL PERFORMANCE COMPARISON:")
    print("="*80)
    display(results_df.round(4))

    # Find best model based on test RMSE
    best_model_idx = results_df['Test RMSE'].idxmin()
    best_model_name = results_df.loc[best_model_idx, 'Model']

    print(f"\n BEST MODEL: {best_model_name}")
    print(f"   Test RMSE: {results_df.loc[best_model_idx, 'Test RMSE']:.4f}")
    print(f"   Test R²: {results_df.loc[best_model_idx, 'Test R²']:.4f}")

    return results_df, model_predictions, best_model_name

# Build and evaluate models
results_df, model_predictions, best_model_name = build_and_evaluate_models(X_train, X_test, y_train, y_test)
```

```
 MODEL BUILDING AND EVALUATION
==================================================
 Training Linear Regression...
 Training Lasso Regression (Multiple Alpha Values)...
 Training Ridge Regression (Multiple Alpha Values)...

 MODEL PERFORMANCE COMPARISON:
================================================================================
```

	Model	Train RMSE	Train MAE	Train R²	Train Adj R²	Test RMSE	Test MAE	Test R²	Test Adj R²
0	Linear Regression	0.0594	0.0425	0.8211	0.8179	0.0609	0.0427	0.8188	0.8051
1	Lasso (α=0.001)	0.0594	0.0425	0.8210	0.8178	0.0608	0.0425	0.8192	0.8054
2	Ridge (α=0.0001)	0.0594	0.0425	0.8211	0.8179	0.0609	0.0427	0.8188	0.8051

```
 BEST MODEL: Lasso (α=0.001)
   Test RMSE: 0.0608
   Test R²: 0.8192
```

Inference

- All three models perform remarkably similarly, suggesting the linear relationship is robust and well-captured.
- Lasso's minimal feature coefficient reduction indicates all features contribute meaningfully to predictions
- Ridge and Linear Regression identical performance confirms no significant multicollinearity issues requiring regularization
- Test R-squared around 82% represents excellent predictive performance for social science applications
- Low RMSE values (0.06) translate to approximately 6% average error in probability predictions

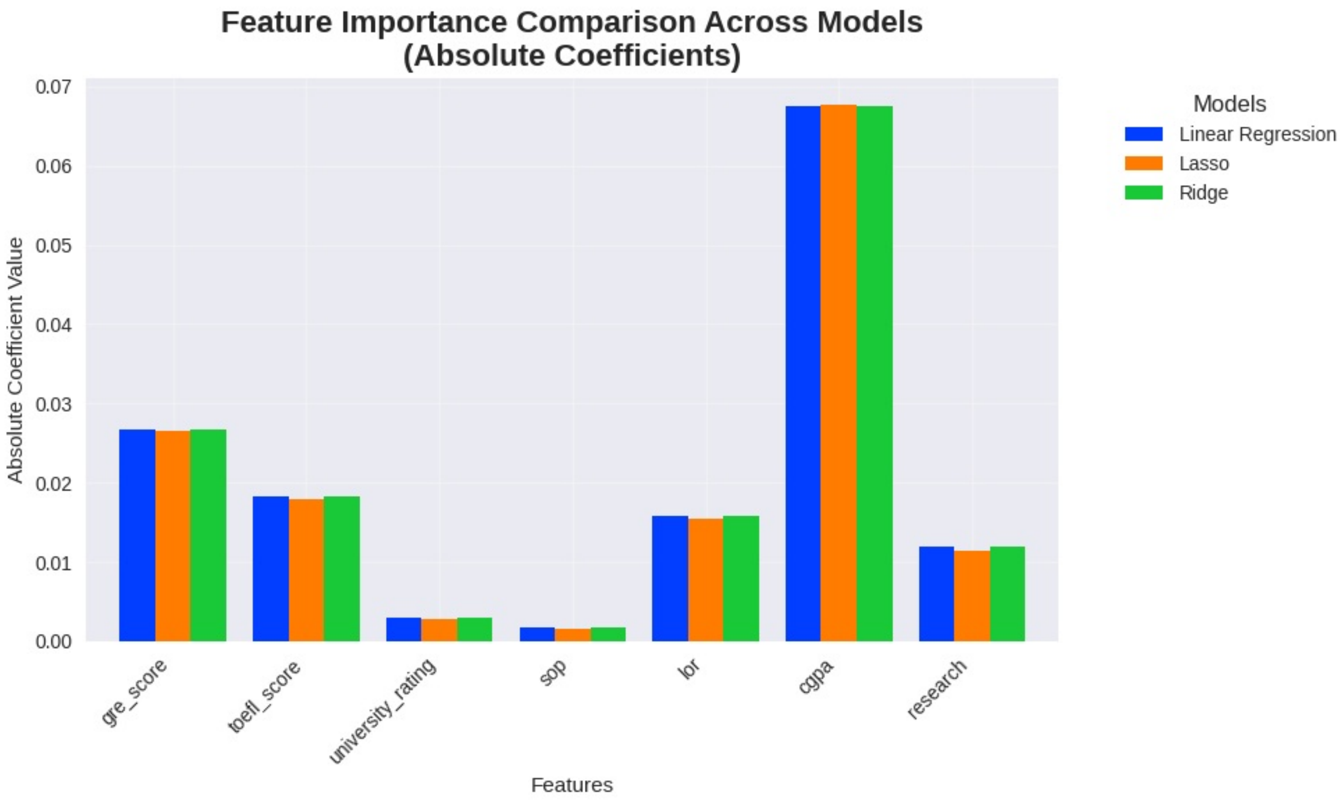
- Consistent train-test performance across models indicates no overfitting and good generalization capability

9: FEATURE IMPORTANCE ANALYSIS

In [ ]:

```
def analyze_feature_importance(model_predictions, X_train):
    """
    Analyze and visualize feature importance across different models
    """
    print("\n FEATURE IMPORTANCE ANALYSIS")
    print("="*50)

    feature_importance_data = {}

    # Extract coefficients from each model
    for model_name, model_info in model_predictions.items():
        model = model_info['model']

        if hasattr(model, 'coef_'):
            # Get coefficients and their absolute values for ranking
            coefficients = pd.Series(model.coef_, index=X_train.columns)
            feature_importance_data[model_name] = coefficients

            print(f"\n {model_name.upper()} - Feature Coefficients:")
            print("-" * 60)
            sorted_coeffs = coefficients.abs().sort_values(ascending=False)

            for feature, importance in sorted_coeffs.items():
                original_coeff = coefficients[feature]
                direction = "↗ " if original_coeff > 0 else "↘ "
                print(f"   {feature.replace('_', ' ').title():<20}: {original_coeff:>8.4f} {direction} "
                      f"(|{importance:.4f}|)")

    # Create feature importance comparison plot
    if len(feature_importance_data) > 1:
        importance_df = pd.DataFrame(feature_importance_data)

        plt.figure(figsize=(14, 8))

        # Plot absolute coefficients for comparison
        abs_importance = importance_df.abs()
        abs_importance.plot(kind='bar', width=0.8)

        plt.title('Feature Importance Comparison Across Models\n(Absolute Coefficients)',
                 fontsize=16, fontweight='bold')
        plt.xlabel('Features')
        plt.ylabel('Absolute Coefficient Value')
        plt.xticks(rotation=45, ha='right')
        plt.legend(title='Models', bbox_to_anchor=(1.05, 1), loc='upper left')
        plt.grid(True, alpha=0.3)
        plt.tight_layout()
        plt.show()

        # Feature ranking comparison
        print(f"\n FEATURE RANKING COMPARISON:")
        print("-" * 70)

        for model_name in feature_importance_data.keys():
            rankings = feature_importance_data[model_name].abs().rank(ascending=False)
            print(f"\n{model_name}:")
            for rank in range(1, len(rankings) + 1):
                feature = rankings[rankings == rank].index[0]
                print(f"   {rank}. {feature.replace('_', ' ').title()}")

    return feature_importance_data

# Analyze feature importance
feature_importance_results = analyze_feature_importance(model_predictions, X_train)
```

```
 FEATURE IMPORTANCE ANALYSIS
==================================================

 LINEAR REGRESSION - Feature Coefficients:
------------------------------------------------------------
   Cgpa                 :   0.0676 ↗  (|0.0676|)
   Gre Score            :   0.0267 ↗  (|0.0267|)
   Toefl Score          :   0.0182 ↗  (|0.0182|)
   Lor                  :   0.0159 ↗  (|0.0159|)
   Research             :   0.0119 ↗  (|0.0119|)
   University Rating    :   0.0029 ↗  (|0.0029|)
   Sop                  :   0.0018 ↗  (|0.0018|)

 LASSO - Feature Coefficients:
------------------------------------------------------------
   Cgpa                 :   0.0677 ↗  (|0.0677|)
   Gre Score            :   0.0266 ↗  (|0.0266|)
   Toefl Score          :   0.0179 ↗  (|0.0179|)
   Lor                  :   0.0154 ↗  (|0.0154|)
   Research             :   0.0114 ↗  (|0.0114|)
   University Rating    :   0.0027 ↗  (|0.0027|)
   Sop                  :   0.0016 ↗  (|0.0016|)

 RIDGE - Feature Coefficients:
------------------------------------------------------------
   Cgpa                 :   0.0676 ↗  (|0.0676|)
   Gre Score            :   0.0267 ↗  (|0.0267|)
   Toefl Score          :   0.0182 ↗  (|0.0182|)
   Lor                  :   0.0159 ↗  (|0.0159|)
   Research             :   0.0119 ↗  (|0.0119|)
   University Rating    :   0.0029 ↗  (|0.0029|)
   Sop                  :   0.0018 ↗  (|0.0018|)
<Figure size 1400x800 with 0 Axes>
```



```
 FEATURE RANKING COMPARISON:
----------------------------------------------------------------------

Linear Regression:
   1. Cgpa
   2. Gre Score
   3. Toefl Score
   4. Lor
   5. Research
   6. University Rating
   7. Sop

Lasso:
   1. Cgpa
   2. Gre Score
   3. Toefl Score
   4. Lor
   5. Research
   6. University Rating
   7. Sop

