Luna Baalbaki - LSE Datathon 2022 ¶

Email: l.baalbaki@lse.ac.uk (mailto:l.baalbaki@lse.ac.uk)

Introduction:

IBM is struggling from a 16% employee attrition rate. But luckily they have great data that we can help to analyze. With the use of machine learning models and parameter optimization we can clearly point out to a few of the top reasons and signals that contribute to this high rate. We could also to help the HR department develop strategies to decrease the employee attrition rate.

In [ ]:
```
# Luna Baalbaki - LSE Datathon 2022

# Introduction:
The aim behind this notebook is to predict employee turnover in the IBM
```

In [205]:
```
# Import libraries
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import pandas as pd
import seaborn as sns
import matplotlib as plt
import matplotlib.pyplot as plt
import numpy as np
from numpy import mean, std, percentile
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, train_test_split,
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, Standa
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.exceptions import NotFittedError
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingReg
from sklearn.feature_selection import RFECV
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

In [206]:
```
df=pd.read_csv("/Users/user/Desktop/LSE/HR_IBM.csv")
```

In [207]:
```
df.head(10)
```

Out[207]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educatio
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sc
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sc
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sc
4	27	No	Travel_Rarely	591	Research & Development	2	1	M
5	32	No	Travel_Frequently	1005	Research & Development	2	2	Life Sc
6	59	No	Travel_Rarely	1324	Research & Development	3	3	M
7	30	No	Travel_Rarely	1358	Research & Development	24	1	Life Sc
8	38	No	Travel_Frequently	216	Research & Development	23	3	Life Sc
9	36	No	Travel_Rarely	1299	Research & Development	27	3	M

10 rows × 35 columns

Data Cleaning

In [208]:
```
# Are there any duplicates in the dataset?
dups = df.duplicated()
print(dups.any())
print('Yay! No duplicates!')
```

```
False
Yay! No duplicates!
```

In [209]:
```
# Check for missing data
df.isna().sum()
```

Out[209]:
```
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
```

In [210]:
```
# Display summary statistics
df.describe()
```

Out[210]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumb
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.0000(
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.8653(
std	9.135373	403.509100	8.106864	1.024165	0.0	602.0243:
min	18.000000	102.000000	1.000000	1.000000	1.0	1.0000(
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.2500(
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.5000(
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.7500(
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.0000(

8 rows × 26 columns

Exploratory Data Analysis (EDA)

In this section, we will visualize the relationship between features and the target variable.

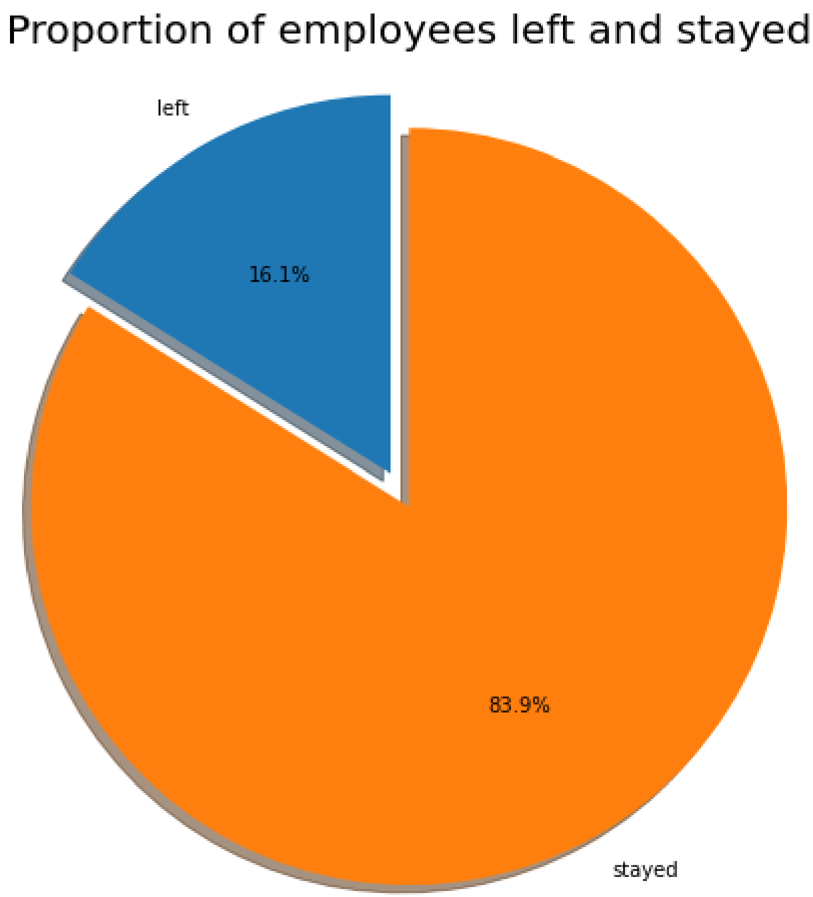
This would help us interpret which features affect customer churn through graphical representation.

In [211]:
```
# How many employees are churning?
df['Attrition'].value_counts()
```

Out[211]:
```
No     1233
Yes     237
Name: Attrition, dtype: int64
```

In [212]:

```
labels = 'left', 'stayed'
sizes = [df.Attrition[df['Attrition']=='Yes'].count(), df.Attrition[df[
explode = (0, 0.1)
fig1, ax1 = plt.subplots(figsize=(10, 8))
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
plt.title("Proportion of employees left and stayed", size = 20)
plt.show()
```

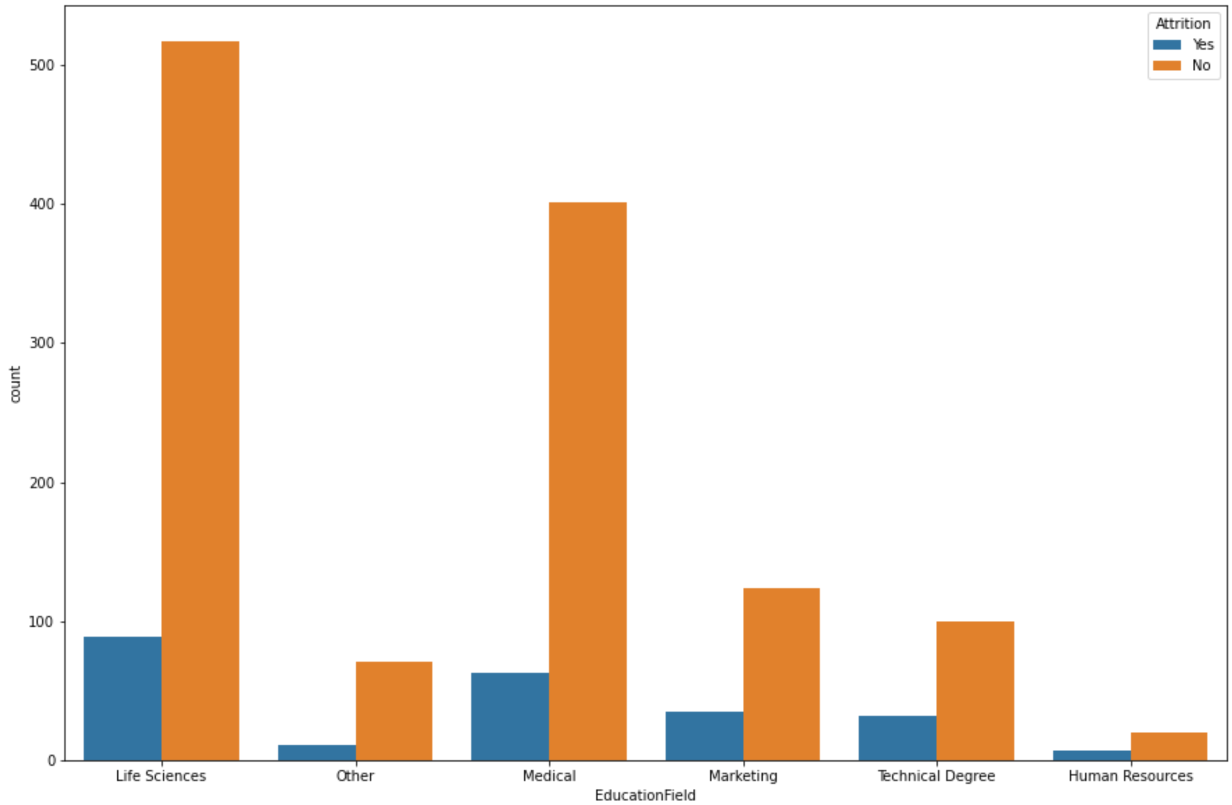


Note: 17.5% employees have decided to discontinue their services with the company, which is a major source of profit loss.

Moving forward, we aim to identify the attributes and characteristics of churners, in hopes of predicting the likelihood of future churners. Subsequently, we would be able to design personalized marketing campaigns to ensure employee retention.

In [222]:
```
# Visualize the churn count by education field


