In [44]:

```python
#Q1
import numpy as np
import pandas as pd
from scipy import stats

def simCorn(overallEffect = 0, fertilizerEffect = [0,0,0], rowEffect = [0,0,
            colEffect = [0,0,0], seed = None, dist = stats.norm.rvs, **kwarg
    # Error checking
    if len(fertilizerEffect) != 3:
        raise ValueError("fertilizerEffect must be length 3")
    if len(rowEffect) != 3:
        raise ValueError("rowEffect must be length 3")
    if len(colEffect) != 3:
        raise ValueError("colEffect must be length 3")
    if seed is not None and not isinstance(seed, int):
        raise ValueError("seed must be an integer or None")

    # Set seed
    if seed is not None:
        np.random.seed(seed)

    # Latin square design
    data = pd.DataFrame({
        'Fertilizer': ['A', 'B', 'C', 'C', 'A', 'B', 'B', 'C', 'A'],
        'Row': [1, 1, 1, 2, 2, 2, 3, 3, 3],
        'Column': [1, 2, 3, 1, 2, 3, 1, 2, 3]
    })

    # Convert to categorical (but ensure mapping to numeric)
    data['Fertilizer'] = pd.Categorical(data['Fertilizer'], categories = ['A
    data['Row'] = pd.Categorical(data['Row'])
    data['Column'] = pd.Categorical(data['Column'])

    # Calculate fixed effects (convert to numeric)
    fert_map = {'A': fertilizerEffect[0], 'B': fertilizerEffect[1], 'C': fer
    row_map = {1: rowEffect[0], 2: rowEffect[1], 3: rowEffect[2]}
    col_map = {1: colEffect[0], 2: colEffect[1], 3: colEffect[2]}

    fert_effect = data['Fertilizer'].map(fert_map).astype(float)
    row_effect = data['Row'].map(row_map).astype(float)
    col_effect = data['Column'].map(col_map).astype(float)

    data['fixed'] = overallEffect + fert_effect + row_effect + col_effect

    # Generate errors
    errors = dist(size = 9, **kwargs)
    data['Yield'] = data['fixed'] + errors

    return data[['Fertilizer', 'Row', 'Column', 'Yield']]

# Example 3
mu = 7
alpha = [1, 2, 3]
beta = [2, 2, 1]
```

```
gamma = [3, 3, 2]
y3 = simCorn(overallEffect = mu, fertilizerEffect = alpha, rowEffect = beta,
             colEffect = gamma, seed = 29429, dist = stats.norm.rvs, loc = 3
print(y3.head())
```

```
  Fertilizer Row Column      Yield
0          A   1      1  16.836497
1          B   1      2  18.749914
2          C   1      3  20.136163
3          C   2      1  24.599134
4          A   2      2  18.519364
```

In [45]:

```
#Q3
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Function to run a single simulation
def run_simulation(sim_num, mu, alpha, beta, gamma, error_dist, seed_val, n_
    p_values = np.zeros(n_sims)

    for i in range(n_sims):
        # For each simulation, use a different seed taken from the base seed
        sim_seed = seed_val + i

        # Generate data based on simulation
        if error_dist == "normal":
            y = simCorn(overallEffect = mu,
                        fertilizerEffect = alpha,
                        rowEffect = beta,
                        colEffect = gamma,
                        seed = sim_seed,
                        dist = np.random.normal)
        elif error_dist == "exponential":
            y = simCorn(overallEffect = mu,
                        fertilizerEffect = alpha,
                        rowEffect = beta,
                        colEffect = gamma,
                        seed = sim_seed,
                        dist = np.random.exponential,
                        scale = 1)

        # Fit the model and extract p-value
        model = ols('Yield ~ C(Fertilizer) + C(Row) + C(Column)', data = y).
        anova_table = sm.stats.anova_lm(model, typ = 1)
        p_values[i] = anova_table.loc['C(Fertilizer)', 'PR(>F)']

    # Calculate percentage less than 0.05
    pct_sig = 100 * np.mean(p_values <= 0.05)

    # Create histogram
    plt.figure(figsize = (10, 6))
    plt.hist(p_values, bins = np.arange(0, 1.05, 0.05), color = 'blue', edge
    plt.axvline(x = 0.05, color = 'red', linewidth = 2)
```

```
    plt.title(f"Simulation {sim_num} ({int(pct_sig)}% Less Than 0.05)")
    plt.xlabel("p values")
    plt.ylabel("Frequency")
    plt.xlim(0, 1)
    plt.grid(alpha = 0.3)
    plt.show()

    return p_values

# Run all 7 simulations
# Simulation 1
sim1 = run_simulation(1, 10, [0, 0, 0], [0, 0, 0], [0, 0, 0], "normal", 1331

# Simulation 2
sim2 = run_simulation(2, 10, [1, 2, 3], [0, 0, 1], [0, 0, 1], "normal", 1869

# Simulation 3
sim3 = run_simulation(3, 10, [1, 2, 3], [1, 0, 1], [0, 1, 1], "normal", 6516

# Simulation 4
sim4 = run_simulation(4, 10, [1, 2, 3], [0, 1, 0], [0, 1, 0], "normal", 5)

# Simulation 5
sim5 = run_simulation(5, 10, [1, 2, 3], [0, 0, 1], [0, 0, 1], "exponential",

# Simulation 6
sim6 = run_simulation(6, 10, [1, 2, 3], [1, 0, 1], [0, 1, 1], "exponential",

# Simulation 7
sim7 = run_simulation(7, 10, [1, 2, 3], [0, 1, 0], [0, 1, 0], "exponential",
```