Best Supplier Suggestion
using Machine Learning

FOR ACME CORPORATION

Prepared by: BeePy

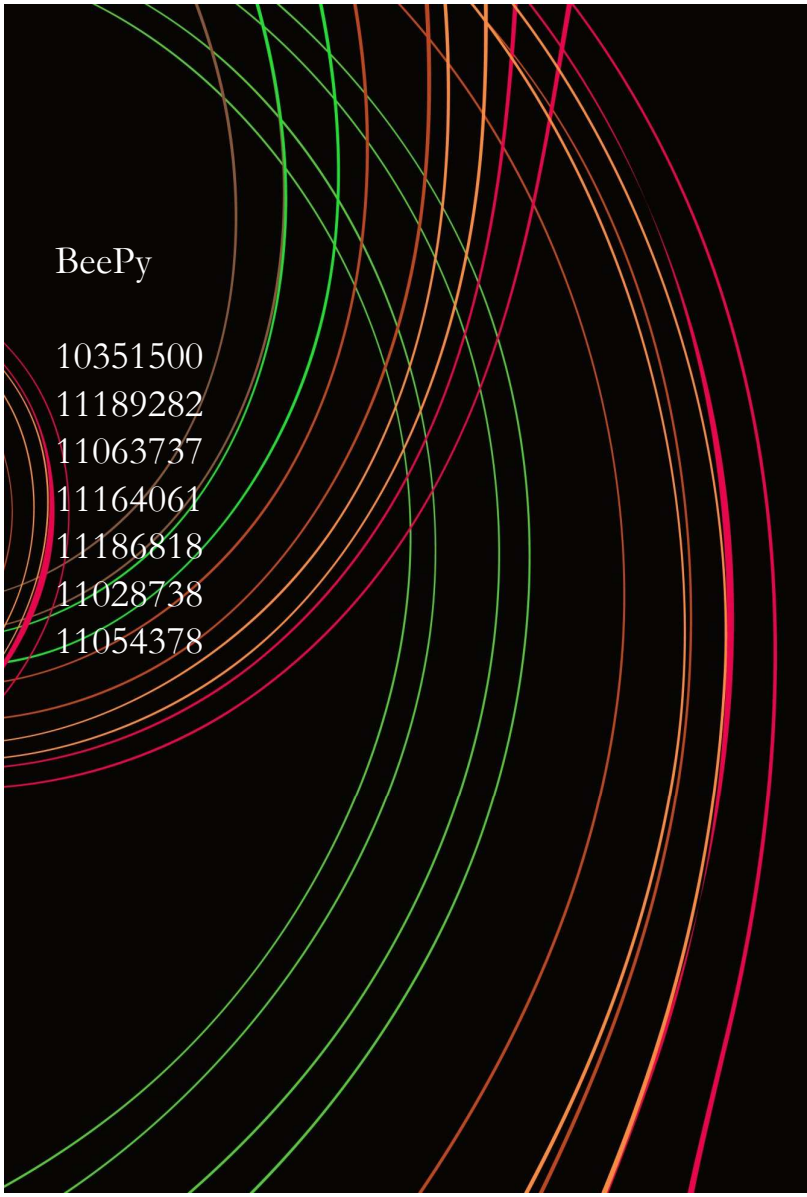


Table of Contents

Introduction: Data Description

1. Data Preparation
2. Exploration Data Analysis
3. Machine Learning
4. Cross-Validation
5. Hyper-Parameters Optimization

Data Description

- 130 Task IDs
- 116 Task Features

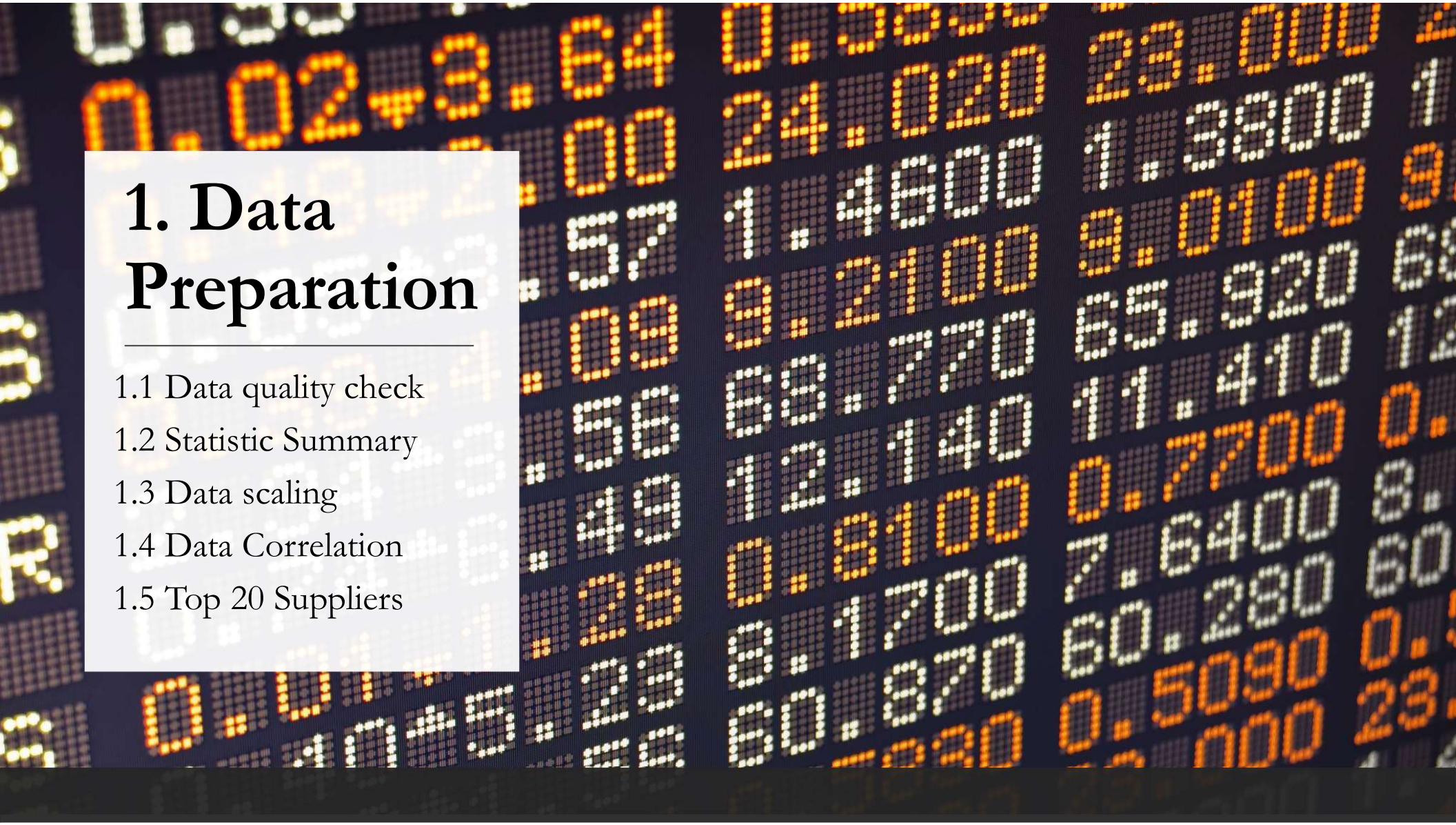
Task.xlsx

- 64 Suppliers
- 18 Supplier features

Suppliers.csv

- 120 Task IDs & 64 Supplier IDs
- 7680 values

Cost.csv



1. Data Preparation

1.1 Data quality check

1.2 Statistic Summary

1.3 Data scaling

1.4 Data Correlation

1.5 Top 20 Suppliers

MHBH0

1.1.Data Quality Check

1. Format modification of Task ID (DataTime format)

❶ Before

```
tasks['Task ID'].head(2)
```

```
0    2019 05 30
1    2019 09 26
Name: Task ID, dtype: object
```

```
costs['Task ID'].head(2)
```

```
0    30/05/2019
1    30/05/2019
Name: Task ID, dtype: object
```

❷ After

```
tasks['Task ID'].head(2)
```

```
0   2019-05-30
1   2019-09-26
Name: Task ID, dtype: datetime64[ns]
```

```
costs['Task ID'].head(2)
```

```
0   2019-05-30
1   2019-05-30
Name: Task ID, dtype: datetime64[ns]
```

```
#To ensure deleting any whitespace from Task ID
tasks['Task ID']=tasks['Task ID'].str.replace(' ', '')
costs['Task ID']=costs['Task ID'].str.replace(' ', '')
```

```
#Format change
costs['Task ID']=pd.to_datetime(costs['Task ID'],infer_datetime_format=True, \
                                format='%d/%m/%y', dayfirst=True)
tasks['Task ID']=pd.to_datetime(tasks['Task ID'],infer_datetime_format=True, \
                                format='%d/%m/%y', dayfirst=True)
```

Slide 5

MHBH0 Show the different date formats!