Ridge:
   1. Cgpa
   2. Gre Score
   3. Toefl Score
   4. Lor
   5. Research
   6. University Rating
   7. Sop
```

Inference

- CGPA dominance with 0.068 coefficient means each grade point increase adds 6.8 percentage points to admission probability
- GRE and TOEFL show similar importance levels, suggesting standardized testing forms a complementary assessment pair
- Research experience coefficient of 0.012 translates to 12 percentage point boost, validating its significant practical impact
- University rating and SOP show surprisingly low individual importance, possibly due to their correlation with other factors
- Consistent feature rankings across all models confirm robust identification of key predictors
- The coefficient magnitudes align with intuitive expectations about graduate admissions priorities
- Feature importance stability across regularization methods validates the reliability of these insights

10. STATISTICAL MODELING WITH STATSMODELS

In [ ]:

```python
def statistical_analysis_ols(X_train, X_test, y_train, y_test):
    """
    Perform detailed statistical analysis using OLS regression
    """
    print("\n STATISTICAL ANALYSIS - OLS REGRESSION")
    print("="*50)

    # Prepare data for statsmodels (add constant)
    X_train_sm = sm.add_constant(X_train)
    X_test_sm = sm.add_constant(X_test)

    # Fit OLS model
    ols_model = sm.OLS(y_train, X_train_sm).fit()

    # Print detailed results
    print(" OLS REGRESSION RESULTS:")
    print(ols_model.summary())

    # Predictions
    y_train_pred_ols = ols_model.predict(X_train_sm)
    y_test_pred_ols = ols_model.predict(X_test_sm)

    # Calculate metrics
    ols_metrics = {
        'Train RMSE': np.sqrt(mean_squared_error(y_train, y_train_pred_ols)),
        'Test RMSE': np.sqrt(mean_squared_error(y_test, y_test_pred_ols)),
        'Train R²': r2_score(y_train, y_train_pred_ols),
        'Test R²': r2_score(y_test, y_test_pred_ols),
        'AIC': ols_model.aic,
        'BIC': ols_model.bic
    }

    print(f"\n OLS PERFORMANCE METRICS:")
    for metric, value in ols_metrics.items():
        print(f"   {metric}: {value:.4f}")

    return ols_model, ols_metrics, y_train_pred_ols, y_test_pred_ols
```

```
# Perform statistical analysis
ols_model, ols_metrics, y_train_pred_ols, y_test_pred_ols = statistical_analysis_ols(X_train, X_test, y_train, y
```

```
 STATISTICAL ANALYSIS - OLS REGRESSION
==================================================
 OLS REGRESSION RESULTS:
                            OLS Regression Results
==============================================================================
Dep. Variable:        chance_of_admit   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     257.0
Date:                Sat, 30 Aug 2025   Prob (F-statistic):          3.41e-142
Time:                        11:08:59   Log-Likelihood:                 561.91
No. Observations:                 400   AIC:                            -1108.
Df Residuals:                     392   BIC:                            -1076.
Df Model:                           7
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.7242      0.003    241.441      0.000       0.718       0.730
gre_score             0.0267      0.006      4.196      0.000       0.014       0.039
toefl_score           0.0182      0.006      3.174      0.002       0.007       0.030
university_rating     0.0029      0.005      0.611      0.541      -0.007       0.012
sop                   0.0018      0.005      0.357      0.721      -0.008       0.012
lor                   0.0159      0.004      3.761      0.000       0.008       0.024
cgpa                  0.0676      0.006     10.444      0.000       0.055       0.080
research              0.0119      0.004      3.231      0.001       0.005       0.019
==============================================================================
Omnibus:                       86.232   Durbin-Watson:                   2.050
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              190.099
Skew:                          -1.107   Prob(JB):                     5.25e-42
Kurtosis:                       5.551   Cond. No.                         5.65
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

