University of Manchester

Programming in Python for Business Analytics: Supplier Recommendation to Acme Corporation

Prepared By: BeePy

Sergio Chavez Lazo
Dahye Jeong
Mohamed Hadhry Bin Haslimi
Devesh Prasad
Suparna Roy
Samit Shrinivas Uttarkar
Afifa Pirzada

Table of Contents

1. Abstract

This report discusses the use of machine learning (ML) models in recommending the best suppliers to Acme Corporation, given a set of task features. This process begins with data cleaning and preparation, exploratory data analysis (EDA) and machine learning, where results from predicted data can be obtained. Cross-validation (Leave-One-Group-Out) is used to validate the ML model score, and hyper-parameter optimization is further used to find the optimal set of hyper-parameters for improved predictions.

2. Introduction

The Acme Corporation is considering ways to improve its day-to-day task. Out of 64 vendors who offer resources to perform tasks, they want to choose one. In this report, machine learning models will be used to determine which supplier is the best fit for each task by training the models with information about the task and the supplier features.

3. Data Coding & Analysis

Three datasets, 'tasks.xlsx', 'costs.csv' and 'suppliers.csv' have been provided. The attributes for all three datasets were then obtained. *Table 1 (before)* describes the obtained attributes.

3.1 Data Preparation

To obtain suitable datasets for exploratory data analysis and machine learning algorithms, the data preparation phase was carried out prior to the main data analysis process.

The data cleaning consisted of 5 tasks with different objectives.

1. Checking for missing values and the proper format of variables according to their nature (e.g., Cost as floats).
2. Aligning the format of the key variables in all databases (Task ID/ Supplier ID) to identify any task that does not have a related cost (dependent variable in future models) and to merge the datasets without conflict.
3. Variables (task features) with low variability, indicated by a variance of less than 0.01, were eliminated to keep only attributes heterogeneous enough to provide relevant insights for exploration and modelling. 35 Task Features were dropped.
4. Variables (all features) were standardized on a scale of -1 to 1 to facilitate comparisons between features in EDA and model training.
5. Highly correlated variables were filtered out to avoid multicollinearity and redundancy problems in subsequent models. 54 task features that exhibited 0.8 or more correlation with other task features were dropped.
6. Filtering out low-performance suppliers indicated by the inability to appear in the top 20 costs for any task. Only one supplier did not meet this requirement, hence, was dropped.

'tasks'		'suppliers'		'costs'	
Before	**After**	**Before**	**After**	**Before**	**After**
130 tasks	120 tasks	64 suppliers	63 suppliers	120 tasks	120 tasks
116 task features	27 task features	18 supplier features	18 supplier features	64 suppliers	63 suppliers
				7680 cost values	7560 cost values

Table 1. Dataset Attributes before and after data preparation

3.2 Exploratory Data Analysis

Exploring and visualizing datasets help discover trends and patterns and understand statistical summaries utilising graphical results.

3.2.1 Distribution of Feature values

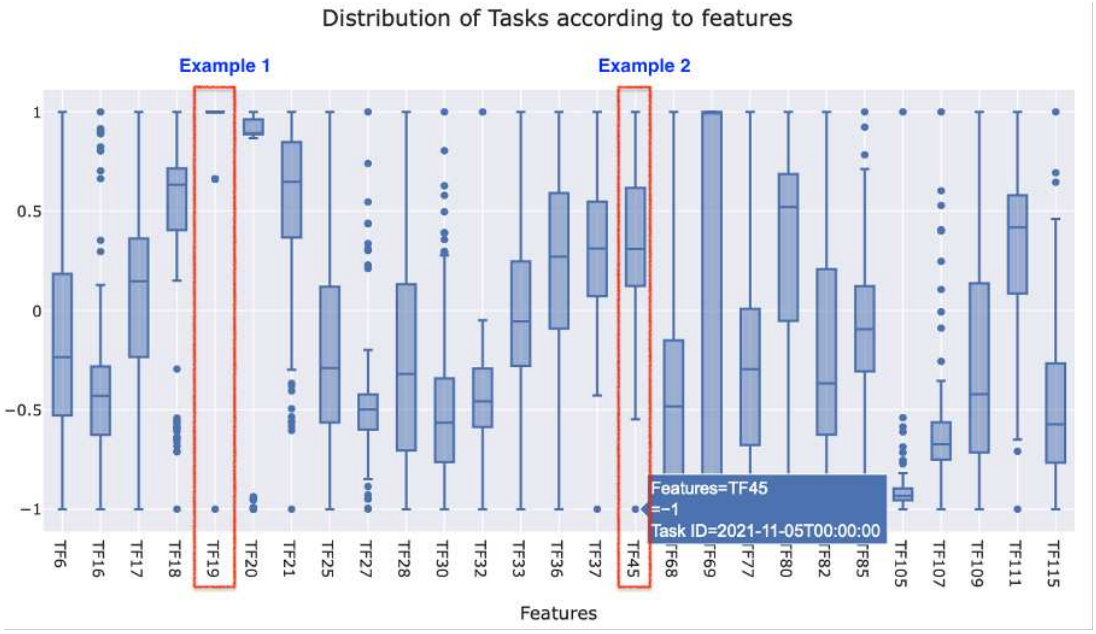


Figure 1. Distribution of tasks for each task feature

An interactive graph (*Figure 1*) was created to observe the distribution of tasks based on each task feature. Through this, some apparent exceptions were identifiable due to the outliers. For example, tasks 20/03/2021 and 05/11/2021 are the only exceptions found in TF19, whereas all other tasks display the same single value. Similarly, for TF37, TF45, TF85, and TF111, task ID 05/11/2021 is clearly distinguishable. As a result, Task ID 05/11/2021 stands out from the others in several ways.