Mohamed Hadhry Bin Haslimi, 2022-12-14T15:40:31.554

```
print('Number of NA values in suppliers dataset: ', suppliers.isna().sum().sum())
print('Number of NA values in costs dataset: ', costs.isna().sum().sum())
print('Number of NA values in tasks dataset: ', tasks.isna().sum().sum())
```

```
Number of NA values in suppliers dataset:  0
Number of NA values in costs dataset:  0
Number of NA values in tasks dataset:  0
```

1.1.Data Quality Check

2. Missing Value Detection

Dropping Task ID(rows): not related Cost Values

```
print('Number of tasks in the tasks dataset:',tasks['Task ID'].nunique(),
      '\nNumber of tasks in the Costs dataset:',costs['Task ID'].nunique())

costs_id=costs['Task ID'].unique()
#We get an array with all the tasks present in the costs table

tasks=tasks.loc[tasks['Task ID'].isin(costs_id)]

print('Number of tasks in task dataset(Excluding Task ID is not in Costs):', len(tasks))
```

```
Number of tasks in the tasks dataset: 130
Number of tasks in the Costs dataset: 120
Number of tasks in task dataset(Excluding Task ID is not in Costs): 120
```

1.1.Data Quality Check

3. Removing tasks with no cost information

Supplier dataset

	max	min	mean	std	variance
SF1	1000.000	10.000	353.125	441.163	191583.984
SF2	2000.000	100.000	1031.250	757.371	564648.438
SF3	2000.000	100.000	1031.250	757.371	564648.438
SF4	500.000	5.000	176.562	220.582	47895.996
SF5	200.000	2.000	70.625	88.233	7663.359
SF6	100.000	1.000	35.312	44.116	1915.840
SF7	3.000	0.000	1.969	0.835	0.687
SF8	500.000	8.000	188.750	213.615	44918.438
SF9	4000.000	200.000	2062.500	1514.742	2258593.750
SF10	500.000	10.000	196.875	209.359	43146.484
SF11	2000.000	100.000	1031.250	757.371	564648.438
SF12	8.000	1.000	4.688	2.800	7.715
SF13	2000.000	100.000	1031.250	757.371	564648.438
SF14	2000.000	100.000	1031.250	757.371	564648.438
SF15	2000.000	50.000	828.125	818.044	658740.234
SF16	8000.000	500.000	4531.250	3010.399	8920898.438
SF17	50000.000	0.000	17307.812	18934.686	352920407.715
SF18	98.000	90.000	94.750	3.333	10.938

Task dataset

	max	min	mean	std	variance
TF1	706.000	0.000	83.100	190.003	35800.257
TF2	7908.000	1127.000	4436.525	1515.095	2276383.483
TF3	769905777.000	71490927.000	504524365.108	112022687.975	12444506932623140.000
TF4	3344.000	367.000	1516.933	685.000	465314.462
TF5	0.642	0.144	0.337	0.079	0.006
...	...	...	...	...	...
TF111	69920452.000	0.000	45878221.983	13923461.065	192247244973766.531
TF113	1211.000	30.000	382.808	286.252	81257.255

1.2. Statistic Summary

DJ0

1.2 Statistic Summary – Remove low variance features

❶ Drop the column if the variance is less than 0.01

1. Tasks

```
to_drop = [column for column in tasks.iloc[:,1:].columns \
           if (np.var(tasks[column]) <= 0.0100)]
tasks.drop(to_drop, axis=1, inplace=True)
print(len(to_drop))
```

```
35
```

```
len(tasks.columns)
#Total number of columns changed from 117 to 82
```

```
82
```

❷ Check the length of each dataset after dropping the columns

2. Suppliers

```
to_drop = [column for column in suppliers.iloc[:,1:].columns \
           if (np.var(suppliers[column]) <= 0.0100)]
tasks.drop(to_drop, axis=1, inplace=True)
print(len(to_drop))
```

```
0
```

```
len(suppliers.columns)
#Total number of supplier columns did not change
```

```
19
```

Slide 9

DJ0 Up to here [@Suparna Roy]
Dahye Jeong, 2022-12-14T16:07:37.484

DJ0

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-1, 1))
#Create the object "Scaler"
```

```
tasks.iloc[:,1:] = scaler.fit_transform(tasks.iloc[:,1:])
#To avoid touching the first column (ID) we use iloc in both parts
```

```
tasks.head()
```

	Task ID	TF1	TF2	TF3	TF4	TF6	TF8	TF10	TF12	TF14	...
0	30/05/2019	1.000000	-0.583542	0.022445	-1.000000	-0.821200	-0.454300	0.059902	-0.697013	-0.090516	...
1	26/09/2019	0.974504	-0.683822	-0.088391	-0.957004	-1.000000	-0.786583	-0.333066	-0.931721	-0.394759	...
2	29/11/2019	-0.257790	-0.106621	0.227168	-0.232113	-0.183833	-0.291310	0.092017	-0.184922	0.220932	...
3	03/01/2020	0.328612	-0.050583	0.364002	-0.322136	-0.174803	-0.212967	0.201342	-0.153627	0.409446	...
4	07/01/2020	0.572238	-0.172099	0.289163	-0.548539	-0.417667	-0.351643	0.095886	-0.369844	0.243666	...

5 rows × 84 columns

```
#As per the insturction "conversion to all feature", supplier features also scaled.
suppliers.iloc[:,1:]=scaler.fit_transform(suppliers.iloc[:,1:])
```

```
suppliers.head()
```

	Supplier ID	SF1	SF2	SF3	SF4	SF5	SF6	SF7	SF8	SF9	
0	S1	-0.818182	-0.052632	-0.052632	-0.818182	-0.818182	-0.818182	0.333333	-0.707317	-0.052632	-0.63
1	S2	-0.818182	-0.052632	-0.052632	-0.818182	-0.818182	-0.818182	-1.000000	-0.707317	-0.052632	-0.63
2	S3	-0.818182	-0.052632	-0.052632	-0.818182	-0.818182	-0.818182	0.333333	-0.707317	-0.052632	-0.63

1.3. Data Scaling

Slide 10

DJ0 [@Afifa Pirzada] Presentation preparation from here to the 16
Dahye Jeong, 2022-12-14T22:45:27.631

1.4. Data Correlation

```
tasks_corr = tasks.iloc[:,1:].corr().abs()

tasks_corr.style.background_gradient(cmap='coolwarm',axis=None).format(precision=2)
#Converting correlation matrix in a heatmap.

masks = np.zeros_like(tasks_corr, dtype=bool)
masks[np.triu_indices_from(masks)] = True
#To avoid redundancy of the matrix (no repeated correlation coefficients)

tasks_corr[masks] = np.nan

tasks_corr_styled=(tasks_corr
                   .style
                   .format(precision=2)
                   .background_gradient(cmap='coolwarm', axis=None, vmin=0, vmax=1)
                   .highlight_null(null_color='#f1f1f1'))
```

Slide 11

DJ0 [@Sergio Ignacio Chavez Lazo] Do we call this coloured data frame as heatmap ??
Dahye Jeong, 2022-12-06T23:16:08.910

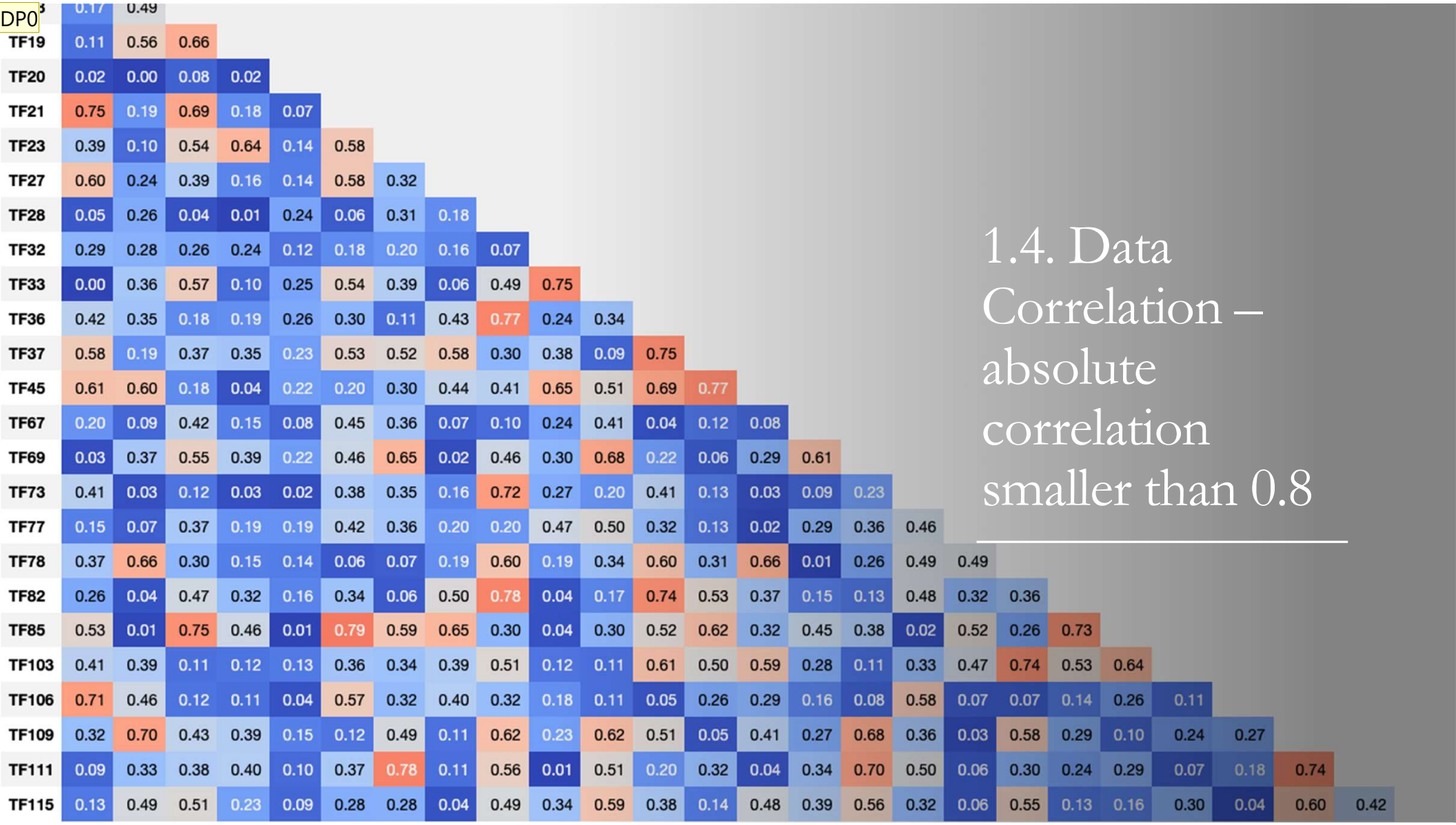
```
for i in range(len(tasks2.columns)):
    corr=pd.DataFrame(tasks2.iloc[:,1:].corr().abs().unstack()).reset_index()\
    .rename(columns={'level_0': 'FeatureGroup1', 'level_1': 'FeatureGroup2',0:'Corr'})
    #.corr().abs(): Calculation of absolute correlation.
    #.unstack: Turn all index to Column ()
    #.reset_index: Reset the index because we will need the variable 1 for groupping.
    #.rename: Column name renaming to Feature_Group1, Feature_Group2 and Corr

    corr['Number']=corr.Corr>=0.8
    #New column: Corr is larger than or equal to 0.8(True) or not(False).