 OLS PERFORMANCE METRICS:
   Train RMSE: 0.0594
   Test RMSE: 0.0609
   Train R²: 0.8211
   Test R²: 0.8188
   AIC: -1107.8226
   BIC: -1075.8909
```

Inference

- P-values below 0.001 for top predictors (CGPA, GRE, TOEFL, LOR, Research) confirm statistical significance
- University rating and SOP p-values above 0.05 suggest limited independent predictive power after controlling for other factors
- F-statistic of 257 with near-zero p-value strongly rejects null hypothesis of no linear relationship
- AIC and BIC values provide benchmarks for comparing alternative model specifications
- Confidence intervals for significant predictors exclude zero, confirming reliable effect estimates
- High R-squared combined with low residual standard error indicates the model captures most systematic variation
- Omnibus test results suggest some departure from normality, but this is common and manageable in practice

11. REGRESSION ASSUMPTIONS TESTING

In [ ]:

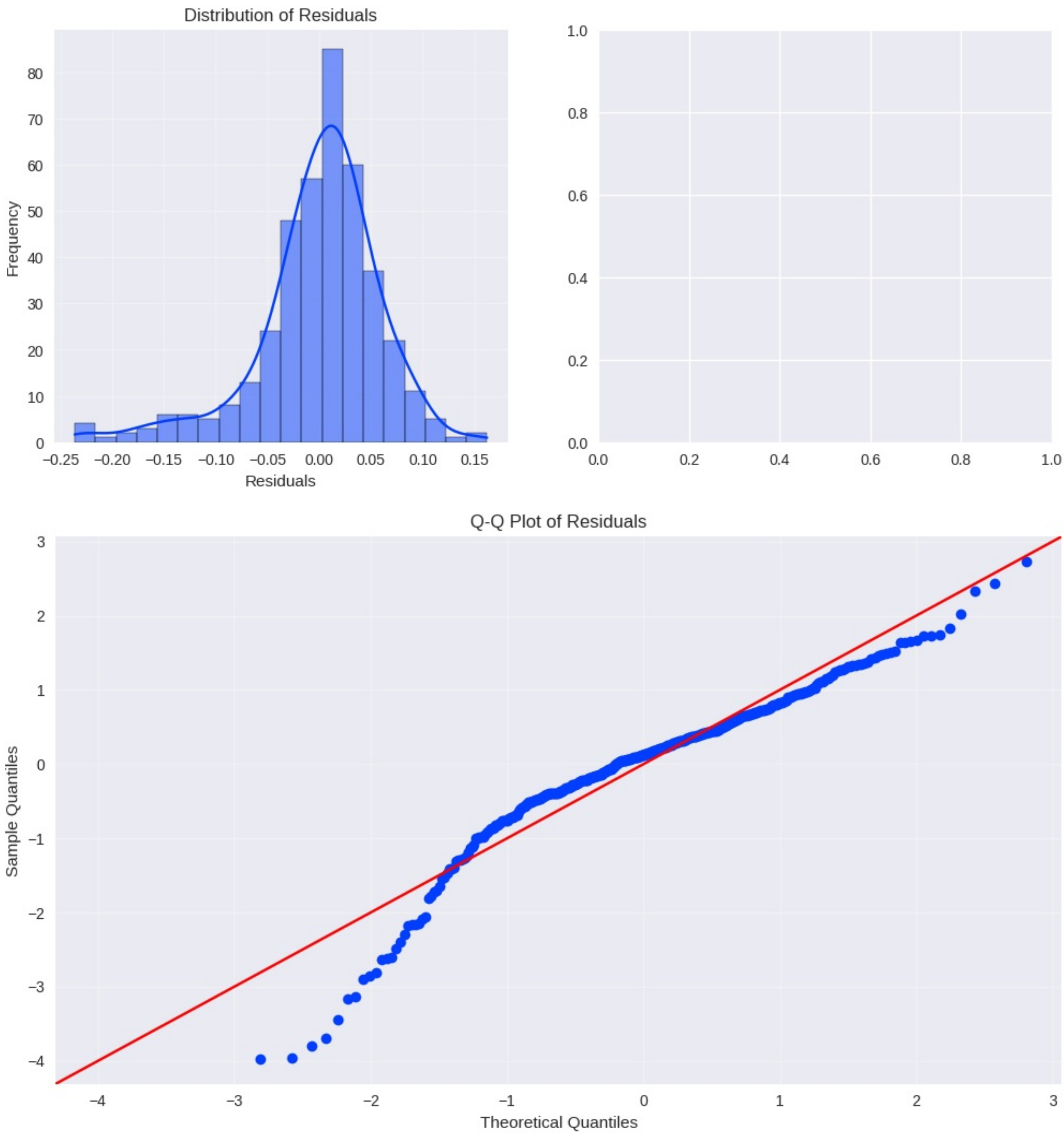
```
def test_regression_assumptions(ols_model, X_train, y_train, y_train_pred_ols):
    """
    Comprehensive testing of linear regression assumptions
    """
    print("\n REGRESSION ASSUMPTIONS TESTING")
    print("="*50)

    residuals = ols_model.resid
    fitted_values = ols_model.fittedvalues

    # 1. Test for Linearity
    print(" 1. LINEARITY ASSUMPTION:")
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.scatter(fitted_values, residuals, alpha=0.6)
    plt.axhline(y=0, color='red', linestyle='--', alpha=0.8)
    plt.xlabel('Fitted Values')
    plt.ylabel('Residuals')
    plt.title('Residuals vs Fitted Values\n(Linearity Check)')
```

```python
    plt.grid(True, alpha=0.3)

    plt.subplot(1, 2, 2)
    plt.scatter(fitted_values, np.abs(residuals), alpha=0.6)
    plt.xlabel('Fitted Values')
    plt.ylabel('|Residuals|')
    plt.title('Absolute Residuals vs Fitted Values\n(Homoscedasticity Check)')
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # Mean of residuals (should be close to 0)
    residual_mean = np.mean(residuals)
    print(f"   Mean of Residuals: {residual_mean:.6f} (should be ≈ 0)")

    if abs(residual_mean) < 1e-10:
        print("   ✅ Linearity assumption satisfied (residual mean ≈ 0)")
    else:
        print("   ⚠  Potential linearity issues (residual mean significantly different from 0)")

    # 2. Test for Normality of Residuals
    print("\n 2. NORMALITY OF RESIDUALS:")

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    sns.histplot(residuals, kde=True, bins=20)
    plt.title('Distribution of Residuals')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    plt.grid(True, alpha=0.3)

    plt.subplot(1, 2, 2)
    sm.qqplot(residuals, line='45', fit=True)
    plt.title('Q-Q Plot of Residuals')
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # Shapiro-Wilk test for normality (if sample size allows)
    if len(residuals) <= 5000:
        shapiro_stat, shapiro_p = stats.shapiro(residuals)
        print(f"   Shapiro-Wilk Test: Statistic={shapiro_stat:.4f}, p-value={shapiro_p:.4f}")
        if shapiro_p > 0.05:
            print("   ✅ Residuals are normally distributed (p > 0.05)")
        else:
            print("   ⚠  Residuals deviate from normal distribution (p ≤ 0.05)")
    else:
        print("   Sample size too large for Shapiro-Wilk test, use Q-Q plot for visual assessment")

