Project 4

Monte Carlo Simulation: LSTM, Random Forest, and Linear Regression

Objective

This Monte Carlo simulation compares the performance of three regression models: - Linear Regression - Ridge Regression - Random Forest in predicting WAR, using 30 synthetic datasets (10 per correlation level: 0.0, 0.5, 0.99). The models will be evaluated on Mean Squared Error (MSE). The results will be summarized using the mean and standard deviation of the MSE for each combination of model and correlation structure.

Question: How do Linear Regression, Random Forest, and Ridge Regression perform in predicting a baseball player's WAR under varying levels of predictor correlation (none, mild, high)?

Note: We removed the LSTM model because of time concerns (that one takes very long to run) and because Random Forest outperformed it quite significantly anyways.

Setup

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score  # Added missing import
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns
```

```
import matplotlib.pyplot as plt

np.random.seed(116)
```

Data Generation Function

We will use the mean and standard deviations of the actual data.

```
complete_dataset = pd.read_csv("../Project 1/Data/Complete_Data.csv")
key_columns = ['xba', 'barrel_batted_rate', 'player_age', 'WAR', 'k_percent', 'bb_percent',
complete_dataset[key_columns].describe()
def generate_data(n=100, correlation=0.0): # we set default n to be 100
    stats = {
      # all the data comes from the complete_dataset.csv.describe()
        'xba': (0.251, 0.025),
        'barrel_batted_rate': (7.632, 4.131),
        'player_age': (28.284, 3.675),
        'batting_avg': (0.257, 0.030),
        'k_percent': (21.245, 5.921),
        'bb_percent': (8.581, 3.078),
        'on_base_plus_slg': (0.757, 0.094),
        'Rbat+': (105.041, 26.463),
        'WAR_lag1': (2.027, 1.972),
        'WAR_lag2': (2.027, 1.972),
        'WAR_lag3': (2.027, 1.972)
    }

    features = ['xba', 'barrel_batted_rate', 'player_age', 'batting_avg',
                'k_percent', 'bb_percent', 'on_base_plus_slg', 'Rbat+',
                'WAR_lag1', 'WAR_lag2', 'WAR_lag3']

    num_features = len(features)

    # covariance matrix
    cov_matrix = []
    for i in range(num_features):
        row = []
        for j in range(num_features):
            if i == j:
                row.append(1.0)  # Same feature has correlation 1 with itself
            else:
```

```
            row.append(correlation)
        cov_matrix.append(row)

    cov_matrix = np.array(cov_matrix)

    # standard deviations
    stds = []
    for feature in features:
        stds.append(stats[feature][1])
    stds = np.array(stds)

    for i in range(num_features):
        for j in range(num_features):
            cov_matrix[i][j] = cov_matrix[i][j] * stds[i] * stds[j]

    # means of features
    means = []
    for feature in features:
        means.append(stats[feature][0])

    # generate data
    data = np.random.multivariate_normal(means, cov_matrix, n)
    df = pd.DataFrame(data, columns=features)

    # make WAR (target variable)
    coefficients = {
        'xba': 0.1,
        'barrel_batted_rate': 0.1,
        'player_age': -0.1,  # older players regress; not perfect since young players can be
        'batting_avg': 0.1,
        'k_percent': -0.1,  # more strikeouts = lower WAR
        'bb_percent': 0.1,
        'on_base_plus_slg': 0.1,
        'Rbat+': 0.1,
        'WAR_lag1': 0.1,
        'WAR_lag2': 0.1,
        'WAR_lag3': 0.1
    }

    # calculate WAR
    war = []
    for i in range(n):
```

```
        value = 0
        for feature in features:
            value += df[feature][i] * coefficients[feature]
        noise = np.random.normal(0, 1)
        value += noise
        war.append(value)

    df['WAR'] = war

    return df
```

Feature Ranges: Using the mean and standard deviations of the key columns of the complete dataset using complete_dataset.describe().

Correlation: Pairwise correlations (0.0, 0.5, 0.99) applied to all predictors.

WAR Generation: Coefficients reflect realistic contributions (e.g., strong AR_lag1). Noise mimics real-world variability (e.g., 2020 season).

WAR = Sum(X + c), where

coefficient matrix = [0.1, 0.1, −0.1, 0.1, −0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

X = Multivariate Normal(mean vector (from dataset.describe(), covariance matrix)

c = noise. follows standard normal distribution

Simulation Loop

Generate 10 datasets per correlation level, split data (70% train, 30% test), fit models, and compute MSE. Ridge uses parameter alpha=0.1.

```
results = []
correlations = [['No Corr', 0.0], ['Mild Corr', 0.5], ['High Corr', 0.99]]
models = {
    'Linear': LinearRegression(),
    'RandomForest': RandomForestRegressor(n_estimators=200, max_depth=12),
    'Ridge': Ridge(alpha=0.1)
}

# Run simulation
for label, rho in correlations:
    # 10 datasets for each correlation value
    for i in range(10):
```

```
        # generate data
        df = generate_data(n=100, correlation=rho)

        # Split data into features and target
        X = df[['xba', 'barrel_batted_rate', 'player_age', 'batting_avg',
                'k_percent', 'bb_percent', 'on_base_plus_slg', 'Rbat+',
                'WAR_lag1', 'WAR_lag2', 'WAR_lag3']]
        y = df['WAR']

        # train test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                            random_state=i)

        # scale
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # fit and predict for all models
        for model_name, model in models.items():
            model.fit(X_train_scaled, y_train)
            y_pred = model.predict(X_test_scaled)

