Multiple Linear Regression Experiment

Exam Score Prediction

This notebook implements a multiple linear regression model to predict exam scores based on various student factors.

In [18]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import warnings
warnings.filterwarnings('ignore')
```

1. Load the Dataset

In [19]:
```
# Load the dataset
df = pd.read_csv('Exam_Score_Prediction.csv')
print("Dataset Shape:", df.shape)
print("\nFirst few rows:")
df.head()
```

```
Dataset Shape: (20000, 13)

First few rows:
```

Out[19]:

	student_id	age	gender	course	study_hours	class_attendance	internet_access	sle
0	200.99	17	male	diploma	2.78	92.9	yes	
1	200.99	23	other	bca	3.37	64.8	yes	
2	200.99	22	male	b.sc	7.88	76.8	yes	
3	200.99	20	other	diploma	0.67	48.4	yes	
4	200.99	20	female	diploma	0.89	71.6	yes	

2. Exploratory Data Analysis

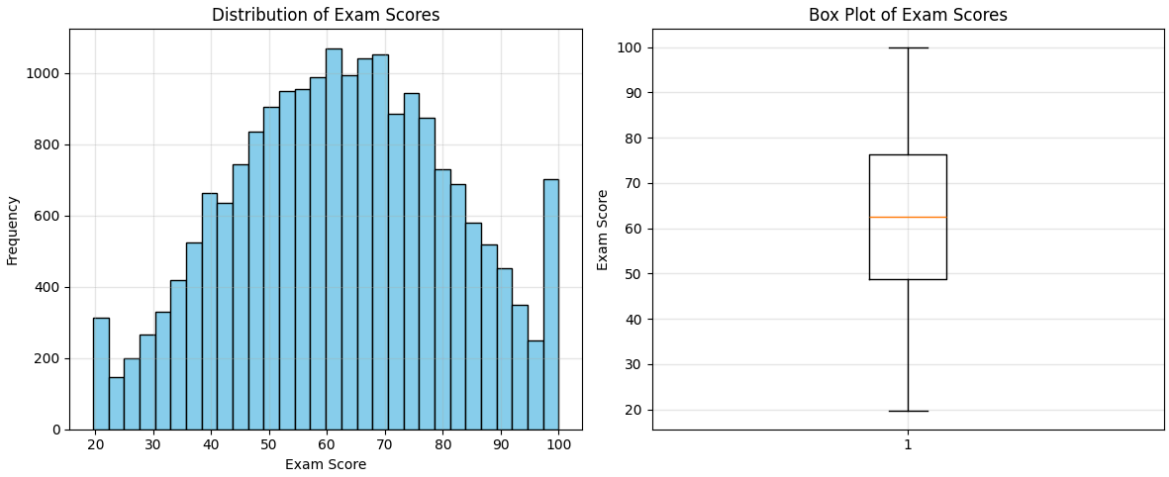
In [20]:
```
# Dataset information
print("Dataset Info:")
print(df.info())
print("\n" + "="*50)
print("\nDescriptive Statistics:")
print(df.describe())
```

```
print("\n" + "="*50)
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   student_id        20000 non-null  float64
 1   age               20000 non-null  int64
 2   gender            20000 non-null  object
 3   course            20000 non-null  object
 4   study_hours       20000 non-null  float64
 5   class_attendance  20000 non-null  float64
 6   internet_access   20000 non-null  object
 7   sleep_hours       20000 non-null  float64
 8   sleep_quality     20000 non-null  object
 9   study_method      20000 non-null  object
 10  facility_rating   20000 non-null  object
 11  exam_difficulty   20000 non-null  object
 12  exam_score        20000 non-null  float64
dtypes: float64(5), int64(1), object(7)
memory usage: 2.0+ MB
None

