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

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

In [3]:
```
training_data
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9
0	0.866667	0.353333	0.372905	0.041250	0.000000	0.003861	0.639175	0.003271	0.214381	0.042381
1	0.533333	0.313333	0.399441	0.050000	0.001156	0.001365	0.639175	0.012165	0.182423	0.052381
2	0.400000	0.295556	0.410615	0.052500	0.001156	0.000906	0.639175	0.015140	0.169108	0.055238
3	0.466667	0.300000	0.410615	0.051250	0.000578	0.000575	0.628866	0.008681	0.175766	0.053810
4	0.333333	0.295556	0.405028	0.053125	0.000578	0.000073	0.659794	0.009860	0.162450	0.056190
...	...	...	...	...	...	...	...	...	...	...
1281	0.666667	0.186667	0.734637	0.018125	0.300578	0.002811	0.072165	0.073767	0.364847	0.020952
1282	0.866667	0.311111	0.555866	0.015625	0.368786	0.000563	0.958763	0.000000	0.023968	0.017143
1283	0.200000	0.011111	0.997207	0.016250	0.234104	0.000000	0.051546	0.007593	0.328895	0.019524
1284	0.000000	0.044444	0.927374	0.015000	0.096532	0.000000	0.793814	0.011283	0.312916	0.018571
1285	0.933333	0.337778	0.516760	0.014375	0.375145	0.000571	0.917526	0.000000	0.390146	0.016190

1286 rows × 20 columns

In [4]:
```
testing_data
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	
0	0.000000	0.246085	0.436288	0.058667	0.000000	0.000855	0.628866	0.098491	0.151515	0.0610	0.
1	0.166667	0.277405	0.407202	0.058000	0.000000	0.000896	0.649485	0.008970	0.163059	0.0610	0.
2	0.083333	0.266219	0.002770	0.058667	0.000000	0.001365	0.639175	0.027945	0.157287	0.0610	0.
3	0.916667	0.340045	0.375346	0.046000	0.000000	0.006319	0.670103	0.031328	0.223665	0.0465	0.
4	0.000000	0.655481	0.018006	0.000667	0.237402	0.007776	0.969072	0.000202	0.633478	0.0005	0.
...	...	...	...	...	...	...	...	...	...	...	
357	0.500000	0.185682	0.673130	0.021333	0.115901	0.000877	0.061856	0.006014	0.246753	0.0255	0.

	0	1	2	3	4	5	6	7	8	9	
358	0.416667	0.306488	0.727147	0.022000	0.125420	0.000150	0.814433	0.005160	0.240981	0.0260	0.
359	0.000000	0.022371	0.948753	0.016667	0.095745	0.000000	0.762887	0.005946	0.343434	0.0195	0.
360	0.083333	0.011186	0.099723	0.016667	0.247480	0.000000	0.731959	0.003417	0.349206	0.0200	0.
361	0.250000	0.000000	1.000000	0.018000	0.243561	0.000000	0.680412	0.000348	0.360750	0.0210	0.

362 rows × 20 columns

In [5]:

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

Out[5]:

```
array([0.35333333, 0.31333333, 0.29555556, ..., 0.01111111, 0.04444444,
       0.33777778])
```

In [6]:

```
# we do not include the year column
train_X
```

Out[6]:

```
array([[0.37290503, 0.04125   , 0.        , ..., 0.62633452, 0.50213675,
        0.34756098],
       [0.39944134, 0.05      , 0.00115607, ..., 0.66903915, 0.46260684,
        0.27439024],
       [0.41061453, 0.0525    , 0.00115607, ..., 0.68327402, 0.43269231,
        0.23780488],
       ...,
       [0.9972067 , 0.01625   , 0.23410405, ..., 0.34875445, 0.4465812 ,
        0.32317073],
       [0.9273743 , 0.015     , 0.09653179, ..., 0.39501779, 0.46367521,
        0.34146341],
       [0.51675978, 0.014375  , 0.37514451, ..., 0.19928826, 0.53205128,
        0.37195122]])
```

In [7]:

```
# Load testing values
test_X = testing_data.iloc[:, 2:20].values
test_Y = testing_data.iloc[:, 1].values
test_Y
```

