

Fraudulent Auto Insurance Claim Detection Model

Overview

According to Verisk Analytics, auto insurance fraud is a \$29 billion problem. This is a result of omitted or misrepresented underwriting information and criminally inflated claims, leading to inadequate insurance and lower rates. But, there is no such thing as a free lunch. As you can imagine, this means that Insurance Companies are getting scammed out of money, and their customer's wallets are collectively taking the hit. The goal of our model is to predict what auto insurance claims are likely to be overinflated.

The Fraudulent Auto Insurance Claim Detection Model developed in this project could be of great value to any insurance company seeking to probe for and detect fraudulent or inflated insurance claims.

Business Understanding

According to the FBI, the average (and most likely hard working, rule following) American family spends an extra 400 to 700 on insurance premiums every year because of insurance fraud.

A major insurance company (think All-State, StateFarm, Geico, etc.) approached John and I a few weeks ago to help out their fraudulent claim division. Putting their customers' needs first, they believe they can save their company and their customers a substantial dollar amount if they had a better way to detect inflated and fraudulent insurance claims.

There must be something in the air in the "Windy City," because Chicago proper is one of our clients' most fraudulent territories in the United States. Before implementing nationally, they want to test a beta model in Illinois to gauge efficacy. Utilizing the city of Chicago's transportation data portal, we were able to access information on every single documented car crash. Specifically, we used three sizable dataframes holding information about:

- 1) The crash itself
- 2) The people involved
- 3) The vehicles involved

Data Understanding and Preparation

All the data used was gathered from the city of Chicago's "Chicago Data Portal". In order to get the most relevant data, we isolated the data taken between January of 2017 and January of 2022. We used three dataframes: 1) "Traffic Crashes - Crashes" 2) "Traffic Crashes - People" 3) "Traffic Crashes - Cars"

Raw Data:

Traffic Crashes - Crashes: 617,346 rows × 49 features

Traffic Crashes - People: 777,348 rows × 11 features

Traffic Crashes - Cars: 1,266,486 rows × 72 features

Refined and merged data, before OneHotEncoding: 616067 rows × 41 columns

Our target variable comes from the "Traffic Crashes - Crashes dataset". It was originally called "DAMAGE", and contained information on the cost of damages to the car, which could be one of three categories: "Under 500 dollars"(12 percent), "500-1500 dollars"(28 percent), and "Over 1500 dollars(60 percent)".

In order to make our target binary and more balanced we combined the first two categories, making our new target: "Under 1500 dollars"(40 percent), "Over 1500 dollars"(60 percent).

```
In [ ]: 1 #import modules
        2
        3 import pandas as pd
        4 import numpy as np
        5
        6 import matplotlib.pyplot as plt
        7 import seaborn as sns
        8 from scipy import stats as stats
        9
       10 from sklearn.preprocessing import StandardScaler
       11 from sklearn.linear_model import LogisticRegression
       12 from sklearn.tree import DecisionTreeClassifier
       13 from sklearn.model_selection import train_test_split, GridSearchCV,\
       14 cross_val_score, RandomizedSearchCV
       15 from sklearn.metrics import accuracy_score, recall_score, precision_score
       16 from sklearn.metrics import plot_confusion_matrix
       17 from sklearn.metrics import roc_auc_score, plot_roc_curve
       18 from sklearn.metrics import log_loss
       19 from sklearn.metrics import make_scorer
       20 from sklearn.model_selection import StratifiedKFold
       21 from sklearn.base import clone
       22 from sklearn.dummy import DummyClassifier
       23 from sklearn.feature_selection import SelectFromModel
       24 from sklearn.impute import MissingIndicator, SimpleImputer
       25
       26 from sklearn.preprocessing import OneHotEncoder
       27 from sklearn.impute import SimpleImputer
       28 from sklearn.pipeline import Pipeline
       29 from sklearn.compose import ColumnTransformer
```

Import, explore, and clean "Crash" Data

```
In [ ]: 1 #import Crash DataFrame
        2 crash_df = pd.read_csv('data/Traffic_Crashes_-_Crashes.csv')

In [ ]: 1 crash_df

In [ ]: 1 crash_df.info()

In [ ]: 1 crash_df.describe()

In [ ]: 1 #Drop Irrelevant columns
        2 crash_df.drop(['RD_NO', 'LANE_CNT', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_COND']

In [ ]: 1 #crash_df.info()

In [ ]: 1 #Fill/Drop relevant nulls
        2 crash_df["INTERSECTION_RELATED_I"].fillna("Unknown", inplace=True)
        3 crash_df["NOT_RIGHT_OF_WAY_I"].fillna("Unknown", inplace=True)
        4 crash_df["HIT_AND_RUN_I"].fillna("Unknown", inplace=True)
        5 crash_df["MOST_SEVERE_INJURY"].fillna("Unknown", inplace=True)
        6 crash_df.dropna(subset=["INJURIES_INCAPACITATING"], inplace=True)

In [ ]: 1 #create plot to show distribution of damage categories
        2 sns.histplot(crash_df['DAMAGE'])
        3
```

