

## ✓ Modeling Attrition Prediction

Will Byrd June 2024

### Introduction

In this notebook, we will take a look at an HR Analytics dataset and perform a binary classification to determine Attrition. Attrition is the departure of an employee from an organization for any reason. In this dataset we do not have context on why the employee leaves the company—they may have been fired or they may have resigned.

For companies and organizations—the employees are the most important asset. Therefore, it is vitally important to be able to predict behavior of these employees. Knowing which employees will stay with the company can:

- Reduce turnover costs
  - Hiring and firing employees is expensive
- Increase employee engagement and morale
  - It can be difficult to maintain strong company culture if turnover is high
- Assist with resource planning
  - Companies can add resources to employees they believe will contribute to longterm growth
  - Conversely, companies can take proactive measures to retain at risk employees
- Improve customer relationships
  - Customer perceive companies that retain talent more positively
- Enhance company reputation
  - Company brand is often times tied to the employees interacting with the customers



## ✓ Data

HR Analytics data was collected [here](#). This is a 228 kB, publicly available csv file with 1479 rows and 35 columns from Kaggle. Important features are going to be our target value '**Attrition**', and various features such as:

- Job Satisfaction
- Age
- Sex
- Job Title
- Salary

### Data Preperation

This dataset is already cleaned, but some processing still needs to occur. We will need to:

- Engineer Features
  - Combining, Transforming columns
  - Label Encoding, One-Hot Encoding categorical features
- Dropping Columns
- Balancing Classes
  - The class imbalance issue will be a limitation of this dataset and a tradeoff between overall accuracy and Recall will happen

### Goals

The main goal of our model is to maximize **accuracy**. Accuracy will be important here as being able to accurately predict which employees will stay and which will leave is important. It's also important to understand that since we don't know if employees are fired and which ones quit, simply targeting the positive Attrition class (Recall) will still not tell the entire story.

Here are the metrics we will be looking at for this business case:

- Accuracy-Overall accuracy of our model

- Precision-Accuracy of positive predictions made by our model
  - High Precision indicates that an employee will leave, it is usually correct. **High Precision minimizes false positives.**
- Recall-The ability of our model to identify the actual positive observations
  - High Recall is going to be tough with this dataset specifically since we are battling a **class imbalance** issue. As we will see in our dataset, nearly 15% of our data is the positive **Attrition** class.
- F1-Score that factors in both Precision and Recall

## ✓ Loading in the Data

As mentioned earlier, this is publicly available dataset. Run these cells below to download the data so you can run this notebook anywhere!

We will load in the data and then add our own libraries for manipulation and analysis.



```
1 !pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.14)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.6.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.4)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7)
```

```
1 !kaggle datasets download -d pavansubhasht/ibm-hr-analytics-attrition-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset
License(s): DbCL-1.0
Downloading ibm-hr-analytics-attrition-dataset.zip to /content
 0% 0.00/50.1k [00:00<?, ?B/s]
100% 50.1k/50.1k [00:00<00:00, 43.1MB/s]
```

Importing all of our libraries and modules for data analysis, feature engineering and modeling.

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
4 from sklearn.tree import DecisionTreeClassifier
5 from sklearn.model_selection import train_test_split, GridSearchCV
6 from sklearn.metrics import classification_report, accuracy_score, precision_score, roc_auc_score, recall_score, confusion_matrix, Confus
7 from sklearn.impute import SimpleImputer
8 from sklearn.ensemble import RandomForestClassifier, VotingClassifier
9 from imblearn.over_sampling import SMOTE
10 from imblearn.under_sampling import RandomUnderSampler
11 from imblearn.pipeline import Pipeline
12 from imblearn.combine import SMOTEENN
13 import seaborn as sns
14 import matplotlib.pyplot as plt
15 from sklearn.neighbors import KNeighborsClassifier
```

Reading in our dataset as df.

```
1 !unzip -o ibm-hr-analytics-attrition-dataset.zip
```

```
Archive: ibm-hr-analytics-attrition-dataset.zip
  inflating: WA_Fn-UseC_-HR-Employee-Attrition.csv
```

```
1 Start coding or generate with AI.
```

```
1 import os
2 print(os.listdir())
3
```

```
↳ ['.config', 'ibm-hr-analytics-attrition-dataset.zip', 'WA_Fn-UseC_-HR-Employee-Attrition.csv', 'sample_data']
```

```
1 # Load the dataset into a DataFrame
2 df = pd.read_csv('WA_Fn-UseC_-HR-Employee-Attrition.csv')
3
4 # Display the first few rows of the DataFrame to verify
5 df.head()
6
```

↳

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education
0	41	Yes	Travel_Rarely	1102	Sales	1	2
1	49	No	Travel_Frequently	279	Research & Development	8	1
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2
3	33	No	Travel_Frequently	1392	Research & Development	3	4
4	27	No	Travel_Rarely	591	Research & Development	2	1

5 rows × 35 columns

◀ ▶

```
1 # Load the dataset
2 df = pd.read_csv('WA_Fn-UseC_-HR-Employee-Attrition.csv')
3
4 # Display the first few rows of the dataset
5 print(df.head())
```

↳

	Age	Attrition	BusinessTravel	DailyRate	Department	\
0	41	Yes	Travel_Rarely	1102	Sales	
1	49	No	Travel_Frequently	279	Research & Development	
2	37	Yes	Travel_Rarely	1373	Research & Development	
3	33	No	Travel_Frequently	1392	Research & Development	
4	27	No	Travel_Rarely	591	Research & Development	

	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	\
0	1	2	Life Sciences	1	1	
1	8	1	Life Sciences	1	2	
2	2	2	Other	1	4	
3	3	4	Life Sciences	1	5	
4	2	1	Medical	1	7	

	RelationshipSatisfaction	StandardHours	StockOptionLevel	\
0	...	1	80	0
1	...	4	80	1
2	...	2	80	0
3	...	3	80	0
4	...	4	80	1

	TotalWorkingYears	TrainingTimesLastYear	WorkLifeBalance	YearsAtCompany	\
0	8	0	1	6	
1	10	3	3	10	
2	7	3	3	0	
3	8	3	3	8	
4	6	3	3	2	

	YearsInCurrentRole	YearsSinceLastPromotion	YearsWithCurrManager
0	4	0	5
1	7	1	7
2	0	0	0
3	7	3	0
4	2	2	2

[5 rows x 35 columns]

## EDA

Let's explore the data to enhance our analysis. This is a crucial step in the modeling process as we can add context to the data.

We will perform the following:

- View the info
- Dropping unnecessary columns

- Feature engineering
- Creating Visualizations to better understand the data

Quick view of info. Notice no NaN values in our dataset!

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Age                                    1470 non-null   int64
1   Attrition                            1470 non-null   object
2   BusinessTravel                       1470 non-null   object
3   DailyRate                            1470 non-null   int64
4   Department                           1470 non-null   object
5   DistanceFromHome                    1470 non-null   int64
6   Education                            1470 non-null   int64
7   EducationField                       1470 non-null   object
8   EmployeeCount                       1470 non-null   int64
9   EmployeeNumber                      1470 non-null   int64
10  EnvironmentSatisfaction              1470 non-null   int64
11  Gender                              1470 non-null   object
12  HourlyRate                          1470 non-null   int64
13  JobInvolvement                      1470 non-null   int64
14  JobLevel                            1470 non-null   int64
15  JobRole                             1470 non-null   object
16  JobSatisfaction                     1470 non-null   int64
17  MaritalStatus                       1470 non-null   object
18  MonthlyIncome                       1470 non-null   int64
19  MonthlyRate                         1470 non-null   int64
20  NumCompaniesWorked                  1470 non-null   int64
21  Over18                              1470 non-null   object
22  OverTime                            1470 non-null   object
23  PercentSalaryHike                   1470 non-null   int64
24  PerformanceRating                   1470 non-null   int64
25  RelationshipSatisfaction             1470 non-null   int64
26  StandardHours                       1470 non-null   int64
27  StockOptionLevel                    1470 non-null   int64
28  TotalWorkingYears                   1470 non-null   int64
29  TrainingTimesLastYear               1470 non-null   int64
30  WorkLifeBalance                     1470 non-null   int64
31  YearsAtCompany                      1470 non-null   int64
32  YearsInCurrentRole                  1470 non-null   int64
33  YearsSinceLastPromotion              1470 non-null   int64
34  YearsWithCurrManager                1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

Confirmation of no NaN values

```
1 df.isna().sum()
```

```
Age                                0
Attrition                          0
BusinessTravel                     0
DailyRate                          0
Department                         0
DistanceFromHome                   0
Education                          0
EducationField                      0
EmployeeCount                      0
EmployeeNumber                     0
EnvironmentSatisfaction             0
Gender                             0
HourlyRate                         0
JobInvolvement                     0
JobLevel                           0
JobRole                            0
JobSatisfaction                     0
MaritalStatus                       0
MonthlyIncome                       0
MonthlyRate                         0
NumCompaniesWorked                  0
Over18                             0
OverTime                           0
PercentSalaryHike                   0
PerformanceRating                   0
RelationshipSatisfaction             0
```

```

StandardHours      0
StockOptionLevel   0
TotalWorkingYears  0
TrainingTimesLastYear  0
WorkLifeBalance    0
YearsAtCompany     0
YearsInCurrentRole  0
YearsSinceLastPromotion  0
YearsWithCurrManager  0
dtype: int64

```

These columns below don't offer any value to our statistical analysis, so they can be removed.

```

1 df = df.drop(columns=[
2     'Over18',
3     'EmployeeCount',
4     'StandardHours',
5     'EmployeeNumber'
6
7 ])

```



Let's create a new column that shows the relationship between 2 similar features- 'YearsInCurrentRole' and 'YearsAtCompany'. Because some people will have been at the company for less than a year, we need to turn NaN values to 0, since this will affectively have the same effect we are looking for.

```

1 # Create the new feature with division
2 df['YearsInCurrentRole_vs_YearsAtCompany'] = df['YearsInCurrentRole'] / df['YearsAtCompany']
3
4 # Replace NaN values with 0
5 df['YearsInCurrentRole_vs_YearsAtCompany'].fillna(0, inplace=True)
6
7 # Verify changes
8 print(df[['YearsInCurrentRole', 'YearsAtCompany', 'YearsInCurrentRole_vs_YearsAtCompany']].head())
9

```

	YearsInCurrentRole	YearsAtCompany	YearsInCurrentRole_vs_YearsAtCompany
0	4	6	0.666667
1	7	10	0.700000
2	0	0	0.000000
3	7	8	0.875000
4	2	2	1.000000

## Feature Engineering

Let's create some more new columns to better illustrate the relationship between years in current job status, amount of travel, and overall satisfaction.

```

1 df['TotalYearsCurrentJob'] = df['YearsInCurrentRole'] + df['YearsWithCurrManager']
2
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
2
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education
0	41	Yes	Travel_Rarely	1102	Sales	1	2
1	49	No	Travel_Frequently	279	Research & Development	8	1
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2
3	33	No	Travel_Frequently	1392	Research & Development	3	4
4	27	No	Travel_Rarely	591	Research & Development	2	1

5 rows × 34 columns

Let's take a look at the distribution of the values in our **'Attrition'** column.

Here is the imbalancing mentioned earlier. We can see the majority class 'No' outweighs our minority class 'yes'.

```
1 attrition_counts = df['Attrition'].value_counts(normalize=True) * 100
2
3 # Print the results
4 print(attrition_counts)
```

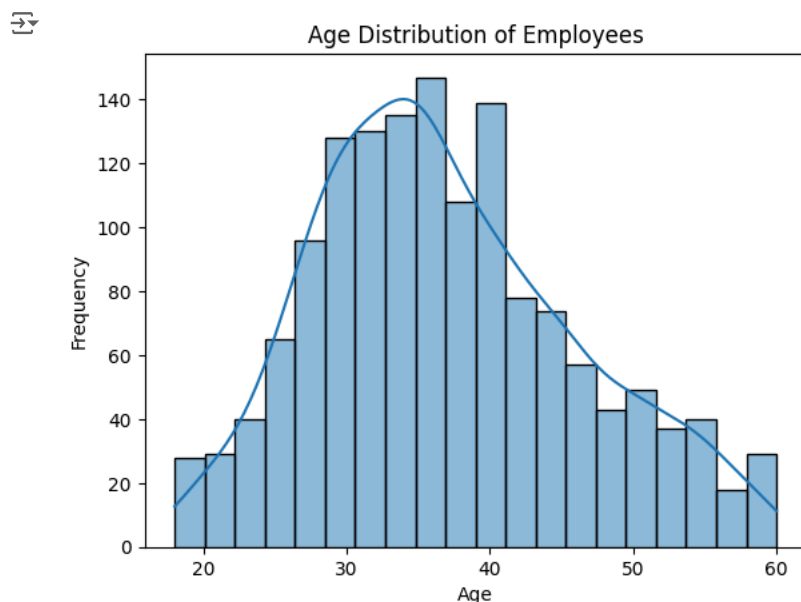
```
Attrition
No      83.877551
Yes     16.122449
Name: proportion, dtype: float64
```

## Visualizations

Now let's look at some visualizations. First things first is the distribution of 'Age' across all employees. We can see it has a slight skew to the right, which makes sense, because most employees tend to be entry level or early in their career and then some older executives will skew the distribution.

We will use the Seaborn library to build most of these visualizations.

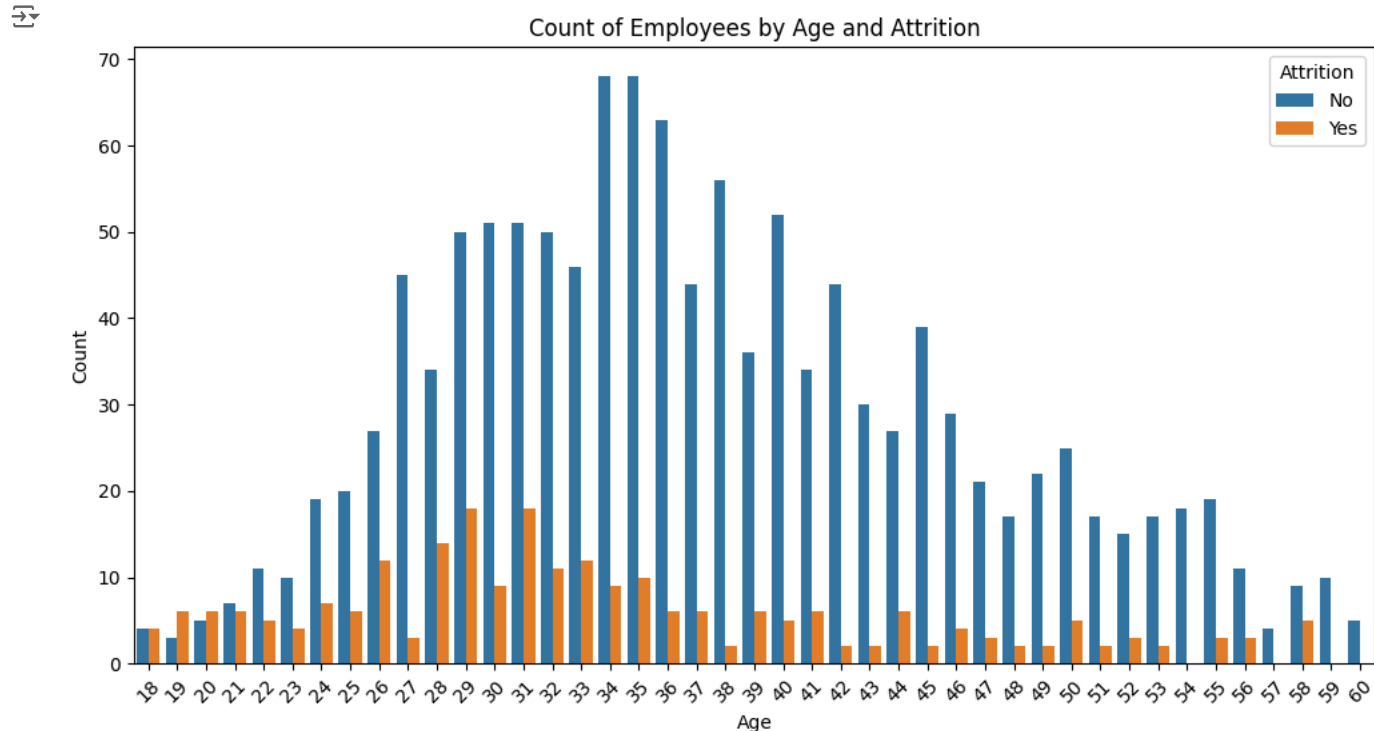
```
1 sns.histplot(df['Age'], bins=20, kde=True)
2 plt.title('Age Distribution of Employees')
3 plt.xlabel('Age')
4 plt.ylabel('Frequency')
5 plt.show()
```



Let's see if there is any relationship between 'Age' and 'Attrition'

Based on what we see, it looks like **mostly employees early in their career or in mid-level positions are leaving** the company and **fewer senior/executive aged employees are leaving**.

```
1 att_age_group = df.groupby(['Attrition', 'Age']) # creating 'att_age_group' for visualizing
2 group_sizes = att_age_group.size()
3
4 # Convert group_sizes to DataFrame for easier manipulation
5 group_sizes_df = group_sizes.reset_index(name='Count')
6
7 # Plotting the bar graph
8 plt.figure(figsize=(12, 6))
9 sns.barplot(data=group_sizes_df, x='Age', y='Count', hue='Attrition')
10 plt.title('Count of Employees by Age and Attrition')
11 plt.xlabel('Age')
12 plt.ylabel('Count')
13 plt.xticks(rotation=45)
14 plt.show()
```



Let's take a quick look at the employees who leave the company based on their 'Gender'. While this column name is 'Gender', it is actually referring to sex.

1st thing to notice is there are **more males than females**.

```
1 att_gen_group = df.groupby(['Attrition', 'Gender'])
2 att_gen_group = att_gen_group.size()
3 print(att_gen_group)
4
```

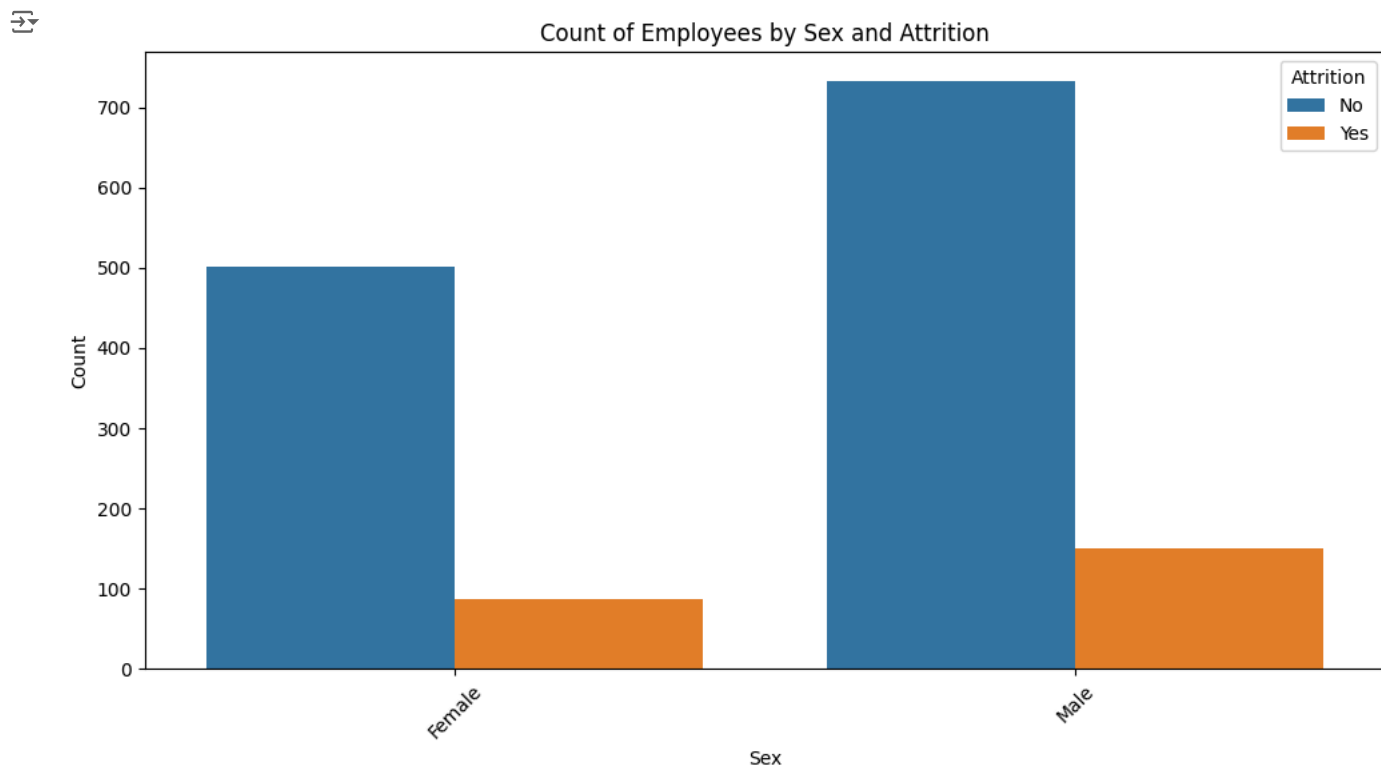
```
Attrition  Gender
No         Female    501
           Male      732
Yes        Female     87
           Male     150
dtype: int64
```

Let's visualize that.

```

1 # Convert group_sizes to DataFrame for easier manipulation
2 att_gen_df = att_gen_group.reset_index(name='Count')
3
4 # Plotting the bar graph
5 plt.figure(figsize=(12, 6))
6 sns.barplot(data=att_gen_df, x='Gender', y='Count', hue='Attrition')
7 plt.title('Count of Employees by Sex and Attrition')
8 plt.xlabel('Sex')
9 plt.ylabel('Count')
10 plt.xticks(rotation=45)
11 plt.show()

```



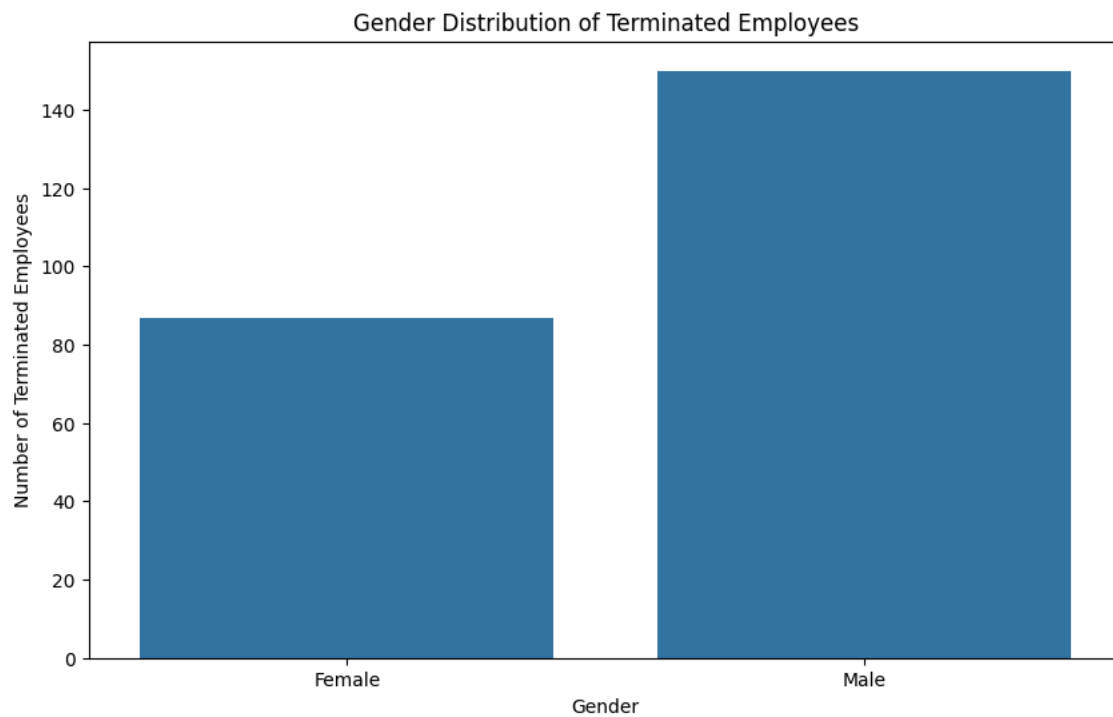
Again, we can visualize, of the sexes, which is leaving the company more and it is Male. We will also create a new **'terminated\_df'** that holds only the rows where the **'Attrition'** value is **Yes**.

```

1 terminated_df = df[df['Attrition'] == 'Yes']
2
3 # Plot the gender distribution of terminated employees
4 plt.figure(figsize=(10, 6))
5 sns.countplot(data=terminated_df, x='Gender')
6 plt.title('Gender Distribution of Terminated Employees')
7 plt.xlabel('Gender')
8 plt.ylabel('Number of Terminated Employees')
9 plt.show()

```



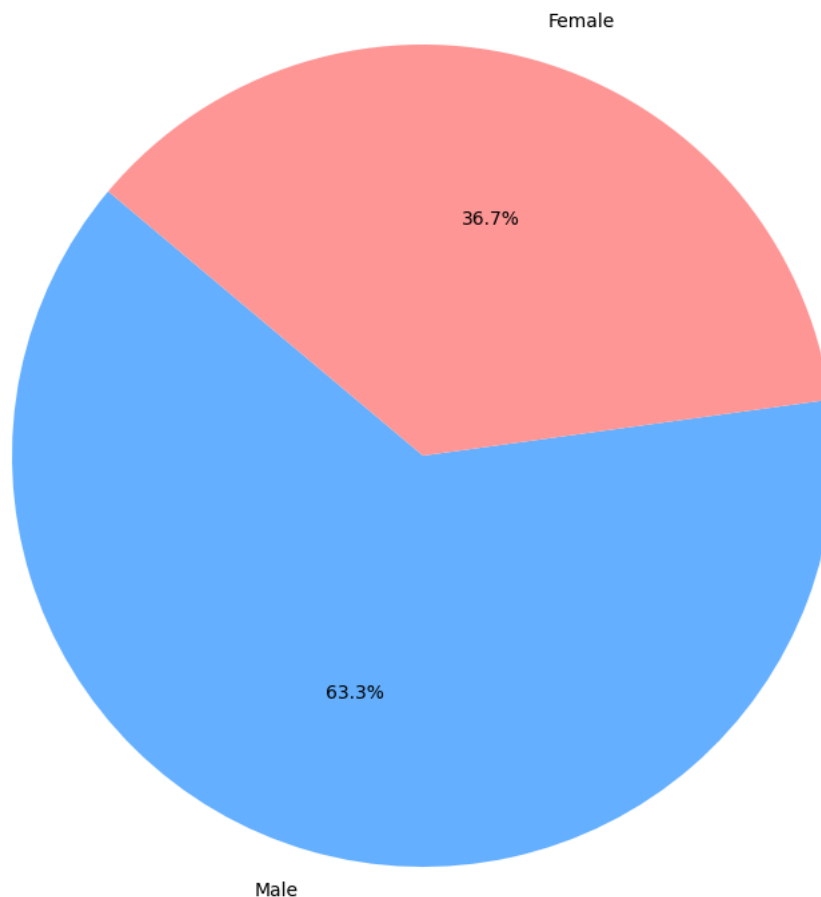


And let's visualize this another way.

```
1 # Calculate the distribution of genders
2 gender_counts = terminated_df['Gender'].value_counts()
3
4 # Plot the gender distribution of terminated employees as a pie chart
5 plt.figure(figsize=(10, 10))
6 plt.pie(gender_counts, labels=gender_counts.index, autopct='%1.1f%%', startangle=140, colors=['#66b3ff', '#ff9999'])
7 plt.title('Gender Distribution of Terminated Employees', fontsize=24)
8
9 # Show the plot
10 plt.show()
```

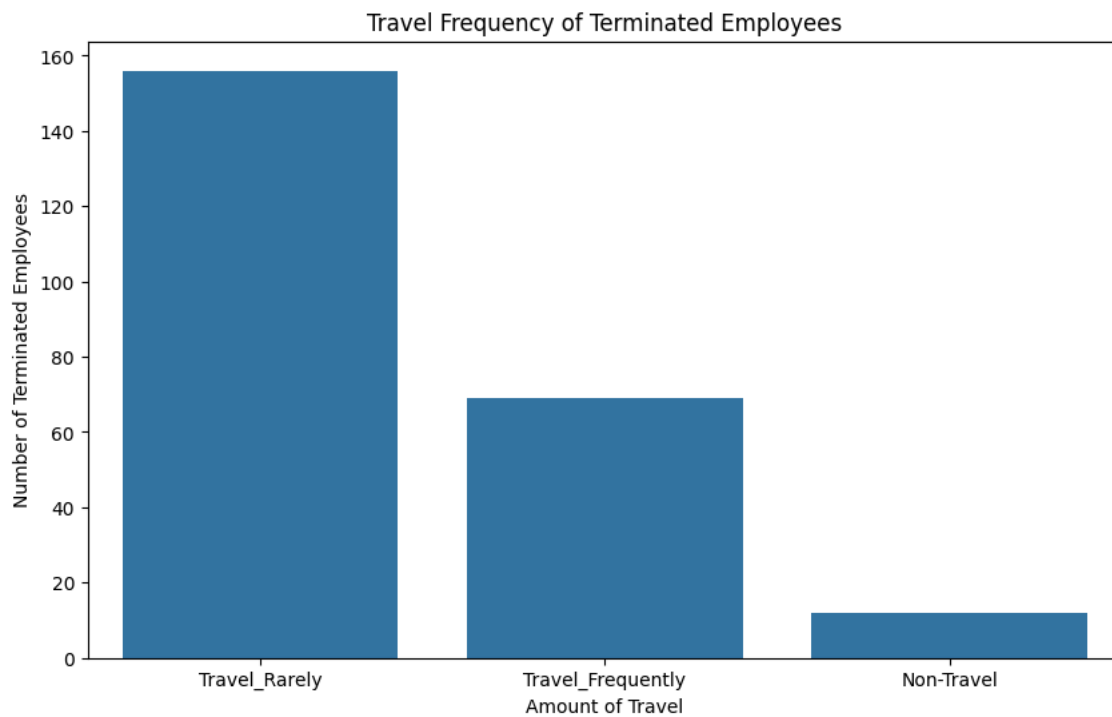


# Gender Distribution of Terminated Employees



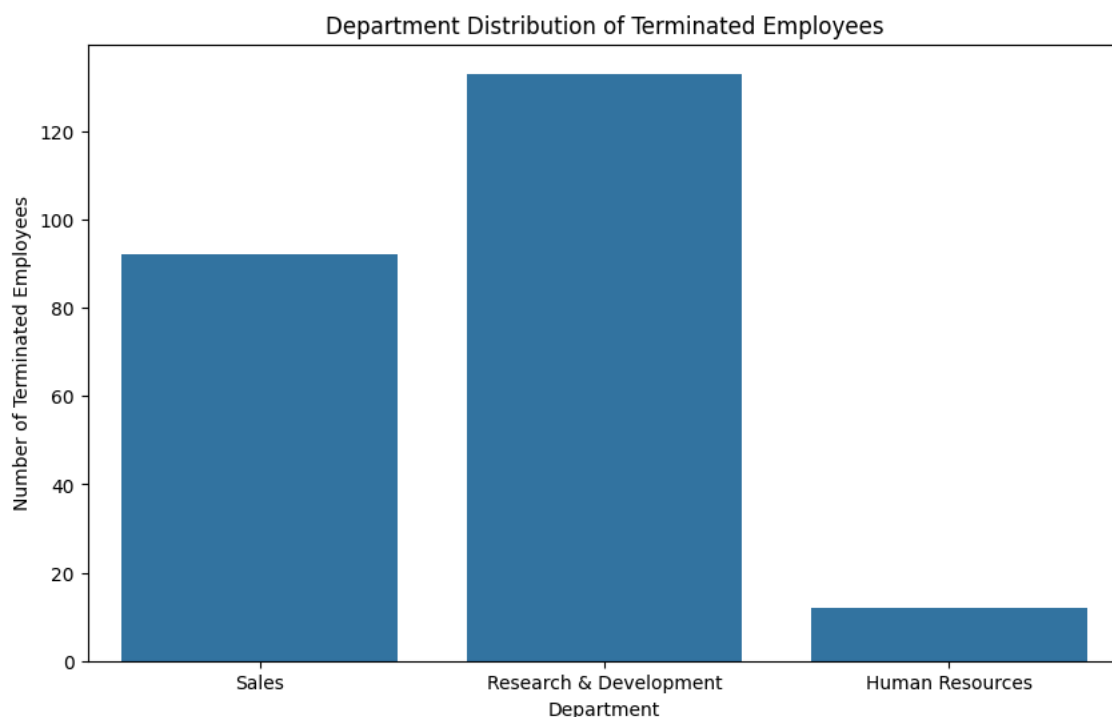
Next, let's take a look at Frequency of travel amongst employees who left the company.

```
1 terminated_df = df[df['Attrition'] == 'Yes']
2
3 # Plot the gender distribution of terminated employees
4 plt.figure(figsize=(10, 6))
5 sns.countplot(data=terminated_df, x='BusinessTravel')
6 plt.title('Travel Frequency of Terminated Employees')
7 plt.xlabel('Amount of Travel')
8 plt.ylabel('Number of Terminated Employees')
9 plt.show()
```



Now let's take a look at the distribution of terminated employees based on the department they work in. The **Research and Development Department has seen the most amount of employees who 'Attrition'.**

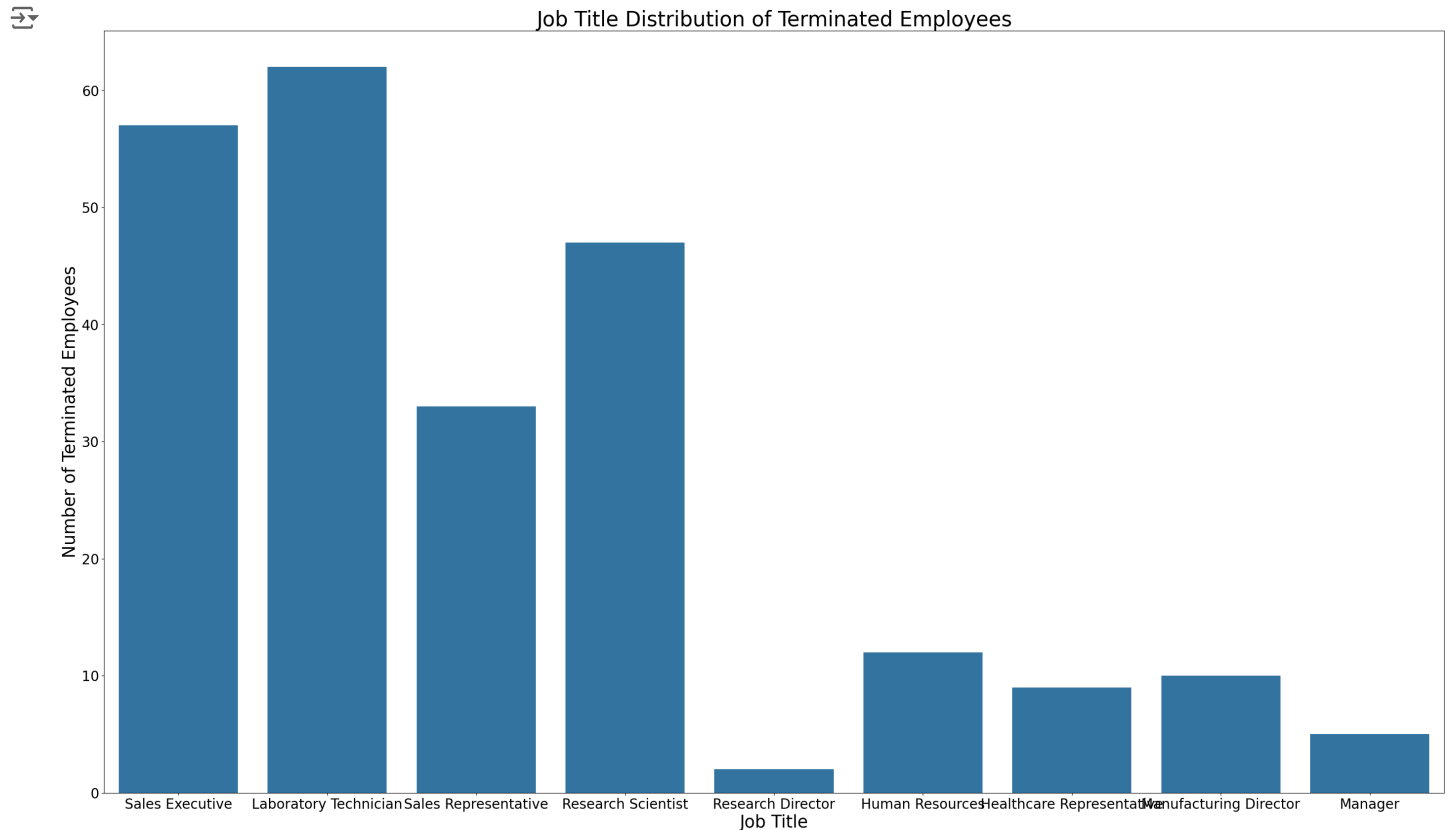
```
1 terminated_df = df[df['Attrition'] == 'Yes']
2
3 # Plot the gender distribution of terminated employees
4 plt.figure(figsize=(10, 6))
5 sns.countplot(data=terminated_df, x='Department')
6 plt.title('Department Distribution of Terminated Employees')
7 plt.xlabel('Department')
8 plt.ylabel('Number of Terminated Employees')
9 plt.show()
```



Now let's take a look at the **distribution of employees who leave the company based on their job title**. Below are the top 3 titles of employees of leave the company.

- Lab Tech
- Sales Exec
- Research Scientist

```
1 # Plot the gender distribution of terminated employees
2 plt.figure(figsize=(35, 20))
3 sns.countplot(data=terminated_df, x='JobRole')
4 plt.title('Job Title Distribution of Terminated Employees', fontsize=30)
5 plt.xlabel('Job Title', fontsize=25)
6 plt.ylabel('Number of Terminated Employees', fontsize=25)
7 plt.xticks(fontsize=20)
8 plt.yticks(fontsize=20)
9 plt.show()
```



Next, let's create a **pie chart** to view that same data above, but as percentages of the whole.

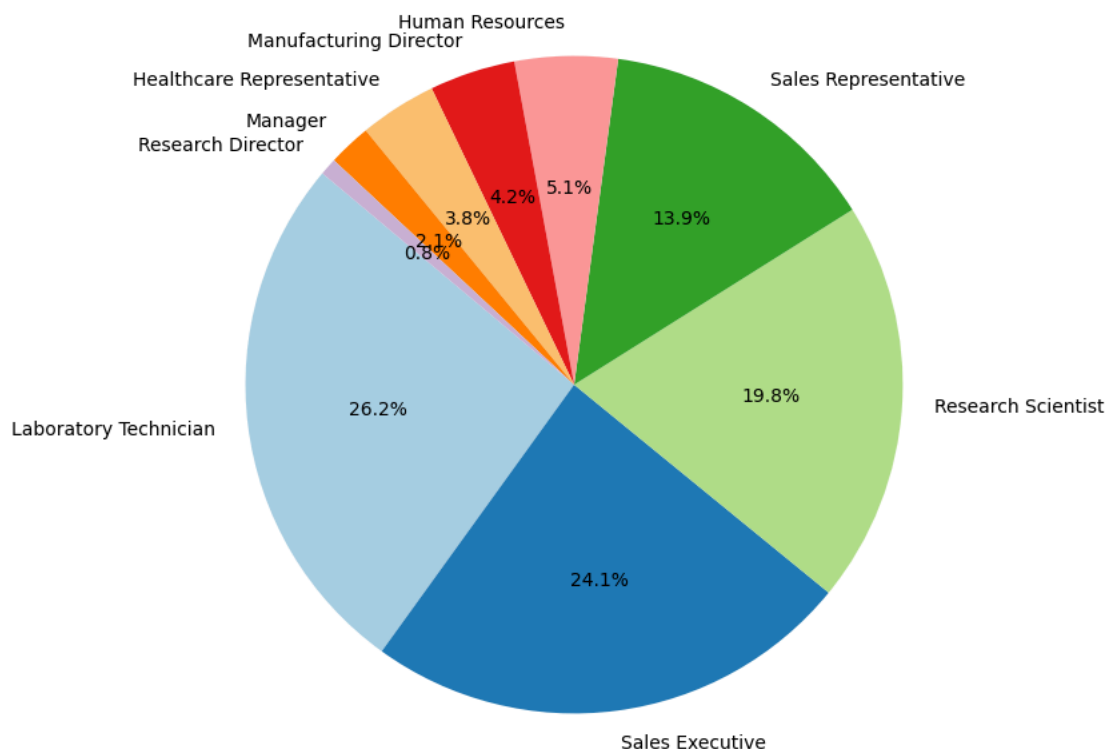
```

1 # Calculate the distribution of job roles
2 jobrole_counts = terminated_df['JobRole'].value_counts()
3
4 # Plot the job role distribution of terminated employees as a pie chart
5 plt.figure(figsize=(12, 8))
6 plt.pie(jobrole_counts, labels=jobrole_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
7 plt.title('Job Role Distribution of Terminated Employees', fontsize=24)
8
9 # Show the plot
10 plt.show()

```



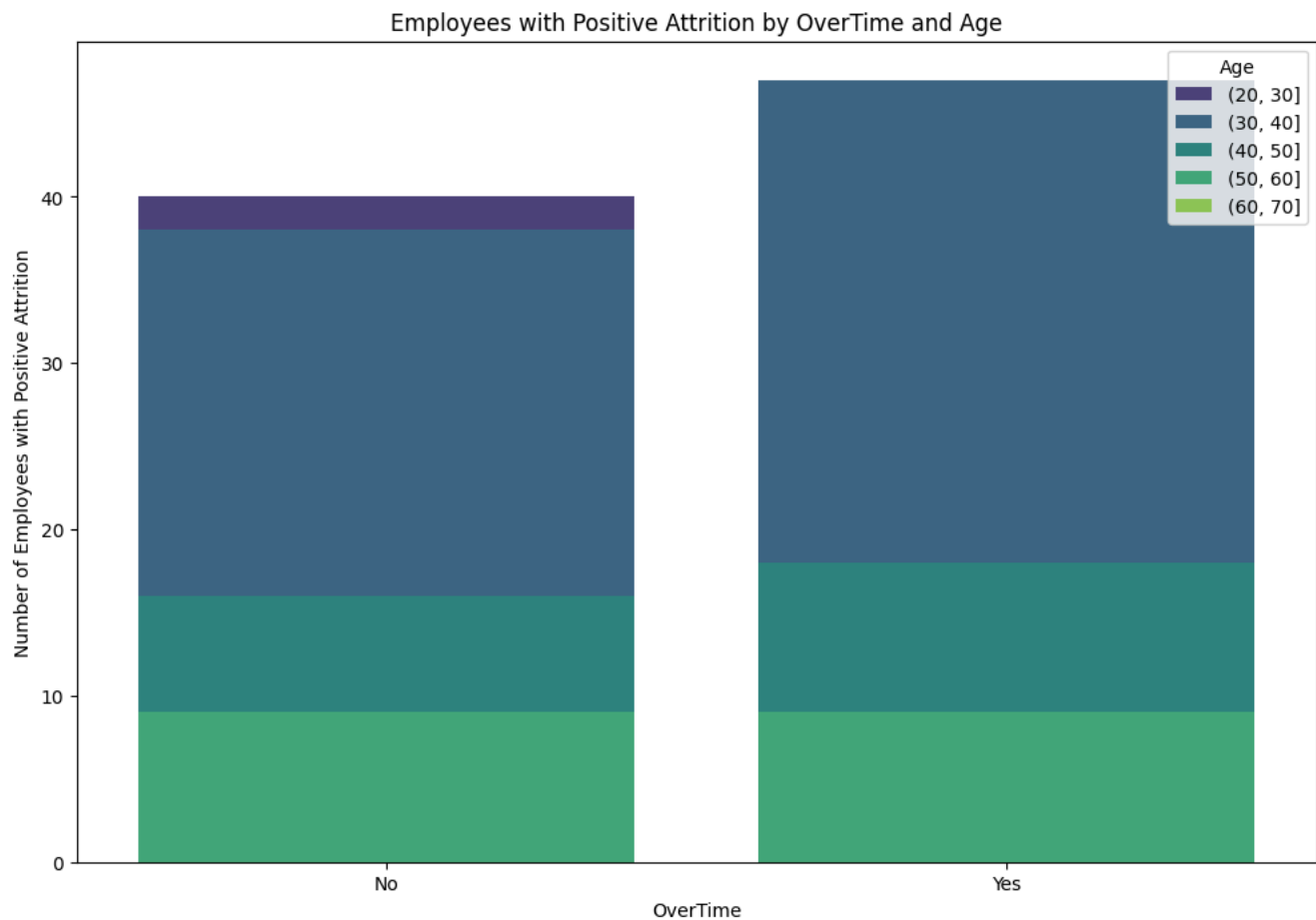
## Job Role Distribution of Terminated Employees



```

1 attrition_overtime_counts = terminated_df.groupby(['OverTime', pd.cut(terminated_df['Age'], bins=[20, 30, 40, 50, 60, 70])]).size().reset
2
3 # Plotting the bar chart
4 plt.figure(figsize=(12, 8))
5 sns.barplot(data=attrition_overtime_counts, x='OverTime', y='Count', hue='Age', palette='viridis', dodge=False)
6 plt.xlabel('OverTime')
7 plt.ylabel('Number of Employees with Positive Attrition')
8 plt.title('Employees with Positive Attrition by OverTime and Age')
9 plt.legend(title='Age', loc='upper right')
10 plt.show()

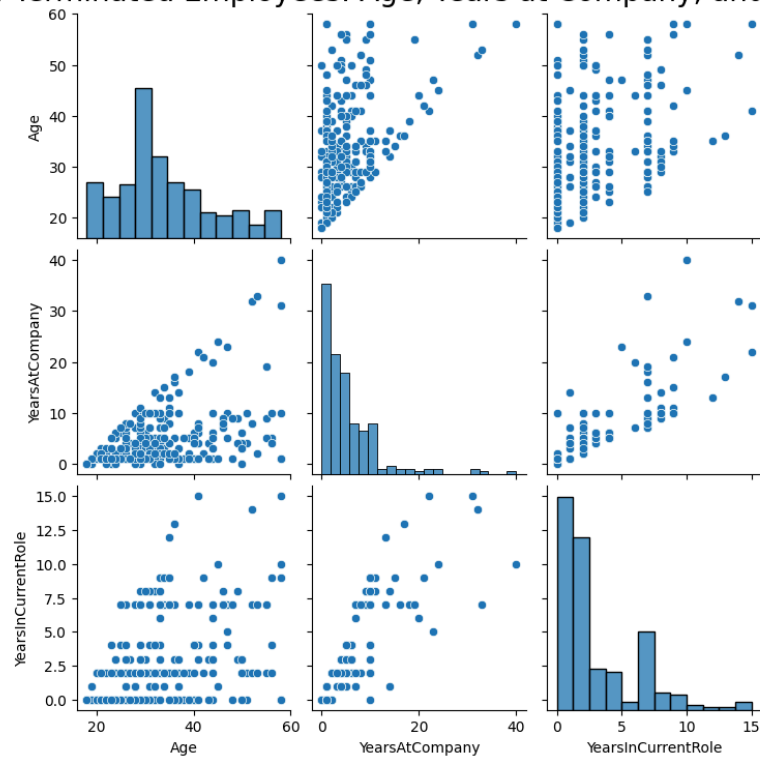
```



Let's use pairplot from the Seaborn library to view a variety of graphs.

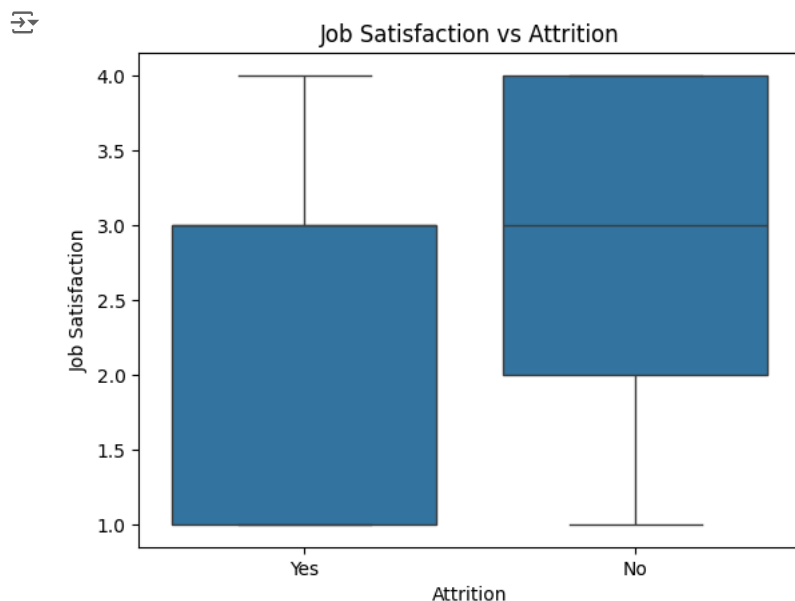
```
1 # Plot using pairplot
2 sns.pairplot(terminated_df[['Age', 'YearsAtCompany', 'YearsInCurrentRole']])
3 plt.suptitle('Scatter Plot Matrix of Terminated Employees: Age, Years at Company, and Years in Current Role', y=1.02, fontsize=20)
4 plt.show()
```

## Scatter Plot Matrix of Terminated Employees: Age, Years at Company, and Years in Current Role



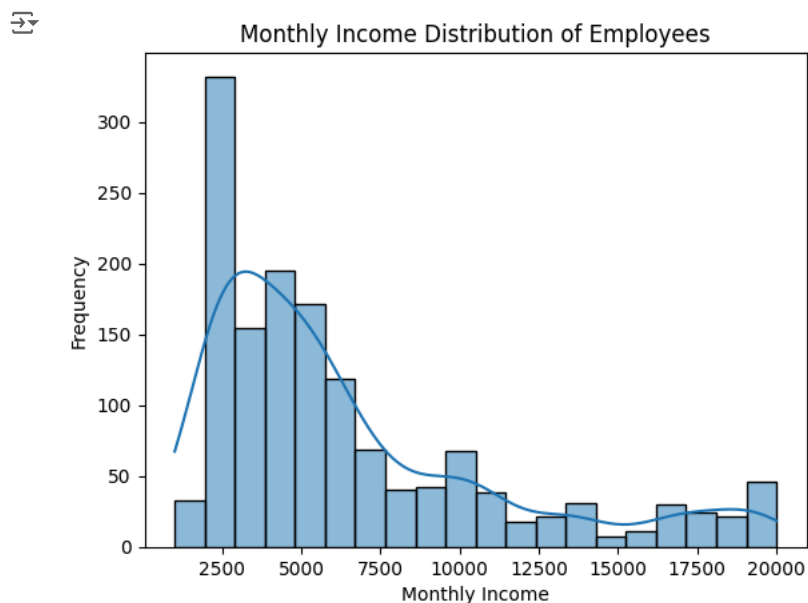
We can see '**Attrition**' based on '**Job Satisfaction**' and how most of the employees who leave the company have lower ratings of 'Job Satisfaction'.

```
1 sns.boxplot(data=df, x='Attrition', y='JobSatisfaction')
2 plt.title('Job Satisfaction vs Attrition')
3 plt.xlabel('Attrition')
4 plt.ylabel('Job Satisfaction')
5 plt.show()
```



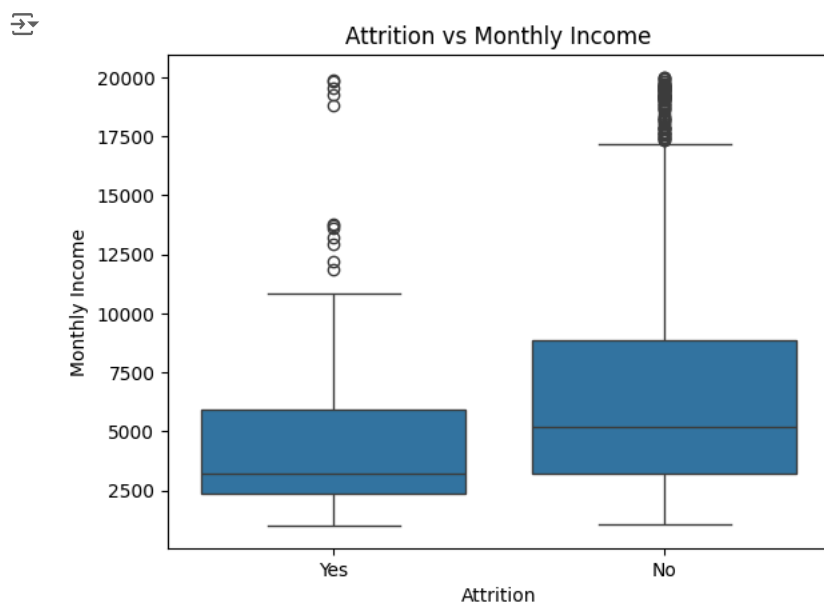
Next, let's take a look at **'Monthly Income'**. This is really just to get an idea of the distribution across the entire dataset.

```
1 sns.histplot(df['MonthlyIncome'], bins=20, kde=True)
2 plt.title('Monthly Income Distribution of Employees')
3 plt.xlabel('Monthly Income')
4 plt.ylabel('Frequency')
5 plt.show()
```



Now let's visualize that data as it relates to **'Attrition'**. We can see employees who leave the company tend to have lower **'Monthly Income'**.

```
1 sns.boxplot(data=df, x='Attrition', y='MonthlyIncome')
2 plt.title('Attrition vs Monthly Income')
3 plt.xlabel('Attrition')
4 plt.ylabel('Monthly Income')
5 plt.show()
```



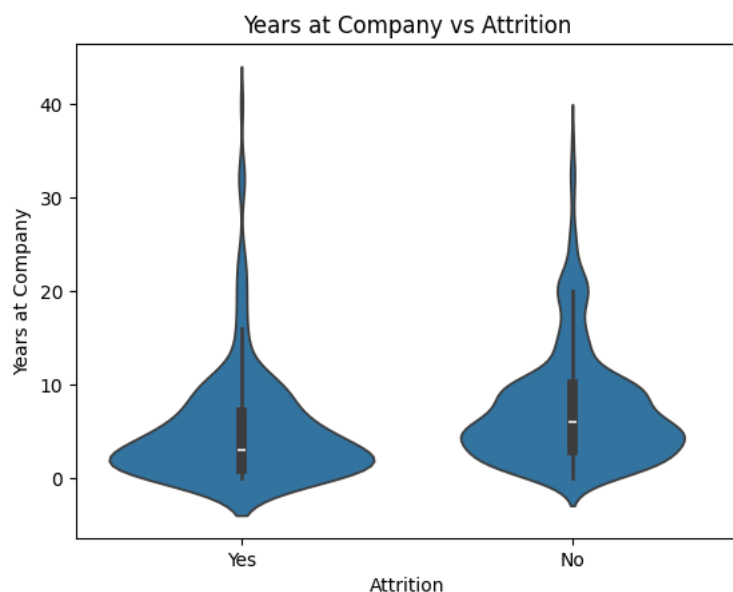
This below now looks at **'Attrition' vs Years at Company'**. It's interesting how similar these 2 graphs for **'Monthly Income'** and **'Years at Company'** are. This all leads us to believe that fewer employees are sr level/executive and most are junior level or mid level.



```

1 sns.violinplot(data=df, x='Attrition', y='YearsAtCompany')
2 plt.title('Years at Company vs Attrition')
3 plt.xlabel('Attrition')
4 plt.ylabel('Years at Company')
5 plt.show()

```



let's take a look at **'Work-Life Balance'**, **'Environment Satisfaction'**, and then one of our engineered features **'Environment/Job Satisfaction'**. Overall, it looks like these features don't seem to indicate if an employee leaves the company or not.

```

1 sns.countplot(data=df, x='WorkLifeBalance', hue='Attrition')
2 plt.title('Work-Life Balance vs Attrition')
3 plt.xlabel('Work-Life Balance')
4 plt.ylabel('Number of Employees')
5 plt.show()

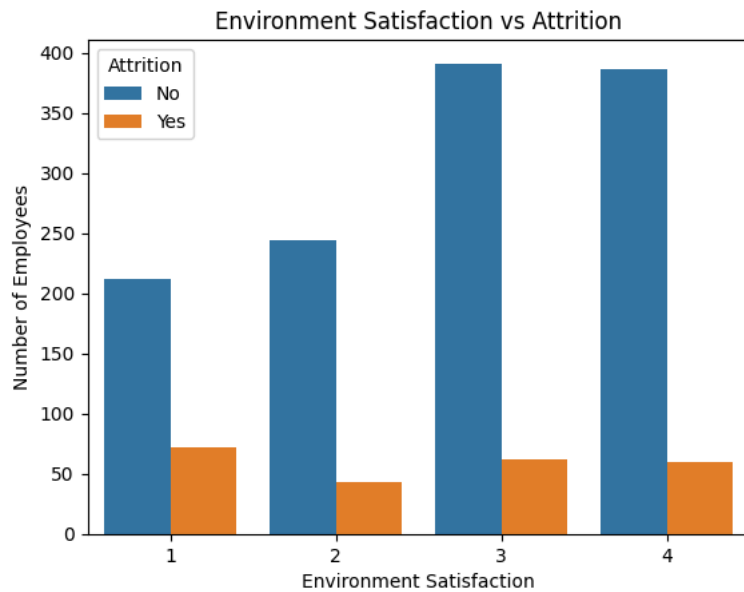
```



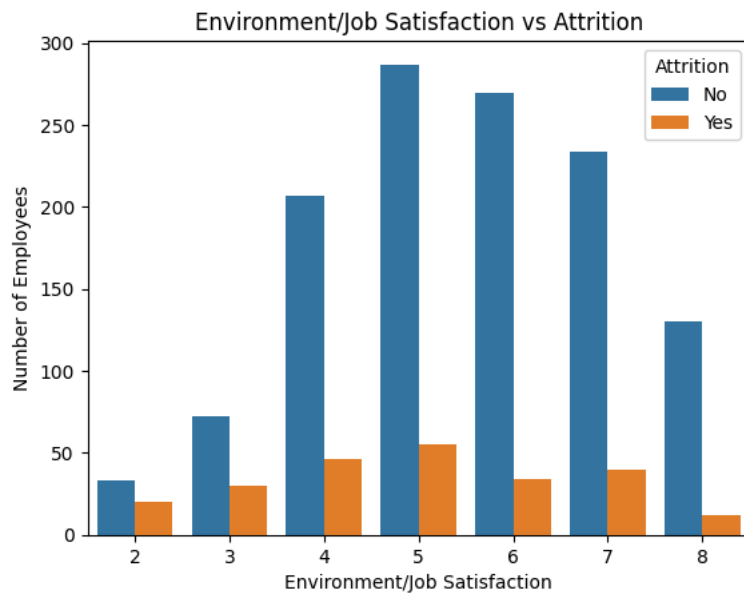
```

1 sns.countplot(data=df, x='EnvironmentSatisfaction', hue='Attrition')
2 plt.title('Environment Satisfaction vs Attrition')
3 plt.xlabel('Environment Satisfaction')
4 plt.ylabel('Number of Employees')
5 plt.show()

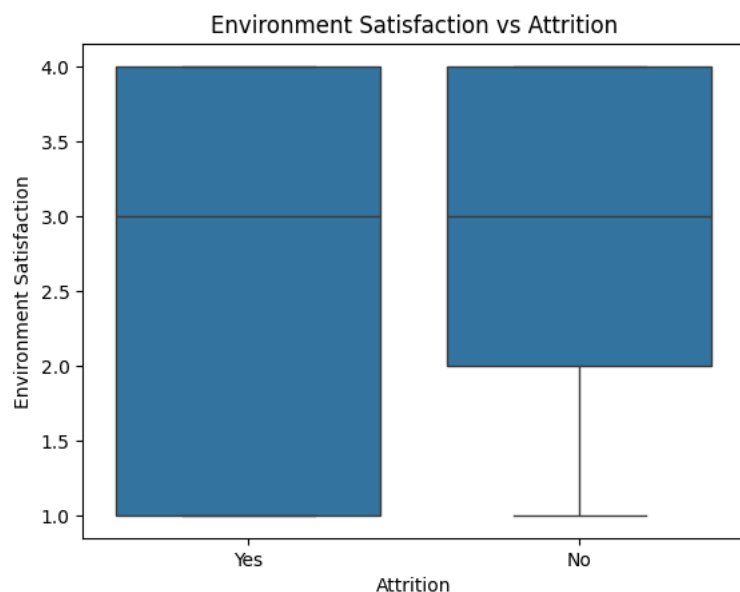
```



```
1 sns.countplot(data=df, x='JobEnvSatisfaction', hue='Attrition')
2 plt.title('Environment/Job Satisfaction vs Attrition')
3 plt.xlabel('Environment/Job Satisfaction')
4 plt.ylabel('Number of Employees')
5 plt.show()
6
```



```
1 sns.boxplot(data=df, x='Attrition', y='EnvironmentSatisfaction')
2 plt.title('Environment Satisfaction vs Attrition')
3 plt.xlabel('Attrition')
4 plt.ylabel('Environment Satisfaction')
5 plt.show()
```



```

1 # Filter the data to include only employees with positive attrition
2 df_attrition = df[df['Attrition'] == 'Yes']
3
4 # Count the number of employees with positive attrition based on 'OverTime'
5 attrition_overtime_counts = df_attrition['OverTime'].value_counts()
6
7 # Plotting the bar chart
8 plt.figure(figsize=(10, 6))
9 sns.barplot(x=attrition_overtime_counts.index, y=attrition_overtime_counts.values, palette='viridis')
10 plt.xlabel('OverTime')
11 plt.ylabel('Number of Employees with Positive Attrition')
12 plt.title('Employees with Positive Attrition and OverTime')
13 plt.show()

```

Ok, so here is what we learned:

- Most employees are junior or mid level
- Most employees are under 40 and most employees who leave the company are under 40
- Mostly males work here and are therefore more often leaving the company
- Most employees leaving rarely travel
- Research and Sales departments experience more Attrition
- Job Satisfaction has minimal impact on Attrition
- Employees making less money are more likely to experience Attrition
- Less experienced employees are more likely to experience Attrition
- Work Life Balance has minimal impact on Attrition

Alright, we're almost ready to model. Let's take a look at the data one more time. As we can see, we need to encode our categorical features before we move on to the modeling. Let's use **LabelEncoder and One-Hot Encoding**.

```

1 df.head()

```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfaction	Gender
0	41	Yes	Travel_Rarely	1102	Sales		1	2	Life Sciences	2 Female
1	49	No	Travel_Frequently	279	Research & Development		8	1	Life Sciences	3 Male
2	37	Yes	Travel_Rarely	1373	Research & Development		2	2	Other	4 Male
3	33	No	Travel_Frequently	1392	Research & Development		3	4	Life Sciences	4 Female
4	27	No	Travel_Rarely	591	Research & Development		2	1	Medical	1 Male

5 rows × 34 columns

**Label Encoder** is great when there is an ordinal relationship in the data such as travel or overtime or Attrition.

```

1 label_encoder = LabelEncoder()
2
3 columns_to_encode = [
4     'BusinessTravel',
5     'OverTime',
6     'Attrition'
7 ]
8
9 for column in columns_to_encode:
10 #     df.loc[:, column] = label_encoder.fit_transform(df[column]).astype(int)
11     df[column] = label_encoder.fit_transform(df[column])
12     df[column] = pd.to_numeric(df[column], errors='coerce').astype(int)
13 # Print df_clean to verify the encoding
14 df.head()
15 df.dtypes

```

Age	int64
Attrition	int64
BusinessTravel	int64
DailyRate	int64
Department	object
DistanceFromHome	int64
Education	int64
EducationField	object
EnvironmentSatisfaction	int64
Gender	object
HourlyRate	int64
JobInvolvement	int64
JobLevel	int64
JobRole	object
JobSatisfaction	int64
MaritalStatus	object
MonthlyIncome	int64
MonthlyRate	int64
NumCompaniesWorked	int64
OverTime	int64
PercentSalaryHike	int64
PerformanceRating	int64
RelationshipSatisfaction	int64
StockOptionLevel	int64
TotalWorkingYears	int64
TrainingTimesLastYear	int64
WorkLifeBalance	int64
YearsAtCompany	int64
YearsInCurrentRole	int64
YearsSinceLastPromotion	int64
YearsWithCurrManager	int64
YearsInCurrentRole_vs_YearsAtCompany	float64
TotalYearsCurrentJob	int64
JobEnvSatisfaction	int64
dtype:	object

And One-Hot Encoding is best when there is no ordinal relationship amongst the values, like in Department or Job Role.

```
1 columns_to_encode = ['Department',
2                       'EducationField',
3                       'Gender',
4                       'JobRole',
5                       'MaritalStatus']
6
7 # Create an instance of OneHotEncoder
8 ohe = OneHotEncoder(drop='first', sparse_output=False)
9
10 # Fit and transform the data
11 encoded_data = ohe.fit_transform(df[columns_to_encode])
12
13 # Create a DataFrame with the encoded data
14 encoded_df = pd.DataFrame(encoded_data, columns=ohe.get_feature_names_out(columns_to_encode))
15
16 # Drop the original columns and concatenate the encoded columns
17 df_encoded = df.drop(columns_to_encode, axis=1).reset_index(drop=True)
18 encoded_df = encoded_df.reset_index(drop=True)
19 df_cleaned = pd.concat([df_encoded, encoded_df], axis=1)
20
21 # Display the DataFrame
22 df_cleaned
```



	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobLev
0	41	1	2	1102	1	2	2	94	3	
1	49	0	1	279	8	1	3	61	2	
2	37	1	2	1373	2	2	4	92	2	
3	33	0	1	1392	3	4	4	56	3	
4	27	0	2	591	2	1	1	40	3	
...	...	...	...	...	...	...	...	...	...	...
1465	36	0	1	884	23	2	3	41	4	
1466	39	0	2	613	6	1	4	42	2	
1467	27	0	2	155	4	3	2	87	4	
1468	49	0	1	1023	2	3	4	63	2	
1469	34	0	2	628	8	3	2	82	4	

1470 rows × 47 columns

One thing to note. With all imbalanced datasets, we need to verify that balancing the classes **Actually** improves our model. As noted in this [Oxford Academic Article](#), sometimes adjusting for **class imbalances** can have a detrimental impact on modeling.

```
1 # Initialize the scaler
2 scaler = StandardScaler()
3
4 # Select numeric columns to scale
5 numeric_columns = df_cleaned.select_dtypes(include=['int64', 'float64']).columns
6 numeric_columns = numeric_columns.drop('Attrition', errors='ignore') # Exclude 'Attrition'
7
8 # Apply the scaler to the numeric columns
9 df_cleaned[numeric_columns] = scaler.fit_transform(df_cleaned[numeric_columns])
```

Let's take a look at correlations in our dataset.

```
1 df_cleaned.corr()
```



	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	EnvironmentSatisfac
Age	1.000000	-0.159205	0.024751	0.010661	-0.001686	0.208034	0.0
Attrition	-0.159205	1.000000	0.000074	-0.056652	0.077924	-0.031373	-0.1
BusinessTravel	0.024751	0.000074	1.000000	-0.004086	-0.024469	0.000757	0.0
DailyRate	0.010661	-0.056652	-0.004086	1.000000	-0.004985	-0.016806	0.0
DistanceFromHome	-0.001686	0.077924	-0.024469	-0.004985	1.000000	0.021042	-0.0
Education	0.208034	-0.031373	0.000757	-0.016806	0.021042	1.000000	-0.0
EnvironmentSatisfaction	0.010146	-0.103369	0.004174	0.018355	-0.016075	-0.027128	0.0
HourlyRate	0.024287	-0.006846	0.026528	0.023381	0.031131	0.016775	-0.0
JobInvolvement	0.029820	-0.130016	0.039062	0.046135	0.008783	0.042438	-0.0
JobLevel	0.509604	-0.169105	0.019311	0.002966	0.005303	0.101589	0.0
JobSatisfaction	-0.004892	-0.103481	-0.033962	0.030571	-0.003669	-0.011296	-0.0
MonthlyIncome	0.497855	-0.159840	0.034319	0.007707	-0.017014	0.094961	-0.0
MonthlyRate	0.028051	0.015170	-0.014107	-0.032182	0.027473	-0.026084	0.0
NumCompaniesWorked	0.299635	0.043494	0.020875	0.038153	-0.029251	0.126317	0.0
OverTime	0.028062	0.246118	0.016543	0.009135	0.025514	-0.020322	0.0
PercentSalaryHike	0.003634	-0.013478	-0.029377	0.022704	0.040235	-0.011111	-0.0
PerformanceRating	0.001904	0.002889	-0.026341	0.000473	0.027110	-0.024539	-0.0
RelationshipSatisfaction	0.053535	-0.045872	-0.035986	0.007846	0.006557	-0.009118	0.0
StockOptionLevel	0.037510	-0.137145	-0.016727	0.042143	0.044872	0.018422	0.0
TotalWorkingYears	0.680381	-0.171063	0.034226	0.014515	0.004628	0.148280	-0.0
TrainingTimesLastYear	-0.019621	-0.059478	0.015240	0.002453	-0.036942	-0.025100	-0.0
WorkLifeBalance	-0.021490	-0.063939	-0.011256	-0.037848	-0.026556	0.009819	0.0
YearsAtCompany	0.311309	-0.134392	-0.014575	-0.034055	0.009508	0.069114	0.0
YearsInCurrentRole	0.212901	-0.160545	-0.011497	0.009932	0.018845	0.060236	0.0
YearsSinceLastPromotion	0.216513	-0.033019	-0.032591	-0.033229	0.010029	0.054254	0.0
YearsWithCurrManager	0.202089	-0.156199	-0.022636	-0.026363	0.014406	0.069065	-0.0
YearsInCurrentRole_vs_YearsAtCompany	-0.007213	-0.130324	-0.006886	0.048004	0.017989	0.023963	0.0
TotalYearsCurrentJob	0.224158	-0.171075	-0.018388	-0.008724	0.017975	0.069792	0.0
JobEnvSatisfaction	0.003681	-0.146763	-0.021255	0.034752	-0.013970	-0.027213	0.7
Department_Research & Development	0.017883	-0.085293	0.002598	0.014871	-0.008117	-0.018604	0.0
Department_Sales	-0.027549	0.080855	-0.006534	-0.003616	0.014085	0.014215	-0.0
EducationField_Life Sciences	0.016824	-0.032703	-0.023132	0.004028	-0.024499	0.013184	-0.0
EducationField_Marketing	0.038162	0.055781	0.037568	-0.064449	0.039294	0.072405	0.0
EducationField_Medical	-0.006354	-0.046999	-0.008520	0.034202	0.013486	-0.072335	-0.0
EducationField_Other	-0.041466	-0.017898	0.018654	-0.003893	-0.007969	0.038043	0.0
EducationField_Technical Degree	-0.027604	0.069355	0.010059	0.030869	-0.014802	-0.026742	0.0
Gender_Male	-0.036311	0.029453	-0.032981	-0.011716	-0.001851	-0.016547	0.0
JobRole_Human Resources	-0.029856	0.036215	0.013346	-0.021156	-0.024089	-0.005295	-0.0
JobRole_Laboratory Technician	-0.143176	0.098290	-0.014328	-0.006728	0.012369	-0.063566	-0.0
JobRole_Manager	0.294248	-0.083316	0.012221	-0.013224	-0.039190	0.028453	0.0
JobRole_Manufacturing Director	0.049726	-0.082994	0.006567	-0.005302	0.011848	-0.005290	0.0
JobRole_Research Director	0.185891	-0.088870	0.033365	-0.000021	-0.022351	0.049694	-0.0
JobRole_Research Scientist	-0.146518	-0.000360	0.011829	-0.002624	-0.010986	0.000709	0.0
JobRole_Sales Executive	-0.002001	0.019774	-0.022251	-0.000513	0.030761	0.053398	-0.0
JobRole_Sales Representative	-0.175785	0.157234	-0.001866	0.005375	-0.015994	-0.091465	0.0

<b>MaritalStatus_Married</b>	0.083919	-0.090984	0.057808	0.040035	0.030232	-0.001865	-0.0
<b>MaritalStatus_Single</b>	-0.119185	0.175419	-0.012097	-0.075835	-0.027445	0.004168	0.0

47 rows × 47 columns

**Feature Scaling** is not necessary for Decision Tree Classifiers (our baseline model), but this is still a valuable tool for models built using K-Nearest Neighbors.

## Modeling

Let's begin our iterative process to build the best model for our data. 1st thing we will need to do is create our train and test sets. Before that, let's discuss our plan.

We will end up building **20 models!** Here is a quick synopsis of our methodology:

- baseline model
- baseline model addressing **class imbalance**
- Baseline model addressing **class imbalance** with GridSearchCV performed to optimize hyperparameters
- model with unimportant features dropped
- model with unimportant features dropped addressing **class imbalance**
- model with unimportant features dropped addressing **class imbalance** with GridSearchCV performed to optimize hyperparameters

This methodology will be applied to all 3 types of models we are building:

- Decision Tree
- Random Forest
- K-Nearest Neighbors

Then we will read in all results and pick the 4 best models. Those 4 best models will be used to create a **Stacking Ensemble** and then a **Stacking Ensemble with GridSearchCV** to finetune hyperparameters. Once our best model is created, we can discuss results.

### Decision Tree

A Decision Tree is a supervised learning approach that is great for binary classifications because of the way the model splits the features based on simple rules to detect patterns and also because of the way the model handles outliers.

More documentation on Decision Trees can be found [here](#).

1st thing we need to do is create our X and y variables and then build our train and test sets.

```
1 # Separate target and features
2 X = df_cleaned.drop('Attrition', axis=1) # all columns except the target column our our independent variables
3 y = df_cleaned['Attrition']
4
5 # Split the data into train and test sets
6 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True, stratify=y)
```

For our first model, we will build a Decision Tree Classifier. These are great for **Binary Classifications** because we can gather **Feature Importances** and they are great at handling **non-linear data**.

This model will be built using our original dataset **before we address the class imbalancing**.

```

1 clf_tree =DecisionTreeClassifier(random_state=42)
2 clf_tree.fit(X_train, y_train)
3
4 y_pred_tree = clf_tree.predict(X_test)
5
6
7 # Evaluate the classifier
8 print("\nAccuracy Score:")
9 print(accuracy_score(y_test, y_pred_tree))
10 print("\nClassification Report:")
11 print(classification_report(y_test, y_pred_tree))
12
13 print("\nAccuracy Score:")
14 print(accuracy_score(y_test, y_pred_tree))
15
16 print("\nConfusion Matrix:")
17 cm_tree = confusion_matrix(y_test, y_pred_tree)
18 print(cm_tree)
19
20 # Plot the confusion matrix
21 ConfusionMatrixDisplay(confusion_matrix=cm_tree, display_labels=clf_tree.classes_).plot()
22 plt.show()

```



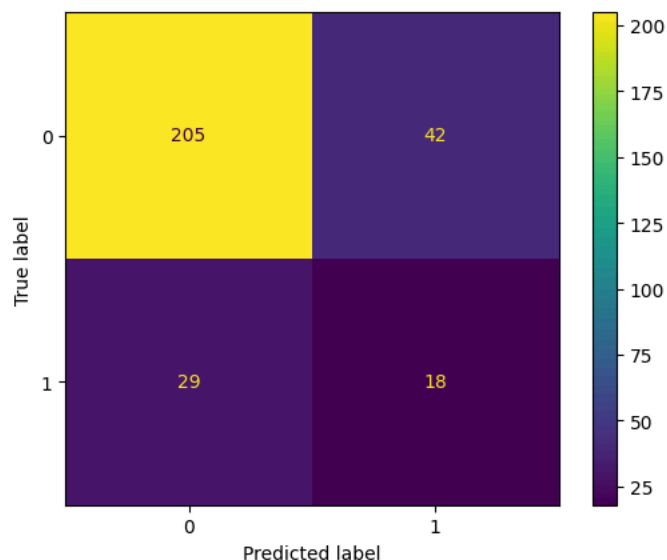
Accuracy Score:  
0.7585034013605442

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.83	0.85	247
1	0.30	0.38	0.34	47
accuracy			0.76	294
macro avg	0.59	0.61	0.59	294
weighted avg	0.78	0.76	0.77	294

Accuracy Score:  
0.7585034013605442

Confusion Matrix:  
[[205 42]  
 [ 29 18]]



Now we need to address our **Class Imbalance**. As we saw earlier, our minority class is roughly 16% of our data. In a perfect world, the data would be evenly split 50/50. However, we can still work with this data and use sampling techniques to improve our results. We will combine SMOTE oversampling and Undersampling with stratification. Undersampling with stratification reduces the number of samples in the majority class, while maintaining the class distribution. This helps create a more balanced dataset and should improve modeling.

75% accuracy is not a bad start. We can definitely improve this though!

Now let's build a **decision tree with our balanced data**.



```

1 # Initialize SMOTE and RandomUnderSampler
2 smote = SMOTE(sampling_strategy=.5, random_state=42) # Adjust sampling_strategy as needed
3 under_sampler = RandomUnderSampler(sampling_strategy=1, random_state=42, replacement=False) # Adjust sampling_strategy as needed
4
5 # Apply SMOTE to the training data
6 X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
7
8 # Apply RandomUnderSampler to the training data
9 X_train_balanced, y_train_balanced = under_sampler.fit_resample(X_train_smote, y_train_smote)
10 # Print the class distribution after resampling
11 print("Class distribution before resampling:")
12 print(y_train.value_counts())
13
14 print("\nClass distribution after SMOTE:")
15 print(y_train_smote.value_counts())
16
17 print("\nClass distribution after SMOTE and undersampling:")
18 print(y_train_balanced.value_counts())
19
20 # Print the shapes of the resulting DataFrames
21 print(f"\nX_train shape after resampling: {X_train_balanced.shape}")
22 print(f"\ny_train shape after resampling: {y_train_balanced.shape}")

```

↗ Class distribution before resampling:

```

Attrition
0    986
1    190
Name: count, dtype: int64

```

Class distribution after SMOTE:

```

Attrition
0    986
1    493
Name: count, dtype: int64

```

Class distribution after SMOTE and undersampling:

```

Attrition
0    493
1    493
Name: count, dtype: int64

```

X\_train shape after resampling: (986, 46)

y\_train shape after resampling: (986,)

Quick checks below to make sure our splits are shaped properly for modeling.

```

1 # Print the shapes of the resulting DataFrames
2 print(f"X_train shape: {X_train_balanced.shape}")
3 print(f"X_test shape: {X_test.shape}")
4 print(f"y_train shape: {y_train_balanced.shape}")
5 print(f"y_test shape: {y_test.shape}")

```

↗ X\_train shape: (986, 46)

X\_test shape: (294, 46)

y\_train shape: (986,)

y\_test shape: (294,)

```

1 # Check the class distribution in y_train and y_test
2 print("Class distribution in y_train:\n", y_train_balanced.value_counts())
3 print("Class distribution in y_test:\n", y_test.value_counts())

```

↗ Class distribution in y\_train:

```

Attrition
0    493
1    493
Name: count, dtype: int64

```

Class distribution in y\_test:

```

Attrition
0    247
1     47
Name: count, dtype: int64

```

Here is the **Decision Tree with balanced classes**.

```

1 clf_tree_balanced =DecisionTreeClassifier(random_state=42)
2 clf_tree_balanced.fit(X_train_balanced, y_train_balanced)
3
4 y_pred_tree_balanced = clf_tree_balanced.predict(X_test)
5
6
7 # Evaluate the classifier
8 print("\nAccuracy Score:")
9 print(accuracy_score(y_test, y_pred_tree_balanced))
10 print("\nClassification Report:")
11 print(classification_report(y_test, y_pred_tree_balanced))
12
13 print("\nAccuracy Score:")
14 print(accuracy_score(y_test, y_pred_tree_balanced))
15
16 print("\nConfusion Matrix:")
17 cm_tree_balanced = confusion_matrix(y_test, y_pred_tree_balanced)
18 print(cm_tree_balanced)
19
20 # Plot the confusion matrix
21 ConfusionMatrixDisplay(confusion_matrix=cm_tree_balanced, display_labels=clf_tree_balanced.classes_).plot()
22 plt.show()

```



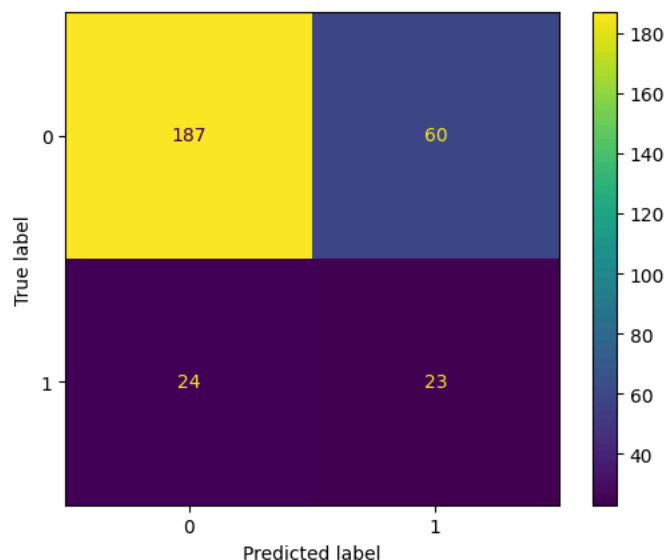
Accuracy Score:  
0.7142857142857143

Classification Report:

	precision	recall	f1-score	support
0	0.89	0.76	0.82	247
1	0.28	0.49	0.35	47
accuracy			0.71	294
macro avg	0.58	0.62	0.59	294
weighted avg	0.79	0.71	0.74	294

Accuracy Score:  
0.7142857142857143

Confusion Matrix:  
[[187 60]  
[ 24 23]]



As we anticipated based on the Oxford Article, balancing the classes had a negative impact on overall model accuracy. **Recall and F1 score** for the minority class improved though. This is part of the trade-off we have to make. Since recall of our minority class is important, we will still consider balancing the data for the remainder of this notebook.

Next, let's use **gridsearchCV** to finetune the parameters of our Decision Tree.

```
1 # Define the parameter grid
2 param_grid_tree = {
3     'criterion': ['gini', 'entropy'],
4     'max_depth': [None, 10, 20, 100],
5     'min_samples_split': [2, 15, 20],
6     'min_samples_leaf': [1, 10, 15, 20],
7     'max_features': [None, 'sqrt', 'log2', 0.5, 0.75],
8     'class_weight': [None, 'balanced', {0: 1, 1: 2}, {0: 1, 1: 4}]
9 }
10
11 # Initialize the classifier
12 clf_tree_balanced_cv = DecisionTreeClassifier(random_state=42)
13
14 # Initialize GridSearchCV
15 grid_search = GridSearchCV(estimator=clf_tree_balanced_cv, param_grid=param_grid_tree, cv=5, scoring='recall', n_jobs=-1)
16
17 # Fit GridSearchCV
18 grid_search.fit(X_train_balanced, y_train_balanced)
19
20 # Print the best parameters found by GridSearchCV
21 print("Best parameters found: ", grid_search.best_params_)
22
23 # Use the best estimator to make predictions
24 best_clf_tree_balanced_cv = grid_search.best_estimator_
25 y_pred_tree_balanced_cv = best_clf_tree_balanced_cv.predict(X_test)
26
27 # Evaluate the classifier
28 print("\nAccuracy Score:")
29 print(accuracy_score(y_test, y_pred_tree_balanced_cv))
30 print("\nClassification Report:")
31 print(classification_report(y_test, y_pred_tree_balanced_cv))
32
33 print("\nConfusion Matrix:")
34 cm_tree_balanced_cv = confusion_matrix(y_test, y_pred_tree_balanced_cv)
35 print(cm_tree_balanced_cv)
36
37 # Plot the confusion matrix
38 ConfusionMatrixDisplay(confusion_matrix=cm_tree_balanced_cv, display_labels=best_clf_tree_balanced_cv.classes_).plot()
39 plt.show()
```



Best parameters found: {'class\_weight': {0: 1, 1: 4}, 'criterion': 'entropy', 'max\_depth': None, 'max\_features': 'sqrt', 'min\_samples\_l

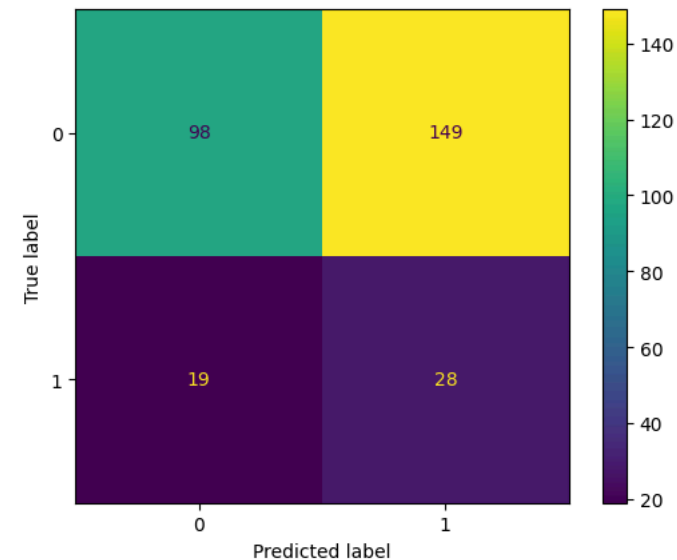
Accuracy Score:  
0.42857142857142855

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.40	0.54	247
1	0.16	0.60	0.25	47
accuracy			0.43	294
macro avg	0.50	0.50	0.39	294
weighted avg	0.73	0.43	0.49	294

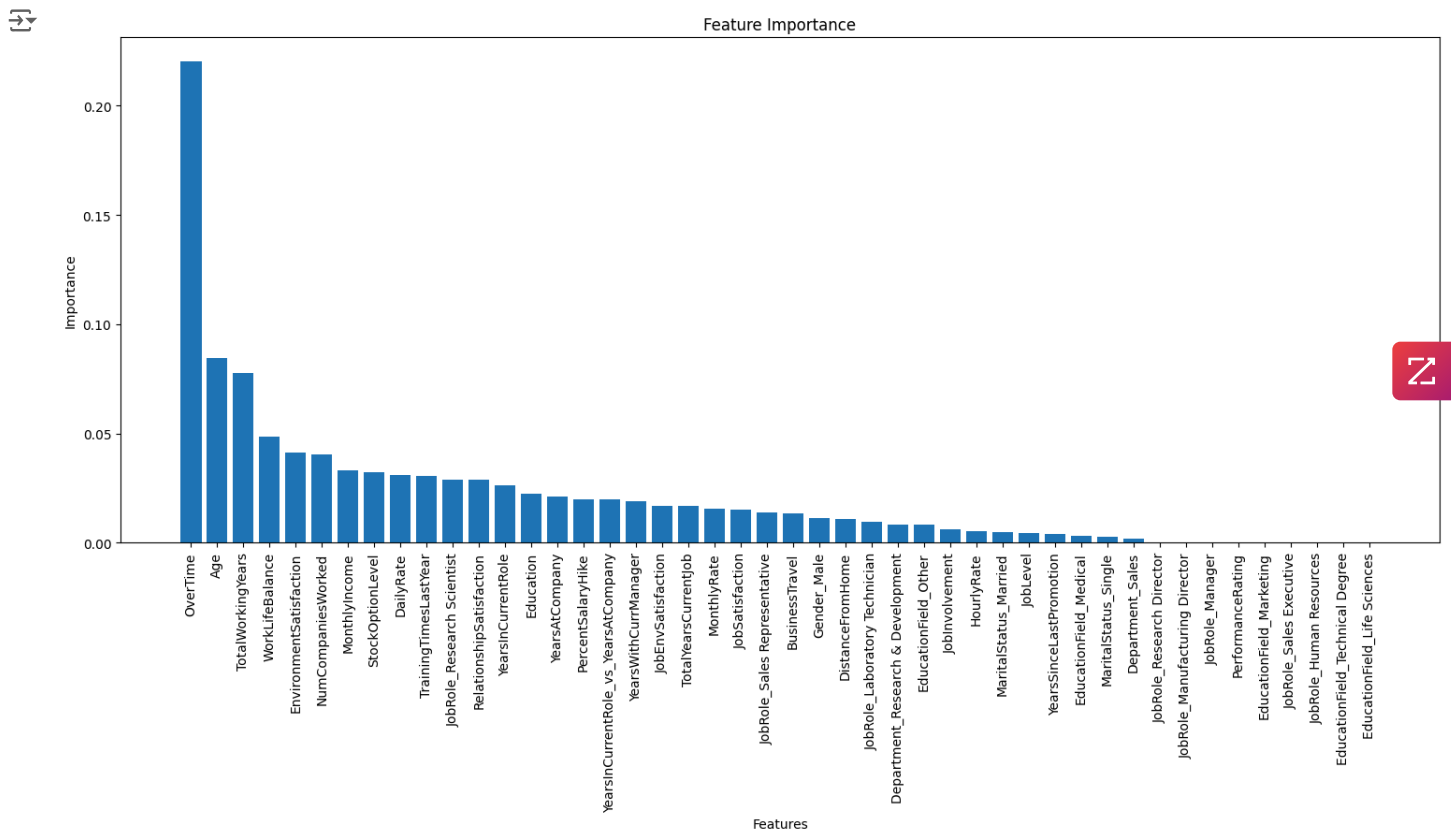
Confusion Matrix:

```
[[ 98 149]
 [ 19  28]]
```



Now let's take a moment to look at the feature importances from our balanced dataset to see if we can **drop some columns that did not contribute to model performance**.

```
1 # Function definition
2 def plot_feature_importance(model, X_train):
3     # Get feature importances
4     feature_importances = model.feature_importances_
5
6     # Sort feature importances in descending order
7     indices = np.argsort(feature_importances)[::-1]
8
9     # Plot the feature importances
10    plt.figure(figsize=(15, 9))
11    plt.bar(range(X_train.shape[1]), feature_importances[indices], align='center')
12    plt.xticks(range(X_train.shape[1]), X_train.columns[indices], rotation=90)
13    plt.xlabel('Features')
14    plt.ylabel('Importance')
15    plt.title('Feature Importance')
16    plt.tight_layout()
17    plt.show()
18
19 # Call the function at the end of the notebook
20 plot_feature_importance(clf_tree_balanced, X_train_balanced)
21
```



There are a few features at the left side of the graph that have little to no predictive relevance here. Let's drop them and see what happens.

Now, let's retrain the model on a new df that drops some of these unimportant features and start this process over.

```
1 df-fi = df-cleaned.drop(columns=[
2     'EducationField_Life Sciences',
3     'EducationField_Technical Degree',
4     'JobRole_Human Resources',
5     'MaritalStatus_Married'
6 ])
```

Now we need to split and train our data again, since we have dropped some columns. We will call the new variables **X-fi** and **y-fi**.

```
1 # Separate target and features
2 X-fi = df-fi.drop('Attrition', axis=1) # all columns except the target column our our independent variables
3 y-fi = df-fi['Attrition']
4
5 # Split the data into train and test sets
6 X_train-fi, X_test-fi, y_train-fi, y_test-fi = train_test_split(X-fi, y-fi, test_size=0.2, random_state=42, shuffle=True, stratify=y)
```

Below we have our **Decision Tree** trained on the new df with dropped features.

```

1 clf_tree-fi =DecisionTreeClassifier(random_state=42)
2 clf_tree-fi.fit(X_train-fi, y_train-fi)
3
4 y_pred_tree-fi = clf_tree-fi.predict(X_test-fi)
5
6
7 # Evaluate the classifier
8 print("\nAccuracy Score:")
9 print(accuracy_score(y_test-fi, y_pred_tree-fi))
10 print("\nClassification Report:")
11 print(classification_report(y_test-fi, y_pred_tree-fi))
12
13 print("\nAccuracy Score:")
14 print(accuracy_score(y_test-fi, y_pred_tree-fi))
15
16 print("\nConfusion Matrix:")
17 cm_tree-fi = confusion_matrix(y_test-fi, y_pred_tree-fi)
18 print(cm_tree-fi)
19
20 # Plot the confusion matrix
21 ConfusionMatrixDisplay(confusion_matrix=cm_tree-fi, display_labels=clf_tree-fi.classes_).plot()
22 plt.show()

```



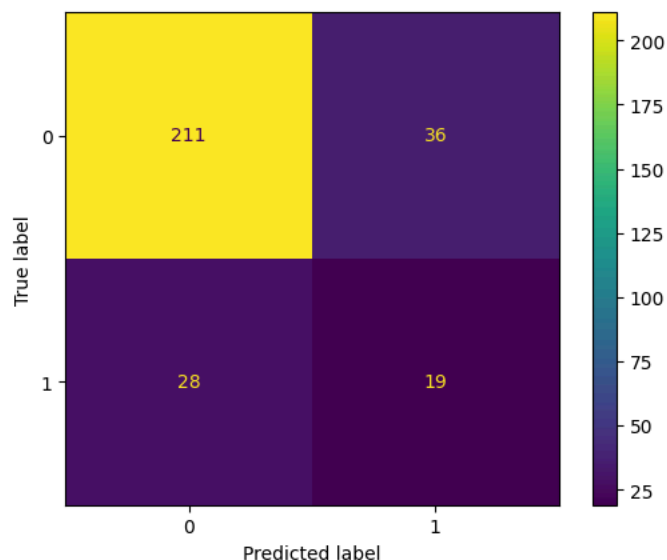
Accuracy Score:  
0.782312925170068

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.85	0.87	247
1	0.35	0.40	0.37	47
accuracy			0.78	294
macro avg	0.61	0.63	0.62	294
weighted avg	0.80	0.78	0.79	294

Accuracy Score:  
0.782312925170068

Confusion Matrix:  
[[211 36]  
[ 28 19]]



Now let's **balance the new X\_train-fi and y\_train-fi** the same way we trained X\_train and y\_train above.

```

1 # Initialize SMOTE and RandomUnderSampler
2 smote = SMOTE(sampling_strategy=.75, random_state=42) # Adjust sampling_strategy as needed
3 under_sampler = RandomUnderSampler(sampling_strategy=.75, random_state=42, replacement=False) # Adjust sampling_strategy as needed
4
5 # Apply SMOTE to the training data
6 X_train-smote-fi, y_train-smote-fi = smote.fit_resample(X_train-fi, y_train-fi)
7

```

```

8 # Apply RandomUnderSampler to the training data
9 X_train_balanced_fi, y_train_balanced_fi = under_sampler.fit_resample(X_train_smote_fi, y_train_smote_fi)
10 # Print the class distribution after resampling
11 print("Class distribution before resampling:")
12 print(y_train_fi.value_counts())
13
14 print("\nClass distribution after SMOTE:")
15 print(y_train_smote_fi.value_counts())
16
17 print("\nClass distribution after SMOTE and undersampling:")
18 print(y_train_balanced_fi.value_counts())
19
20 # Print the shapes of the resulting DataFrames
21 print(f"\nX_train shape after resampling: {X_train_balanced_fi.shape}")
22 print(f"\ny_train shape after resampling: {y_train_balanced_fi.shape}")

```

↩ Class distribution before resampling:

```

Attrition
0    986
1    190
Name: count, dtype: int64

```

Class distribution after SMOTE:

```

Attrition
0    986
1    739
Name: count, dtype: int64

```

Class distribution after SMOTE and undersampling:

```

Attrition
0    985
1    739
Name: count, dtype: int64

```

X\_train shape after resampling: (1724, 42)

y\_train shape after resampling: (1724,)

Below we have the **Decision Tree with balanced data** from the new df with dropped, unimportant features!

```

1 clf_tree_fi_balanced = DecisionTreeClassifier(random_state=42)
2 clf_tree_fi_balanced.fit(X_train_balanced_fi, y_train_balanced_fi)
3
4 y_pred_tree_fi_balanced = clf_tree_fi_balanced.predict(X_test_fi)
5
6
7 # Evaluate the classifier
8 print("\nAccuracy Score:")
9 print(accuracy_score(y_test_fi, y_pred_tree_fi_balanced))
10 print("\nClassification Report:")
11 print(classification_report(y_test_fi, y_pred_tree_fi_balanced))
12
13 print("\nAccuracy Score:")
14 print(accuracy_score(y_test_fi, y_pred_tree_fi_balanced))
15
16 print("\nConfusion Matrix:")
17 cm_tree_fi_balanced = confusion_matrix(y_test_fi, y_pred_tree_fi_balanced)
18 print(cm_tree_fi_balanced)
19
20 # Plot the confusion matrix
21 ConfusionMatrixDisplay(confusion_matrix=cm_tree_fi_balanced, display_labels=clf_tree_fi_balanced.classes_).plot()
22 plt.show()

```



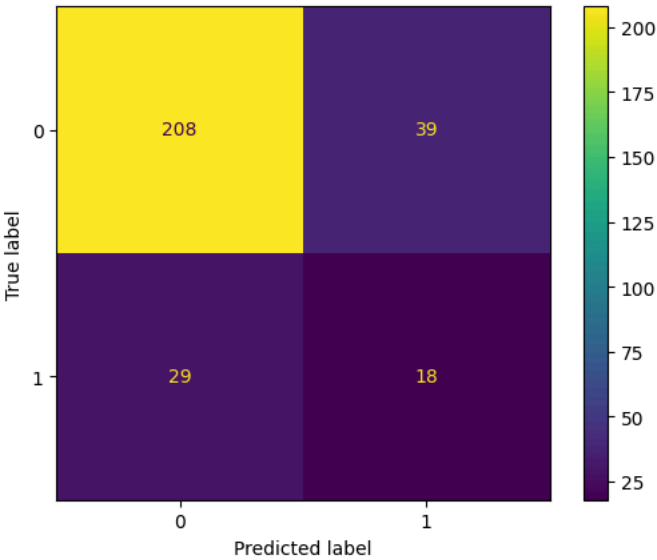
Accuracy Score:  
0.7687074829931972

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.84	0.86	247
1	0.32	0.38	0.35	47
accuracy			0.77	294
macro avg	0.60	0.61	0.60	294
weighted avg	0.79	0.77	0.78	294

Accuracy Score:  
0.7687074829931972

Confusion Matrix:  
[[208 39]  
 [ 29 18]]



Last step for our Decision Tree is to **finetune hyperparameters via GridSearchCV**.





```
1 # Define the parameter grid
2 param_grid_tree_fi_cv = {
3     'max_depth': [None, 10, 20, 30, 40, 50],
4     'min_samples_split': [2, 5, 10, 20],
5     'min_samples_leaf': [1, 2, 5, 10],
6     'criterion': ['gini', 'entropy']
7 }
8
9 # Initialize the classifier
10 clf_tree_fi_balanced_cv = DecisionTreeClassifier(random_state=42)
11
12 # Initialize GridSearchCV with the classifier and parameter grid
13 grid_search = GridSearchCV(estimator=clf_tree_fi_balanced_cv, param_grid=param_grid_tree_fi_cv,
14                             scoring='accuracy', cv=5, n_jobs=-1, verbose=1)
15
16 # Fit GridSearchCV to the training data
17 grid_search.fit(X_train_balanced_fi, y_train_balanced_fi)
18
19 # Get the best estimator
20 best_clf_tree_fi_balanced_cv = grid_search.best_estimator_
21
22 # Make predictions with the best estimator
23 y_pred_tree_fi_balanced_cv = best_clf_tree_fi_balanced_cv.predict(X_test_fi)
24
25 # Evaluate the best estimator
26 print("\nAccuracy Score:")
27 print(accuracy_score(y_test_fi, y_pred_tree_fi_balanced_cv))
28 print("\nClassification Report:")
29 print(classification_report(y_test_fi, y_pred_tree_fi_balanced_cv))
30
31 print("\nConfusion Matrix:")
32 cm_tree_fi_balanced_cv = confusion_matrix(y_test_fi, y_pred_tree_fi_balanced_cv)
33 print(cm_tree_fi_balanced_cv)
34
35 # Plot the confusion matrix
36 ConfusionMatrixDisplay(confusion_matrix=cm_tree_fi_balanced_cv, display_labels=best_clf_tree_fi_balanced_cv.classes_).plot()
37 plt.show()
```



↻ Fitting 5 folds for each of 192 candidates, totalling 960 fits

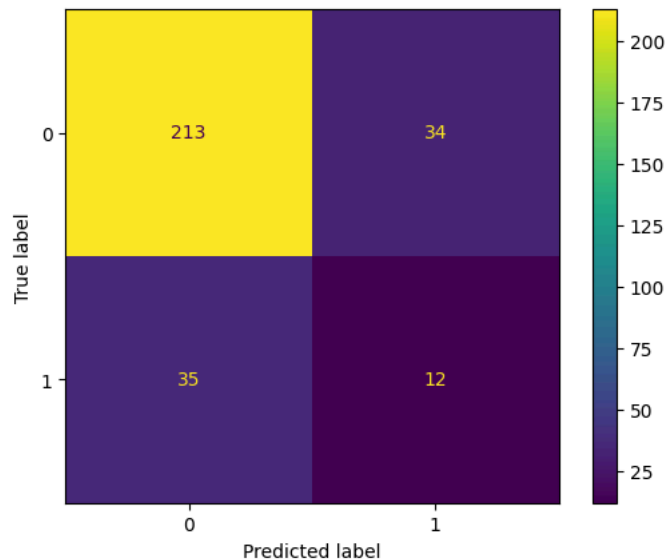
Accuracy Score:  
0.7653061224489796

Classification Report:

	precision	recall	f1-score	support
0	0.86	0.86	0.86	247
1	0.26	0.26	0.26	47
accuracy			0.77	294
macro avg	0.56	0.56	0.56	294
weighted avg	0.76	0.77	0.76	294

Confusion Matrix:

```
[[213  34]
 [ 35  12]]
```



We finished our process for the Decision Tree. Let's complete this same methodology on our **Random Forest and KNN**.

#### ✓ Random Forest

A Random Forest is great for binary classifications for the same great reason Decision Trees are with the added advantage of being able to prevent overfitting.

More documentation on Decision Trees can be found [here](#).

We will start with our original **X\_train** and **y\_train**.

```

1 # Initialize and train your classifier, adjusting class weights if needed
2 clf_rf = RandomForestClassifier(class_weight='balanced', random_state=42)
3 clf_rf.fit(X_train, y_train)
4
5 # Make predictions and evaluate the model
6 y_pred_rf = clf_rf.predict(X_test)
7 print("\nAccuracy Score:")
8 print(accuracy_score(y_test, y_pred_rf))
9 print(classification_report(y_test, y_pred_rf))
10
11 # Print confusion matrix
12 print("Confusion Matrix:")
13 cm_rf = confusion_matrix(y_test, y_pred_rf)
14 print(cm_rf)
15
16 # Plot confusion matrix
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(cm_rf, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
19             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
20 plt.xlabel('Predicted')
21 plt.ylabel('True')
22 plt.title('Confusion Matrix')
23 plt.show()

```



```

Accuracy Score:
0.8367346938775511

```

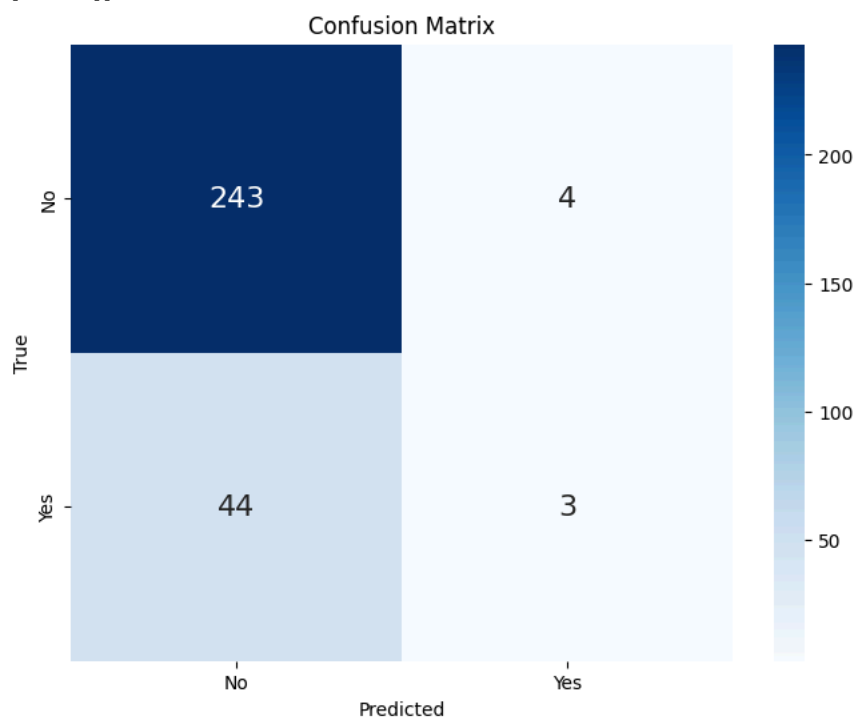
	precision	recall	f1-score	support
0	0.85	0.98	0.91	247
1	0.43	0.06	0.11	47
accuracy			0.84	294
macro avg	0.64	0.52	0.51	294
weighted avg	0.78	0.84	0.78	294

Confusion Matrix:

```

[[243  4]
 [ 44  3]]

```



Following the same methodology from the Decision Tree Classifier, let's **build another Random Forest with the balanced data**.

```

1 # Initialize and train your classifier, adjusting class weights if needed
2 clf_rf_balanced = RandomForestClassifier(class_weight='balanced', random_state=42)
3 clf_rf_balanced.fit(X_train_balanced, y_train_balanced)
4
5 # Make predictions and evaluate the model
6 y_pred_rf_balanced = clf_rf_balanced.predict(X_test)
7 print("\nAccuracy Score:")
8 print(accuracy_score(y_test, y_pred_rf_balanced))
9 print(classification_report(y_test, y_pred_rf_balanced))
10
11 # Print confusion matrix
12 print("Confusion Matrix:")
13 cm_rf_balanced = confusion_matrix(y_test, y_pred_rf_balanced)
14 print(cm_rf_balanced)
15
16 # Plot confusion matrix
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(cm_rf_balanced, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
19             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
20 plt.xlabel('Predicted')
21 plt.ylabel('True')
22 plt.title('Confusion Matrix')
23 plt.show()

```

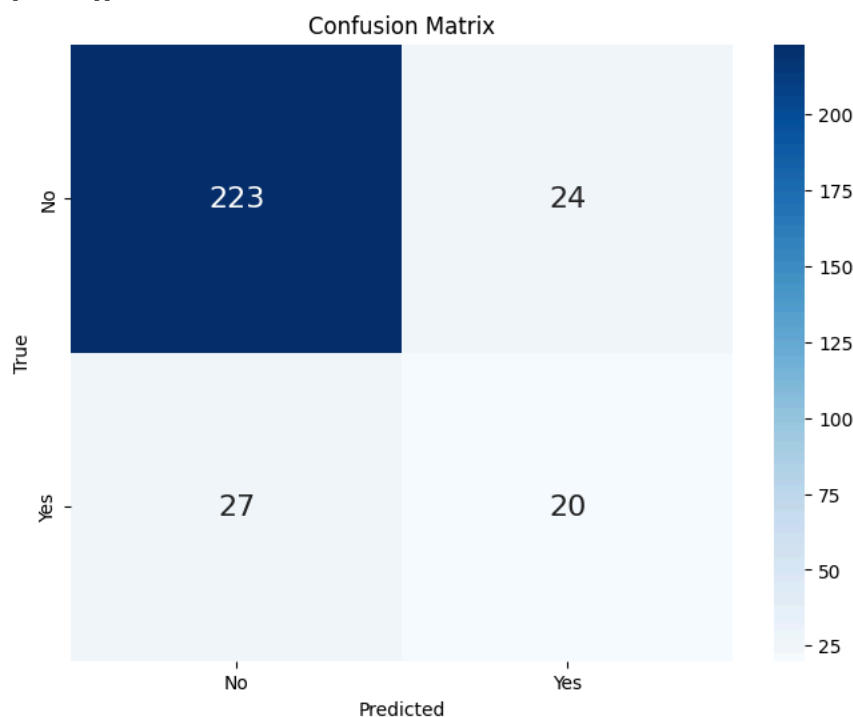


Accuracy Score:  
0.826530612244898

	precision	recall	f1-score	support
0	0.89	0.90	0.90	247
1	0.45	0.43	0.44	47
accuracy			0.83	294
macro avg	0.67	0.66	0.67	294
weighted avg	0.82	0.83	0.82	294

Confusion Matrix:

```
[[223  24]
 [ 27  20]]
```



Now we can take a look at the **finetuned random forest built with class imbalancing addressed**.

```

1 # Define the parameter grid
2 param_grid_rf = {
3     'n_estimators': [100, 200, 300],
4     'max_depth': [None, 10, 20, 30, 40],
5     'min_samples_split': [2, 5, 10],
6     'min_samples_leaf': [1, 2, 4],
7     'criterion': ['gini', 'entropy']
8 }
9
10 # Initialize the classifier
11 clf_rf_balanced_cv = RandomForestClassifier(class_weight='balanced', random_state=42)
12
13 # Initialize GridSearchCV with the classifier and parameter grid
14 grid_search = GridSearchCV(estimator=clf_rf_balanced, param_grid=param_grid_rf,
15                             scoring='accuracy', cv=5, n_jobs=-1, verbose=1)
16
17 # Fit GridSearchCV to the training data
18 grid_search.fit(X_train_balanced, y_train_balanced)
19
20 # Get the best estimator
21 best_clf_rf_balanced_cv = grid_search.best_estimator_
22
23 # Make predictions with the best estimator
24 y_pred_rf_balanced_cv = best_clf_rf_balanced_cv.predict(X_test)
25
26 # Evaluate the best estimator
27 print("\nAccuracy Score:")
28 print(accuracy_score(y_test, y_pred_rf_balanced_cv))
29 print(classification_report(y_test, y_pred_rf_balanced_cv))
30
31 # Print confusion matrix
32 print("Confusion Matrix:")
33 cm_rf_balanced = confusion_matrix(y_test, y_pred_rf_balanced_cv)
34 print(cm_rf_balanced)
35
36 # Plot confusion matrix
37 plt.figure(figsize=(8, 6))
38 sns.heatmap(cm_rf_balanced, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
39             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
40 plt.xlabel('Predicted')
41 plt.ylabel('True')
42 plt.title('Confusion Matrix')
43 plt.show()

```



↻ Fitting 5 folds for each of 270 candidates, totalling 1350 fits

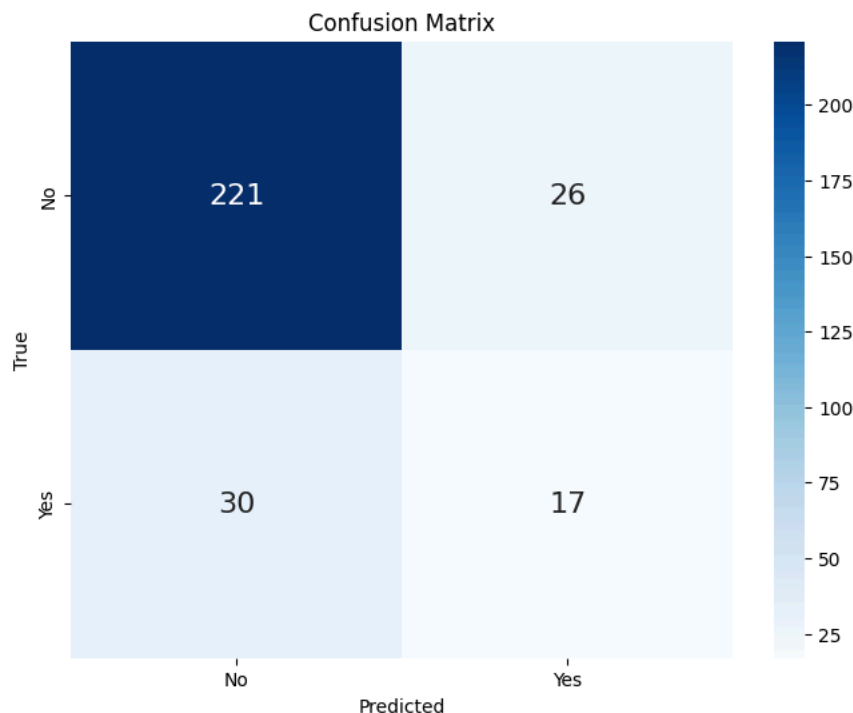
Accuracy Score:

0.8095238095238095

	precision	recall	f1-score	support
0	0.88	0.89	0.89	247
1	0.40	0.36	0.38	47
accuracy			0.81	294
macro avg	0.64	0.63	0.63	294
weighted avg	0.80	0.81	0.81	294

Confusion Matrix:

```
[[221  26]
 [ 30  17]]
```



And now we remove the **unimportant features in our Random Forest**.

Notice the `X_train_fi` and `y_train_fi` used here when fitting the Random Forest.

```
1 # Initialize and train your classifier, adjusting class weights if needed
2 clf_rf_fi = RandomForestClassifier(class_weight='balanced', random_state=42)
3 clf_rf_fi.fit(X_train_fi, y_train_fi)
4
5 # Make predictions and evaluate the model
6 y_pred_rf_fi = clf_rf_fi.predict(X_test_fi)
7 print("\nAccuracy Score:")
8 print(accuracy_score(y_test_fi, y_pred_rf_fi))
9 print(classification_report(y_test_fi, y_pred_rf_fi))
10
11 # Print confusion matrix
12 print("Confusion Matrix:")
13 cm_rf_fi = confusion_matrix(y_test_fi, y_pred_rf_fi)
14 print(cm_rf_fi)
15
16 # Plot confusion matrix
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(cm_rf_fi, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
19             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
20 plt.xlabel('Predicted')
21 plt.ylabel('True')
22 plt.title('Confusion Matrix')
23 plt.show()
```



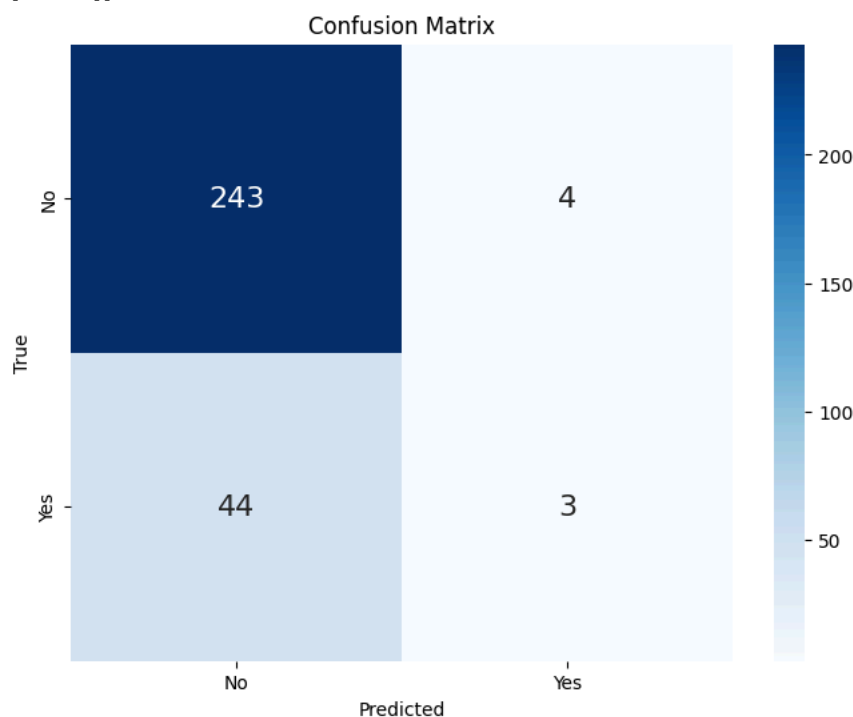
Accuracy Score:

0.8367346938775511

	precision	recall	f1-score	support
0	0.85	0.98	0.91	247
1	0.43	0.06	0.11	47
accuracy			0.84	294
macro avg	0.64	0.52	0.51	294
weighted avg	0.78	0.84	0.78	294

Confusion Matrix:

```
[[243  4]
 [ 44  3]]
```



Now addressing **class imbalance** and with our **Random Forest** that has **unimportant features dropped**.

```
1 # Initialize and train your classifier, adjusting class weights if needed
2 clf_rf_fi_balanced = RandomForestClassifier(class_weight='balanced', random_state=42)
3 clf_rf_fi_balanced.fit(X_train_balanced_fi, y_train_balanced_fi)
4
5 # Make predictions and evaluate the model
6 y_pred_rf_fi_balanced = clf_rf_fi_balanced.predict(X_test_fi)
7 print("\nAccuracy Score:")
8 print(accuracy_score(y_test_fi, y_pred_rf_fi_balanced))
9 print(classification_report(y_test_fi, y_pred_rf_fi_balanced))
10
11 # Print confusion matrix
12 print("Confusion Matrix:")
13 cm_fi_balanced = confusion_matrix(y_test_fi, y_pred_rf_fi_balanced)
14 print(cm_fi_balanced)
15
16 # Plot confusion matrix
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(cm_fi_balanced, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
19             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
20 plt.xlabel('Predicted')
21 plt.ylabel('True')
22 plt.title('Confusion Matrix')
23 plt.show()
```



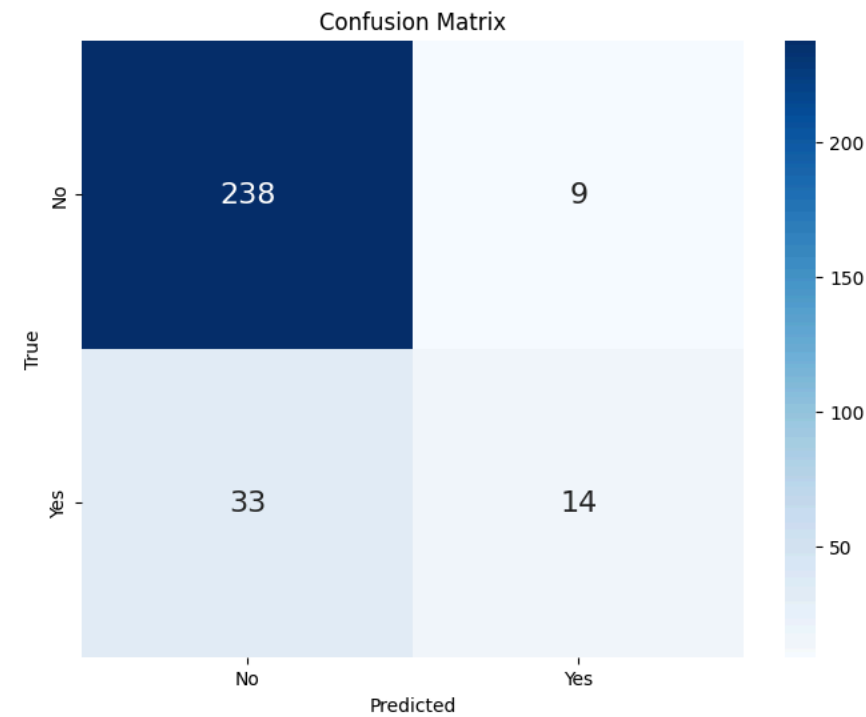
Accuracy Score:

0.8571428571428571

	precision	recall	f1-score	support
0	0.88	0.96	0.92	247
1	0.61	0.30	0.40	47
accuracy			0.86	294
macro avg	0.74	0.63	0.66	294
weighted avg	0.84	0.86	0.84	294

Confusion Matrix:

```
[[238  9]
 [ 33 14]]
```



Now let's **finetune the Random Forest** after unimportant features are dropped.



```
1 # Define the parameter grid
2 param_grid_rf_fi = {
3     'n_estimators': [100, 200, 300],
4     'max_depth': [None, 10, 20, 30],
5     'min_samples_split': [2, 5, 10],
6     'min_samples_leaf': [1, 2, 4],
7     'criterion': ['gini', 'entropy']
8 }
9
10 # Initialize the classifier
11 clf_rf_fi_balanced_cv = RandomForestClassifier(class_weight='balanced', random_state=42)
12
13 # Initialize GridSearchCV with the classifier and parameter grid
14 grid_search = GridSearchCV(estimator=clf_rf_fi_balanced_cv, param_grid=param_grid_rf_fi,
15                             scoring='accuracy', cv=5, n_jobs=-1, verbose=1)
16
17 # Fit GridSearchCV to the training data
18 grid_search.fit(X_train_balanced_fi, y_train_balanced_fi)
19
20 # Get the best estimator
21 best_clf_rf_fi_balanced_cv = grid_search.best_estimator_
22
23 # Make predictions with the best estimator
24 y_pred_rf_fi_balanced_cv = best_clf_rf_fi_balanced_cv.predict(X_test_fi)
25
26 # Evaluate the best estimator
27 print("\nAccuracy Score:")
28 print(accuracy_score(y_test_fi, y_pred_rf_fi_balanced_cv))
29 print(classification_report(y_test_fi, y_pred_rf_fi_balanced_cv))
30
31 # Print confusion matrix
32 print("Confusion Matrix:")
33 cm_rf_fi_balanced_cv = confusion_matrix(y_test_fi, y_pred_rf_fi_balanced_cv)
34 print(cm_rf_fi_balanced_cv)
35
36 # Plot confusion matrix
37 plt.figure(figsize=(8, 6))
38 sns.heatmap(cm_rf_fi_balanced_cv, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
39             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
40 plt.xlabel('Predicted')
41 plt.ylabel('True')
42 plt.title('Confusion Matrix')
43 plt.show()
```



↻ Fitting 5 folds for each of 216 candidates, totalling 1080 fits

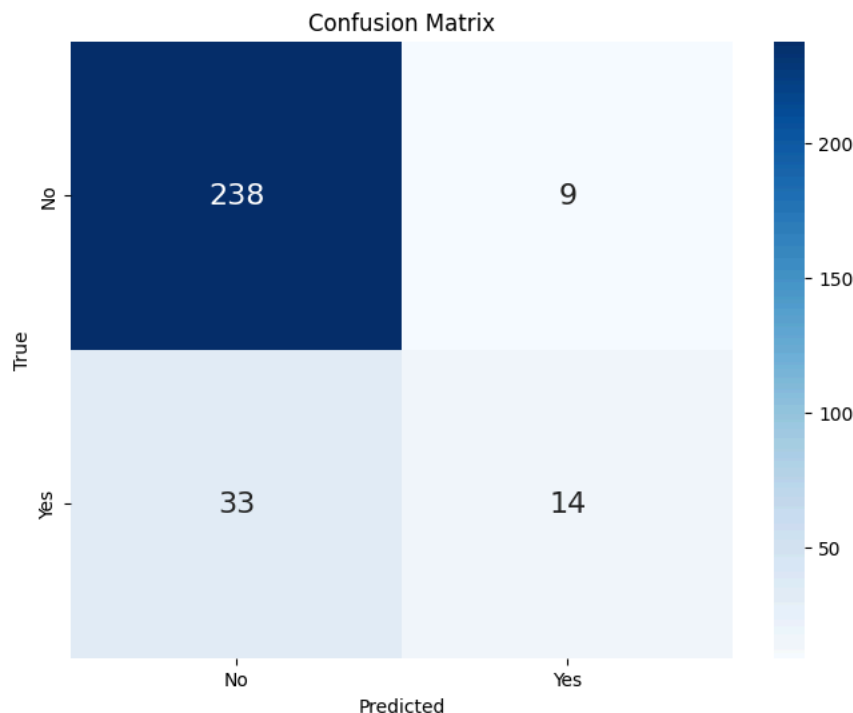
Accuracy Score:

0.8571428571428571

	precision	recall	f1-score	support
0	0.88	0.96	0.92	247
1	0.61	0.30	0.40	47
accuracy			0.86	294
macro avg	0.74	0.63	0.66	294
weighted avg	0.84	0.86	0.84	294

Confusion Matrix:

```
[[238  9]
 [ 33 14]]
```



## ✓ K-Nearest Neighbors

KNN is great for binary classifications because of the way it predicts patterns based on **euclidean distance** of data points. This is a great method for classifications.

More Documentation on KNN can be found [here](#).

Here is our KNN baseline.

```

1 # Initialize and train the KNN classifier
2 clf_knn = KNeighborsClassifier()
3 clf_knn.fit(X_train, y_train)
4
5 # Make predictions and evaluate the model
6 y_pred_knn = clf_knn.predict(X_test)
7 print("\nAccuracy Score:")
8 print(accuracy_score(y_test, y_pred_knn))
9 print(classification_report(y_test, y_pred_knn))
10
11 # Print confusion matrix
12 print("Confusion Matrix:")
13 cm_knn = confusion_matrix(y_test, y_pred_knn)
14 print(cm_knn)
15
16 # Plot confusion matrix
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(cm_knn, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
19             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
20 plt.xlabel('Predicted')
21 plt.ylabel('True')
22 plt.title('Confusion Matrix')
23 plt.show()

```



```

Accuracy Score:
0.8435374149659864

```

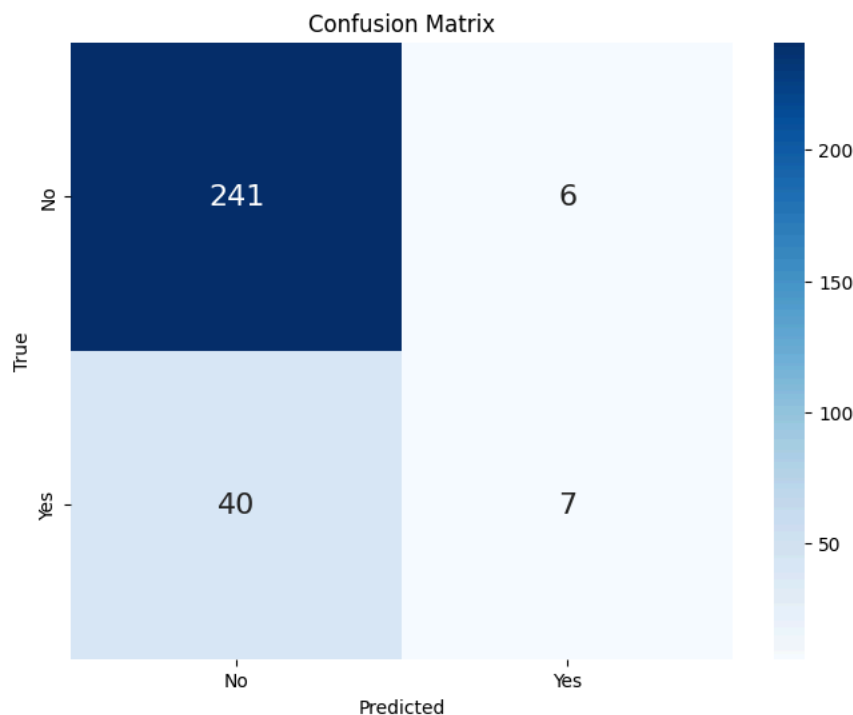
	precision	recall	f1-score	support
0	0.86	0.98	0.91	247
1	0.54	0.15	0.23	47
accuracy			0.84	294
macro avg	0.70	0.56	0.57	294
weighted avg	0.81	0.84	0.80	294

Confusion Matrix:

```

[[241  6]
 [ 40  7]]

```



Now let's use the **balanced data for our KNN model**.

```

1 # Initialize and train the KNN classifier
2 clf_knn_balanced = KNeighborsClassifier()
3 clf_knn_balanced.fit(X_train_balanced, y_train_balanced)
4
5 # Make predictions and evaluate the model
6 y_pred_knn_balanced = clf_knn_balanced.predict(X_test)
7 print("\nAccuracy Score:")
8 print(accuracy_score(y_test, y_pred_knn_balanced))
9 print(classification_report(y_test, y_pred_knn_balanced))
10
11 # Print confusion matrix
12 print("Confusion Matrix:")
13 cm_knn_balanced = confusion_matrix(y_test, y_pred_knn_balanced)
14 print(cm_knn_balanced)
15
16 # Plot confusion matrix
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(cm_knn_balanced, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
19             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
20 plt.xlabel('Predicted')
21 plt.ylabel('True')
22 plt.title('Confusion Matrix')
23 plt.show()

```



```

Accuracy Score:
0.6122448979591837

```

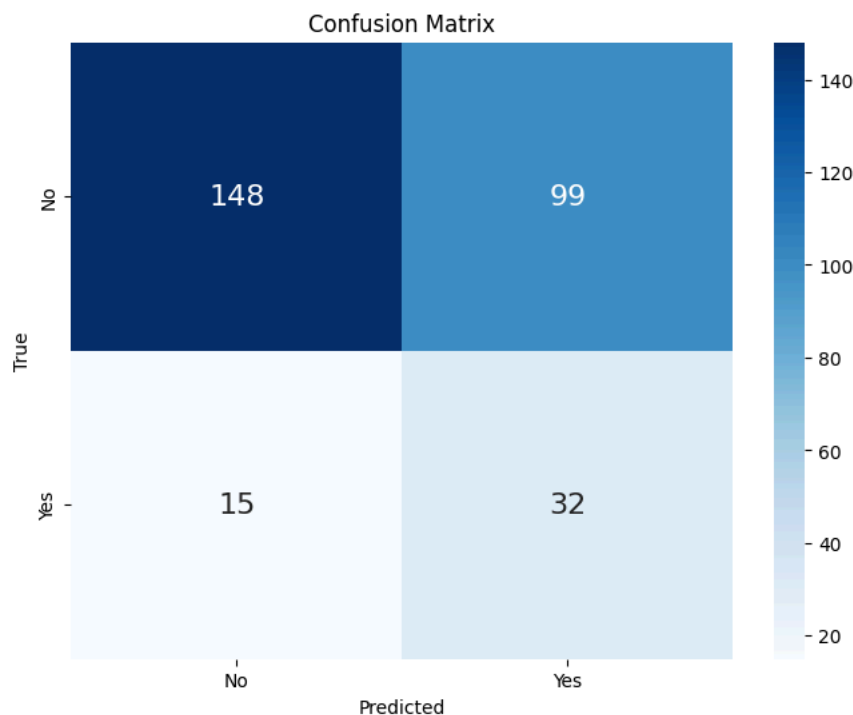
	precision	recall	f1-score	support
0	0.91	0.60	0.72	247
1	0.24	0.68	0.36	47
accuracy			0.61	294
macro avg	0.58	0.64	0.54	294
weighted avg	0.80	0.61	0.66	294

Confusion Matrix:

```

[[148  99]
 [ 15  32]]

```



Now performing **GridSearchCV** on th KNN model with balanced data.

```
1
2 # Define the parameter grid
3 param_grid_knn = {
4     'n_neighbors': [3, 5, 7, 9, 11],
5     'weights': ['uniform', 'distance'],
6     'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
7     'leaf_size': [10, 20, 30, 40, 50]
8 }
9
10 # Initialize the KNN classifier
11 clf_knn_balanced_cv = KNeighborsClassifier()
12
13 # Initialize GridSearchCV with the classifier and parameter grid
14 grid_search = GridSearchCV(estimator=clf_knn_balanced_cv, param_grid=param_grid_knn,
15                             scoring='accuracy', cv=5, n_jobs=-1, verbose=1)
16
17 # Fit GridSearchCV to the training data
18 grid_search.fit(X_train_balanced, y_train_balanced)
19
20 # Get the best estimator
21 best_clf_knn_balanced_cv = grid_search.best_estimator_
22
23 # Make predictions with the best estimator
24 y_pred_knn_balanced_cv = best_clf_knn_balanced_cv.predict(X_test)
25
26 # Evaluate the best estimator
27 print("\nAccuracy Score:")
28 print(accuracy_score(y_test, y_pred_knn_balanced_cv))
29 print("\nClassification Report:")
30 print(classification_report(y_test, y_pred_knn_balanced_cv))
31
32 print("\nConfusion Matrix:")
33 cm_knn_balanced_cv = confusion_matrix(y_test, y_pred_knn_balanced_cv)
34 print(cm_knn_balanced_cv)
35
36 # Plot the confusion matrix
37 plt.figure(figsize=(8, 6))
38 sns.heatmap(cm_knn_balanced_cv, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
39             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
40 plt.xlabel('Predicted')
41 plt.ylabel('True')
42 plt.title('Confusion Matrix')
43 plt.show()
```

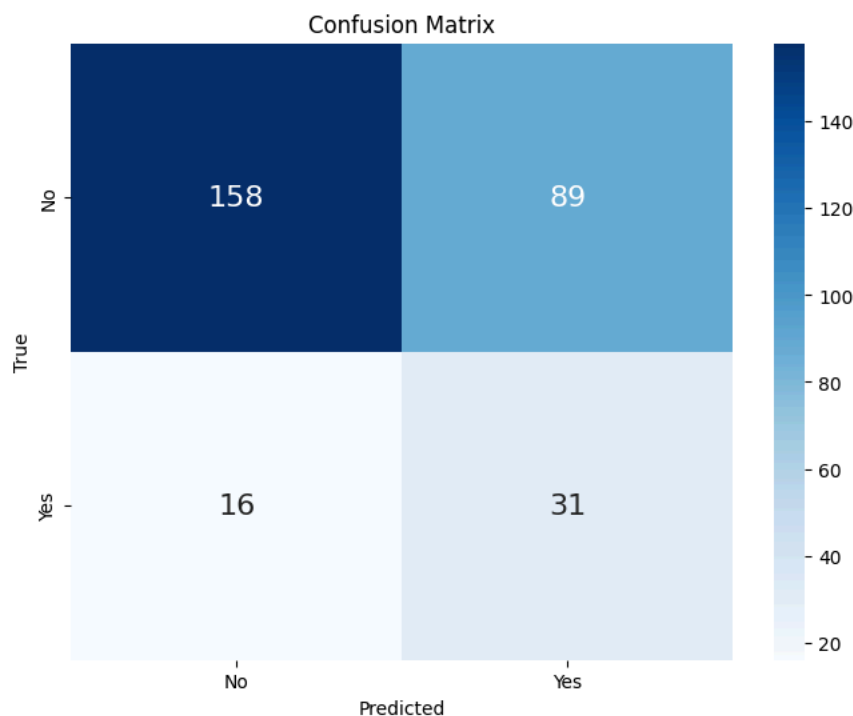


↻ Fitting 5 folds for each of 200 candidates, totalling 1000 fits

Accuracy Score:  
0.6428571428571429

Classification Report:				
	precision	recall	f1-score	support
0	0.91	0.64	0.75	247
1	0.26	0.66	0.37	47
accuracy			0.64	294
macro avg	0.58	0.65	0.56	294
weighted avg	0.80	0.64	0.69	294

Confusion Matrix:  
[[158 89]  
[ 16 31]]



Now let's build a **KNN model based on the data after we dropped feature importances.**

```

1 # Initialize and train the KNN classifier
2 clf_knn_fi = KNeighborsClassifier()
3 clf_knn_fi.fit(X_train_fi, y_train_fi)
4
5 # Make predictions and evaluate the model
6 y_pred_knn_fi = clf_knn_fi.predict(X_test_fi)
7 print("\nAccuracy Score:")
8 print(accuracy_score(y_test_fi, y_pred_knn_fi))
9 print(classification_report(y_test_fi, y_pred_knn_fi))
10
11 # Print confusion matrix
12 print("Confusion Matrix:")
13 cm_knn_fi = confusion_matrix(y_test_fi, y_pred_knn_fi)
14 print(cm_knn_fi)
15
16 # Plot confusion matrix
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(cm_knn_fi, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
19             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
20 plt.xlabel('Predicted')
21 plt.ylabel('True')
22 plt.title('Confusion Matrix')
23 plt.show()

```



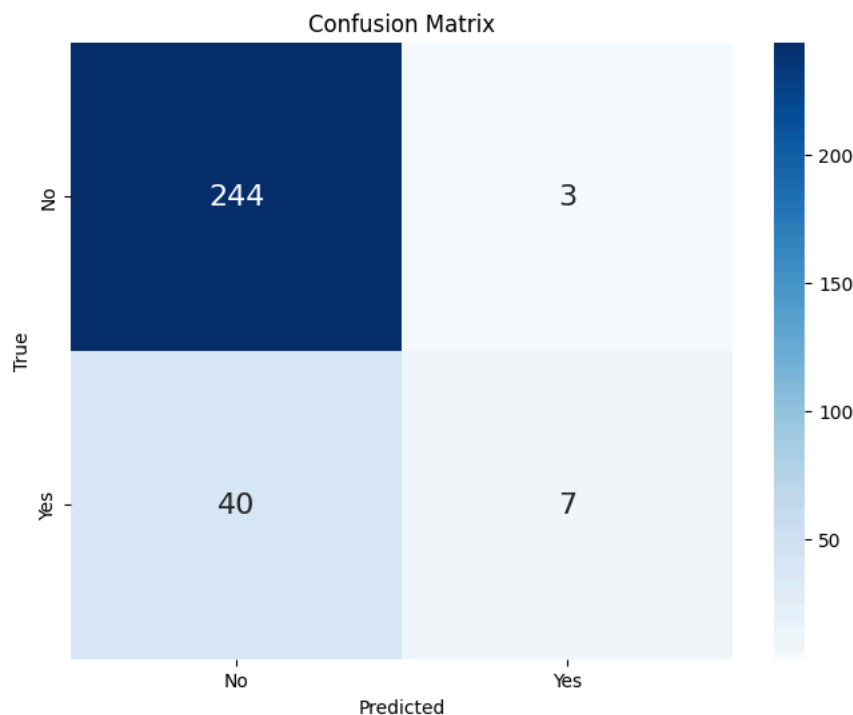
Accuracy Score:

0.8537414965986394

	precision	recall	f1-score	support
0	0.86	0.99	0.92	247
1	0.70	0.15	0.25	47
accuracy			0.85	294
macro avg	0.78	0.57	0.58	294
weighted avg	0.83	0.85	0.81	294

Confusion Matrix:

```
[[244  3]
 [ 40  7]]
```



Now we can build a **KNN model with the balanced data after dropping unimportant features.**

```
1 # Initialize and train the KNN classifier
2 clf_knn_fi_balanced = KNeighborsClassifier()
3 clf_knn_fi_balanced.fit(X_train_balanced_fi, y_train_balanced_fi)
4
5 # Make predictions and evaluate the model
6 y_pred_knn_fi_balanced = clf_knn_fi_balanced.predict(X_test_fi)
7 print("\nAccuracy Score:")
8 print(accuracy_score(y_test_fi, y_pred_knn_fi_balanced))
9 print(classification_report(y_test_fi, y_pred_knn_fi_balanced))
10
11 # Print confusion matrix
12 print("Confusion Matrix:")
13 cm_knn_fi_balanced = confusion_matrix(y_test_fi, y_pred_knn_fi_balanced)
14 print(cm_knn_fi_balanced)
15
16 # Plot confusion matrix
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(cm_knn_fi_balanced, annot=True, cmap='Blues', fmt='g', annot_kws={"size": 16},
19             xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
20 plt.xlabel('Predicted')
21 plt.ylabel('True')
22 plt.title('Confusion Matrix')
23 plt.show()
```



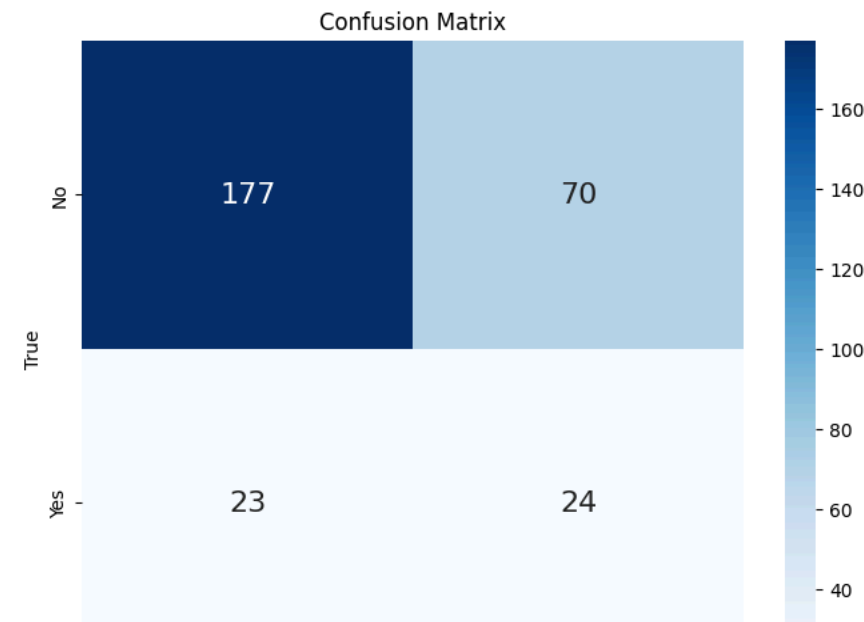
Accuracy Score:

0.6836734693877551

	precision	recall	f1-score	support
0	0.89	0.72	0.79	247
1	0.26	0.51	0.34	47
accuracy			0.68	294
macro avg	0.57	0.61	0.57	294
weighted avg	0.78	0.68	0.72	294

Confusion Matrix:

```
[[177  70]
 [ 23  24]]
```



Lastly, let's **finetune this final KNN model**.

```
1 # Define the parameter grid
2 param_grid_knn_fi = {
3     'n_neighbors': [3, 5, 7, 9, 11],
4     'weights': ['uniform', 'distance'],
5     'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
6     'leaf_size': [10, 20, 30, 40, 50]
7 }
8
9 # Initialize the KNN classifier
10 clf_knn_fi_balanced_cv = KNeighborsClassifier()
11
12 # Initialize GridSearchCV with the classifier and parameter grid
13 grid_search = GridSearchCV(estimator=clf_knn_fi_balanced_cv, param_grid=param_grid_knn_fi,
14                             scoring='accuracy', cv=5, n_jobs=-1, verbose=1)
15
16 # Fit GridSearchCV to the training data
17 grid_search.fit(X_train_balanced_fi, y_train_balanced_fi)
18
19 # Get the best estimator
20 best_clf_knn_fi_balanced_cv = grid_search.best_estimator_
21
22 # Make predictions with the best estimator
```