    corr_group=corr.groupby('FeatureGroup1').sum('Number').sort_values('Number',ascending=False).head(1)
    #.Groupby 'Feature_Group1'
    #.sum('Number'): sum of True(greater than 0.8)
    #.sort_values.head(1): sort the value and get the first row only.

    lista=corr_group.index.tolist()
    #Store the row(feature) with greater than 0.8 corr.

    if corr_group.Number[0]>1:
        tasks2=tasks2.drop(columns=lista)
        #Drop the column using stored the feature(lisa)
```

```
print("After the process, the number of features in tasks dataset changed from ",\
      len(tasks.iloc[:,1:].columns),"to ",len(corr['FeatureGroup1'].unique()))
```

```
After the process, the number of features in tasks dataset changed from  81 to  27
```

1.4. Data Correlation –

remove the most correlated features



1.4. Data Correlation – absolute correlation smaller than 0.8

Slide 13

DP0 [@Dahye Jeong] Should we go with these variables like 'tasks2' & 'tasks_corr_styled2' or change some variables name? Just confirming thoughts?
Devesh Prasad, 2022-12-08T08:40:10.462

DP0 0 For everyone also?
Devesh Prasad, 2022-12-08T08:40:21.634

DJ0 1 [@Devesh Prasad] Any suggestion for changing variables name?
Dahye Jeong, 2022-12-08T09:42:57.230

SR0 2 Let’s call it tasks2 as we need to otherwise refine the entire code
Suparna Roy, 2022-12-08T11:48:50.461

SR0 3 But would be good to have intuitive var names (others should also vote)
Suparna Roy, 2022-12-08T11:49:27.074

1.5. Top 20 Suppliers

❶ Add ranking column by Task ID

```
costs1=costs.set_index(['Task ID','Supplier ID']).sort_values(by=['Task ID','Cost'])
#Establish index to better readability

costs1['Ranking']=costs1.groupby('Task ID')['Cost'].rank()
#Get ranking of cost(Supplier) by Task ID

costs1.head(68)
```

		Cost	Ranking
Task ID	Supplier ID		
2019-05-30	S57	0.290967	1.0
	S40	0.293925	2.0
	S24	0.296388	3.0
	S56	0.302213	4.0
	S32	0.302394	5.0
	...	...	...
	S3	0.521679	64.0
2019-09-26	S19	0.283752	1.0
	S30	0.285043	2.0
	S14	0.289172	3.0
	S57	0.292823	4.0

68 rows × 2 columns

❷ Making an Array of Top 20 ranking suppliers

```
#Now we are going to find those Suppliers that never appear in the top 20 of cheapers
costs_top=costs1[costs1['Ranking']<21].reset_index()
#cost dataframe with only top 20 suppliers by each Task ID

sup_top=costs_top['Supplier ID'].unique()
#Array of any suppliers included in the top 20 ranking

print("At least one or more times appeared in TOP 20 ranking",len(sup_top), \
      "out of total suppliers",len(costs['Supplier ID'].unique()))
#63 suppliers appeared in the top 20 ranking at least one or more times
```

```
At least one or more times appeared in TOP 20 ranking 63 out of total suppliers 64
```

❸ Update the dataset
with suppliers included in top 20 ranking array

```
#Cost DataFrame
print("Original supplier number in Cost DataFrame:", len(costs))
#Keep 63 supplier appeared in top 20 ranking
costs=costs[costs['Supplier ID'].isin(sup_top)]
print("After removing always expensive supplier(Top 20):", len(costs))

#Supplier DataFrame
print('Original supplier number in Suppliers DataFrame:', len(suppliers))
#Keep 63 supplier appeared in top 20 ranking
suppliers=suppliers[suppliers['Supplier ID'].isin(sup_top)]
print('After removing always expensive supplier(Top 20):', len(suppliers))
```

