

## ▼ Modeling Notebook One

This notebook contains the logic for performing Linear Regression, Ridge Regression, Random Forest Regression, and Random Forest Classification.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt

1 from google.colab import drive
2 drive.mount('/content/drive')

1 !ls

1 cleaned = pd.read_csv("/content/drive/Shared drives/Data Science 303 Group Project/csv/cleaned_fires_data_with_four_closest_stations_nov21.csv")
2 cleaned = cleaned.replace([np.inf, -np.inf], np.nan)
3 cleaned = cleaned.dropna()

1 display(cleaned.head())
2 display(cleaned.info(verbose=True))
3
4 STATION_LIST = ["BODIE", "BROOKS", "COHASSET", "EEL_RIVER", "HELL_HOLE", "HERNANDEZ"]
5 STATION_LIST += ["HUNTER_MOUNTAIN", "JUANITA_LAKE", "LADDER_BUTTE", "LAS_TABLAS", "LA_HONDA", "OAK_CREEK"]
6 STATION_LIST += ["PANAMINT", "PILOT_HILL", "SCORPION", "SOLDIER_MOUNTAIN", "SQUAW_LAKE", "STAMPEDE", "VAN_BREMNER", "WOLVERTON"]
7 STATION_LIST = set(STATION_LIST)
```

Final Part of Normalization: MinMax scaling latitude and longitude

```
1 from sklearn.preprocessing import MinMaxScaler
2 scaler = MinMaxScaler()
3
4 lat = scaler.fit_transform(np.array(cleaned["LATITUDE"]).reshape(-1, 1))
5 long = scaler.fit_transform(np.array(cleaned["LONGITUDE"]).reshape(-1, 1))
6
7 # DROP ALL LATITUDE LONGITUDE COLUMNS, THEN READD THE SCALED LATITUDE AND LONGITUDE
8 drops = []
9 for col in cleaned.columns:
10     if "LATITUDE" in col or "LONGITUDE" in col:
11         drops.append(col)
12
13 for col in drops:
14     del cleaned[col]
15
16 cleaned["S_LATITUDE"] = lat
17 cleaned["S_LONGITUDE"] = long
18
1 # Elevation for the primary station is redundant. Drop all elevation columns from the primary station, except for one
2 elevation = cleaned["PRIMARY_STATION_1_MONTHS_PRIOR_ELEVATION"]
3
4 drops = []
5 for col in cleaned.columns:
6     if "PRIMARY_STATION" in col and "ELEVATION" in col:
7         drops.append(col)
8
9 for col in drops:
10     del cleaned[col]
11
12 cleaned["PRIMARY_STATION_ELEVATION"] = elevation
13
```

```
1 # We should have a large number of fires. If not, fail
2 print(len(cleaned))
3 assert(len(cleaned) > 70000)
```

```
1 print( (1 - 360) % 365)
```

```
1 cleaned['DURATION'] = (cleaned['CONT_DOY']-cleaned['DISCOVERY_DOY'])%365
```

```
1 cleaned['DURATION'].max()
```

```
1 from sklearn.linear_model import LinearRegression, Ridge
```

```
1 # Working off of closest_station_1 for now, we can replicate later with other stations if we want
2 # 1. Get number of days that the fire was going (containment date - discovery date)
3 # Model 1a: fire duration
```

```
1 numerical = cleaned.select_dtypes(include="number")
2 numerical.head()
3 numerical.info(verbose=True)
```

```
1 cols = numerical.columns
2 closest = []
3
4 for col in cols:
5     if (col[:16] == 'CLOSEST_STATION_' or col in STATION_LIST or col == "S_LATITUDE" or col == "S_LONGITUDE"):
6         closest.append(col)
```

```
1 primary_station_data_cols = []
2 for col in cols:
```

```

3 if ("PRIMARY_STATION" in col or col in STATION_LIST or col == "S_LATITUDE" or col == "S_LONGITUDE"):
4     primary_station_data_cols.append(col)
5
6 primary_station_df = pd.DataFrame()
7 for col in primary_station_data_cols:
8     primary_station_df[col] = numerical[col]
9
10 primary_station_df["FIRE_SIZE"] = numerical["FIRE_SIZE"]
11 primary_station_df["DURATION"] = numerical["DURATION"]
12

```

