Questions

1. We will consider out-of-sample predictions at x0 = (2, 2, . . . , 2). What value of f(x0) do you expect?

Because $x_0 = (2, 2, \dots)$ and $\beta = [2, 3] + [0] * (p - 2)$, we expect the value of $f(x_0)$ to be as follows:

$$\beta_1 * 2 + \beta_2 * 3 = 2 * 2 + 3 * 2 = 4 + 6 = 10 = f(x_0)$$

In the rest of this section, we convert the code to python. We start by importing the necessary libraries.

In [9]:
```
# Necessary imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import LassoCV
from sklearn.linear_model import RidgeCV
from sklearn.metrics import mean_squared_error
import seaborn as sns
```

In [10]:
```
# Define empty dictionaries
OLS = {"prediction":[], "coefficient":[], "bias":[], "variance":[], "MSE":[]}
LASSO = {"prediction":[], "coefficient":[], "alpha":[], "bias":[], "variance":[], "MSE":[]}
RIDGE = {"prediction":[], "coefficient":[], "alpha":[], "bias":[], "variance":[], "MSE":[]}

# Simulation parameters
nsim = 100  # Number of simulations
nobs = 100  # Number of observations in each simulation
p = 10      # Number of predictors
beta = np.array([2, 3] + [0] * (p - 2))  # Coefficients: β1=2, β2=3, others=0

x0 = np.full(p, 2)  # Out-of-sample observation
# Prepare x0 for prediction (add constant and reshape for prediction)
x0 = np.insert(x0, 0, 1)  # Add `1` for the Intercept
x0 = x0.reshape(1, -1)  # Convertation to an array

# Mittelwertvektor und Kovarianzmatrix erstellen
mu = np.zeros(p)  # Erzeugt einen Vektor mit Nullen der Länge p
Sigma = np.eye(p)  # Erzeugt eine p x p Einheitsmatrix

# Function to simulate data
def simulate_data(nobs, p, beta, Sigma):
    X = np.random.multivariate_normal(mu, Sigma, nobs)
    noise = np.random.normal(0, 1, nobs)
    y = X @ beta + noise
    return X, y
```

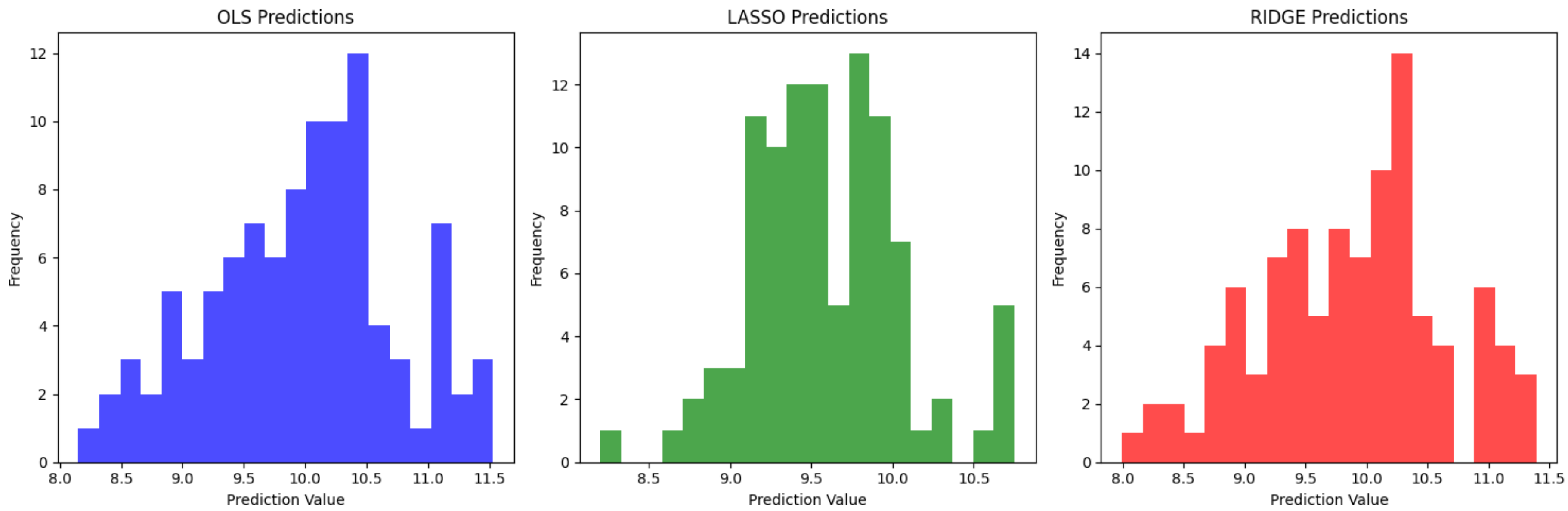
In [11]:
```
# Create the first predictions for each model
for _ in range(0,nsim):
    X_example, y_example = simulate_data(nobs, p, beta, Sigma)
    X_example = sm.add_constant(X_example)
    fitOLS = sm.OLS(y_example,X_example)
    model = fitOLS.fit()
    prediction = model.predict(x0)
    OLS["prediction"].append(prediction)
    OLS["coefficient"].append(model.params)

    fitLASSO = LassoCV(cv=5, random_state=42).fit(X_example,y_example)
    prediction = fitLASSO.predict(x0)
    LASSO["prediction"].append(prediction)
    LASSO["coefficient"].append(fitLASSO.coef_)
    LASSO["alpha"].append(fitLASSO.alpha_)

    fitRidge = RidgeCV(cv=5).fit(X_example,y_example)
    prediction = fitRidge.predict(x0)
    RIDGE["prediction"].append(prediction)
    RIDGE["coefficient"].append(fitRidge.coef_)
    RIDGE["alpha"].append(fitRidge.alpha_)
```