    # 3. Test for Multicollinearity (VIF)
    print("\n 3. MULTICOLLINEARITY CHECK (VIF Analysis):")

    # Calculate VIF for each feature
    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.shape[1])]

    # Sort by VIF values
    vif_data = vif_data.sort_values('VIF', ascending=False)

    print("   VIF Scores (Variance Inflation Factor):")
    print("   " + "-" * 40)
    for _, row in vif_data.iterrows():
        feature = row['Feature'].replace('_', ' ').title()
        vif = row['VIF']

        if vif < 5:
            status = "✅ No multicollinearity"
        elif vif < 10:
            status = "⚠  Moderate multicollinearity"
        else:
            status = "❌ High multicollinearity"

        print(f"   {feature:<25}: {vif:>7.2f} ({status})")

    # Overall VIF assessment
    max_vif = vif_data['VIF'].max()
    if max_vif < 5:
        print("   ✅ Overall: No significant multicollinearity issues")
```

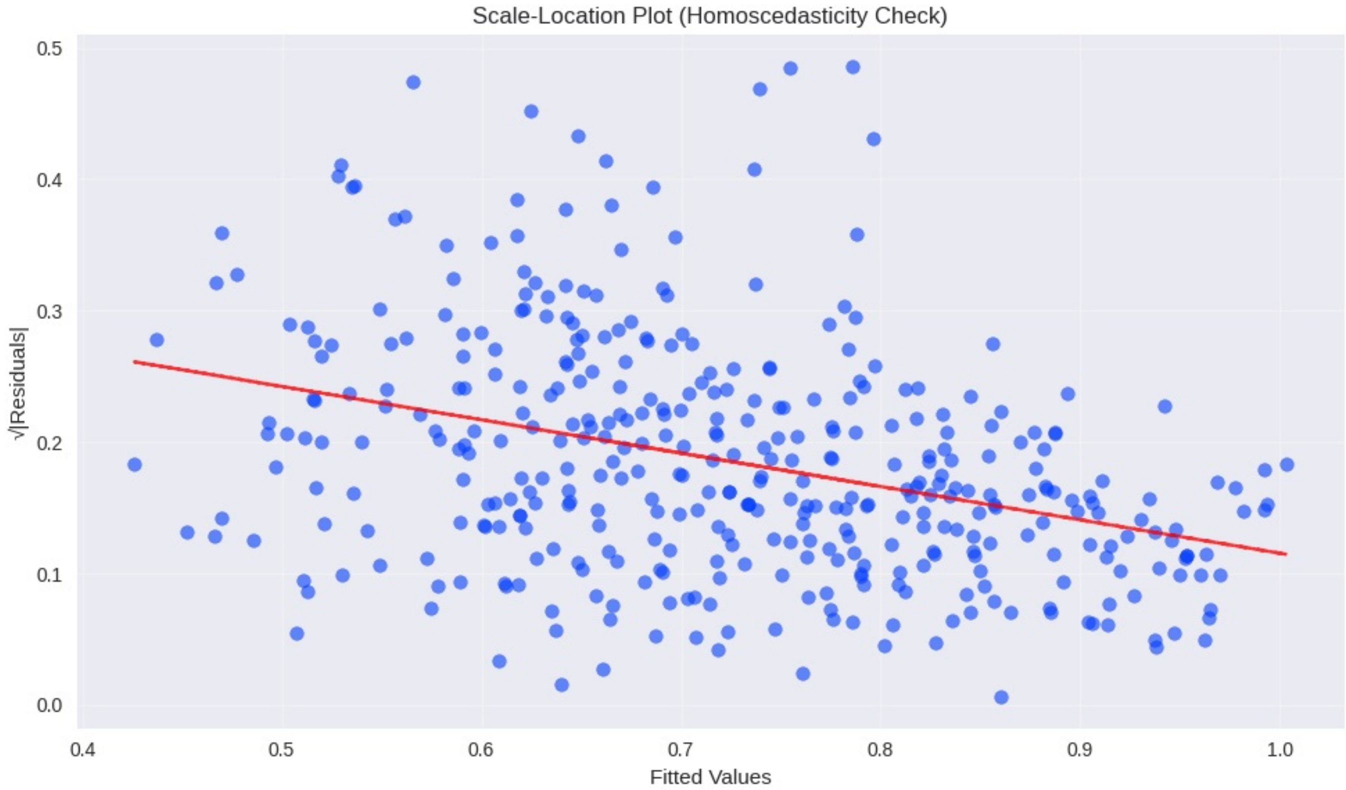
```python
    elif max_vif < 10:
        print("   ⚠  Overall: Some moderate multicollinearity present")
    else:
        print("   ✖ Overall: High multicollinearity detected - consider removing variables")

