Optimization Project 3:
Feature Selection using MIQP vs Lasso

Group Members:

Krish Engineer (ke7466)
Rudraksh Garg (rg44778)
Ryan Lee (rl32227)
Samarth Mishra (sm79247)

Purpose

One of the most common problems in predictive analytics is variable selection for regression. Direct variable selection using optimization has long been dismissed by the statistics/analytics community because of computational difficulties. This computational issue was part of the motivation for the development of LASSO and ridge regression. However, in the recent past there have been tremendous advancements in optimization software, specifically the ability to solve mixed integer quadratic programs (MIQP) in order to find the best set of coefficients for regression.

Objective

In this study, we tested the feasibility of direct variable selection using two approaches:

1. Mixed Integer Quadratic Programming (MIQP)
2. LASSO Regression

The MIQP variable selection problem for regression will be solved using Gurobi and its results will be compared to the results given by LASSO regression. The comparison of these results will then be used to advise management on which variable selection method should be preferred.

Approach 1 : Direct Variable Selection - MIQP

Objective Function

This problem can be structured as an optimization problem where our objective is to minimize the squared loss incurred during regression. So our objective becomes minimizing the sum of squared errors as specified below:

$$\min_{\beta,z} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_m x_{im} - y_i)^2$$

Objective Function Manipulation

β can be constructed as a vector of size m+1. This is m+1 because we have m variables and 1 intercept (β_0). X can be constructed as an n by (m+1) matrix where the first column is all ones and the second column onwards has the data from each of the n records. The target vector would be the output from each of the n records. We can then construct a vector of size n using the following:

$$(X\beta - y)$$

This looks similar to our objective function. Since we are using the squared errors, we would have to take the following instead:

$$(X\beta - y)^T * (X\beta - y)$$

We can simplify this for Gurobi and pose our objective function as a MIQP problem as given below:

$$\min_{\beta,z} \beta^T (X^T X)\, \beta + (-2\, y^T X)\, \beta$$

With this setup, while β may look as being of size m+1, it actually needs to be of dimension 2m+1 as there are m+1 additional binary variables z_j's, that represent if those β variables should be included in the regression or not.

Given below are the Quadratic (Q) and Linear Term matrices (c) for this MIQP problem:

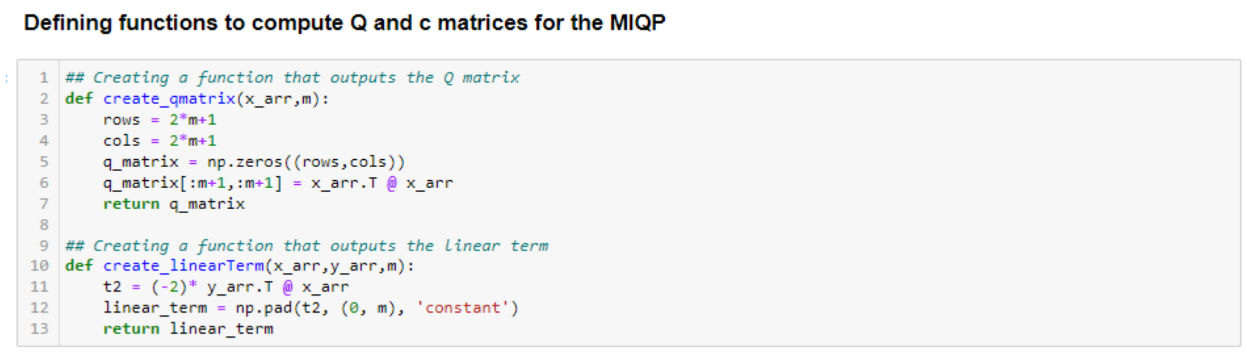
Quadratic Matrix (Q)

Q= $X^T X$ (Dimension: 2m+1 * 2m+1)
Here, only the upper left corner has non-zero values equal to $X^T X$ while all other values are zero (matrix manipulated to keep consistent in the Gurobi format)

Linear Term Matrix (c)

c= $-\ 2y^T X$ (Dimension: 2m+1 * 1)
Here, only the first m components have non-zero values equal to $-\ 2y^T X$ while while all other values are zero (matrix manipulated to keep consistent in the Gurobi format)

