In [1]:
```
# Make needed imports
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.tree import export_graphviz
from matplotlib import pyplot as plt
```

In [2]:
```
# Load train and testing data
training_data = pd.read_pickle("../data/train.pkl")
testing_data = pd.read_pickle("../data/test.pkl")
training_data
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	2013	59.9	268	66	0.01	73.219243	64	430	18.1	89	62	8.13	64	0.1	631.744976	31
7	2008	58.1	287	80	0.03	25.873925	64	1599	15.7	110	64	8.33	64	0.1	373.361116	2
9	2006	57.3	295	84	0.03	17.171518	64	1990	14.7	116	58	7.43	58	0.1	272.563770	2
8	2007	57.5	295	82	0.02	10.910156	63	1141	15.2	113	63	6.73	63	0.1	369.835796	26
10	2005	57.3	291	85	0.02	1.388648	66	1296	14.2	118	58	8.70	58	0.1	25.294130	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4	2010	52.4	527	29	5.21	53.308581	9	9696	29.4	44	89	5.37	89	15.7	713.635620	1
1	2013	58.0	399	25	6.39	10.666707	95	0	3.8	36	95	6.88	95	6.8	111.227396	
11	2003	44.5	715	26	4.06	0.000000	7	998	26.7	41	7	6.52	68	36.7	453.351155	12
14	2000	46.0	665	24	1.68	0.000000	79	1483	25.5	39	78	7.10	78	43.5	547.358878	12
0	2014	59.2	371	23	6.50	10.822595	91	0	31.3	34	92	6.44	91	6.3	127.474620	15

1286 rows × 20 columns

In [3]:
```
testing_data
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
14	2001	55.3	316	88	0.01	10.574728	63	8762	12.6	122	35	7.80	33	0.1	117.496980	
12	2003	56.7	295	87	0.01	11.089053	65	798	13.4	122	41	8.82	41	0.1	198.728544	
13	2002	56.2	3	88	0.01	16.887351	64	2486	13.0	122	36	7.76	36	0.1	187.845950	2
3	2012	59.5	272	69	0.01	78.184215	67	2787	17.6	93	67	8.52	67	0.1	669.959000	
14	2001	73.6	14	1	4.25	96.205571	96	18	46.0	1	97	6.00	97	0.1	1326.973390	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7	2007	52.6	487	32	2.08	10.851482	8	535	19.2	51	77	4.37	8	13.6	114.587985	1
8	2006	58.0	526	33	2.25	1.860004	81	459	18.8	52	83	6.11	81	15.9	13.154199	1
13	2001	45.3	686	25	1.72	0.000000	76	529	25.9	39	76	6.16	75	42.1	548.587312	1
12	2002	44.8	73	25	4.43	0.000000	73	304	26.3	40	73	6.53	71	39.8	57.348340	
10	2004	44.3	723	27	4.36	0.000000	68	31	27.1	42	67	7.13	65	33.6	454.366654	1

362 rows × 20 columns

In [4]:
```
# Seperate training X and Y values
train_X = training_data.iloc[:, 2:20].values
train_Y = training_data.iloc[:, 1].values
train_Y
```

Out[4]:
```
array([59.9, 58.1, 57.3, ..., 44.5, 46. , 59.2])
```

In [5]:
```
train_X
```

Out[5]:
```
array([[2.68e+02, 6.60e+01, 1.00e-02, ..., 1.77e+01, 4.70e-01, 9.90e+00],
       [2.87e+02, 8.00e+01, 3.00e-02, ..., 1.89e+01, 4.33e-01, 8.70e+00],
       [2.95e+02, 8.40e+01, 3.00e-02, ..., 1.93e+01, 4.05e-01, 8.10e+00],
       ...,
       [7.15e+02, 2.60e+01, 4.06e+00, ..., 9.90e+00, 4.18e-01, 9.50e+00],
       [6.65e+02, 2.40e+01, 1.68e+00, ..., 1.12e+01, 4.34e-01, 9.80e+00],
       [3.71e+02, 2.30e+01, 6.50e+00, ..., 5.70e+00, 4.98e-01, 1.03e+01]])
```