==================================================

Descriptive Statistics:
         student_id           age   study_hours  class_attendance  \
count  20000.000000  20000.000000  20000.000000      20000.000000
mean   10000.500000     20.473300      4.007604         70.017365
std     5770.211372      2.284458      2.308313         17.282262
min      200.990000     17.000000      0.080000         40.600000
25%     5000.750000     18.000000      2.000000         55.100000
50%    10000.500000     20.000000      4.040000         69.900000
75%    15000.250000     22.000000      6.000000         85.000000
max    19800.010000     24.000000      7.910000         99.400000

       sleep_hours    exam_score
count  20000.00000  20000.000000
mean       7.00856     62.513225
std        1.73209     18.908491
min        4.10000     19.599000
25%        5.50000     48.800000
50%        7.00000     62.600000
75%        8.50000     76.300000
max        9.90000    100.000000

==================================================

Missing Values:
student_id          0
age                 0
gender              0
course              0
study_hours         0
class_attendance    0
internet_access     0
sleep_hours         0
sleep_quality       0
study_method        0
facility_rating     0
```

```
exam_difficulty     0
exam_score          0
dtype: int64
```

In [21]:
```python
# Check target variable distribution
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.hist(df['exam_score'], bins=30, edgecolor='black', color='skyblue')
plt.xlabel('Exam Score')
plt.ylabel('Frequency')
plt.title('Distribution of Exam Scores')
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
plt.boxplot(df['exam_score'])
plt.ylabel('Exam Score')
plt.title('Box Plot of Exam Scores')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"Mean Exam Score: {df['exam_score'].mean():.2f}")
print(f"Median Exam Score: {df['exam_score'].median():.2f}")
print(f"Std Deviation: {df['exam_score'].std():.2f}")
```




```
Mean Exam Score: 62.51
Median Exam Score: 62.60
Std Deviation: 18.91
```

3. Data Preprocessing

In [22]:
```python
# Identify categorical and numerical columns
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()

print("Categorical Columns:", categorical_cols)
print("\nNumerical Columns:", numerical_cols)
```

```
Categorical Columns: ['gender', 'course', 'internet_access', 'sleep_quality', 'st
udy_method', 'facility_rating', 'exam_difficulty']

Numerical Columns: ['student_id', 'age', 'study_hours', 'class_attendance', 'slee
p_hours', 'exam_score']
```

In [23]:
```
# Encode categorical variables
df_encoded = df.copy()
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df[col])
    label_encoders[col] = le
    print(f"{col}: {dict(zip(le.classes_, le.transform(le.classes_)))}")