Code Snippet for computing the Q and c matrices



Defining functions to compute Q and c matrices for the MIQP

```python
## Creating a function that outputs the Q matrix
def create_qmatrix(x_arr,m):
    rows = 2*m+1
    cols = 2*m+1
    q_matrix = np.zeros((rows,cols))
    q_matrix[:m+1,:m+1] = x_arr.T @ x_arr
    return q_matrix

## Creating a function that outputs the linear term
def create_linearTerm(x_arr,y_arr,m):
    t2 = (-2)* y_arr.T @ x_arr
    linear_term = np.pad(t2, (0, m), 'constant')
    return linear_term
```

Decision Variables:

β_i: Beta coefficients for each of the predictor variables in linear regression

Z_j: The z variables tell us whether a beta gets picked or not during variable selection

<u>Constraints</u>

1. <u>Big-M Constraints:</u>

   The z variables tell us whether a beta gets picked or not. Thus, a beta can exist only if z is non-zero.

   Using "big-M" constraints, we can set a max that all coefficients could possibly be:

$$s.t. -Mz_j \le \beta_j \le Mz_j \quad for\ j = 1,2,3,\dots,m$$

   Big M value taken in our MIQP model=100 (sufficiently large)

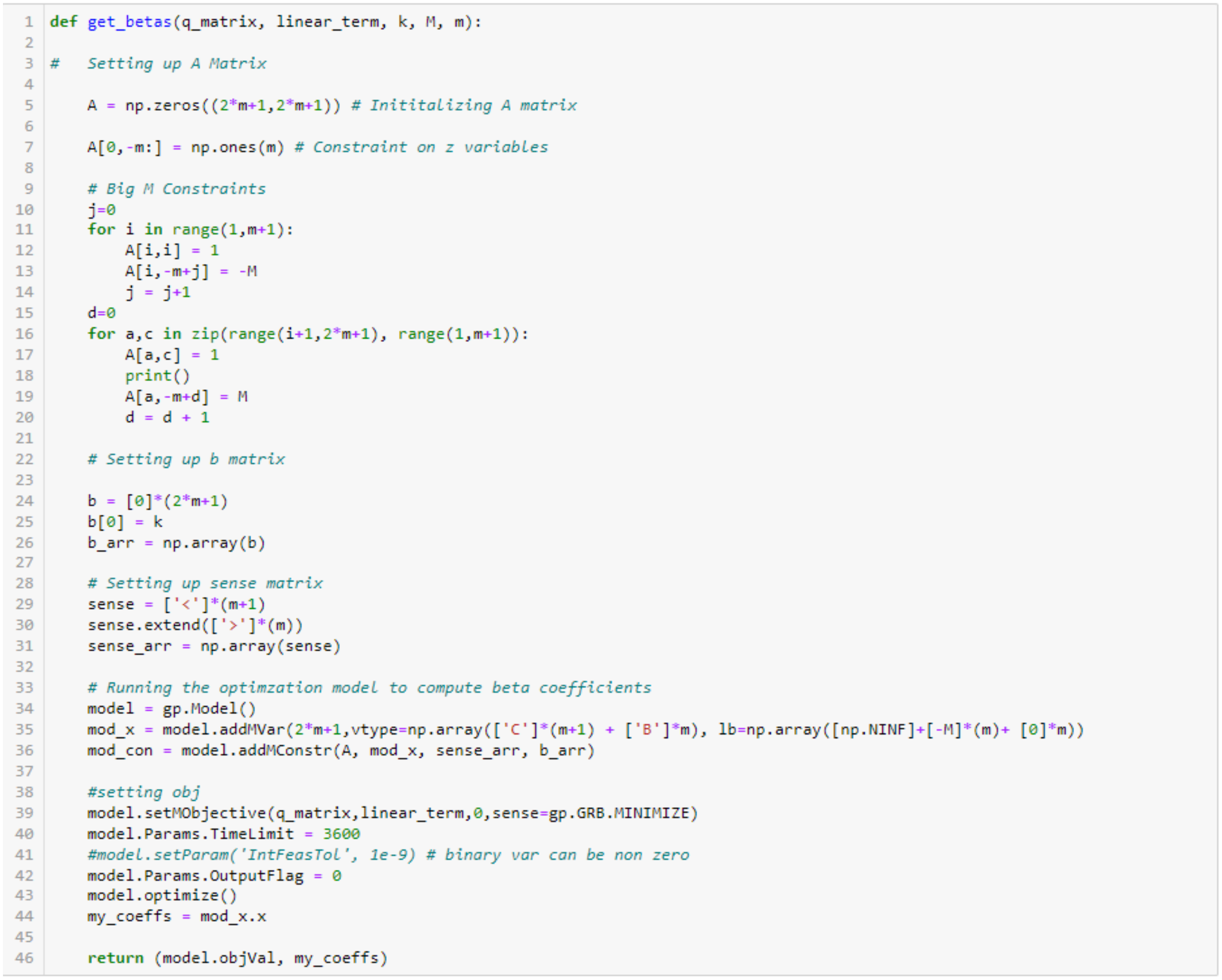
2. <u>Number of variables to be chosen (k)</u>