2) Run the code as is. Plot the distribution of predictions f(x0) for the different models. What do you observe?

In the below section, we plot the distributions of the three models by using histogrammes. Our results show all models share the same range of 8 to 12 in their predictions. We notice that the OLS and Ridge models display very similar distributions and share a higher variance than LASSO. The OLS has the best distribution around the mean. The other models have a small skewness.

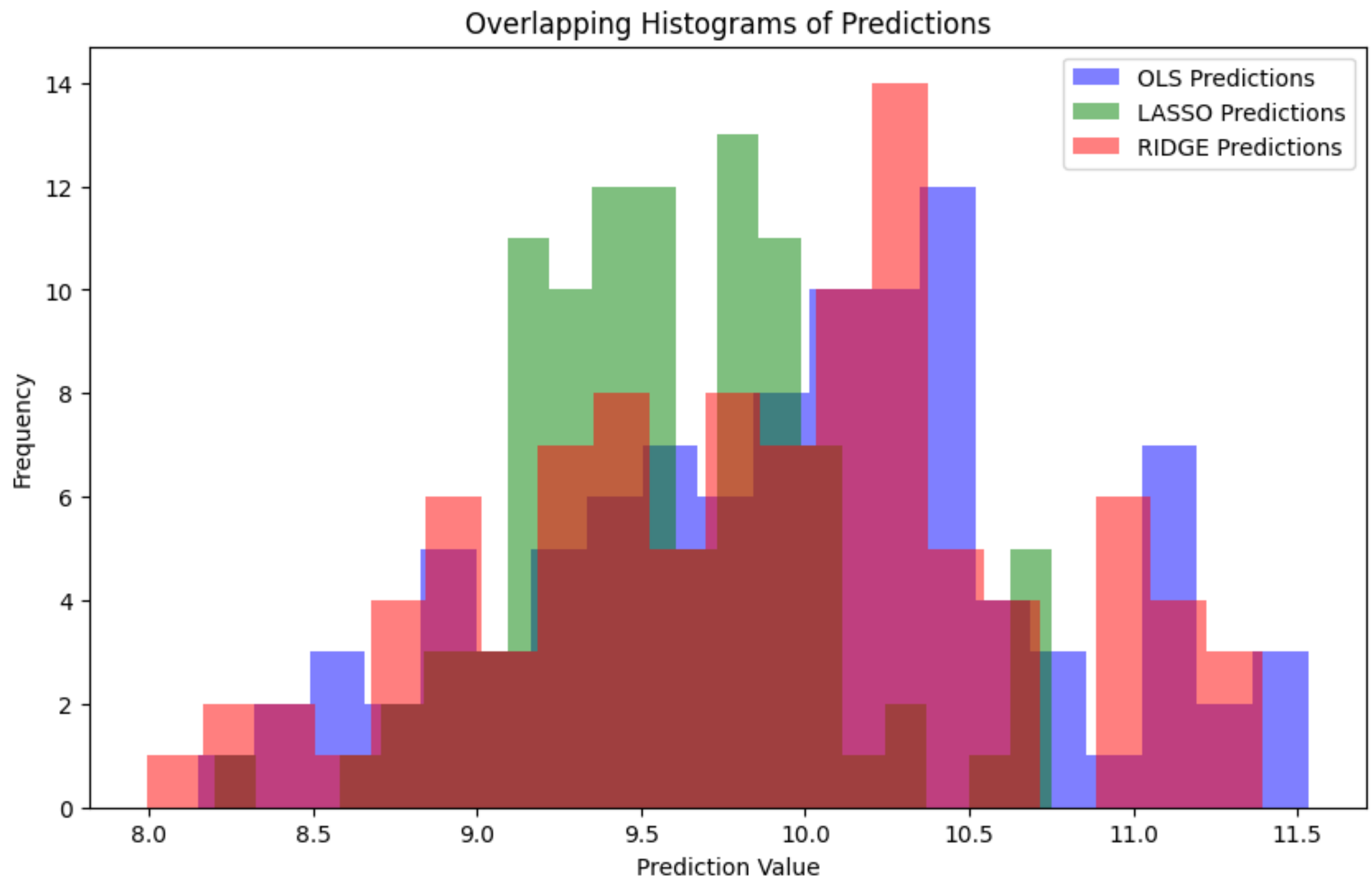
In [12]:
```
# Create flat array for the predictions
OLS["prediction"] = np.concatenate(OLS['prediction'])
```

```python
LASSO['prediction'] = np.concatenate(LASSO['prediction'])
RIDGE['prediction'] = np.concatenate(RIDGE['prediction'])

# Create histogram
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

# OLS
axes[0].hist(OLS["prediction"], bins=20, color='blue', alpha=0.7)
axes[0].set_title('OLS Predictions')
axes[0].set_xlabel('Prediction Value')
axes[0].set_ylabel('Frequency')

# LASSO
axes[1].hist(LASSO['prediction'], bins=20, color='green', alpha=0.7)
axes[1].set_title('LASSO Predictions')
axes[1].set_xlabel('Prediction Value')
axes[1].set_ylabel('Frequency')

# RIDGE
axes[2].hist(RIDGE['prediction'], bins=20, color='red', alpha=0.7)
axes[2].set_title('RIDGE Predictions')
axes[2].set_xlabel('Prediction Value')
axes[2].set_ylabel('Frequency')


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



In [13]:

```python
# Plot combined histogram
plt.figure(figsize=(10, 6))

# OLS
plt.hist(OLS["prediction"], bins=20, color='blue', alpha=0.5, label='OLS Predictions')

# LASSO
plt.hist(LASSO['prediction'], bins=20, color='green', alpha=0.5, label='LASSO Predictions')

# RIDGE
plt.hist(RIDGE['prediction'], bins=20, color='red', alpha=0.5, label='RIDGE Predictions')


plt.title('Overlapping Histograms of Predictions')
plt.xlabel('Prediction Value')
plt.ylabel('Frequency')
plt.legend()


plt.show()
```



To make the next steps faster, we create a "perform_simulation" function that creates the predictions with the given parameters.

In [14]:
```
# Create mean vector and covariance matrix
mu = np.zeros(p)
Sigma = np.eye(p)

# Function to simulate data
def simulate_data(nobs, p, beta, Sigma):
    mu = np.zeros(p)
    X = np.random.multivariate_normal(mu, Sigma, nobs)
    noise = np.random.normal(0, 1, nobs)
    y = X @ beta + noise
    return X, y

# Function to compute bias, variance and MSE
def compute_metrics(predictions, true_value):
    predictions = np.array(predictions).flatten()
    bias = np.mean(predictions) - true_value
    variance = np.var(predictions)
    mse = mean_squared_error(np.full_like(predictions, true_value), predictions)
    return bias, variance, mse

# Function to create a correlation matrix for correlated and uncorrelated regressors. (For Question 6)
# Regressors are not correlated when rho = 0 and correlated when rho > 0
def create_correlated_cov_matrix(p, rho):
    cov_matrix = np.full((p, p), rho)  # Add rho
    np.fill_diagonal(cov_matrix, 1)
    return cov_matrix

# Create predictions
# The function takes the specified parameters and performs a simulation depending on these with 3 different models
def perform_simulation(nsim, nobs, p, rho):
    beta = np.array([2, 3] + [0] * (p - 2))  # Adjust beta for p
    x0 = np.insert(np.full(p, 2), 0, 1).reshape(1, -1)  # Adjust x0 for p
    true_value = 10  # Expected true value at x0 based on the given beta and x0
    mu = np.zeros(p)  # Vector of zeros of length p
    Sigma = create_correlated_cov_matrix(p,rho)  # p x p identity matrix

    # Data structures to store results
    results = {
        "model": [],
        "p": [],
        "bias": [],
        "variance": [],
        "MSE": [],
        "nsim": [],
        "nobs": [],
        "rho": []
    }

    # Simulation loop
    for model_name, Model in [("OLS", sm.OLS), ("Lasso", LassoCV), ("Ridge", RidgeCV)]:
        predictions = []
        for _ in range(nsim):
```