An additional boxplot (*Figure 2*) has been created with inverted axes showcasing each task's distribution of feature values. It was found that most of the functions featured diverse values. However, specific tasks, such as 05/11/2021, had a much smaller variance.

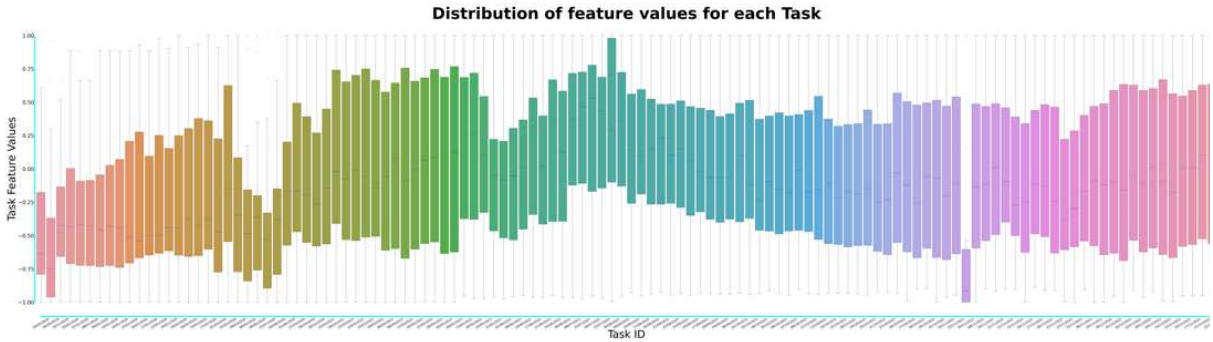


Figure 2. Distribution of feature values for each Task

3.2.2 Distribution of error based on supplier chosen

Figure 3 was used to visualize the variance between the competitiveness of supplier prices using error values, and they were calculated assuming a particular supplier was selected for

all tasks. RMSE scores were calculated with the obtained error values and annotated onto each boxplot.

An error was calculated by deducting the supplier's cost for a particular task with the minimum cost for the same task. Thus, a high error value indicates the supplier offered a high price for the particular task. In contrast, the lower and concentrated distribution indicates that the supplier provided a generally lower price for all the tasks.

The graph demonstrates that on average, S03 has the highest cost, whereas S56 is the cheapest. In conclusion, if only one supplier would be chosen for all tasks, S56 would be the ideal choice.

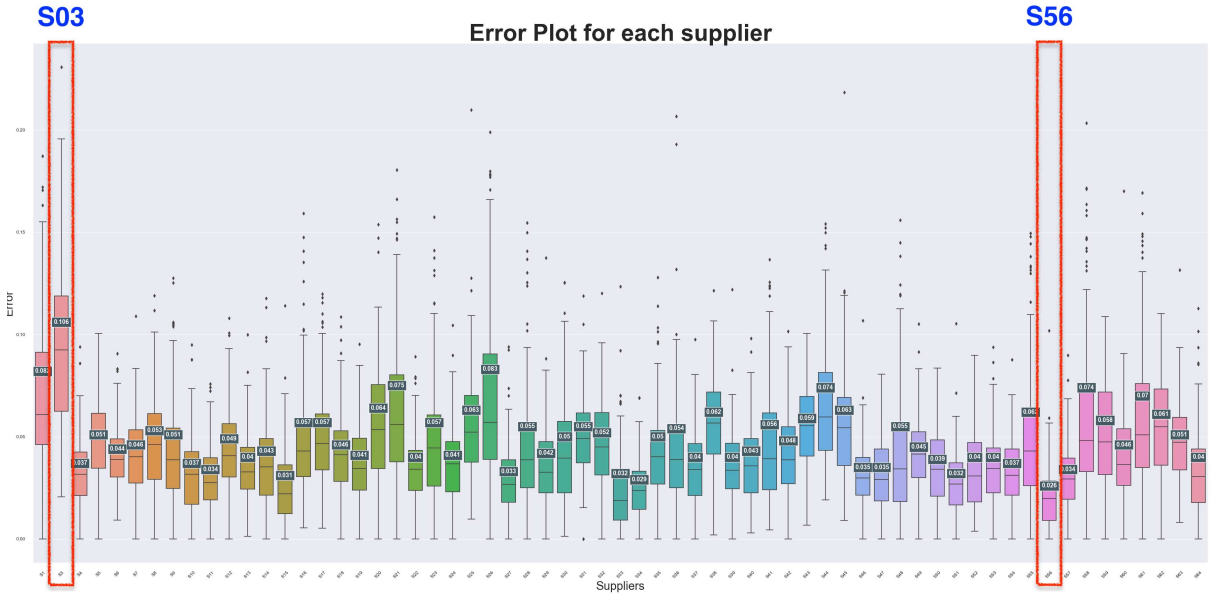


Figure 3. Error distribution for each supplier

3.2.3 Cost comparison between suppliers

The cost dataset's format was converted to a wide format to show the correlation between Task ID and Supplier ID. The 'natsort' module was used for sorting the alphanumeric column names.