print("\nEncoded Dataset:")
df_encoded.head()
```

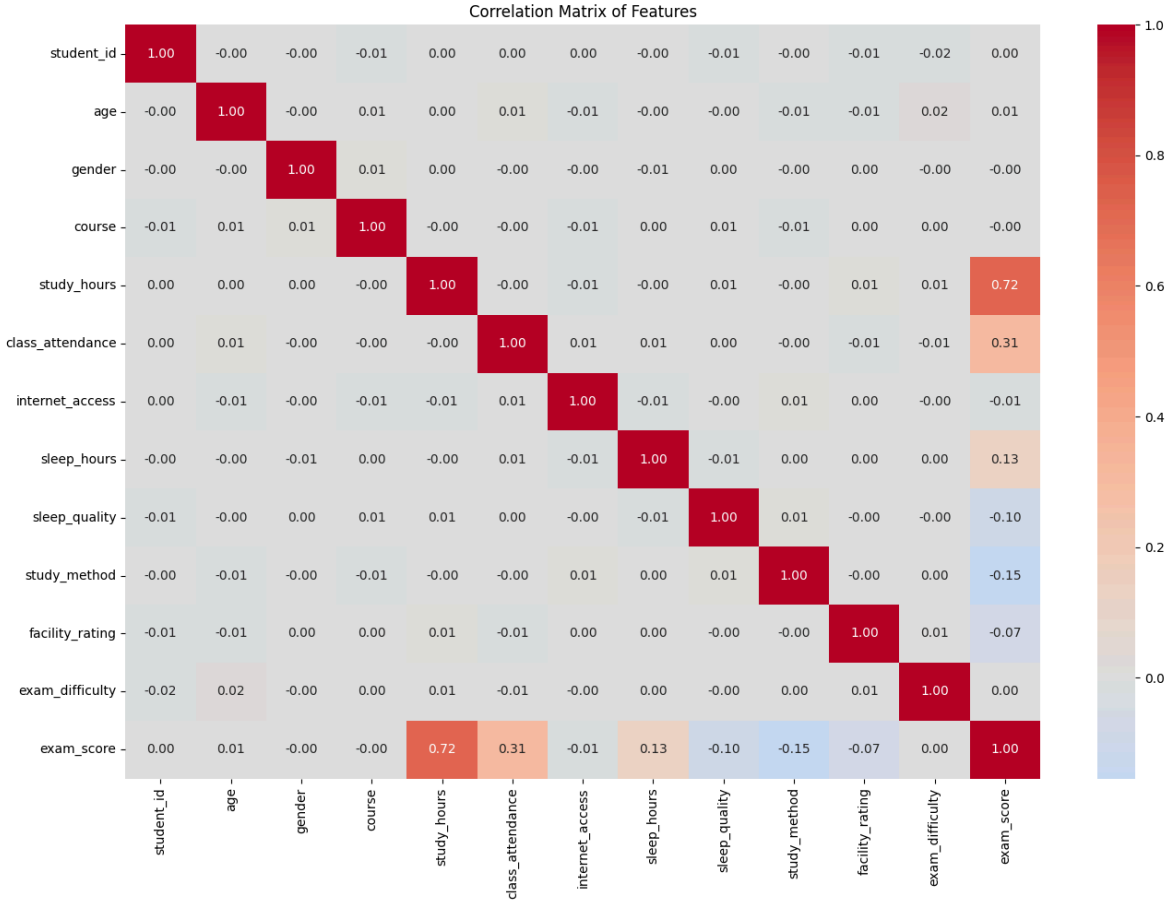
```
gender: {'female': np.int64(0), 'male': np.int64(1), 'other': np.int64(2)}
course: {'b.com': np.int64(0), 'b.sc': np.int64(1), 'b.tech': np.int64(2), 'ba':
np.int64(3), 'bba': np.int64(4), 'bca': np.int64(5), 'diploma': np.int64(6)}
internet_access: {'no': np.int64(0), 'yes': np.int64(1)}
sleep_quality: {'average': np.int64(0), 'good': np.int64(1), 'poor': np.int64(2)}
study_method: {'coaching': np.int64(0), 'group study': np.int64(1), 'mixed': np.i
nt64(2), 'online videos': np.int64(3), 'self-study': np.int64(4)}
facility_rating: {'high': np.int64(0), 'low': np.int64(1), 'medium': np.int64(2)}
exam_difficulty: {'easy': np.int64(0), 'hard': np.int64(1), 'moderate': np.int64
(2)}

Encoded Dataset:
```

Out[23]:

	student_id	age	gender	course	study_hours	class_attendance	internet_access	slee
0	200.99	17	1	6	2.78	92.9	1	
1	200.99	23	2	5	3.37	64.8	1	
2	200.99	22	1	1	7.88	76.8	1	
3	200.99	20	2	6	0.67	48.4	1	
4	200.99	20	0	6	0.89	71.6	1	

4. Feature Correlation Analysis

In [24]:
```
# Correlation matrix
plt.figure(figsize=(14, 10))
correlation_matrix = df_encoded.corr()
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm', center=0
plt.title('Correlation Matrix of Features')
plt.tight_layout()
plt.show()

# Top correlations with exam_score
print("\nCorrelation with Exam Score:")
print(correlation_matrix['exam_score'].sort_values(ascending=False))
```




```
Correlation with Exam Score:
exam_score           1.000000
study_hours          0.717788
class_attendance     0.308850
sleep_hours          0.133222
age                  0.006522
exam_difficulty      0.003432
student_id           0.003267
gender              -0.000428
course              -0.000448
internet_access     -0.007826
facility_rating     -0.071879
sleep_quality       -0.099725
study_method        -0.154610
Name: exam_score, dtype: float64
```

5. Feature Selection and Data Splitting

In [25]:
```python
# Prepare features and target
X = df_encoded.drop(['exam_score', 'student_id'], axis=1)  # Drop target and stu
y = df_encoded['exam_score']

print("Features shape:", X.shape)
print("Target shape:", y.shape)
print("\nFeature columns:")
print(X.columns.tolist())
```