```
Original supplier number in Cost DataFrame: 7680
After removing always expensive supplier(Top 20): 7560
Original supplier number in Suppliers DataFrame: 64
After removing always expensive supplier(Top 20): 63
```

DJ0

2. Exploratory Data Analysis

2.1 Boxplot

– Distribution of feature values for each task

2.2 Boxplot

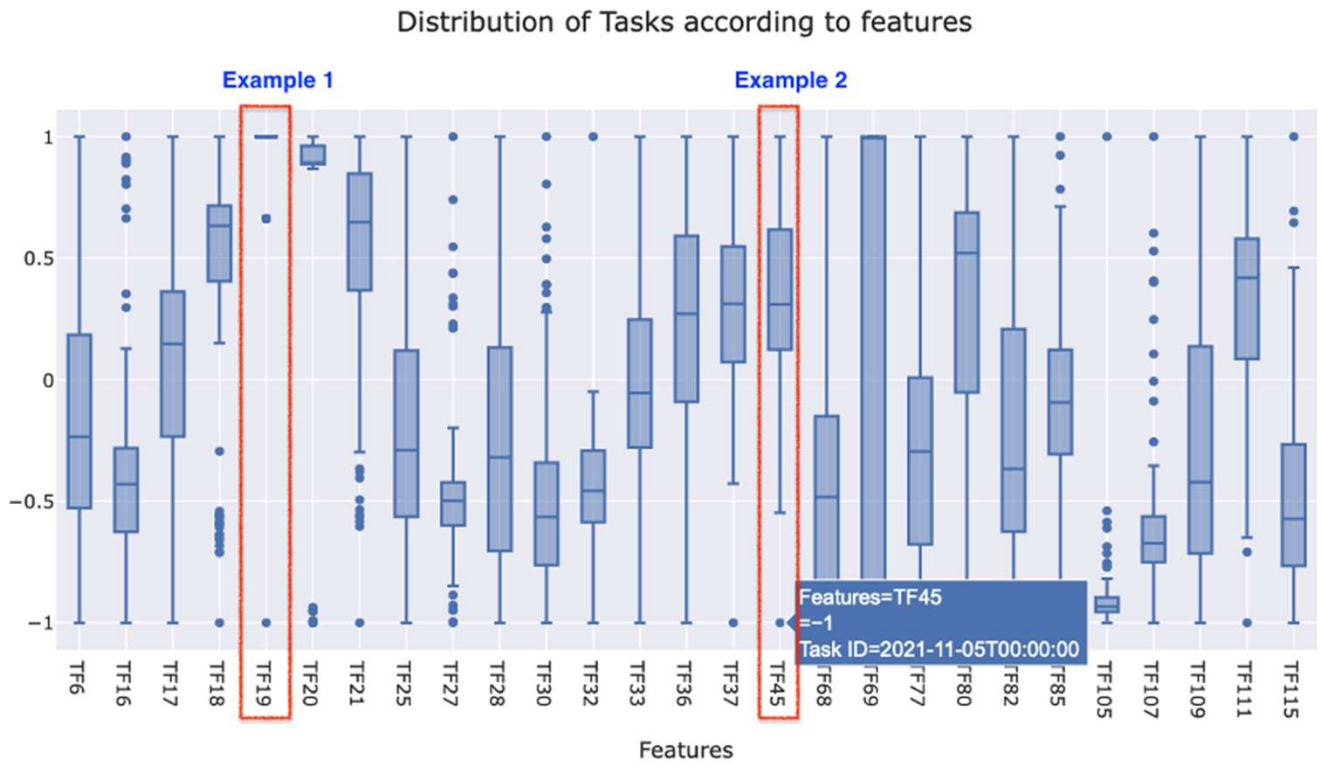
– Distribution of errors for each supplier

2.3 Heatmap Plot

– Cost values as a matrix of tasks (rows) and suppliers (columns)

Slide 15

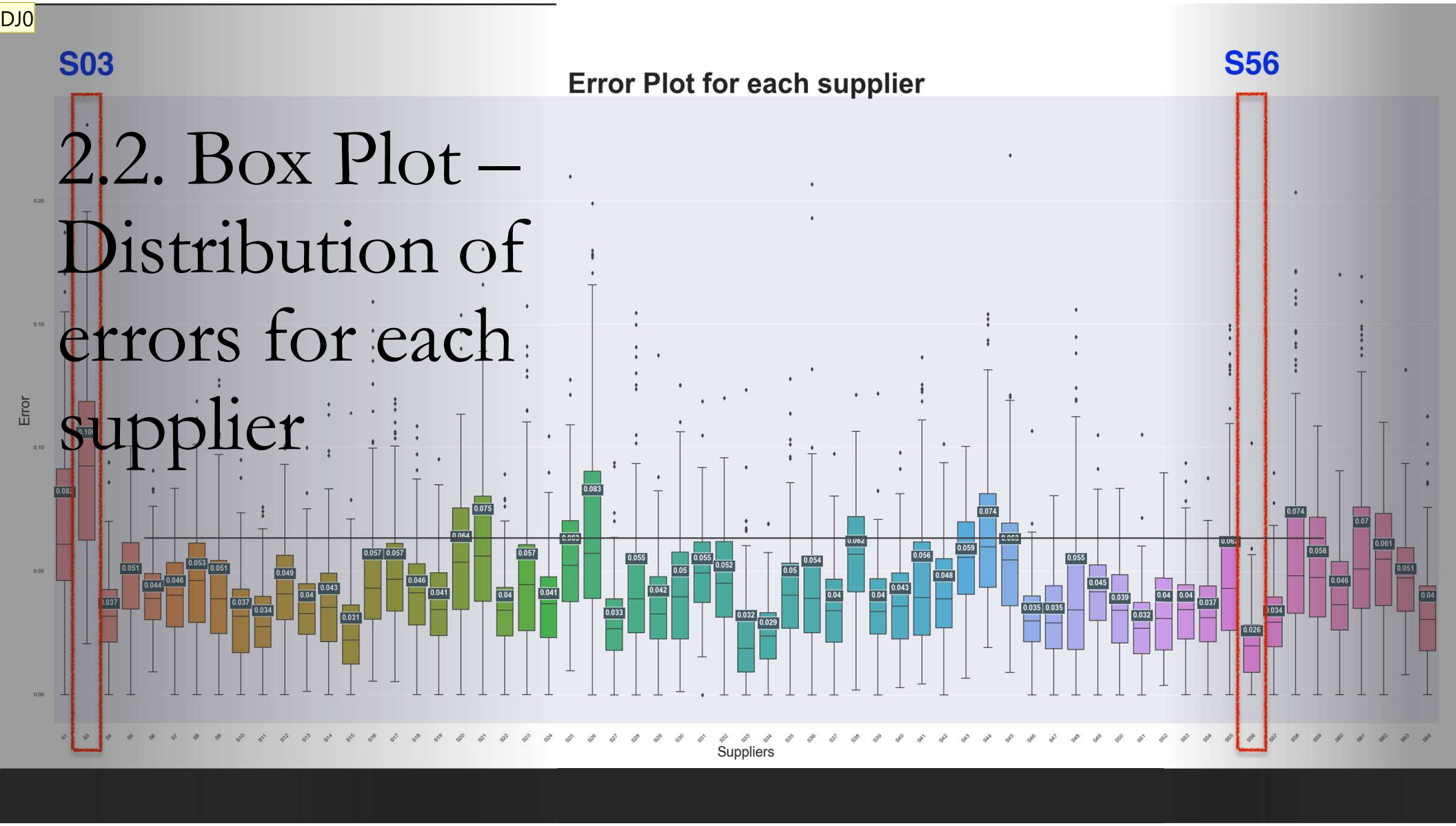
DJ0 Up to here: Afifa
[@Afifa Pirzada]
Dahye Jeong, 2022-12-14T16:08:17.672



2.1 Box Plot – Distribution of tasks per feature

DJ0

2.2. Box Plot – Distribution of errors for each supplier



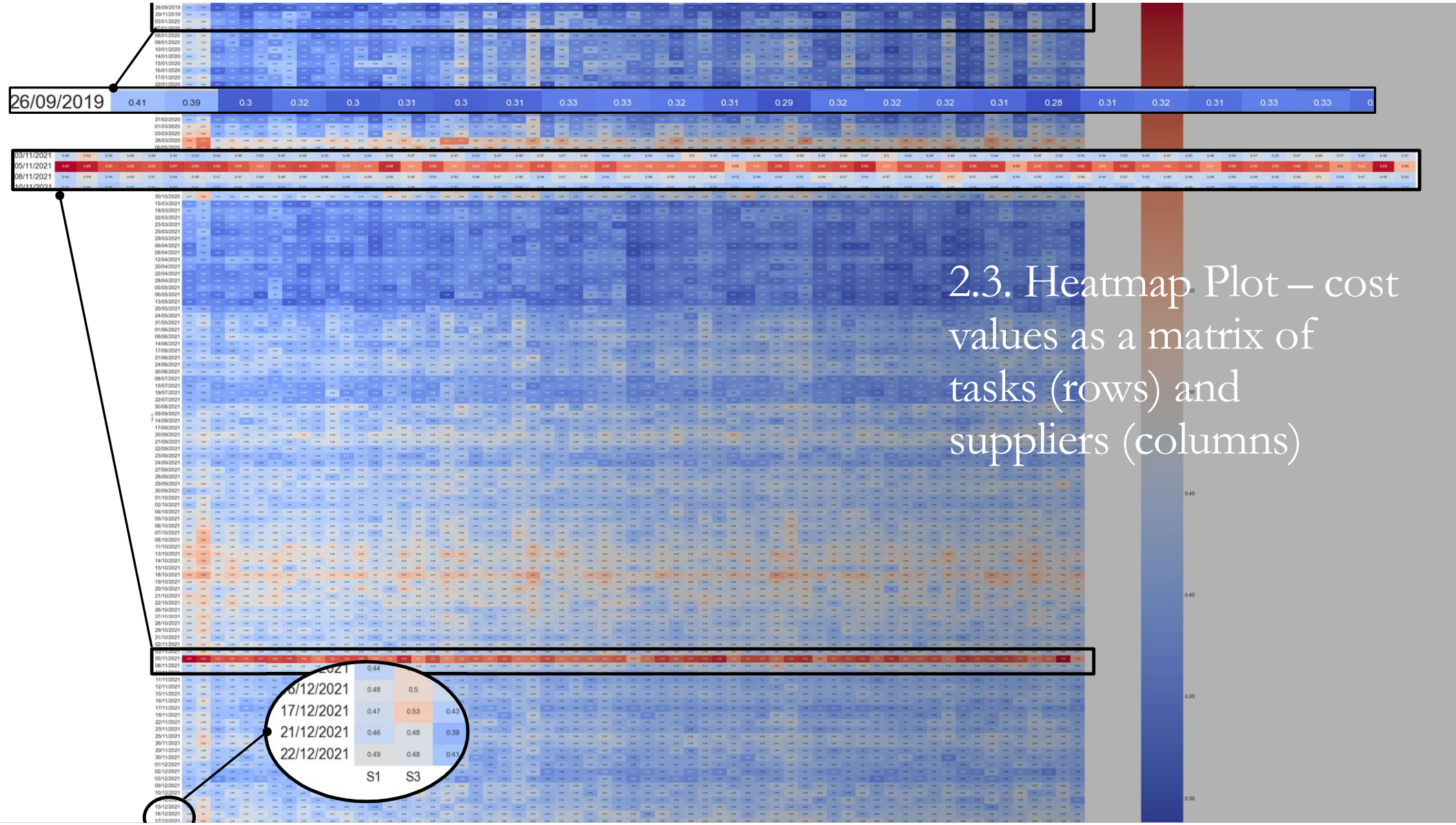
Slide 17

DJ0 Are we going to mention/show the error equation or RMSE equation somewhere in ppt? Maybe section 3? comparison??
Dahye Jeong, 2022-12-14T23:03:38.967

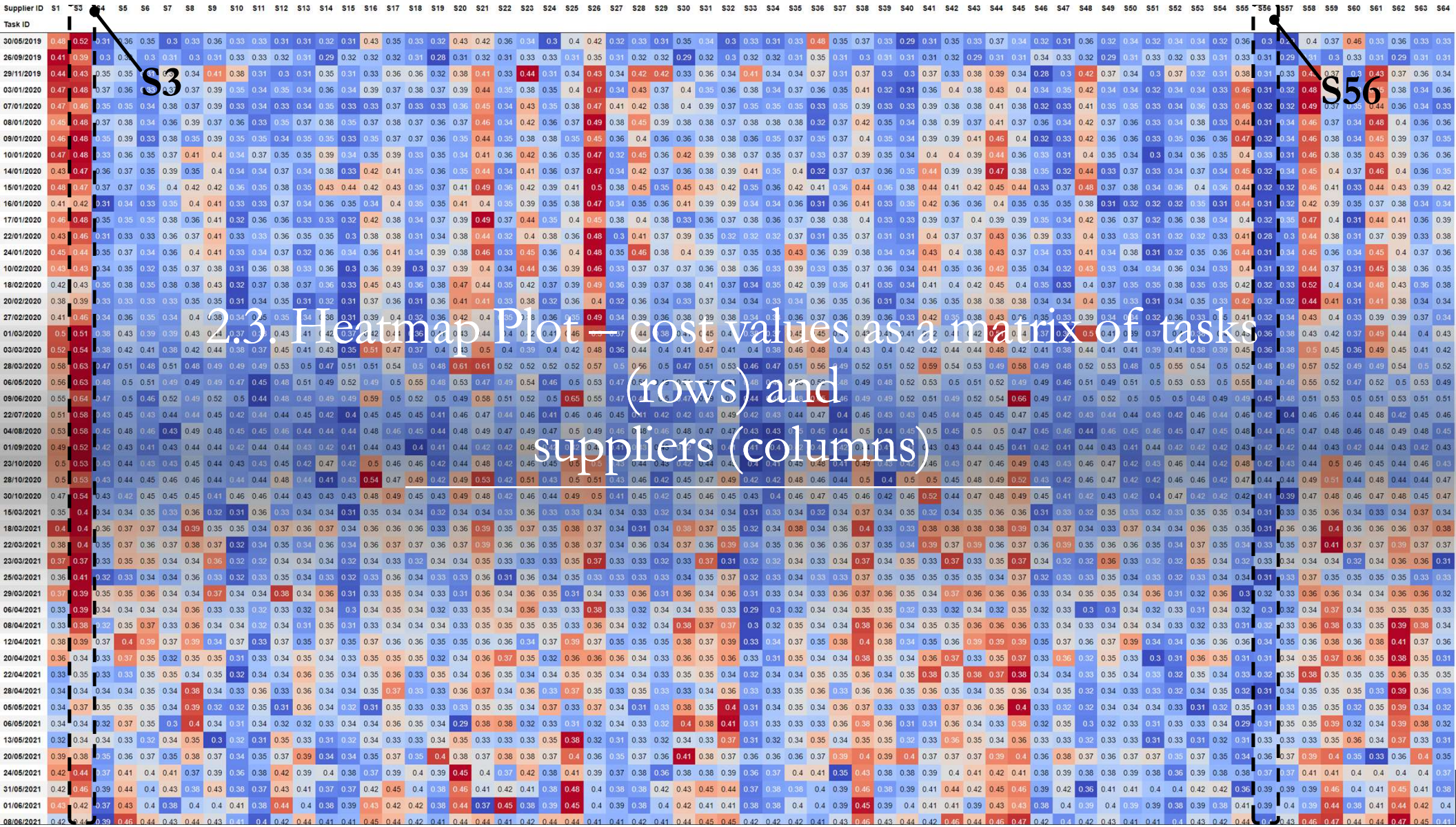
SR0 0 Well, it was mentioned to not quote the equations. Not sure, if that was just for the report
Suparna Roy, 2022-12-15T00:05:53.066

DJ0 1 Ah, I see. The equation helped me understand better. It's okay NeverMind! :)
Dahye Jeong, 2022-12-15T00:35:58.169

SR0 2 Let's discuss tomorrow and we can take a group consensus. :)
Suparna Roy, 2022-12-15T00:39:18.178



2.3. Heatmap Plot – cost values as a matrix of tasks (rows) and suppliers (columns)


S3
S56
2.3. Heatmap Plot – cost values as a matrix of tasks (rows) and suppliers (columns)

DJ0

3. Model Fitting & Scoring

3.1 Combining & Splitting the Datasets(x and y)

3.2 Dividing TestGroup (20 unique task IDs) & TrainGroup

3.3 Training the Model (Lasso & Ridge Regression Models)

3.4 Compute the Error and RMSE Score

3.5 Comparison of Error and RMSE score (Section 2.2 and Section 3.4)

* Section 2.2: Error and RMSE score by suppliers
* Section 3.4: Error and RMSE score after Machine Learning Model fitting

Slide 20

DJ0 Up to here: Hadry
[@Mohamed Hadhry Bin Haslimi]
Dahye Jeong, 2022-12-14T16:08:59.869

DJ0

1 Merging Dataset