```

1 NUM_FEATURES_TO_KEEP = 300 + 2
2 correlation = primary_station_df.corr(method='pearson')
3 highest_correlation = (correlation.nlargest(NUM_FEATURES_TO_KEEP, 'FIRE_SIZE').index)
4 print(f'TOP {NUM_FEATURES_TO_KEEP} FEATURES WITH HIGHEST CORRELATION TO FIRE SIZE')
5 print(list(highest_correlation))
6
7 print(f'TOP 10 FEATURES WITH HIGHEST CORRELATION TO FIRE SIZE')
8 display(highest_correlation[2:12])
9 display(correlation.nlargest(NUM_FEATURES_TO_KEEP, 'FIRE_SIZE')[2:12]["FIRE_SIZE"])
10
11 fire_size_prediction_df = primary_station_df[highest_correlation]
12 del fire_size_prediction_df["FIRE_SIZE"]
13 del fire_size_prediction_df["DURATION"]

```

```

1 display(fire_size_prediction_df)

```

```

1 NUM_FEATURES_TO_KEEP = 300 + 2
2 correlation = primary_station_df.corr(method='pearson')
3 highest_correlation = (correlation.nlargest(NUM_FEATURES_TO_KEEP, 'DURATION').index)
4 print(f'TOP {NUM_FEATURES_TO_KEEP} FEATURES WITH HIGHEST CORRELATION TO FIRE DURATION')
5 display(highest_correlation)
6
7 print(f'TOP 10 FEATURES WITH HIGHEST CORRELATION TO FIRE DURATION')
8 display(highest_correlation[2:12])
9 display(correlation.nlargest(NUM_FEATURES_TO_KEEP, 'DURATION')[2:12]["DURATION"])
10
11 fire_duration_prediction_df = primary_station_df[highest_correlation]
12 del fire_duration_prediction_df["FIRE_SIZE"]
13 del fire_duration_prediction_df["DURATION"]

```

```

1 display(fire_duration_prediction_df)

```

```

1 closest_df = pd.DataFrame()
2
3 for col in closest:
4     closest_df[col] = numerical[col]

```

```

1 closest_df.head()

```

## • Model 1: Linear Regression

```

1 from sklearn.model_selection import train_test_split
2 from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, median_absolute_error, r2_score
3 from sklearn.metrics import explained_variance_score
4 from sklearn import svm
5 from sklearn.preprocessing import PowerTransformer
6 from sklearn.ensemble import RandomForestRegressor
7
8 # def log_transform(column):
9 #     """
10 #     Transforms a feature so that it is scaled logarithmically. Useful for correcting floating point errors
11 #     """
12
13
14 # def min_max_transform(column):
15 #     """
16 #     Transforms a feature using min-max scaling
17 #     """
18 #     pass
19
20 def run_full_linear_regression_with_accuracy(features, response, n_highest = 0):
21     print(f"Response Minimum: {response.min()}")
22     print(f"Response Maximum: {response.max()}")
23
24     model = LinearRegression()
25     # Split into test and training set
26     X_train, X_test, y_train, y_test = train_test_split(features, response, test_size=0.2, random_state=2020)
27     model.fit(X_train, y_train)
28
29     training_predictions = model.predict(X_train)
30     test_predictions = model.predict(X_test)
31     print(f"Linear Regression Coefficients: { model.coef_ }\nLinear Regression Intercept: { model.intercept_ }\n")
32
33     print("Training: mean absolute error: ", mean_absolute_error(y_train, training_predictions))
34     print("Test mean absolute error: ", mean_absolute_error(y_test, test_predictions))
35
36     print("Training average error rate: ", median_absolute_error(y_train, training_predictions))
37     print("Test average error rate: ", median_absolute_error(y_test, test_predictions))
38
39     print("Explained variance in training set is: ", explained_variance_score(y_train, training_predictions))
40     print("Explained variance in test set is: ", explained_variance_score(y_test, test_predictions))
41
42     print("R2 Score training set: ", r2_score(y_train, training_predictions))
43     print("R2 Score testing set: ", r2_score(y_test, test_predictions))
44
45     fig = plt.figure()

```