```
Features shape: (20000, 11)
Target shape: (20000,)

Feature columns:
['age', 'gender', 'course', 'study_hours', 'class_attendance', 'internet_access',
'sleep_hours', 'sleep_quality', 'study_method', 'facility_rating', 'exam_difficul
ty']
```

In [26]:
```python
# Split the dataset into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

print("Training set size:", X_train.shape[0])
print("Testing set size:", X_test.shape[0])
print(f"\nTraining set: {len(X_train)} samples ({len(X_train)/len(X)*100:.1f}%)"
print(f"Testing set: {len(X_test)} samples ({len(X_test)/len(X)*100:.1f}%)")
```

```
Training set size: 16000
Testing set size: 4000

Training set: 16000 samples (80.0%)
Testing set: 4000 samples (20.0%)
```

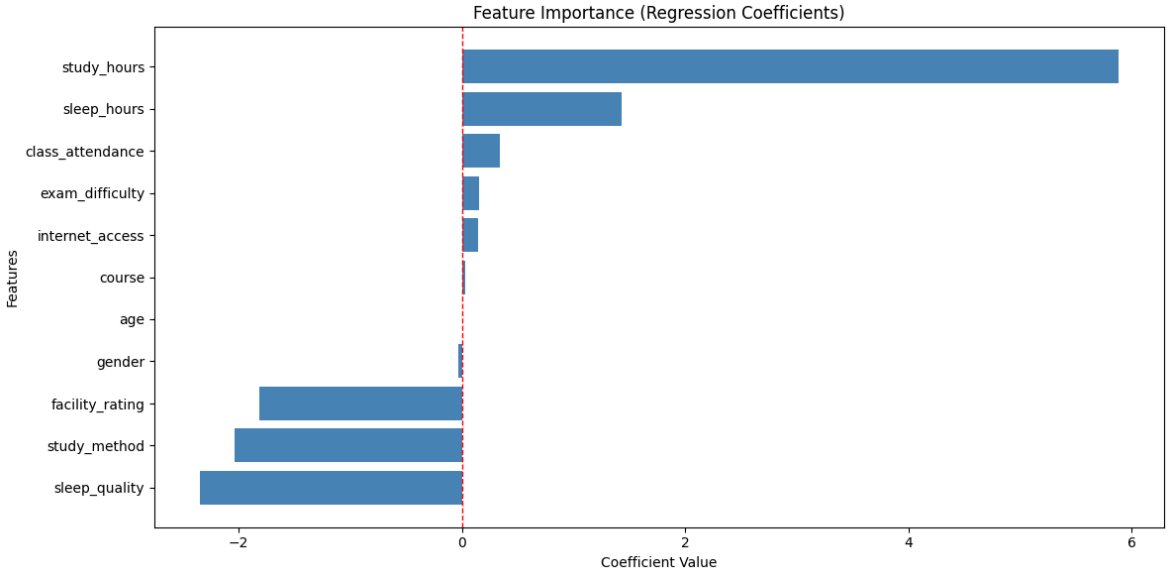
6. Build Multiple Linear Regression Model

In [27]:
```python
# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

print("Model trained successfully!")
print("\nModel Coefficients:")
coef_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model.coef_
}).sort_values('Coefficient', key=abs, ascending=False)

print(coef_df.to_string(index=False))
print(f"\nIntercept: {model.intercept_:.4f}")
```

```
Model trained successfully!