In [6]:
```
#Train Model
regressor = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 150, boots
# Fit the regressor with x and y data
regressor.fit(train_X, train_Y)
# Load testing values
test_X = testing_data.iloc[:, 2:20].values
test_Y = testing_data.iloc[:, 1].values
# Predict with testing X
predict_Y = regressor.predict(test_X)
print(predict_Y)
```

```
[56.94654763 57.39006534 66.45553903 59.31395733 72.9510977  72.80782217
 73.02570141 76.66313678 71.93709643 72.43032699 73.16092398 53.0918471
 53.33952251 74.95288238 76.63650269 75.59305686 72.84305807 72.77633938
 72.88778394 83.45164543 82.89503479 81.90825147 79.35323869 80.1164286
 79.42648596 71.46704925 71.58042675 71.08205365 66.61530507 67.6825275
 68.12547379 67.76638076 67.74839086 67.99479132 81.03164453 79.46593857
 82.69550056 69.39161949 69.31007939 69.67971526 56.58409878 56.56013779
 56.99724127 67.26498278 62.02995926 63.12416323 71.08273131 72.49884281
 73.08605632 48.40895464 48.64431607 52.81070423 72.3216005  72.69180832
 73.14323632 73.49060563 73.29860152 73.12132457 57.81867438 56.87602762
 55.16225822 55.060093   54.22394589 72.39349157 71.48990238 71.61093131
 66.74229274 64.89649905 55.44503764 55.51563464 81.52813003 82.46550051
 82.96959215 51.32929519 51.874548   52.1535728  53.90606015 79.24362195
 78.42739156 71.9794643  72.07201486 72.27877981 72.51382281 72.63840036
 73.2929391  61.78639924 61.25054634 61.99113843 74.33068674 75.36002045
 76.54964466 76.88288832 77.0225645  77.93479076 78.37929613 79.45913059
 61.86979383 61.68833355 70.48377886 70.4356766  68.9259415  73.26690005
 73.05863353 73.46787954 68.97344658 70.94527287 70.19260168 57.80027398
 57.08814713 74.26875871 74.69401395 74.88020782 58.43248444 58.49508289
 68.37021419 69.00524649 68.43633574 80.4841023  80.87595847 81.62147723
 61.48991989 61.1641857  72.91442281 72.8124369  73.32507098 81.3612001
 81.320547   80.79486405 56.69518821 58.48629216 58.85353653 78.27398675
 79.21868886 80.16904886 73.06182773 70.66303849 57.19088411 55.33093324
 57.14992979 57.55253371 65.61440992 65.77618445 65.82989949 62.59129593
 69.58337573 70.32961202 71.82739663 65.01232868 66.56724874 66.65686928
 67.08052913 67.20759265 67.39352989 69.07558791 66.44527089 66.86614362
 82.38829254 81.0935798  81.11467294 81.70533337 80.60533743 80.64146822
 83.39570485 72.45533312 73.29135225 73.34653004 73.51276652 73.69905035
 73.68477343 64.65639462 64.85391821 64.82166208 51.41057172 52.31812508
 53.56166314 65.99578041 66.21008573 67.5295871  69.36716647 69.64025166
 74.45373207 72.61832615 71.62204941 72.47174241 46.06125419 47.09725173
 47.20376029 60.0600379  59.2624594  72.76856424 72.69777546 72.06871995
 78.92888729 79.2093732  79.75897623 59.47469008 59.95305076 61.7343988
 47.25121028 48.11415837 49.21581777 73.3030986  73.40261408 74.02484989
 72.71745055 72.9254588  73.33207826 55.76947742 56.17954711 56.17154084
 77.83801298 78.55505167 79.51187963 62.74119053 63.3467729  71.54149998
 70.80105273 72.36090147 74.14733503 74.57523882 75.38405192 64.4028314
 64.96491612 64.75116604 75.52816002 74.75020323 68.5970533  68.83647599
 71.62891197 51.68166005 51.59348958 51.05577231 64.6584466  64.13085996
 64.80563019 63.30412859 65.66901167 66.42852103 67.95255682 81.80854988
 70.6905979  71.39876612 72.11893806 58.24998633 60.34890708 54.44946742
 51.77687602 65.35324388 66.28282621 65.6679873  74.68976668 74.80763145
 75.39147526 59.93213716 60.36044988 60.83300724 72.69043155 72.86009349
 73.02573793 73.59447351 73.83034134 73.9346203  67.60251258 67.75563361
 67.6419858  74.06693365 74.38884748 74.61765858 77.6303933  77.76713097
```

```
78.17269644 71.78461462 71.63977396 72.50258042 66.18222759 66.32467043
66.56156442 55.48825341 55.81539797 55.83289336 73.60690762 73.84384758
73.87668576 63.04256498 63.24218649 64.80403742 60.27901041 62.05290268
62.26192555 74.35059951 74.18215827 74.13763114 71.10028549 71.45956195
71.36156532 52.76697958 52.31063987 66.15030839 67.29220543 67.9853075
56.6531422  55.80510107 54.28555105 80.24801344 81.34852413 82.19788113
69.8840592  73.9675978  73.89051437 70.53263453 69.46430985 47.67816762
47.60916624 47.54313592 81.95524168 72.5842563  72.67289003 66.79694655
67.28340463 66.84544579 69.65035298 69.80756804 70.45770279 67.87464881
67.22418073 59.70707764 58.24247231 72.6754571  72.68266982 72.92911277
69.95975969 71.13462269 70.74136543 73.30589498 72.43155534 73.46212725
72.99242955 73.41002394 73.01292105 64.93539426 65.07380533 63.69628112
54.32041351 53.4105826  53.83161664 67.46908523 67.63890202 67.70669473
75.94030938 75.78591322 76.49041061 67.67157209 67.81345155 67.98260802
67.89761044 68.36444863 69.11568759 51.548006   51.07146079 47.3637858
48.50797577 45.52866383]
```