```
df_1 = pd.merge(costs, tasks2, how='inner', on='Task ID')
df = pd.merge(df_1, suppliers,how='inner',on='Supplier ID')
print('The number of rows in the costs dataframe were: ', str(len(costs)), \
      ' and the number of rows of the new dataframe is: ',str(len(df)))
```

```
The number of rows in the costs dataframe were:  7560  and the number of rows of the new dataframe is:  7560
```

2 Sorting the order of columns

```
cols_to_order = ['Task ID','Supplier ID'] #Columns to be placed at the beginning
cols_to_order2=['Cost'] #Columns to be placed at the end

cols=cols_to_order+cols_to_order2
new_columns = cols_to_order + (df.columns.drop(cols).tolist()) + cols_to_order2
#To avoid writing everything again and rearrange columns more efficiently

df = df[new_columns] #Create the ordered dataframe
df_copy=copy.deepcopy(df) #Before dropping the column of suppliers we create a copy
df
```

	Task ID	Supplier ID	TF6	TF16	TF17	TF18	TF19	TF20	TF21
0	2019-05-30	S1	-0.821200	-0.559193	-0.350672	-0.294403	1.000000	0.961792	-0.379070

3.1 Combining the datasets

Slide 21

DJ0 [@Samit Shrinivas Uttarkar] Presentation preparation from here to the 27
Dahye Jeong, 2022-12-14T22:44:48.034

3.1 Splitting the dataset

Splitting the dataset into x, y and Groups.

❶ Splitting Dataset (x and y)

```
x=df.iloc[:,2:-1] #Tasks features
y=df.iloc[:,-1] #Costs
Groups=df.iloc[:,0] #Task ID
```

❷ Data x

	TF6	TF16	TF17	TF18	TF19	TF20	TF21	TF25	TF27	TF28	...	SF9	SF10	S
0	-0.821200	-0.559193	-0.350672	-0.294403	1.000000	0.961792	-0.379070	0.224388	-0.599289	-0.746568	...	-0.052632	-0.632653	-0.052
1	-1.000000	0.017794	-0.560807	-1.000000	0.662732	0.961792	-1.000000	0.297053	-0.599289	-0.833528	...	-0.052632	-0.632653	-0.052
2	-0.183833	-0.424278	-0.642854	-0.659203	0.662732	-0.951252	-0.001434	0.446932	-0.599289	-0.285041	...	-0.052632	-0.632653	-0.052
3	-0.174803	-0.435824	-0.627916	-0.636074	0.662732	0.886693	-0.017407	0.571802	-0.599289	-0.214910	...	-0.052632	-0.632653	-0.052
4	-0.417667	-0.353895	-0.624852	-0.709065	0.662732	0.886693	-0.147246	0.571341	-0.599289	-0.311000	...	-0.052632	-0.632653	-0.052

❸ Data y

```
y.head()
```

```
0    0.478219
1    0.407784
2    0.437884
3    0.473479
4    0.466028
Name: Cost, dtype: float64
```

❹ Groups: Task ID

```
Groups.head()
```

```
0    30/05/2019
1    26/09/2019
2    29/11/2019
3    03/01/2020
4    07/01/2020
Name: Task ID, dtype: object
```

❶ Selecting 20 unique Task IDs

```
random.seed(50)
sampletasks=Groups.unique()
TestGroup = random.sample(list(sampletasks), 20)
print(TestGroup)
len(TestGroup)
```

```
['21/09/2021', '01/12/2021', '29/03/2021', '31/05/2021', '15/10/2021', '14/12/2021', '22/03/2021', '27/10/2021', '14/09/2021', '12/11/2021', '06/05/2021', '16/01/2020', '28/09/2021', '28/04/2021', '30/10/2020', '22/10/2021', '01/10/2021', '22/12/2021', '03/03/2020', '20/05/2021']
```

```
20
```

3.2 20 Unique Task IDs

TestGroups

❶ Splitting Dataset (trains & tests datasets)

```
x_train=trains.iloc[:,2:-1] #X train (Tasks and Suppliers features of Tasks ID's that were NOT selected in the TestGroup)
x_test=tests.iloc[:,2:-1] #X test (Tasks and Suppliers features of Tasks ID's that were selected in the TestGroup )
y_train=trains.iloc[:,-1] #Y train (Costs of Tasks ID's that were NOT selected in the TestGroup)
y_test=tests.iloc[:,-1] #Y test (Costs of Tasks ID's that were selected selected in the TestGroup)
```

❷ x_train

```
x_train.head()
```

	TF6	TF16	TF17	TF18	TF19	TF20	TF21	TF25	TF27	TF28	...	SF9	SF10
0	0.40658	-0.803796	-0.745126	-0.540603	0.662732	-1.0	0.672742	0.718965	-0.476023	0.045788	...	1.000000	1.000000
1	0.40658	-0.803796	-0.745126	-0.540603	0.662732	-1.0	0.672742	0.718965	-0.476023	0.045788	...	-0.052632	-1.000000

❸ x_test

```
x_test.head()
```

	TF6	TF16	TF17	TF18	TF19	TF20	TF21	TF25	TF27	TF28	...	SF9	SF1(
0	-0.153113	-0.338236	0.204198	0.737921	1.0	1.0	0.706328	-0.334539	-0.52944	-0.460089	...	-0.052632	-1.00000(
1	-0.153113	-0.338236	0.204198	0.737921	1.0	1.0	0.706328	-0.334539	-0.52944	-0.460089	...	-1.000000	-0.63265
2	-0.153113	-0.338236	0.204198	0.737921	1.0	1.0	0.706328	-0.334539	-0.52944	-0.460089	...	-0.052632	1.00000(
3	-0.153113	-0.338236	0.204198	0.737921	1.0	1.0	0.706328	-0.334539	-0.52944	-0.460089	...	1.000000	-0.63265

❹ y_train

```
y_train.head()
```

```
0    0.396702
1    0.412103
2    0.376691
3    0.429424
4    0.367398
Name: Cost, dtype: float64
```

❺ y_test

```
y_test.head()
```

```
0    0.427725
1    0.430542
2    0.428300
3    0.416030
4    0.418590
Name: Cost, dtype: float64
```

3.2 Training and Testing Datasets

3.3 Training the Model

- Lasso Regression Model

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha = 0.01, random_state=42, max_iter=10000)
lasso.fit(x_train, y_train)

pred_lasso = lasso.predict(x_test)
pred_lasso

lasso.score(x_test,y_test) #R2 score
```

```
0.2728868228826856
```

DJ0

3.3 Training the Model

- Ridge Regression Model

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha = 1,random_state=42,max_iter=10000)
ridge.fit(x_train, y_train)

pred_ridge = ridge.predict(x_test)
pred_ridge

ridge.score(x_test,y_test) #R2 score
```

```
0.6497735526825286
```

Slide 26

DJ0 @Sergio Ignacio Chavez Lazo Presentation preparation from here to the 34
Dahye Jeong, 2022-12-14T22:48:32.291

3.3 Training the Model

- Elastic Net Model