Model Coefficients:
         Feature  Coefficient
     study_hours     5.881195
   sleep_quality    -2.345414
    study_method    -2.040250
 facility_rating    -1.812587
     sleep_hours     1.435547
class_attendance     0.342715
 exam_difficulty     0.149939
 internet_access     0.147043
          gender    -0.036329
          course     0.029321
             age     0.007428

Intercept: 12.6640
```

In [28]:
```python
# Visualize feature importance (coefficients)
plt.figure(figsize=(12, 6))
coef_df_sorted = coef_df.sort_values('Coefficient')
```

```
plt.barh(coef_df_sorted['Feature'], coef_df_sorted['Coefficient'], color='steelb
plt.xlabel('Coefficient Value')
plt.ylabel('Features')
plt.title('Feature Importance (Regression Coefficients)')
plt.axvline(x=0, color='red', linestyle='--', linewidth=1)
plt.tight_layout()
plt.show()
```



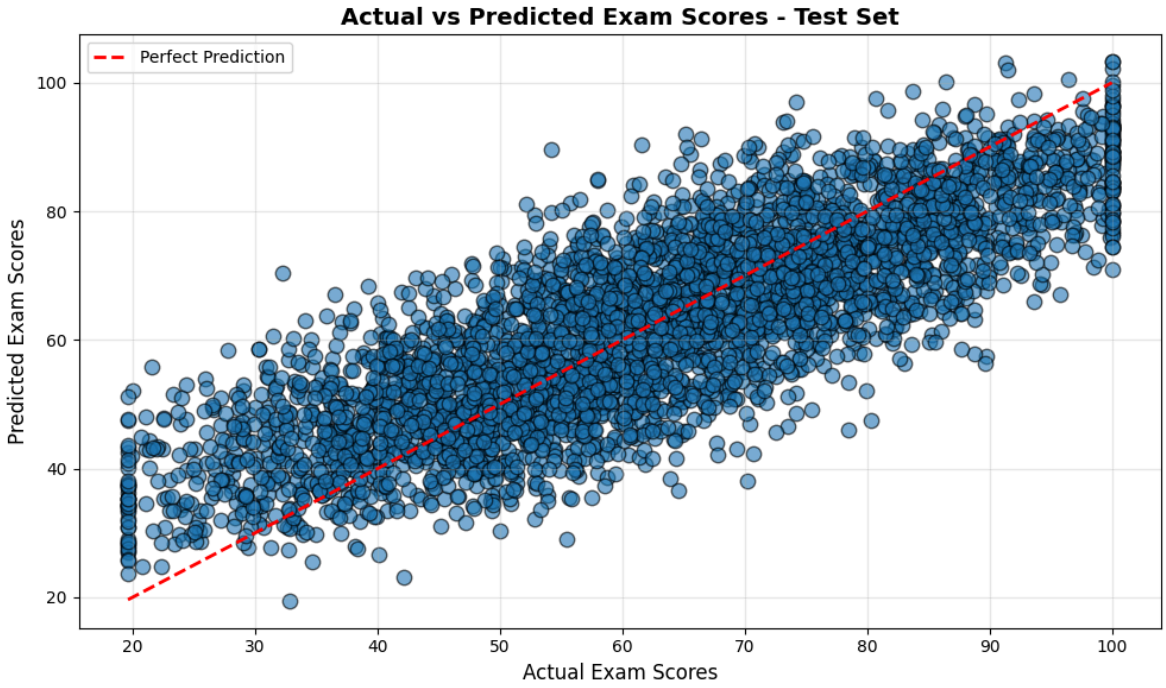
7. Model Predictions

In [29]:
```
# Make predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print("Predictions made successfully!")
print(f"\nSample predictions (first 10):")
comparison_df = pd.DataFrame({
    'Actual': y_test.values[:10],
    'Predicted': y_test_pred[:10],
    'Difference': y_test.values[:10] - y_test_pred[:10]
})
print(comparison_df)
```

```
Predictions made successfully!

Sample predictions (first 10):
   Actual  Predicted  Difference
0    31.1  35.989417   -4.889417
1    81.6  71.391764   10.208236
2    68.0  52.903226   15.096774
3   100.0  93.428419    6.571581
4    84.8  80.749395    4.050605
5    52.8  32.126485   20.673515
6    53.1  55.510023   -2.410023
7    81.0  67.908527   13.091473
8    72.6  64.550403    8.049597
9    25.4  30.249939   -4.849939
```

In [30]:
```
# Scatter plot: Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, edgecolors='k', s=80)
```

```
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2
plt.xlabel('Actual Exam Scores', fontsize=12)
plt.ylabel('Predicted Exam Scores', fontsize=12)
plt.title('Actual vs Predicted Exam Scores - Test Set', fontsize=14, fontweight=
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



