ENM319 Production and Operations Planning

&

BIM213 Data Structures and Algorithms

Term Project

Final Report

Student Name- Student ID	Department of Student
Bahar Gürsoy	Industrial Engineering
Pelin Kanar	Industrial Engineering
Sinem Türkçü	Computer Engineering

İçindekiler

1. Assignment

Work Packages For Industrial Engineering Students :

Task (Calculating)	Responsible Person	Due Date
Exponential Smoothing	Pelin Kanar, Bahar Gürsoy	28.11.2021
Double Exponential Smoothing	Pelin Kanar, Bahar Gürsoy	28.11.2021
Regression analysis	Pelin Kanar, Bahar Gürsoy	30.11.2021
Deseasonalized regression analysis	Pelin Kanar, Bahar Gürsoy	03.11.2021
Calculating MSE and determining the best method	Pelin Kanar, Bahar Gürsoy	03.11.2021
Explanation about the best method	Pelin Kanar, Bahar Gürsoy	04.11.2021

Work Packages For Computer Engineering Students :

Task	Responsible Person	Due Date
Interface designing	Sinem Türkçü	10.12.2021
Integration of the interface	Kawa Alismail	17.12.2021
Data structures manipulation	Ebru Şara Bağca	25.12.2021

2. Datasets

Dataset 1:

Month	Jan.	Feb	Mar.	Apr.	May.	Jun.	Jul	Aug.	Sep.	Oct.	Nov.	Dec.
Demand Year 1	300	350	330	340	390	430	480	460	490	510	550	560
Demand Year 2	550	590	600	610	630	620	680	690	710	730	740	770

Dataset 2:

Month	Jan.	Feb	Mar.	Apr.	May.	Jun.	Jul	Aug.	Sep.	Oct.	Nov.	Dec.
Demand Year 1	200	300	250	600	650	670	400	440	430	900	980	990
Demand Year 2	300	370	380	710	730	790	450	480	490	930	960	980

2. Exponential Smoothing

3.1 Solution on Excel

3.1.1 For Dataset 1;

Month	Demand Year 1	Demand Year 2	Forecast Year 1	Forecast Year 2	Error Year 1	Error Year 2	Error^2 Year 1	Error^2 Year 2	MSE
Jan.	300	550	300,00	477,05	0,00	-72,95	0,00	5.321,48	7.507,52
Feb	350	590	300,00	491,64	-50,00	-98,36	2.500,00	9.674,45	
Mar.	330	600	310,00	511,31	-20,00	-88,69	400,00	7.865,39	
Apr.	340	610	314,00	529,05	-26,00	-80,95	676,00	6.552,84	
May.	390	630	319,20	545,24	-70,80	-84,76	5.012,64	7.184,21	
Jun.	430	620	333,36	562,19	-96,64	-57,81	9.339,29	3.341,74	
Jul	480	680	352,69	573,75	-127,31	-106,25	16.208,35	11.288,26	
Aug.	460	690	378,15	595,00	-81,85	-95,00	6.699,36	9.024,42	
Sep.	490	710	394,52	614,00	-95,48	-96,00	9.116,37	9.215,53	
Oct.	510	730	413,62	633,20	-96,38	-96,80	9.289,83	9.369,86	
Nov.	550	740	432,89	652,56	-117,11	-87,44	13.714,05	7.645,48	
Dec.	560	770	456,31	670,05	-103,69	-99,95	10.750,70	9.990,15	

3.1.2 For Dataset 2;

Month	Demand Year 1	Demand Year 2	Forecast Year 1	Forecast Year 2	Error Year 1	Error Year 2	Error^2 Year 1	Error^2 Year 2	MSE
Jan.	200	300	200,00	689,15	0,00	389,15	0,00	151.435,65	69.188,27
Feb	300	370	200,00	611,32	-100,00	241,32	10.000,00	58.234,31	
Mar.	250	380	220,00	563,05	-30,00	183,05	900,00	33.508,88	
Apr.	600	710	226,00	526,44	-374,00	-183,56	139.876,00	33.693,01	
May.	650	730	300,80	563,15	-349,20	-166,85	121.940,64	27.837,34	
Jun.	670	790	370,64	596,52	-299,36	-193,48	89.616,41	37.433,04	
Jul	400	450	430,51	635,22	30,51	185,22	930,98	34.306,09	
Aug.	440	480	424,41	598,18	-15,59	118,18	243,06	13.965,39	
Sep.	430	490	427,53	574,54	-2,47	84,54	6,11	7.147,04	
Oct.	900	930	428,02	557,63	-471,98	-372,37	222.763,10	138.657,82	
Nov.	980	960	522,42	632,11	-457,58	-327,89	209.381,55	107.514,66	
Dec.	990	980	613,93	697,68	-376,07	-282,32	141.425,51	79.702,00	

3.2 Solution on Java

For dataset 1:

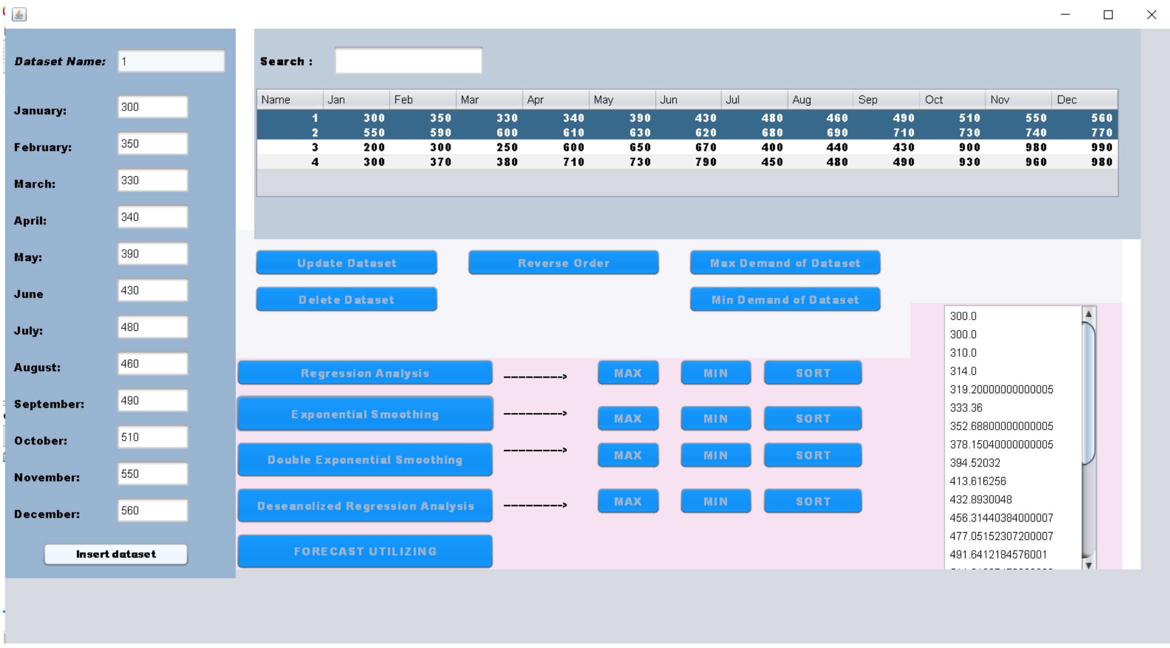


Figure 1

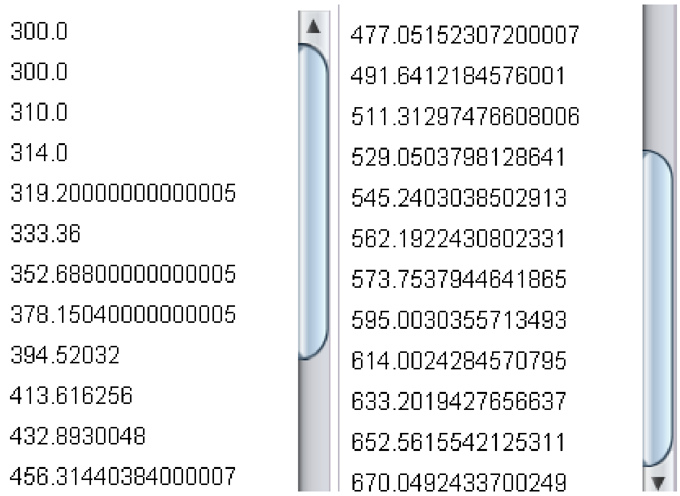


Figure 2

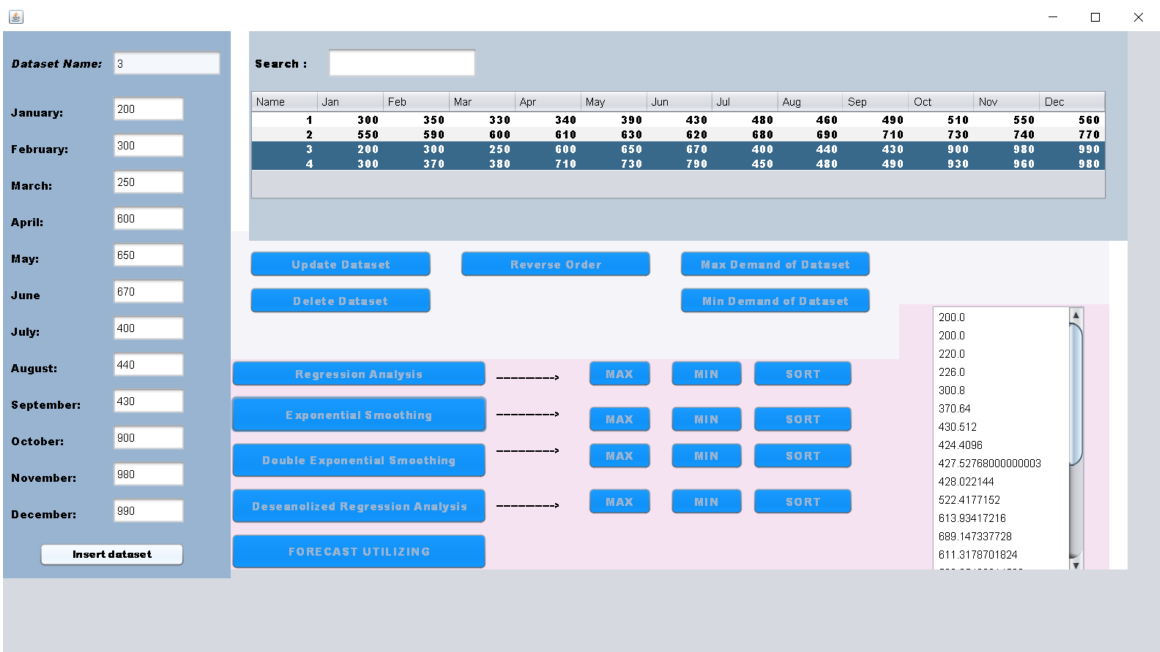


Figure 3

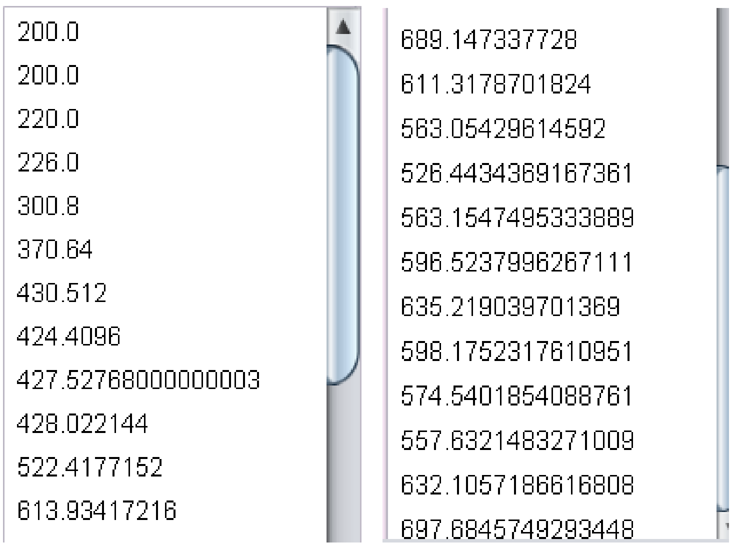


Figure 4

Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For exponential smoothing , MSE:7507.5164024277665

Figure 5

Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For exponential smoothing max forecasted demand is: 670.0492433700249

Figure 6

Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For exponential smoothing min forecasted demand is: 300.0

Figure 7

670.0492433700249
652.5615542125311
633.2019427656637
614.0024284570795
595.0030355713493
573.7537944641865
562.1922430802331
545.2403038502913
529.0503798128641
511.31297476608006
491.6412184576001
477.05152307200007
456.31440384000007
432.8930048

413.616256
394.52032
378.15040000000005
352.68800000000005
333.36
319.20000000000005
314.0
310.0
300.0
300.0

Figure 8

- Figure 1 and Figure 2 shows Dataset1 Forecast year 1 and forecast year 2 values.Figure 3 and Figure 4 shows Dataset2 Forecast year1 and forecast year 2 values. To obtain this data, I first pulled the data from the selected rows from the table and added this data to an arraylist that I created myself.Then I applied the exponential smoothing formula and added the forecasted years to a new arraylist using this method. Then I printed the arraylist containing the estimated years into a Jtable.