```
            X, y = simulate_data(nobs, p, beta, Sigma)
            X = sm.add_constant(X)  # Add constant column
            if model_name in ["Lasso", "Ridge"]:
                if model_name == "Lasso":
                    model = Model(cv=5, random_state=42).fit(X, y)
                else:
                    model = Model(cv=5).fit(X, y)
                predictions.append(model.predict(x0))
            else:
                model = Model(y, X).fit()
                predictions.append(model.predict(x0))

        # Calculate metrics
        bias, variance, mse = compute_metrics(predictions, true_value)

        # Save results
        results["model"].append(model_name)
        results["p"].append(p)
        results["bias"].append(bias)
        results["variance"].append(variance)
        results["MSE"].append(mse)
        results["nsim"].append(nsim)
        results["nobs"].append(nobs)
        results["rho"].append(rho)

    return pd.DataFrame(results)
```

3. Compute bias, variance and mean squared error for the three different models at x0, our out-of-sample observation.

In [15]:
```
df_metrics = perform_simulation(nsim, nobs, p, rho=0)
df_metrics
```

Out[15]:

	model	p	bias	variance	MSE	nsim	nobs	rho
0	OLS	10	0.096154	0.437263	0.446509	100	100	0
1	Lasso	10	-0.274627	0.292617	0.368037	100	100	0
2	Ridge	10	-0.189293	0.538932	0.574764	100	100	0

4. How do bias and variance depend on the number of irrelevant regressors?

4. To answer this question, we run our simulation with p values starting from 10 to 100 with intervals of 20.

As seen in the table below, we observe that:

- *Variance:*
  - In all models, the **variance** increases as the number of irrelevant regressors **increase**
  - **OLS and Ridge** face a significantly **higher increase** in variance in comparison to LASSO. This is in our opinion according to the fact, that LASSO performs a **feature selection** by setting the coefficients of irrelevant regressors to 0.
- *Bias:*
  - In all models, **bias** increases as the number of irrelevant regressors **increase.**
  - In **OLS**, the bias stays quite **constant**, whereas we see a **small increase** in bias in **LASSO**. This is in our opiniton because of the penalty term.
  - Surprisingly, **Ridge** suffers the **most** from this increase. In our opinion, this could be due to the fact that Ridge regression's penalty term becomes relatively stronger with larger **p** since Ridge regression becomes more aggressive in shrinking the coefficients towards zero, leading to a larger increase in bias compared to OLS.

In [20]:
```
df_results_1 = pd.DataFrame(columns=["model", "p", "bias", "variance", "MSE", "nsim", "nobs", "rho"])

for p_value in range(10, 100, 20):
    simulation_df = perform_simulation(100, 100, p_value, 0)
    df_results_1 = pd.concat([df_results_1, simulation_df], ignore_index=True)