```

46 ax1 = fig.add_subplot()
47 ax1.set_xlabel("True Value from Test Set")
48 ax1.set_ylabel('Prediction from Test Set')
49 ax1.set_title('True Value vs Predicted Value for Test Set: Linear Regression')
50 ax1.scatter(y_test, test_predictions)
51 plt.show()
52
53 # Residual plot against predictor
54 fig = plt.figure()
55 ax1 = fig.add_subplot()
56 ax1.set_xlabel("Test Set Prediction")
57 ax1.set_ylabel('Residuals from Test Set')
58 ax1.set_title('Residual Graph: Linear Regression')
59 ax1.scatter(test_predictions, y_test - test_predictions)
60 plt.show()
61
62
63
64 def run_full_ridge_regression_with_accuracy(features, response, n_highest = 0):
65     print(f"Response Minimum: {response.min()}")
66     print(f"Response Maximum: {response.max()}")
67
68
69     model = Ridge()
70     # Split into test and training set
71     X_train, X_test, y_train, y_test = train_test_split(features, response, test_size=0.2, random_state=2020)
72     model.fit(X_train, y_train)
73
74     training_predictions = model.predict(X_train)
75     test_predictions = model.predict(X_test)
76     print(f"\Ridge Regression Coefficients: { model.coef_ }\Ridge Regression Intercept: { model.intercept_ }\n")
77
78     print("Training: mean absolute error: ", mean_absolute_error(y_train, training_predictions))
79     print("Test mean absolute error: ", mean_absolute_error(y_test, test_predictions))
80
81     print("Training average error rate: ", median_absolute_error(y_train, training_predictions))
82     print("Test average error rate: ", median_absolute_error(y_test, test_predictions))
83
84     print("Explained variance in training set is: ", explained_variance_score(y_train, training_predictions))
85     print("Explained variance in test set is: ", explained_variance_score(y_test, test_predictions))
86
87     print("R2 Score training set: ", r2_score(y_train, training_predictions))
88     print("R2 Score testing set: ", r2_score(y_test, test_predictions))
89
90     fig = plt.figure()
91     ax1 = fig.add_subplot()
92     ax1.set_xlabel("True Value from Test Set")
93     ax1.set_ylabel('Prediction from Test Set')
94     ax1.set_title('True Value vs Predicted Value for Test Set: Ridge Regression')
95     ax1.scatter(y_test, test_predictions)
96     plt.show()
97
98     # Residual plot against predictor
99     fig = plt.figure()
100    ax1 = fig.add_subplot()
101    ax1.set_xlabel("Test Set Prediction")
102    ax1.set_ylabel('Residuals from Test Set')
103    ax1.set_title('Residual Graph: Ridge Regression')
104    ax1.scatter(test_predictions, y_test - test_predictions)
105    plt.show()
106
107 def run_full_SVM_regression_with_accuracy(features, response, n_highest = 0):
108     print(f"Response Minimum: {response.min()}")
109     print(f"Response Maximum: {response.max()}")
110
111
112     model = svm.SVR()
113     # Split into test and training set
114     X_train, X_test, y_train, y_test = train_test_split(features, response, test_size=0.2, random_state=2020)
115     model.fit(X_train, y_train)
116
117     training_predictions = model.predict(X_train)
118     test_predictions = model.predict(X_test)
119
120     print("Training: mean absolute error: ", mean_absolute_error(y_train, training_predictions))
121     print("Test mean absolute error: ", mean_absolute_error(y_test, test_predictions))
122
123     print("Training average error rate: ", median_absolute_error(y_train, training_predictions))
124     print("Test average error rate: ", median_absolute_error(y_test, test_predictions))
125
126     print("Explained variance in training set is: ", explained_variance_score(y_train, training_predictions))
127     print("Explained variance in test set is: ", explained_variance_score(y_test, test_predictions))
128
129     fig = plt.figure()
130     ax1 = fig.add_subplot()
131     ax1.set_xlabel("True Value from Test Set")
132     ax1.set_ylabel('Prediction from Test Set')
133     ax1.set_title('True Value vs Predicted Value for Test Set: SVM Regression')
134     ax1.scatter(y_test, test_predictions)
135     plt.show()
136
137     # Residual plot against predictor
138     fig = plt.figure()
139     ax1 = fig.add_subplot()
140     ax1.set_xlabel("Test Set Prediction")
141     ax1.set_ylabel('Residuals from Test Set')
142     ax1.set_title('Residual Graph: SVM Regression')
143     ax1.scatter(test_predictions, y_test - test_predictions)
144     plt.show()
145
146 def run_full_random_forest_regression_with_accuracy(features, response, n_highest = 0):
147     print(f"Response Minimum: {response.min()}")
148     print(f"Response Maximum: {response.max()}")
149
150

```

