CSC478 Final Project

Allstate Claims Analysis and Prediction

By Yiming WANG

Objectives of the data analysis project:

This project is a data analysis competition holds by insurance company Allstate (deadline of the competition is 12 Dec 2016, not due yet).
The data set is real world insurance claims. The objective of this project is to predict the value of ‘loss’ in each claims (continuous value) in test set.

Why it is important?
If an insurance company can predict ‘loss’ very accuracy, then, the pricing could be much more accuracy which means more sales and profit for the insurance company.

Data Preprocessing:

The data set is downloaded from www.kaggle.com case competition: Allstate Claims Severity

The dataset has no label, so, can’t figure out the meaning of each feature. It contains 188,319 tuples for training and 125,547 tuples for testing.

There are 116 categorical and 14 continuous attributes:

id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	cat11	\
0	1	0	1	0	1	0	0	0	0	1	0	1
1	2	0	1	0	0	0	0	0	0	1	1	0
2	5	0	1	0	0	1	0	0	0	1	1	1

	cat12	cat13	cat14	cat15	cat16	cat17	cat18	cat19	cat20	cat21	\
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0

	cat22	cat23	cat24	cat25	cat26	cat27	cat28	cat29	cat30	cat31	\
0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0

	cat32	cat33	cat34	cat35	cat36	cat37	cat38	cat39	cat40	cat41	\
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0

	cat42	cat43	cat44	cat45	cat46	cat47	cat48	cat49	cat50	cat51	\
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0

```
2      0      0      0      0      0      0      0      0      0      0

   cat52  cat53  cat54  cat55  cat56  cat57  cat58  cat59  cat60  cat61  \
0      0      0      0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      0      0      0

   cat62  cat63  cat64  cat65  cat66  cat67  cat68  cat69  cat70  cat71  \
0      0      0      0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      0      0      0

   cat72  cat73  cat74  cat75  cat76  cat77  cat78  cat79  cat80  cat81  \
0      0      0      0      1      0      3      1      1      3      3
1      0      0      0      0      0      3      1      1      3      3
2      0      0      0      0      0      3      1      1      1      3

   cat82  cat83  cat84  cat85  cat86  cat87  cat88  cat89  cat90  cat91  \
0      1      3      2      1      3      1      0      0      0      0
1      0      1      2      1      3      1      0      0      0      0
2      1      3      2      1      1      1      0      0      0      0

   cat92  cat93  cat94  cat95  cat96  cat97  cat98  cat99  cat100  cat101  \
0      0      3      1      2      4      0      2     15       1       6
1      0      3      3      2      4      4      3     15      11       5
2      0      3      3      2      4      4      0      1      11      14

   cat102  cat103  cat104  cat105  cat106  cat107  cat108  cat109  cat110  \
0       0       0       8       4       6       9       6      45      28
1       0       0       4       4       8      10      10      33      65
2       0       1       4       5       7       5       0       2      85

   cat111  cat112  cat113  cat114  cat115  cat116     cont1     cont2  \
0       2      19      55       0      14     269  0.726300  0.245921
1       0      22      38       0      14      85  0.330514  0.737068
2       0      28       5       0       8     153  0.261841  0.358319

      cont3     cont4     cont5     cont6     cont7    cont8    cont9  \
0  0.187583  0.789639  0.310061  0.718367  0.335060  0.30260  0.67135
1  0.592681  0.614134  0.885834  0.438917  0.436585  0.60087  0.35127
2  0.484196  0.236924  0.397069  0.289648  0.315545  0.27320  0.26076

    cont10    cont11    cont12    cont13    cont14     loss
0  0.83510  0.569745  0.594646  0.822493  0.714843  2213.18
```

1	0.43919	0.338312	0.366307	0.611431	0.304496	1283.60
2	0.32446	0.381398	0.373424	0.195709	0.774425	3005.09

Statistic data for the data set:

	id	cat1	cat2	cat3 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	294135.982561	0.248346	0.433294	0.054827
std	169336.084867	0.432055	0.495532	0.227644
min	1.000000	0.000000	0.000000	0.000000
25%	147748.250000	0.000000	0.000000	0.000000
50%	294539.500000	0.000000	0.000000	0.000000
75%	440680.500000	0.000000	1.000000	0.000000
max	587633.000000	1.000000	1.000000	1.000000

	cat4	cat5	cat6	cat7 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.318201	0.342936	0.300688	0.024289
std	0.465779	0.474692	0.458559	0.153944
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	1.000000	1.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat8	cat9	cat10	cat11 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.058645	0.399303	0.149242	0.106904
std	0.234961	0.489757	0.356328	0.308992
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	1.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat12	cat13	cat14	cat15 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.151303	0.103373	0.012091	0.000181
std	0.358345	0.304446	0.109294	0.013436
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat16	cat17	cat18	cat19 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.034383	0.006951	0.005241	0.009601
std	0.182212	0.083083	0.072206	0.097512
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat20	cat21	cat22	cat23 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.001083	0.002193	0.000228	0.163941
std	0.032895	0.046779	0.015109	0.370223
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat24	cat25	cat26	cat27 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.033672	0.097436	0.059469	0.106564
std	0.180383	0.296552	0.236500	0.308559
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat28	cat29	cat30	cat31 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.039189	0.019780	0.018894	0.028346
std	0.194045	0.139245	0.136150	0.165959
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat32	cat33	cat34	cat35 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.006431	0.005082	0.003101	0.001131