Heatmaps were generated to visualize the gradient between costs, where red indicates the most expensive and blue indicates the cheapest. *Figure 4* was created using seaborn to visualise the gradient between costs throughout all the tasks. This figure provides an understanding of the general price differences among tasks.
It can be observed from figure 4 that task '05/11/2021' is the most expensive one as the entire row is red describing that it behaves differently from others. This is also observed in *Figure 1*.

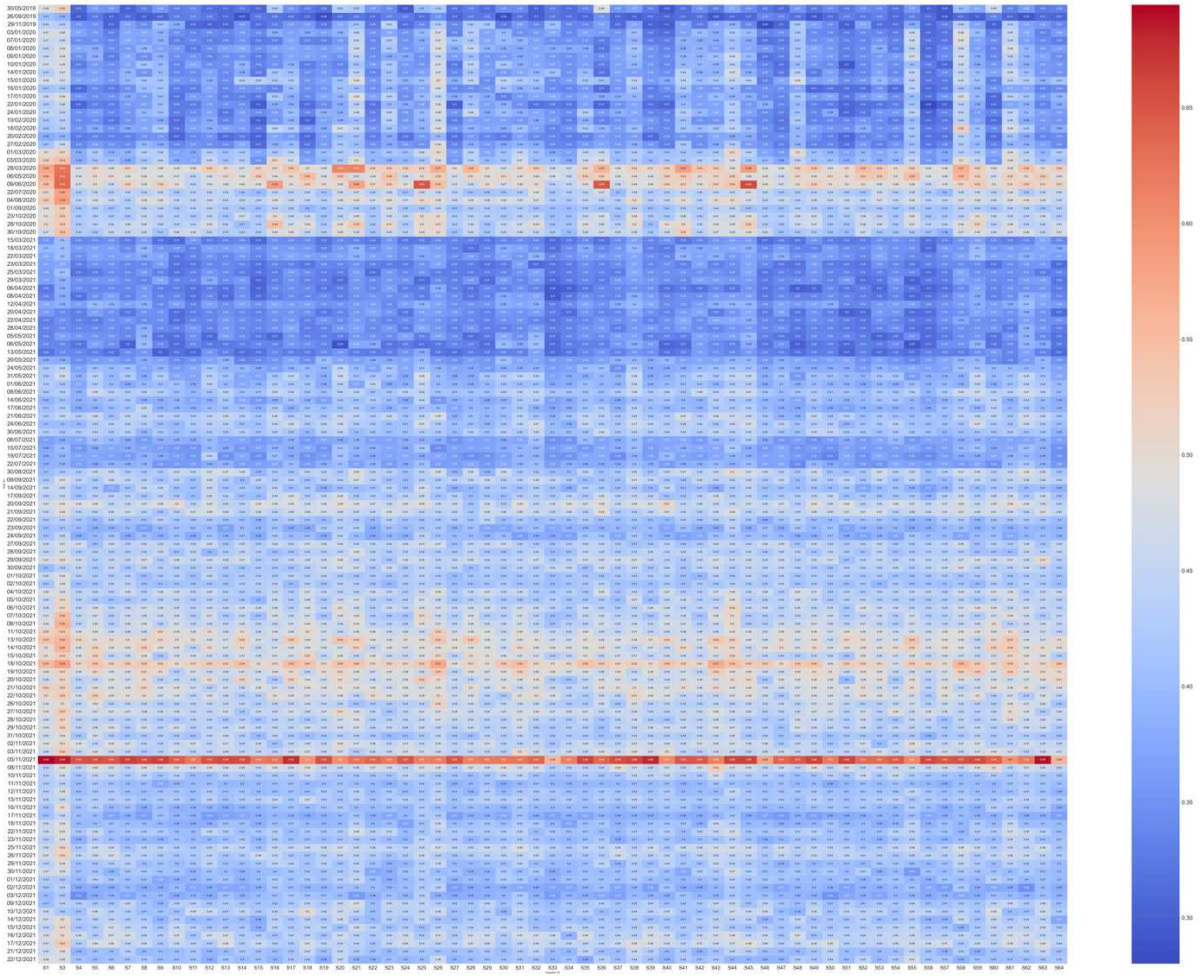


Figure 4. Heatmap showing gradient between costs among all tasks

Figure 5 was generated as a styler dataframe. This heatmap highlights the supplier cost gradient for each task. This figure indicates the relative difference in price that each supplier offers. The findings are similar to *Figure 3*; S3 is generally the most expensive while S56 is the cheapest.