In [7]:
```
test_X
```

Out[7]:
```
array([[3.16e+02, 8.80e+01, 1.00e-02, ..., 2.40e+00, 3.40e-01, 5.90e+00],
       [2.95e+02, 8.70e+01, 1.00e-02, ..., 1.99e+01, 3.73e-01, 6.50e+00],
       [3.00e+00, 8.80e+01, 1.00e-02, ..., 2.20e+00, 3.41e-01, 6.20e+00],
       ...,
       [6.86e+02, 2.50e+01, 1.72e+00, ..., 1.70e+00, 4.27e-01, 9.80e+00],
       [7.30e+01, 2.50e+01, 4.43e+00, ..., 1.30e+00, 4.27e-01, 1.00e+01],
       [7.23e+02, 2.70e+01, 4.36e+00, ..., 9.40e+00, 4.07e-01, 9.20e+00]])
```

In [8]:
```
test_Y
```

Out[8]:
```
array([55.3, 56.7, 56.2, 59.5, 73.6, 72.8, 73.3, 76.9, 72.9, 73.4, 73.8,
       49.6, 49.1, 74.1, 74.7, 74.9, 72.6, 72.6, 73. , 79.9, 86. , 81. ,
       78.6, 78.7, 79.3, 67.8, 68.4, 68.4, 67.3, 67.8, 68.2, 67.7, 67.2,
       68.2, 78. , 78. , 78.8, 68.2, 68.5, 68.7, 55.8, 56.1, 56.5, 61.7,
       62.5, 64.2, 75. , 75.7, 75.4, 46.7, 46. , 48.1, 71. , 71.4, 72. ,
       71.6, 71.8, 72.2, 57.5, 56.1, 53.4, 54.1, 54.8, 71.1, 71.4, 71.8,
       66.6, 65.6, 53.6, 53.3, 80. , 81. , 85. , 49.2, 53. , 51.8, 57. ,
       79.1, 79.6, 72.2, 72.7, 73.5, 71.5, 71.8, 72.8, 59.8, 60. , 63. ,
       77.5, 78.3, 77.7, 76.6, 76.3, 78.2, 78.4, 78.6, 61.3, 69. , 71.2,
       71.4, 69.3, 73.4, 73.6, 74.4, 68.9, 73. , 70. , 59.1, 58.8, 72.8,
       73. , 73. , 61.8, 68. , 67.8, 67.9, 68.1, 79. , 79.2, 82. , 61.6,
       61.4, 73. , 71.7, 72.3, 78.3, 78.4, 79.1, 57.9, 58.3, 58.9, 78.7,
       79. , 79.2, 75. , 69.7, 57.8, 57.3, 56.7, 57.6, 65.3, 65.1, 65. ,
       62.7, 71.3, 71.6, 72.2, 64.4, 64.8, 65.2, 66.5, 66.7, 65.3, 66.8,
       64.7, 65.9, 84. , 79.3, 79.3, 81. , 79.8, 80. , 89. , 73.3, 73.5,
       74. , 71.9, 72.1, 72.5, 64.4, 64.7, 64.7, 52.4, 53. , 54.1, 64.3,
       64.6, 64.8, 69.9, 73. , 71. , 73. , 73.2, 73.7, 44.8, 44.5, 45.3,