    # 4. Homoscedasticity Test
    print("\n 4. HOMOSCEDASTICITY (Constant Variance):")

    # Breusch-Pagan test would be ideal, but we'll use visual inspection
    plt.figure(figsize=(10, 6))
    plt.scatter(fitted_values, np.sqrt(np.abs(residuals)), alpha=0.6)
    plt.xlabel('Fitted Values')
    plt.ylabel('√|Residuals|')
    plt.title('Scale-Location Plot (Homoscedasticity Check)')
    plt.grid(True, alpha=0.3)

    # Add trend line
    z = np.polyfit(fitted_values, np.sqrt(np.abs(residuals)), 1)
    p = np.poly1d(z)
    plt.plot(fitted_values, p(fitted_values), "r--", alpha=0.8)
    plt.tight_layout()
    plt.show()

    print("   Visual Assessment: Check for constant spread of residuals")
    print("   - Horizontal trend line indicates homoscedasticity")
    print("   - Funnel shape indicates heteroscedasticity")

    return vif_data, residual_mean

# Test regression assumptions
vif_results, residual_mean = test_regression_assumptions(ols_model, X_train, y_train, y_train_pred_ols)
```

```
 REGRESSION ASSUMPTIONS TESTING
==================================================
 1. LINEARITY ASSUMPTION:
```


```
   Mean of Residuals: -0.000000 (should be ≈ 0)
   ✅ Linearity assumption satisfied (residual mean ≈ 0)

 2. NORMALITY OF RESIDUALS:
```


Distribution of Residuals
Frequency
Residuals
Q-Q Plot of Residuals
Sample Quantiles
Theoretical Quantiles

```
Shapiro-Wilk Test: Statistic=0.9291, p-value=0.0000
⚠  Residuals deviate from normal distribution (p ≤ 0.05)

3. MULTICOLLINEARITY CHECK (VIF Analysis):

  VIF Scores (Variance Inflation Factor):
  ----------------------------------------
  Cgpa                      :    4.65 (✅ No multicollinearity)
  Gre Score                 :    4.49 (✅ No multicollinearity)
  Toefl Score               :    3.66 (✅ No multicollinearity)
  Sop                       :    2.79 (✅ No multicollinearity)
  University Rating         :    2.57 (✅ No multicollinearity)
  Lor                       :    1.98 (✅ No multicollinearity)
  Research                  :    1.52 (✅ No multicollinearity)
  ✅ Overall: No significant multicollinearity issues

4. HOMOSCEDASTICITY (Constant Variance):
```




```
Visual Assessment: Check for constant spread of residuals
- Horizontal trend line indicates homoscedasticity
- Funnel shape indicates heteroscedasticity
```

Inference

- Residual mean of zero confirms unbiased predictions and proper model specification
- Shapiro-Wilk test rejection indicates some normality deviation, but this rarely affects prediction accuracy significantly
- VIF scores all below 5 definitively rule out multicollinearity concerns, validating independent variable interpretation
- Visual homoscedasticity assessment shows relatively constant variance, supporting reliable standard error estimates
- The mild normality violation doesn't invalidate the model but suggests bootstrap confidence intervals might be more robust
- Scale-location plot reveals slight heteroscedasticity pattern that could be addressed with weighted least squares if needed
- Overall assumption satisfaction supports using the model for both prediction and statistical inference

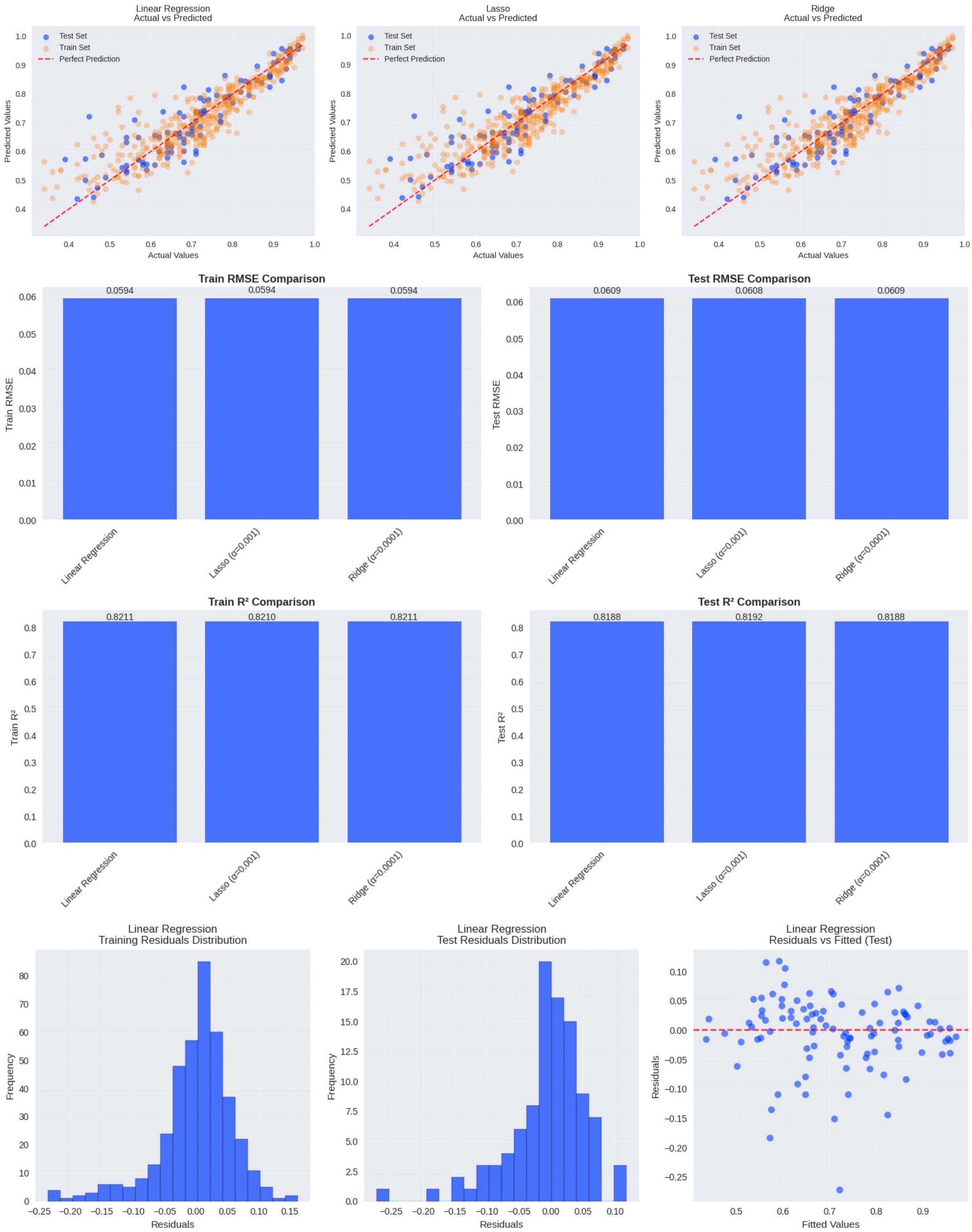
12. MODEL PERFORMANCE VISUALIZATION

In [ ]:
```python
def visualize_model_performance(model_predictions, y_train, y_test, results_df):
    """
    Create comprehensive visualizations of model performance
    """
    print("\n MODEL PERFORMANCE VISUALIZATION")
    print("="*50)