```
from sklearn.linear_model import ElasticNet

Elnet=ElasticNet(alpha=0.01, l1_ratio=0.5, max_iter=10000)
Elnet.fit(x_train, y_train)

pred_ElasticNet = Elnet.predict(x_test)
Elnet.score(x_test,y_test)
```

```
0.4626343948617977
```

3.4 Compute Error and RMSE - Lasso

Calculating the error of the trained model for each task in TestGroup and calculating RMSE

❶ Error table

	Task ID	Supplier ID	Cost	Pred_cost	Min_cost	Error
0	2020-01-16	S1	0.411120	0.390210	0.309074	-0.102046
63	2020-03-03	S33	0.404604	0.384666	0.350738	-0.053866
126	2020-10-30	S56	0.411498	0.422696	0.388173	-0.023325
189	2021-03-22	S19	0.363016	0.404554	0.321541	-0.041474
252	2021-03-29	S3	0.385688	0.403449	0.300676	-0.085011
315	2021-04-28	S25	0.373955	0.400656	0.313595	-0.060360
378	2021-05-06	S42	0.361783	0.398833	0.288931	-0.072852
441	2021-05-20	S26	0.359672	0.407989	0.333052	-0.026620
504	2021-05-31	S29	0.417547	0.420327	0.358073	-0.059474
567	2021-09-14	S32	0.429985	0.416433	0.365667	-0.064318
630	2021-09-21	S5	0.442972	0.420972	0.419387	-0.023585
693	2021-09-28	S22	0.435937	0.424019	0.396764	-0.039173
756	2021-10-01	S42	0.397064	0.423554	0.397064	0.000000
819	2021-10-15	S49	0.449514	0.429451	0.416431	-0.033083
882	2021-10-22	S41	0.449481	0.429316	0.421872	-0.027609
945	2021-10-27	S58	0.448928	0.432607	0.419289	-0.029639
1008	2021-11-12	S54	0.416210	0.424287	0.393463	-0.022747
1071	2021-12-01	S28	0.404496	0.423056	0.375877	-0.028619
1134	2021-12-14	S61	0.425816	0.422299	0.374692	-0.051124
1197	2021-12-22	S23	0.407867	0.418099	0.377868	-0.029999

❷ Calculation of RMSE

```
Test_data4['Error_Squared']=Test_data4['Error']**2
mse=np.sum(Test_data4['Error_Squared'])/len(Test_data4['Task ID'].unique())
```

```
rmse3_lasso=np.sqrt(mse)
print(rmse3_lasso)
```

```
0.049907607786402734
```

3.4 Compute Error and RMSE - Ridge

Calculating the error of the trained model for each task in TestGroup and calculating RMSE

❶ Error table

	Task ID	Supplier ID	Cost	Pred_cost	Min_cost	Error
0	2020-01-16	S39	0.329161	0.353917	0.309074	-0.020087
63	2020-03-03	S39	0.402641	0.360758	0.350738	-0.051904
126	2020-10-30	S39	0.424665	0.426658	0.388173	-0.036492
189	2021-03-22	S39	0.346083	0.324950	0.321541	-0.024541
252	2021-03-29	S39	0.356513	0.348769	0.300676	-0.055837
315	2021-04-28	S39	0.359524	0.336150	0.313595	-0.045929
378	2021-05-06	S39	0.359673	0.307208	0.288931	-0.070743
441	2021-05-20	S39	0.385325	0.339556	0.333052	-0.052273
504	2021-05-31	S39	0.380944	0.400815	0.358073	-0.022871
567	2021-09-14	S39	0.396566	0.436659	0.365667	-0.030899
630	2021-09-21	S39	0.449464	0.428158	0.419387	-0.030077
693	2021-09-28	S39	0.445416	0.413661	0.396764	-0.048652
756	2021-10-01	S39	0.432532	0.433935	0.397064	-0.035467
819	2021-10-15	S39	0.447382	0.458749	0.416431	-0.030951
882	2021-10-22	S39	0.486907	0.449733	0.421872	-0.065035
945	2021-10-27	S39	0.445483	0.445327	0.419289	-0.026194
1008	2021-11-12	S39	0.410363	0.381384	0.393463	-0.016900
1071	2021-12-01	S39	0.403838	0.382575	0.375877	-0.027960
1134	2021-12-14	S39	0.407399	0.429629	0.374692	-0.032706
1197	2021-12-22	S39	0.415940	0.407969	0.377868	-0.038072

❷ Calculation of RMSE

```
Test_dataRidge4['Error_Squared']=Test_dataRidge4['Error']**2
mse=np.sum(Test_dataRidge4['Error_Squared'])/len(Test_dataRidge4['Task ID'].unique())

rmse3_ridge=np.sqrt(mse)
print(rmse3_ridge)
```

```
0.0409043084265918
```

3.4 Compute Error and RMSE - Elastic Net

Calculating the error of the trained model for each task in TestGroup and calculating RMSE

❶ Error table

	Task ID	Supplier ID	Cost	Pred_cost	Min_cost	Error
0	2020-01-16	S48	0.377193	0.368343	0.309074	-0.068119
63	2020-03-03	S33	0.404604	0.357429	0.350738	-0.053866
126	2020-10-30	S56	0.411498	0.430300	0.388173	-0.023325
189	2021-03-22	S19	0.363016	0.386804	0.321541	-0.041474
252	2021-03-29	S49	0.349185	0.384887	0.300676	-0.048509
315	2021-04-28	S29	0.330813	0.379616	0.313595	-0.017219
378	2021-05-06	S7	0.302261	0.376430	0.288931	-0.013330
441	2021-05-20	S27	0.349233	0.395271	0.333052	-0.016181
504	2021-05-31	S29	0.417547	0.422681	0.358073	-0.059474
567	2021-09-14	S56	0.365667	0.414778	0.365667	0.000000
630	2021-09-21	S48	0.445781	0.424596	0.419387	-0.026393
693	2021-09-28	S9	0.453824	0.429366	0.396764	-0.057060
756	2021-10-01	S13	0.418590	0.428420	0.397064	-0.021525
819	2021-10-15	S49	0.449514	0.439807	0.416431	-0.033083
882	2021-10-22	S29	0.440674	0.438656	0.421872	-0.018802
945	2021-10-27	S50	0.470873	0.447866	0.419289	-0.051584
1008	2021-11-12	S54	0.416210	0.429763	0.393463	-0.022747
1071	2021-12-01	S40	0.398837	0.421140	0.375877	-0.022960
1134	2021-12-14	S6	0.443407	0.418743	0.374692	-0.068714
1197	2021-12-22	S23	0.407867	0.410191	0.377868	-0.029999

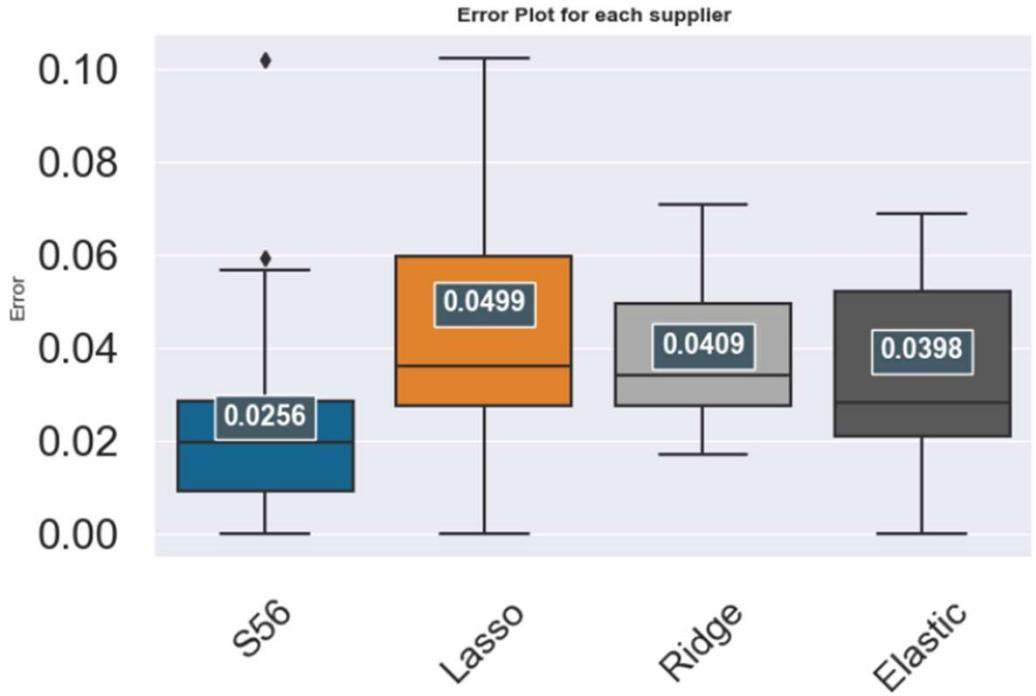
❷ Calculation of RMSE

```
Test_dataElasticNet4['Error_Squared']=Test_dataElasticNet4['Error']**2
mse=np.sum(Test_dataElasticNet4['Error_Squared'])/len(Test_dataElasticNet4['Task ID'].unique())

rmse3_ElasticNet=np.sqrt(mse)
print(rmse3_ElasticNet)
```

```
0.039763753646178596
```

AP0

3.5 Comparing Errors and RMSE



Slide 31

AP0 comparison table to be added

Afifa Pirzada, 2022-12-12T10:30:25.847

4. Cross-Validation

4.1 Leave-One-Group-Out Cross Validation(Lasso)
& Creating BeePy Scorer

4.2 Computing RMSE scores

4.3 Cross-Validation – Ridge Regression Model

4.4 Cross-Validation – Ridge Regression Model

4.1 Cross-Validation - Lasso

- Using Leave-One-Out Cross Validation
- Creating BeePy Scorer using make_scorer