   We also need to specify the max number of variables that need to be selected. We would need binary variables that would act as switches that would state if a variable would be added or not. This can be framed using the following constraint where k is the maximum number of variables for our regression:

$$\sum_{j=1}^{m} z_j \le k$$

$$z_j \ \ are\ binary$$

Code Snippet for Computing the Beta Coefficients

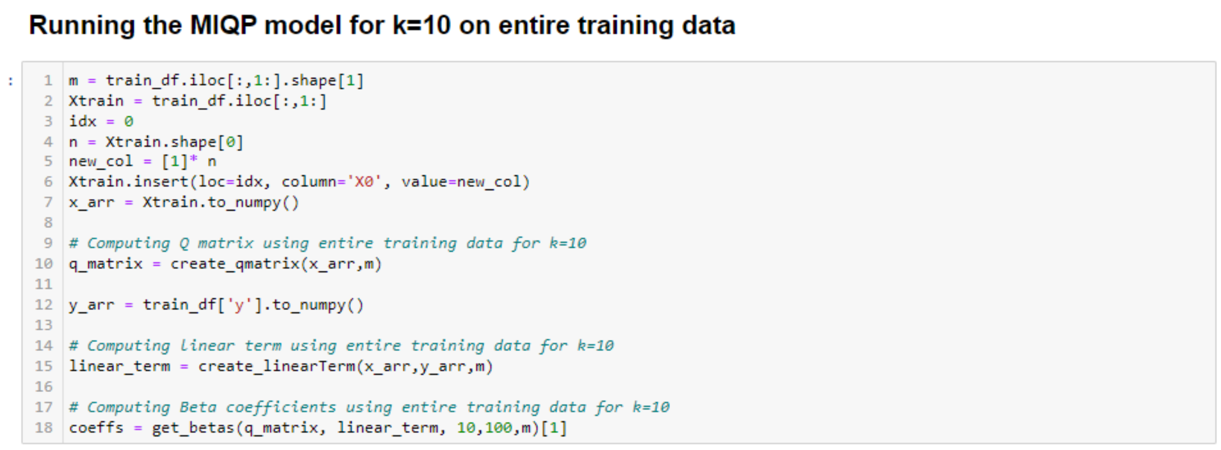


```python
def get_betas(q_matrix, linear_term, k, M, m):

#   Setting up A Matrix

    A = np.zeros((2*m+1,2*m+1)) # Inititalizing A matrix

    A[0,-m:] = np.ones(m) # Constraint on z variables

    # Big M Constraints
    j=0
    for i in range(1,m+1):
        A[i,i] = 1
        A[i,-m+j] = -M
        j = j+1
    d=0
    for a,c in zip(range(i+1,2*m+1), range(1,m+1)):
        A[a,c] = 1
        print()
        A[a,-m+d] = M
        d = d + 1

    # Setting up b matrix

    b = [0]*(2*m+1)
    b[0] = k
    b_arr = np.array(b)

    # Setting up sense matrix
    sense = ['<']*(m+1)
    sense.extend(['>']*(m))
    sense_arr = np.array(sense)

    # Running the optimzation model to compute beta coefficients
    model = gp.Model()
    mod_x = model.addMVar(2*m+1,vtype=np.array(['C']*(m+1) + ['B']*m), lb=np.array([np.NINF]+[-M]*(m)+ [0]*m))
    mod_con = model.addMConstr(A, mod_x, sense_arr, b_arr)

    #setting obj
    model.setMObjective(q_matrix,linear_term,0,sense=gp.GRB.MINIMIZE)
    model.Params.TimeLimit = 3600
    #model.setParam('IntFeasTol', 1e-9) # binary var can be non zero
    model.Params.OutputFlag = 0
    model.optimize()
    my_coeffs = mod_x.x

    return (model.objVal, my_coeffs)
```

Results

Running the MIQP Model for different values of k with 10-fold CV

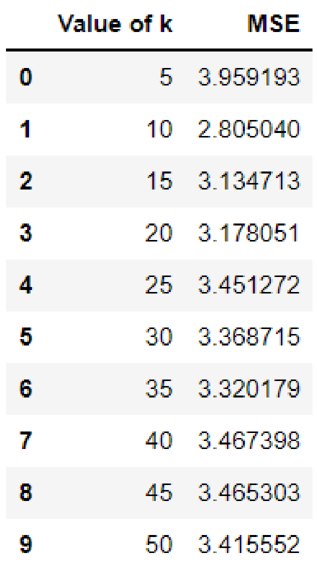
To solve the MIQP problem, we first started off by defining the cross-validation function to split the training data into 10 folds. The function was set to try values of k ranging from 5 to 50 since there were 50 variables to select from. We then set up the optimization problem through Gurobi in the format described in the Direct Variable Selection section of this report. Our objective was to find out the value of k that gives out the least Cross Validation error.



Running the MIQP model for k=10 on entire training data

```
m = train_df.iloc[:,1:].shape[1]
Xtrain = train_df.iloc[:,1:]
idx = 0
n = Xtrain.shape[0]
new_col = [1]* n
Xtrain.insert(loc=idx, column='X0', value=new_col)
x_arr = Xtrain.to_numpy()

# Computing Q matrix using entire training data for k=10
q_matrix = create_qmatrix(x_arr,m)

y_arr = train_df['y'].to_numpy()

# Computing linear term using entire training data for k=10
linear_term = create_linearTerm(x_arr,y_arr,m)

# Computing Beta coefficients using entire training data for k=10
coeffs = get_betas(q_matrix, linear_term, 10,100,m)[1]
```

Given below are the Cross Validation error that we obtained for different values of k:



	Value of k	MSE
0	5	3.959193
1	10	2.805040
2	15	3.134713
3	20	3.178051
4	25	3.451272
5	30	3.368715
6	35	3.320179
7	40	3.467398
8	45	3.465303
9	50	3.415552

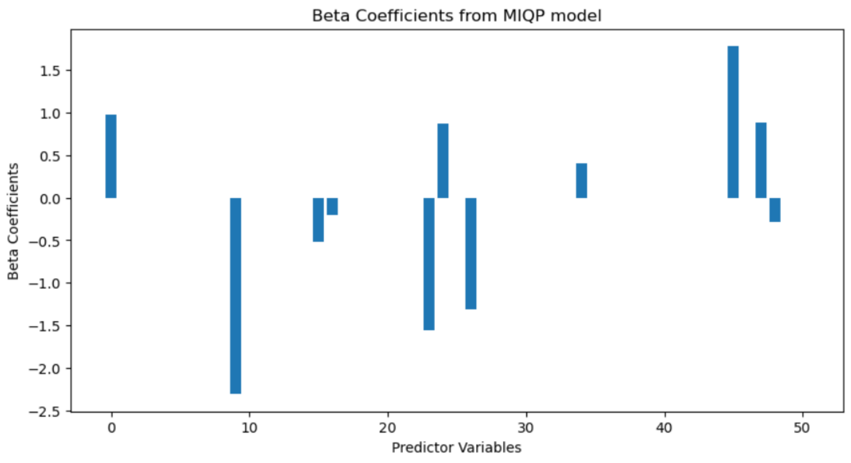
We see that the least Cross Validation error was obtained for k=10.

Hence we chose k=10 for our final MIQP model and ran the MIQP model on the entire training dataset. We determined the β's for each feature and then used it to predict the y values in the test set.

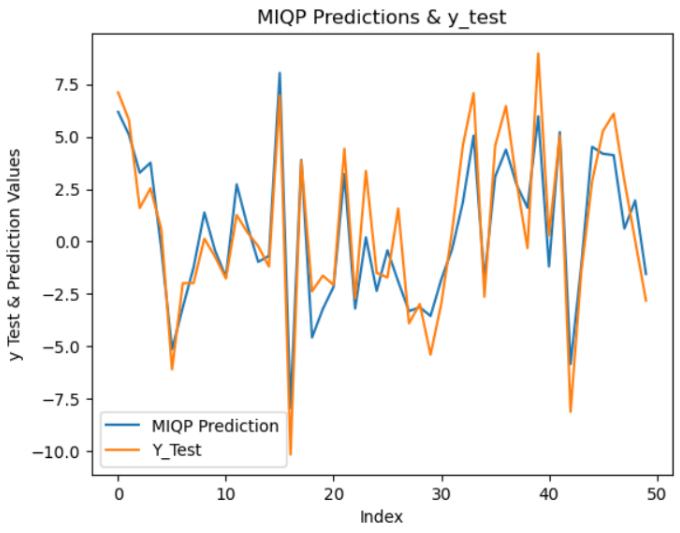
Given below are the Training and Test MSE's obtained:

Training MSE obtained: 2.392
Test MSE obtained: 2.337

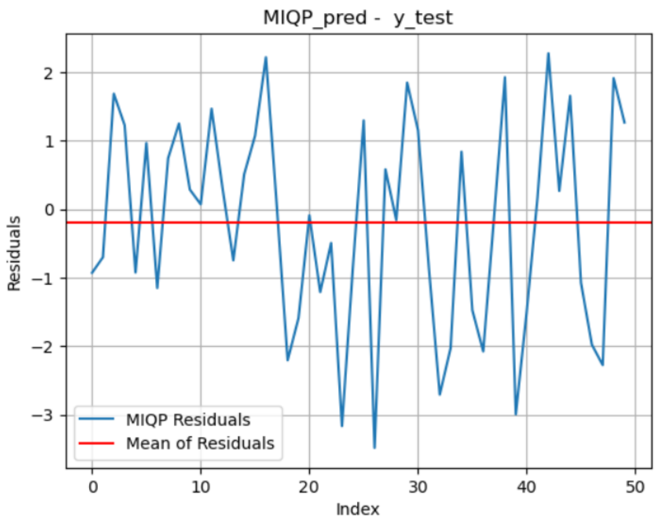
Beta Coefficients Obtained for MIQP



MIQP Predictions vs Actual y values (Test Set)



MIQP Prediction Residuals (Test Set)



Approach 2 : Indirect Variable Selection using LASSO

We applied the lasso regularization to a linear regression model trained on our training set to compare the outcomes of our MIQP with those of other indirect feature selection techniques. By including the absolute sum of coefficients penalty in the regression loss function, Lasso aids in the reduction of the regression coefficients. The indirect variable selection or LASSO Regularization method is posed as

$$\min_{\beta} \sum_{i=1}^{n} (\beta_o + \beta_1 x_{i1} + \ldots\ldots\ldots\ldots + \beta \quad x_i \quad - y_i)^2 + \lambda \sum_{j=1}^{m} |\beta \quad |$$

Where λ is a hyperparameter that can be chosen during cross validation. The LASSO regularization model has the advantage of 'shrinking' the β*s* closer to zero, leading to variance reduction i.e. prevents overfitting. Furthermore, if λ is large enough, several values of β will be forced to be equal to 0.

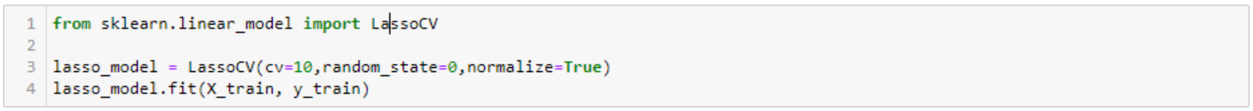
There are more and more variables with zero coefficients as the penalty term grows. The penalty term is taken into consideration using the regularization parameter. This is how we can employ the Lasso to choose our features in an indirect manner.

Results

Running Lasso Regression doing 10-Fold CV to determine optimal *λ*

We used Lasso CV to do a 10-fold cross validation on the training set and determined the optimal value of *λ*.

Code Snippet for Lasso regression using 10-fold CV



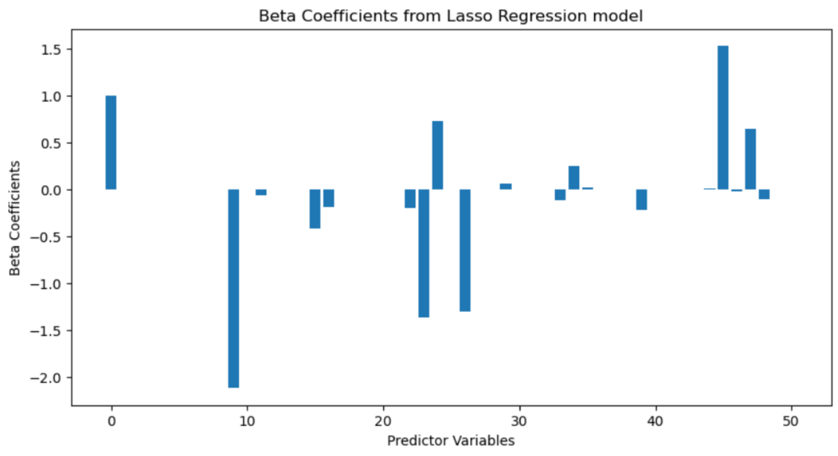
```python
from sklearn.linear_model import LassoCV

lasso_model = LassoCV(cv=10,random_state=0,normalize=True)
lasso_model.fit(X_train, y_train)
```

Optimal lambda obtained: 0.0057

We used this value of *λ* to fit the LASSO model on the entire training set and determined Beta coefficients. Using the Beta coefficients, we determined the Lasso prediction on the test set.
Given below are the Training and Test MSE's obtained:

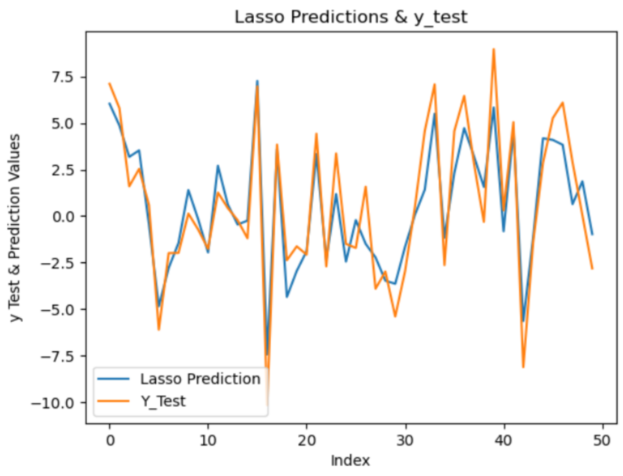
Training MSE obtained: 2.401
Test MSE obtained: 2.360

Beta Coefficients Obtained for Lasso Model

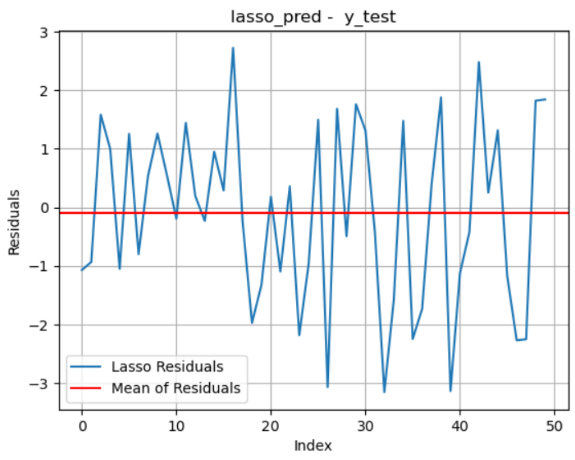


We see that Lasso model selects 18 out of the 50 predictor variables, more than what we chose through our most optimal MIQP model

Lasso Predictions vs Actual y values (Test Set)



Lasso Prediction Residuals (Test Set)



Recommendation

We are in a better position to use specialized optimization techniques as opposed to only depending on easily accessible ML libraries as computation resources become more freely available. Due to the assumptions they make for faster convergence in their internal optimizations, the existing modules and packages may have some limitations. In our use-case, our unique optimization can produce a model that is less complex and has a higher holdout sample MSE. A slightly more reliable model is created by our MIQP approach not taking into account LASSO features that only had coefficient values close to zero in the lasso regression.

However, there are advantages and disadvantages to each approach depending on the use case. This must be taken into consideration when deciding which one to pick. With this in mind, the company may choose any approach based on the needs of the business case.

Finding the best value of k computationally takes much longer for the direct variable selection technique (MIQP model) than it does for the LASSO model. The direct variable selection model might be challenging in instances where there are many features to choose from. However, this strategy performs only slightly better. In the workplace, this model can be run overnight or on a cloud server and provide results without affecting productive hours, however that still may be costly in terms of time and money. It is also worth noting that MIQP chose less variables than LASSO which might be considered an advantage as there is less data dependency and lower complexity in the model.

Because there is only a very small difference in MSE and mean residuals between our MIQP and LASSO, in this situation, with the given data, it might not be worth the extra training time for the potentially insignificant improvement in accuracy. However, as there is still a tradeoff between accuracy and computation cost, the decision between the two should be determined by the application and use cases of these variable selection methods.

In contrast to the MIQP model, LASSO regression is a built-in Python feature that can be quickly accessible and utilized with few lines of code. For clients and stakeholders who might not be familiar with conventional optimization techniques, the LASSO model's outcomes are also simpler to explain and performs almost as well as MIQP in terms of MSE. Out of regularization and feature selection models, LASSO is more popular as it balances accuracy, time, and cost much better than direct variable selection through MIQP.