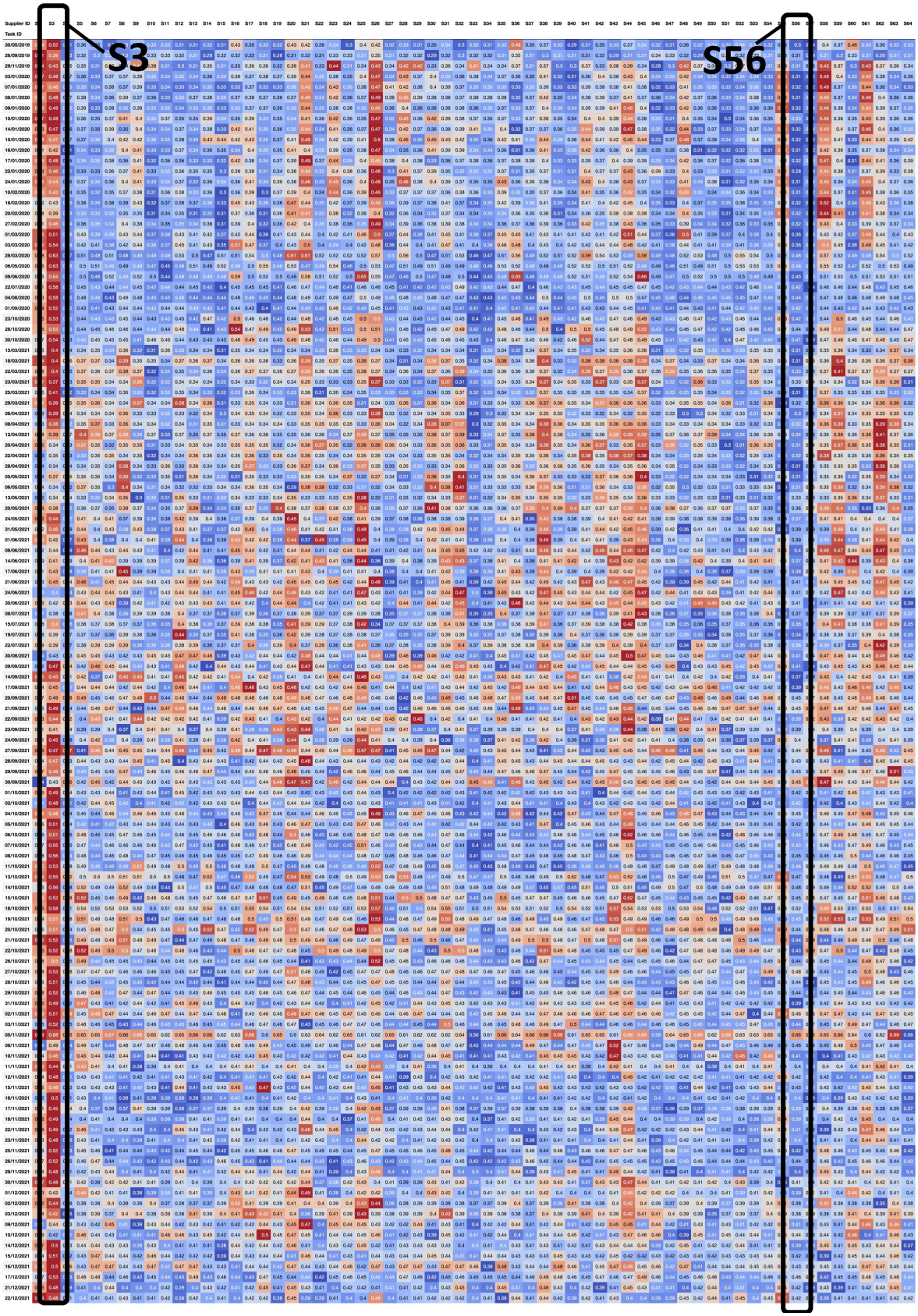


Figure 5. Heatmap indicating supplier cost gradient for each task

Figure 4 illustrates the entire dataset, i.e., how expensive a task could be for all the suppliers, whereas *Figure 5* shows the minimum or highest cost for each row of the dataset.

3.2.4 Summary of EDA Results

Task 05/11/2021 is different from the rest in terms of features and, consequently, in terms of cost. Regardless of the supplier chosen, this task remains the most expensive. Furthermore, S56 has the best performance on average, and S3 is the worst, according to *Equation 1*.

3.3 Machine Learning

The three datasets were merged, resulting in a dataset consisting of 7560 rows of Task–Supplier combinations, each with 45 features and the cost. It was then split into datasets 'x' and 'y', containing the features and costs, respectively. 20 Task IDs were selected randomly as the test dataset, and the remaining were assigned as the training data.

3.3.1 Selection and training models – Model Fitting

The Lasso regression model applies a strict penalty that eliminates (coefficients shrink to zero) variables irrelevant to the prediction. It is especially useful in a testing context where all possible variables are included, regardless of whether they are theoretically/logically useful for predicting the dependent variable.

Given the nature of variables and the assumed idea that they are relevant to the model, it was considered that the penalty applied should not eliminate the features. Therefore, with a less stringent penalty on the parameters, the Ridge model was opted to compare the results with the lasso model. In addition, the ElasticNet model was used which combines the Ridge and Lasso penalties and is especially useful in contexts where the nature of the variables considered is unknown.

Lasso Model

Lasso model was fitted using x_train and y_train. As for the hyper-parameters, the default alpha value of 1 was initially selected. This, however, returned a negative R2 score, indicating an extremely poor fitting and would result in predictions worse than just selecting the mean of the data. Thus, an alpha value of 0.01 was assigned.

A random state was also used to allow consistent iteration variables, which allows reliable model comparisons. The max_iter was chosen as 10000 to ensure convergence.