```
from sklearn.model_selection import LeaveOneGroupOut, cross_val_score
from sklearn.metrics import make_scorer
import os

logo=LeaveOneGroupOut()
train_Groups=trains['Task ID']
logo.get_n_splits(x,y,train_Groups)
logo.get_n_splits(groups=train_Groups)
```

```
def error_2(y_true,y_pred):
    selected=y_true.iloc[np.argmin(y_pred)]
    #We select the real cost of the supplier recommended by the model (the cheapest one)
    minimal=np.min(y_true)
    # We select the minimal value for each task
    error=minimal-selected
    #Finally we applied equation1 (the minimal cost - the cost if we select what the model suggest)
    return(error)
```

$$\text{error\_2}(y_{true}, y_{pred}) = Cost(Y_{true})_{min} - Cost(y_{pred})_{Selected}$$

```
BeePy_scorer = make_scorer(error_2)
scores_lasso = cross_val_score(lasso, x_train, y_train, scoring = BeePy_scorer, \
                               cv=logo, groups=train_Groups, n_jobs=-1)
#Or we can just set as -1 : -1 is for using all the CPU cores available.
```

4.2 Computing RMSE -Lasso

RMSE of the scores returned

```
#rmse calculation
error=np.power(scores_lasso,2)
mse=np.sum(error)/len(scores_lasso)
rmse4_lasso=np.sqrt(mse)
rmse4_lasso
```

```
0.05787512394794050 4
```

```
rmse_lasso['Cross Val']= rmse4_lasso
rmse_lasso
```

	Manual approach (2.2)	Initial ML	Cross Val
0	0.025594	0.049908	0.057875

*T= *Number of Task*

$$\sqrt{\frac{\sum_{t=1}^{T} error\_2(t)^2}{T}}$$

4.3 Cross-Validation - Ridge

```
quadraticerror=np.sum(np.power(scores_ridge,2))
quadraticerror_2=quadraticerror/len(scores_ridge)
rmse4_ridge=np.sqrt(quadraticerror_2)
rmse4_ridge
```

```
0.03996679999798722
```

```
rmse_ridge['Cross Val']=rmse4_ridge
rmse_ridge
```

	Manual approach (2.2)	Initial ML	Cross Val
0	0.025594	0.040904	0.039967

4.3 Cross-Validation - ElasticNet

```
quadraticerror=np.sum(np.power(scores_elastic,2))
quadraticerror_2=quadraticerror/len(scores_elastic)
rmse4_elastic=np.sqrt(quadraticerror_2)
rmse4_elastic
```

```
0.04225220602235697
```

```
rmse_elastic['Cross Val']=rmse4_elastic
rmse_elastic
```

	Manual approach (2.2)	Initial ML	Cross Val
0	0.025594	0.039764	0.042252

5. Hyper-Parameter Optimization

5.1 GridSearch - Lasso Regression Model

5.2 Getting the best alpha value

5.3 Training the Model with the best hyper parameters -Lasso

5.4 Compute RMSE scores

5.5 GridSearch - Ridge Regression Model

5.6 GridSearch - ElasticNet Regression Model

5.5 RMSE Comparison of Three Different Models

DJ0

5.1 Applying Grid Search - Lasso

Using GridSearch to find the best hyper parameters for our model

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

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

# alphas = np.linspace (0, 1, 100)
# param_grid={"alpha": alphas}
# alphas = np.logspace(-4, -0.5, 30)

# bottom line limit 0.001, when alpha becomes 0 -> model closer to linear regression ->thus,choosing the smallest alpha (1e-08)

grid_search = GridSearchCV(lasso, param_grid, scoring=BeePy_scorer, cv=logo, n_jobs = -1)
grid_search.fit(x_train, y_train, groups=train_Groups)
```

```
GridSearchCV(cv=LeaveOneGroupOut(),
             estimator=Lasso(alpha=0.01, max_iter=10000, random_state=42),
             n_jobs=-1,
             param_grid={'alpha': array([0.1       , 0.09899091, 0.09798182, 0.09697273, 0.09596364,
       0.09495455, 0.09394545, 0.09293636, 0.09192727, 0.09091818,
       0.08990909, 0.0889    , 0.08789091, 0.08688182, 0.08587273,
       0.08486364, 0.08385455, 0.08284545, 0.08183636, 0.08082727,
       0.07981818...
       0.02936364, 0.02835455, 0.02734545, 0.02633636, 0.02532727,
       0.02431818, 0.02330909, 0.0223    , 0.02129091, 0.02028182,
       0.01927273, 0.01826364, 0.01725455, 0.01624545, 0.01523636,
       0.01422727, 0.01321818, 0.01220909, 0.0112    , 0.01019091,
       0.00918182, 0.00817273, 0.00716364, 0.00615455, 0.00514545,
       0.00413636, 0.00312727, 0.00211818, 0.00110909, 0.0001    ])},
             scoring=make_scorer(error_2))
```

Slide 38

DJ0 Up to here: Dahye
Dahye Jeong, 2022-12-14T16:11:34.807

DJ0

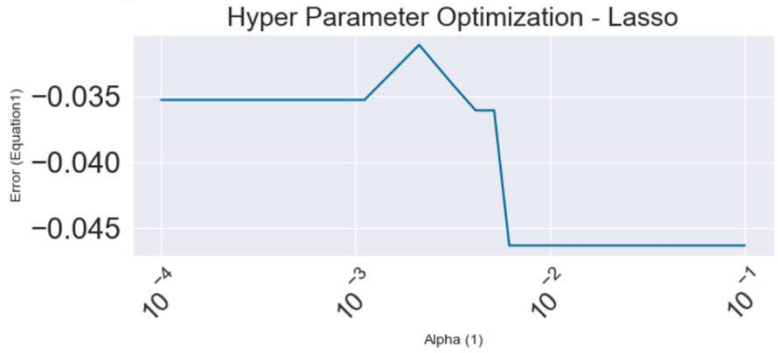
5.2 Getting the best alpha value - Lasso

Getting the best parameters from the values derived from grid search

```
print(grid_search.best_params_)
print(grid_search.best_score_)
grid_search.score(x_train, y_train)
gsTable = pd.DataFrame(grid_search.cv_results_)
gsTable[['params','mean_test_score','rank_test_score']]
```

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

	params	mean_test_score	rank_test_score
0	{'alpha': 0.1}	-0.046304	7
1	{'alpha': 0.0989090909090091}	-0.046304	7
2	{'alpha': 0.0979818181818818}	-0.046304	7
3	{'alpha': 0.0969727272727728}	-0.046304	7
4	{'alpha': 0.0959636363636636}	-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	
99	{'alpha': 0.0001}	-0.035272	



Slide 39

DJ0 [@Devesh Prasad] Presentation preparation from here to the end
Dahye Jeong, 2022-12-14T22:43:21.075

```
alpha_param=grid_search.best_params_['alpha']

lassoHyper = Lasso(alpha = alpha_param, max_iter=10000, random_state=42)
lassoHyper.fit(x_train, y_train)

pred_lassoHyper = lassoHyper.predict(x_test)
pred_lassoHyper

lassoHyper.score(x_test,y_test)
```

```
0.5580893348396483
```

5.3 Training the Model with the best hyper parameters - Lasso

Training the model with the best hyper parameters we got from grid search and scoring it

5.4 Compute RMSE scores - Lasso

Getting RMSE from the predicted values from hyper parameters and comparing it with previous sections

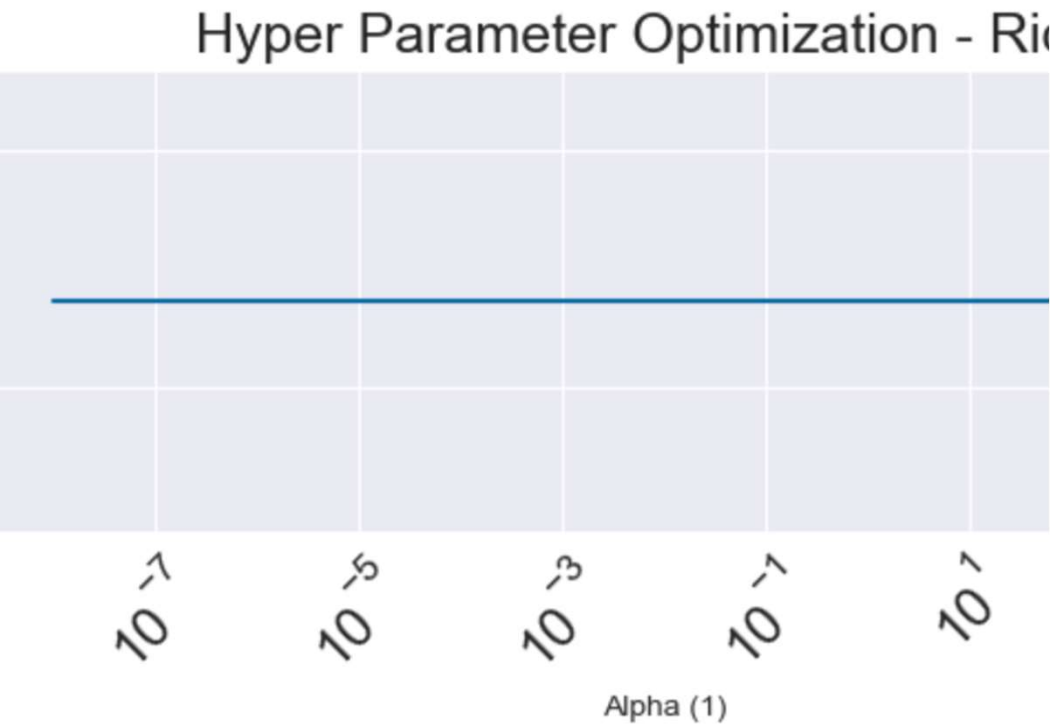
```
scores_lasso = cross_val_score(lassoHyper, x_train, y_train, scoring = BeePy_scorer, \
                               cv=logo, groups=train_Groups, n_jobs=-1)
```

```
error=np.power(scores_lasso,2)
mse=np.sum(error)/len(scores_lasso)
rmse5_lasso=np.sqrt(mse)
rmse4_lasso
```

```
0.057875123947940504
```

```
rmse_lasso['Hyper-parameter']=rmse5_lasso
rmse_lasso
```