```
151 model = RandomForestRegressor(n_estimators=100)
152 # split into test and training set
153 X_train, X_test, y_train, y_test = train_test_split(features, response, test_size=0.2, random_state=2020)
154 model.fit(X_train, y_train)
155
156 training_predictions = model.predict(X_train)
157 test_predictions = model.predict(X_test)
158
159 print("Training: mean absolute error: ", mean_absolute_error(y_train, training_predictions))
160 print("Test mean absolute error: ", mean_absolute_error(y_test, test_predictions))
161
162 print("Training average error rate: ", median_absolute_error(y_train, training_predictions))
163 print("Test average error rate: ", median_absolute_error(y_test, test_predictions))
164
165 print("Explained variance in training set is: ", explained_variance_score(y_train, training_predictions))
166 print("Explained variance in test set is: ", explained_variance_score(y_test, test_predictions))
167
168 fig = plt.figure()
169 ax1 = fig.add_subplot()
170 ax1.set_xlabel("True Value from Test Set")
171 ax1.set_ylabel('Prediction from Test Set')
172 ax1.set_title('True Value vs Predicted Value for Test Set: Random Forest Regression')
173 ax1.scatter(y_test, test_predictions)
174 plt.show()
175
176 fig = plt.figure()
177
178 ax1 = fig.add_subplot()
179 ax1.set_xlabel("True Value from Training Set")
180 ax1.set_ylabel('Prediction from Training Set')
181 ax1.set_title('True Value vs Predicted Value for Training Set: Random Forest Regression')
182 ax1.scatter(y_train, training_predictions)
183 plt.show()
184
185 # Residual plot against predictor
186 fig = plt.figure()
187 ax1 = fig.add_subplot()
188 ax1.set_xlabel("Test Set Prediction")
189 ax1.set_ylabel('Residuals from Test Set')
190 ax1.set_title('Residual Graph: Random Forest')
191 ax1.scatter(test_predictions, y_test - test_predictions)
192 plt.show()
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Fire Size Prediction using Linear Regression and Ridge Regression

Here, we predict the size of a fire using linear regression

```
1 FIRE_SIZE = numerical["FIRE_SIZE"]
2 FIRE_SIZE.hist()

1 log_fire_size = pd.DataFrame(np.log(FIRE_SIZE))
2 log_fire_size = log_fire_size.replace([np.inf, -np.inf], np.nan)
3 log_fire_size = log_fire_size.fillna(0)
4 log_fire_size = (log_fire_size - log_fire_size.mean()) / log_fire_size.std()
5 log_fire_size.hist()
6 plt.show()
7
8 # run_full_linear_regression_with_accuracy(fire_size_prediction_df[:SUBSET], log_fire_size[:SUBSET])
9 run_full_ridge_regression_with_accuracy(fire_size_prediction_df[:SUBSET], log_fire_size[:SUBSET])
```

LINEAR REGRESSION and RIDGE REGRESSION for Summer Months

```
1 # LINEAR REGRESSION FOR SUMMER MONTHS
2 log_fire_size = pd.DataFrame(np.log(FIRE_SIZE))
3 log_fire_size = log_fire_size.replace([np.inf, -np.inf], np.nan)
4 log_fire_size = log_fire_size.fillna(0)
5 log_fire_size = (log_fire_size - log_fire_size.mean()) / log_fire_size.std()
6 log_fire_size.hist()
7 plt.show()
8
9 #x = 11
10 # x = 6
11 # x = 1
12 summer_fires = ((cleaned['DISCOVERY_MONTH'] >= 5) & (cleaned['DISCOVERY_MONTH'] <= 8))
13 summer_fires_log_size = log_fire_size[summer_fires]
14 summer_fires_df = fire_size_prediction_df[summer_fires]
15 print(len(summer_fires_log_size))
16
17 run_full_linear_regression_with_accuracy(summer_fires_df, summer_fires_log_size)
18 run_full_ridge_regression_with_accuracy(summer_fires_df, summer_fires_log_size)
```

