James Okoth

2023/HD05/04711U | 2300704711

```
# Importing the required libraries
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
```

Double-click (or enter) to edit

Exploratory Data Analysis (EDA)

- Load the dataset and perform initial data exploration
- Summarize key statistics and visualizations for the dataset.
- Identify missing values and decide on an appropriate strategy to handle them.
- Explore the distribution of the target variable ('Attrition').

```
# Reading the CSV file into a pandas DataFrame
data = pd.read_csv('/content/employee_attrition_dataset.csv')
```

```
# Show a few rows and columns of the dataset
data.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	..
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	.
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	.
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	.
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	.
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	.

5 rows × 35 columns

```
# Shape of the dataset | shows how many rows and columns are in the dataset
data.shape
```

```
(1470, 35)
```

```
# Show all the attributes of the dataset -> its columns
data.columns
```

```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
```

```
            'YearsWithCurrManager'],
           dtype='object')
```

```
len(data.columns)
```

```
35
```

```
# Check for Null Values
data.isnull().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
```

```
# Check data types of each attribute
data.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
```

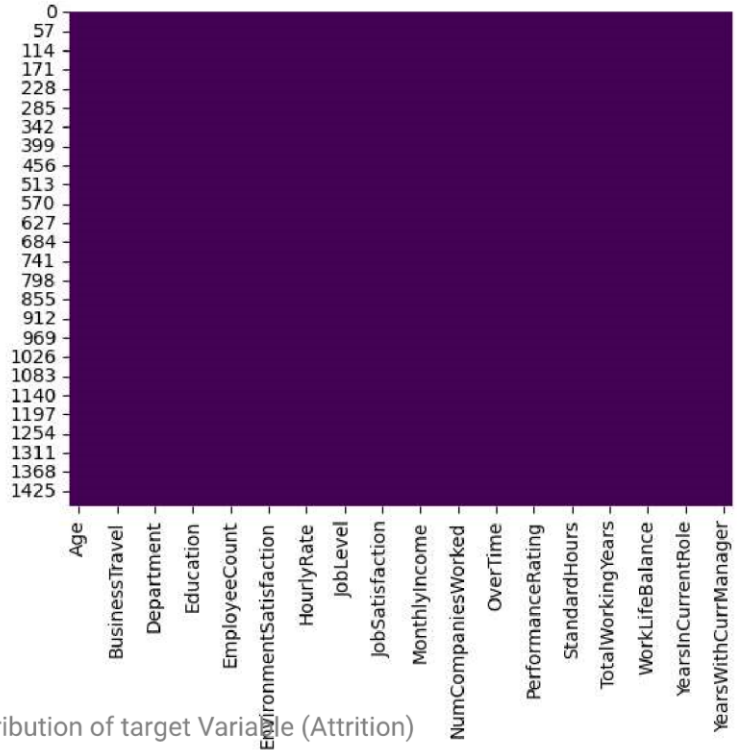
```
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 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
```

```
#Checking for wrong entries like symbols -,?,#,*,etc.
for col in data.columns:
    print('{} : {}'.format(col, data[col].unique()))
```