std	0.079933	0.071106	0.055602	0.033612
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat36	cat37	cat38	cat39 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.169952	0.119951	0.100867	0.026153
std	0.375592	0.324906	0.301153	0.159589
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat40	cat41	cat42	cat43 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.043538	0.037920	0.009001	0.022345
std	0.204065	0.191003	0.094445	0.147804
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat44	cat45	cat46	cat47 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.082849	0.022977	0.004684	0.003722
std	0.275655	0.149831	0.068276	0.060898
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat48	cat49	cat50	cat51 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.001428	0.048806	0.269263	0.006622
std	0.037768	0.215462	0.443578	0.081105
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000

```
75%          0.000000       0.000000       1.000000       0.000000
max          1.000000       1.000000       1.000000       1.000000

                cat52          cat53          cat54          cat55  \
count  188318.000000  188318.000000  188318.000000  188318.000000
mean         0.046799       0.081612       0.024193       0.000770
std          0.211208       0.273773       0.153649       0.027738
min          0.000000       0.000000       0.000000       0.000000
25%          0.000000       0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000       0.000000
75%          0.000000       0.000000       0.000000       0.000000
max          1.000000       1.000000       1.000000       1.000000

                cat56          cat57          cat58          cat59  \
count  188318.000000  188318.000000  188318.000000  188318.000000
mean         0.000966       0.016047       0.001269       0.001593
std          0.031073       0.125658       0.035602       0.039881
min          0.000000       0.000000       0.000000       0.000000
25%          0.000000       0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000       0.000000
75%          0.000000       0.000000       0.000000       0.000000
max          1.000000       1.000000       1.000000       1.000000

                cat60          cat61          cat62          cat63  \
count  188318.000000  188318.000000  188318.000000  188318.000000
mean         0.002368       0.003834       0.000239       0.000420
std          0.048608       0.061800       0.015456       0.020478
min          0.000000       0.000000       0.000000       0.000000
25%          0.000000       0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000       0.000000
75%          0.000000       0.000000       0.000000       0.000000
max          1.000000       1.000000       1.000000       1.000000

                cat64          cat65          cat66          cat67  \
count  188318.000000  188318.000000  188318.000000  188318.000000
mean         0.000250       0.012012       0.044266       0.003675
std          0.015796       0.108938       0.205685       0.060507
min          0.000000       0.000000       0.000000       0.000000
25%          0.000000       0.000000       0.000000       0.000000
50%          0.000000       0.000000       0.000000       0.000000
75%          0.000000       0.000000       0.000000       0.000000
max          1.000000       1.000000       1.000000       1.000000

                cat68          cat69          cat70          cat71  \
```

count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.000754	0.001630	0.000122	0.051360
std	0.027450	0.040343	0.011051	0.220731
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000

	cat72	cat73	cat74	cat75	\
count	188318.000000	188318.000000	188318.000000	188318.000000	
mean	0.371690	0.180912	0.019186	0.180609	
std	0.483258	0.385305	0.138180	0.384709	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	
75%	1.000000	0.000000	0.000000	0.000000	
max	1.000000	2.000000	2.000000	2.000000	

	cat76	cat77	cat78	cat79	\
count	188318.000000	188318.000000	188318.000000	188318.000000	
mean	0.041202	2.993251	1.003053	1.254453	
std	0.218799	0.109850	0.123392	0.740161	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	3.000000	1.000000	1.000000	
50%	0.000000	3.000000	1.000000	1.000000	
75%	0.000000	3.000000	1.000000	1.000000	
max	2.000000	3.000000	3.000000	3.000000	

	cat80	cat81	cat82	cat83	\
count	188318.000000	188318.000000	188318.000000	188318.000000	
mean	2.474734	2.683296	1.111211	1.055736	
std	0.876677	0.705550	0.709766	0.704867	
min	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	3.000000	1.000000	1.000000	
50%	3.000000	3.000000	1.000000	1.000000	
75%	3.000000	3.000000	1.000000	1.000000	
max	3.000000	3.000000	3.000000	3.000000	

	cat84	cat85	cat86	cat87	\
count	188318.000000	188318.000000	188318.000000	188318.000000	
mean	1.703517	1.006643	1.817102	1.167106	
std	0.747330	0.142922	0.968104	0.521547	
min	0.000000	0.000000	0.000000	0.000000	

25%	2.000000	1.000000	1.000000	1.000000
50%	2.000000	1.000000	1.000000	1.000000
75%	2.000000	1.000000	3.000000	1.000000
max	3.000000	3.000000	3.000000	3.000000

	cat88	cat89	cat90	cat91 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.206353	0.025999	0.059670	1.173356
std	0.609435	0.172455	0.259225	2.047520
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	1.000000
max	3.000000	7.000000	6.000000	7.000000

	cat92	cat93	cat94	cat95 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	1.675092	2.794911	2.372774	2.567009
std	2.356841	0.443348	0.908014	0.742076
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	3.000000	1.000000	2.000000
50%	0.000000	3.000000	3.000000	3.000000
75%	5.000000	3.000000	3.000000	3.000000
max	6.000000	4.000000	6.000000	4.000000

	cat96	cat97	cat98	cat99 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	3.940144	2.437154	1.254001	12.464512
std	0.492229	1.778027	1.474147	3.384187
min	0.000000	0.000000	0.000000	0.000000
25%	4.000000	2.000000	0.000000	12.000000
50%	4.000000	2.000000	0.000000	12.000000
75%	4.000000	4.000000	3.000000	15.000000
max	7.000000	6.000000	4.000000	15.000000

	cat100	cat101	cat102	cat103 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	7.455777	2.555072	0.097989	0.645376
std	3.329916	3.831894	0.435820	1.138599
min	0.000000	0.000000	0.000000	0.000000
25%	5.000000	0.000000	0.000000	0.000000
50%	8.000000	0.000000	0.000000	0.000000
75%	10.000000	5.000000	0.000000	1.000000
max	14.000000	18.000000	8.000000	12.000000

	cat104	cat105	cat106	cat107 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	5.568868	4.776952	6.651515	6.982747
std	2.312084	1.167578	1.757116	2.191122
min	0.000000	0.000000	0.000000	0.000000
25%	4.000000	4.000000	5.000000	5.000000
50%	5.000000	5.000000	6.000000	7.000000
75%	7.000000	5.000000	8.000000	9.000000
max	16.000000	19.000000	16.000000	19.000000