Fire Duration Prediction using Linear Regression and Ridge Regression

Here, we predict the duration of a fire using linear regression

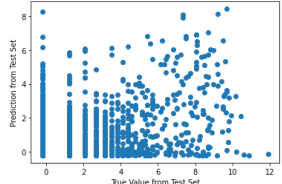
```
1 FIRE_DURATION = numerical["DURATION"]
2 FIRE_DURATION.hist()

1 log_fire_duration = pd.DataFrame(np.log(FIRE_DURATION))
2 log_fire_duration = log_fire_duration.replace([np.inf, -np.inf], np.nan)
3 log_fire_duration = log_fire_duration.fillna(0)
4 log_fire_duration = (log_fire_duration - log_fire_duration.mean()) / log_fire_duration.std()
5 log_fire_duration.hist()
6 plt.show()
7
```

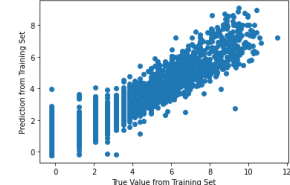
```
8 # run_full_linear_regression_with_accuracy(fire_duration_prediction_df[:SUBSET], log_fire_duration[:SUBSET])
9 run_full_ridge_regression_with_accuracy(fire_duration_prediction_df[:SUBSET], log_fire_duration[:SUBSET])
10

1 run_full_random_forest_regression_with_accuracy(fire_duration_prediction_df[:SUBSET], log_fire_duration[:SUBSET])

Response Minimum: DURATION      -0.214565
dtype: float64
Response Maximum: DURATION      11.92416
dtype: float64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:154: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
Training: mean absolute error: 0.11754670591465904
Test mean absolute error: 0.32449745615517356
Training average error rate: 1.2490009027033011e-15
Test average error rate: 0.04173831202560002
Explained variance in training set is: 0.8867669076109592
Explained variance in test set is: 0.22634597507339815
True Value vs Predicted Value for Test Set: Random Forest Regression
```



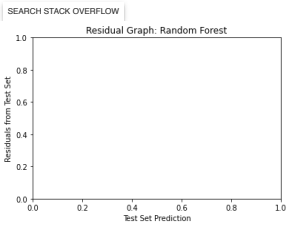
True Value vs Predicted Value for Training Set: Random Forest Regression



```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-31-f1646410162a> in <module>()
----> 1 run_full_random_forest_regression_with_accuracy(fire_duration_prediction_df[:SUBSET], log_fire_duration[:SUBSET])
```

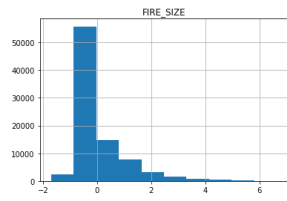
```
3 frames
/usr/local/lib/python3.7/dist-packages/pandas/core/ops/_init_.py in to_series(right)
464     if len(left.columns) != len(right):
465         raise ValueError(
--> 466             msg.format(req_len=len(left.columns), given_len=len(right))
467         )
468     right = left._constructor_sliced(right, index=left.columns)

ValueError: Unable to coerce to Series, length must be 1: given 17421
```



▼ Random Forest Regression FIRE\_SIZE

```
1 log_fire_size = pd.DataFrame(np.log(FIRE_SIZE))
2 log_fire_size = log_fire_size.replace([np.inf, -np.inf], np.nan)
3 log_fire_size = log_fire_size.fillna(0)
4 log_fire_size = (log_fire_size - log_fire_size.mean()) / log_fire_size.std()
5 log_fire_size.hist()
6 plt.show()
7
8 # run_full_svm_regression_with_accuracy(fire_size_prediction_df[:SUBSET], log_fire_size[:SUBSET])
9 run_full_random_forest_regression_with_accuracy(fire_size_prediction_df[:SUBSET], log_fire_size[:SUBSET])
10
```



```

Response Minimum: FIRE_SIZE    -1.698422
dtype: float64
Response Maximum: FIRE_SIZE     6.638909
dtype: float64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:173: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().

KeyboardInterrupt                                Traceback (most recent call last)
<ipython-input-30-f49f4398cab1> in <module>()
      7
      8 # run_full_svm_regression_with_accuracy(fire_size_prediction_df[!SUBSET], log_fire_size[!SUBSET])
----> 9 run_full_random_forest_regression_with_accuracy(fire_size_prediction_df[!1], log_fire_size[!1])