An R2 score of 0.272887 was obtained after fitting the model; this indicates underfitting.

Ridge Model

Similar to the Lasso regression, Ridge also puts a constraint on the coefficients by introducing a penalty factor. The main difference between the two is that Ridge applies a penalty equal to the square of the coefficients; hence, variables will never be reduced to 0. As all variables describe the nature of tasks, it can be assumed that all features are relevant to the target variable. Therefore, Ridge regression was selected for the second model.

The default alpha value of 1 was used. A Random state and max_iter of 10000 were chosen for the same reason as the Lasso model.

An R2 score of 0.649773 was obtained after fitting the model showing that the Ridge model is better than the Lasso model with these initial hyper-parameters.

ElasticNet Model

Additionally, an ElasticNet Regression model was fitted. This method can be seen as a combination of the Lasso and Ridge models by applying both the coefficient shrinkage methods. In theory, this will produce a model that overcomes the limitations of the previous two models.

With hyper-parameter values of 0.01 for alpha and l1 ratio of 0.5 (default), an R2 result of 0.46 was obtained, showing that this model is not as good as the Ridge model but better than the Lasso with initial hyper-parameters.s

3.3.2 ERROR & RMSE Calculation and Comparison (Test Group)

The error for each task is calculated as the difference in costs between the actual cheapest supplier and the predicted cheapest supplier for the corresponding task. From the EDA, it was found that S56 provides the best cost on average. The machine learning models were compared against the method of selecting the best performing supplier for all tasks (*Figure 3)*.

Lasso returned the largest RMSE and error distribution, indicating that it is the worst performing model at this stage. Despite showing similar RMSE values, the ridge model suggested only a single supplier (S39) for all tasks. This is not the case for ElasticNet (15 different suppliers) and Lasso (19 different suppliers).

The Lasso model has a wider distribution of errors than the ElasticNet model, indicating greater magnitude of errors. From this comparison, it can be concluded that up to this stage, the ElasticNet model is the best fitted model among the three. Even so, 'the one-supplier approach' determined by *Figure 3* is still the best performing approach to select a supplier.

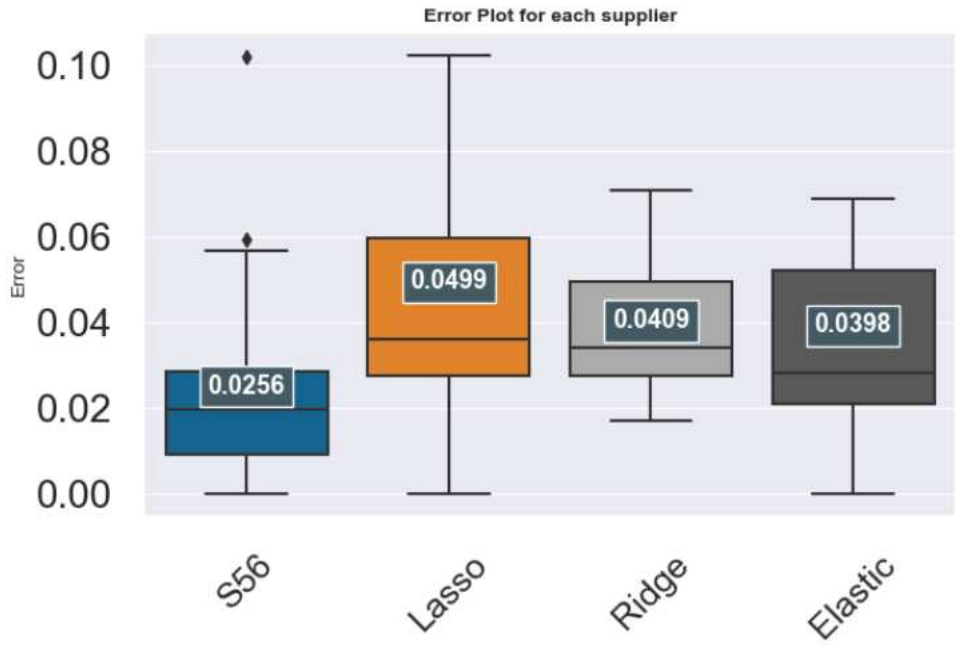


Figure 6. Error Distribution and RMSE comparison

3.4 Cross-Validation

Leave-One-Group-Out cross-validation was performed on *train data.* At each iteration, the model was trained considering 99 groups of tasks and its performance on one task was evaluated. A custom scoring function 'BeepyScorer', was created to calculate error and used for performance calculation.

The results indicates that the RMSE for Ridge (0.039967) are better than the ElasticNet (0.042252) and Lasso (0.057875) with the initial hyper-parameters. This means that Ridge is less sensitive when tested on new data meanwhile lasso is very sensitive because the number of suppliers recommended to perform each task: by ridge and lasso are different; 1 and 19, respectively.

3.5 Hyper-Parameter Optimization

GridSearch was utilised for Hyper-Parameter Optimization. Leave-One-Group-Out was used as a cross-validation splitting strategy and 'BeePy Scorer' as the scoring method.