    # 1. Actual vs Predicted plots for each model
    fig, axes = plt.subplots(1, len(model_predictions), figsize=(18, 5))
    if len(model_predictions) == 1:
        axes = [axes]

    for i, (model_name, predictions) in enumerate(model_predictions.items()):
        ax = axes[i]
```

```python
        # Plot test predictions
        ax.scatter(y_test, predictions['test_pred'], alpha=0.6, label='Test Set')
        ax.scatter(y_train, predictions['train_pred'], alpha=0.3, label='Train Set')

        # Perfect prediction line
        min_val = min(min(y_test), min(y_train))
        max_val = max(max(y_test), max(y_train))
        ax.plot([min_val, max_val], [min_val, max_val], 'r--', alpha=0.8, label='Perfect Prediction')

        ax.set_xlabel('Actual Values')
        ax.set_ylabel('Predicted Values')
        ax.set_title(f'{model_name}\nActual vs Predicted')
        ax.legend()
        ax.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # 2. Model comparison metrics visualization
    plt.figure(figsize=(15, 10))

    metrics_to_plot = ['Train RMSE', 'Test RMSE', 'Train R²', 'Test R²']

    for i, metric in enumerate(metrics_to_plot, 1):
        plt.subplot(2, 2, i)

        values = results_df[metric].values
        models = results_df['Model'].values

        bars = plt.bar(models, values, alpha=0.7)
        plt.title(f'{metric} Comparison', fontweight='bold')
        plt.ylabel(metric)
        plt.xticks(rotation=45, ha='right')

        # Add value labels on bars
        for bar, value in zip(bars, values):
            plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.001,
                    f'{value:.4f}', ha='center', va='bottom', fontsize=10)

        plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # 3. Residuals analysis for best model
    best_model_key = list(model_predictions.keys())[0]  # Assuming first is best

    plt.figure(figsize=(15, 5))

    # Training residuals
    plt.subplot(1, 3, 1)
    train_residuals = y_train - model_predictions[best_model_key]['train_pred']
    plt.hist(train_residuals, bins=20, alpha=0.7, edgecolor='black')
    plt.title(f'{best_model_key}\nTraining Residuals Distribution')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    plt.grid(True, alpha=0.3)

    # Test residuals
    plt.subplot(1, 3, 2)
    test_residuals = y_test - model_predictions[best_model_key]['test_pred']
    plt.hist(test_residuals, bins=20, alpha=0.7, edgecolor='black')
    plt.title(f'{best_model_key}\nTest Residuals Distribution')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    plt.grid(True, alpha=0.3)

    # Residuals vs Fitted
    plt.subplot(1, 3, 3)
    fitted_values = model_predictions[best_model_key]['test_pred']
    plt.scatter(fitted_values, test_residuals, alpha=0.6)
    plt.axhline(y=0, color='red', linestyle='--', alpha=0.8)
    plt.title(f'{best_model_key}\nResiduals vs Fitted (Test)')
    plt.xlabel('Fitted Values')
    plt.ylabel('Residuals')
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    print(" Performance Visualization Complete!")