```

Double-click (or enter) to edit

```

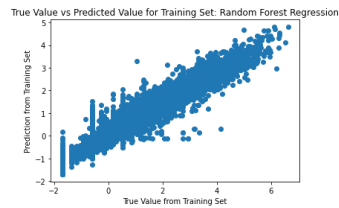
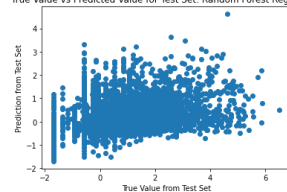
1 # Random forests for summer months
2 run_full_random_forest_regression_with_accuracy(summer_fires_df, summer_fires_log_size)

```

```

Response Minimum: FIRE_SIZE    -1.698422
dtype: float64
Response Maximum: FIRE_SIZE     6.638909
dtype: float64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:154: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
Training: mean absolute error: 0.25446062326206703
Test mean absolute error: 0.6691863760662583
Training average error rate: 0.1714524020813536
Test average error rate: 0.46032027422169611
Explained variance in training set is: 0.8693374232241285
Explained variance in test set is: 0.14721311827405048
True Value vs Predicted Value for Test Set: Random Forest Regression

```



1

## ▼ Model 2: Random Forest Classifier

```

1 # Model 2:
2 stat = numerical['STAT_CAUSE_CODE']
3 stat.value_counts()
4
5 cleaned_stat1 = numerical[numerical.STAT_CAUSE_CODE != 13.0]
6 cleaned_stat = cleaned_stat1[cleaned_stat1.STAT_CAUSE_CODE != 9.0]
7
8 col_index = cleaned_stat.columns.get_loc('STAT_CAUSE_CODE')
9 #print(col_index)
10 cleaned_stat.head()
11
12 # use either a random forest or neural network to predict the

```

```

1 from sklearn.ensemble import RandomForestClassifier
2 # Instantiate model with 135 decision trees (figured out in the next few blocks of code that this is optimal)
3 forest = RandomForestClassifier(n_estimators = 135, oob_score = True)

```

-0.155696

-0.232680

0.429544

0.396579

0.480207

```

1 x_train_mod2 = np.array(cleaned_stat.iloc[:, col_index+1:])
2 y_mod2 = cleaned_stat.iloc[:, [col_index]]
3 y_train_mod2 = np.array(y_mod2)
4
5 forest.fit(x_train_mod2, y_train_mod2.ravel())
6 print(forest.score(x_train_mod2, y_train_mod2))
7 print(forest.oob_score_)

```

```

1.0
0.6247818786422151

```

```

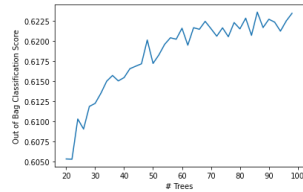
1 oob_list = []
2
3 for i in range(20, 100, 2):
4     print("curr estimators:", i)
5     forest = RandomForestClassifier(n_estimators = i, oob_score = True)
6     forest.fit(x_train_mod2, y_train_mod2.ravel())
7     oob_list.append(forest.oob_score_)
8 plt.plot(list(range(20, 100, 2)), oob_list)
9 plt.xlabel('# Trees')
10 plt.ylabel('Out of Bag Classification Score')
11 plt.show()

```

```

curr estimators: 20
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.
  UserWarning,
curr estimators: 22
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.
  UserWarning,
curr estimators: 24
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.
  UserWarning,
curr estimators: 26
curr estimators: 28
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:554: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable OOB estimates.
  UserWarning,
curr estimators: 30
curr estimators: 32
curr estimators: 34
curr estimators: 36
curr estimators: 38
curr estimators: 40
curr estimators: 42
curr estimators: 44
curr estimators: 46
curr estimators: 48
curr estimators: 50
curr estimators: 52
curr estimators: 54
curr estimators: 56
curr estimators: 58
curr estimators: 60
curr estimators: 62
curr estimators: 64
curr estimators: 66
curr estimators: 68
curr estimators: 70
curr estimators: 72
curr estimators: 74
curr estimators: 76
curr estimators: 78
curr estimators: 80
curr estimators: 82
curr estimators: 84
curr estimators: 86
curr estimators: 88
curr estimators: 90
curr estimators: 92
curr estimators: 94
curr estimators: 96
curr estimators: 98