8. Model Evaluation

In [31]:

```
# Calculate evaluation metrics
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_rmse = np.sqrt(train_mse)
test_rmse = np.sqrt(test_mse)
train_mae = mean_absolute_error(y_train, y_train_pred)
test_mae = mean_absolute_error(y_test, y_test_pred)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

print("="*60)
print("MODEL EVALUATION METRICS")
print("="*60)
print("\nTraining Set Performance:")
print(f"  Mean Squared Error (MSE):  {train_mse:.4f}")
print(f"  Root Mean Squared Error (RMSE): {train_rmse:.4f}")
print(f"  Mean Absolute Error (MAE): {train_mae:.4f}")
print(f"  R² Score: {train_r2:.4f}")

print("\nTesting Set Performance:")
print(f"  Mean Squared Error (MSE):  {test_mse:.4f}")
print(f"  Root Mean Squared Error (RMSE): {test_rmse:.4f}")
print(f"  Mean Absolute Error (MAE): {test_mae:.4f}")
print(f"  R² Score: {test_r2:.4f}")
print("="*60)
```

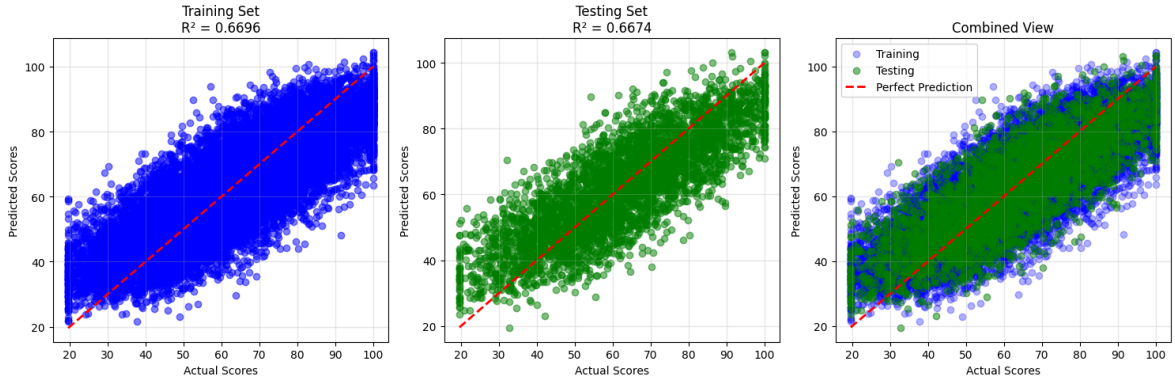
```
# Check for overfitting/underfitting
print(f"\nR² Difference (Train - Test): {train_r2 - test_r2:.4f}")
if abs(train_r2 - test_r2) < 0.05:
    print("✓ Model is well-balanced (no significant overfitting)")
elif train_r2 > test_r2:
    print("⚠ Model may be slightly overfitting")
else:
    print("⚠ Model may be underfitting")
```

```
============================================================
MODEL EVALUATION METRICS
============================================================

Training Set Performance:
  Mean Squared Error (MSE):  118.1183
  Root Mean Squared Error (RMSE): 10.8682
  Mean Absolute Error (MAE): 8.7191
  R² Score: 0.6696

Testing Set Performance:
  Mean Squared Error (MSE):  118.9531
  Root Mean Squared Error (RMSE): 10.9066
  Mean Absolute Error (MAE): 8.8006
  R² Score: 0.6674
============================================================

R² Difference (Train - Test): 0.0021
✓ Model is well-balanced (no significant overfitting)
```