df_results_1
```

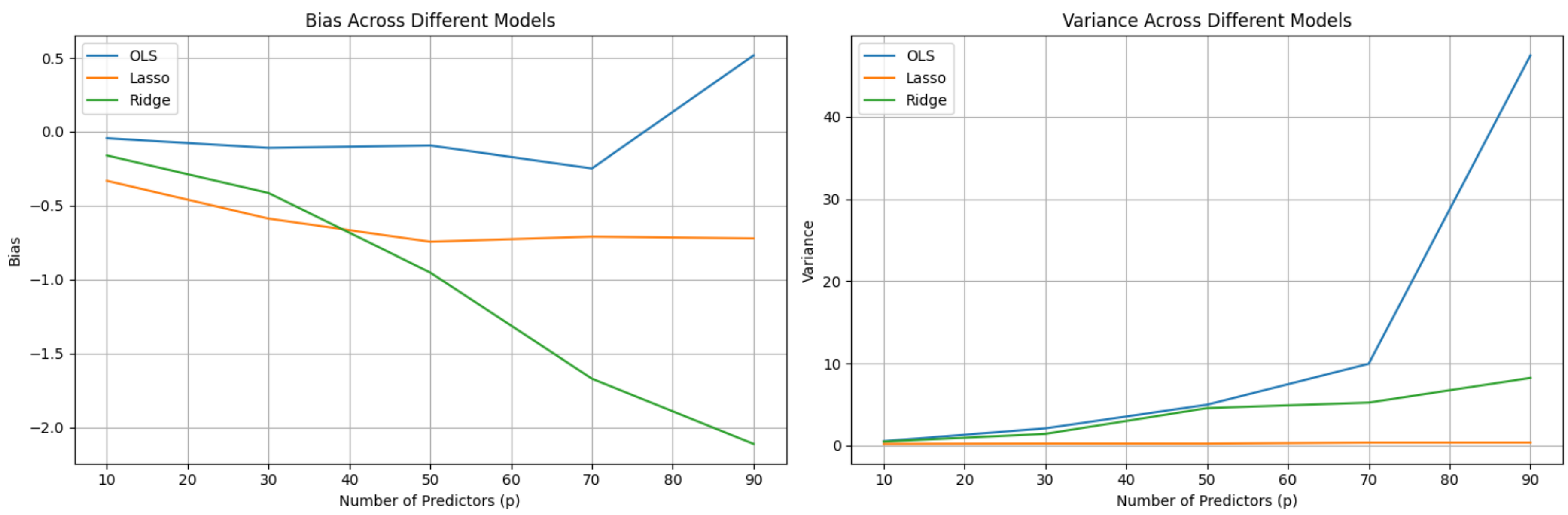
```
/var/folders/x4/q675smv52q76wfncl4h9040h0000gn/T/ipykernel_77839/1475746318.py:5: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.
  df_results_1 = pd.concat([df_results_1, simulation_df], ignore_index=True)
```

Out[20]:

	model	p	bias	variance	MSE	nsim	nobs	rho
0	OLS	10	-0.044080	0.512238	0.514181	100	100	0
1	Lasso	10	-0.331577	0.180274	0.290218	100	100	0
2	Ridge	10	-0.160331	0.460373	0.486080	100	100	0
3	OLS	30	-0.109719	2.079827	2.091865	100	100	0
4	Lasso	30	-0.587750	0.220900	0.566350	100	100	0
5	Ridge	30	-0.414334	1.404949	1.576622	100	100	0
6	OLS	50	-0.093274	4.961124	4.969824	100	100	0
7	Lasso	50	-0.744505	0.213235	0.767523	100	100	0
8	Ridge	50	-0.952183	4.546264	5.452916	100	100	0
9	OLS	70	-0.248468	9.943778	10.005514	100	100	0
10	Lasso	70	-0.709972	0.342300	0.846361	100	100	0
11	Ridge	70	-1.670282	5.215164	8.005005	100	100	0
12	OLS	90	0.516789	47.477000	47.744071	100	100	0
13	Lasso	90	-0.721965	0.345541	0.866775	100	100	0
14	Ridge	90	-2.112053	8.222862	12.683628	100	100	0

In [21]:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# Plt for Bias
for model in df_results_1['model'].unique():
    df_model = df_results_1[df_results_1['model'] == model]
    ax1.plot(df_model['p'], df_model['bias'], label=model)

ax1.set_xlabel('Number of Predictors (p)')
ax1.set_ylabel('Bias')
ax1.set_title('Bias Across Different Models')
ax1.legend()
ax1.grid(True)

# Plt for Variance
for model in df_results_1['model'].unique():
    df_model = df_results_1[df_results_1['model'] == model]
    ax2.plot(df_model['p'], df_model['variance'], label=model)

ax2.set_xlabel('Number of Predictors (p)')
ax2.set_ylabel('Variance')
ax2.set_title('Variance Across Different Models')
ax2.legend()
ax2.grid(True)


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



5. How do bias and variance depend on the number of observations?

- *Bias*:
  - We observe that bias **becomes constant/stabilizes itself** as we increase the number of total observations.

This effect can be seen on the line chart, around the breakpoint of **300 observations** the changes in bias become much smaller.

- *Variance*:
  - We observe that variance **decreases** as we increase the number of total observations. This effect can be seen on the line

chart, the variance **decreases rapidly** until the breakpoint of **300 observations** and **slower** after this point.

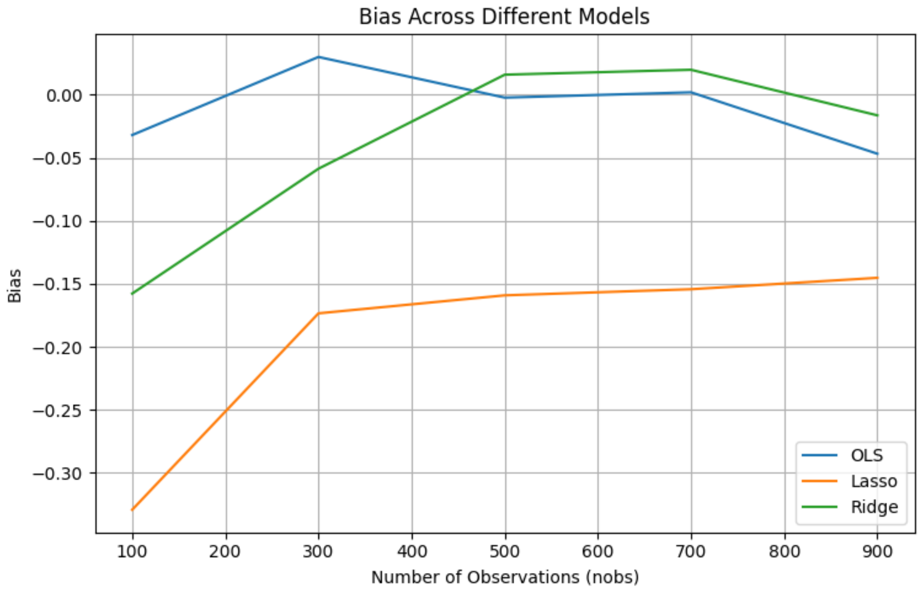
In [16]:
```
df_results_2 = pd.DataFrame(columns=["model", "p", "bias", "variance", "MSE", "nsim", "nobs", "rho"])