3.5.1. First model – Lasso Regression

Various alpha value ranges were attempted. With the range shown on the first line of *Figure 7*, alpha values less than 0.001 return the same mean_test_score. Thereby, the range was narrowed down to between 0.1 and 0.001.

```
# param_grid = {"alpha":[1e-8, 1e-7, 1e-6, 1e-5,1e-4, 1e-3,1e-2, 1, 5, 10]}

alphas = np.linspace (0.1, 0.0001, 100)
param_grid={"alpha": alphas}
```

Figure 7. Hyper-parameter range for Lasso regression model

As a result, an alpha value of 0.00212 was obtained. In *Figure 8*, the trend of error by alpha value is displayed.

```
{'alpha': 0.0021181818181818246}
-0.03109728411
```

	params	mean_test_score	rank_test_score
0	{'alpha': 0.1}	-0.046304	7
1	{'alpha': 0.0989909090909091}	-0.046304	7
2	{'alpha': 0.09798181818181818}	-0.046304	7
3	{'alpha': 0.09697272727272728}	-0.046304	7
4	{'alpha': 0.09596363636363636}	-0.046304	7
...	...	...	...
95	{'alpha': 0.0041363636363636325}	-0.036067	5
96	{'alpha': 0.0031272727272727285}	-0.034047	2
97	{'alpha': 0.0021181818181818246}	-0.031097	1
98	{'alpha': 0.0011090909090909068}	-0.035272	3
99	{'alpha': 0.0001}	-0.035272	3

100 rows × 3 columns

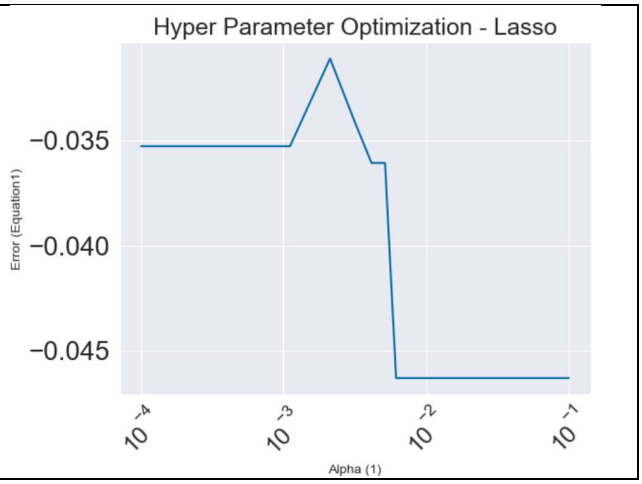


Figure 8. GridSearch Result: mean_test_score by different alpha values

Table 3 shows the R2 scores. A significant increment was observed in the result obtained from the model fitting with the best_param_ compared to model fitting with initial hyper-parameter.

Machine Learning	After GridSearch
0.27288	0.55808

Table 2. R2 Scores before and after using best_params_ obtained with GridSearch - Lasso

3.5.2. Second model – Ridge Regression

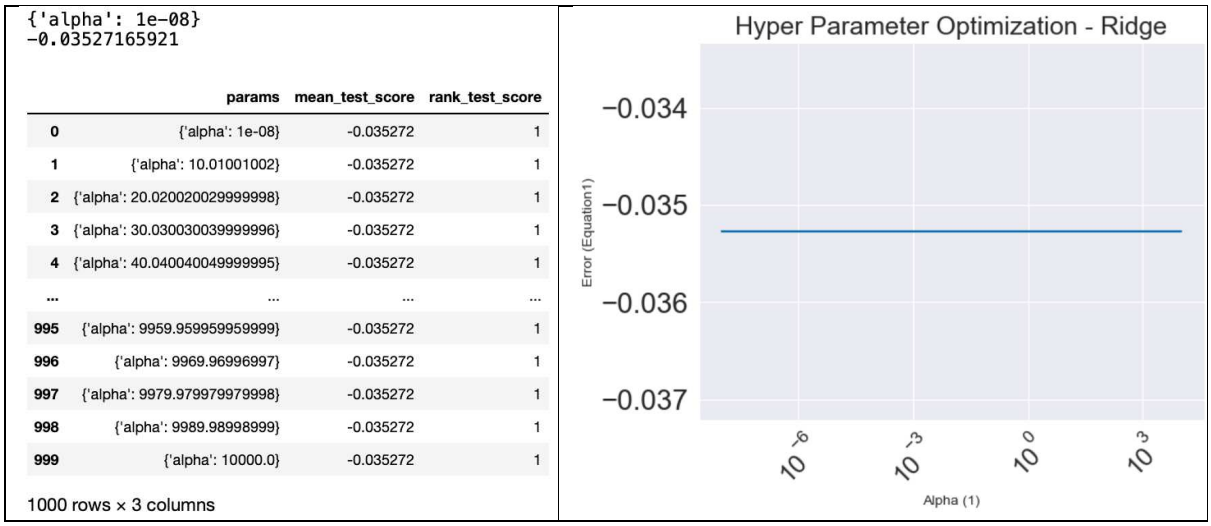
With the same approach used for Lasso, 1e-8 and 10000 range was applied for alpha.

```
# param_grid = {"alpha":[1e-8, 1e-7, 1e-6, 1e-5,1e-4, 1e-3,1e-2, 1, 5, 10]}

alphas = np.linspace (1e-8, 10000, 1000)
param_grid={"alpha": alphas}
```