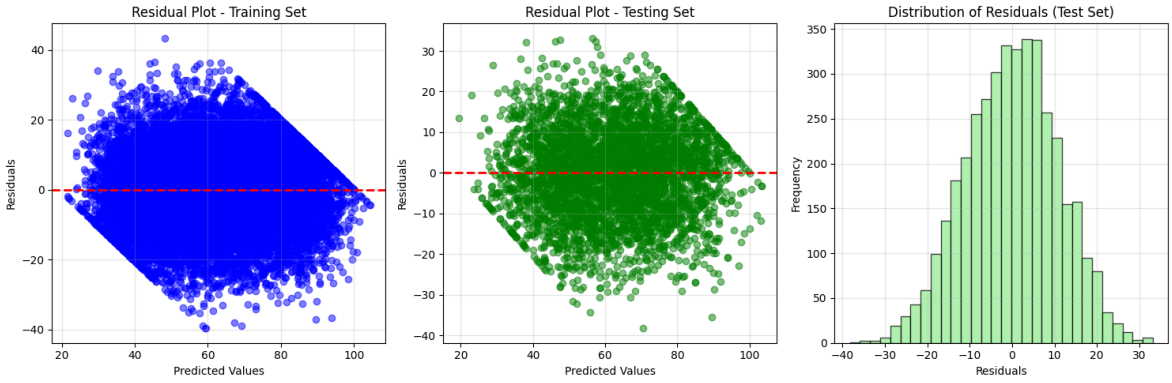
9. Visualization of Results

In [32]:
```
# Actual vs Predicted values
plt.figure(figsize=(15, 5))

# Training set
plt.subplot(1, 3, 1)
plt.scatter(y_train, y_train_pred, alpha=0.5, color='blue')
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'r--',
plt.xlabel('Actual Scores')
plt.ylabel('Predicted Scores')
plt.title(f'Training Set\nR² = {train_r2:.4f}')
plt.grid(True, alpha=0.3)

# Testing set
plt.subplot(1, 3, 2)
plt.scatter(y_test, y_test_pred, alpha=0.5, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2
plt.xlabel('Actual Scores')
plt.ylabel('Predicted Scores')
plt.title(f'Testing Set\nR² = {test_r2:.4f}')
plt.grid(True, alpha=0.3)

# Combined view
plt.subplot(1, 3, 3)
plt.scatter(y_train, y_train_pred, alpha=0.3, color='blue', label='Training')
plt.scatter(y_test, y_test_pred, alpha=0.5, color='green', label='Testing')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2, label='Perfect Pre
plt.xlabel('Actual Scores')
plt.ylabel('Predicted Scores')
```

```
plt.title('Combined View')
plt.legend()
plt.grid(True, alpha=0.3)

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

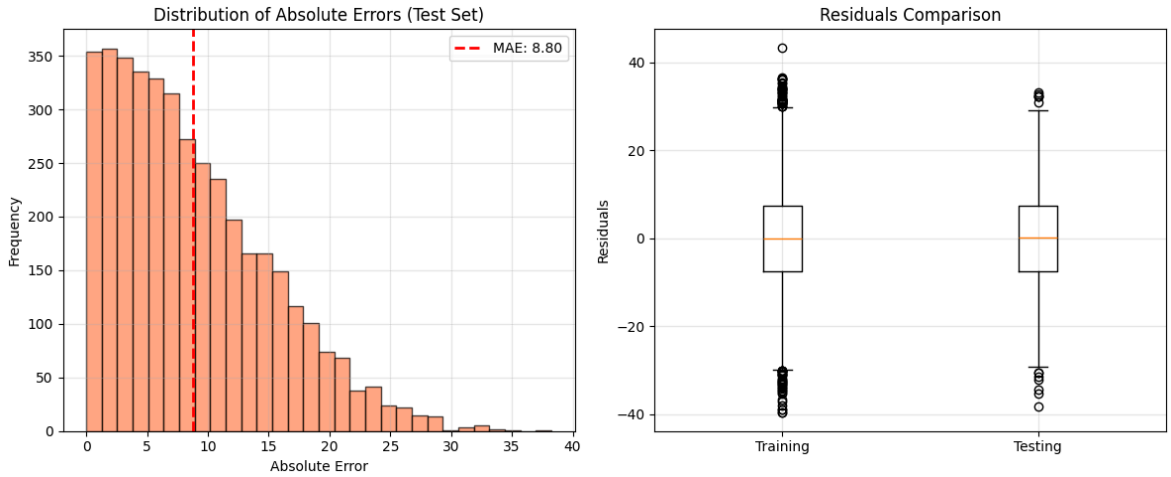


In [33]:
```
# Residual analysis
residuals_train = y_train - y_train_pred
residuals_test = y_test - y_test_pred