for nobs in range(100, 1000, 200):
    simulation_df = perform_simulation(100, nobs, 10, 0)
    df_results_2 = pd.concat([df_results_2, simulation_df], ignore_index=True)

df_results_2
```

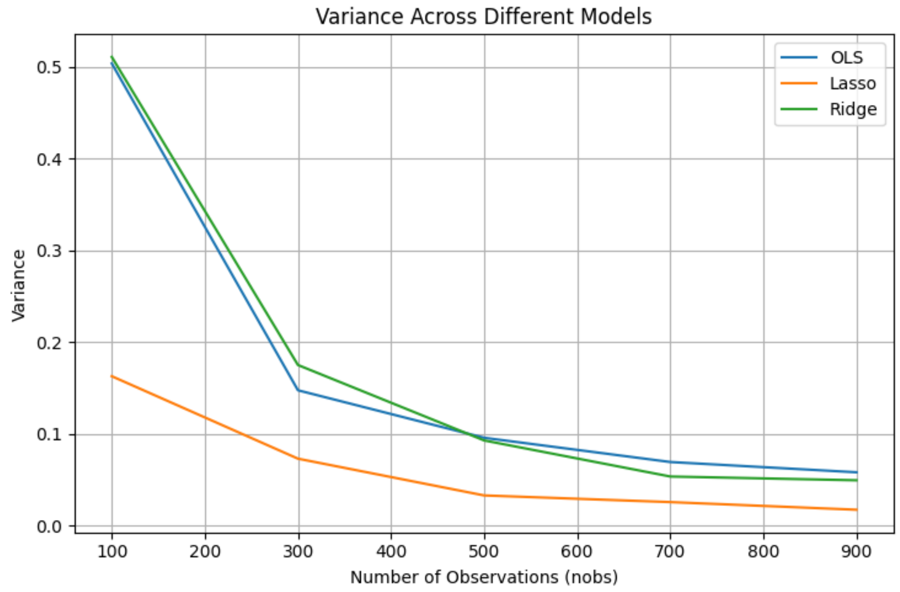
```
/var/folders/x4/q675smv52q76wfncl4h9040h0000gn/T/ipykernel_77839/2025371750.py:5: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.
  df_results_2 = pd.concat([df_results_2, simulation_df], ignore_index=True)
```

Out[16]:

	model	p	bias	variance	MSE	nsim	nobs	rho
0	OLS	10	-0.031964	0.503386	0.504408	100	100	0
1	Lasso	10	-0.329157	0.162757	0.271101	100	100	0
2	Ridge	10	-0.157701	0.510472	0.535342	100	100	0
3	OLS	10	0.029948	0.147455	0.148352	100	300	0
4	Lasso	10	-0.173384	0.072973	0.103035	100	300	0
5	Ridge	10	-0.058677	0.174976	0.178419	100	300	0
6	OLS	10	-0.002313	0.095727	0.095732	100	500	0
7	Lasso	10	-0.159113	0.032934	0.058251	100	500	0
8	Ridge	10	0.015900	0.092839	0.093092	100	500	0
9	OLS	10	0.001907	0.069329	0.069332	100	700	0
10	Lasso	10	-0.154271	0.025669	0.049468	100	700	0
11	Ridge	10	0.019784	0.053536	0.053928	100	700	0
12	OLS	10	-0.046788	0.058144	0.060333	100	900	0
13	Lasso	10	-0.145225	0.017352	0.038442	100	900	0
14	Ridge	10	-0.016351	0.049422	0.049689	100	900	0

In [17]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
# Plt for Bias
for model in df_results_2['model'].unique():
    df_model = df_results_2[df_results_2['model'] == model]
    ax1.plot(df_model['nobs'], df_model['bias'], label=model)

ax1.set_xlabel('Number of Observations (nobs)')
ax1.set_ylabel('Bias')
ax1.set_title('Bias Across Different Models')
ax1.legend()
ax1.grid(True)

# Plt for Variance
for model in df_results_2['model'].unique():
    df_model = df_results_2[df_results_2['model'] == model]
    ax2.plot(df_model['nobs'], df_model['variance'], label=model)

ax2.set_xlabel('Number of Observations (nobs)')
ax2.set_ylabel('Variance')
ax2.set_title('Variance Across Different Models')
ax2.legend()
ax2.grid(True)


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





6. The simulation above assumes that regressors are uncorrelated. How do results change when correlation between regressors is instead given by ρ > 0?

As the correlation of the parameters β increases, it can be seen that bias and variance decrease. The bias remains constant over the changed correlation, indicating that the model is correctly specified. From the left chart, we can observe that:

- The bias for OLS fluctuates but overall tends to perform better with the correlation ratio.
- Lasso and Ridge regression have higher bias when there is no correlation among regressors (*rho* = 0), but as *rho* increases, their biases decrease and then plateau.

From the right chart, it is clear that:

- The variance for OLS drops significantly as the correlation between regressors increases.
- Lasso and Ridge start with lower variance compared to OLS at *rho* = 0 and continue to decrease at a steadier rate as the correlation increases.

Note: We expected that the variance would increase with a higher *rho* and bias to stay relative constant.

In [18]:
```
df_results_3 = pd.DataFrame(columns=["model", "p", "bias", "variance", "MSE", "nsim", "nobs", "rho"])

for rho_value in range(0, 100, 10):
    rho_value = rho_value / 100
    simulation_df = perform_simulation(100, 100, 10, rho_value)
    df_results_3 = pd.concat([df_results_3, simulation_df], ignore_index=True)

df_results_3
```

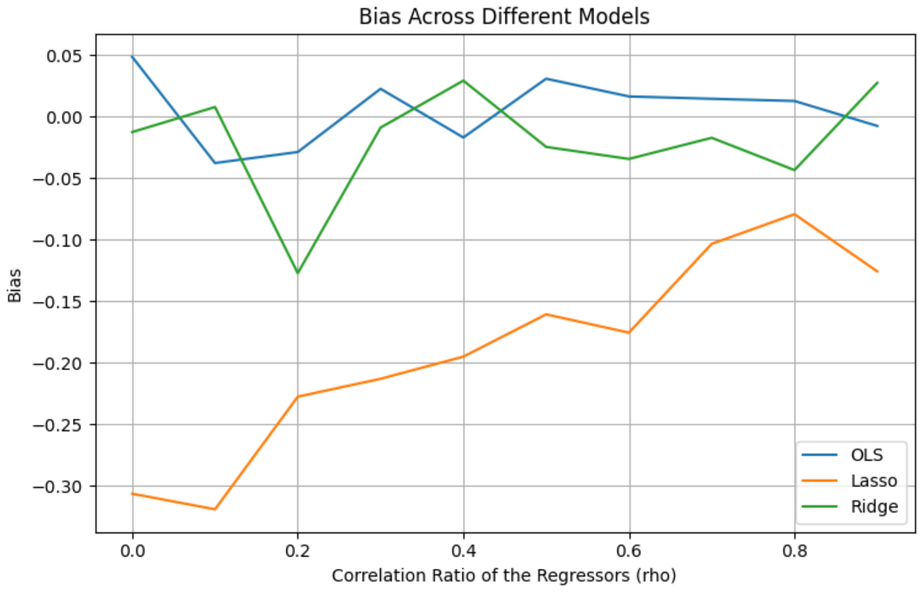
```
/var/folders/x4/q675smv52q76wfncl4h9040h0000gn/T/ipykernel_77839/3958436154.py:6: FutureWarning: The behavior of Dat
aFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude em
pty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries b
efore the concat operation.
  df_results_3 = pd.concat([df_results_3, simulation_df], ignore_index=True)
```

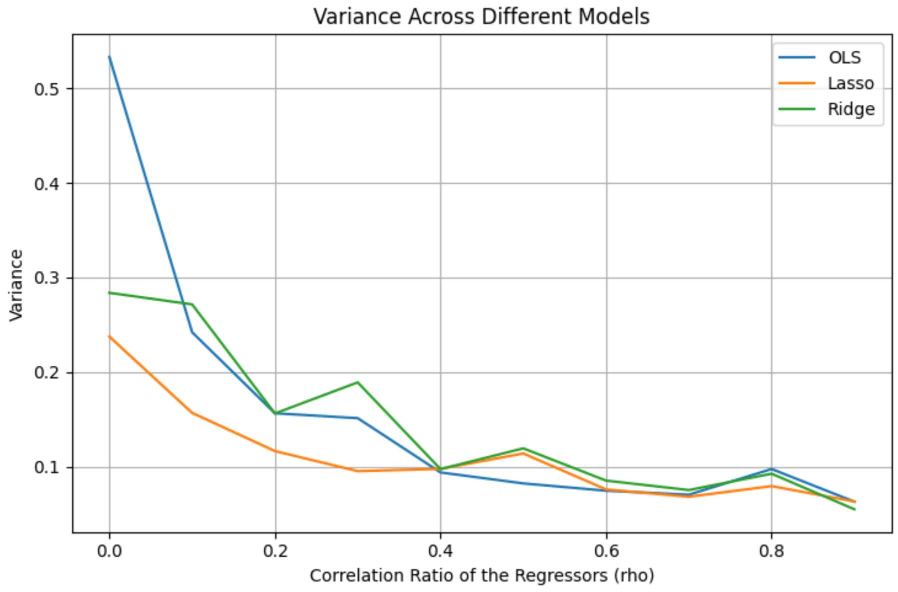
Out[18]:

	model	p	bias	variance	MSE	nsim	nobs	rho
0	OLS	10	0.048379	0.533318	0.535658	100	100	0.0
1	Lasso	10	-0.306617	0.237438	0.331452	100	100	0.0
2	Ridge	10	-0.012719	0.283713	0.283875	100	100	0.0
3	OLS	10	-0.037876	0.242024	0.243458	100	100	0.1
4	Lasso	10	-0.319301	0.156693	0.258646	100	100	0.1
5	Ridge	10	0.007662	0.271495	0.271553	100	100	0.1
6	OLS	10	-0.028909	0.156286	0.157121	100	100	0.2
7	Lasso	10	-0.227746	0.116289	0.168158	100	100	0.2
8	Ridge	10	-0.127372	0.156070	0.172294	100	100	0.2
9	OLS	10	0.022436	0.151101	0.151604	100	100	0.3
10	Lasso	10	-0.213224	0.095048	0.140512	100	100	0.3
11	Ridge	10	-0.009078	0.188963	0.189045	100	100	0.3
12	OLS	10	-0.017115	0.093686	0.093979	100	100	0.4
13	Lasso	10	-0.195218	0.097470	0.135580	100	100	0.4
14	Ridge	10	0.029090	0.097193	0.098039	100	100	0.4
15	OLS	10	0.030665	0.082063	0.083003	100	100	0.5
16	Lasso	10	-0.160842	0.113776	0.139646	100	100	0.5
17	Ridge	10	-0.024770	0.119176	0.119789	100	100	0.5
18	OLS	10	0.016240	0.074349	0.074613	100	100	0.6