Figure 9. Hyper-parameter range for Ridge regression model

Regardless of the alpha, error scoring result turned out to be the same as *Figure 10*. The reason is given by *Figure 11 which* shows the evolution of R2 score by different alphas selected by GridSearch. The difference in the variability explained by the model is never greater than 0.1 which means the suggested suppliers are still the same for all tasks no matter the size of alpha. Given that, it is expected that the calculation of the errors and the RMSE remain the same.



```
{'alpha': 1e-08}
-0.03527165921
```

	params	mean_test_score	rank_test_score
0	{'alpha': 1e-08}	-0.035272	1
1	{'alpha': 10.01001002}	-0.035272	1
2	{'alpha': 20.020020029999998}	-0.035272	1
3	{'alpha': 30.030030039999996}	-0.035272	1
4	{'alpha': 40.040040049999995}	-0.035272	1
...	...	...	...
995	{'alpha': 9959.959959959999}	-0.035272	1
996	{'alpha': 9969.96996997}	-0.035272	1
997	{'alpha': 9979.979979979998}	-0.035272	1
998	{'alpha': 9989.98998999}	-0.035272	1
999	{'alpha': 10000.0}	-0.035272	1

1000 rows × 3 columns

Figure 10. GridSearch Result of Ridge model: mean_test_score by different alpha values

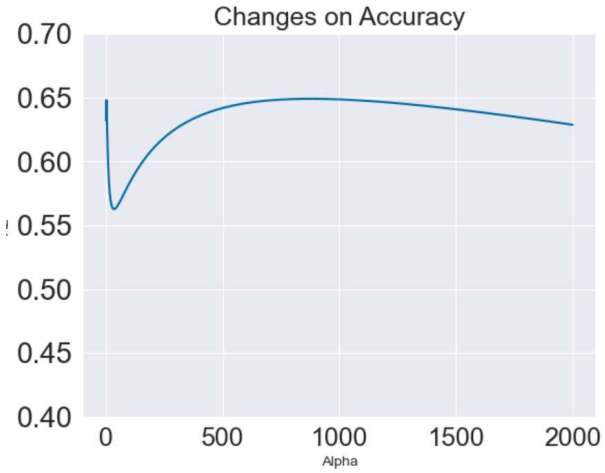


Figure 11. GridSearch for R2 Score of Ridge Regression Model

Table 3 shows the R2 scores. A drop is identified because it seems that the best possible alpha (1) was selected in the initial stage and the range considered during GridSearch does not contain that specific value.

Machine Learning	After GridSearch
0.64977	0.63214

Table 3. R2 Scores before and after using best_params_ obtained with GridSearch – Ridge

3.5.3. Third model – ElasticNet Regression

With the same approach used for previous models, the hyper-parameters' range was applied as *Figure 12 depicts*.

```
# param_grid = {"alpha":[1e-5,1e-4,1e-3,1e-2, 1, 10], "l1_ratio":[1e-5,1e-4, 1e-3,1e-2, 1]} #First range
param_grid = {"alpha":[0.01, 0.1, 1.0, 10, 100], "l1_ratio":np.linspace (0.1,0.001, 100)} #Second range
```

Figure 12. Hyper-parameter range for ElasticNet regression model

As a result, the best combination found was alpha of 0.1 and l1_ratio 0.0230 (*Figure 13 Green line graph*).

```
{'alpha': 0.1, 'l1_ratio': 0.023000000000000007}
-0.031047226130000005
```

	params	mean_test_score	rank_test_score
0	{'alpha': 0.01, 'l1_ratio': 0.1}	-0.035272	27
1	{'alpha': 0.01, 'l1_ratio': 0.099}	-0.035272	27
2	{'alpha': 0.01, 'l1_ratio': 0.098}	-0.035272	27
3	{'alpha': 0.01, 'l1_ratio': 0.097}	-0.035272	27
4	{'alpha': 0.01, 'l1_ratio': 0.096}	-0.035272	27
...	...	...	...
495	{'alpha': 100, 'l1_ratio': 0.0050000000000000044}	-0.046304	164
496	{'alpha': 100, 'l1_ratio': 0.0040000000000000036}	-0.046304	164
497	{'alpha': 100, 'l1_ratio': 0.0030000000000000027}	-0.046304	164
498	{'alpha': 100, 'l1_ratio': 0.0020000000000000018}	-0.046304	164
499	{'alpha': 100, 'l1_ratio': 0.001}	-0.046304	164

500 rows × 3 columns

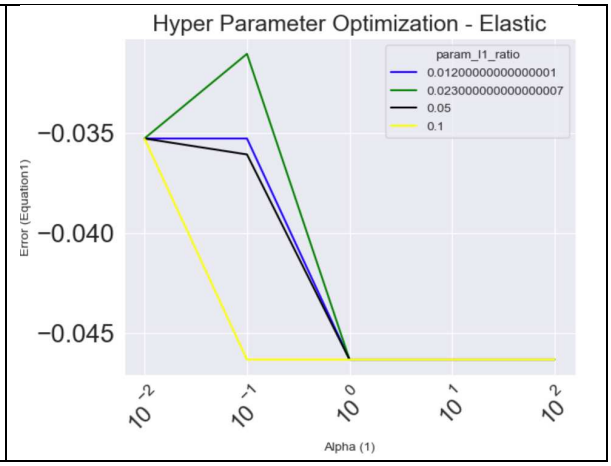


Figure 13. GridSearch Result of ElasticNet model: mean_test_score by different alpha values

4. Conclusion

In case of Lasso, the value obtained from the cross-validation increased compared to the RMSE result carried out by the model fitting with the initial hyper-parameters, whereas Ridge and ElasticNet models show only slight changes. The main reason for these distinct trends is the different sensitiveness to validation sets.