Out[7]:

```
array([0.24608501, 0.27740492, 0.26621924, 0.34004474, 0.65548098,
       0.63758389, 0.64876957, 0.72930649, 0.63982103, 0.65100671,
       0.65995526, 0.11856823, 0.10738255, 0.66666667, 0.68008949,
       0.68456376, 0.63310962, 0.63310962, 0.64205817, 0.79642058,
       0.93288591, 0.82102908, 0.76733781, 0.76957494, 0.78299776,
       0.52572707, 0.53914989, 0.53914989, 0.51454139, 0.52572707,
       0.53467562, 0.52348993, 0.51230425, 0.53467562, 0.75391499,
       0.75391499, 0.77181208, 0.53467562, 0.54138702, 0.5458613 ,
       0.25727069, 0.2639821 , 0.27293065, 0.38926174, 0.40715884,
       0.44519016, 0.68680089, 0.70246085, 0.69574944, 0.05369128,
       0.03803132, 0.08501119, 0.59731544, 0.60626398, 0.6196868 ,
       0.61073826, 0.61521253, 0.62416107, 0.29530201, 0.2639821 ,
       0.20357942, 0.21923937, 0.23489933, 0.59955257, 0.60626398,
       0.61521253, 0.49888143, 0.47651007, 0.20805369, 0.20134228,
```

```
0.79865772, 0.82102908, 0.91051454, 0.10961969, 0.19463087,
0.16778523, 0.28411633, 0.77852349, 0.78970917, 0.62416107,
0.63534676, 0.65324385, 0.60850112, 0.61521253, 0.63758389,
0.34675615, 0.35123043, 0.41834452, 0.74272931, 0.7606264 ,
0.74720358, 0.72259508, 0.71588367, 0.75838926, 0.76286353,
0.76733781, 0.3803132 , 0.55257271, 0.60178971, 0.60626398,
0.55928412, 0.65100671, 0.65548098, 0.67337808, 0.55033557,
0.64205817, 0.57494407, 0.3310962 , 0.32438479, 0.63758389,
0.64205817, 0.64205817, 0.39149888, 0.53020134, 0.52572707,
0.52796421, 0.53243848, 0.77628635, 0.78076063, 0.84340045,
0.38702461, 0.38255034, 0.64205817, 0.61297539, 0.62639821,
0.7606264 , 0.76286353, 0.77852349, 0.30425056, 0.31319911,
0.32662192, 0.76957494, 0.77628635, 0.78076063, 0.68680089,
0.56823266, 0.30201342, 0.29082774, 0.27740492, 0.29753915,
0.46979866, 0.46532438, 0.46308725, 0.41163311, 0.60402685,
0.61073826, 0.62416107, 0.44966443, 0.45861298, 0.46756152,
0.4966443 , 0.50111857, 0.46979866, 0.5033557 , 0.45637584,
0.48322148, 0.88814318, 0.78299776, 0.78299776, 0.82102908,
0.79418345, 0.79865772, 1.        , 0.64876957, 0.65324385,
0.66442953, 0.61744966, 0.62192394, 0.63087248, 0.44966443,
0.45637584, 0.45637584, 0.18120805, 0.19463087, 0.21923937,
0.44742729, 0.4541387 , 0.45861298, 0.57270694, 0.64205817,
0.59731544, 0.64205817, 0.64653244, 0.65771812, 0.01118568,
0.00447427, 0.02237136, 0.50782998, 0.33333333, 0.60178971,
0.60626398, 0.61073826, 0.75391499, 0.7606264 , 0.76957494,
0.34899329, 0.44071588, 0.55257271, 0.00671141, 0.01789709,
0.03803132, 0.63534676, 0.63982103, 0.64653244, 0.75391499,
0.61521253, 0.65100671, 0.1901566 , 0.20805369, 0.22371365,
0.76957494, 0.77628635, 0.78299776, 0.37807606, 0.55257271,
0.60850112, 0.60850112, 0.61744966, 0.68680089, 0.68680089,
0.69574944, 0.42281879, 0.43624161, 0.44071588, 0.6935123 ,
0.67785235, 0.55257271, 0.56375839, 0.6196868 , 0.12304251,
0.21700224, 0.3064877 , 0.4295302 , 0.43847875, 0.44519016,
0.44742729, 0.45637584, 0.47203579, 0.48545861, 0.82326622,
0.64205817, 0.68680089, 0.59731544, 0.31096197, 0.41834452,
0.2393736 , 0.12304251, 0.43400447, 0.41610738, 0.44519016,
0.70246085, 0.70469799, 0.70469799, 0.3310962 , 0.33557047,
0.34899329, 0.61744966, 0.62192394, 0.62639821, 0.62416107,
0.63758389, 0.6689038 , 0.5033557 , 0.5033557 , 0.51454139,
0.6689038 , 0.67561521, 0.68456376, 0.72930649, 0.7360179 ,
0.75391499, 0.75391499, 0.73154362, 0.61297539, 0.45861298,
0.46085011, 0.46308725, 0.17225951, 0.20357942, 0.24608501,
0.68680089, 0.70917226, 0.60626398, 0.43624161, 0.44742729,
0.45637584, 0.46308725, 0.3803132 , 0.39821029, 0.64205817,
0.65548098, 0.65995526, 0.6196868 , 0.62192394, 0.62192394,
0.08501119, 0.06263982, 0.48993289, 0.4966443 , 0.51006711,
0.26174497, 0.23713647, 0.21029083, 0.7852349 , 0.78747204,
0.82102908, 0.55480984, 0.6689038 , 0.65995526, 0.56375839,
0.55928412, 0.06263982, 0.04697987, 0.02908277, 0.83668904,
0.64205817, 0.63758389, 0.46756152, 0.48322148, 0.47427293,
0.60178971, 0.60626398, 0.61073826, 0.51677852, 0.49888143,
0.32662192, 0.27740492, 0.61521253, 0.61744966, 0.62416107,
0.55928412, 0.56152125, 0.56375839, 0.64653244, 0.65324385,
0.66442953, 0.75391499, 0.60178971, 0.6196868 , 0.42729306,
0.4295302 , 0.42505593, 0.14988814, 0.15659955, 0.19910515,
0.52348993, 0.5212528 , 0.51677852, 0.69127517, 0.69574944,
0.69574944, 0.51230425, 0.52572707, 0.51454139, 0.55480984,
0.55928412, 0.56599553, 0.18568233, 0.3064877 , 0.02237136,
0.01118568, 0.        ])
```