       67. , 59.2, 71.2, 71.4, 71.6, 78. , 78.3, 78.7, 59.9, 64. , 69. ,
       44.6, 45.1, 46. , 72.7, 72.9, 73.2, 78. , 71.8, 73.4, 52.8, 53.6,
       54.3, 78.7, 79. , 79.3, 61.2, 69. , 71.5, 71.5, 71.9, 75. , 75. ,
       75.4, 63.2, 63.8, 64. , 75.3, 74.6, 69. , 69.5, 72. , 49.8, 54. ,
       58. , 63.5, 63.9, 64.2, 64.3, 64.7, 65.4, 66. , 81.1, 73. , 75. ,
       71. , 58.2, 63. , 55. , 49.8, 63.7, 62.9, 64.2, 75.7, 75.8, 75.8,
       59.1, 59.3, 59.9, 71.9, 72.1, 72.3, 72.2, 72.8, 74.2, 66.8, 66.8,
       67.3, 74.2, 74.5, 74.9, 76.9, 77.2, 78. , 78. , 77. , 71.7, 64.8,
       64.9, 65. , 52. , 53.4, 55.3, 75. , 76. , 71.4, 63.8, 64.3, 64.7,
       65. , 61.3, 62.1, 73. , 73.6, 73.8, 72. , 72.1, 72.1, 48.1, 47.1,
       66.2, 66.5, 67.1, 56. , 54.9, 53.7, 79.4, 79.5, 81. , 69.1, 74.2,
       73.8, 69.5, 69.3, 47.1, 46.4, 45.6, 81.7, 73. , 72.8, 65.2, 65.9,
       65.5, 71.2, 71.4, 71.6, 67.4, 66.6, 58.9, 56.7, 71.8, 71.9, 72.2,
       69.3, 69.4, 69.5, 73.2, 73.5, 74. , 78. , 71.2, 72. , 63.4, 63.5,
       63.3, 51. , 51.3, 53.2, 67.7, 67.6, 67.4, 75.2, 75.4, 75.4, 67.2,
       67.8, 67.3, 69.1, 69.3, 69.6, 52.6, 58. , 45.3, 44.8, 44.3])
```

In [9]:
```
#basline estimate
# The baseline predictions are the historical averages
baseline_preds = testing_data.iloc[:, 8:9].values
# Baseline errors, and display average baseline error
baseline_errors = abs(baseline_preds - test_Y)
print('Average baseline error: ', round(np.mean(baseline_errors), 2))
```

```
Average baseline error:  32.38
```

In [10]:
```
# Use the forest's predict method on the test data
predictions = regressor.predict(test_X)
# Calculate the absolute errors
errors = abs(predictions - test_Y)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 1.39 degrees.
```