Import, explore, and clean "People" Data

```
In [ ]: 1 #import People DataFrame
        2 people_df = pd.read_csv('data/Traffic_Crashes_-_People.csv')

In [ ]: 1 #people_df

In [ ]: 1 #people_df.info()

In [ ]: 1 #Drop irrelevant columns
        2 people_df.drop(['RD_NO', 'CRASH_DATE', 'SEAT_NO', 'CITY', 'STATE', 'ZIPCODE']
```

```
In [ ]: 1 #Remove nulls from relevant rows
2 people_df.dropna(subset=["VEHICLE_ID"], inplace=True)
3 people_df.dropna(subset=["SEX"], inplace=True)
4 people_df.dropna(subset=["SAFETY_EQUIPMENT"], inplace=True)
5 people_df.dropna(subset=["AIRBAG_DEPLOYED"], inplace=True)
6 people_df.dropna(subset=["DRIVER_ACTION"], inplace=True)
7 people_df.dropna(subset=["DRIVER_VISION"], inplace=True)
8 people_df.dropna(subset=["PHYSICAL_CONDITION"], inplace=True)
9 people_df.dropna(subset=["AGE"], inplace=True)
```

```
In [ ]: 1 people_df.info()
```

Import, explore, and clean "Car" Data

```
In [ ]: 1 car_df = pd.read_csv('data/Traffic_Crashes_-_Vehicles.csv')
```

```
In [ ]: 1 #car_df
```

```
In [ ]: 1 #car_df.info()
```

```
In [ ]: 1 #Create new Car DataFrame with only relevant columns
2 clean_car_df = car_df[['CRASH_RECORD_ID', 'UNIT_TYPE', 'MAKE', 'MODEL', 'VEHICLE_YEAR', 'VEHICLE_DEFECT', 'VEHICLE_USE', 'MANEUVER', 'TOWED_I', 'EXCEED_SPEED_LIMIT_I']]
3
```

```
In [ ]: 1 #clean_car_df
```

```
In [ ]: 1 #clean_car_df.info()
```

```
In [ ]: 1 #drop nulls
2 clean_car_df.dropna(subset=["UNIT_TYPE"], inplace=True)
3 clean_car_df.dropna(subset=["MAKE"], inplace=True)
4 clean_car_df.dropna(subset=["MODEL"], inplace=True)
5 clean_car_df.dropna(subset=["VEHICLE_YEAR"], inplace=True)
6 clean_car_df.dropna(subset=["VEHICLE_DEFECT"], inplace=True)
7 clean_car_df.dropna(subset=["VEHICLE_USE"], inplace=True)
8 clean_car_df.dropna(subset=["MANEUVER"], inplace=True)
9 clean_car_df["TOWED_I"].fillna("Unknown", inplace=True)
10 clean_car_df["EXCEED_SPEED_LIMIT_I"].fillna("Unknown", inplace=True)
```

```
In [ ]: 1 clean_car_df.info()
```

Merge Crash, People, and Car Data

```
In [ ]: 1 #merge crash data and people data
2 crash_people_df = pd.merge(crash_df, people_df, how='left', left_on = 'CRASH_RECORD_ID', right_on = 'PEOPLE_RECORD_ID')
3
4 #remove duplicates
5 crash_people_df.drop_duplicates(subset = 'CRASH_RECORD_ID', inplace = True)
```

```
In [ ]: 1 #rename '_merge' column to 'Check', necessary for second merge
2 crash_people_df.rename(columns = {'_merge': 'Check'}, inplace = True)
```

```
In [ ]: 1 #merge crash and people, and car DataFrames together(CPC)
2 cpc_df = pd.merge(crash_people_df, clean_car_df, how='left', left_on = 'CRASH_RECORD_ID', right_on = 'CAR_RECORD_ID')
3
4 #drop duplicates
5 cpc_df.drop_duplicates(subset = 'CRASH_RECORD_ID', inplace = True)
```

Explore and clean new DataFrame

```
In [ ]: 1 pd.set_option('display.max_columns', None)
```

```
In [ ]: 1 cpc_df
```

```
In [ ]: 1 cpc_df.info()
```

We predicted that the make of the car could be, to some extent, correlated with the cost of the repairs. You would image the repairs for fender-bender on a Rolls-Royce would be far more expensive than, say, a Toyota.

That being said, we also knew that we would have to OneHotEncode(OHE) every single make(which would've been several hundred new features), so we decided to just OHE the most popular 150 makes.

Further, the Car-Model could've been even more valuable, but without more time we didn't think we could create an efficient model adding that many more features. As you can imagine, nearly every car model built under the sun was on that list.

```
In [ ]: 1 #create a new column with only the 150 most occurring "Makes", and an 'Other' category
2 TOP_MAKES = cpc_df['MAKE'].value_counts()
3 threshold = 150
4 cpc_df['TOP_MAKES'] = np.where(cpc_df['MAKE'].isin(TOP_MAKES.index[0:TOP_MAKES.index.size - 1]), TOP_MAKES.index[0:TOP_MAKES.index.size - 1], 'Other')
```

```
In [ ]: 1 #create plot for damage density
2 damage_density = sns.histplot(crash_df['DAMAGE'], stat = 'density', color
3 damage_density.set_xlabel("Repair Cost", fontsize = 15)
4 damage_density.set_ylabel("Percent of Crashes", fontsize = 15)
5 damage_density.set_title("Cost Of Repair For Car Crashes", fontsize = 20)
```

Here we see an pretty imbalanced distribution within our target feature. In order to make these more even, we decided to combine the two lowest categories into one.

```
In [ ]: 1 #Use map function to create a binary target column
2 #This helps to create more balanced dataset
3 map = {"OVER $1,500":1, "$501 - $1,500": 0, "$500 OR LESS": 0}
4
5 cpc_df["Target"] = cpc_df["DAMAGE"].map(map)
```

```
In [ ]: 1 #check for balanced dataset
2 #check to see the number of "events" vs "non-events" or most frequent out
3 cpc_df["Target"].value_counts(normalize=True)
```

Here, we see that an "event" (1)("over \$1,500") occurs about 60% of the time.

```
In [ ]: 1 #cpc_df.info()
```

```
In [ ]: 1 #drop irrelevant columns
2 cpc_df.drop(['PERSON_ID', 'CRASH_RECORD_ID', 'DAMAGE', 'CRASH_DATE', 'PERSON_ID',
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
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
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2
```

```
In [ ]: 1 top_5_low = low_cost_df['PRIM_CONTRIBUTORY_CAUSE'].value_counts(normalize=
2
```

```
In [ ]: 1 top_5_high = high_cost_df['PRIM_CONTRIBUTORY_CAUSE'].value_counts(normali
2 top_5_high
```

```
In [ ]: 1 top_5_high.plot(kind = 'barh', title = "Top 5 Primary Cause for High Cost
2
```

Looking at the top 5 primary causes for high cost and low cost accidnets.

```
In [ ]: 1 ax = top_5_high.plot(kind = 'barh', title = "Top 5 Primary Cause for High
2 ax.set_xlabel("Percent of High Cost Accidents")
3 patches, labels = ax.get_legend_handles_labels()
```

```
In [ ]: 1 ax = top_5_low.plot(kind = 'barh', title = "Top 5 Primary Cause for Low C
2 ax.set_xlabel("Percent of Low Cost Accidents")
```

Modeling

Test Train Split

```
In [ ]: 1 #create a numeric feature dataframe
2 #perform test train split
3
4 numeric_df = cpc_df[['POSTED_SPEED_LIMIT', 'NUM_UNITS', 'INJURIES_INCAPACIT
5 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH', 'AGE
6 'VEHICLE_YEAR', 'Target']]
7 X = numeric_df.drop("Target", axis=1)
8 y = numeric_df["Target"]
9 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=20
```

1st Model - "Dummy Model" (Baseline)

This model will predict the most frequent class for every observation. In other words, our model will "guess" the target that occurs most often. This will be a good baseline to compare future models against.

```
In [ ]: 1 #instantiate dummy model
2 dummy_model = DummyClassifier(strategy="most_frequent")
```

```
In [ ]: 1 #fit model
        2 dummy_model.fit(X_train, y_train)
```

```
In [ ]: 1 dummy_model.predict(y_train)[:50]
```

Here we see that guessing the most frequent event (1) every time, our model will be correct about 60% of the time(as this is the proportion of events(1) to nonevents(0)).

```
In [ ]: 1 #create confusion matrix
        2 plot_confusion_matrix(dummy_model, X_train, y_train)
```

Model Evaluation

Cross-validation will allow us to see how the model would do in generalizing to new data it's never seen.

```
In [ ]: 1 cv_results = cross_val_score(dummy_model, X_train, y_train, cv=5)
        2 cv_results
```

As we predicted, our model was correct approximately 60% of the time.

To show the spread, we'll make a convenient class that can help us organize the model and the cross-validation:


```

In [ ]: 1 class ModelWithCV():
2         '''Structure to save the model and more easily see its crossvalidation
3
4     def __init__(self, model, model_name, X, y, cv_now=True):
5         self.model = model
6         self.name = model_name
7         self.X = X
8         self.y = y
9         # For CV results
10        self.cv_results = None
11        self.cv_mean = None
12        self.cv_median = None
13        self.cv_std = None
14        #
15        if cv_now:
16            self.cross_validate()
17
18    def cross_validate(self, X=None, y=None, kfold=10):
19        '''
20        Perform cross-validation and return results.
21
22        Args:
23            X:
24                Optional; Training data to perform CV on. Otherwise use X from
25            y:
26                Optional; Training data to perform CV on. Otherwise use y from
27            kfold:
28                Optional; Number of folds for CV (default is 10)
29        '''
30
31        cv_X = X if X else self.X
32        cv_y = y if y else self.y
33
34        self.cv_results = cross_val_score(self.model, cv_X, cv_y, cv=kfold)
35        self.cv_mean = np.mean(self.cv_results)
36        self.cv_median = np.median(self.cv_results)
37        self.cv_std = np.std(self.cv_results)
38
39
40    def print_cv_summary(self):
41        cv_summary = (
42            f'''CV Results for `{self.name}` model:
43                {self.cv_mean:.5f} ± {self.cv_std:.5f} accuracy
44            ''')
45        print(cv_summary)
46
47
48    def plot_cv(self, ax):
49        '''
50        Plot the cross-validation values using the array of results and
51        Axis for plotting.
52        '''
53        ax.set_title(f'CV Results for `{self.name}` Model')
54        # Thinner violinplot with higher bw
55        sns.violinplot(y=self.cv_results, ax=ax, bw=.4)
56        sns.swarmplot(

```

```

57         y=self.cv_results,
58         color='orange',
59         size=10,
60         alpha= 0.8,
61         ax=ax
62     )
63
64     return ax

```

```

In [ ]: ▶ 1 dummy_model_results = ModelWithCV(
          2             model=dummy_model,
          3             model_name='dummy',
          4             X=X_train,
          5             y=y_train
          6 )

```

```

In [ ]: ▶ 1 fig, ax = plt.subplots()
          2
          3 ax = dummy_model_results.plot_cv(ax)
          4 plt.tight_layout();
          5
          6 dummy_model_results.print_cv_summary()

```

```

In [ ]: ▶ 1 fig, ax = plt.subplots()
          2
          3 fig.suptitle("Dummy Model")
          4
          5 plot_confusion_matrix(dummy_model, X_train, y_train, ax=ax, cmap="plasma")

```

```

In [ ]: ▶ 1 from sklearn.metrics import accuracy_score

```

```

In [ ]: ▶ 1 plot_roc_curve(dummy_model, X_train, y_train);

```

2nd Model - Logistic Regression

Next we will create a logistic regression model and compare its performance.

We're going to specifically avoid any regularization (the default) to see how the model does with little change. Set penalty paramter = 'none' = no regularization.

```

In [ ]: ▶ 1 #setting penalty = none means there is no regulaization, and thus we will
          2 simple_logreg_model = LogisticRegression(random_state=2021, penalty='none')

```

```

In [ ]: ▶ 1 #fit model and then predict
          2 simple_logreg_model.fit(X_train, y_train)

```

```
In [ ]: 1 simple_logreg_model.predict(X_train)[200000:200050]
```

Looking at 50 random samples, we see a mix of events and non-events this time.

2nd Model - Model Evaluation

```
In [ ]: 1 simple_logreg_results = ModelWithCV(  
2         model=simple_logreg_model,  
3         model_name='simple_logreg',  
4         X=X_train,  
5         y=y_train  
6     )
```

```
In [ ]: 1 # Saving variable for convenience  
2 model_results = simple_logreg_results  
3  
4 # Plot CV results  
5 fig, ax = plt.subplots()  
6 ax = model_results.plot_cv(ax)  
7 plt.tight_layout();  
8 # Print CV results  
9 model_results.print_cv_summary()
```

We see that with no regularization and default parameters, the model performs nearly the same as our baseline model.

```
In [ ]: 1 plot_confusion_matrix(simple_logreg_model, X_train, y_train)
```

```
In [ ]: 1 fig, ax = plt.subplots()  
2  
3 fig.suptitle("Logistic Regression with Numeric Features Only")  
4  
5 plot_confusion_matrix(simple_logreg_model, X_train, y_train, ax=ax, cmap=
```

```
In [ ]: 1 plot_roc_curve(simple_logreg_model, X_train, y_train);
```

BUT, our ROC has improved. Our ROC curve now has an AUC of 0.56. This is better than our original model, but still not great. We hope by adding in more data preparation and feature engineering we can increase this more.

More Data Preparation

This time we performed a train-test split that contains all of the features.

```
In [ ]: ▶ 1 X = cpc_df.drop("Target", axis=1)
          2 y = cpc_df["Target"]
          3 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=20)
```

Handling Missing Values

```
In [ ]: ▶ 1 indicator_demo = MissingIndicator()
          2
          3 indicator_demo.fit(X_train)
          4
          5 indicator_demo.features_
```

```
In [ ]: ▶ 1 indicator_demo.transform(X_train)[:5, :]
```

```
In [ ]: ▶ 1 # below creates a missing indicator column to help us see if something is
          2 # missing a value for a partial
          3 # column, --- NOT NECESSARY
          4
          5 # what is essential !! is an imputer!!
          6 indicator = MissingIndicator(features="all")
          7 indicator.fit(X_train)
```

```
In [ ]: ▶ 1 def add_missing_indicator_columns(X, indicator):
          2     """
          3     Helper function for transforming features
          4
          5     For every feature in X, create another feature indicating whether the
          6     is missing. (This doubles the number of columns in X.)
          7     """
          8
          9     # create a 2D array of True and False values indicating whether a given
         10     # is missing for that row
         11     missing_array_bool = indicator.transform(X)
         12
         13     # transform into 1 and 0 for modeling
         14     missing_array_int = missing_array_bool.astype(int)
         15
         16     # helpful for readability but not needed for modeling
         17     missing_column_names = [col + "_missing" for col in X.columns]
         18
         19     # convert to df so it we can concat with X
         20     missing_df = pd.DataFrame(missing_array_int, columns=missing_column_names)
         21
         22     return pd.concat([X, missing_df], axis=1)
```

```
In [ ]: ▶ 1 X_train = add_missing_indicator_columns(X=X_train, indicator=indicator)
```

```

In [ ]: 1 X_train.head()

In [ ]: 1 #seperate into numeric and categ. features
2 numeric_feature_names = ['POSTED_SPEED_LIMIT', 'NUM_UNITS', 'INJURIES_INCAI
3 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH
4 categorical_feature_names = [c for c in cpc_df.columns if cpc_df[c].dtype
5
6 X_train_numeric = X_train[numeric_feature_names]
7 X_train_categorical = X_train[categorical_feature_names]

In [ ]: 1 #imputing numeric columns using the mean for imputing, bc that is the de
2 numeric_imputer = SimpleImputer()
3 numeric_imputer.fit(X_train_numeric)

In [ ]: 1 categorical_imputer = SimpleImputer(strategy="most_frequent") #here, we
2 categorical_imputer.fit(X_train_categorical)

In [ ]: 1 def impute_missing_values(X, imputer):
2     """
3     Given a DataFrame and an imputer, use the imputer to fill in all
4     missing values in the DataFrame
5     """
6     imputed_array = imputer.transform(X)
7     imputed_df = pd.DataFrame(imputed_array, columns=X.columns, index=X.i
8     return imputed_df

In [ ]: 1 X_train_numeric = impute_missing_values(X_train_numeric, numeric_imputer
2 X_train_categorical = impute_missing_values(X_train_categorical, categor

In [ ]: 1 X_train_imputed = pd.concat([X_train_numeric, X_train_categorical], axis
2 X_train_imputed.isna().sum()

In [ ]: 1 X_train = X_train.drop(numeric_feature_names + categorical_feature_names
2 X_train = pd.concat([X_train_imputed, X_train], axis=1)

In [ ]: 1 X_train.columns

In [ ]: 1 #confirmed there were no null values before OneHotEncoding
2 X_train.isna().sum()

```

One Hot Encode

```
In [ ]: 1 X = cpc_df.drop(columns='Target')
2 y = cpc_df["Target"]
3
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
```

```
In [ ]: 1 categorical_feature_names = [c for c in cpc_df.columns if cpc_df[c].dtype
2 numerical_feature_names = ['POSTED_SPEED_LIMIT', 'NUM_UNITS', 'INJURIES_INC
3 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH
```

```
In [ ]: 1
2 def encode_and_concat_feature_train(X_train, feature_name):
3     """
4     Helper function for transforming training data. It takes in the full
5     feature name, makes a one-hot encoder, and returns the encoder as well
6     with that feature transformed into multiple columns of 1s and 0s
7     """
8     # make a one-hot encoder and fit it to the training data
9     ohe = OneHotEncoder(categories="auto", handle_unknown="ignore")
10    single_feature_df = X_train[[feature_name]]
11    ohe.fit(single_feature_df)
12
13    # call helper function that actually encodes the feature and concats
14    X_train = encode_and_concat_feature(X_train, feature_name, ohe)
15
16    return ohe, X_train
```

```
In [ ]: 1
2 def encode_and_concat_feature(X, feature_name, ohe):
3     """
4     Helper function for transforming a feature into multiple columns of 1s
5     in both training and testing steps. Takes in the full X dataframe, the
6     and encoder, and returns the dataframe with that feature transformed into
7     columns of 1s and 0s
8     """
9     # create new one-hot encoded df based on the feature
10    single_feature_df = X[[feature_name]]
11    feature_array = ohe.transform(single_feature_df).toarray()
12    ohe_df = pd.DataFrame(feature_array, columns=ohe.categories_[0], index=s
13
14    # drop the old feature from X and concat the new one-hot encoded df
15    X = X.drop(feature_name, axis=1)
16    X = pd.concat([X, ohe_df], axis=1)
17
18    return X
```

```
In [ ]: 1 encoders = {}
2
3 for categorical_feature in categorical_feature_names:
4     ohe, X_train = encode_and_concat_feature_train(X_train, categorical_
5     encoders[categorical_feature] = ohe
```

```
In [ ]: 1 encoders
```

```
In [ ]: 1 X_train.head()
```

```
In [ ]: 1 X_train.shape
```

Decision Tree - For Feature Importance

```
In [ ]: 1 #Instantiate Decision Tree
2 dt = DecisionTreeClassifier(max_depth=13, random_state=42)
3
4 dt.fit(X_train, y_train)
5
6 CV_results = cross_val_score(dt,X_train,y_train,cv=5)
7 CV_results
```

```
In [ ]: 1 plot_confusion_matrix(dt,X_train,y_train)
2
```

```
In [ ]: 1 #create dictionary of feature importance
2 list = {}
3 for fi, feature in zip(dt.feature_importances_,X_train):
4     list.update({fi:feature})
```

```
In [ ]: 1 #Order by most important
2 import collections
3 od = collections.OrderedDict(sorted(list.items()),reverse=True))
4 od
```

```
In [ ]: 1 #visualize
2 n_features = dt.n_features_
3 plt.figure(figsize=(15, 70))
4 plt.barh(range(n_features), dt.feature_importances_);
5 plt.yticks(np.arange(n_features), X_train.columns.values, fontsize = 12)
6 plt.xlabel('Feature importance', fontsize = 20)
7 plt.ylabel('Features', fontsize = 20)
8 plt.title('FSM Feature Importance', fontsize = 20)
9 plt.tight_layout()
10
```

With more time, we would impute all of our "unknown" data and determine feature importance again. Based on the results, we would remove the the unimportant features and focus on the most important ones.

"3rd Model"

```
In [ ]: ▶ 1 logreg_model = LogisticRegression(random_state=2021, penalty='none')
          2 logreg_model.fit(X_train, y_train)
```

```
In [ ]: ▶ 1 #more iterations
          2 logreg_model_more_iterations = LogisticRegression(
          3                                     random_state=2021,
          4                                     penalty='none',
          5                                     max_iter=100
          6 )
          7 logreg_model_more_iterations.fit(X_train, y_train)
```

```
In [ ]: ▶ 1 #higher tolerance (C-parameter is inverse of regularization strength)
          2 #higher tolerance means that our models will stop training earlier (when
          3 #true values are not as close as they could be).
          4 logreg_model_higher_tolerance = LogisticRegression(
          5                                     random_state=2021,
          6                                     penalty='none',
          7                                     tol=25
          8 )
          9 logreg_model_higher_tolerance.fit(X_train, y_train)
```

3rd Model - Model Evaluations

```
In [ ]: ▶ 1 fix, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 6))
          2
          3 axes[0].set_title("More Iterations")
          4 axes[1].set_title("Higher Tolerance")
          5
          6 plot_confusion_matrix(logreg_model_more_iterations, X_train, y_train,
          7                   ax=axes[0], cmap="plasma")
          8 plot_confusion_matrix(logreg_model_higher_tolerance, X_train, y_train,
          9                   ax=axes[1], cmap="plasma");
```



```

In [ ]: 1 logreg_model_more_iterations_results = ModelWithCV(
        2         logreg_model_more_iterations,
        3         'more_iterations',
        4         X_train,
        5         y_train
        6     )
        7
        8 logreg_model_higher_tolerance_results = ModelWithCV(
        9         logreg_model_higher_tolerance,
       10         'higher_tolerance',
       11         X_train,
       12         y_train
       13     )
       14
       15 model_results = [
       16     logreg_model_more_iterations_results,
       17     logreg_model_higher_tolerance_results
       18 ]

```

```

In [ ]: 1 f, axes = plt.subplots(ncols=2, sharey=True, figsize=(12, 6))
        2
        3 for ax, result in zip(axes, model_results):
        4     ax = result.plot_cv(ax)
        5     result.print_cv_summary()
        6 plt.tight_layout();

```

Here we see a slight improvement from our previous scores.

```

In [ ]: 1 fig, ax = plt.subplots()
        2
        3 plot_roc_curve(logreg_model_more_iterations, X_train, y_train,
        4                 name='logreg_model_more_iterations', ax=ax)
        5 plot_roc_curve(logreg_model_higher_tolerance, X_train, y_train,
        6                 name='logreg_model_higher_tolerance', ax=ax);

```

Here, we see a major improvement! Could be result of overfitting.

4th Model - After Scaling

More Data Preparation - Scaling

```

In [ ]: 1 scaler = StandardScaler()
        2
        3 scaler.fit(X_train)

```

```
In [ ]: 1 def scale_values(X, scaler):
        2     """
        3     Given a DataFrame and a fitted scaler, use the scaler to scale all of
        4     """
        5     scaled_array = scaler.transform(X)
        6     scaled_df = pd.DataFrame(scaled_array, columns=X.columns, index=X.index)
        7     return scaled_df
```

```
In [ ]: 1 X_train = scale_values(X_train, scaler)
```

```
In [ ]: 1 X_train.head()
```

Now that we have scaled data, let's see how well our logistic regression model fits without adjusting any hyperparameters.

```
In [ ]: 1 logreg_model = LogisticRegression(random_state=2021)
        2 logreg_model.fit(X_train, y_train)
```

```
In [ ]: 1 fig, ax = plt.subplots()
        2
        3 fig.suptitle("Logistic Regression with All Features, Scaled")
        4
        5 plot_confusion_matrix(logreg_model, X_train, y_train, ax=ax, cmap="plasma")
```

```
In [ ]: 1 all_features_results = ModelWithCV(
        2     logreg_model,
        3     'all_features',
        4     X_train,
        5     y_train
        6 )
```

```
In [ ]: 1 # Saving variable for convenience
        2 model_results = all_features_results
        3
        4 # Plot CV results
        5 fig, ax = plt.subplots()
        6 ax = model_results.plot_cv(ax)
        7 plt.tight_layout();
        8 # Print CV results
        9 model_results.print_cv_summary()
```

We see that scaling improved our accuracy scores. We also see below that the AUC increased slightly.

```
In [ ]: 1 plot_roc_curve(logreg_model, X_train, y_train)
```

```
In [ ]: 1 # sorted(list(zip(X_train.columns, logreg_model.coef_[0])),
2 #               key=lambda x: abs(x[1]), reverse=True)[:50])
```

```
In [ ]: 1 #so now Lets increase the regularization - the correct the overfitting
```

Hyperparameter Adjustment

Different Regularization Strengths

```
In [ ]: 1 all_features_results.print_cv_summary()
```

```
In [ ]: 1 model_results = [all_features_results]
2 C_values = [0.0001, 0.001, 0.01, 0.1, 1]
3
4 for c in C_values:
5     logreg_model = LogisticRegression(random_state=2021, C=c)
6     logreg_model.fit(X_train, y_train)
7     # Save Results
8     new_model_results = ModelWithCV(
9         logreg_model,
10        f'all_features_c{c:e}',
11        X_train,
12        y_train
13    )
14    model_results.append(new_model_results)
15    new_model_results.print_cv_summary()
```

Here, we don't see any any significant improvement in accuracy with C-values.

```
In [ ]: 1 f,axes = plt.subplots(ncols=3, nrows=2, sharey='all', figsize=(18, 12))
2
3 for ax,result in zip(axes.ravel(),model_results):
4     ax = result.plot_cv(ax)
5
6 plt.tight_layout();
```

```
In [ ]: 1 model_results = [all_features_results]
2 all_features_cross_val_score = all_features_results.cv_results
```

Different Solvers

```
In [ ]: 1 model_results = [all_features_results]
2 all_features_cross_val_score = all_features_results.cv_results
```

```
In [ ]: ► 1 ogreg_model = LogisticRegression(random_state=2021, solver="liblinear")
          2 logreg_model.fit(X_train, y_train)
```

```
In [ ]: ► 1 # Save for Later comparison
          2 model_results.append(
          3     ModelWithCV(
          4         logreg_model,
          5         'solver:liblinear',
          6         X_train,
          7         y_train
          8     )
          9 )
         10
         11 # Plot both all_features vs new model
         12 f, axes = plt.subplots(ncols=2, sharey='all', figsize=(12, 6))
         13
         14 model_results[0].plot_cv(ax=axes[0])
         15 model_results[-1].plot_cv(ax=axes[1])
         16
         17 plt.tight_layout();
```

```
In [ ]: ► 1 print("Old:", all_features_cross_val_score)
          2 print("New:", model_results[-1].cv_results)
```

No major difference in the scores. Let's try adding some more regularization:

```
In [ ]: ► 1 logreg_model = LogisticRegression(random_state=2021, solver="liblinear",
          2 logreg_model.fit(X_train, y_train)
```

```
In [ ]: ► 1 # Save for Later comparison
          2 model_results.append(
          3     ModelWithCV(
          4         logreg_model,
          5         'solver:liblinear_C:0.01',
          6         X_train,
          7         y_train
          8     )
          9 )
         10
         11 # Plot both all_features vs new model
         12 f, axes = plt.subplots(ncols=2, sharey='all', figsize=(12, 6))
         13
         14 model_results[0].plot_cv(ax=axes[0])
         15 model_results[-1].plot_cv(ax=axes[1])
         16
         17 plt.tight_layout();
```

```
In [ ]: ► 1 print("Old:", all_features_cross_val_score)
          2 print("New:", model_results[-1].cv_results)
```

Slightly better, if any. Lets try another different type of penalty.

```
In [ ]: 1 logreg_model = LogisticRegression(random_state=2021, solver="liblinear",
2 logreg_model.fit(X_train, y_train)
```

Type *Markdown* and LaTeX: α^2

```
In [ ]: 1 #Save for Later comparison
2 # model_results.append(
3 #     ModelWithCV(
4 #         logreg_model,
5 #         'solver:liblinear_penalty:l1',
6 #         X_train,
7 #         y_train
8 #     )
9 # )
10
11 # # Plot both all_features vs new model
12 # f,axes = plt.subplots(ncols=2, sharey='all', figsize=(12, 6))
13
14 # model_results[0].plot_cv(ax=axes[0])
15 # model_results[-1].plot_cv(ax=axes[1])
16
17 # plt.tight_layout();
```

```
In [ ]: 1 print("Old:", all_features_cross_val_score)
2 print("New:", model_results[-1].cv_results)
```

This took too long to run.

```
In [ ]: 1 logreg_model = LogisticRegression(random_state=2021, solver="liblinear",
2 logreg_model.fit(X_train, y_train)
```

```
In [ ]: 1 # Save for Later comparison
2 model_results.append(
3     ModelWithCV(
4         logreg_model,
5         'solver:liblinear_penalty:l1_C:0.01',
6         X_train,
7         y_train
8     )
9 )
10
11 # Plot both all_features vs new model
12 f,axes = plt.subplots(ncols=2, sharey='all', figsize=(12, 6))
13
14 model_results[0].plot_cv(ax=axes[0])
15 model_results[-1].plot_cv(ax=axes[1])
16
17 plt.tight_layout();
```

```
In [ ]: 1 print("Old:", all_features_cross_val_score)
        2 print("New:", model_results[-1].cv_results)
```

```
In [ ]: 1 logreg_model = LogisticRegression(random_state=2021, solver="liblinear",
        2 logreg_model.fit(X_train, y_train)
        3
        4 fig, ax = plt.subplots()
        5
        6 fig.suptitle("Logistic Regression with All Features (Scaled, Hyperparameter Tuned)")
        7
        8 plot_confusion_matrix(logreg_model, X_train, y_train, ax=ax, cmap="plasma")
```

Very Similar to our previous models scores.

As we said previously, our model could be overfitting. One way to address this is to remove features, specifically, ones that have small modeling coefficients. We did this using `SelectFromModel`.

SelectFromModel

```
In [ ]: 1 selector = SelectFromModel(logreg_model)
        2
        3 selector.fit(X_train, y_train)
```

```
In [ ]: 1 #use a default threshold
        2 thresh = selector.threshold_
        3 thresh
```

```
In [ ]: 1 #Checking to see how many features will be eliminated
        2 coefs = selector.estimator_.coef_
        3 coefs
```

```
In [ ]: 1 coefs.shape
```

```
In [ ]: 1 coefs[coefs > thresh].shape
```

```
In [ ]: 1 selector.get_support()
```

```
In [ ]: 1 dict(zip(X_train.columns, selector.get_support()))
```

```
In [ ]: 1 def select_important_features(X, selector):
2         """
3         Given a DataFrame and a selector, use the selector to choose
4         the most important columns
5         """
6         imps = dict(zip(X.columns, selector.get_support()))
7         selected_array = selector.transform(X)
8         selected_df = pd.DataFrame(selected_array,
9                                     columns=[col for col in X.columns if imps
10                                              index=X.index])
11         return selected_df
```

```
In [ ]: 1 X_train_selected = select_important_features(X=X_train, selector=selector)
```

```
In [ ]: 1 X_train_selected.head()
```

```
In [ ]: 1 logreg_sel = LogisticRegression(random_state=2021, solver="liblinear", p
2
3 logreg_sel.fit(X_train_selected, y_train)
```

```
In [ ]: 1 # Save for Later comparison
2 # select_results = ModelWithCV(
3 #         logreg_sel,
4 #         'logreg_sel',
5 #         X_train_selected,
6 #         y_train
7 # )
8
9 # Plot both all_features vs new model
10 #f,axes = plt.subplots(ncols=2, sharey='all', figsize=(12, 6))
11
12 # model_results[0].plot_cv(ax=axes[0])
13 # select_results.plot_cv(ax=axes[1])
14
15 #plt.tight_layout();
```

```
In [ ]: 1 # print("Old:", all_features_cross_val_score)
2 # print("New:", select_results.cv_results)
```

Unfortunately, our final two models were taking too long to run. My kernal kept stopping. So we were not able to get our final models or run a final model evaluation at this time.

With more time, there is a lot more I would have liked to do. For starters, there were alot of "unknown"s in our data. I think that running an imputer to impute data into those features could've been very helpful. As seen, the "Unknowns" were ranked among the most important features. From this, we could then run though a decision tree again to find the most important features, allowing us to eliminate the unimportant or overinflating ones, and assigning proper weight to the important ones. I beleive doing all of this would've given us better results on our test.

Final Model Evaluation

Now that we have a final model, run `X_test` through all of the preprocessing steps so we can evaluate the model's performance

```
In [ ]: ▶ 1 # X_test_no_transformations = X_test.copy()

In [ ]: ▶ 1 # add missing indicators
          2 # X_test_mi = add_missing_indicator_columns(X_test_no_transformations, in

In [ ]: ▶ 1 # separate out values for imputation
          2 # X_test_numeric = X_test_mi[numeric_feature_names]
          3 # X_test_categorical = X_test_mi[categorical_feature_names]

In [ ]: ▶ 1 # separate out values for imputation
          2 # impute missing values
          3 # X_test_numeric = impute_missing_values(X_test_numeric, numeric_imputer,
          4 # X_test_categorical = impute_missing_values(X_test_categorical, categor
          5 # X_test_imputed = pd.concat([X_test_numeric, X_test_categorical], axis=1
          6 # X_test_new = X_test_mi.drop(numeric_feature_names + categorical_feature
          7 # X_test_final = pd.concat([X_test_imputed, X_test_new], axis=1)

In [ ]: ▶ 1 # one-hot encode categorical data
          2 # for categorical_feature in categorical_feature_names:
          3 #     X_test_final = encode_and_concat_feature(X_test_final,
          4 #                                                     categorical_feature, encoders[c

In [ ]: ▶ 1 # # scale values
          2 # X_test_scaled = scale_values(X_test_final, scaler)

In [ ]: ▶ 1 # select features
          2 # X_test_selected = select_important_features(X_test_scaled, selector)

In [ ]: ▶ 1 # X_test_selected.head()

In [ ]: ▶ 1 # final_model = LogisticRegression(random_state=2021, solver="liblinear",
          2 # final_model.fit(X_train_selected, y_train)
          3
          4 # final_model.score(X_test_selected, y_test)
```

Compare the past models


```

In [ ]: ▶ 1 # Create a way to categorize our different models
2 # model_candidates = [
3 #     {
4 #         'name': 'dummy_model'
5 #         , 'model': dummy_model
6 #         , 'X_test': X_test
7 #         , 'y_test': y_test
8 #     },
9 #     {
10 #         'name': 'simple_logreg_model'
11 #         , 'model': simple_logreg_model
12 #         , 'X_test': X_test_no_transformations[["SibSp", "Parch", "Fare"]]
13 #         , 'y_test': y_test
14 #     },
15 #     {
16 #         'name': 'logreg_model_more_iterations'
17 #         , 'model': logreg_model_more_iterations
18 #         , 'X_test': X_test_final
19 #         , 'y_test': y_test
20 #     },
21 #     {
22 #         'name': 'logreg_model_higher_tolerance'
23 #         , 'model': logreg_model_higher_tolerance
24 #         , 'X_test': X_test_final
25 #         , 'y_test': y_test
26 #     },
27 #     {
28 #         'name': 'final_model'
29 #         , 'model': final_model
30 #         , 'X_test': X_test_selected
31 #         , 'y_test': y_test
32 #     }
33 # ]

```

```

In [ ]: ▶ 1 # final_scores_dict = {
2 #     "Model Name": [candidate.get('name') for candidate in model_candidates],
3 #     "Mean Accuracy": [
4 #         candidate.get('model').score(
5 #             candidate.get('X_test'),
6 #             candidate.get('y_test')
7 #         )
8 #         for candidate in model_candidates
9 #     ]
10
11 # }
12 # final_scores_df = pd.DataFrame(final_scores_dict).set_index('Model Name')
13 # final_scores_df

```

```

In [ ]: ▶ 1 # nrows = 2
2 # ncols = math.ceil(len(model_candidates)/nrows)
3
4 # fig, axes = plt.subplots(
5 #             nrows=nrows,
6 #             ncols=ncols,
7 #             figsize=(12, 6)
8 # )
9 # fig.suptitle("Confusion Matrix Comparison")
10
11 # # Turn off all the axes (in case nothing to plot); turn on while iterating
12 # [ax.axis('off') for ax in axes.ravel()]
13
14
15 # for i,candidate in enumerate(model_candidates):
16 #     # Logic for making rows and columns for matrices
17 #     row = i // 3
18 #     col = i % 3
19 #     ax = axes[row][col]
20
21 #     ax.set_title(candidate.get('name'))
22 #     ax.set_axis_on()
23 #     cm_display = plot_confusion_matrix(
24 #         candidate.get('model'),
25 #         candidate.get('X_test'),
26 #         candidate.get('y_test'),
27 #         normalize='true',
28 #         cmap='plasma',
29 #         ax=ax,
30
31 #     )
32 #     cm_display.im_.set_clim(0, 1)
33
34 # plt.tight_layout()

```

```

In [ ]: ▶ 1 # fig, ax = plt.subplots()
2
3 # # Plot only the last models we created (so it's not too cluttered)
4 # for model_candidate in model_candidates[3:]:
5 #     plot_roc_curve(
6 #         model_candidate.get('model'),
7 #         model_candidate.get('X_test'),
8 #         model_candidate.get('y_test'),
9 #         name=model_candidate.get('name'),
10 #         ax=ax
11 #     )

```

In []: ▶

```
1 # fig, ax = plt.subplots()
2
3 # # Plot the final model against the other earlier models
4 # plot_roc_curve(
5 #     final_model,
6 #     X_test_selected,
7 #     y_test,
8 #     name='final_model',
9 #     ax=ax
10 # )
11
12 # for model_candidate in model_candidates[:3]:
13 #     plot_roc_curve(
14 #         model_candidate.get('model'),
15 #         model_candidate.get('X_test'),
16 #         model_candidate.get('y_test'),
17 #         name=model_candidate.get('name'),
18 #         ax=ax
19 #     )
```