	cat108	cat109	cat110	cat111 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	4.513567	30.811691	73.723218	1.115406
std	3.638820	12.590811	29.367421	1.967299
min	0.000000	0.000000	0.000000	0.000000
25%	1.000000	33.000000	60.000000	0.000000
50%	3.000000	33.000000	67.000000	0.000000
75%	8.000000	33.000000	102.000000	2.000000
max	10.000000	83.000000	130.000000	15.000000

	cat112	cat113	cat114	cat115 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	23.923295	31.245346	1.451879	11.805690
std	12.955761	19.392185	2.674327	2.460944
min	0.000000	0.000000	0.000000	0.000000
25%	11.000000	12.000000	0.000000	10.000000
50%	23.000000	38.000000	0.000000	11.000000
75%	35.000000	48.000000	2.000000	14.000000
max	50.000000	60.000000	18.000000	22.000000

	cat116	cont1	cont2	cont3 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	152.282347	0.493861	0.507188	0.498918
std	73.961441	0.187640	0.207202	0.202105
min	0.000000	0.000016	0.001149	0.002634
25%	79.000000	0.346090	0.358319	0.336963
50%	161.000000	0.475784	0.555782	0.527991
75%	191.000000	0.623912	0.681761	0.634224
max	325.000000	0.984975	0.862654	0.944251

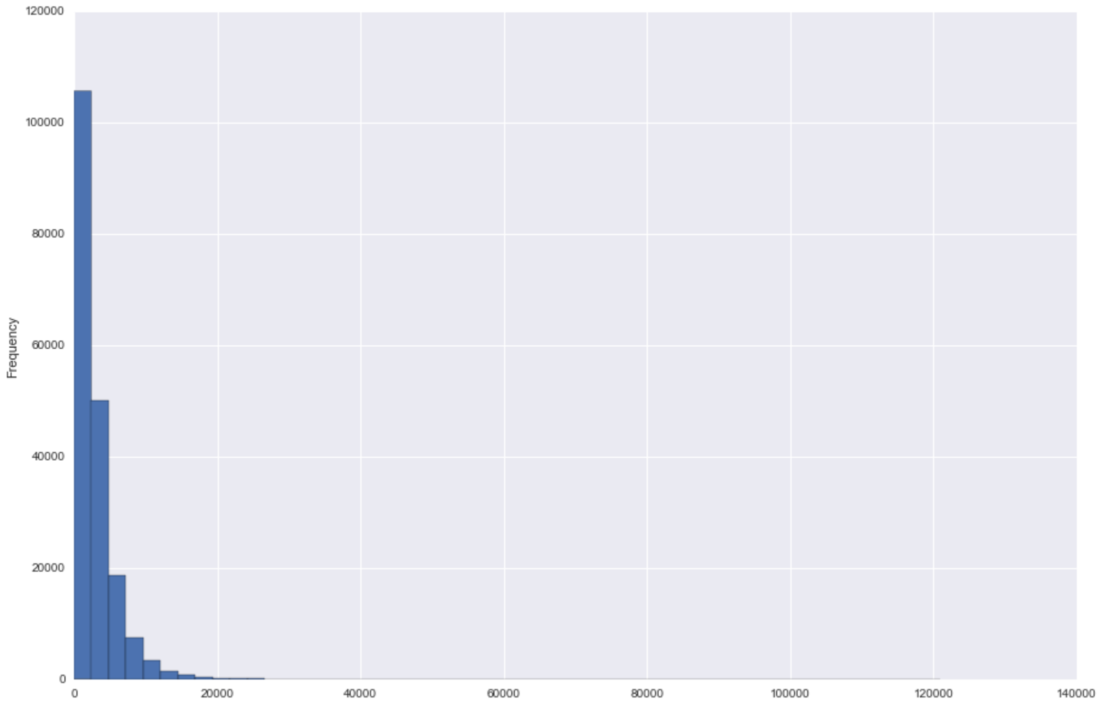
	cont4	cont5	cont6	cont7 \
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.491812	0.487428	0.490945	0.484970

std	0.211292	0.209027	0.205273	0.178450
min	0.176921	0.281143	0.012683	0.069503
25%	0.327354	0.281143	0.336105	0.350175
50%	0.452887	0.422268	0.440945	0.438285
75%	0.652072	0.643315	0.655021	0.591045
max	0.954297	0.983674	0.997162	1.000000

	cont8	cont9	cont10	cont11	\
count	188318.000000	188318.000000	188318.000000	188318.000000	
mean	0.486437	0.485506	0.498066	0.493511	
std	0.199370	0.181660	0.185877	0.209737	
min	0.236880	0.000080	0.000000	0.035321	
25%	0.312800	0.358970	0.364580	0.310961	
50%	0.441060	0.441450	0.461190	0.457203	
75%	0.623580	0.566820	0.614590	0.678924	
max	0.980200	0.995400	0.994980	0.998742	

