Course Project Code
Hunter Worssam
August 2nd, 2025
Statistical Models & Regression

Figure 1: Code and Results for Dataset Standardization

```
In [6]: # Load the dataset
df = pd.read_excel("NFL_1976_Team_Performance.xlsx")

# Set proper headers (first row is actual header)
df.columns = df.iloc[0]
df = df.drop(index=0).reset_index(drop=True)

# Convert numeric columns to float
for col in df.columns:
    if col != "Team":
        df[col] = pd.to_numeric(df[col], errors='coerce')

# Split into X (predictors) and y (response)
X = df.loc[:, df.columns.str.startswith("x")]
y = pd.to_numeric(df["y"], errors='coerce')

# Standardize X
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert back to DataFrame for easier inspection
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Optional: Merge back for modeling
df_scaled = pd.concat([df[["Team"]], y, X_scaled_df], axis=1)

# Show a preview
print(df_scaled.head())
```

```
          Team   y        x1        x2        x3        x4        x5  \
0   Washington  10  0.007355 -0.290161  0.133853  0.509429  0.390493
1    Minnesota  11 -0.283065  1.484906  0.082512  0.182625  0.292870
2  New England  11  2.235670 -0.796157  0.749941  0.057671  1.366725
3      Oakland  13  0.461466  1.586922  1.520052 -1.355273 -0.390493
4   Pittsburgh  10  2.272632 -0.941019  0.287875 -0.538264  1.464348

         x6        x7        x8        x9
0  0.649540  0.291519  0.268505 -0.724101
1 -1.453450 -0.596534 -0.034893 -1.898559
2  1.031902  1.406309 -0.727974  0.161893
3  1.389327  0.612730 -0.572101  1.195552
4  0.383549  1.500783 -1.813524 -0.899240
```

Figure 2: Code and Results for Null Model Specification

Null Model Specification

In [15]:
```
# Add intercept
X_scaled_with_intercept = sm.add_constant(X_scaled_df)

# Fit OLS model
model = sm.OLS(y, X_scaled_with_intercept).fit()

# View full model summary
print(model.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.815
Model:                            OLS   Adj. R-squared:                  0.723
Method:                 Least Squares   F-statistic:                     8.839
Date:                Sun, 03 Aug 2025   Prob (F-statistic):           5.33e-05
Time:                        13:28:36   Log-Likelihood:                -50.477
No. Observations:                  28   AIC:                             121.0
Df Residuals:                      18   BIC:                             134.3
Df Model:                           9                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.9643      0.346     20.129      0.000       6.237       7.691
x1             0.3136      0.759      0.413      0.684      -1.281       1.909
x2             1.7858      0.411      4.341      0.000       0.922       2.650
x3             0.2428      0.506      0.480      0.637      -0.821       1.306
x4             0.3381      0.433      0.781      0.445      -0.572       1.248
x5            -0.0090      0.480     -0.019      0.985      -1.017       0.999
x6             0.1925      0.391      0.493      0.628      -0.628       1.013
x7             0.8397      0.801      1.048      0.308      -0.844       2.523
x8            -1.3827      0.730     -1.895      0.074      -2.916       0.150
x9            -0.5209      0.413     -1.261      0.223      -1.389       0.347
==============================================================================
Omnibus:                        0.121   Durbin-Watson:                   1.755
Prob(Omnibus):                  0.941   Jarque-Bera (JB):                0.148
Skew:                          -0.123   Prob(JB):                        0.929
Kurtosis:                       2.742   Cond. No.                         5.19
==============================================================================

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

Figure 3: Code and Results for Variance Inflation Factor Calculation

In [22]:
```
# X_scaled_df is your standardized predictor DataFrame (without intercept)
vif_data = pd.DataFrame()
vif_data["feature"] = X_scaled_df.columns
vif_data["VIF"] = [variance_inflation_factor(X_scaled_df.values, i) for i in range(X_scale

print(vif_data)
```

```
  feature       VIF
0      x1  4.814532
1      x2  1.413644
2      x3  2.141417
3      x4  1.566738
4      x5  1.922913
5      x6  1.275302
6      x7  5.362643
7      x8  4.447482
8      x9  1.425342
```

Figure 4: Code and Results for Stepwise Feature Selection