```
Age : [41 49 37 33 27 32 59 30 38 36 35 29 31 34 28 22 53 24 21 42 44 46 39 43
 50 26 48 55 45 56 23 51 40 54 58 20 25 19 57 52 47 18 60]
Attrition : ['Yes' 'No']
BusinessTravel : ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
DailyRate : [1102  279 1373 1392  591 1005 1324 1358  216 1299  809  153  670 1346
  103 1389  334 1123 1219  371  673 1218  419  391  699 1282 1125  691
  477  705  924 1459  125  895  813 1273  869  890  852 1141  464 1240
 1357  994  721 1360 1065  408 1211 1229  626 1434 1488 1097 1443  515
  853 1142  655 1115  427  653  989 1435 1223  836 1195 1339  664  318
 1225 1328 1082  548  132  746  776  193  397  945 1214  111  573 1153
 1400  541  432  288  669  530  632 1334  638 1093 1217 1353  120  682
  489  807  827  871  665 1040 1420  240 1280  534 1456  658  142 1127
 1031 1189 1354 1467  922  394 1312  750  441  684  249  841  147  528
  594  470  957  542  802 1355 1150 1329  959 1033 1316  364  438  689
  201 1427  857  933 1181 1395  662 1436  194  967 1496 1169 1145  630
  303 1256  440 1450 1452  465  702 1157  602 1480 1268  713  134  526
 1380  140  629 1356  328 1084  931  692 1069  313  894  556 1344  290
  138  926 1261  472 1002  878  905 1180  121 1136  635 1151  644 1045
  829 1242 1469  896  992 1052 1147 1396  663  119  979  319 1413  944
 1323  532  818  854 1034  771 1401 1431  976 1411 1300  252 1327  832
 1017 1199  504  505  916 1247  685  269 1416  833  307 1311  128  488
  529 1210 1463  675 1385 1403  452  666 1158  228  996  728 1315  322
 1479  797 1070  442  496 1372  920  688 1449 1117  636  506  444  950
  889  555  230 1232  566 1302  812 1476  218 1132 1105  906  849  390
  106 1249  192  553  117  185 1091  723 1220  588 1377 1018 1275  798
  672 1162  508 1482  559  210  928 1001  549 1124  738  570 1130 1192
  343  144 1296 1309  483  810  544 1062 1319  641 1332  756  845  593
 1171  350  921 1144  143 1046  575  156 1283  755  304 1178  329 1362
 1371  202  253  164 1107  759 1305  982  821 1381  480 1473  891 1063
  645 1490  317  422 1485 1368 1448  296 1398 1349  986 1099 1116 1499
  983 1009 1303 1274 1277  587  413 1276  988 1474  163  267  619  302
  443  828  561  426  232 1306 1094  509  775  195  258  471  799  956
  535 1495  446 1245  703  823 1246  622 1287  448  254 1365  538  525
  558  782  362 1236 1112  204 1343  604 1216  646  160  238 1397  306
  991  482 1176  913 1076  727  885  243  806  817 1410 1207 1442  693
  929  562  608  580  970 1179  294  314  316  654  168  381  217  501
  650  141  804  975 1090  346  430  268  167  621  527  883  954  310
  719  725  715  657 1146  182  376  571  384  791 1111 1243 1092 1325
  805  213  118  676 1252  286 1258  932 1041  859  720  946 1184  436
  589  760  887 1318  625  180  586 1012  661  930  342 1230 1271 1278
  607  130  300  583 1418 1269  379  395 1265 1222  341  868 1231  102
  881 1383 1075  374 1086  781  177  500 1425 1454  617 1085  995 1122
  618  546  462 1198 1272  154 1137 1188  188 1333  867  263  938  129
  616  498 1404 1053  289 1376  231  152  882  903 1379  335  722  461
  974 1126  840 1134  248  955  939 1391 1206  287 1441  109 1066  277
  466 1055  265  135  247 1035  266  145 1038 1234 1109 1089  788  124
  660 1186 1464  796  415  769 1003 1366  330 1492 1204  309 1330  469
  697 1262 1050  770  406  203 1308  984  439  793 1451 1182  174  490
  718  433  773  603  874  367  199  481  647 1384  902  819  862 1457
  977  942 1402 1421 1361  917  200  150  179  696  116  363  107 1465
  458 1212 1103  966 1010  326 1098  969 1167  694 1320  536  373  599
  251  131  237 1429  648  735  531  429  968  879  640  412  848  360
 1138  325 1322  299 1030  634  524  256 1060  935  495  282  206  943
  523  507  601  855 1291 1405 1369  999 1202  285  404  736 1498 1200
 1439  499  205  683 1462  949  652  332 1475  337  971 1174  667  560
  172  383 1255  359  401  377  592 1445 1221  866  981  447 1326  748
  990  405  115  790  830 1193 1423  467  271  410 1083  516  224  136
 1029  333 1440  674 1342  898  824  492  598  740  888 1288  104 1108
```

```
# Using a Heatmap to show if there are any missing Values
sns.heatmap(data.isnull(),cbar=False,cmap='viridis')
```

```
<Axes: >
```



Distribution of target Variable (Attrition)

```
# Calculating the distribution of Attrition
attrition_distribution = data['Attrition'].value_counts()

attrition_distribution
```

```
No     1233
Yes     237
Name: Attrition, dtype: int64
```

Feature Selection and Importance Analysis

- Preprocess the data, including encoding categorical variables and scaling numerical features.
- Apply feature selection techniques (e.g. correlation analysis, feature importance from tree-based models) to identify the most relevant features for attrition prediction.
- Justify your selection of features based on their importance.