```



```

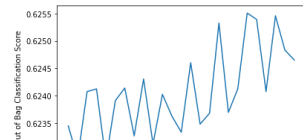
1 oob_list2 = []
2
3 for i in range(100, 150, 2):
4     print("curr estimators:", i)
5     forest = RandomForestClassifier(n_estimators = i, oob_score = True)
6     forest.fit(x_train_mod2, y_train_mod2.ravel())
7     oob_list2.append(forest.oob_score_)
8 plt.plot(list(range(100, 150, 2)), oob_list2)
9 plt.xlabel('# Trees')
10 plt.ylabel('Out of Bag Classification Score')
11 plt.show()

```

```

curr estimators: 100
curr estimators: 102
curr estimators: 104
curr estimators: 106
curr estimators: 108
curr estimators: 110
curr estimators: 112
curr estimators: 114
curr estimators: 116
curr estimators: 118
curr estimators: 120
curr estimators: 122
curr estimators: 124
curr estimators: 126
curr estimators: 128
curr estimators: 130
curr estimators: 132
curr estimators: 134
curr estimators: 136
curr estimators: 138
curr estimators: 140
curr estimators: 142
curr estimators: 144
curr estimators: 146
curr estimators: 148

```



```

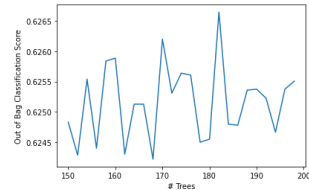
1 oob_list3 = []
2
3 for i in range(150, 200, 2):
4     print("curr estimators:", i)
5     forest = RandomForestClassifier(n_estimators = i, oob_score = True)
6     forest.fit(x_train_mod2, y_train_mod2.ravel())
7     oob_list3.append(forest.oob_score_)
8 plt.plot(list(range(150, 200, 2)), oob_list3)
9 plt.xlabel('# Trees')
10 plt.ylabel('Out of Bag Classification Score')
11 plt.show()

```

```

curr estimators: 150
curr estimators: 152
curr estimators: 154
curr estimators: 156
curr estimators: 158
curr estimators: 160
curr estimators: 162
curr estimators: 164
curr estimators: 166
curr estimators: 168
curr estimators: 170
curr estimators: 172
curr estimators: 174
curr estimators: 176
curr estimators: 178
curr estimators: 180
curr estimators: 182
curr estimators: 184
curr estimators: 186
curr estimators: 188
curr estimators: 190
curr estimators: 192
curr estimators: 194
curr estimators: 196
curr estimators: 198

```



```

1 forest = RandomForestClassifier(n_estimators = 182, oob_score = True)
2 forest.fit(x_train_mod2, y_train_mod2.ravel())
3 val_list = forest.feature_importances_
4 idx_list = np.argsort(val_list)[::-1]
5
6 print('From high to low:')
7 for idx in idx_list:
8     print('Feature %d: %f' % (idx, val_list[idx]))

```

```

From high to low:
Feature 0: 0.215777
Feature 4: 0.187571
Feature 3: 0.174209
Feature 1: 0.166072
Feature 2: 0.162683
Feature 5: 0.050670
Feature 6: 0.043017

```

```

1 x_train_mod3 = np.array(cleaned_stat.iloc[:, col_index+1:])
2 y_mod3 = cleaned_stat.iloc[:, col_index]
3 y_train_mod3 = np.array(y_mod3)

```



```
4
5 forest.fit(x_train_mod3, y_train_mod3.ravel())
6 print(forest.score(x_train_mod3, y_train_mod3))
7 print(forest.oob_score_)
```

```
1.0
0.6264445395581603
```

```
1 forest = RandomForestClassifier(n_estimators = 182, oob_score = True)
2
3 x_train_mod2 = np.array(cleaned_stat.iloc[:, col_index+1:])
4 y_mod2 = cleaned_stat.iloc[:, [col_index]]
5 y_train_mod2 = np.array(y_mod2)
6
7 X_train, X_test, y_train, y_test = train_test_split(cleaned_stat, y_train_mod2, test_size=0.2, random_state=2020)
8
9 forest.fit(X_train, y_train.ravel())
10 print(forest.score(X_test, y_test))
11 print(forest.oob_score_)
12
13
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  "X does not have valid feature names, but"
0.8585185185185186
0.8519837023623343
```