```
In [31]: # Stepwise feature selection function ===
def stepwise_selection(X, y, threshold_in=0.05, threshold_out=0.10, verbose=True):
    included = []
    while True:
        changed = False

        # Forward step
        excluded = list(set(X.columns) - set(included))
        new_pval = pd.Series(index=excluded, dtype=float)
        for new_col in excluded:
            model = sm.OLS(y, sm.add_constant(X[included + [new_col]])).fit()
            new_pval[new_col] = model.pvalues[new_col]
        if not new_pval.empty:
            best_pval = new_pval.min()
            if best_pval < threshold_in:
                best_feature = new_pval.idxmin()
                included.append(best_feature)
                changed = True
                if verbose:
                    print(f"Add   {best_feature:8} with p-value {best_pval:.6f}")

        # Backward step
        if included:
            model = sm.OLS(y, sm.add_constant(X[included])).fit()
            pvals = model.pvalues.iloc[1:]  # exclude intercept
            worst_pval = pvals.max()
            if worst_pval > threshold_out:
                worst_feature = pvals.idxmax()
                included.remove(worst_feature)
                changed = True
                if verbose:
                    print(f"Drop  {worst_feature:8} with p-value {worst_pval:.6f}")

        if not changed:
            break
    return included

# Run it
threshold_in = 0.20
threshold_out = 0.30

selected_features = stepwise_selection(X_scaled_df, y,
                                       threshold_in=threshold_in,
                                       threshold_out=threshold_out,
                                       verbose=True)

print("\nFinal selected variables:", selected_features)
```

```
Add   x8       with p-value 0.000008
Add   x2       with p-value 0.000192
Add   x7       with p-value 0.031347

Final selected variables: ['x8', 'x2', 'x7']
```

Figure 5: Code and Results for Final Model Specification

Subset Model Performance

In [37]:
```
selected_X = X_scaled_df[['x2', 'x7', 'x8']]

# Add intercept
X_subset_with_const = sm.add_constant(selected_X)

# Fit the model
subset_model = sm.OLS(y, X_subset_with_const).fit()

# Print model summary
print(subset_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.786
Model:                            OLS   Adj. R-squared:                  0.759
Method:                 Least Squares   F-statistic:                     29.35
Date:                Sun, 03 Aug 2025   Prob (F-statistic):           3.36e-08
Time:                        14:37:40   Log-Likelihood:                -52.564
No. Observations:                  28   AIC:                             113.1
Df Residuals:                      24   BIC:                             118.5
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.9643      0.323     21.574      0.000       6.298       7.631
x2             1.7716      0.341      5.203      0.000       1.069       2.474
x7             1.0569      0.462      2.286      0.031       0.103       2.011
x8            -1.7060      0.454     -3.760      0.001      -2.643      -0.769
==============================================================================
Omnibus:                        0.653   Durbin-Watson:                   1.504
Prob(Omnibus):                  0.721   Jarque-Bera (JB):                0.578
Skew:                           0.319   Prob(JB):                        0.749
Kurtosis:                       2.703   Cond. No.                         2.46
==============================================================================
```

Figure 6: Code and Results for Basic Residuals Plot

Residuals vs Fitted Values Plot

In [53]:
```
plt.figure(figsize=(6, 4))
sns.residplot(x=fitted_vals, y=residuals, lowess=True,
              line_kws={'color': 'red', 'lw': 2})
plt.axhline(0, color='black', linestyle='dotted')
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted")
plt.tight_layout()
plt.show()
```

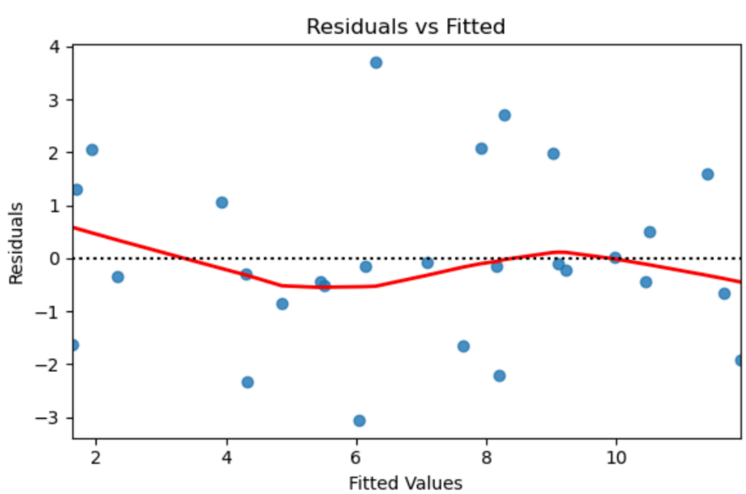


Figure 7: Code and Results for Normality Plot

Normal Plot

In [57]:
```
plt.figure(figsize=(6, 4))
sm.qqplot(residuals, line='45', fit=True)
plt.title("Normal Q-Q")
plt.tight_layout()
plt.show()
```