In [8]:
```
test_X
```

Out[8]:
```
array([[0.43628809, 0.05866667, 0.        , ..., 0.08214286, 0.36916395,
        0.08641975],
       [0.40720222, 0.058     , 0.        , ..., 0.70714286, 0.40499457,
        0.12345679],
       [0.00277008, 0.05866667, 0.        , ..., 0.075     , 0.37024973,
        0.10493827],
       ...,
       [0.94875346, 0.01666667, 0.09574468, ..., 0.05714286, 0.46362649,
        0.32716049],
       [0.09972299, 0.01666667, 0.2474804 , ..., 0.04285714, 0.46362649,
        0.33950617],
       [1.        , 0.018     , 0.24356103, ..., 0.33214286, 0.44191097,
        0.29012346]])
```

In [9]:
```
#basline estimate
#getting life_expectancy from the oroginal testing data
baseline_preds = testing_data.iloc[:, 1:2].values
# print(baseline_preds)
# Baseline errors, and display average baseline error
base_errors = abs(baseline_preds - test_Y)
print('Average baseline error: ', round(np.mean(base_errors), 2))
```

```
Average baseline error:  0.23
```

In [10]:
```
#Training Model using RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 300, bootstrap =
# Fit the regressor with x and y data
regressor.fit(train_X, train_Y)
# Predict with testing X
predict_Y = regressor.predict(test_X)
print(predict_Y)
```

```
[0.27504307 0.28788842 0.50786075 0.34819737 0.64556044 0.64615807
 0.65145876 0.72186517 0.61646096 0.62821983 0.64978213 0.20295531
 0.21062895 0.68409864 0.73039606 0.70032116 0.63641351 0.63922366
 0.64581089 0.8753819  0.85843309 0.84856639 0.78229605 0.81360658
 0.80695947 0.60499636 0.61605688 0.61082774 0.49657058 0.53111275
 0.53765986 0.54066192 0.54006141 0.55112788 0.836137   0.80868763
 0.84023475 0.56137666 0.56179439 0.57140595 0.2763375  0.27232715
 0.286599   0.51135448 0.39437198 0.42016247 0.61959998 0.65070249
 0.6656964  0.14499304 0.11769439 0.18671795 0.62902874 0.64498539
 0.65684604 0.6564018  0.64355802 0.64401823 0.31416473 0.28479585
 0.24747693 0.24719346 0.2289703  0.62037067 0.61827628 0.61207819
 0.49800199 0.4619032  0.25096274 0.24393281 0.84796324 0.87411945
 0.87866309 0.12727594 0.1680529  0.17772666 0.21915255 0.79465494
 0.79933475 0.61957161 0.62534721 0.62875971 0.62699763 0.63521017
 0.64686939 0.38924642 0.39079613 0.4008404  0.68584011 0.69483042
 0.72617883 0.76359389 0.76424326 0.77428046 0.78319594 0.8025978
 0.39863516 0.39658249 0.5886195  0.59114908 0.55985104 0.64119796
 0.6478707  0.66714799 0.55838447 0.59222921 0.58581667 0.32319586
 0.31834097 0.68144272 0.6837991  0.68966167 0.33275118 0.31584175
 0.5435143  0.56688833 0.5575133  0.83771151 0.84629767 0.8813196
 0.39036802 0.38270572 0.63974754 0.63979449 0.66422718 0.84388919
 0.84672833 0.83394114 0.28238655 0.30585747 0.33622244 0.78944692
 0.80769697 0.81526708 0.64610132 0.61373807 0.29033452 0.2580703