In [11]:
```
#Determine Performance Metrics
# Calculate mean absolute percentage error
feature = 100 * (errors / predict_Y)
# Calculate and display accuracy
accuracy = 100 - np.mean(feature)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 97.85 %.
```

In [12]:
```
# Get numerical feature importances
importances = list(regressor.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance ir
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse =
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_import
```

```
Variable:                   16 Importance: 0.58
Variable:                   11 Importance: 0.23
Variable:                    0 Importance: 0.11
Variable:                    2 Importance: 0.01
Variable:                    6 Importance: 0.01
Variable:                    9 Importance: 0.01
Variable:                   14 Importance: 0.01
Variable:                   15 Importance: 0.01
Variable:                   17 Importance: 0.01
Variable:                    1 Importance: 0.0
Variable:                    3 Importance: 0.0
Variable:                    4 Importance: 0.0
Variable:                    5 Importance: 0.0
Variable:                    7 Importance: 0.0
Variable:                    8 Importance: 0.0
Variable:                   10 Importance: 0.0
Variable:                   12 Importance: 0.0
Variable:                   13 Importance: 0.0
```

In [13]:
```
train_important = training_data.iloc[:, 11:17].values
test_important = testing_data.iloc[:, 11:17].values
updated_testing = testing_data.iloc[:, 11:17]
```

In [14]:
```
# Get numerical feature importances
importances = list(regressor.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance ir
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse =
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_import
```

```
Variable:                   11 Importance: 0.11
Variable:                   13 Importance: 0.01
Variable:                   12 Importance: 0.0
Variable:                   14 Importance: 0.0
Variable:                   15 Importance: 0.0
Variable:                   16 Importance: 0.0
```

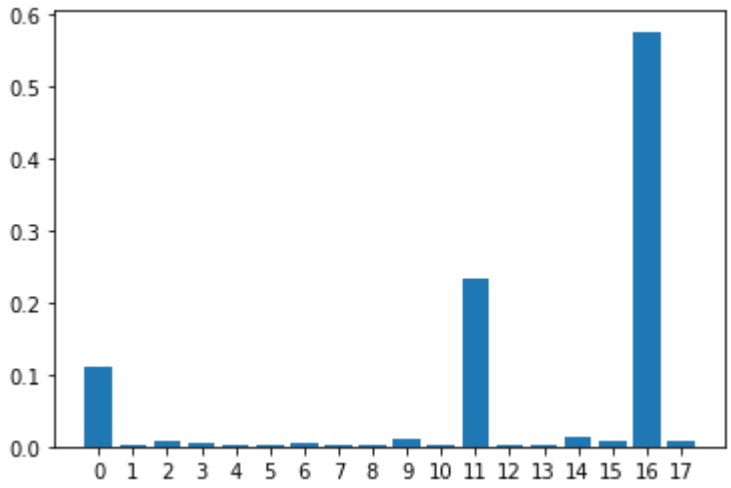
In [15]:
```
# New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 15
# Train the random forest
rf_most_important.fit(train_important, train_Y)
# Make predictions and determine the error
predictions = rf_most_important.predict(test_important)
errors = abs(predictions - test_Y)
# Display the performance metrics
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
mape = np.mean(100 * (errors / test_Y))
accuracy = 100 - mape
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Mean Absolute Error: 2.24 degrees.
Accuracy: 96.54 %.
```

In [16]:
```
import matplotlib.pyplot as plt

x_value = list(range(len(importances)))
print(x_value)

plt.bar(x_value, importances, orientation = 'vertical')

plt.xticks(x_value, testing_data, rotation = 6)

plt.ylabel('importance');
plt.xlable('Variable');
plt.title('Variable Importance');
plt.show()
```