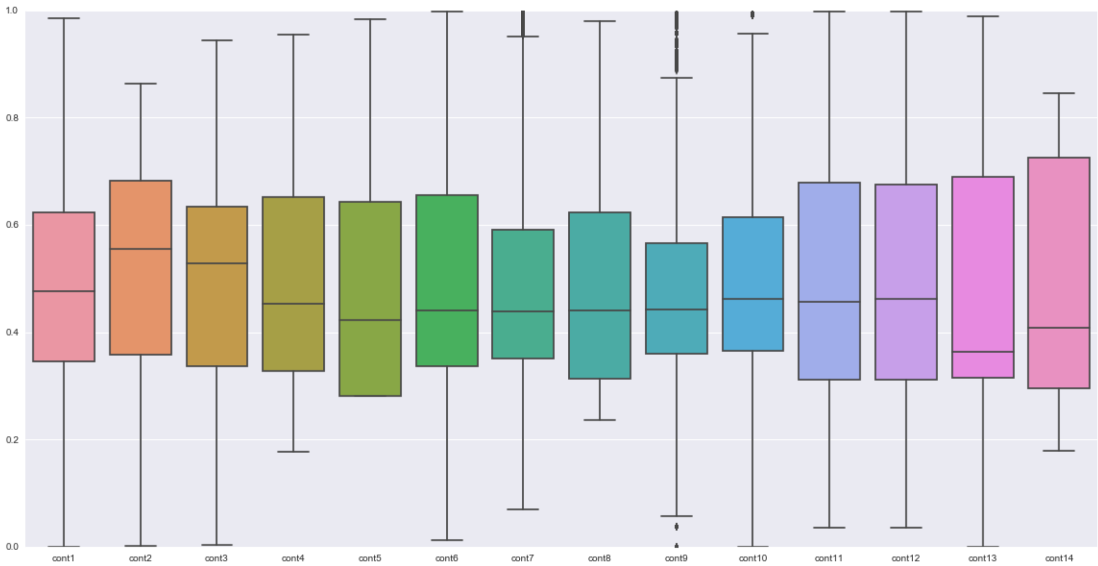
	cont12	cont13	cont14	loss
count	188318.000000	188318.000000	188318.000000	188318.000000
mean	0.493150	0.493138	0.495717	3037.337686
std	0.209427	0.212777	0.222488	2904.086186
min	0.036232	0.000228	0.179722	0.670000
25%	0.311661	0.315758	0.294610	1204.460000
50%	0.462286	0.363547	0.407403	2115.570000
75%	0.675759	0.689974	0.724623	3864.045000
max	0.998484	0.988494	0.844848	121012.250000

Plot target value:



The target data 'loss' looks significant right skewed. Would do log() transformation to see if it could be better.

Plot continuous feature:



Check how many different categorical value in each categorical variable

cat1	2	cat4	2	cat7	2	cat10	2
cat2	2	cat5	2	cat8	2	cat11	2
cat3	2	cat6	2	cat9	2	cat12	2

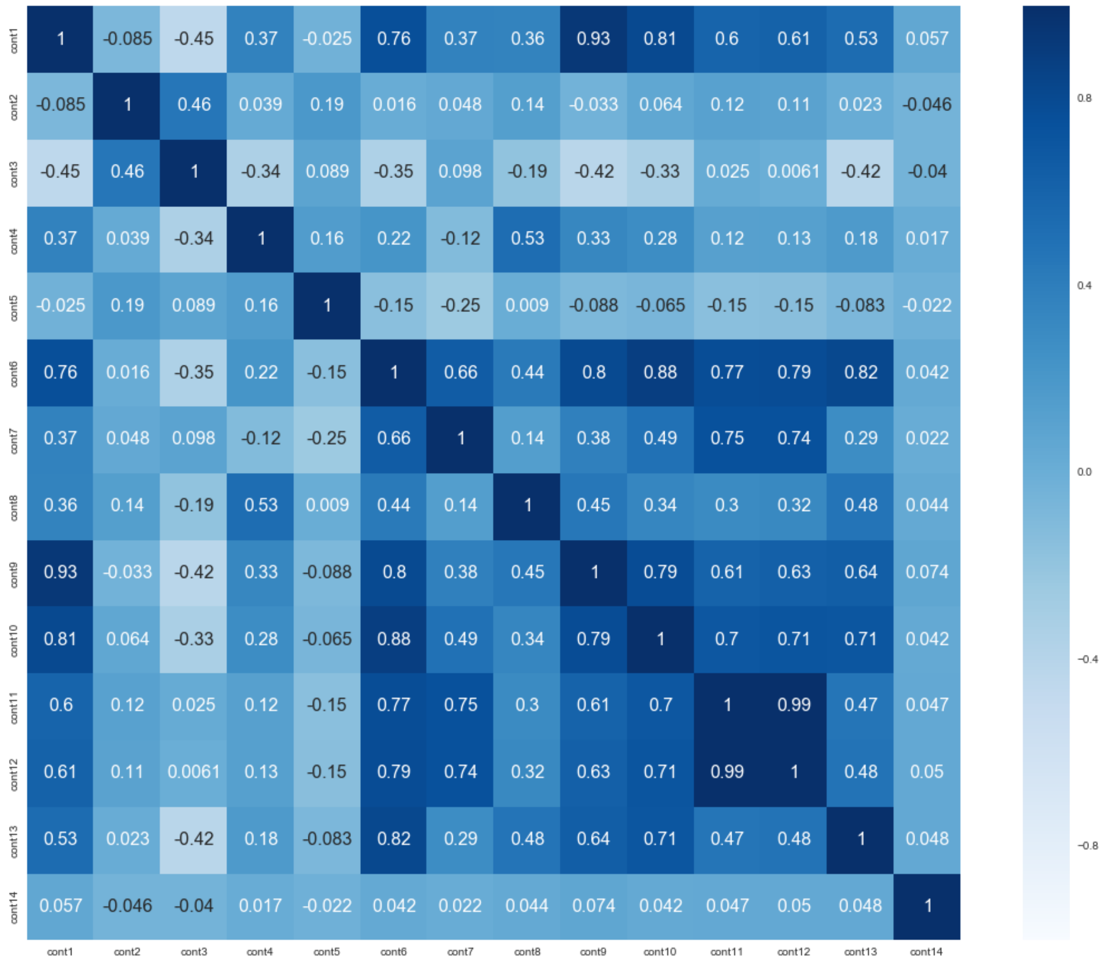
cat13	2	cat39	2	cat65	2	cat91	8
cat14	2	cat40	2	cat66	2	cat92	7
cat15	2	cat41	2	cat67	2	cat93	5
cat16	2	cat42	2	cat68	2	cat94	7
cat17	2	cat43	2	cat69	2	cat95	5
cat18	2	cat44	2	cat70	2	cat96	8
cat19	2	cat45	2	cat71	2	cat97	7
cat20	2	cat46	2	cat72	2	cat98	5
cat21	2	cat47	2	cat73	3	cat99	16
cat22	2	cat48	2	cat74	3	cat100	15
cat23	2	cat49	2	cat75	3	cat101	19
cat24	2	cat50	2	cat76	3	cat102	9
cat25	2	cat51	2	cat77	4	cat103	13
cat26	2	cat52	2	cat78	4	cat104	17
cat27	2	cat53	2	cat79	4	cat105	20
cat28	2	cat54	2	cat80	4	cat106	17
cat29	2	cat55	2	cat81	4	cat107	20
cat30	2	cat56	2	cat82	4	cat108	11
cat31	2	cat57	2	cat83	4	cat109	84
cat32	2	cat58	2	cat84	4	cat110	131
cat33	2	cat59	2	cat85	4	cat111	16
cat34	2	cat60	2	cat86	4	cat112	51
cat35	2	cat61	2	cat87	4	cat113	61
cat36	2	cat62	2	cat88	4	cat114	19
cat37	2	cat63	2	cat89	8	cat115	23
cat38	2	cat64	2	cat90	7	cat116	326