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

# Residual plot for training
plt.subplot(1, 3, 1)
plt.scatter(y_train_pred, residuals_train, alpha=0.5, color='blue')
plt.axhline(y=0, color='r', linestyle='--', lw=2)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot - Training Set')
plt.grid(True, alpha=0.3)

# Residual plot for testing
plt.subplot(1, 3, 2)
plt.scatter(y_test_pred, residuals_test, alpha=0.5, color='green')
plt.axhline(y=0, color='r', linestyle='--', lw=2)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot - Testing Set')
plt.grid(True, alpha=0.3)

# Residual distribution
plt.subplot(1, 3, 3)
plt.hist(residuals_test, bins=30, edgecolor='black', color='lightgreen', alpha=0
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Distribution of Residuals (Test Set)')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"Mean of residuals (should be close to 0): {residuals_test.mean():.4f}")
print(f"Std of residuals: {residuals_test.std():.4f}")
```




```
Mean of residuals (should be close to 0): -0.0394
Std of residuals: 10.9079
```

In [34]:
```python
# Error distribution comparison
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
errors_test = np.abs(y_test - y_test_pred)
plt.hist(errors_test, bins=30, edgecolor='black', color='coral', alpha=0.7)
plt.xlabel('Absolute Error')
plt.ylabel('Frequency')
plt.title('Distribution of Absolute Errors (Test Set)')
plt.axvline(test_mae, color='red', linestyle='--', linewidth=2, label=f'MAE: {te
plt.legend()
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
plt.boxplot([residuals_train, residuals_test], labels=['Training', 'Testing'])
plt.ylabel('Residuals')
plt.title('Residuals Comparison')
plt.grid(True, alpha=0.3)

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



10. Model Summary

In [35]:
```python
# Model equation
print("="*80)
print("MULTIPLE LINEAR REGRESSION MODEL EQUATION")
print("="*80)
print(f"\nexam_score = {model.intercept_:.4f}", end="")
```

```
for feature, coef in zip(X.columns, model.coef_):
    sign = "+" if coef >= 0 else "-"
    print(f" {sign} {abs(coef):.4f} * {feature}", end="")
print("\n" + "="*80)

# Summary statistics
summary_data = {
    'Metric': ['R² Score', 'RMSE', 'MAE', 'MSE'],
    'Training': [f'{train_r2:.4f}', f'{train_rmse:.4f}', f'{train_mae:.4f}', f'{
    'Testing': [f'{test_r2:.4f}', f'{test_rmse:.4f}', f'{test_mae:.4f}', f'{test
}
summary_df = pd.DataFrame(summary_data)

print("\n\nMODEL PERFORMANCE SUMMARY:")
print(summary_df.to_string(index=False))
print("\n" + "="*80)
```

```
================================================================================
MULTIPLE LINEAR REGRESSION MODEL EQUATION
================================================================================

exam_score = 12.6640 + 0.0074 * age - 0.0363 * gender + 0.0293 * course + 5.8812
* study_hours + 0.3427 * class_attendance + 0.1470 * internet_access + 1.4355 * s
leep_hours - 2.3454 * sleep_quality - 2.0403 * study_method - 1.8126 * facility_r
ating + 0.1499 * exam_difficulty
================================================================================


MODEL PERFORMANCE SUMMARY:
  Metric Training  Testing
R² Score   0.6696   0.6674
    RMSE  10.8682  10.9066
     MAE   8.7191   8.8006
     MSE 118.1183 118.9531

================================================================================
```

11. Conclusions

Key Findings:

1. The multiple linear regression model has been successfully trained on the exam score prediction dataset
2. Model performance can be evaluated using R^2 score, RMSE, MAE, and MSE metrics
3. Feature importance is determined by the magnitude of regression coefficients
4. Residual analysis helps identify potential issues with the model assumptions