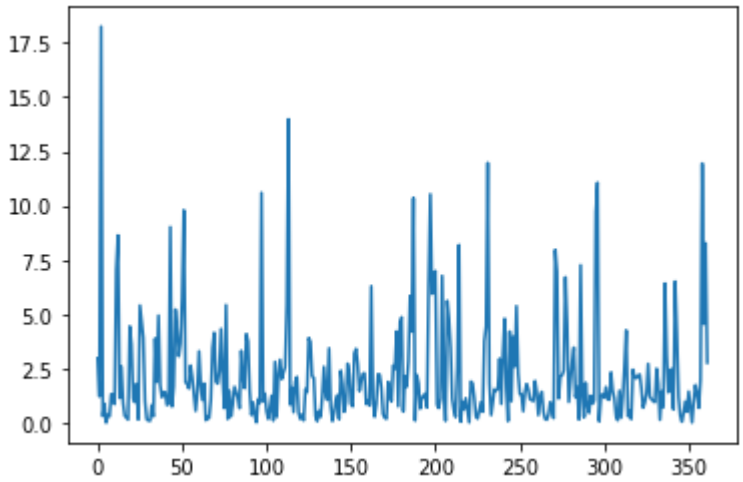
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_44500/1242508121.py in <module>
      6 plt.bar(x_value, importances, orientation = 'vertical')
      7
----> 8 plt.xticks(x_value, testing_data, rotation = 6)
      9
     10 plt.ylabel('importance');

~\anaconda3\lib\site-packages\matplotlib\pyplot.py in xticks(ticks, labels, **k
wargs)
   1812         labels = ax.get_xticklabels()
   1813     else:
-> 1814         labels = ax.set_xticklabels(labels, **kwargs)
   1815     for l in labels:
   1816         l.update(kwargs)

~\anaconda3\lib\site-packages\matplotlib\axes\_base.py in wrapper(self, *args,
 **kwargs)
     71
     72         def wrapper(self, *args, **kwargs):
---> 73             return get_method(self)(*args, **kwargs)
     74
     75         wrapper.__module__ = owner.__module__

~\anaconda3\lib\site-packages\matplotlib\_api\deprecation.py in wrapper(*args,
 **kwargs)
    469                 "parameter will become keyword-only %(removal)s.",
    470                 name=name, obj_type=f"parameter of {func.__name__}()")
--> 471         return func(*args, **kwargs)
    472
    473     return wrapper

~\anaconda3\lib\site-packages\matplotlib\axis.py in _set_ticklabels(self, label
s, fontdict, minor, **kwargs)
   1793         if fontdict is not None:
   1794             kwargs.update(fontdict)
-> 1795         return self.set_ticklabels(labels, minor=minor, **kwargs)
   1796
   1797     def set_ticks(self, ticks, *, minor=False):
```

```
~\anaconda3\lib\site-packages\matplotlib\axis.py in set_ticklabels(self, tickla
bels, minor, **kwargs)
   1714             # remove all tick labels, so only error for > 0 ticklabels
   1715             if len(locator.locs) != len(ticklabels) and len(ticklabels)
!= 0:
-> 1716                 raise ValueError(
   1717                     "The number of FixedLocator locations"
   1718                     f" ({len(locator.locs)}), usually from a call to"

ValueError: The number of FixedLocator locations (18), usually from a call to s
et_ticks, does not match the number of ticklabels (20).
```



In [17]:
```
losses = []
# calculate the percentage
for i in range(len(test_Y)):
    dif = (abs(float(test_Y[i] - predict_Y[i])) / float(test_Y[i])) * 100
    losses.append(dif)
# print(losses)
plt.plot(losses)
#calculate total value of loss
totalValue = np.sum(losses) % 100
print("Loss percentage on testing set: " + str(totalValue / len(losses)))
errors = mean_squared_error(test_Y, predict_Y)
print("MSE loss on testing set: " + str(mean_squared_error(test_Y, predict_Y)))
print("Random forest regressor score: " + str(regressor.score(test_X, test_Y)))
```

```
Loss percentage on testing set: 0.22182062282914328
MSE loss on testing set: 4.084487000249579
Random forest regressor score: 0.9513420018691462
```



In [ ]:
```
```