Data cleaning:

After checking, the data is clean. No missing value. Some extremely data, but since the data set is right skewed, would not treat them as outlier.

Data integration:

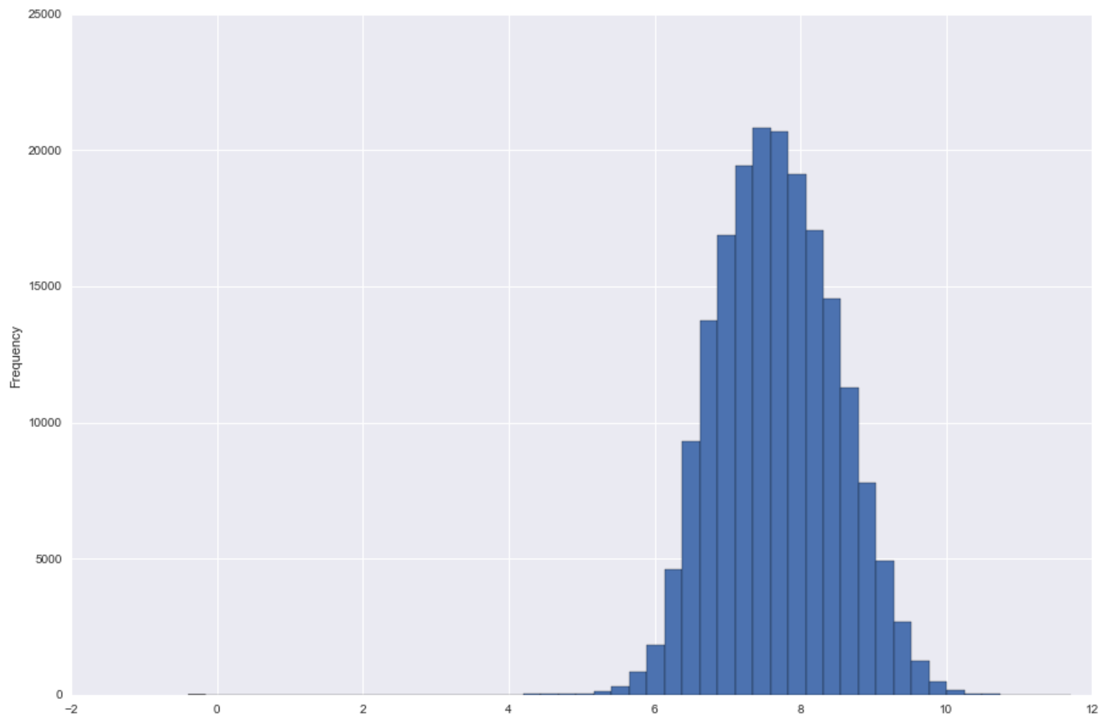
Heat map of the correlation between continuous features:



Data transformation:

Transformation:

Log() transform target value ‘loss’:



Looks much normal distributed. So, would use the loss_log to do data mining. Would transfer the data back to 'loss' value while prediction.

Normalization:

Use preprocessing.MinMaxScaler() from sklearn package to normalize data to [0, 1]

Split data:

Use cross_validation .train_test_split () to split data into 80% training set and 20% testing set.

Dimensionality reduction:

Since the target of this data set is to predict the 'loss' value, which is 120,000 different continuous data. The unsupervised learning algorithms such as Kmeans Clustering and Hierarchical Clustering are not appropriate.

However, CSC478 final project require the usage of unsupervised learning algorithms, so would use PCA (unsupervised learning) to do dimensionality reduction.

Use PCA to catch at least 95% information. Which can cut model computation time. Then compare the result by runtime and accuracy changing.

The PCA preprocessed data would only be used for long run time models in this project, such as KNN.

Use decomposition.PCA(n_components=60) function from sklearn package, print (pca.explained_variance_ratio_):

[0.13 0.09 0.08 0.06 0.05 0.03 0.03 0.03 0.03 0.03 0.02 0.02
0.02 0.02 0.02 0.02 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
sum(pca.explained_variance_ratio_)
0.95405298740329614

So, 60 components is what we need. Which reduced 70 features.

Pattern discovery & evaluation:

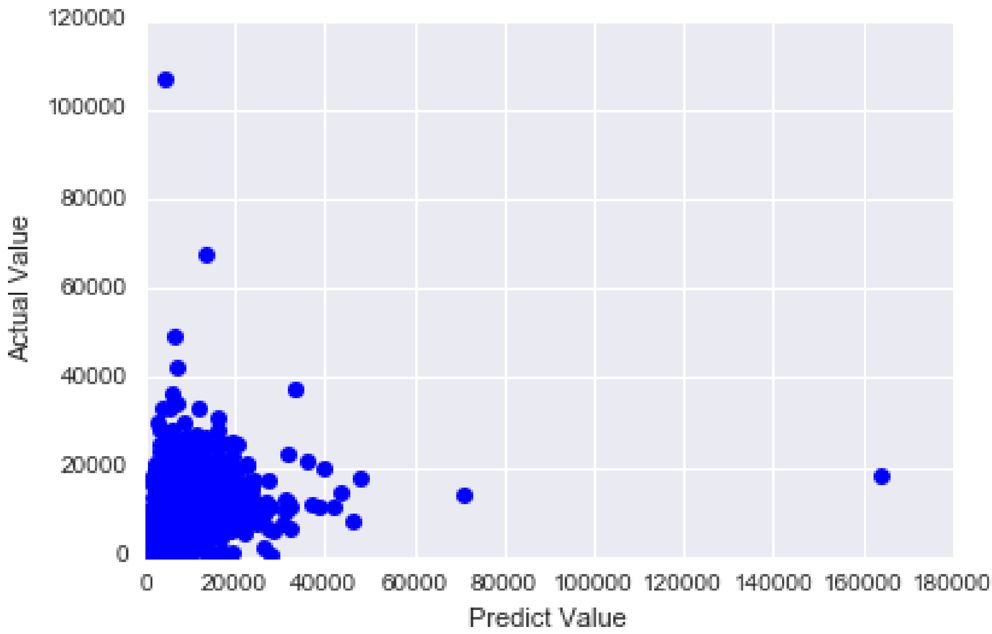
Algorithms would be tested:
Because what we are trying to predict is a continuous value 'loss' (120K different value), so, association rules and unsupervised learning methods such as K-means, clustering would not be appropriate. Would try:
Regression models: Linear Regression, Ridge Regression, Stochastic Gradient Descent Regression;
Supervised learning: KNN, Decision Tree;
Random Forest.

In this project, cross validation plus MAE and MAPE would be used to value the performance.
Cross validation: would set 20% testing and 80% training.
MAE: mean_absolute_error(y, p) function from sklearn package.
MAPE: mean_absolute_percentage_error(y, p) function code:

```
def mean_absolute_percentage_error(y_true, y_pred):
    #in this project y_true is always > 0
        return np.mean(np.abs((y_true - y_pred) / y_true))
```

For each model, would start with looking for the best parameter for each single algorithm. Then compare best models from each different algorithm and pick one of the best for this project.

Linear Regression:
Use the LinearRegression() function in sklearn package:
After read the documentation, it looks like the default parameters is good enough. No need to do parameter selection.
Return best model with:
Mae=1287.7041325114733
Mape=0.54173169267283827



The mae and mape value looks fine. But the graph looks not good.

Ridge Regression

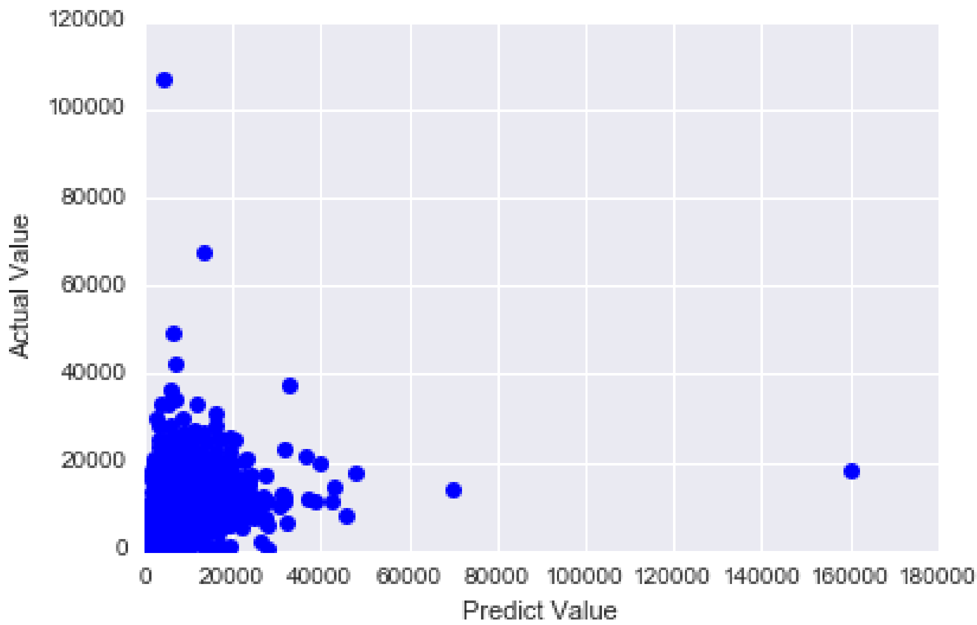
Use Ridge() function in sklearn package:

After read the documentation, it looks like the default parameters is good enough. No need to do parameter selection.

Return best model with:

mae=1287.365393293152

mape=0.54168809725527556



The mae and mape value looks fine. But the graph looks not good.