For this methods code:

```
Arraylist list = new Arraylist();

list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 1).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 2).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 3).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 4).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 5).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 6).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 7).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 8).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 9).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 10).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 11).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[0], 12).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 1).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 2).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 3).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 4).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 5).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 6).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 7).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 8).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 9).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 10).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 11).toString()));
list.add(Double.parseDouble(model.getValueAt(selectedRow[1], 12).toString()));
```

Figure 9

In Figure 9 , code snipped, pulled the data from the selected rows from the table.

```
Arraylist tahmin1 = new Arraylist();
tahmin1.add(list.get(0));
tahmin1.add((Double) list.get(0) * a + (Double) tahmin1.get(0) * (1 - a));
tahmin1.add((Double) list.get(1) * a + (Double) tahmin1.get(1) * (1 - a));
tahmin1.add((Double) list.get(2) * a + (Double) tahmin1.get(2) * (1 - a));
tahmin1.add((Double) list.get(3) * a + (Double) tahmin1.get(3) * (1 - a));
tahmin1.add((Double) list.get(4) * a + (Double) tahmin1.get(4) * (1 - a));
tahmin1.add((Double) list.get(5) * a + (Double) tahmin1.get(5) * (1 - a));
tahmin1.add((Double) list.get(6) * a + (Double) tahmin1.get(6) * (1 - a));
tahmin1.add((Double) list.get(7) * a + (Double) tahmin1.get(7) * (1 - a));
tahmin1.add((Double) list.get(8) * a + (Double) tahmin1.get(8) * (1 - a));
tahmin1.add((Double) list.get(9) * a + (Double) tahmin1.get(9) * (1 - a));
tahmin1.add((Double) list.get(10) * a + (Double) tahmin1.get(10) * (1 - a));
tahmin1.add(a * (Double) list.get(11) + (Double) tahmin1.get(11) * (1 - a));

tahmin1.add((Double) list.get(12) * a + (Double) tahmin1.get(12) * (1 - a));
tahmin1.add((Double) list.get(13) * a + (Double) tahmin1.get(13) * (1 - a));
tahmin1.add((Double) list.get(14) * a + (Double) tahmin1.get(14) * (1 - a));
tahmin1.add((Double) list.get(15) * a + (Double) tahmin1.get(15) * (1 - a));
tahmin1.add((Double) list.get(16) * a + (Double) tahmin1.get(16) * (1 - a));
tahmin1.add((Double) list.get(17) * a + (Double) tahmin1.get(17) * (1 - a));
tahmin1.add((Double) list.get(18) * a + (Double) tahmin1.get(18) * (1 - a));
tahmin1.add((Double) list.get(19) * a + (Double) tahmin1.get(19) * (1 - a));
tahmin1.add((Double) list.get(20) * a + (Double) tahmin1.get(20) * (1 - a));
tahmin1.add((Double) list.get(21) * a + (Double) tahmin1.get(21) * (1 - a));
tahmin1.add((Double) list.get(22) * a + (Double) tahmin1.get(22) * (1 - a));
tahmin1.add((Double) list.get(23) * a + (Double) tahmin1.get(23) * (1 - a));
```

Figure 10

- In Figure 10, code snipped, applied an exponential smoothing formula and added the forecasted years to a new arraylist.
- Figure 5 shows MSE values for exponential smoothing. To find the MSE value, I created a new arraylist after the codes
- I wrote in figure1, figure 2 (also figure3, figure 4), and entered the data about the errors of these values in the arraylist, and found it by doing the necessary operations in Figure11.

```
error1.add(0);
error1.add(((Double) tahmin1.get(1) - (Double) list.get(1)) * ((Double) tahmin1.get(1) - (Double) list.get(1)));
error1.add(((Double) tahmin1.get(2) - (Double) list.get(2)) * ((Double) tahmin1.get(2) - (Double) list.get(2)));
error1.add(((Double) tahmin1.get(3) - (Double) list.get(3)) * ((Double) tahmin1.get(3) - (Double) list.get(3)));
error1.add(((Double) tahmin1.get(4) - (Double) list.get(4)) * ((Double) tahmin1.get(4) - (Double) list.get(4)));
error1.add(((Double) tahmin1.get(5) - (Double) list.get(5)) * ((Double) tahmin1.get(5) - (Double) list.get(5)));
error1.add(((Double) tahmin1.get(6) - (Double) list.get(6)) * ((Double) tahmin1.get(6) - (Double) list.get(6)));
error1.add(((Double) tahmin1.get(7) - (Double) list.get(7)) * ((Double) tahmin1.get(7) - (Double) list.get(7)));
error1.add(((Double) tahmin1.get(8) - (Double) list.get(8)) * ((Double) tahmin1.get(8) - (Double) list.get(8)));
error1.add(((Double) tahmin1.get(9) - (Double) list.get(9)) * ((Double) tahmin1.get(9) - (Double) list.get(9)));
error1.add(((Double) tahmin1.get(10) - (Double) list.get(10)) * ((Double) tahmin1.get(10) - (Double) list.get(10)));
error1.add(((Double) tahmin1.get(11) - (Double) list.get(11)) * ((Double) tahmin1.get(11) - (Double) list.get(11)));
error1.add(((Double) tahmin1.get(12) - (Double) list.get(12)) * ((Double) tahmin1.get(12) - (Double) list.get(12)));
error1.add(((Double) tahmin1.get(13) - (Double) list.get(13)) * ((Double) tahmin1.get(13) - (Double) list.get(13)));
error1.add(((Double) tahmin1.get(14) - (Double) list.get(14)) * ((Double) tahmin1.get(14) - (Double) list.get(14)));
error1.add(((Double) tahmin1.get(15) - (Double) list.get(15)) * ((Double) tahmin1.get(15) - (Double) list.get(15)));
error1.add(((Double) tahmin1.get(16) - (Double) list.get(16)) * ((Double) tahmin1.get(16) - (Double) list.get(16)));
error1.add(((Double) tahmin1.get(17) - (Double) list.get(17)) * ((Double) tahmin1.get(17) - (Double) list.get(17)));
error1.add(((Double) tahmin1.get(18) - (Double) list.get(18)) * ((Double) tahmin1.get(18) - (Double) list.get(18)));
error1.add(((Double) tahmin1.get(19) - (Double) list.get(19)) * ((Double) tahmin1.get(19) - (Double) list.get(19)));
error1.add(((Double) tahmin1.get(20) - (Double) list.get(20)) * ((Double) tahmin1.get(20) - (Double) list.get(20)));
error1.add(((Double) tahmin1.get(21) - (Double) list.get(21)) * ((Double) tahmin1.get(21) - (Double) list.get(21)));
error1.add(((Double) tahmin1.get(22) - (Double) list.get(22)) * ((Double) tahmin1.get(22) - (Double) list.get(22)));
error1.add(((Double) tahmin1.get(23) - (Double) list.get(23)) * ((Double) tahmin1.get(23) - (Double) list.get(23)));
```

Figure 11

- Then I find the average value of this data. MSE was calculated at the end of these processes.
- Figure 6, Figure 7 and Figure 8 shows maximum and minimum forecasted value in the dataset and shows Sort forecasted sales in descending order respectively. At first, I listed these values. I used the insertion sort algorithm to sort these values. While using this algorithm, my data was arranged in order from smallest to largest. I called the last number from the sorted data to find the largest number, and the first number from these sorted data to find the smallest value.

```java
int j, P, Tmp;
for (P = 1; P < 12; P++) {
    double Tmp2 = (Double) max2[P];
    for (j = P; j > 0 && (Double) max2[j - 1] > Tmp2; j--) {
        max2[j] = max2[j - 1]; //Shift A[j-1] to right
    }
    max2[j] = Tmp2;
}
```

Figure 12

Figure 12 shows the insertion sort algorithm.

For dataset 2:

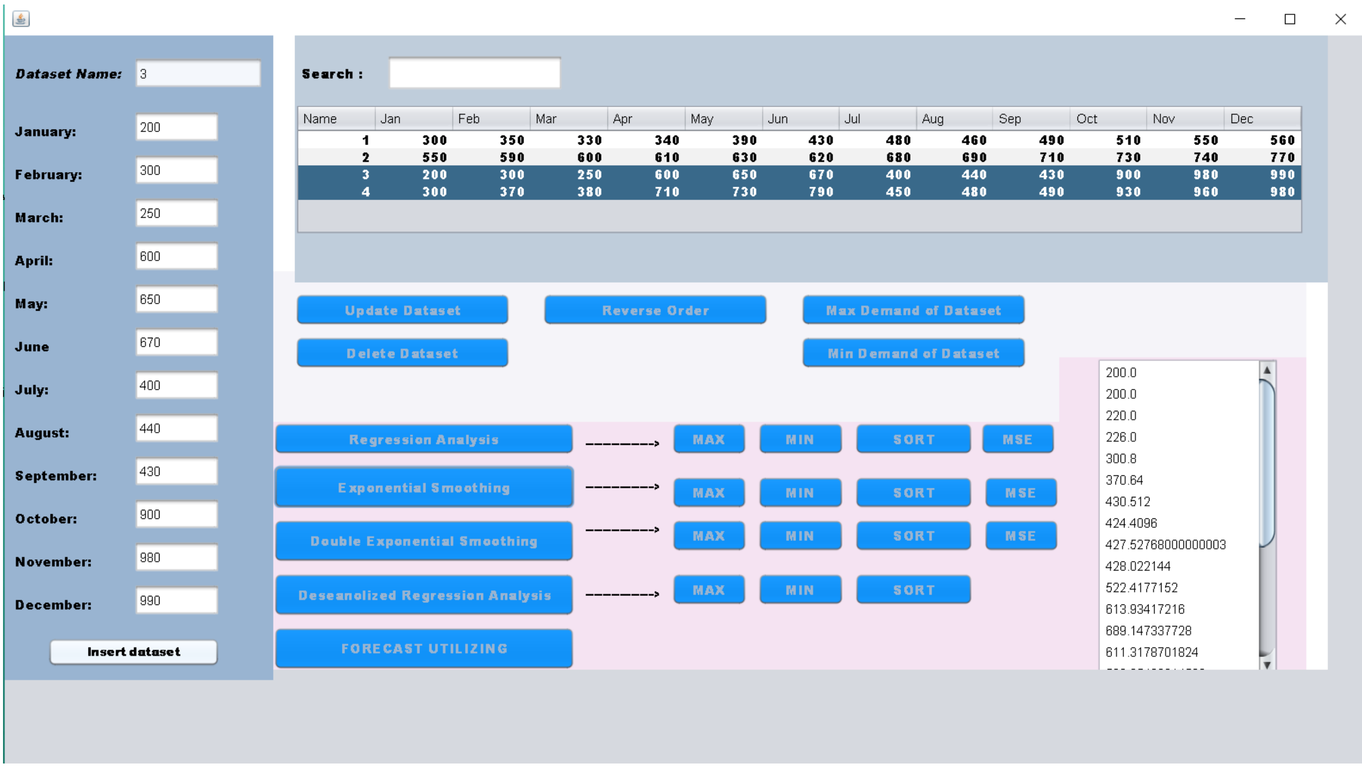


Figure 13

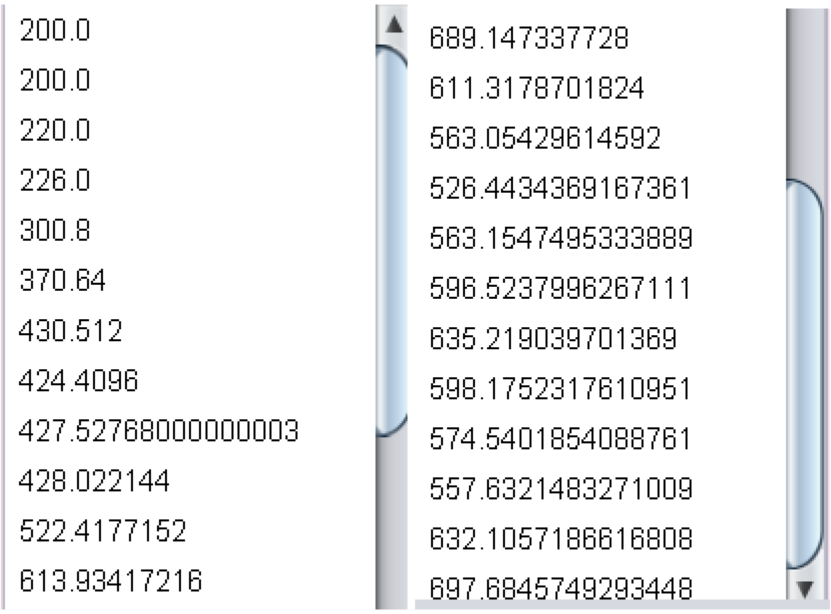
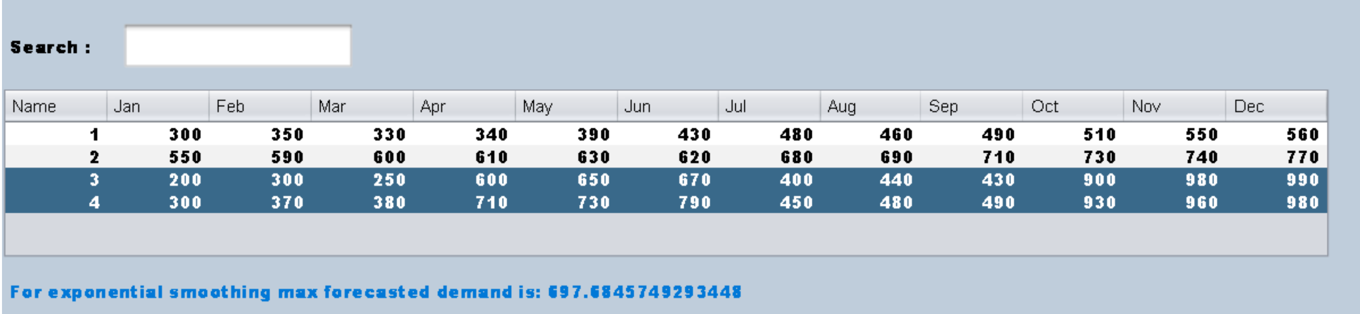


Figure 14

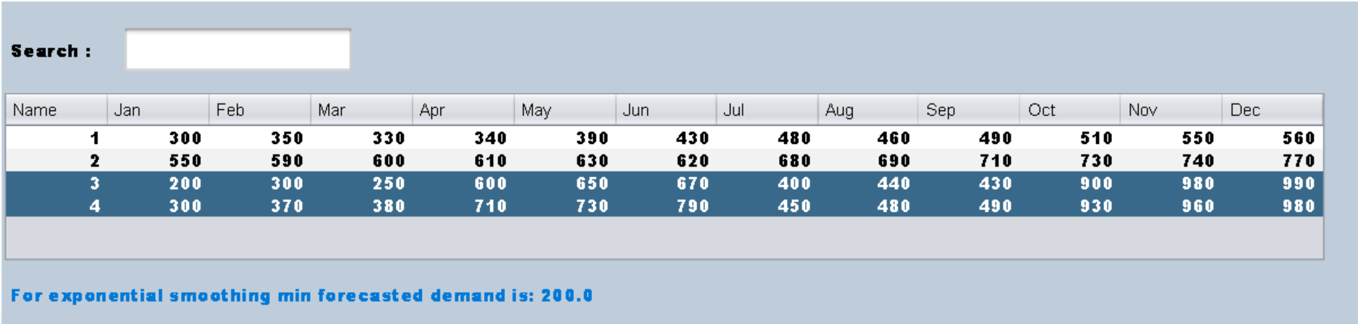


Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For exponential smoothing max forecasted demand is: 697.6845749293448

Figure 15

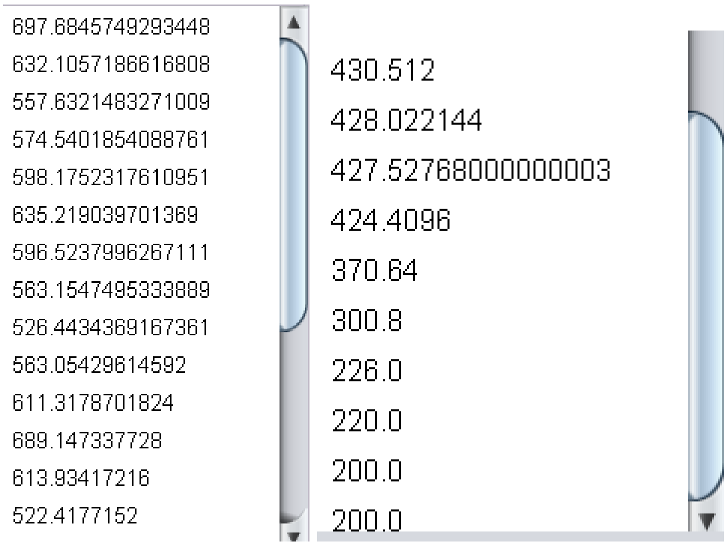


Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For exponential smoothing min forecasted demand is: 200.0

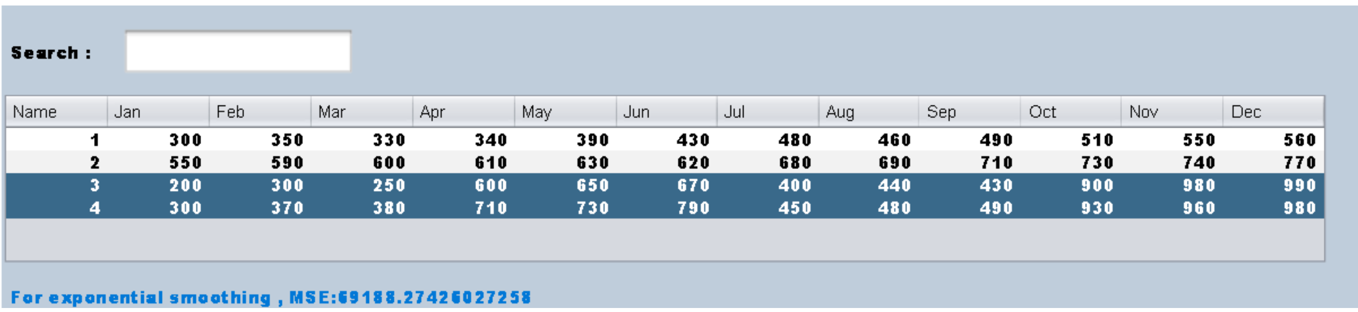
Figure 16



697.6845749293448
632.1057186616808
557.6321483271009
574.5401854088761
598.1752317610951
635.219039701369
596.5237996267111
563.1547495333889
526.4434369167361
563.05429614592
611.3178701824
689.147337728
613.93417216
522.4177152

430.512
428.022144
427.52768000000003
424.4096
370.64
300.8
226.0
220.0
200.0
200.0

Figure 17



Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For exponential smoothing , MSE:69188.27426027258

Figure 18

Figure 13 and Figure14 shows Dataset2 Forecast year1 and forecast year 2 values. To obtain this data, I first pulled the

data from the selected rows from the table and added this data to an arraylist that I created myself.Then I applied the exponential smoothing formula and added the forecasted years to a new arraylist using this method. Then I printed the

arraylist containing the estimated years into a Jtable.

- Figure 15, Figure 16 and Figure 17shows maximum and minimum forecasted value in the dataset and shows Sort forecasted sales in descending order respectively. At first, I listed these values. I used the insertion sort algorithm to sort these values. While using this algorithm, my data was arranged in order from smallest to largest. I called the last number from the sorted data to find the largest number, and the first number from these sorted data to find the smallest value.
- Figure 18 shows MSE values for dataset2.
- Dataset2 codes use the same code for Dataset 1.

3. Double-Exponential Smoothing

4.1 Solution on Excel

4.1.1 For Dataset 1;

Month	Demand Year 1	Demand Year 2	S	G	Forecast Year 1	S	G	Forecast Year 2	Error Year 1	Error Year 2	Error^2 Year 1	Error^2 Year 2	MSE

4.1.2 For Dataset 2;

Month	Demand Year 1	Demand Year 2	S	G	Forecast Year 1	S	G	Forecast Year 2	Error Year 1	Error Year 2	Error^2 Year 1	Error^2 Year 2	MSE
Jan.	200	300	240,00	48,00	250,00	813,99	36,69	942,49	50,00	642,49	2.500,00	412.790,97	68.287,99
Feb	300	370	290,40	48,48	288,00	754,54	17,46	850,68	-12,00	480,68	144,00	231.053,50	
Mar.	250	380	321,10	44,92	338,88	693,61	1,78	772,01	88,88	392,01	7.899,65	153.669,29	
Apr.	600	710	412,82	54,28	366,03	698,31	2,37	695,39	-233,97	-14,61	54.742,52	213,52	
May.	650	730	503,69	61,60	467,11	706,54	3,54	700,68	-182,89	-29,32	33.449,96	859,84	
Jun.	670	790	586,23	65,79	565,28	726,06	6,74	710,08	-104,72	-79,92	10.965,29	6.387,01	
Jul	400	450	601,61	55,71	652,02	676,24	-4,58	732,80	252,02	282,80	63.511,95	79.976,65	
Aug.	440	480	613,86	47,01	657,32	633,33	-12,24	671,67	217,32	191,67	47.227,97	36.735,68	
Sep.	430	490	614,70	37,78	660,87	594,87	-17,49	621,09	230,87	131,09	53.301,21	17.184,64	
Oct.	900	930	701,98	47,68	652,48	647,91	-3,38	577,39	-247,52	-352,61	61.268,04	124.336,41	
Nov.	980	960	795,73	56,89	749,66	707,62	9,24	644,53	-230,34	-315,47	53.055,76	99.522,73	
Dec.	990	980	880,10	62,39	852,62	769,49	19,76	716,86	-137,38	-263,14	18.872,29	69.242,76	

For dataset1:

Figure 19

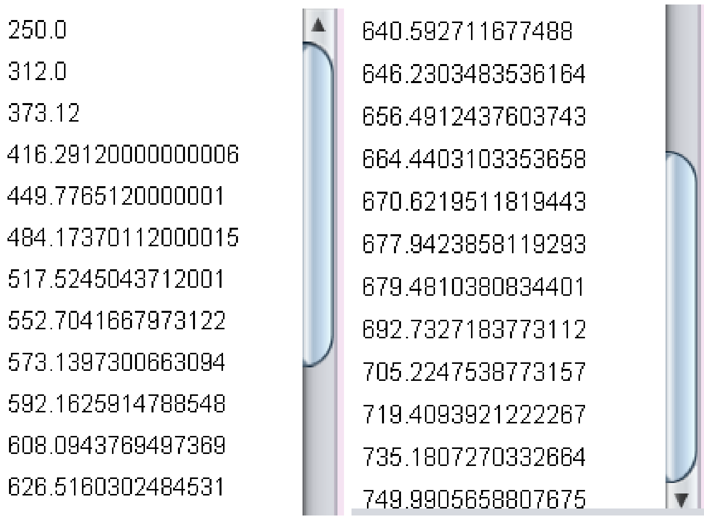
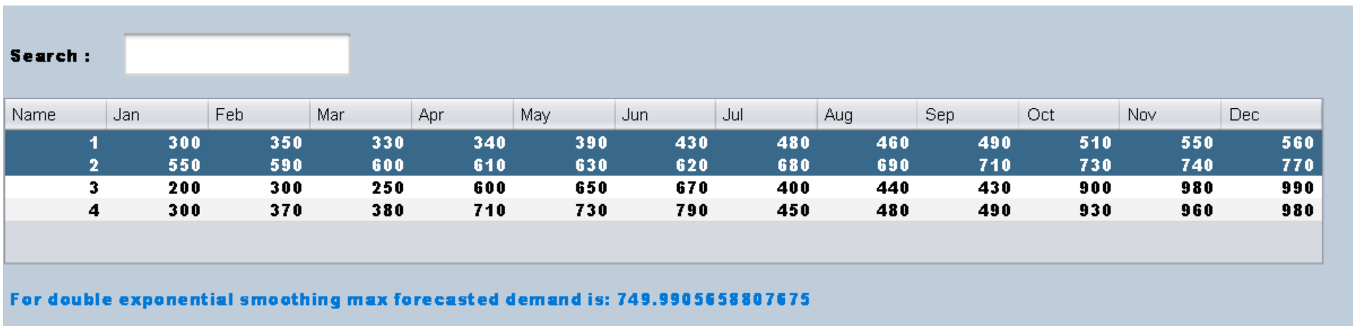