 0.28481324 0.30314057 0.48041733 0.48363375 0.48562416 0.41659476
```

```
0.59460836 0.61083617 0.63094625 0.4635011  0.51552502 0.50885115
0.51434563 0.51617193 0.51568646 0.56132054 0.50338465 0.52122809
0.86415635 0.85951447 0.85440115 0.84704094 0.82710129 0.84491378
0.84474722 0.64394442 0.65586687 0.67262527 0.64576466 0.64463787
0.64967762 0.47537373 0.49143024 0.4923914  0.15873764 0.166557
0.2195769  0.48450632 0.48739858 0.4858873  0.57873258 0.58560862
0.68252405 0.6359969  0.6242721  0.6368705  0.05133227 0.07843379
0.07921823 0.36231345 0.34319446 0.63393874 0.63662343 0.63272195
0.79667274 0.80746494 0.8407322  0.34872736 0.36916839 0.40025435
0.07068677 0.08520356 0.10870705 0.64205914 0.64601141 0.6611517
0.6335919  0.63368355 0.64379339 0.26883007 0.2682431  0.26978411
0.79394487 0.86417898 0.85765859 0.40699211 0.41929942 0.61406594
0.60210462 0.62731063 0.69102709 0.69119685 0.705868   0.45819615
0.46578604 0.45751811 0.6985441  0.68357876 0.55854867 0.58179986
0.62750418 0.1669114  0.17084007 0.16194405 0.45218223 0.4453131
0.46019322 0.43756475 0.47389824 0.49161953 0.52468877 0.84405831
0.59921785 0.60944154 0.59577402 0.32241263 0.3658595  0.22712452
0.18092377 0.47244065 0.49205159 0.47832288 0.68204846 0.68452367
0.69857755 0.37411414 0.37868599 0.38993552 0.63238502 0.63625192
0.64158981 0.663093   0.66582035 0.67252545 0.51483784 0.51658348
0.52720475 0.67603847 0.68524339 0.68035274 0.74744408 0.75621903
0.77062634 0.62130383 0.62243905 0.63046161 0.51181206 0.51026411
0.51329363 0.25353136 0.26319324 0.25912717 0.66720666 0.66984564
0.63424589 0.43026528 0.43564974 0.46404012 0.40220647 0.41017803
0.41276692 0.67057488 0.66576051 0.67049725 0.61120277 0.61735582
0.61417258 0.17463138 0.1681053  0.50441367 0.52095144 0.53603492
0.2822847  0.26625469 0.24365964 0.81646535 0.83801865 0.86559292
0.57633306 0.66276123 0.66228896 0.59934935 0.56725274 0.08668886
0.08758749 0.08311816 0.85687938 0.64123146 0.63602106 0.51420122
0.5183487  0.50766829 0.57665048 0.58366308 0.59107771 0.54473488
0.52104258 0.34977362 0.31866972 0.63802314 0.64062716 0.64193104
0.57247052 0.6014446  0.60342037 0.65170351 0.63055154 0.65709739
0.6291407  0.64990973 0.64453556 0.46711918 0.46575578 0.44203288
0.22171498 0.21667061 0.23041285 0.53373139 0.53476868 0.54033444
0.71050193 0.70687292 0.71735205 0.536801   0.54154913 0.53026827
0.54087819 0.55451601 0.56977762 0.17147477 0.1538171  0.0768272
0.10311717 0.02951271]
```

In [11]:

```python
# making prediction on the test set, we will get the prediction life expectancy
predict_Y = regressor.predict(test_X)
# Calculate the absolute errors
errors = abs(predict_Y - test_Y)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.03 degrees.
```