Stochastic Gradient Descent Regression

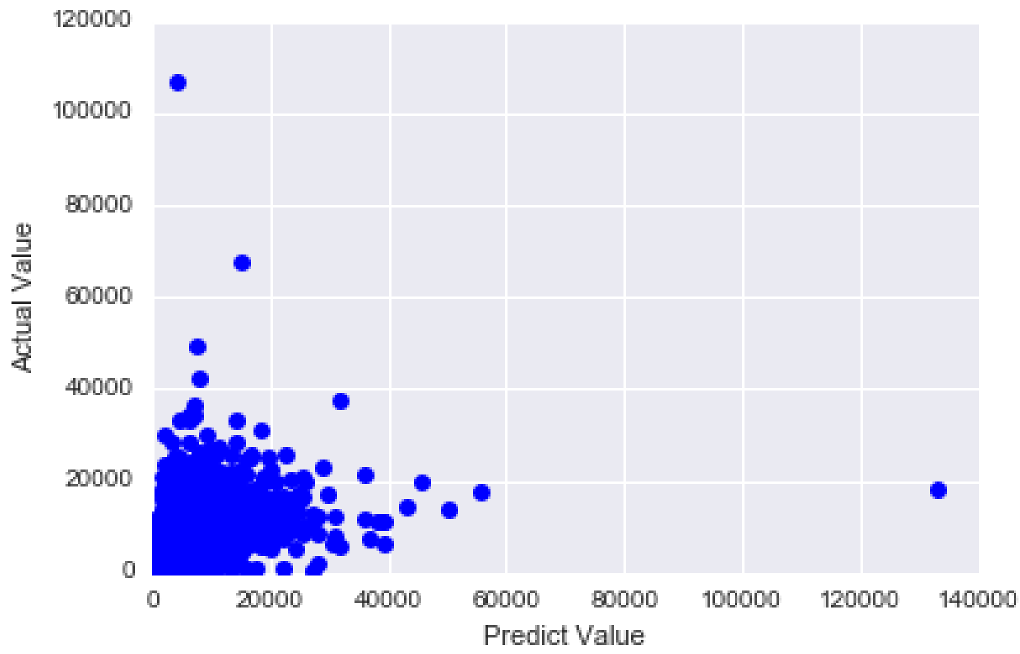
Use SGDRegressor() in sklearn package:

After read the documentation, it looks like the default parameters is good enough. No need to do parameter selection.

Return best model with:

mae= 1302.3575877161838

mape= 0.55744896931658261



The mae and mape value looks fine. But the graph looks not good.

K-Nearest Neighbors

Use KNeighborsRegressor() function in sklearn package.

Test different metric (distance calculation) and different value of k from 1 to 20.

Since the data set is real-valued vector spaces, so, compare the performance between “euclidean” and “manhattan”.

The default value for parameter: "algorithm" is 'auto' that means, it would decide the most appropriate algorithm from {‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’} based on the values passed to fit method. So, no need to test this parameter.

Euclidean:

K mae mape

[1, 1961.8216007327956, 0.8792028195629642]

[2, 1715.1688169813217, 0.71951935855864702]

[3, 1654.9532232922495, 0.67226622983947448]

[4, 1626.0270152072151, 0.64999442763785598]

[5, 1613.1027556977399, 0.63930503915695858]

[6, 1607.9590843728054, 0.63366112748612968]

[7, 1603.6703337887382, 0.62923861801563197]
[8, 1602.5323619362212, 0.62562677238938558]
[9, 1602.1657916465081, 0.62438264846056413]
[10, 1603.5990070081309, 0.62500790504071835]
[11, 1605.4575969202722, 0.62528286376779463]
[12, 1606.0064323709928, 0.62596085375559318]
[13, 1606.803374327189, 0.62562081730495145]
[14, 1607.7408912170715, 0.62506843120179068]
[15, 1609.5794514766724, 0.62630172624421376]
[16, 1611.1485531788544, 0.62778734750857612]
[17, 1612.0409481618717, 0.62850311132877568]
[18, 1612.834596872224, 0.62923079088320977]
[19, 1614.4683810592242, 0.62972882767816862]
[20, 1615.639187480185, 0.63023874274142566]

Manhattan:
K mae mape
[1, 1863.6908278462192, 0.8196024613876447]
[2, 1652.8845804557809, 0.66946736898460302]
[3, 1597.6203090170545, 0.62445210498541603]
[4, 1577.1060335583791, 0.60444732422211878]
[5, 1564.0756002760229, 0.59164711669827363]
[6, 1562.1914372164565, 0.58537461136486357]
[7, 1560.5518906639938, 0.58019867032152128]
[8, 1559.2065147541891, 0.57601818722048581]
[9, 1561.0436955983116, 0.57461790393069112]
[10, 1561.3304599828818, 0.57237203840871043]
[11, 1562.2564835264368, 0.57059392980075407]
[12, 1565.1078609124229, 0.57046290471040362]
[13, 1566.5540245156499, 0.56966741771460128]
[14, 1568.7743296125329, 0.57003367840415486]
[15, 1570.8623612768677, 0.57076973538591336]
[16, 1572.7774201061536, 0.57137087627839644]
[17, 1573.8069631910016, 0.57154206525539009]
[18, 1576.3402161965221, 0.57168541721781874]
[19, 1578.6220333021363, 0.57305884307933419]
[20, 1579.8877404253378, 0.57349374196806613]

The best result is K=13, with Manhattan metric:
Mae= 1566.5540245156499
Mape=0.5696674177146013

The computation time of KNN is extremely long due to data size and feature numbers. So, try the data set did dimensionality reduction by PCA, the best return is:

Mae=1763.67082785
Mape=0.729602461388
The runing time speed up because 70 features reduced by PCA. But the model performance is bad. So, skip it.

Decision Tree

Since the target value is numerical, so, use the DecisionTreeRegressor() function from sklearn package instead of DecisionTreeClassifier() function.
Test parameter:depth from 1 to 20, return:
Max_depth mae mape
[1, 1570.6702059189636, 0.70016044892353602]
[2, 1499.4049501555103, 0.66168102828864483]
[3, 1435.4122225643764, 0.63128451217930859]
[4, 1397.9644295121877, 0.61272751001393799]
[5, 1360.4769260559847, 0.59354597831916411]
[6, 1334.3000669397354, 0.58207709616775438]
[7, 1314.7256049131552, 0.57378559779945504]
[8, 1300.1133948363752, 0.56640017573956958]
[9, 1289.6021754801441, 0.55920380985414342]
[10, 1290.1546425796364, 0.5583077612361883]
[11, 1300.0075026294787, 0.559002543561539]
[12, 1315.5171276496299, 0.56220054939387887]
[13, 1331.4717976770448, 0.56637114732047933]
[14, 1355.9745003870512, 0.57711682266782138]
[15, 1380.8870591338966, 0.58764364086176812]
[16, 1415.8893215121866, 0.59979463468837713]
[17, 1445.4249715169617, 0.61489610487991975]
[18, 1471.1582450442654, 0.62571624330305953]
[19, 1506.3941279264438, 0.64329061698741863]
[20, 1534.8623602264361, 0.6582308710662943]

It is obvious that max depth=9 returns best model, where:
Mae=1289.6021754801441,
Mape=0.55920380985414342

It takes a little long time for computing during the DT model building and testing. Try the data after PCA dimensional reduction:
Max_depth mae mape
[1, 1675.4127880178369, 0.73846494735962398]
[2, 1603.1098305399671, 0.70800500018507373]
[3, 1563.2627832419087, 0.70194153093000089]
[4, 1542.3547405723641, 0.68406256252166076]
[5, 1529.9624941892598, 0.67179589176104471]

[6, 1516.2711405125874, 0.66008445970496232]
[7, 1520.0587482117235, 0.65490820706567199]
[8, 1547.0861198300061, 0.67232643216125521]
[9, 1554.9274125363277, 0.68136791082026149]
[10, 1590.9778325688769, 0.69632736593568734]
[11, 1607.6075478664709, 0.71050838504221181]
[12, 1645.2241419557724, 0.72623243025023521]
[13, 1715.9795435840463, 0.76092495686896133]
[14, 1769.6835229863427, 0.79448238950982142]
[15, 1841.9418017997318, 0.83588987201246889]
[16, 1903.9849280642691, 0.87222237482889253]
[17, 1966.7551643335084, 0.9149209506784598]
[18, 2008.3495514274741, 0.94872301622950383]
[19, 2053.5549826597385, 0.97306215651450079]
[20, 2089.8965811662097, 0.99866271347690527]

The runing time speed up a little, because 70 features reduced. But the model performance is bad. Skip it.

Random Forest

Use RandomForestRegressor() function from sklearn package.
For the parameter of max_depth. If in default, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. So, would use the default.
Test n_estimators from 1 to 20:

n_estimators mae mape
[1, 1749.8327787276025, 0.80236353946925176]
[2, 1491.6983208886022, 0.67335188209778496]
[3, 1394.916660427747, 0.6176617206936017]
[4, 1345.6868212275451, 0.58686355235795085]
[5, 1325.6857637128867, 0.57605406685793215]
[6, 1293.9493936619331, 0.56349947348499341]
[7, 1282.6985047743335, 0.55692106431326671]
[8, 1278.6665669663303, 0.55157899661506615]
[9, 1269.4458424906779, 0.54614671693271444]
[10, 1265.9494802483327, 0.54224562163806311]
[11, 1257.6144333312136, 0.54020975241808855]
[12, 1248.3893493409762, 0.53664082600415652]
[13, 1247.9037491256299, 0.53444312988804155]
[14, 1244.1715368960442, 0.53358832612233154]
[15, 1243.5722847958928, 0.53119426804317726]
[16, 1239.0329711179204, 0.5286470386850789]
[17, 1236.6766481852351, 0.52739429271817551]

[18, 1236.4390613558028, 0.52972090216887402]
[19, 1233.9685713054582, 0.52811476509234767]
[20, 1229.8765908445425, 0.52432590723270867]

The performance is increasing, probably 20 is not best parameter. Try 10 more:

n_estimators mae mape
[21, 1231.4266583048534, 0.52727125456836532]
[22, 1230.5100042365784, 0.5258795124362825]
[23, 1224.2627477190758, 0.52288869270359828]
[24, 1229.3528323217683, 0.52587807341869297]
[25, 1223.0556612115231, 0.52179373321281208]
[26, 1223.347187957688, 0.5225676322901176]
[27, 1223.2326526261631, 0.52026961935595972]
[28, 1225.2821738651137, 0.52267448315647147]
[29, 1221.8216515877675, 0.52221324622992293]
[30, 1221.2938413187214, 0.52190918201266601]

The run time become extremely long for n_estimators=21 to 30 (around 1 hour)
Consider both the time and performance, would use n_estimators=30 as the relatively best parameter.
So, n_estimators=30, return:
Mae=1221.2938413187214
Mape=0.52190918201266601

Model Selection:

Best models from each of the 6 algorithms:

Model name	mae	mape
['Linear Regression',	1287.7041325114733,	0.54173169267283827]
['Ridge Regression',	1287.365393293152,	0.54168809725527556]
['KNN_k=9_Manhattan',	1566.5540245156499,	0.5696674177146013]
['DecissionTree_depth=9',	1289.6021754801441,	0.55920380985414342]
['Stochastic Gradient Descent Regression',	1302.3575877161838,	0.55744896931658261]
['Random Forest_n=30',	1221.8216515877675,	0.52221324622992293]

Random Forest with n_estimators=30 is the best model. So, use it to do prediction.

Prediction:

predictions_rf=np.exp(rf_p.predict(X_test_n))

```
predictions_rf
array([ 1501.45,    1584.32,    8026.34, ...,    3112.6 ,    1522.28,    3000.93])
```

Write submissions to output file in the correct format (format provided by the competition)

```
l=len(predictions_rf)
with open("predict_loss.csv", "w") as subfile:
    subfile.write("id,loss\n")
    for i in range(0, l):
        subfile.write("%d,%f\n"%(id_test[i],predictions_rf[i]))
```

Conclusion:

The data set has no label, so, we can’t have some interpretation about the data itself.

But, still get from this project:

1. For data set with a lot tuple, KNN is very slow, much slower than decision tree and random forest. Regression algorithms are always fast.
2. PCA can reduce dimensionality, but on the other hand, the accuracy reduced. It is a trade-off, depending on different data set.
   In this project, even though the PCA caught 95% information, the model accuracy from full dimension is much better.
3. When n_estimators is very high, the run time of Random Forest increased quit a lot, though accuracy increased, too. It is another trade-off.
4. The mae=1221.8216515877675, mape=0.52221324622992293 look scary. But, since the data set is large (180K tuples), and the range of target value is wide (120K), the mae and mape of the best model is not bad.