```python
#Simulate Data with AR(1) Errors

import numpy as np


def simulate_regression_with_ar1_errors(n, beta0, beta1, rho, sigma):
    """
    Simulate data for a regression model with AR(1) errors.

    Parameters:
    n (int): Number of observations.
    beta0 (float): Intercept coefficient.
    beta1 (float): Slope coefficient.
    rho (float): Autoregressive parameter (serial correlation).
    sigma (float): Standard deviation of the errors.

    Returns:
    Tuple (X, y): Simulated data where X is the predictor variable and
y is the response variable.
    """
    # Generate predictor variable
    X = np.random.normal(size=n)
    # Generate errors
    errors = np.zeros(n)
    errors[0] = np.random.normal(scale=sigma)  # Initialize the first
error term
    for t in range(1, n):
        errors[t] = rho * errors[t-1] + np.random.normal(scale=sigma)

    # Generate response variable
    y = beta0 + beta1 * X + errors

    return X, y

# Parameters
n = 1000  # Number of observations
beta0 = 1  # Intercept coefficient
beta1 = 0.5  # Slope coefficient
rho = 0.8  # Autoregressive parameter
sigma = 1  # Standard deviation of the errors

X, y = simulate_regression_with_ar1_errors(n, beta0, beta1, rho,
sigma)

#Estimate Coefficients
from statsmodels.regression.linear_model import OLS
from statsmodels.tools.tools import add_constant

model = OLS(y, add_constant(X)).fit()
beta_hat = model.params
```

```
#Calculate bootstrap standard errors

def bootstrap_standard_errors(X, y, beta_hat, num_resamples=1000):
    """

    Parameters:
    X (ndarray): Predictor variable.
    y (ndarray): Response variable.
    beta_hat (ndarray): Estimated coefficients.
    num_resamples (int): Number of bootstrap resamples.

    Returns:
    ndarray: Bootstrap standard errors for the coefficients.
    """
    n = len(y)
    num_coeffs = beta_hat.shape[0]
    beta_boot_se = np.zeros(num_coeffs)

    for i in range(num_resamples):
        resample_indices = np.random.choice(range(n), size=n,
replace=True)
        X_resample = X[resample_indices]
        y_resample = y[resample_indices]
        model = OLS(y_resample, add_constant(X_resample)).fit()
        beta_boot_se += (model.params - beta_hat) ** 2
    beta_boot_se = np.sqrt(beta_boot_se / num_resamples)
    return beta_boot_se

# Bootstrap standard errors
beta_boot_se = bootstrap_standard_errors(X, y, beta_hat)

from scipy.stats import norm

# Calculate confidence intervals
z_critical = norm.ppf(0.975)
ci_lower = beta_hat[1] - z_critical * beta_boot_se[1]
ci_upper = beta_hat[1] + z_critical * beta_boot_se[1]
print("95% Confidence Interval for Beta 1:", (ci_lower, ci_upper))
```

```
95% Confidence Interval for Beta 1: (0.39434589021550964,
0.6088141801651503)
```

```
#Define Simulation Function for Monte Carlo Simulations
def run_simulation(T, beta0, beta1, rho, sigma, z_critical):
    X_sim, y_sim = simulate_regression_with_ar1_errors(T, beta0,
beta1, rho, sigma)
    model_sim = OLS(y_sim, add_constant(X_sim)).fit()
    beta_hat_sim = model_sim.params
    beta_boot_se_sim = bootstrap_standard_errors(X_sim, y_sim,
```

```python
beta_hat_sim)

    ci_lower_sim = beta_hat_sim[1] - z_critical * beta_boot_se_sim[1]
    ci_upper_sim = beta_hat_sim[1] + z_critical * beta_boot_se_sim[1]

    return ci_lower_sim <= beta1 <= ci_upper_sim

from joblib import Parallel, delayed

# Monte Carlo simulations
T_values = [100, 500]
num_runs = 1000

for T in T_values:
    results = Parallel(n_jobs=-1)(
        delayed(run_simulation)(T, beta0, beta1, rho, sigma,
z_critical) for _ in range(num_runs)
    )
    coverage = sum(results) / num_runs
    print(f"Empirical Coverage for T={T}: {coverage}")
```

```
Empirical Coverage for T=100: 0.937
Empirical Coverage for T=500: 0.953
```