```
<Figure size 600x400 with 0 Axes>
```

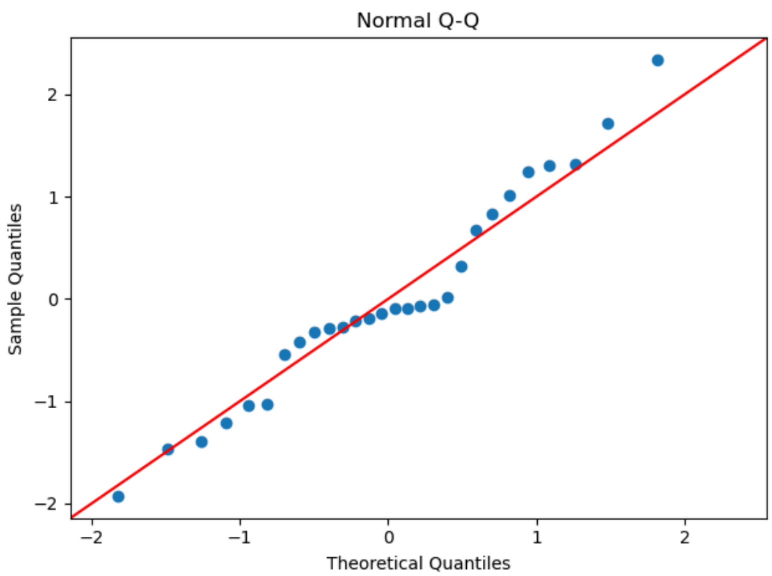


Figure 8: Code and Results for Residuals vs Leverage Plot

Residuals vs Leverage Plot

In [63]:
```
plt.figure(figsize=(6, 4))
plt.scatter(leverage, standardized_residuals, alpha=0.7)
sns.regplot(x=leverage, y=standardized_residuals,
            scatter=False, lowess=True, line_kws={'color': 'red'})
plt.xlabel("Leverage")
plt.ylabel("Standardized Residuals")
plt.title("Residuals vs Leverage")
plt.tight_layout()
plt.show()
```

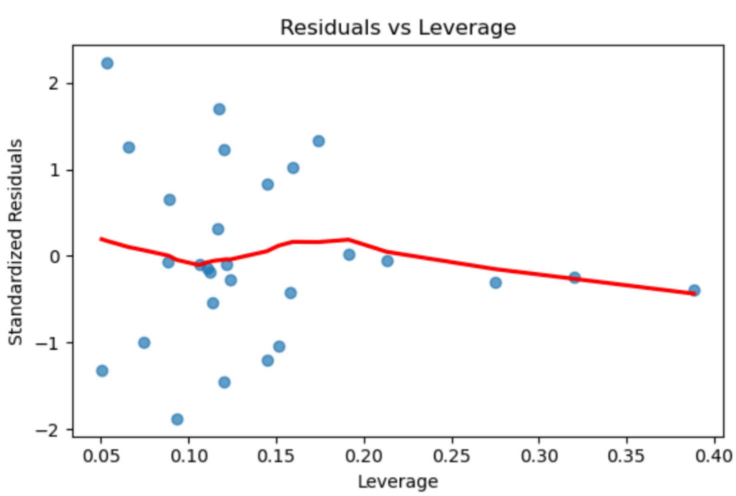


Figure 9: Code and Results for Cook's Distance Plot

Cook's Distance Plot

In [75]:

```python
threshold = 4 / len(cooks_d)  # or use 1.0 if you want to be stricter

plt.figure(figsize=(6, 4))
plt.stem(np.arange(len(cooks_d)), cooks_d, markerfmt=",", basefmt=" ", linefmt="orange")
plt.axhline(threshold, color='red', linestyle='--', label=f'Threshold = {threshold:.3f}')
plt.title("Cook's Distance")
plt.xlabel("Observation Index")
plt.ylabel("Cook's Distance")
plt.legend()
plt.tight_layout()
plt.show()
```

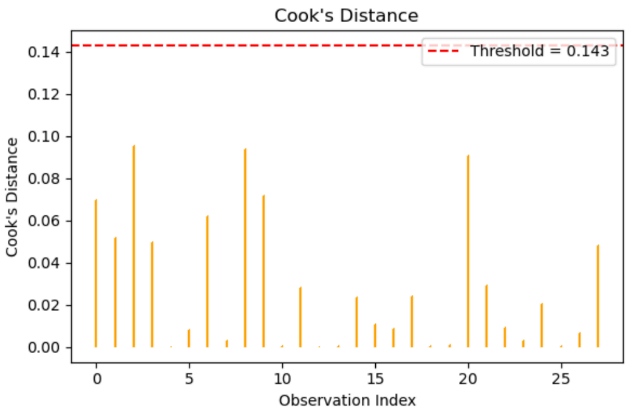


Figure 10: Code and Results for Externally Studentized Residuals Plot

Externally Studentized Residuals plot

In [98]:

```python
# Externally studentized residuals (useful for formal outlier testing)
external_resid = subset_model.get_influence().resid_studentized_external

plt.figure(figsize=(6, 4))
plt.stem(np.arange(len(external_resid)), external_resid,
         markerfmt=",", basefmt=" ", linefmt="darkgreen")
plt.axhline(0, color='black', linestyle='dotted')
plt.axhline(2, color='red', linestyle='--', label='±2 Threshold')
plt.axhline(-2, color='red', linestyle='--')

# Label outliers
for i, val in enumerate(external_resid):
    if np.abs(val) > 2:
        plt.text(i, val, f'{i}', ha='center', va='bottom', fontsize=8)

plt.title("Externally Studentized Residuals")
plt.xlabel("Observation Index")
plt.ylabel("Externally Studentized Residual")
plt.legend()
plt.tight_layout()
plt.show()
```

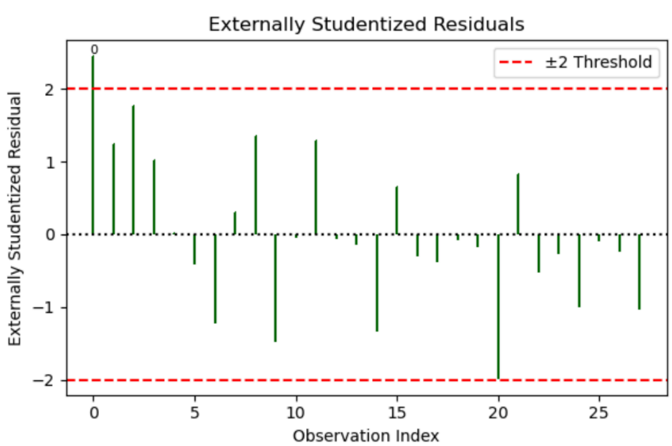


Figure 11: Code and Results for High Leverage Points

High Leverage Points

In [91]:
```
# Calculate leverage (hat values)
leverage = subset_model.get_influence().hat_matrix_diag

# Define thresholds
avg_leverage = (X_subset_with_const.shape[1]) / X_subset_with_const.shape[0]
high_leverage_threshold = 2 * avg_leverage

# Find high-leverage points
leverage_df = pd.DataFrame({
    "Team": df["Team"],
    "Leverage": leverage,
    "High_Leverage": leverage > high_leverage_threshold
})