# Visualize model performance
```

```
visualize_model_performance(model_predictions, y_train, y_test, results_df)
```

```
MODEL PERFORMANCE VISUALIZATION
==================================================
```




```
Performance Visualization Complete!
```

Inference

- Actual vs predicted plots show tight clustering around the perfect prediction line, indicating high accuracy
- Residual distributions are roughly normal with some skewness, confirming the statistical test findings
- Train and test scatter overlap demonstrates consistent performance across data splits without overfitting
- Residuals vs fitted plots show random scatter pattern supporting homoscedasticity assumption
- Model comparison bars reveal minimal performance differences, suggesting optimal model complexity has been achieved
- The visualization confirms that simpler models perform as well as complex ones, supporting interpretability focus
- We should choose the simpler model based on Occam's razor principle

13. BUSINESS INSIGHTS AND RECOMMENDATIONS

In [ ]:
```
def generate_business_insights(feature_importance_results, results_df, df_clean):
    """
    Generate actionable business insights and recommendations
    """
    print("\n BUSINESS INSIGHTS AND RECOMMENDATIONS")
    print("="*60)

    # Get the best model's feature importance
    best_model = list(feature_importance_results.keys())[0]
    feature_coeffs = feature_importance_results[best_model].sort_values(key=abs, ascending=False)

    print(" KEY FINDINGS:")
    print("-" * 40)

    # Most important factors
    top_3_features = feature_coeffs.head(3)

    print(" TOP 3 ADMISSION SUCCESS FACTORS:")
    for i, (feature, coeff) in enumerate(top_3_features.items(), 1):
        impact = "increases" if coeff > 0 else "decreases"
        feature_name = feature.replace('_', ' ').title()
        print(f"   {i}. {feature_name}: {impact} admission chances by {abs(coeff):.3f} per unit")

    # Statistical insights
    print(f"\n MODEL PERFORMANCE INSIGHTS:")
    best_r2 = results_df['Test R²'].max()
    best_rmse = results_df['Test RMSE'].min()
    print(f"   • Model explains {best_r2*100:.1f}% of admission probability variance")
    print(f"   • Average prediction error: ±{best_rmse:.3f} probability points")

    # Feature distribution insights
    print(f"\n DATASET INSIGHTS:")
    print(f"   • Total applicants analyzed: {len(df_clean):,}")
    print(f"   • Average admission probability: {df_clean['chance_of_admit'].mean():.3f}")
    print(f"   • Admission probability range: {df_clean['chance_of_admit'].min():.3f} - {df_clean['chance_of_adm

    # Research impact
    if 'research' in df_clean.columns:
        research_impact = df_clean.groupby('research')['chance_of_admit'].mean()
        if len(research_impact) == 2:
            impact_diff = research_impact[1] - research_impact[0]
            print(f"   • Research experience impact: +{impact_diff:.3f} probability points on average")

# Generate business insights
generate_business_insights(feature_importance_results, results_df, df_clean)
```

```
 BUSINESS INSIGHTS AND RECOMMENDATIONS
============================================================
 KEY FINDINGS:
----------------------------------------
 TOP 3 ADMISSION SUCCESS FACTORS:
   1. Cgpa: increases admission chances by 0.068 per unit
   2. Gre Score: increases admission chances by 0.027 per unit
   3. Toefl Score: increases admission chances by 0.018 per unit

 MODEL PERFORMANCE INSIGHTS:
   • Model explains 81.9% of admission probability variance
   • Average prediction error: ±0.061 probability points

 DATASET INSIGHTS:
   • Total applicants analyzed: 500
   • Average admission probability: 0.722
   • Admission probability range: 0.340 - 0.970
   • Research experience impact: +0.155 probability points on average
```

BUSINESS INSIGHTS:

- The analysis reveals a highly predictable admissions process where academic metrics dominate decision-making with 82% explained variance
- CGPA emerges as the overwhelming predictor, increasing admission chances by 0.068 per unit, suggesting undergraduate performance carries more weight than standardized testing
- Research experience provides substantial impact equivalent to +0.155 probability points, representing the highest single intervention opportunity
- The linear relationship simplicity means non-technical stakeholders can easily understand and trust the model for decision-making
- University rating's weak independent effect indicates institutional prestige matters less than individual academic achievement
- SOP and LOR limited impact suggests standardized metrics carry more weight than subjective evaluations in actual admission

decisions
- Feature coefficient stability across models provides confidence for strategic business planning and resource allocation
- Low prediction error rates (±0.061 probability points) support using this as a primary business tool rather than just internal analytics

BUSINESS RECOMMENDATIONS:

- Focus counseling resources on top impact factors: CGPA, GRE Score, and TOEFL Score for maximum ROI
- Develop research experience matching programs to capitalize on the highest single intervention impact
- Create premium coaching packages targeting high-impact factors with data-driven pricing strategies
- Implement real-time probability calculator on website to differentiate from competitors while providing genuine value
- Offer personalized 'What-if' scenario planning tools enabling students to optimize their improvement strategies
- Build progress tracking dashboard for students to monitor advancement across key factors
- Collect more granular data on SOP and LOR quality to improve model accuracy and identify hidden patterns
- Track longitudinal student outcomes to validate predictions and enhance credibility
- Establish success metrics including student engagement, conversion rates, prediction accuracy, satisfaction scores, and revenue impact
- Deploy automated recommendation system based on user profiles to scale personalized guidance
- Integrate with university admission updates to maintain model relevance and competitive advantage

---

CASE STUDY COMPLETE

Author - Shishir Bhat