            # Calculate MSE and R2
            mse = mean_squared_error(y_test, y_pred)
            r2 = r2_score(y_test, y_pred)

            results.append({
                'Model': model_name,
                'Correlation': label,
                'MSE': mse,
                'R2': r2
            })
```

Notes:

Ridge Regression: alpha=0.1

Random Forest: max_depth=12.

Scaling: Applied for all models

Results Summary

Summarize MSE mean and standard deviation for each model-correlation pair. We are evaluating the MSE (mean squared error) and r^2

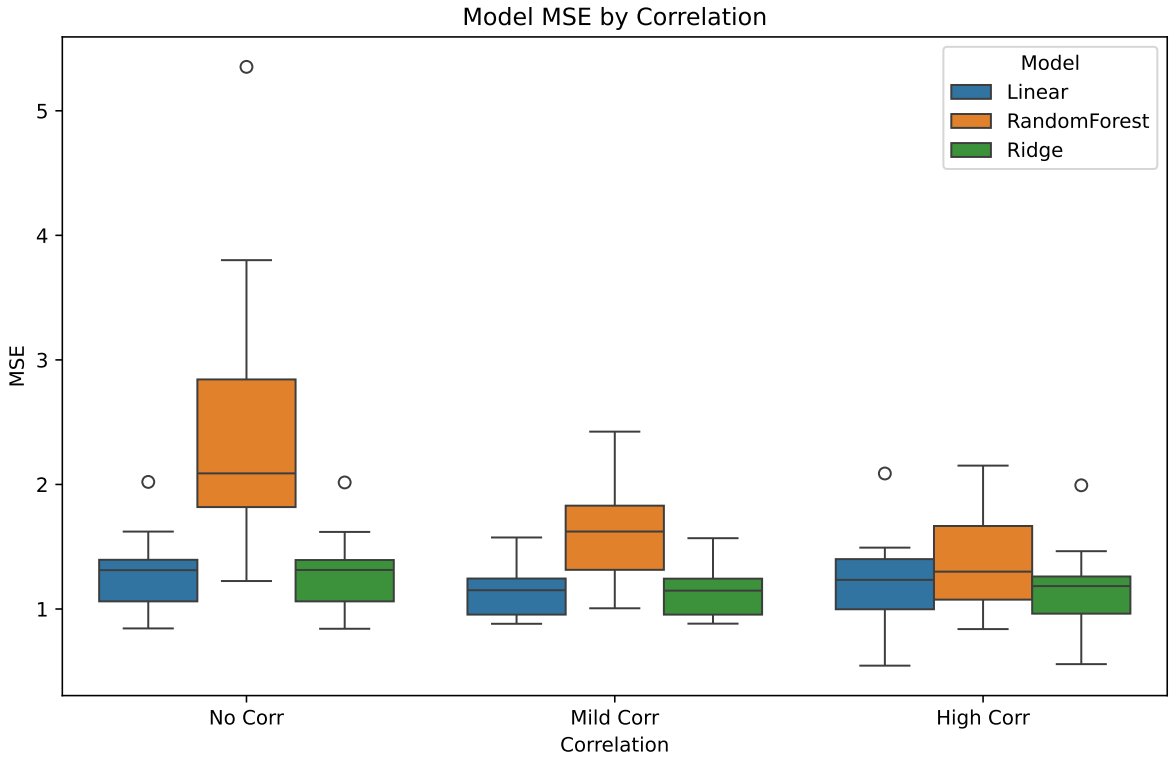
```
results_df = pd.DataFrame(results)

# Summarize results
summary = []
for model_name in ['Linear', 'RandomForest', 'Ridge']:
    for label in ['No Corr', 'Mild Corr', 'High Corr']:
        # Filter data
        temp_df = results_df[(results_df['Model'] == model_name) &
                             (results_df['Correlation'] == label)]

        # calculate mean and std
        mse_mean = temp_df['MSE'].mean()
        mse_std = temp_df['MSE'].std()
        r2_mean = temp_df['R2'].mean()
        r2_std = temp_df['R2'].std()

        summary.append({
            'Model': model_name,
            'Correlation': label,
            'Mean_MSE': mse_mean,
            'SD_MSE': mse_std,
            'Mean_R2': r2_mean,
            'SD_R2': r2_std
        })
summary_df = pd.DataFrame(summary)
print(summary_df)
```

```
          Model Correlation  Mean_MSE    SD_MSE   Mean_R2     SD_R2
0        Linear     No Corr  1.306118  0.336878  0.833703  0.065836
1        Linear   Mild Corr  1.152046  0.227160  0.862205  0.033649
2        Linear   High Corr  1.234598  0.406266  0.842628  0.044997
3  RandomForest     No Corr  2.554535  1.228775  0.701375  0.088503
4  RandomForest   Mild Corr  1.619820  0.409421  0.807488  0.046269
5  RandomForest   High Corr  1.391210  0.420326  0.819926  0.057057
6         Ridge     No Corr  1.304977  0.335870  0.833885  0.065663
7         Ridge   Mild Corr  1.150055  0.224519  0.862439  0.033460
8         Ridge   High Corr  1.179741  0.378526  0.849863  0.039144
```

Visualization

Create a boxplot to visualize MSE distributions.

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Correlation', y='MSE', hue='Model', data=results_df)
plt.title('Model MSE by Correlation')
plt.show()
```



We can now compare the three models numerically and visually.

1. Linear Regression obtains its lowest Mean_MSE and SD_MSE values where there is mild correlation (correlation: 0.5).
2. Random Forest has the highest MSE values overall but it obtains its lowest values when there is high correlation (correlation: 0.99).
3. Ridge Regression has very similar numbers as Linear Regression but performs slightly better than Linear Regression due to its L2 regularization. The gap between ridge and linear becomes larger as there is more correlation.

Correlation did not affect the MSE for the two linear regression models as much as it did the Random Forest. All three models performed fairly similarly when there was high correlation.