	Manual approach (2.2)	Initial ML	Cross Val	Hyper-parameter
0	0.025594	0.049908	0.057875	0.036048




```
# 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}

grid_search2 = GridSearchCV(Ridge(), param_grid, scoring=BeePy_scorer, cv=logo, n_jobs = -1)
grid_search2.fit(x_train, y_train, groups=train_Groups)
```

```
GridSearchCV(cv=LeaveOneGroupOut(), estimator=Ridge(), n_jobs=-1,
             param_grid={'alpha': array([1.00000000e-08, 1.00100100e+01, 2.00200200e+01, 3.00300300e+01,
       4.00400400e+01, 5.00500501e+01, 6.00600601e+01, 7.00700701e+01,
       8.00800801e+01, 9.00900901e+01, 1.00100100e+02, 1.10110110e+02,
       1.20120120e+02, 1.30130130e+02, 1.40140140e+02, 1.50150150e+02,
       1.60160160e+02, 1.701...
       9.76976977e+03, 9.77977978e+03, 9.78978979e+03, 9.79979980e+03,
       9.80980981e+03, 9.81981982e+03, 9.82982983e+03, 9.83983984e+03,
       9.84984985e+03, 9.85985986e+03, 9.86986987e+03, 9.87987988e+03,
       9.88988989e+03, 9.89989990e+03, 9.90990991e+03, 9.91991992e+03,
       9.92992993e+03, 9.93993994e+03, 9.94994995e+03, 9.95995996e+03,
       9.96996997e+03, 9.97997998e+03, 9.98998999e+03, 1.00000000e+04])},
             scoring=make_scorer(error_2))
```

```
print(grid_search2.best_params_)
print(grid_search2.best_score_)
grid_search2.score(x_train, y_train)
gsTable2 = pd.DataFrame(grid_search2.cv_results_)
gsTable2[['params','mean_test_score','rank_test_score']]
```

```
{'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

5.5 GridSearch – Ridge Regression Model

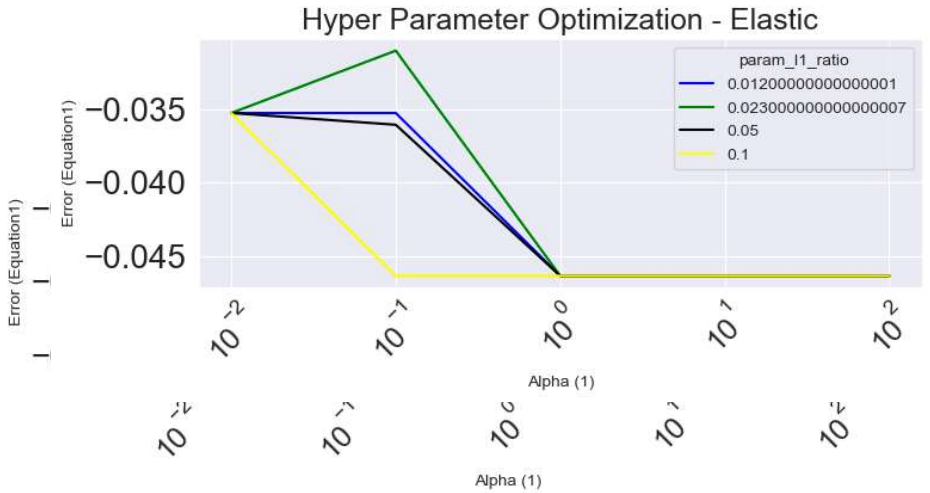
```
param_grid = {"alpha":[1e-5,1e-4,1e-3,1e-2, 1, 10], "l1_ratio":[1e-5,1e-4, 1e-3,1e-2, 1]} #First range
aram_grid = {"alpha":[0.01, 0.1, 1.0, 10, 100], "l1_ratio":np.linspace (0.1,0.001, 100)} #Second range

s3 = GridSearchCV(Elnet, param_grid, scoring=BeePy_scorer, cv=logo, n_jobs = -1)
s3.fit(x_train, y_train, groups=train_Groups)

s3 = GridSearchCV(Elnet, param_grid, scoring=BeePy_scorer, cv=logo, n_jobs = -1)
s3.fit(x_train, y_train, groups=train_Groups)
```

```
ridSearchCV(cv=LeaveOneGroupOut(),
            estimator=ElasticNet(alpha=0.01, max_iter=10000), n_jobs=-1,
            param_grid={'alpha': [0.01, 0.1, 1.0, 10, 100],
                        'l1_ratio': array([0.1  , 0.099, 0.098, 0.097, 0.096, 0.095, 0.094, 0.093, 0.092,
       0.091, 0.09 , 0.089, 0.088, 0.087, 0.086, 0.085, 0.084, 0.083,
       0.082, 0.081, 0.08 , 0.079, 0.078, 0.077, 0.076, 0.075, 0.074,
       0.073, 0.072, 0.071, 0.07 , 0.069, 0.068, 0.06...
       0.055, 0.054, 0.053, 0.052, 0.051, 0.05 , 0.049, 0.048, 0.047,
       0.046, 0.045, 0.044, 0.043, 0.042, 0.041, 0.04 , 0.039, 0.038,
       0.037, 0.036, 0.035, 0.034, 0.033, 0.032, 0.031, 0.03 , 0.029,
       0.028, 0.027, 0.026, 0.025, 0.024, 0.023, 0.022, 0.021, 0.02 ,
       0.019, 0.018, 0.017, 0.016, 0.015, 0.014, 0.013, 0.012, 0.011,
       0.01 , 0.009, 0.008, 0.007, 0.006, 0.005, 0.004, 0.003, 0.002,
       0.001])},
            scoring=make_scorer(error_2))
```

```
rint(gs3.best_params_)
rint(gs3.best_score_)
s3.score(x_train, y_train)
sTable3 = pd.DataFrame(gs3.cv_results_)
sTable3[['params','mean_test_score','rank_test_score']]
```

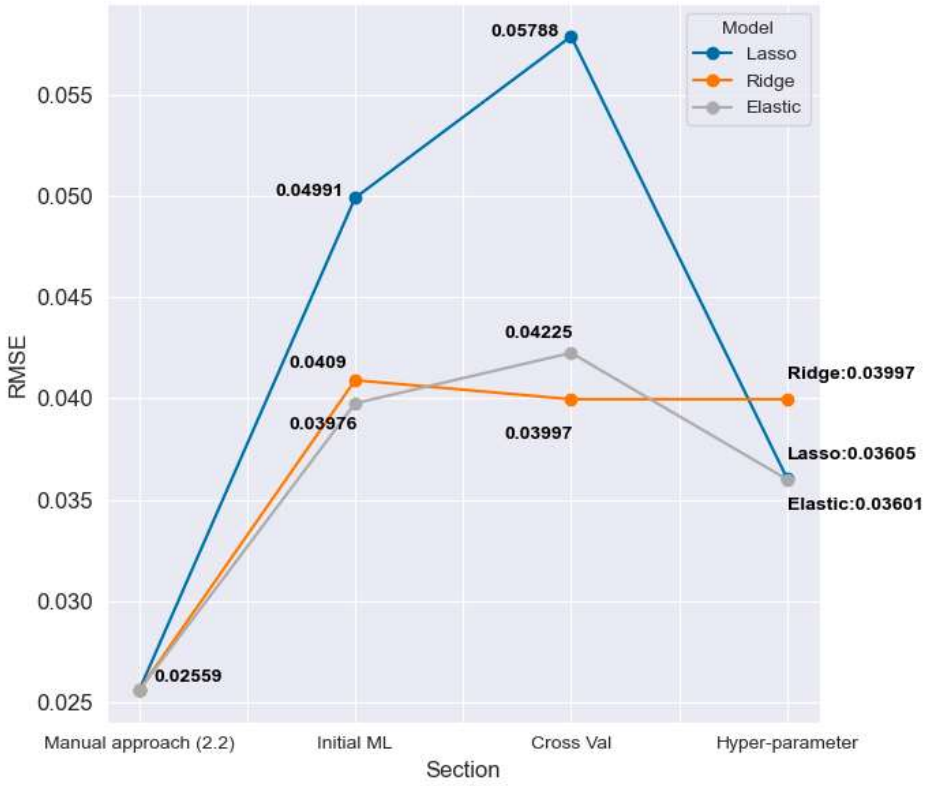
```
'alpha': 0.1, 'l1_ratio': 0.023000000000000007}
0.031047226130000005
```

	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



5.6 GridSearch – ElasticNet Regression Model

DJ0



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

5.7 RMSE Comparison of Three different models

Slide 44

DJ0 Up to here: Devesh
Dahye Jeong, 2022-12-14T16:11:01.945

Thank You!

BeePy

10351500
11189282
11063737
11164061
11186818
11028738
11054378