In [12]:

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

```
Accuracy: 89.84 %.
```

In [13]:

```python
#figuring out the usefulness of all the variable in the entire random forest
# Get numerical feature importances
testing_data_without_year = testing_data.drop(testing_data.columns[[0]],axis = 1)
```

```
# print(testing_data_without_year)
importances = list(regressor.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(t
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances]
```

```
Variable:                    17 Importance: 0.56
Variable:                    12 Importance: 0.25
Variable:                     1 Importance: 0.11
Variable:                     3 Importance: 0.01
Variable:                     7 Importance: 0.01
Variable:                    10 Importance: 0.01
Variable:                    15 Importance: 0.01
Variable:                    16 Importance: 0.01
Variable:                    18 Importance: 0.01
Variable:                     2 Importance: 0.0
Variable:                     4 Importance: 0.0
Variable:                     5 Importance: 0.0
Variable:                     6 Importance: 0.0
Variable:                     8 Importance: 0.0
Variable:                     9 Importance: 0.0
Variable:                    11 Importance: 0.0
Variable:                    13 Importance: 0.0
Variable:                    14 Importance: 0.0
```

In [14]:

```
#picking up the most important variables to recalculate the importance
updated_testing = testing_data.iloc[:, [1,3,7,10,15,16,18,12,17]]
test_important = testing_data.iloc[:, [1,3,7,10,15,16,18,12,17]].values
train_important = training_data.iloc[:, [1,3,7,10,15,16,18,12,17]].values
```

In [15]:

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

```
Variable:                     1 Importance: 0.11
Variable:                     7 Importance: 0.01
Variable:                    18 Importance: 0.01
Variable:                     3 Importance: 0.0
Variable:                    10 Importance: 0.0
Variable:                    15 Importance: 0.0
Variable:                    16 Importance: 0.0
Variable:                    12 Importance: 0.0
Variable:                    17 Importance: 0.0
```

In [16]:

```
# New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 300, boo
# 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:', np.mean(errors), 'degrees.')
updated_error_percentage = np.mean(100 * (errors / test_Y))
accuracy = 100 - updated_error_percentage
print('Accuracy:', accuracy, '%.')
```

```
Mean Absolute Error: 0.0009821423225293165 degrees.
Accuracy: -inf %.
```

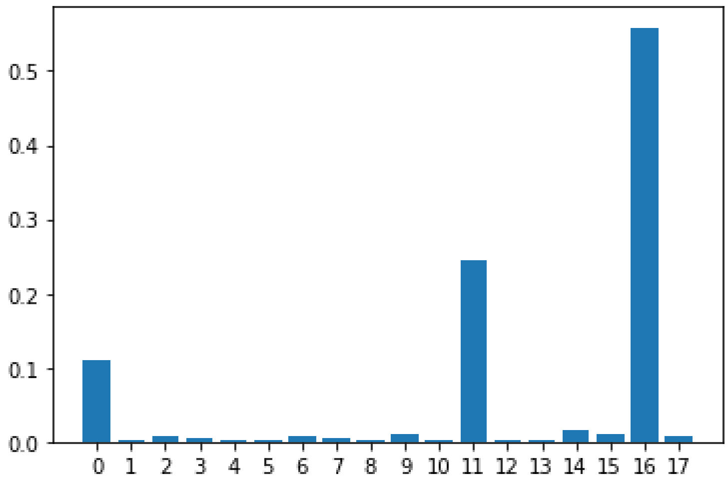
```
C:\Users\xu842\AppData\Local\Temp/ipykernel_44312/2601983393.py:10: RuntimeWarning: divi
de by zero encountered in true_divide
  updated_error_percentage = np.mean(100 * (errors / test_Y))
```

In [17]:

```
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_44312/3724588654.py in <module>
      5
      6 plt.bar(x_value, importances, orientation = 'vertical')
----> 7 plt.xticks(x_value, testing_data, rotation = 6)
      8
      9 plt.ylabel('importance');

~\anaconda3\lib\site-packages\matplotlib\pyplot.py in xticks(ticks, labels, **kwargs)
   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, labels, 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, ticklabels, 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 set_ticks, does not match the number of ticklabels (20).
```



In [18]:

```
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)))
```

```
MSE loss on testing set: 0.002251040519717124
Random forest regressor score: 0.9464184740118268
```

In [ ]: