Statistical learning and data analysis - Exercise 3 (Regression models)

```
In [ ]: from google.colab import drive
        drive.mount('/content/drive')
```

```
In [ ]: import pandas as pd
        import os
        import numpy as np
        import matplotlib.pyplot as plt
        import ssl
        import statsmodels.api as sm
        import statsmodels.formula.api as smf
        import seaborn as sns
        from sklearn.linear_model import LinearRegression, Ridge, Lasso, RidgeCV, LassoCV
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error
        from sklearn.model_selection import KFold
        import random
```

Question 3 - Analysis on regressions with regularization

Load the data

```
In [ ]: df = pd.read_csv('/content/drive/MyDrive/dataset.csv')
        cols = list(df.columns)
        plt.rcParams['figure.figsize'] = (18,9) # for plot size
```

Linear regression model on all the data, keeping the beta and p-value vectors in lists

```
In [ ]: x = df[cols[1:]]
        x_ols = sm.add_constant(x) # intercept vector
        y = pd.DataFrame(df["y"])
        model = sm.OLS(y, x_ols).fit()

        ols_pvals = list(model.pvalues.round(4))[1:]
        ols_beta = list(model.params.round(4))[1:]
```

Finding the optimal Lambda that minimize the MSE risk for both laso and ridge models.

First, we split the data into training and testing sets, then train the Ridge and Lasso models on the training set.

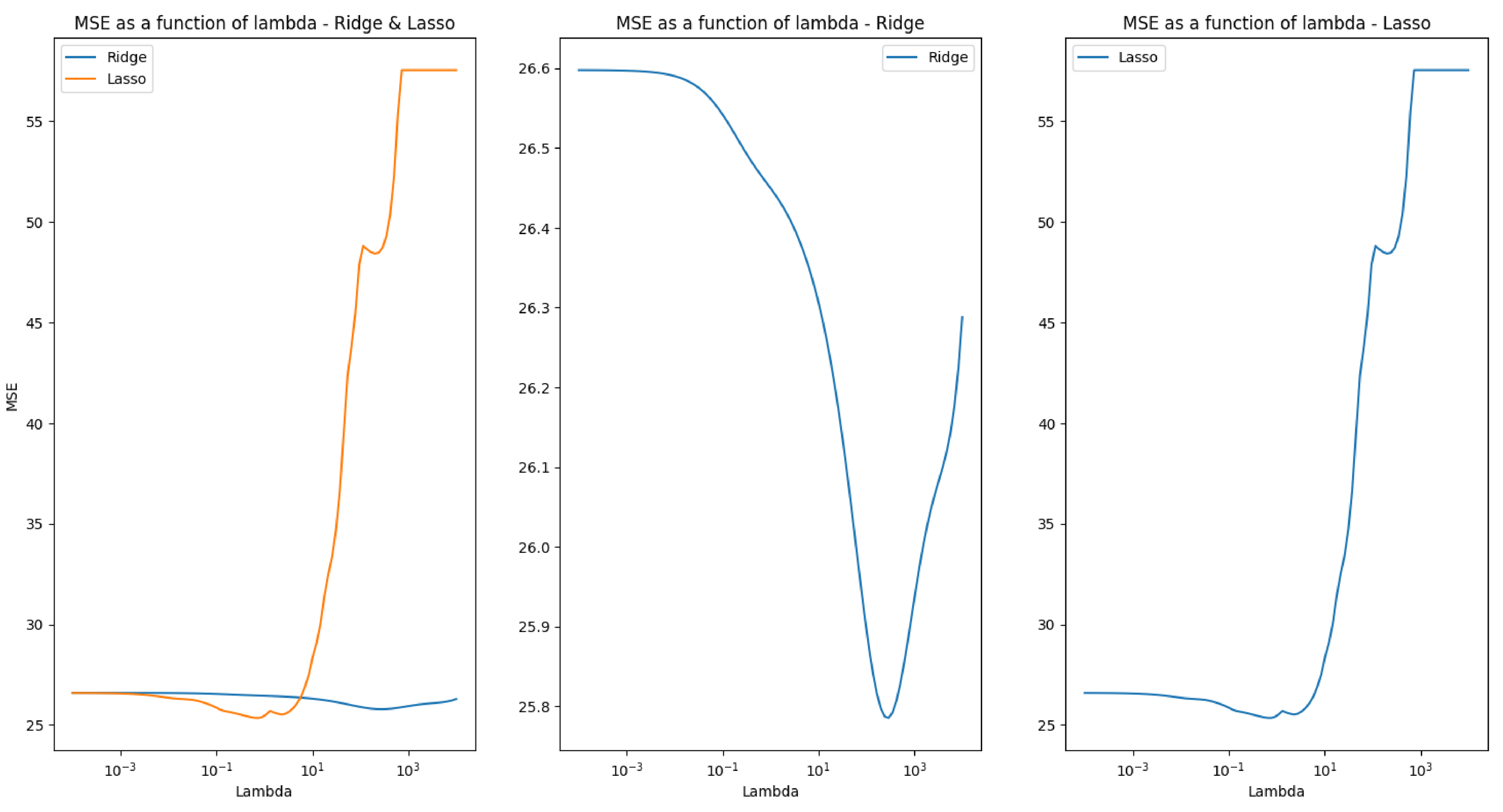
Then, we compute the mean squared error of each model predictions on the testing set and find the lambda value that gives the lowest MSE for each model (np.argmin).

```
In [ ]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

        ridge_mse = []
        lasso_mse = []
        lambdas = np.logspace(-4, 4, 100)

        for l in lambdas:

            ridge = Ridge(alpha=l).fit(x_train, y_train)
            lasso = Lasso(alpha=l).fit(x_train, y_train)

            ridge_mse.append(mean_squared_error(y_test, ridge.predict(x_test)))
            lasso_mse.append(mean_squared_error(y_test, lasso.predict(x_test)))

        ridge_optimal_lambda = lambdas[np.argmin(ridge_mse)]
        lasso_optimal_lambda = lambdas[np.argmin(lasso_mse)]
```

Now, we plot the MSE as a function of lambda in each model seperatly and both in same plot in a 1x3 grid to better see the differences, as well as the optimal lambda.

```
In [ ]: fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(18, 9))


        axs[0].plot(lambdas, ridge_mse, label='Ridge')
        axs[0].plot(lambdas, lasso_mse, label='Lasso')
        axs[0].set_xscale('log')
        axs[0].set_xlabel('Lambda')
        axs[0].set_ylabel('MSE')
        axs[0].set_title('MSE as a function of lambda - Ridge & Lasso')
        axs[0].legend()

        axs[1].plot(lambdas, ridge_mse, label='Ridge')
        axs[1].set_xscale('log')
        axs[1].set_xlabel('Lambda')
        axs[1].set_title('MSE as a function of lambda - Ridge')
        axs[1].legend()

        axs[2].plot(lambdas, lasso_mse, label='Lasso')
        axs[2].set_xscale('log')
        axs[2].set_xlabel('Lambda')
        axs[2].set_title('MSE as a function of lambda - Lasso')
        axs[2].legend()

        print(f"\nRidge Optimal Lambda: {ridge_optimal_lambda}\n\nLasso Optimal Lambda: {lasso_optimal_lambda}\n")
        plt.show()
```

```
Ridge Optimal Lambda: 291.5053062825182

Lasso Optimal Lambda: 0.7564633275546291
```



For both models, in the MSE as a function of lambda plot, we expect to see a U-shaped line in the plot that shows that as lambda increases, the MSE decreases until it reaches the optimal lambda value, after which the MSE increases again due to over-regularization, in the Ridge plot it can be observed easly.

In order to differ the models and see their trade-off we made a simulation.

Sample 100 random and consecutive subsamples from our data, 1000 times (iterations).

In each iteration we separate the subsamples into X and Y, train the three models on each subsample and compute the bias & MSE for each subsample.

All the results will be stored in a 1000 X 12 DataFrame that we use later to plot the distribution of each subsample.

```
In [ ]: bias_mse = []
        ols_full = model

        n_simulations = 1000
        n_subsample = 100

        for i in range(n_simulations):

            rand_sample = df.sample(n_subsample)
            rand_ind = random.randint(0,399)
            con_sample = df.iloc[rand_ind:rand_ind + n_subsample]

            # Separate the subsamples into X and Y
            rand_X = sm.add_constant(rand_sample.drop('y', axis=1))
            rand_Y = rand_sample['y']
            con_X = sm.add_constant(con_sample.drop('y', axis=1))
            con_Y = con_sample['y']

            # Train the models on each subsample
            rand_ols = sm.OLS(rand_Y, rand_X).fit()
            con_ols = sm.OLS(con_Y, con_X).fit()
            rand_ridge = Ridge(alpha=ridge_optimal_lambda).fit(rand_X, rand_Y)
            con_ridge = Ridge(alpha=ridge_optimal_lambda).fit(con_X, con_Y)
            rand_lasso = Lasso(alpha=lasso_optimal_lambda).fit(rand_X, rand_Y)
            con_lasso = Lasso(alpha=lasso_optimal_lambda).fit(con_X, con_Y)

            # MSE & bias calculations for random, the consecutive samples
            rand_bias_mse = [mean_squared_error(ols_full.predict(rand_X), rand_ridge.predict(rand_X)), # MSE
                             mean_squared_error(ols_full.predict(rand_X), rand_lasso.predict(rand_X)),
                             mean_squared_error(ols_full.predict(rand_X), rand_ols.predict(rand_X)),
                             np.mean(rand_ridge.coef_ - ols_full.params), # bias
                             np.mean(rand_lasso.coef_ - ols_full.params),
                             np.mean(rand_ols.params - ols_full.params)]

            con_bias_mse = [mean_squared_error(ols_full.predict(con_X), con_ridge.predict(con_X)),
                            mean_squared_error(ols_full.predict(con_X), con_lasso.predict(con_X)),
                            mean_squared_error(ols_full.predict(con_X), con_ols.predict(con_X)),
                            np.mean(con_ridge.coef_ - ols_full.params),
                            np.mean(con_lasso.coef_ - ols_full.params),
                            np.mean(con_ols.params - ols_full.params)]


            bias_mse.append([rand_bias_mse[0], rand_bias_mse[1], rand_bias_mse[2],
                             rand_bias_mse[3], rand_bias_mse[4],
                             rand_bias_mse[5], con_bias_mse[0], con_bias_mse[1],
                             con_bias_mse[2], con_bias_mse[3],
                             con_bias_mse[4], con_bias_mse[5]])


            bias_mse_df = pd.DataFrame(bias_mse, columns=['MSE_ridge_rand', 'MSE_lasso_Rand', 'MSE_ols_rand',
                                                          'Bias_ridge_rand', 'Bias_lasso_rand', 'Bias_ols_rand',
                                                          'MSE_Ridge_Con', 'MSE_Lasso_con', 'MSE_OLS_Con',
                                                          'Bias_Ridge_Con', 'Bias_Lasso_con', 'Bias_OLS_Con'])
        bias_mse_df
```

Out[ ]:

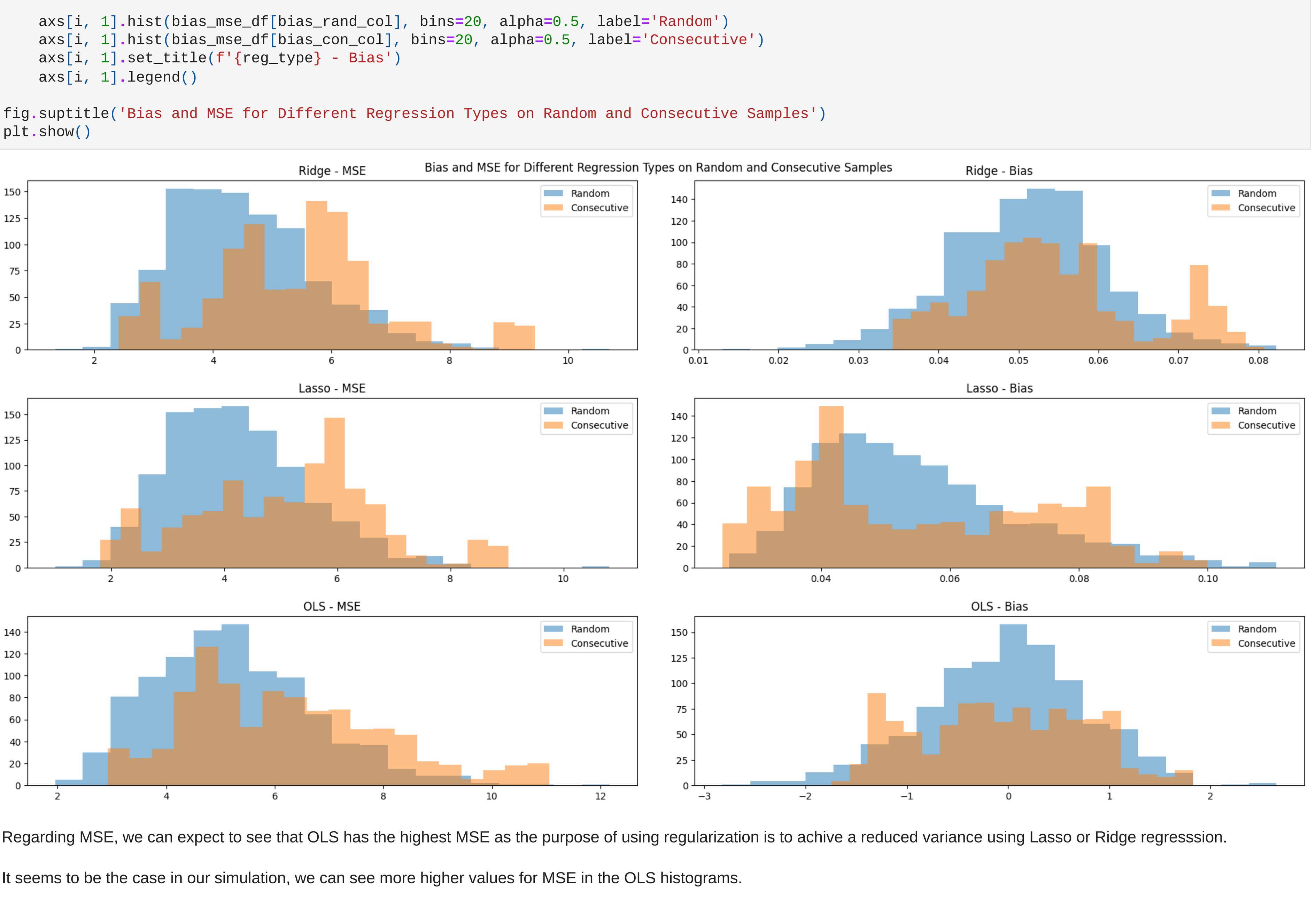
	MSE_ridge_rand	MSE_lasso_Rand	MSE_ols_rand	Bias_ridge_rand	Bias_lasso_rand	Bias_ols_rand	MSE_Ridge_Con	MSE_Lasso_con	MSE_OLS_Con	Bias_Ridge_Con	Bias_Lasso_c
0	3.985252	3.539703	5.045808	0.062259	0.067673	0.198339	5.242751	5.345382	5.020978	0.046332	0.040
1	5.383217	4.781699	5.748906	0.051424	0.074736	-0.514102	5.431101	5.489781	5.951513	0.049914	0.042
2	4.324898	3.944958	6.239255	0.056467	0.063106	0.504143	5.600884	5.622080	5.489501	0.048605	0.037
3	3.754149	3.689119	3.745799	0.043091	0.038202	0.583240	2.448783	1.837438	3.034274	0.070786	0.084
4	4.162005	4.086547	5.711110	0.040270	0.041447	-0.134995	6.156911	6.226758	5.986308	0.039026	0.031
...	...	...	...	...	...	...	...	...	...	...	
995	3.635693	2.987259	4.375921	0.062045	0.088091	-0.473497	2.822758	2.276566	3.582281	0.070246	0.085
996	6.014214	5.859016	6.463025	0.049935	0.045028	-0.939010	6.905436	6.781515	9.318721	0.054196	0.043
997	4.863982	4.187113	6.523842	0.054571	0.096855	-0.228244	6.470065	6.731027	6.543352	0.036148	0.027
998	6.227900	5.632052	6.433742	0.052510	0.049180	0.568773	2.864859	2.244460	3.736647	0.073037	0.084
999	4.629748	4.377833	6.312515	0.064179	0.057489	-1.537400	6.514896	6.821562	6.705072	0.036813	0.027

1000 rows × 12 columns

Now, after we have all the relevent data, we create a 3x2 grid of subplots for the distribution of the subsamples (each row in the grid is a regression type and each column is MSE/bias distributions).

We kept both random and consecutive samples in the same plots in different colors to better see differences and make an overall cleaner plot.

```
In [ ]: fig, axs = plt.subplots(3, 2, figsize=(20, 10))
        fig.tight_layout(pad=3.0)

        regressions = [('Ridge', 'MSE_ridge_rand', 'MSE_Ridge_Con', 'Bias_ridge_rand', 'Bias_Ridge_Con'),
                       ('Lasso', 'MSE_lasso_Rand', 'MSE_Lasso_con', 'Bias_lasso_rand', 'Bias_Lasso_con'),
                       ('OLS', 'MSE_ols_rand', 'MSE_OLS_Con', 'Bias_ols_rand', 'Bias_OLS_Con')]


        for i, (reg_type, mse_rand_col, mse_con_col, bias_rand_col, bias_con_col) in enumerate(regressions):

            axs[i, 0].hist(bias_mse_df[mse_rand_col], bins=20, alpha=0.5, label='Random')
            axs[i, 0].hist(bias_mse_df[mse_con_col], bins=20, alpha=0.5, label='Consecutive')
            axs[i, 0].set_title(f'{reg_type} - MSE')
            axs[i, 0].legend()

            axs[i, 1].hist(bias_mse_df[bias_rand_col], bins=20, alpha=0.5, label='Random')
            axs[i, 1].hist(bias_mse_df[bias_con_col], bins=20, alpha=0.5, label='Consecutive')
            axs[i, 1].set_title(f'{reg_type} - Bias')
            axs[i, 1].legend()

        fig.suptitle('Bias and MSE for Different Regression Types on Random and Consecutive Samples')
        plt.show()
```



Regarding MSE, we can expect to see that OLS has the highest MSE as the purpose of using regularization is to achive a reduced variance using Lasso or Ridge regresssion.

It seems to be the case in our simulation, we can see more higher values for MSE in the OLS histograms.

In terms of bias, we expect to see that Lasso and Ridge have lower biases compared to OLS due to the regularization they apply.

Indeed, in our plot it can be observed easly just from looking at the range of values in the X_axis for bias in the different regression types.

As we can see, in the first two rows of the bias in the plot (Ridge & Lasso models), we have a small range of bias values that are near zero for both models.

In the third row (OLS), we have a range from -3 to 2 which have a lot of values above 1 so the differences in bias are more dramatic.

Overall, it seems that the regressions with the regularization succeed in their missions.

Consecutive samples may have an higher correlation between the observations, if there is some dependence or order, which can lead to high variance.

Random samples, On the other side, seems to have more diverse observations that can increase bias.

The consecutive samples can potentially lead to higher variance (if we know our data it can be avoided) and the Random samples can potentially lead to higher bias becaues they are more diversed.

So there is a tradeoff between bias and variance when conducting each sample strategy.

We would choose the Lasso model in a random sample since its bias is already low in our experiment, so we would like to keep its variance as low as we can and in random samples we expect less variance.

Finally, we fit a Lasso model on all the data and extract its beta coefficients, then use the P-values of the ols regression on all data we kept at the begining and compare.

In order to compare them we kept both as rows in data frame to get a good look at the values.

```
In [ ]: lasso = Lasso(alpha=lasso_optimal_lambda)
        lasso.fit(x, y)

        lasso_ols_df = pd.DataFrame(index = ['x' + str(i) for i in range(1,25)])
        lasso_ols_df["OLS Pvalue"] = ols_pvals
        lasso_ols_df["Lasso Beta"] = list(lasso.coef_.round(4))
        lasso_ols_df.T
```

Out[ ]:

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
OLS Pvalue	0.0898	0.1626	0.0000	0.0854	0.4336	0.0000	0.0031	0.0169	0.9306	0.2889	...	0.4229	0.9124	0.6794	0.3148	0.0000	0.0000	0.6348	0.8145	0.1403	0.000
Lasso Beta	-0.0239	0.0303	0.1531	0.0473	-0.0000	0.1503	0.0884	0.0720	0.0000	-0.0072	...	-0.0083	-0.0000	0.0000	-0.0000	0.0456	0.0493	0.0004	-0.0000	0.0000	0.368

2 rows × 24 columns

We expect a negative correlation between the two values, as Lasso tends to shrink coefficients towards zero (using the regulizer lambda), while OLS may produce non-zero coefficients even for insignificant features, all features are included in the OLS model.

Just from looking at the data frame, its seems to be the case, We'll check the correlation formally now to test our hypothesis.

```
In [ ]: correlation = lasso_ols_df["OLS Pvalue"].corr(lasso_ols_df["Lasso Beta"])
        print("Correlation between OLS P-value and Lasso Beta:", correlation)
```

```
Correlation between OLS P-value and Lasso Beta: -0.4995573537527258
```

As we expected, the correlation between OLS P-values and Lasso Beta coefficients is negative.