# Show high leverage observations
high_leverage_points = leverage_df[leverage_df["High_Leverage"] == True]
high_leverage_points.sort_values(by="Leverage", ascending=False)
```

Out[91]:

	Team	Leverage	High_Leverage
17	Kansas City	0.388536	True
26	Seattle	0.320298	True

Figure 12: Code and Results for Points with Large Residuals

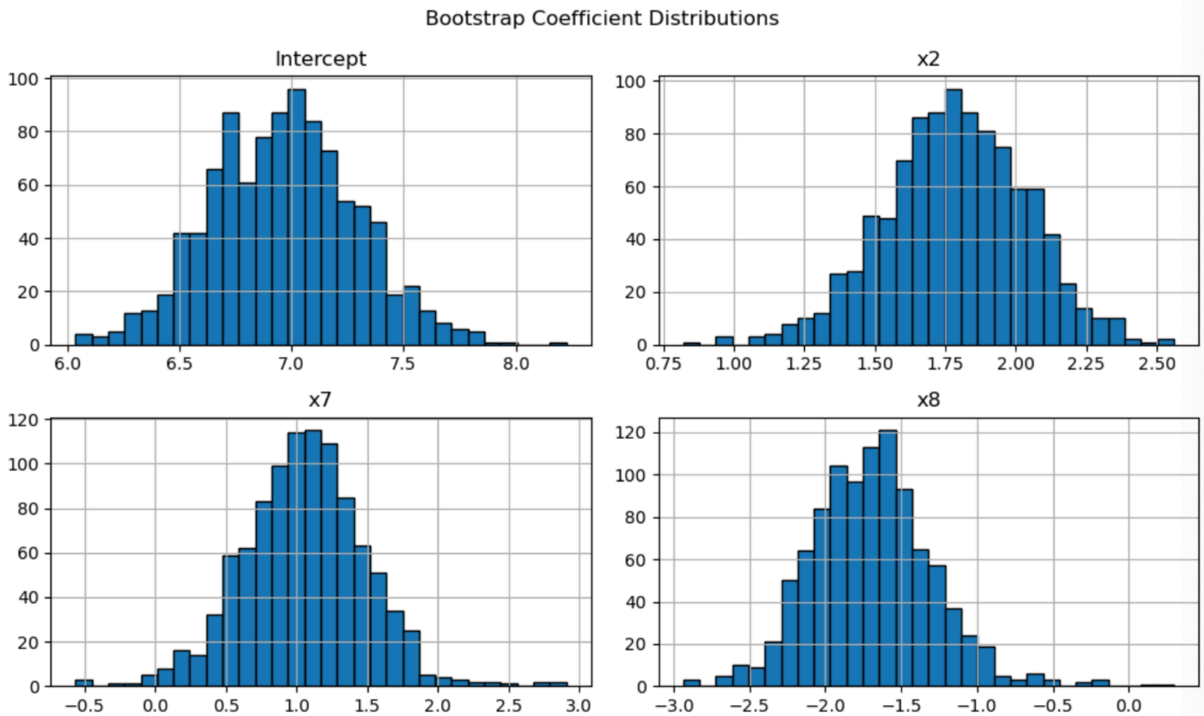
In [95]:
```
# Get residuals from the model
residuals = subset_model.resid

# Identify observations with residuals > |3|
outlier_mask = np.abs(residuals) > 3
outliers = df[outlier_mask]

# Show relevant info for context
outliers_with_resid = df.loc[outlier_mask].copy()
outliers_with_resid["Residual"] = residuals[outlier_mask]

# Sort by absolute residual (optional)
outliers_with_resid = outliers_with_resid.reindex(outliers_with_resid["Residual"].abs().sort_values(ascendi

print(outliers_with_resid)
```

```
0            Team   y    x1    x2    x3     x4  x5   x6     x7    x8    x9  \
0      Washington  10  2113  1985  38.9  0.647   4  868  0.597  2205  1917
20  New York Giants   3  1904  1792  39.7  0.381  -9  734  0.619  2203  1988

0   Residual
0   3.699712
20 -3.051523
```

Figure 13: Code and Results for Bootstrap Sampling Distributions

In [130…

```
# Set random seed for reproducibility
np.random.seed(42)

# Define predictors and outcome
X = X_scaled_df[['x2', 'x7', 'x8']]

# Initialize storage
B = 1000  # Number of bootstrap samples
boot_coefs = []
boot_r2 = []

# Bootstrap loop
for _ in range(B):
    # Sample with replacement
    indices = np.random.choice(len(X), size=len(X), replace=True)
    X_boot = X.iloc[indices]
    y_boot = y.iloc[indices]