```
[col for col in data]
```

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

```
      'RelationshipSatisfaction',
      'StandardHours',
      'StockOptionLevel',
      'TotalWorkingYears',
      'TrainingTimesLastYear',
      'WorkLifeBalance',
      'YearsAtCompany',
      'YearsInCurrentRole',
      'YearsSinceLastPromotion',
      'YearsWithCurrManager']


numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
cat_cols = [col for col in data if data[col].dtype.name not in numeric_dtypes]
cat_cols
```

```
    ['Attrition',
     'BusinessTravel',
     'Department',
     'EducationField',
     'Gender',
     'JobRole',
     'MaritalStatus',
     'Over18',
     'OverTime']
```

```
cat_cols.remove('Attrition')
cat_cols
```

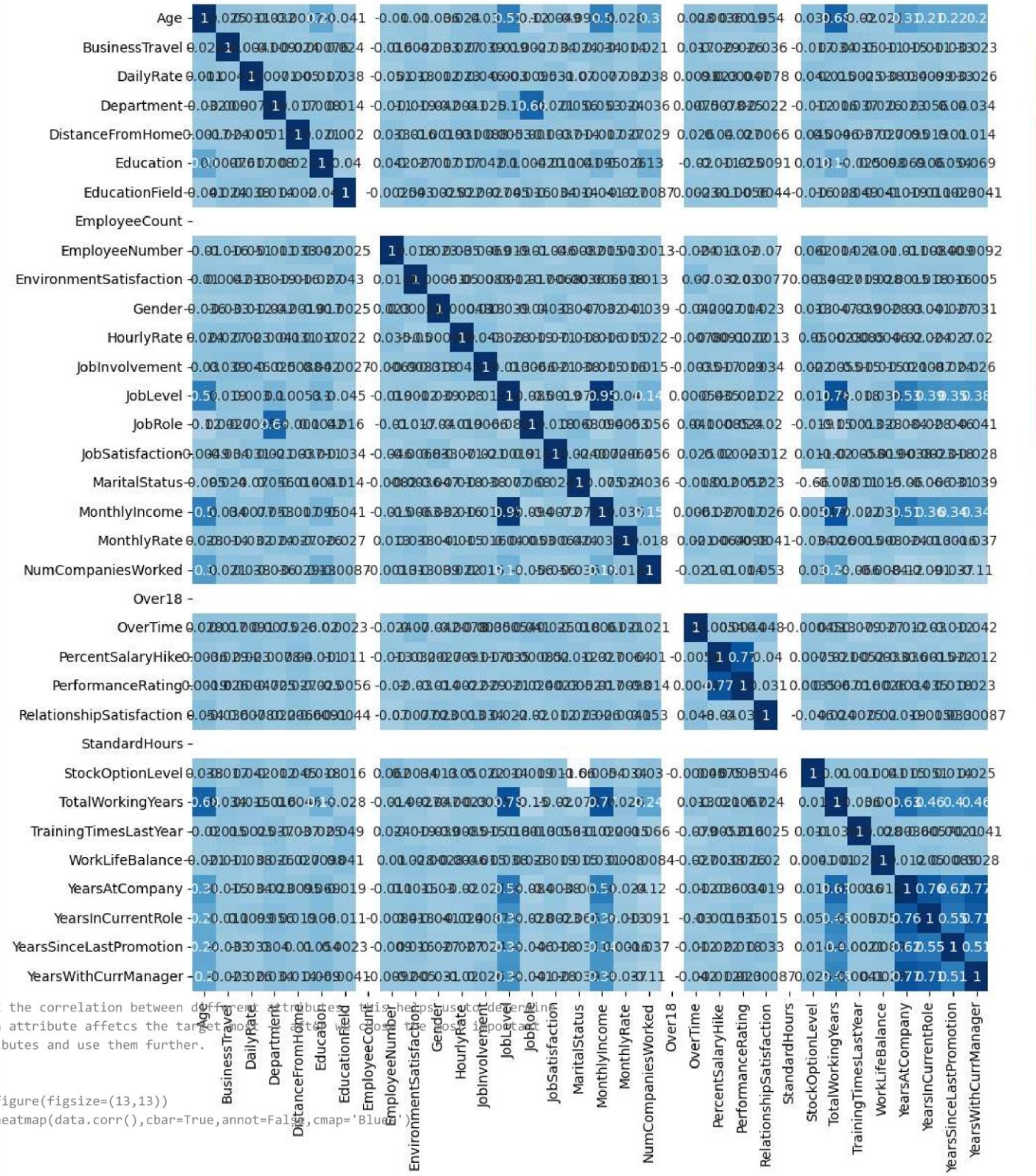
```
    ['BusinessTravel',
     'Department',
     'EducationField',
     'Gender',
     'JobRole',
     'MaritalStatus',
     'Over18',
     'OverTime']
```

```
'''
Using data.info() above, i already know attributes with categorical values and i
can easily convert them into continuous values as below
'''

# Converting categorical columns to numerical using cat.codes
categorical_columns = ['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'Over18', 'OverTime']
for column in categorical_columns:
    data[column] = data[column].astype('category').cat.codes


'''
Scaling is an important operation, however it has an impact while using methods
that are based on measures of how far apart data points are. Such methods include
Support Vector Machines and KNNs using measures such as Euclidean distances
'''
```

```
    '\nScaling is an important operation, however it has an impact while using methods\nthat are based on measures of how far apart data poi
    such as Euclidean distances\n'
```

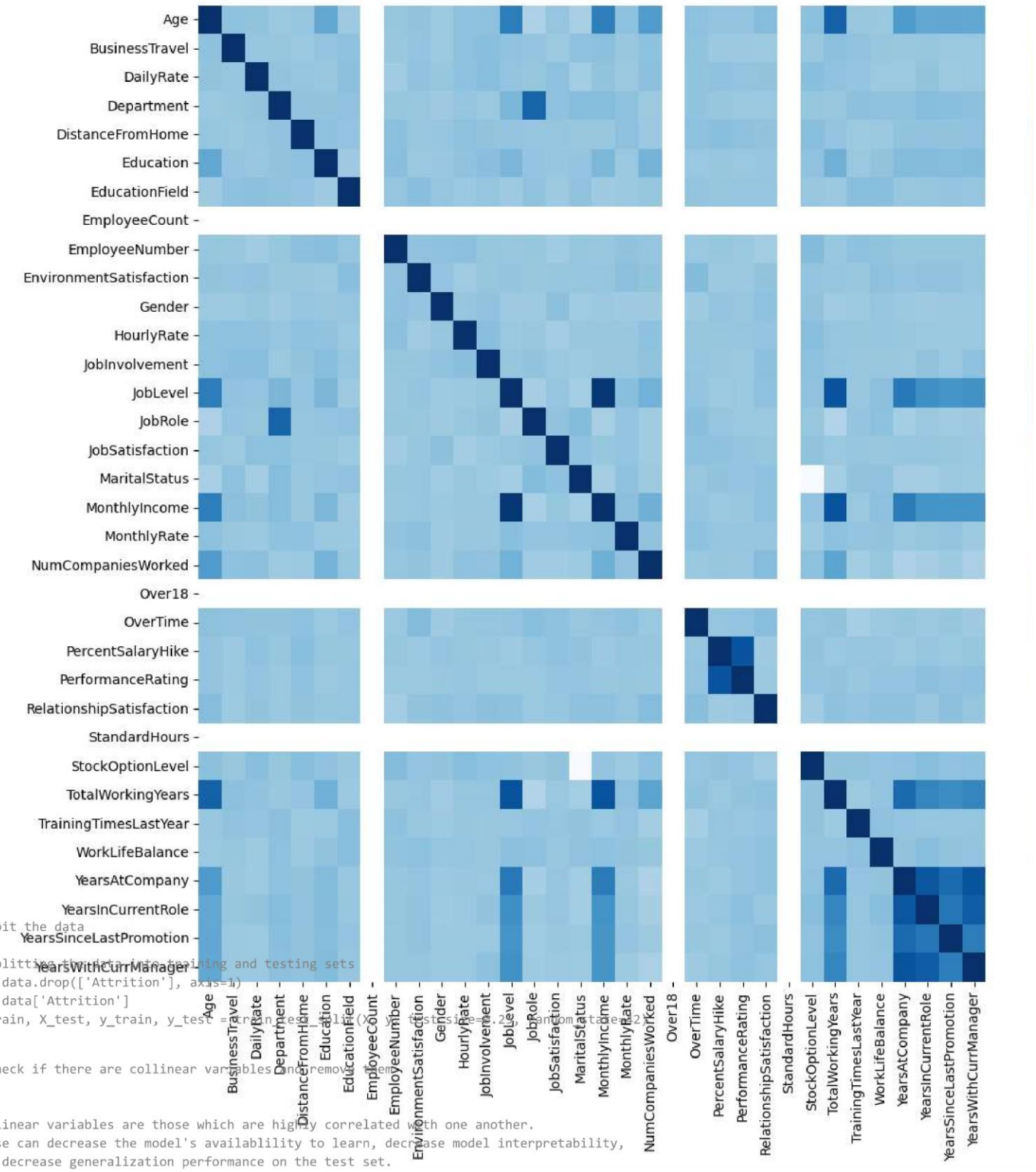
```
'''
Check the correlation between different attributes, this helps us to determine
which attribute affects the target most -> after we chose the most important
attributes and use them further.
'''

plt.figure(figsize=(13,13))
sns.heatmap(data.corr(),cbar=True,annot=True,cmap='Blues')
```

```
<ipython-input-69-0791c0cda746>:8: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  sns.heatmap(data.corr(),cbar=True,annot=True,cmap='Blues')
<Axes: >
```




```
'''
Check the correlation between different attributes. This helps us to determine
which attribute affetcs the target most. We should use the most important
attributes and use them further.
'''

plt.figure(figsize=(13,13))
sns.heatmap(data.corr(),cbar=True,annot=False,cmap='Blues')
```

```
<ipython-input-70-f795d7be4724>:8: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  sns.heatmap(data.corr(),cbar=True,annot=False,cmap='Blues')
<Axes: >
```




```
# Spit the data

# Splitting the data into training and testing sets
X = data.drop(['Attrition'], axis=1)
y = data['Attrition']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Check if there are collinear variables and remove them.

'''
Collinear variables are those which are highly correlated with one another.
These can decrease the model's availablility to learn, decrease model interpretability,
and decrease generalization performance on the test set.
'''

# Absolute value correlation matrix
corr_matrix = X_train.corr().abs()
corr_matrix.head()
```

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeN
Age	1.000000	0.038422	0.008804	0.053907	0.021433	0.229553	0.019645	NaN	0.0
BusinessTravel	0.038422	1.000000	0.002461	0.012581	0.037316	0.015078	0.039966	NaN	0.0
DailyRate	0.008804	0.002461	1.000000	0.012196	0.003311	0.045373	0.062287	NaN	0.0
Department	0.053907	0.012581	0.012196	1.000000	0.000128	0.004111	0.011376	NaN	0.0

```
# Upper triangle of correlations
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
upper.head()
```

```
<ipython-input-73-b8b02147a528>:2: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, u
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
```

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumbe
Age	NaN	0.038422	0.008804	0.053907	0.021433	0.229553	0.019645	NaN	0.01196
BusinessTravel	NaN	NaN	0.002461	0.012581	0.037316	0.015078	0.039966	NaN	0.00186
DailyRate	NaN	NaN	NaN	0.012196	0.003311	0.045373	0.062287	NaN	0.04542
Department	NaN	NaN	NaN	NaN	0.000128	0.004111	0.011376	NaN	0.00784
DistanceFromHome	NaN	NaN	NaN	NaN	NaN	0.024546	0.014871	NaN	0.04088

5 rows × 34 columns

```
# Select columns with correlations above threshold
threshold = 0.8
to_drop = [column for column in upper.columns if any(upper[column] > threshold)]

# print('There are %d columns to remove.' % (len(to_drop)))
to_drop
```

```
['MonthlyIncome']
```

```
# Reassign X
# X = X.drop(to_drop, axis=1)

'''
We can choose important variables from the correlation grid above but the easiest way
while using tree-based algos is using the feature-importance.

Let's check feature importance using Random Forest model
'''

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Creating an instance of the DecisionTreeClassifier
decision_tree_model = DecisionTreeClassifier()
random_forest_model = RandomForestClassifier()

# Training the model on the training data
decision_tree_model.fit(X_train, y_train)
random_forest_model.fit(X_train, y_train)

# Making predictions on the testing data
y_pred_dt = decision_tree_model.predict(X_test)
y_pred_rf = decision_tree_model.predict(X_test)
```

```
# Computing feature importances
importances = random_forest_model.feature_importances_

imp_set = {}
# Printing the feature importances
for feature, importance in zip(X.columns, importances):
    imp_set[feature] = importance
    print(f'{feature}: {importance * 100}')
```

```
Age: 5.8752173001191395
BusinessTravel: 1.2834970598304098
DailyRate: 4.845097893769248
Department: 1.305507206094237
DistanceFromHome: 4.2933075190521786
Education: 1.8995377199866696
EducationField: 2.1344405902817343
EmployeeCount: 0.0
EmployeeNumber: 4.741643672757769
EnvironmentSatisfaction: 2.5748494235233568
Gender: 0.8315523565898726
HourlyRate: 4.3540318833850735
JobInvolvement: 2.0667554744740237
JobLevel: 2.5034663700501265
JobRole: 3.070761345668882
JobSatisfaction: 2.4453469230002716
MaritalStatus: 2.4241354353586826
MonthlyIncome: 7.372803913193865
MonthlyRate: 4.720777280578856
NumCompaniesWorked: 3.612969038437872
Over18: 0.0
OverTime: 5.845521062663033
PercentSalaryHike: 3.3420735547363267
PerformanceRating: 0.37573825676188066
RelationshipSatisfaction: 2.070212615651746
StandardHours: 0.0
StockOptionLevel: 3.1755419425798093
TotalWorkingYears: 5.197915889617773
TrainingTimesLastYear: 2.830976789887474
WorkLifeBalance: 2.0941365684308315
YearsAtCompany: 4.143812755043779
YearsInCurrentRole: 2.6763478247904384
YearsSinceLastPromotion: 2.4518364918705857
YearsWithCurrManager: 3.4401878418140552
```

```
imp_set
```

```
{'Age': 0.05875217300119139,
 'BusinessTravel': 0.012834970598304098,
 'DailyRate': 0.04845097893769248,
 'Department': 0.01305507206094237,
 'DistanceFromHome': 0.04293307519052179,
 'Education': 0.018995377199866697,
 'EducationField': 0.021344405902817342,
 'EmployeeCount': 0.0,
 'EmployeeNumber': 0.04741643672757769,
 'EnvironmentSatisfaction': 0.02574849423523357,
 'Gender': 0.008315523565898725,
 'HourlyRate': 0.04354031883385073,
 'JobInvolvement': 0.020667554744740236,
 'JobLevel': 0.025034663700501267,
 'JobRole': 0.03070761345668882,
 'JobSatisfaction': 0.024453469230002717,
 'MaritalStatus': 0.024241354353586826,
 'MonthlyIncome': 0.07372803913193865,
 'MonthlyRate': 0.04720777280578856,
 'NumCompaniesWorked': 0.03612969038437872,
 'Over18': 0.0,
 'OverTime': 0.058455210626630325,
 'PercentSalaryHike': 0.033420735547363266,
 'PerformanceRating': 0.0037573825676188066,
 'RelationshipSatisfaction': 0.020702126156517463,
 'StandardHours': 0.0,
 'StockOptionLevel': 0.03175541942579809,
 'TotalWorkingYears': 0.051979158896177734,
 'TrainingTimesLastYear': 0.028309767898874738,
 'WorkLifeBalance': 0.020941365684308315,
 'YearsAtCompany': 0.04143812755043779,
 'YearsInCurrentRole': 0.026763478247904385,
 'YearsSinceLastPromotion': 0.024518364918705857,
 'YearsWithCurrManager': 0.034401878418140554}
```

```
# Sort the importances
sorted_dict = sorted(imp_set.items(), key=lambda x: x[1], reverse=True)
sd1 = dict(sorted_dict)
sd1
```