19	Lasso	10	-0.175723	0.075678	0.106557	100	100	0.6
20	Ridge	10	-0.034555	0.085029	0.086223	100	100	0.6
21	OLS	10	0.014350	0.070132	0.070338	100	100	0.7
22	Lasso	10	-0.103558	0.067949	0.078674	100	100	0.7
23	Ridge	10	-0.017359	0.075057	0.075358	100	100	0.7
24	OLS	10	0.012556	0.097297	0.097454	100	100	0.8
25	Lasso	10	-0.079442	0.079223	0.085534	100	100	0.8
26	Ridge	10	-0.043621	0.092339	0.094242	100	100	0.8
27	OLS	10	-0.007721	0.062706	0.062766	100	100	0.9
28	Lasso	10	-0.125960	0.062944	0.078809	100	100	0.9
29	Ridge	10	0.027349	0.054660	0.055408	100	100	0.9

In [19]:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# Plt for Bias
for model in df_results_3['model'].unique():
    df_model = df_results_3[df_results_3['model'] == model]
    ax1.plot(df_model['rho'], df_model['bias'], label=model)

ax1.set_xlabel('Correlation Ratio of the Regressors (rho)')
ax1.set_ylabel('Bias')
ax1.set_title('Bias Across Different Models')
ax1.legend()
ax1.grid(True)

# Plt for Variance
for model in df_results_3['model'].unique():
    df_model = df_results_3[df_results_3['model'] == model]
    ax2.plot(df_model['rho'], df_model['variance'], label=model)

ax2.set_xlabel('Correlation Ratio of the Regressors (rho)')
ax2.set_ylabel('Variance')
ax2.set_title('Variance Across Different Models')
ax2.legend()
ax2.grid(True)


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


Bias Across Different Models
Bias
Correlation Ratio of the Regressors (rho)
OLS
Lasso
Ridge
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
0.0
0.2
0.4
0.6
0.8


Variance Across Different Models
Variance
Correlation Ratio of the Regressors (rho)
OLS
Lasso
Ridge
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8