Figure 20



Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

Figure 21

Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For double exponential smoothing min forecasted demand is: 250.0

Figure 22

749.9905658807675
735.1807270332664
719.4093921222267
705.2247538773157
692.7327183773112
679.4810380834401
677.9423858119293
670.6219511819443
664.4403103353658
656.4912437603743
646.2303483536164
640.592711677488
626.5160302484531
608.0943769497369

592.1625914788548
573.1397300663094
552.7041667973122
517.5245043712001
484.17370112000015
449.7765120000001
416.29120000000006
373.12
312.0
250.0

Figure 23

Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For double exponential smoothing , MSE:3028.9068862324616

Figure 24

- Figure 19 and Figure 20 shows Dataset1 Forecast year 1 and forecast year 2 values. To obtain this data, I first pulled the data from the selected rows from the table and added this data to an arraylist that I created myself.Then I applied the double exponential analysis formula and added the forecasted years to a new arraylist using this method. Then I printed the arraylist containing the estimated years into a Jtable.
- For this method code is the same as in exponential smoothing code. In my code, pulled data from selected rows codes are identical to all methods.
- Figure 21, Figure 22and Figure 23 shows maximum and minimum forecasted value in the dataset and shows Sort forecasted sales in descending order respectively. At first, I listed these values. I used the insertion sort algorithm to sort these values. While using this algorithm, my data was arranged in order from smallest to largest. I called the last number from the sorted data to find the largest number, and the first number from these sorted data to find the smallest value.
- Figure 24 shows MSE values for the double exponential method. To find the MSE value, I created a new arraylist and entered the data about the errors of these values in the arraylist, and found it by doing the necessary operations.
- Then I find the average value of this data. MSE was calculated at the end of these processes.

```
s.add((Double) list2.get(0) * a + (1 - a) * (x + y));
g.add(a * ((Double) s.get(0) - x) + 0.8 * y);
s.add(a * (Double) list2.get(1) + (1 - a) * ((Double) s.get(0) + (Double) g.get(0)));
g.add(a * ((Double) s.get(1) - (Double) s.get(0)) + (1 - a) * (Double) g.get(0));
s.add(a * (Double) list2.get(2) + (1 - a) * ((Double) s.get(1) + (Double) g.get(1)));
g.add(a * ((Double) s.get(2) - (Double) s.get(1)) + (1 - a) * (Double) g.get(1));
s.add(a * (Double) list2.get(3) + (1 - a) * ((Double) s.get(2) + (Double) g.get(2)));
g.add(a * ((Double) s.get(3) - (Double) s.get(2)) + (1 - a) * (Double) g.get(2));
s.add(a * (Double) list2.get(4) + (1 - a) * ((Double) s.get(3) + (Double) g.get(3)));
g.add(a * ((Double) s.get(4) - (Double) s.get(3)) + (1 - a) * (Double) g.get(3));
s.add(a * (Double) list2.get(5) + (1 - a) * ((Double) s.get(4) + (Double) g.get(4)));
g.add(a * ((Double) s.get(5) - (Double) s.get(4)) + (1 - a) * (Double) g.get(4));
s.add(a * (Double) list2.get(6) + (1 - a) * ((Double) s.get(5) + (Double) g.get(5)));
g.add(a * ((Double) s.get(6) - (Double) s.get(5)) + (1 - a) * (Double) g.get(5));
s.add(a * (Double) list2.get(7) + (1 - a) * ((Double) s.get(6) + (Double) g.get(6)));
g.add(a * ((Double) s.get(7) - (Double) s.get(6)) + (1 - a) * (Double) g.get(6));
s.add(a * (Double) list2.get(8) + (1 - a) * ((Double) s.get(7) + (Double) g.get(7)));
g.add(a * ((Double) s.get(8) - (Double) s.get(7)) + (1 - a) * (Double) g.get(7));
s.add(a * (Double) list2.get(9) + (1 - a) * ((Double) s.get(8) + (Double) g.get(8)));
g.add(a * ((Double) s.get(9) - (Double) s.get(8)) + (1 - a) * (Double) g.get(8));
s.add(a * (Double) list2.get(10) + (1 - a) * ((Double) s.get(9) + (Double) g.get(9)));
g.add(a * ((Double) s.get(10) - (Double) s.get(9)) + (1 - a) * (Double) g.get(9));
s.add(a * (Double) list2.get(11) + (1 - a) * ((Double) s.get(10) + (Double) g.get(10)));
g.add(a * ((Double) s.get(11) - (Double) s.get(10)) + (1 - a) * (Double) g.get(10));
```

```
s.add(a * (Double) list2.get(12) + (1 - a) * ((Double) s.get(11) + (Double) g.get(11)));
g.add(a * ((Double) s.get(12) - (Double) s.get(11)) + (1 - a) * (Double) g.get(11));
s.add(a * (Double) list2.get(13) + (1 - a) * ((Double) s.get(12) + (Double) g.get(12)));
g.add(a * ((Double) s.get(13) - (Double) s.get(12)) + (1 - a) * (Double) g.get(12));
s.add(a * (Double) list2.get(14) + (1 - a) * ((Double) s.get(13) + (Double) g.get(13)));
g.add(a * ((Double) s.get(14) - (Double) s.get(13)) + (1 - a) * (Double) g.get(13));
s.add(a * (Double) list2.get(15) + (1 - a) * ((Double) s.get(14) + (Double) g.get(14)));
g.add(a * ((Double) s.get(15) - (Double) s.get(14)) + (1 - a) * (Double) g.get(14));
s.add(a * (Double) list2.get(16) + (1 - a) * ((Double) s.get(15) + (Double) g.get(15)));
g.add(a * ((Double) s.get(16) - (Double) s.get(15)) + (1 - a) * (Double) g.get(15));
s.add(a * (Double) list2.get(17) + (1 - a) * ((Double) s.get(16) + (Double) g.get(16)));
g.add(a * ((Double) s.get(17) - (Double) s.get(16)) + (1 - a) * (Double) g.get(16));
s.add(a * (Double) list2.get(18) + (1 - a) * ((Double) s.get(17) + (Double) g.get(17)));
g.add(a * ((Double) s.get(18) - (Double) s.get(17)) + (1 - a) * (Double) g.get(17));
s.add(a * (Double) list2.get(19) + (1 - a) * ((Double) s.get(18) + (Double) g.get(18)));
g.add(a * ((Double) s.get(19) - (Double) s.get(18)) + (1 - a) * (Double) g.get(18));
s.add(a * (Double) list2.get(20) + (1 - a) * ((Double) s.get(19) + (Double) g.get(19)));
g.add(a * ((Double) s.get(20) - (Double) s.get(19)) + (1 - a) * (Double) g.get(19));
s.add(a * (Double) list2.get(21) + (1 - a) * ((Double) s.get(20) + (Double) g.get(20)));
g.add(a * ((Double) s.get(21) - (Double) s.get(20)) + (1 - a) * (Double) g.get(20));
s.add(a * (Double) list2.get(22) + (1 - a) * ((Double) s.get(21) + (Double) g.get(21)));
g.add(a * ((Double) s.get(22) - (Double) s.get(21)) + (1 - a) * (Double) g.get(21));
s.add(a * (Double) list2.get(23) + (1 - a) * ((Double) s.get(22) + (Double) g.get(22)));
g.add(a * ((Double) s.get(23) - (Double) s.get(22)) + (1 - a) * (Double) g.get(22));
```

Figure 25

Figure 25, I added s and g values in a new arraylist. Then I used this arraylist data;

```
tahmin2.add(x+y);
tahmin2.add((Double) s.get(0) + (Double) g.get(0));
tahmin2.add((Double) s.get(1) + (Double) g.get(1));
tahmin2.add((Double) s.get(2) + (Double) g.get(2));
tahmin2.add((Double) s.get(3) + (Double) g.get(3));
tahmin2.add((Double) s.get(4) + (Double) g.get(4));
tahmin2.add((Double) s.get(5) + (Double) g.get(5));
tahmin2.add((Double) s.get(6) + (Double) g.get(6));
tahmin2.add((Double) s.get(7) + (Double) g.get(7));
tahmin2.add((Double) s.get(8) + (Double) g.get(8));
tahmin2.add((Double) s.get(9) + (Double) g.get(9));
tahmin2.add((Double) s.get(10) + (Double) g.get(10));

tahmin2.add((Double) s.get(11) + (Double) g.get(11));
tahmin2.add((Double) s.get(12) + (Double) g.get(12));
tahmin2.add((Double) s.get(13) + (Double) g.get(13));
tahmin2.add((Double) s.get(14) + (Double) g.get(14));
tahmin2.add((Double) s.get(15) + (Double) g.get(15));
tahmin2.add((Double) s.get(16) + (Double) g.get(16));
tahmin2.add((Double) s.get(17) + (Double) g.get(17));
tahmin2.add((Double) s.get(18) + (Double) g.get(18));
tahmin2.add((Double) s.get(19) + (Double) g.get(19));
tahmin2.add((Double) s.get(20) + (Double) g.get(20));
tahmin2.add((Double) s.get(21) + (Double) g.get(21));
tahmin2.add((Double) s.get(22) + (Double) g.get(22));
tahmin2.add((Double) s.get(23) + (Double) g.get(23));
```

Figure 26

- Figure 26, I created a new arraylist. Then I fill it according to s and g values for double exponential smoothing.

```
Arraylist errorDouble = new Arraylist();
errorDouble.add((Double)tahmin2.get(0)-(Double)list2.get(0));
errorDouble.add((Double) tahmin2.get(1) - (Double) list2.get(0));
errorDouble.add((Double) tahmin2.get(2) - (Double) list2.get(1));
errorDouble.add((Double) tahmin2.get(3) - (Double) list2.get(2));
errorDouble.add((Double) tahmin2.get(4) - (Double) list2.get(3));
errorDouble.add((Double) tahmin2.get(5) - (Double) list2.get(4));
errorDouble.add((Double) tahmin2.get(6) - (Double) list2.get(5));
errorDouble.add((Double) tahmin2.get(7) - (Double) list2.get(6));
errorDouble.add((Double) tahmin2.get(8) - (Double) list2.get(7));
errorDouble.add((Double) tahmin2.get(9) - (Double) list2.get(8));
errorDouble.add((Double) tahmin2.get(10) - (Double) list2.get(9));
errorDouble.add((Double) tahmin2.get(11) - (Double) list2.get(10));
errorDouble.add((Double) tahmin2.get(12) - (Double) list2.get(11));
errorDouble.add((Double) tahmin2.get(13) - (Double) list2.get(12));
errorDouble.add((Double) tahmin2.get(14) - (Double) list2.get(12));
errorDouble.add((Double) tahmin2.get(15) - (Double) list2.get(14));
errorDouble.add((Double) tahmin2.get(16) - (Double) list2.get(15));
errorDouble.add((Double) tahmin2.get(17) - (Double) list2.get(16));
errorDouble.add((Double) tahmin2.get(18) - (Double) list2.get(17));
errorDouble.add((Double) tahmin2.get(19) - (Double) list2.get(18));
errorDouble.add((Double) tahmin2.get(20) - (Double) list2.get(19));
errorDouble.add((Double) tahmin2.get(21) - (Double) list2.get(20));
errorDouble.add((Double) tahmin2.get(22) - (Double) list2.get(21));
errorDouble.add((Double) tahmin2.get(23) - (Double) list2.get(22));
errorDouble.add((Double) tahmin2.get(24) - (Double) list2.get(23));
```

Figure 27

- Figure 27, I create a new arraylist. I used this for MSE calculation.

For dataset2:

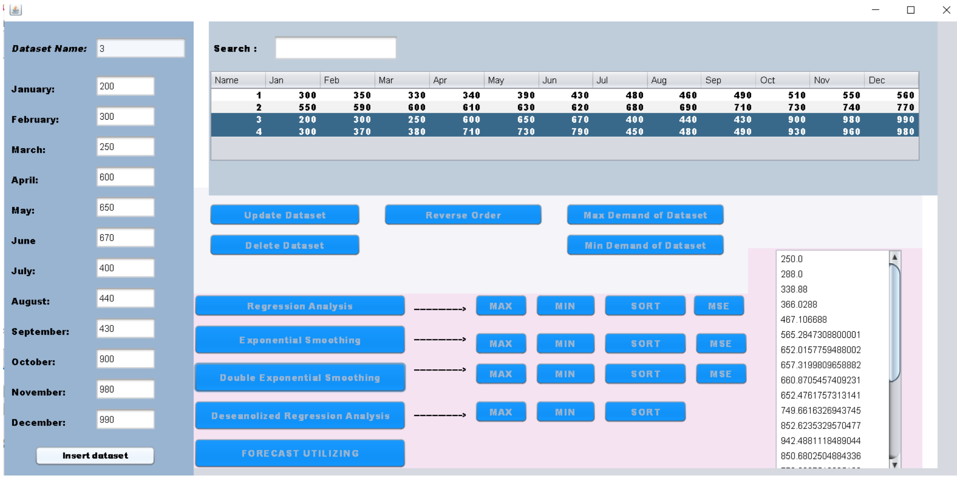


Figure a

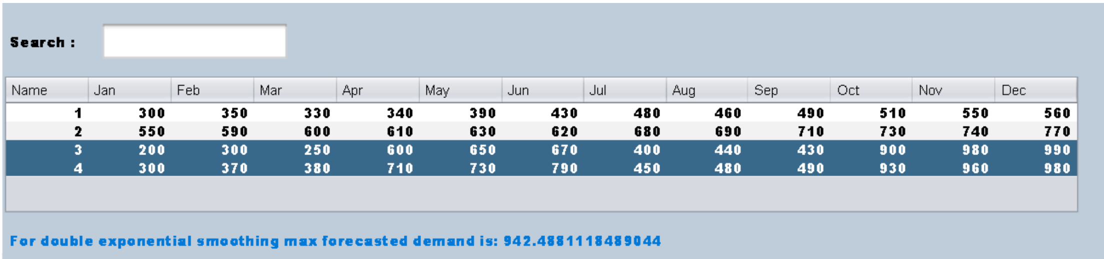


Figure b

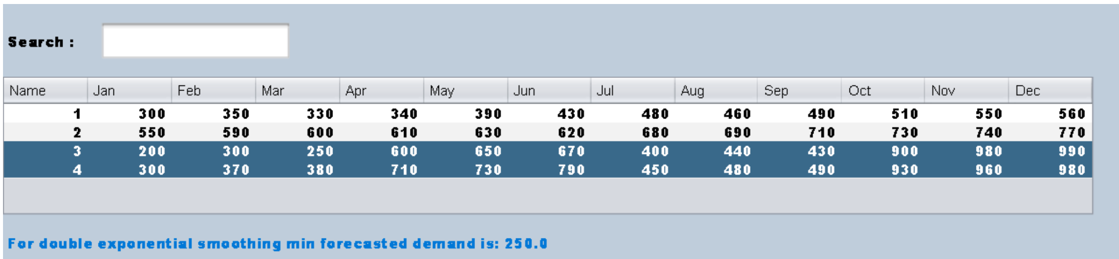


Figure c

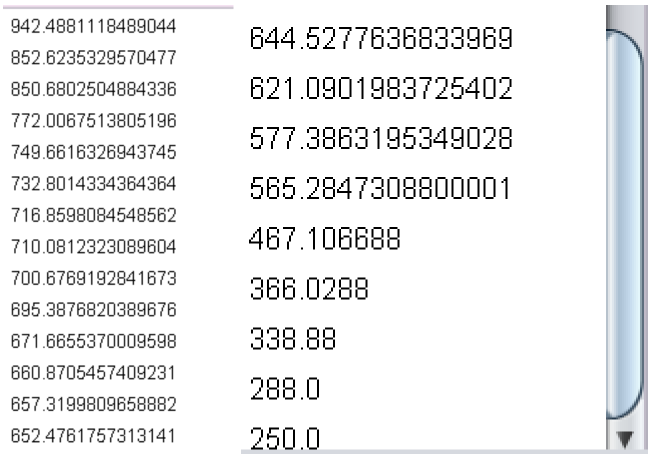


Figure d

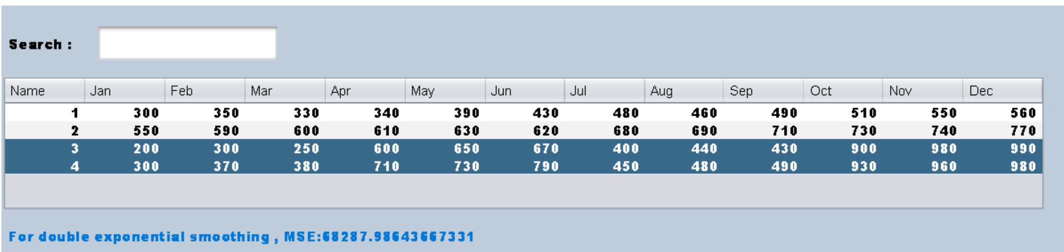


Figure e

- Figure a and Figure b show Dataset2 Forecast year 1 and forecast year 2 values. To obtain this data, I first pulled the data from the selected rows from the table and added this data to an arraylist that I created myself.Then I applied the double exponential analysis formula and added the forecasted years to a new arraylist using this method. Then I printed the arraylist containing the estimated years into a Jtable.
- For this method code is the same as in exponential smoothing code. In my code, pulled data from selected rows codes are identical to all methods.
- Figure c, Figure d and Figure e shows maximum and minimum forecasted value in the dataset and shows Sort forecasted sales in descending order respectively. At first, I listed these values. I used the insertion sort algorithm to sort these values. While using this algorithm, my data was arranged in order from smallest to largest. I called the last number from the sorted data to find the largest number, and the first number from these sorted data to find the smallest value.
- Figure e shows MSE values for the double exponential method. To find the MSE value, I created a new arraylist and entered the data about the errors of these values in the arraylist, and found it by doing the necessary operations.
- Then I find the average value of this data. MSE was calculated at the end of these processes.

5. Regression Analysis

5.1 Solution on Excel

5.1.1 For Dataset 1;

b	19,61
a	301,09

Data Set 1	x	x	x^2	x^2	Month	Demand Year 1	Demand Year 2	x*y	x*y	Forecast Year 1	Forecast Year 2		Error Year 2	Error^2 Year 1	Error^2 Year 2	MSE
	1	13	1	169	Jan.	300	550	300	7.150	320,70	556,06	20,70	6,06	428,49	36,68	357,93
	2	14	4	196	Feb	350	590	700	8.260	340,31	575,67	-9,69	-14,33	93,84	205,36	
	3	15	9	225	Mar.	330	600	990	9.000	359,93	595,28	29,93	-4,72	895,57	22,25	
	4	16	16	256	Apr.	340	610	1.360	9.760	379,54	614,90	39,54	4,90	1.563,34	23,97	
	5	17	25	289	May.	390	630	1.950	10.710	399,15	634,51	9,15	4,51	83,76	20,33	
	6	18	36	324	Jun.	430	620	2.580	11.160	418,77	654,12	-11,23	34,12	126,22	1.164,29	
	7	19	49	361	Jul	480	680	3.360	12.920	438,38	673,73	-41,62	-6,27	1.732,37	39,25	
	8	20	64	400	Aug.	460	690	3.680	13.800	457,99	693,35	-2,01	3,35	4,03	11,21	
	9	21	81	441	Sep.	490	710	4.410	14.910	477,60	712,96	-12,40	2,96	153,65	8,77	
	10	22	100	484	Oct.	510	730	5.100	16.060	497,22	732,57	-12,78	2,57	163,40	6,63	
	11	23	121	529	Nov.	550	740	6.050	17.020	516,83	752,19	-33,17	12,19	1.100,22	148,52	
	12	24	144	576	Dec.	560	770	6.720	18.480	536,44	771,80	-23,56	1,80	554,91	3,24	
Toplam		300		4.900			13.110		186.430							

5.1.2 For Dataset 2;

b	17,90
a	375,36

Data Set 1	x	x	x^2	x^2	Month	Demand Year 1	Demand Year 2	x*y	x*y	Forecast Year 1	Forecast Year 2	Error Year 1	Error Year 2	Error^2 Year 1	Error^2 Year 2	MSE
	1	13	1	169	Jan.	200	300	200	3.900	393,27	608,12	193,27	308,12	37.352,00	94.937,22	49.188,87
	2	14	4	196	Feb	300	370	600	5.180	411,17	626,02	111,17	256,02	12.358,99	65.547,87	
	3	15	9	225	Mar.	250	380	750	5.700	429,08	643,93	179,08	263,93	32.067,99	69.657,74	
	4	16	16	256	Apr.	600	710	2.400	11.360	446,98	661,83	-153,02	-48,17	23.415,21	2.320,17	
	5	17	25	289	May.	650	730	3.250	12.410	464,88	679,74	-185,12	-50,26	34.267,91	2.526,45	
	6	18	36	324	Jun.	670	790	4.020	14.220	482,79	697,64	-187,21	-92,36	35.048,18	8.530,26	
	7	19	49	361	Jul	400	450	2.800	8.550	500,69	715,54	100,69	265,54	10.139,03	70.514,11	
	8	20	64	400	Aug.	440	480	3.520	9.600	518,60	733,45	78,60	253,45	6.177,50	64.236,54	
	9	21	81	441	Sep.	430	490	3.870	10.290	536,50	751,35	106,50	261,35	11.342,56	68.305,72	
	10	22	100	484	Oct.	900	930	9.000	20.460	554,41	769,26	-345,59	-160,74	119.435,35	25.838,00	
	11	23	121	529	Nov.	980	960	10.780	22.080	572,31	787,16	-407,69	-172,84	166.211,02	29.872,86	
	12	24	144	576	Dec.	990	980	11.880	23.520	590,21	805,07	-399,79	-174,93	159.828,45	30.601,67	
Toplam		300		4.900			14.380		200.340							

5.2 Solution on Java

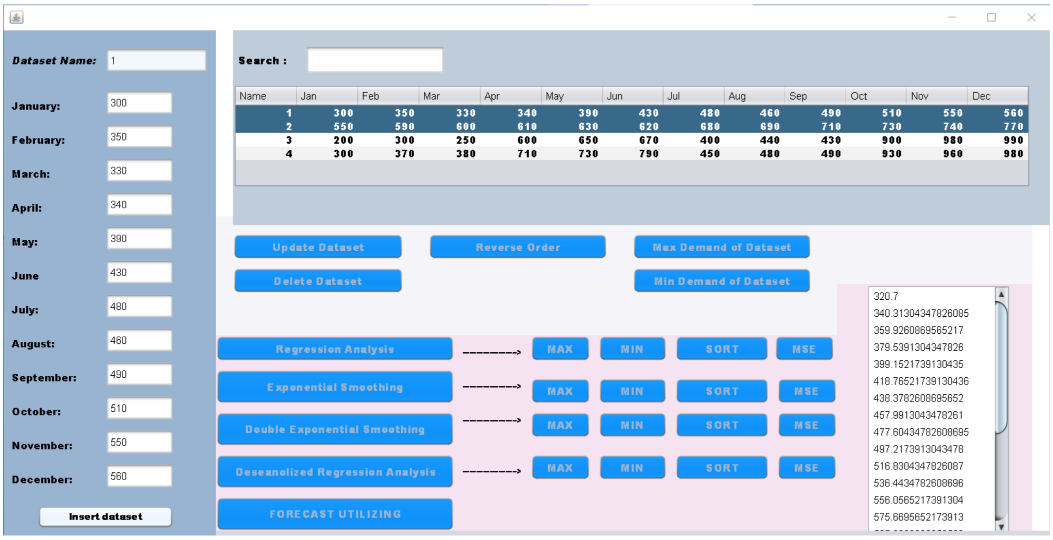


Figure 28

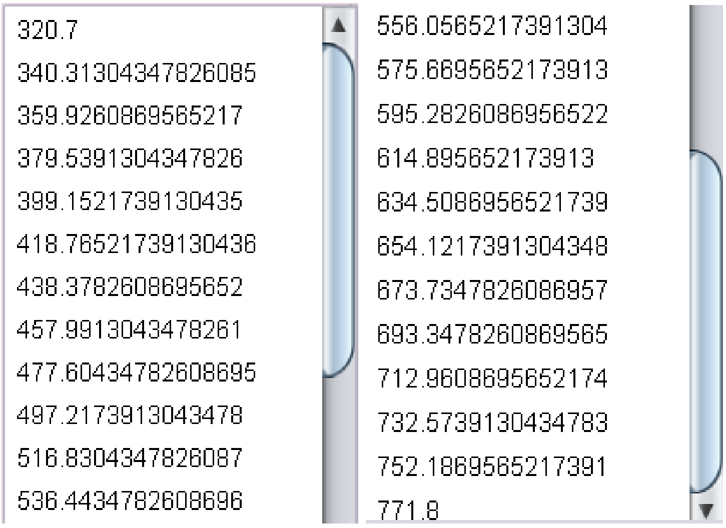


Figure 29

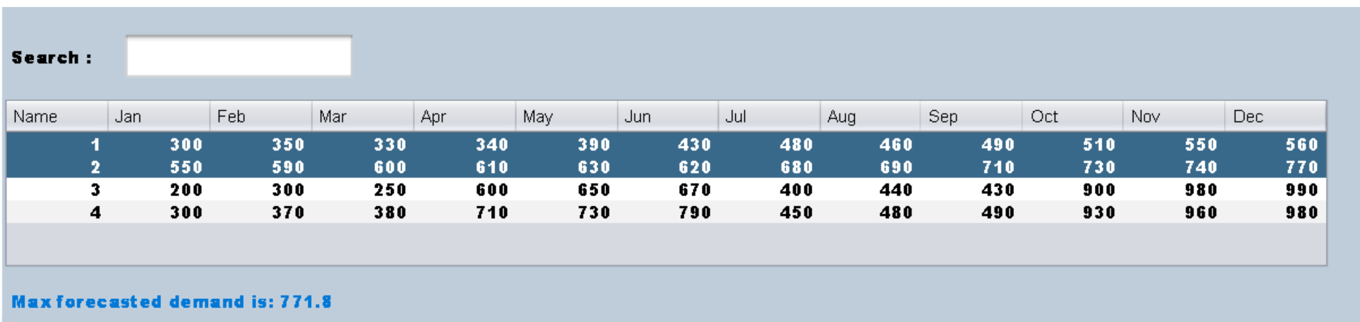
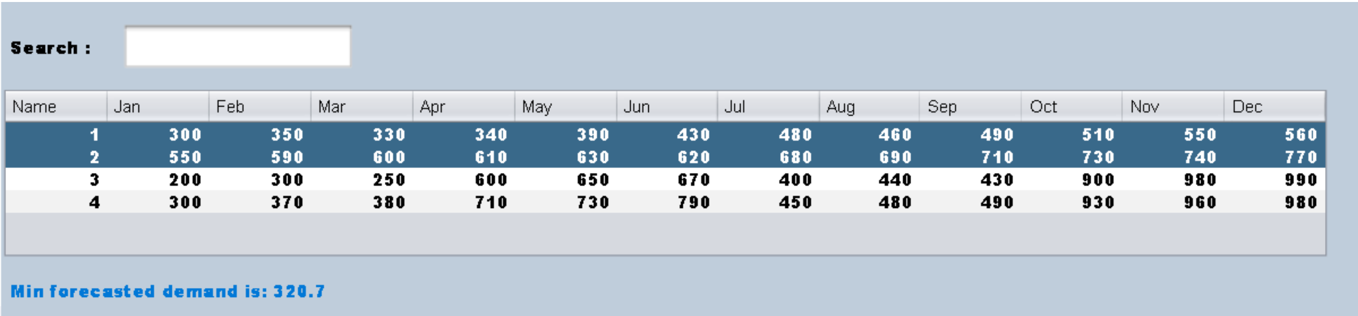


Figure 30



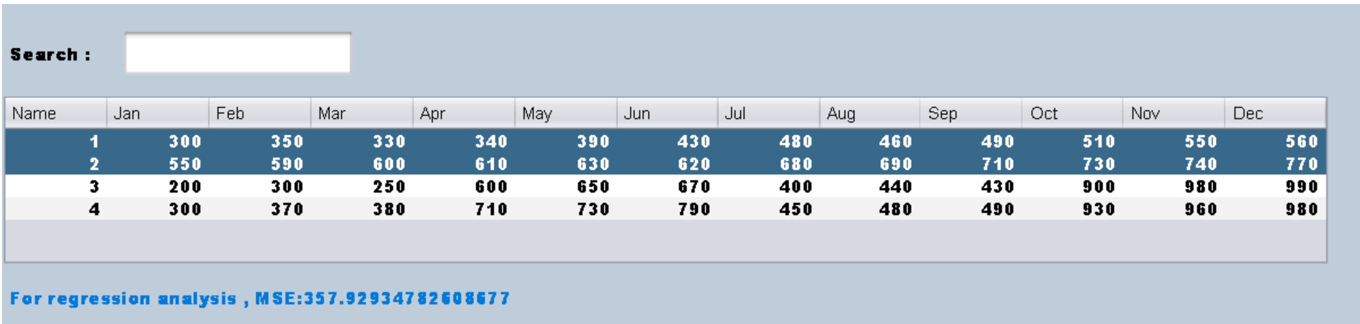
Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

Min forecasted demand is: 320.7

Figure 31

Figure 32



Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For regression analysis , MSE:357.92934782608677

Figure 33

- Figure 28 and Figure 29 Dataset1 Forecast year 1 and forecast year 2 values. To obtain this data, I first pulled the data from the selected rows from the table and added this data to an arraylist that I created myself.Then I applied the regression analysis formula and added the forecasted years to a new arraylist using this method. Then I printed the arraylist containing the estimated years into a Jtable.

```
Arraylist tahmin1 = new Arraylist();
tahmin1.add(list.get(0));
tahmin1.add((Double) list.get(0) * a + (Double) tahmin1.get(0) * (1 - a));
tahmin1.add((Double) list.get(1) * a + (Double) tahmin1.get(1) * (1 - a));
tahmin1.add((Double) list.get(2) * a + (Double) tahmin1.get(2) * (1 - a));
tahmin1.add((Double) list.get(3) * a + (Double) tahmin1.get(3) * (1 - a));
tahmin1.add((Double) list.get(4) * a + (Double) tahmin1.get(4) * (1 - a));
tahmin1.add((Double) list.get(5) * a + (Double) tahmin1.get(5) * (1 - a));
tahmin1.add((Double) list.get(6) * a + (Double) tahmin1.get(6) * (1 - a));
tahmin1.add((Double) list.get(7) * a + (Double) tahmin1.get(7) * (1 - a));
tahmin1.add((Double) list.get(8) * a + (Double) tahmin1.get(8) * (1 - a));
tahmin1.add((Double) list.get(9) * a + (Double) tahmin1.get(9) * (1 - a));
tahmin1.add((Double) list.get(10) * a + (Double) tahmin1.get(10) * (1 - a));
tahmin1.add(a * (Double) list.get(11) + (Double) tahmin1.get(11) * (1 - a));

tahmin1.add((Double) list.get(12) * a + (Double) tahmin1.get(12) * (1 - a));
tahmin1.add((Double) list.get(13) * a + (Double) tahmin1.get(13) * (1 - a));
tahmin1.add((Double) list.get(14) * a + (Double) tahmin1.get(14) * (1 - a));
tahmin1.add((Double) list.get(15) * a + (Double) tahmin1.get(15) * (1 - a));
tahmin1.add((Double) list.get(16) * a + (Double) tahmin1.get(16) * (1 - a));
tahmin1.add((Double) list.get(17) * a + (Double) tahmin1.get(17) * (1 - a));
tahmin1.add((Double) list.get(18) * a + (Double) tahmin1.get(18) * (1 - a));
tahmin1.add((Double) list.get(19) * a + (Double) tahmin1.get(19) * (1 - a));
tahmin1.add((Double) list.get(20) * a + (Double) tahmin1.get(20) * (1 - a));
tahmin1.add((Double) list.get(21) * a + (Double) tahmin1.get(21) * (1 - a));
tahmin1.add((Double) list.get(22) * a + (Double) tahmin1.get(22) * (1 - a));
tahmin1.add((Double) list.get(23) * a + (Double) tahmin1.get(23) * (1 - a));
```

Figure 34

- In Figure 34, this code adds the forecast years to the new arraylist.
- Figure 33shows MSE values for the regression analysis method. To find the MSE value, I created a new arraylist and entered the data about the errors of these values in the arraylist, and found it by doing the necessary operations.
- Then I find the average value of this data. MSE was calculated at the end of these processes.
- Figure 30, Figure 31 and Figure 32 shows maximum and minimum forecasted value in the dataset and shows Sort forecasted sales in descending order respectively. At first, I listed these values. I used the insertion sort algorithm to sort these values. While using this algorithm, my data was arranged in order from smallest to largest. I called the last number from the sorted data to find the largest number, and the first number from these sorted data to find the smallest value.

```
int j, P, Tmp;
for (P = 1; P < 24; P++) {
    double Tmp2 = (Double) max3[P];
    for (j = P; j > 0 && (Double) max3[j - 1] > Tmp2; j--) {
        max3[j] = max3[j - 1]; //Shift A[j-1] to right
    }
    max3[j] = Tmp2;
}
```

Figure 35

- Figure 35 shows the insertion sort algorithm.

For dataset 2:

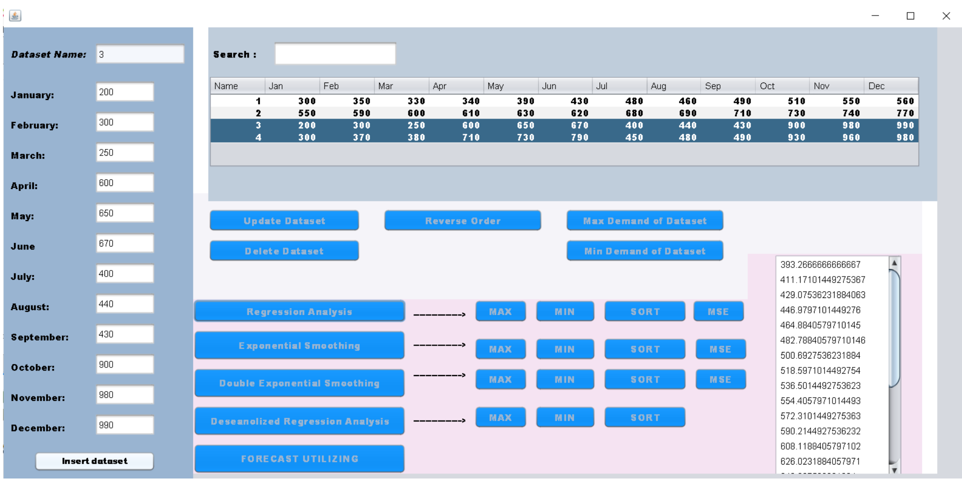


Figure 36

393.2666666666667	608.1188405797102
411.17101449275367	626.0231884057971
429.07536231884063	643.927536231884
446.9797101449276	661.831884057971
464.8840579710145	679.736231884058
482.78840579710146	697.640579710145
500.6927536231884	715.5449275362319
518.5971014492754	733.4492753623189
536.5014492753623	751.3536231884058
554.4057971014493	769.2579710144928
572.3101449275363	787.1623188405797
590.2144927536232	805.0666666666666

Figure 37

- Figure 36 and Figure 37 Dataset2 Forecast year 1 and forecast year 2 values. To obtain this data, I first pulled the data from the selected rows from the table and added this data to an arraylist that I created myself.Then I applied the regression analysis formula and added the forecasted years to a new arraylist using this method. Then I printed the arraylist containing the estimated years into a Jtable.

Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

Max forecasted demand is: 805.0666666666666

Figure 38

Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

Min forecasted demand is: 393.2666666666667

Figure 39

805.0666666666666
787.1623188405797
769.2579710144928
751.3536231884058
733.4492753623189
715.5449275362319
697.640579710145
679.736231884058
661.831884057971
643.927536231884
626.0231884057971
608.1188405797102
590.2144927536232
572.3101449275363

554.4057971014493
536.5014492753623
518.5971014492754
500.6927536231884
482.78840579710146
464.8840579710145
446.9797101449276
429.07536231884063
411.17101449275367
393.2666666666667

Figure 40

- Figure 38, Figure 39 and Figure 40 shows maximum and minimum forecasted value in the dataset and shows Sort forecasted sales in descending order respectively. At first, I listed these values. I used the insertion sort algorithm to sort these values. While using this algorithm, my data was arranged in order from smallest to largest. I called the last number from the sorted data to find the largest number, and the first number from these sorted data to find the smallest value.

Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For regression analysis , MSE:49188.86714975845

Figure 41

- Figure 41 shows MSE values for the regression analysis method. To find the MSE value, I created a new arraylist and entered the data about the errors of these values in the arraylist, and found it by doing the necessary operations.

6. Deseasonalized Regression Analysis

6.1 Solution on Excel

6.1.1 For Dataset 1

Overall average demand	546,25
b	14,69782
a	362,5273

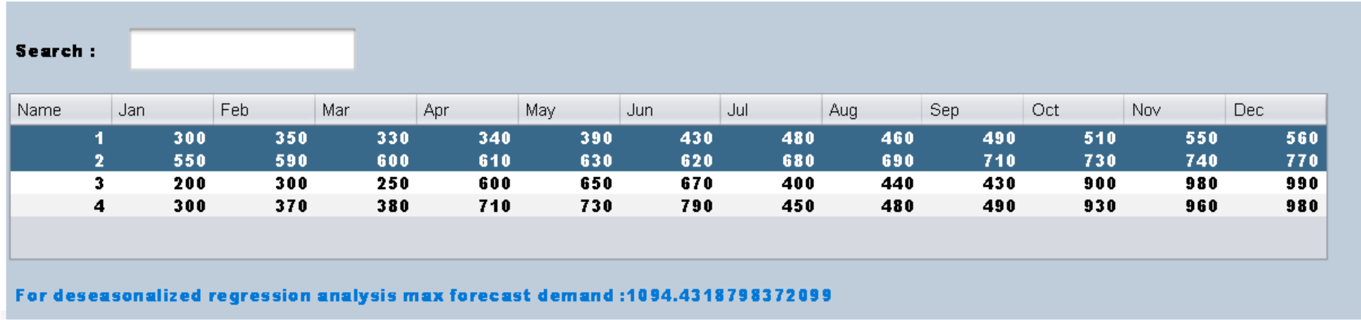
Data Set 1	Month	Demand Year 1	Demand Year 2	Ortalama	Seasonal Demand Factor	Deseason Demand (y)		x		x^2		x*y				Season Forecast
	Jan.	300	550	425	0,78	385,59	706,91	1	13	1	169	385,59	9.189,85	25	729,97	567,94
	Feb	350	590	470	0,86	406,78	685,72	2	14	4	196	813,56	9.600,05	26	744,67	640,72
	Mar.	330	600	465	0,85	387,66	704,84	3	15	9	225	1.162,98	10.572,58	27	759,37	646,42
	Apr.	340	610	475	0,87	391,00	701,50	4	16	16	256	1.564,00	11.224,00	28	774,07	673,10
	May.	390	630	510	0,93	417,72	674,78	5	17	25	289	2.088,60	11.471,25	29	788,76	736,42
	Jun.	430	620	525	0,96	447,40	645,10	6	18	36	324	2.684,43	11.611,71	30	803,46	772,21
	Jul	480	680	580	1,06	452,07	640,43	7	19	49	361	3.164,48	12.168,19	31	818,16	868,71
	Aug.	460	690	575	1,05	437,00	655,50	8	20	64	400	3.496,00	13.110,00	32	832,86	876,69
	Sep.	490	710	600	1,10	446,10	646,40	9	21	81	441	4.014,94	13.574,31	33	847,56	930,95
	Oct.	510	730	620	1,14	449,33	643,17	10	22	100	484	4.493,35	14.149,64	34	862,25	978,67
	Nov.	550	740	645	1,18	465,79	626,71	11	23	121	529	5.123,74	14.414,22	35	876,95	1.035,48
	Dec.	560	770	665	1,22	460,00	632,50	12	24	144	576	5.520,00	15.180,00	36	891,65	1.085,49
Toplam							13.110		300		4.900		180.777,49			

6.1.2 For Dataset 2

Overall average demand	599,1667
b	5,648753
a	528,5572

Data Set 2	Month	Demand Year 1	Demand Year 2	Ortalama	Seasonal Demand Factor	Deseason Demand (y)		x		x^2		x*y				Season Forecast
	Jan.	200	300	250	0,42	479,33	719,00	1	13	1	169	479,33	9.347,00	25	669,78	279,46
	Feb	300	370	335	0,56	536,57	661,77	2	14	4	196	1.073,13	9.264,73	26	675,42	377,64
	Mar.	250	380	315	0,53	475,53	722,80	3	15	9	225	1.426,59	10.842,06	27	681,07	358,06
	Apr.	600	710	655	1,09	548,85	649,48	4	16	16	256	2.195,42	10.391,65	28	686,72	750,71
	May.	650	730	690	1,15	564,43	633,90	5	17	25	289	2.822,16	10.776,32	29	692,37	797,33
	Jun.	670	790	730	1,22	549,92	648,41	6	18	36	324	3.299,52	11.671,44	30	698,02	850,44
	Jul	400	450	425	0,71	563,92	634,41	7	19	49	361	3.947,45	12.053,82	31	703,67	499,13
	Aug.	440	480	460	0,77	573,12	625,22	8	20	64	400	4.584,93	12.504,35	32	709,32	544,57
	Sep.	430	490	460	0,77	560,09	638,24	9	21	81	441	5.040,82	13.403,10	33	714,97	548,90
	Oct.	900	930	915	1,53	589,34	608,99	10	22	100	484	5.893,44	13.397,76	34	720,61	1.100,47
	Nov.	980	960	970	1,62	605,34	592,99	11	23	121	529	6.658,78	13.638,76	35	726,26	1.175,76
	Dec.	990	980	985	1,64	602,21	596,13	12	24	144	576	7.226,50	14.307,01	36	731,91	1.203,23
Toplam							14.380		300		4.900		186.246,07			

6.2 Solution on Java

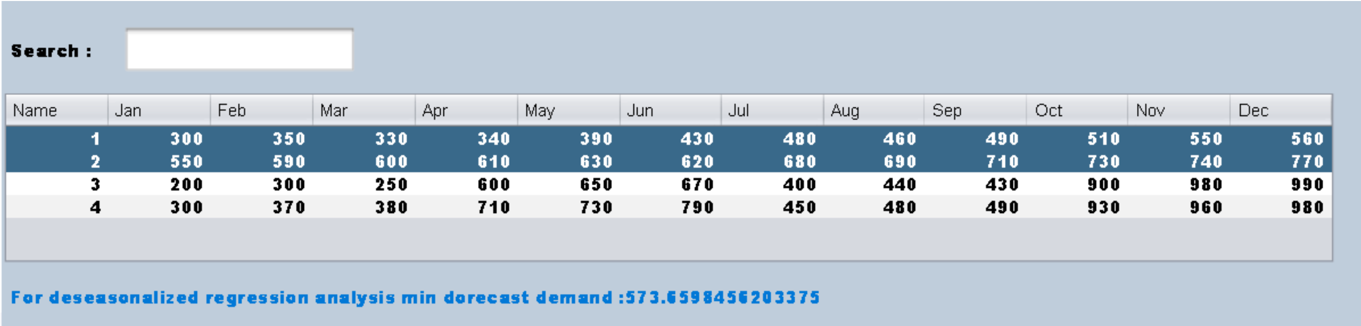


Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For deseasonalized regression analysis max forecast demand :1094.4318798372099

Figure 42



Search :

Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	300	350	330	340	390	430	480	460	490	510	550	560
2	550	590	600	610	630	620	680	690	710	730	740	770
3	200	300	250	600	650	670	400	440	430	900	980	990
4	300	370	380	710	730	790	450	480	490	930	960	980

For deseasonalized regression analysis min dorecast demand :573.6598456203375

Figure 43

1094.4318798372099

1044.1617755253765

987.0081980563623

939.0251682925341

884.4277329654082

876.5124884590258

779.2688762194006

743.2816038965295

679.4913713510523

652.6746467333671

647.0464767749901

573.6598456203375

Figure 44

- Figure 30, shows for deseasonalized regression analysis season forecast’s max element. Figure 31, shows for deseasonalized regression analysis season forecast’s min element. Figure 32, shows values according to the descending sort algorithm. At first, I listed these values. I used the insertion sort algorithm to sort these values. While using this algorithm, my data was arranged in order from smallest to largest. I called the last number from the sorted data to find the largest number, and the first number from these sorted data to find the smallest value.

For dataset 2:

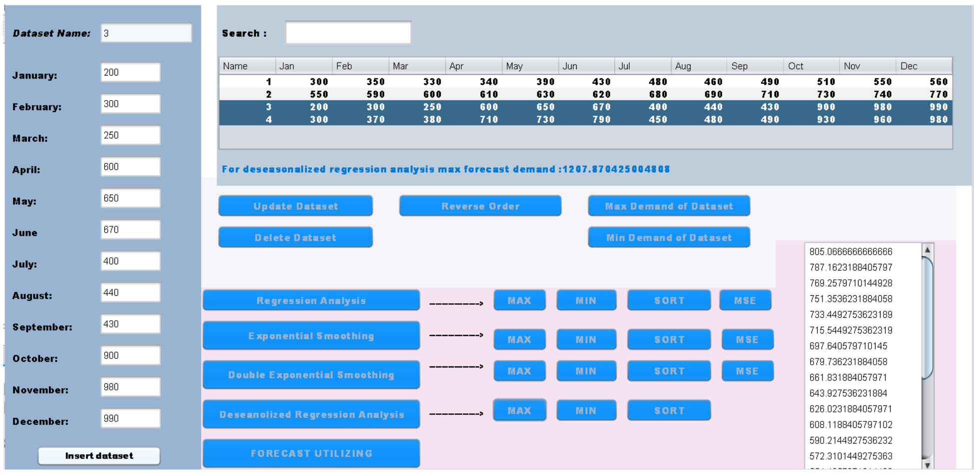


Figure 45

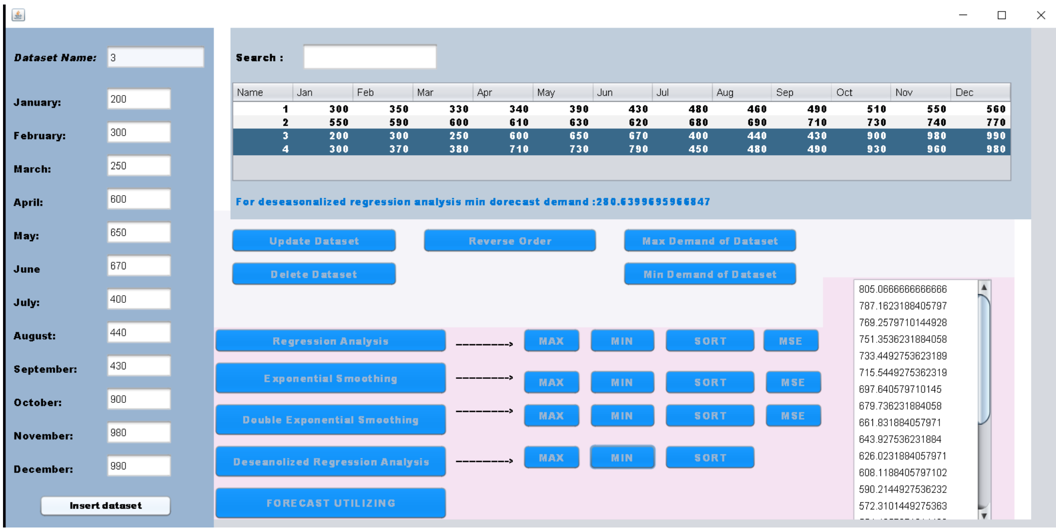


Figure 46

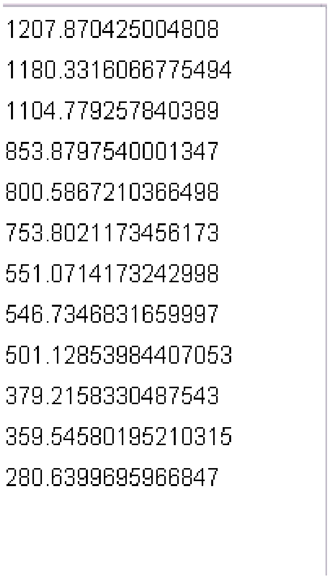


Figure 47

- Figure 45, shows for deseasonalized regression analysis season forecast's max element. Figure 46, shows for deseasonalized regression analysis season forecast's min element. Figure 47, shows values according to the descending sort algorithm. At first, I listed these values. I used the insertion sort algorithm to sort these values. While using this algorithm, my data was arranged in order from smallest to largest. I called the last number from the sorted data to find the largest number, and the first number from these sorted data to find the smallest value.

7. Result

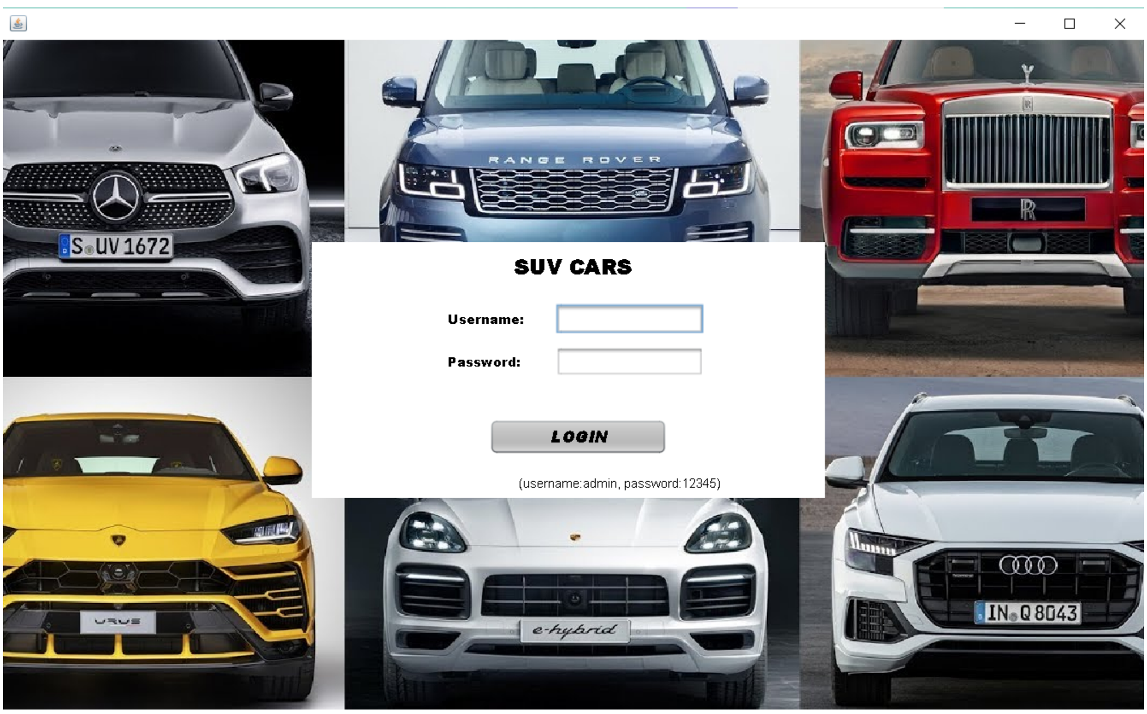
On Excel we found these solution;

	Exponential smoothing	Double exponential smoothing	Regression analysis
Data Set 1	7.507,52	3.028,91	357,93
Data Set 2	69.188,27	68.287,99	49.188,87

- For Dataset 1 Regression Analysis gives the best solution from these solutions as an excel solution.
- For Dataset 2 Regression Analysis gives the best solution from these solutions as an excel solution.

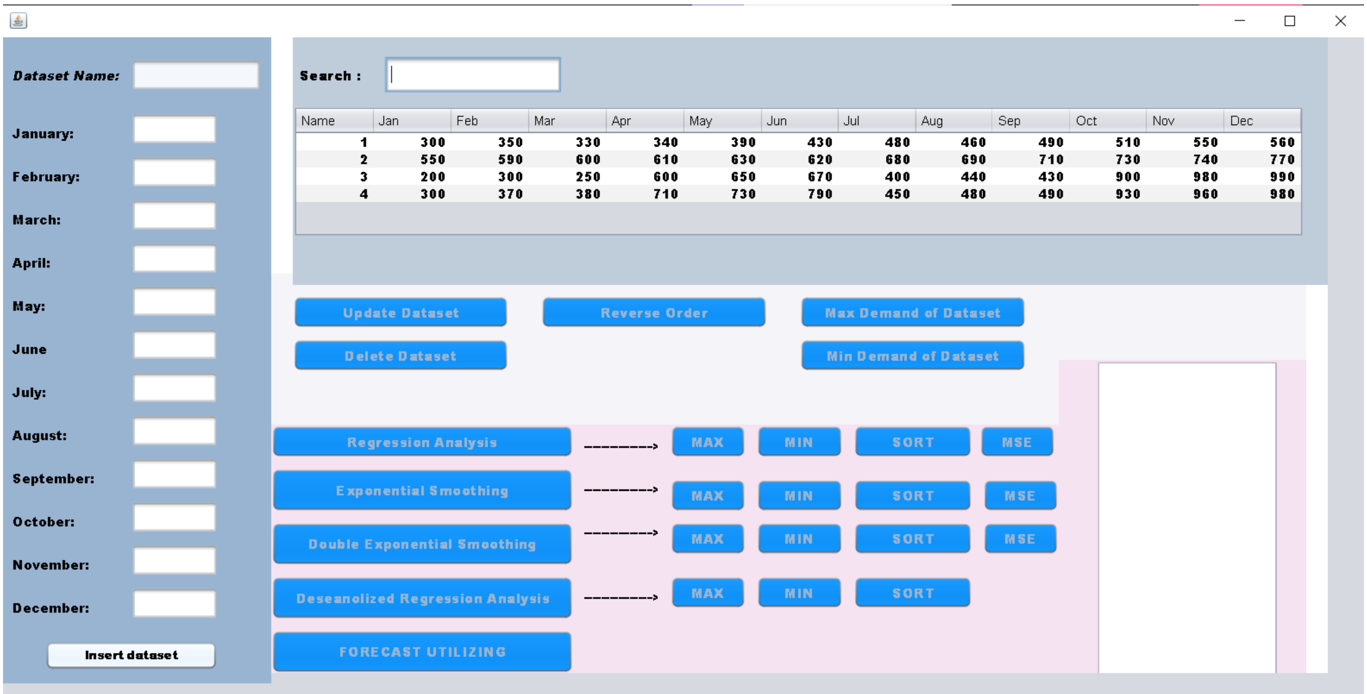
8-About Java Program

I introduce this program briefly.Firstly,This program first page is :



Now the username is "admin" and password is "12345". There is no register page for this final project. User enters the program with only valid username and password combinations.

When user enter the program:



The left side, if the user wants to add a new dataset, the user should enter data and press the "Insert Dataset" button.As soon as the user presses the button, the data comes to the table row.

Buttons:

-Update dataset: When the user presses this button, values come to the left side and the user selects this value and the user should update which they want to .

-Delete dataset: When the user presses this button, dataset delete which the user selects to dataset. It uses the arraylist remove method.

-Reverse order: This button reverse order dataset.

-Max Demand of Dataset: This button uses insertion sort algorithm. Firstly, dataset orders ascending order and max demand button return dataset's biggest index element.

-Min Demand of Dataset: This button uses insertion sort algorithm. Firstly, dataset orders ascending order and min demand button return dataset's smallest index element.

-**Regression Analysis:** Users should select two datasets . This method return forecast utilizing regression analysis.

***MAX:** Find maximum sales count on dataset (insertion sort algorithm)

***MIN:**Find minimum sales count on dataset (insertion sort algorithm)

***SORT:**This method sorts forecasted sales descending order.

*MSE:This method returns MSE value.

-**Exponential Smoothing:**Users should select two datasets . This method return forecast utilizing double exponential smoothing.

*MAX:Find maximum sales count on dataset (insertion sort algorithm)

*MIN:Find minimum sales count on dataset (insertion sort algorithm)

*SORT:This method sorts forecasted sales descending order.

*MSE:This method returns MSE value.

-**Double Exponential Smoothing:**Users should select two datasets . This method returns forecasts utilizing double exponential smoothing.

*MAX:Find maximum sales count on dataset (insertion sort algorithm)

*MIN:Find minimum sales count on dataset (insertion sort algorithm)

*SORT:This method sorts forecasted sales descending order.

*MSE:This method returns MSE value.

-**Deseasonalized Regression Analysis :**Users should select two datasets . This method return forecast utilizing deseasonalized smoothing.

*MAX:Find maximum sales count on dataset (insertion sort algorithm)

*MIN:Find minimum sales count on dataset (insertion sort algorithm)

*SORT:This method sorts forecasted sales descending order.

-**Forecast Utilizing:** This button gives the user the best forecasting method.

Arraylist Methods

```java
public Arraylist() {
    list = new Object[12];
    int listSize = list.length;
}


public Arraylist(int n) {
    int defaultValue = 10;
    int size = n;
    list = new Object[n];

    if (size < defaultValue) {
        list = new Object[size];
    } else {
        while (size >= defaultValue) {
            defaultValue *= 2;
            n = defaultValue;
        }
        list = new Object[n];
    }

}
```

```java
public void add(Object o) {
    int i = 0;
    int length = list.length;
    list = reSize();
    while(i < length){
        if(list[i] == null){
            list[i] = o;
            i = length;
        }else{
            i++;
        }
    }
    arSize++;

}
```

```java
public void add(int index, Object o) {
    Object[] a2 = new Object[list.length];
    reSize();
    for (int i = 0; i < index; i++) {
        a2[i] = list[i];
    }
    a2[index] = o;

    for (int i = index + 1; i < list.length; i++) {
        a2[i] = list[i - 1];
    }

    for (int i = 0; i < list.length; i++) {
        list[i] = a2[i];
    }

    arSize++;
}


public Object get(int index) {

    return list[index];
}


public int size() {
    int listSize = 0;
    for (int i = 0; i < list.length; i++) {
        if (list[i] != null) {
            listSize++;
        }
    }
    return listSize;
}
```

```java
public boolean isEmpty() {
    boolean empty = true;
    for (int i = 0; i < list.length; i++) {
        if (list[i] == null) {

        } else {
            empty = false;
        }
    }
    return empty;
}
```

```java
public int find(Object o) {
    int i = 0;
    if(arSize == 0){
        System.out.println("Error: Your ArrayList is empty.");
    }else
        while (i < list.length) {
            if (list[i] != o) {
                i++;
            } else {
                return i;
            }
        }
    return -1;

}
```

```java
public void remove(Object o) {
    int x = 0;
    Object[] a2 = new Object[list.length];
    if(arSize == 0){
        System.out.println("Error: Your ArrayList is empty.");
    }else
        while (x < list.length) {

            if (list[x] == o) {
                for (int i = 0; i <= x - 1; i++) {
                    a2[i] = list[i];
                }
                for (int i = x; i < list.length - 1; i++) {
                    a2[i] = list[i + 1];
                }

                for (int i = 0; i < list.length - 1; i++) {
                    list[i] = a2[i];
                }
                x = list.length;
            } else {
                x++;
            }
        }

}
```

```java
public String toString() {
    int x = 0;
    String printList = "";
    if (list[0] == null) {
        return null;
    } else {
        printList = printList + "[";
        while (list[x] != null) {
            x++;
        }
        for (int i = 0; i < x - 1; i++) {
            printList = printList + list[i];
            printList = printList + ", ";

        }

        printList = printList + list[x - 1] + "]";
    }
    return printList;

}



private Object [] reSize(){
    Object [] temp;
    if(arSize >=list.length-1){
        temp = new Object[list.length*2];
        for(int i =0; i< list.length; i++){
            temp[i] = list[i];
        }
        list = temp;
    }
    return list;
}
```