    # Fit model
    X_boot_const = sm.add_constant(X_boot)
    model = sm.OLS(y_boot, X_boot_const).fit()

    # Store coefficients
    boot_coefs.append(model.params.values)

    # Store R²
    y_pred = model.predict(X_boot_const)
    boot_r2.append(r2_score(y_boot, y_pred))

# Convert to DataFrame
coef_df = pd.DataFrame(boot_coefs, columns=['Intercept', 'x2', 'x7', 'x8'])
r2_series = pd.Series(boot_r2)

# ------------------------------
# Plot coefficient distributions
# ------------------------------
coef_df.hist(bins=30, figsize=(10, 6), layout=(2, 2), edgecolor='black')
plt.suptitle('Bootstrap Coefficient Distributions')
plt.tight_layout()
plt.show()

# ------------------------------
# Plot R² distribution
# ------------------------------
plt.figure(figsize=(6, 4))
sns.histplot(r2_series, bins=30, kde=True, color='skyblue', edgecolor='black')
plt.axvline(r2_series.mean(), color='red', linestyle='--', label=f'Mean R² = {r2_series.mean():.3f}')
plt.title('Bootstrap Distribution of R²')
plt.xlabel('R²')
plt.ylabel('Frequency')
plt.legend()
plt.tight_layout()
plt.show()
```


Bootstrap Coefficient Distributions
Intercept
x2
x7
x8

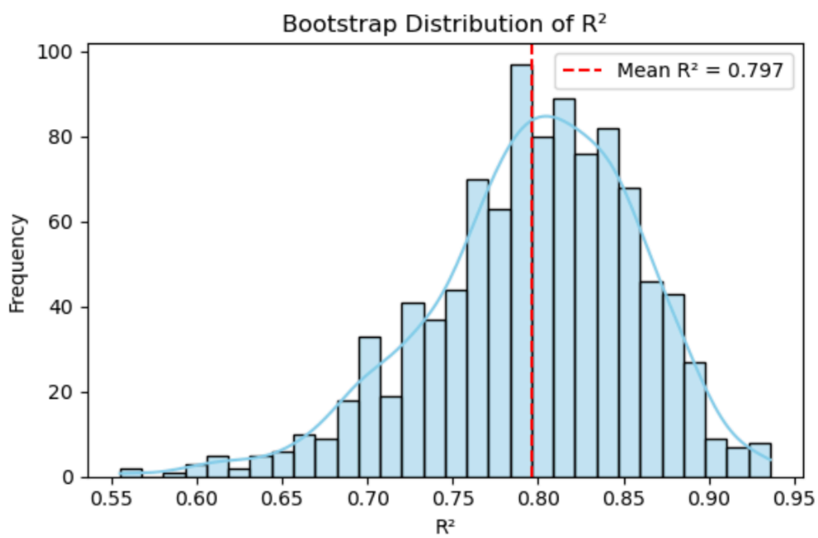

Bootstrap Distribution of R^2
Mean R^2 = 0.797
Frequency
R^2

Figure 14: Code and Results for Bootstrap Sampling 95% CI's (coefficients & R^2)

```
In [135…

# Compute 95% CI for each coefficient
coef_ci = coef_df.quantile([0.025, 0.975])
coef_means = coef_df.mean()

print("95% Confidence Intervals, Midpoints, and Means for Coefficients:\n")
for col in coef_df.columns:
    lower, upper = coef_ci[col].values
    midpoint = (lower + upper) / 2
    mean = coef_means[col]
    print(f"{col:>9}:  CI = ({lower:.4f}, {upper:.4f})   Midpoint = {midpoint:.4f}   Mean = {mean:.4f}")

# Compute 95% CI, midpoint, and mean for R²
r2_lower = np.percentile(r2_series, 2.5)
r2_upper = np.percentile(r2_series, 97.5)
r2_midpoint = (r2_lower + r2_upper) / 2
r2_mean = r2_series.mean()

print("\nR² Summary:")
print(f"R²: CI = ({r2_lower:.4f}, {r2_upper:.4f})   Midpoint = {r2_midpoint:.4f}   Mean = {r2_mean:.4f}")
```

```
95% Confidence Intervals, Midpoints, and Means for Coefficients:

Intercept:  CI = (6.3489, 7.6086)   Midpoint = 6.9788   Mean = 6.9664
       x2:  CI = (1.2638, 2.2678)   Midpoint = 1.7658   Mean = 1.7834
       x7:  CI = (0.1679, 1.8298)   Midpoint = 0.9988   Mean = 1.0535
       x8:  CI = (-2.4239, -0.8950)   Midpoint = -1.6594   Mean = -1.6932

R² Summary:
R²: CI = (0.6617, 0.8963)   Midpoint = 0.7790   Mean = 0.7965
```

Figure 15: Code and Results for Final Model PRESS Statistic

```
In [145…

# Residuals and leverage values
residuals = subset_model.resid
leverage = subset_model.get_influence().hat_matrix_diag

# Compute PRESS: sum of squared leave-one-out residuals
press = np.sum((residuals / (1 - leverage))**2)

# Compute RMSE from PRESS
press_rmse = np.sqrt(press / len(residuals))


# Get predictions on the full dataset
y_pred = subset_model.fittedvalues
# Calculate RMSE
rmse_actual = np.sqrt(mean_squared_error(y, y_pred))

print(f"Model RMSE on Training Data: {rmse_actual:.4f}")
print(f"PRESS: {press:.4f}")
print(f"PRESS RMSE: {press_rmse:.4f}")
```

```
Model RMSE on Training Data: 1.5815
PRESS: 87.6946
PRESS RMSE: 1.7697
```

Figure 16. Raw Dataset

Team	y	x1	x2	x3	x4	x5	x6	x7	x8	x9
WSH	10	2113	1985	38.9	0.647	4	868	0.597	2205	1917
MIN	11	2003	2855	38.8	0.613	3	615	0.55	2096	1575
NE	11	2957	1737	40.1	0.6	14	914	0.656	1847	2175
OAK	13	2285	2905	41.6	0.453	-4	957	0.614	1903	2476
PIT	10	2971	1666	39.2	0.538	15	836	0.661	1457	1866
BAL	11	2309	2927	39.7	0.741	8	786	0.61	1848	2339
LA	10	2528	2341	38.1	0.654	12	754	0.661	1564	2092
DAL	11	2147	2737	37.0	0.783	-1	761	0.58	1821	1909
ATL	4	1689	1414	42.1	0.476	-3	714	0.57	2577	2001
BUF	2	2566	1838	42.3	0.542	-1	797	0.589	2476	2254
CHI	7	2363	1480	37.3	0.48	19	984	0.675	1984	2217
CIN	10	2109	2191	39.5	0.519	6	700	0.572	1917	1758
CLE	9	2295	2229	37.4	0.536	-5	1037	0.588	1761	2032
DEN	9	1932	2204	35.1	0.714	3	986	0.586	1709	2025
DET	6	2213	2140	38.8	0.583	6	819	0.592	1901	1686
GB	5	1722	1730	36.6	0.526	-19	791	0.544	2288	1835
HOU	5	1498	2072	35.3	0.593	-5	776	0.496	2027	1914
KC	5	1873	2929	41.1	0.553	10	789	0.543	2861	2496
MIA	6	2118	2268	38.2	0.696	6	582	0.587	2411	2670
NO	4	1775	1983	39.3	0.783	7	901	0.517	2289	2202
NYG	3	1904	1792	39.7	0.381	-9	734	0.619	2203	1988
NYJ	3	1929	1606	39.7	0.688	-21	627	0.527	2592	2324
PHI	4	2080	1492	35.5	0.688	-8	722	0.578	2053	2550
STL	10	2301	2835	35.3	0.741	2	683	0.597	1979	2110
SD	6	2040	2416	38.7	0.5	0	576	0.549	2048	2628
SF	8	2447	1638	39.9	0.571	-8	848	0.653	1786	1776

Team	y	x1	x2	x3	x4	x5	x6	x7	x8	x9
SEA	2	1416	2649	37.4	0.563	-22	684	0.438	2876	2524
TB	0	1503	1503	39.3	0.47	-9	875	0.535	2560	2241

y : Games won (per 14-game season)
x_1: Rushing yards (season)
x_2: Passing yards (season)
x_3: Punting average (yards/punt)
x_4: Field goal percentage (FGs made / FGs attempted, season)
x_5: Turnover differential
x_6: Penalty yards (season)
x_7: Percent rushing (rushing plays / total plays)
x_8: Opponents' rushing yards (season)
x_9: Opponents' passing yards (season)