```
{'MonthlyIncome': 0.07372803913193865,
 'Age': 0.05875217300119139,
 'OverTime': 0.058455210626630325,
 'TotalWorkingYears': 0.051979158896177734,
 'DailyRate': 0.04845097893769248,
 'EmployeeNumber': 0.04741643672757769,
 'MonthlyRate': 0.04720777280578856,
 'HourlyRate': 0.04354031883385073,
 'DistanceFromHome': 0.04293307519052179,
 'YearsAtCompany': 0.04143812755043779,
 'NumCompaniesWorked': 0.03612969038437872,
 'YearsWithCurrManager': 0.034401878418140554,
 'PercentSalaryHike': 0.033420735547363266,
 'StockOptionLevel': 0.03175541942579809,
 'JobRole': 0.03070761345668882,
 'TrainingTimesLastYear': 0.028309767898874738,
 'YearsInCurrentRole': 0.026763478247904385,
 'EnvironmentSatisfaction': 0.02574849423523357,
 'JobLevel': 0.025034663700501267,
 'YearsSinceLastPromotion': 0.02451836491870586,
 'JobSatisfaction': 0.024453469230002717,
 'MaritalStatus': 0.024241354353586826,
 'EducationField': 0.021344405902817342,
 'WorkLifeBalance': 0.020941365684308315,
 'RelationshipSatisfaction': 0.020702126156517463,
 'JobInvolvement': 0.020667554744740236,
 'Education': 0.018995377199866697,
 'Department': 0.01305507206094237,
 'BusinessTravel': 0.012834970598304098,
 'Gender': 0.008315523565898725,
 'PerformanceRating': 0.0037573825676188066,
 'EmployeeCount': 0.0,
 'Over18': 0.0,
 'StandardHours': 0.0}
```

```
# Find the average | we can drop less important columns
importances_mean = np.mean(list(sd1.values()))
importances_mean
```

```
0.029411764705882353
```

```
# Important attributes above average
important_attributes = [key for key, value in imp_set.items() if value >= importances_mean]
important_attributes
```

```
['Age',
 'DailyRate',
 'DistanceFromHome',
 'EmployeeNumber',
 'HourlyRate',
 'JobRole',
 'MonthlyIncome',
 'MonthlyRate',
 'NumCompaniesWorked',
 'OverTime',
 'PercentSalaryHike',
 'StockOptionLevel',
 'TotalWorkingYears',
 'YearsAtCompany',
 'YearsWithCurrManager']
```

```
# Reassign X (Train Set)
X = X[important_attributes]
X
```

	Age	DailyRate	DistanceFromHome	EmployeeNumber	HourlyRate	JobRole	MonthlyIncome	MonthlyRate	NumCompaniesWorked	OverTime	Pe
0	41	1102	1	1	94	7	5993	19479	8	1	
1	49	279	8	2	61	6	5130	24907	1	0	
2	37	1373	2	4	92	2	2090	2396	6	1	
3	33	1392	3	5	56	6	2909	23159	1	1	
4	27	591	2	7	40	2	3468	16632	9	0	
...	...	...	...	...	...	...	...	...	...	...	...
1465	36	884	23	2061	41	2	2571	12290	4	0	

Model Building

```
# Cross Validation

# Apply 10-fold cross-validation
scores_dt = cross_val_score(decision_tree_model, X, y, cv=10)
scores_rf = cross_val_score(random_forest_model, X, y, cv=10)

scores_rf
```

```
array([0.82993197, 0.85034014, 0.85034014, 0.85714286, 0.8707483 ,
       0.82993197, 0.85034014, 0.85034014, 0.85714286, 0.85034014])
```

```
scores_dt.mean()
```

```
0.7340136054421768
```

```
scores_rf.mean()
```

```
0.8496598639455781
```

Model Training, Testing, and Evaluation

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Creating an instance of the DecisionTreeClassifier
decision_tree_model = DecisionTreeClassifier()
random_forest_model = RandomForestClassifier()



# Training the model on the training data
decision_tree_model.fit(X_train, y_train)
random_forest_model.fit(X_train, y_train)

# Making predictions on the testing data
y_pred_dt = decision_tree_model.predict(X_test)
y_pred_rf = decision_tree_model.predict(X_test)



# Calculating the accuracy of the model
accuracy_dt = accuracy_score(y_test, y_pred_dt)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# Printing the accuracy of the model
print(f"The accuracy of the decision tree model is: {accuracy_dt}")
print(f"The accuracy of the random forest model is: {accuracy_rf}")
```

```
The accuracy of the decision tree model is: 0.7717391304347826
The accuracy of the random forest model is: 0.7717391304347826
```

```
class_names = list(y.unique())
print(classification_report(y_test, y_pred_rf, target_names=class_names))
print(confusion_matrix(y_test, y_pred_rf))
```