The Lasso model depicts drastic improvements due to optimized hyper-parameters, almost matching the optimized ElasticNet predictions. There was no change on RMSE of the Ridge model even after hyper-parameter optimization (established in *Figure 10*).

The Elastic model is the best model for supplier recommendation. The Lasso model did, however return a similar result. The only difference between the two models is the initial hyper-parameter setting, which for lasso model led to its poor RMSE score.

Nevertheless, the main conclusion of this analysis is that even with the best hyper-parameter settings, the implemented models do not perform better than the manual approach. Therefore, the final suggestion to Acme Corporation is either to hire Supplier 56 for any task based on their relative costs or consider a different model.

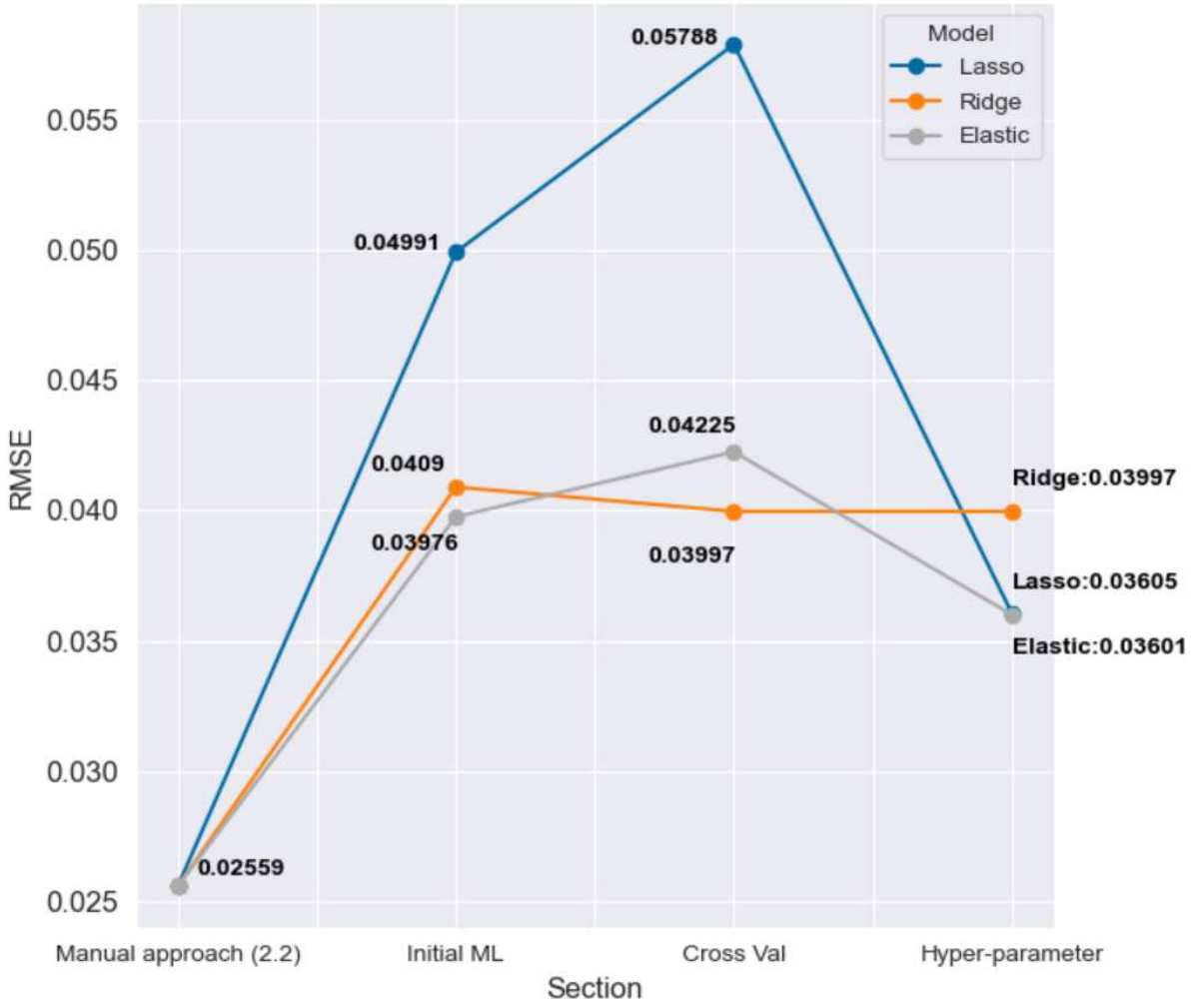


Figure 14. Comparison of RMSE for all approaches

Model	Manual approach (2.2)	Initial ML	Cross Val	Hyper-parameter
Lasso	0.025594	0.049908	0.057875	0.036048
Ridge	0.025594	0.040904	0.039967	0.039967
Elastic	0.025594	0.039764	0.042252	0.036005

Table 4. Comparison of RMSE for all approaches

Appendix: Python File