fig = plt.gcf()
fig.set_size_inches(15,10)
sns.countplot(x='EducationField', hue = 'Attrition',data = df)
```

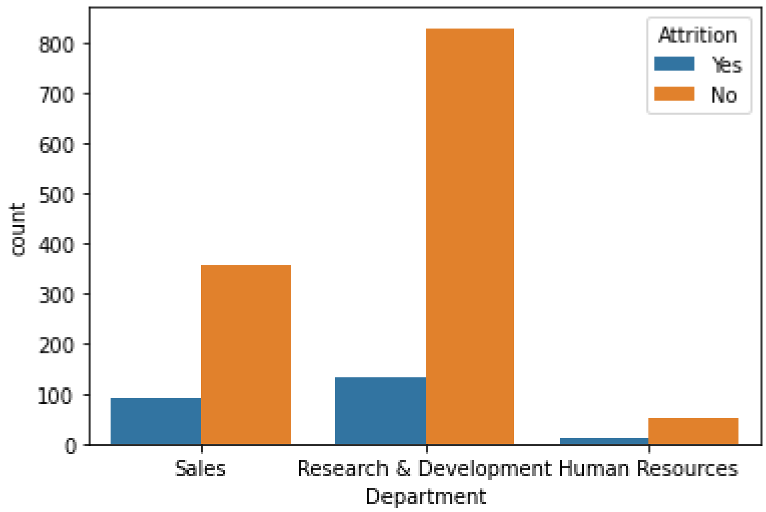
Out[222]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6aff8e1f0>



In [214]:
```
#People from Life Science Degrees are more likely to attrition
```

In [215]:
```
# Visualize the churn count for Department
sns.countplot(x='Department', hue='Attrition', data= df)

```

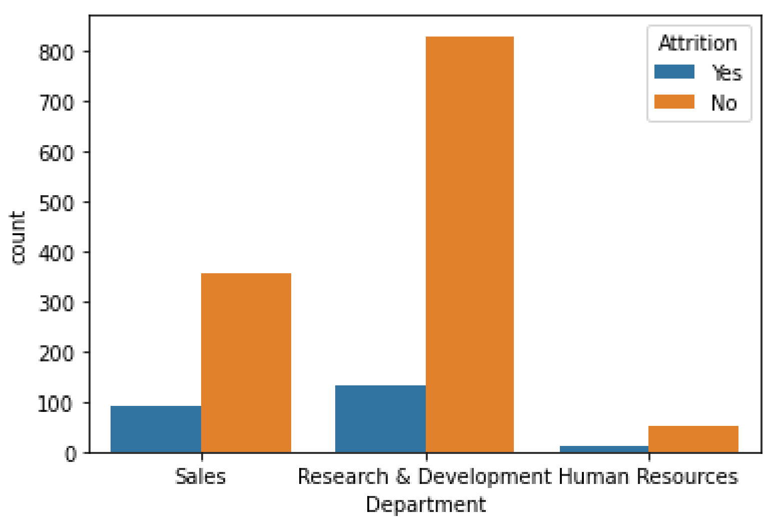
Out[215]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6b0812040>



In [216]:
```
#Employees from the Research Department left the company more than from
```

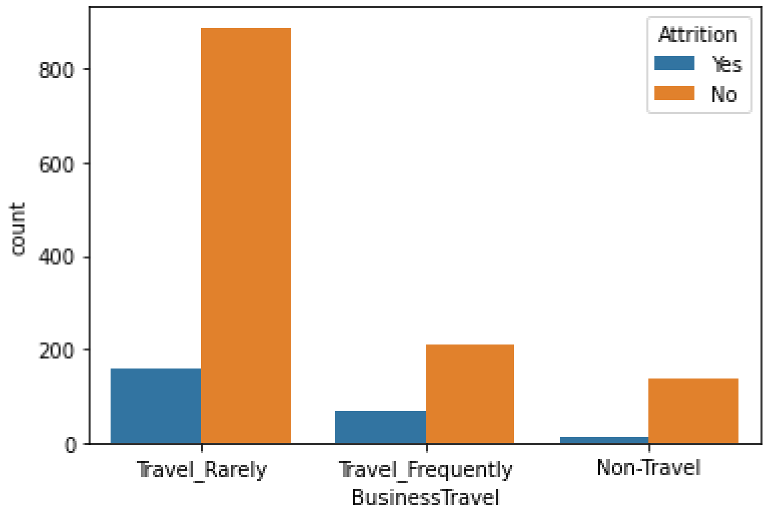
In [217]:
```
#People from Life Science Degrees are more likely to attrition

# Visualize the churn count for Department
sns.countplot(x='Department', hue='Attrition', data= df)


#Employees from the Research Department left the company more than from
```

Out[217]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6b24449d0>



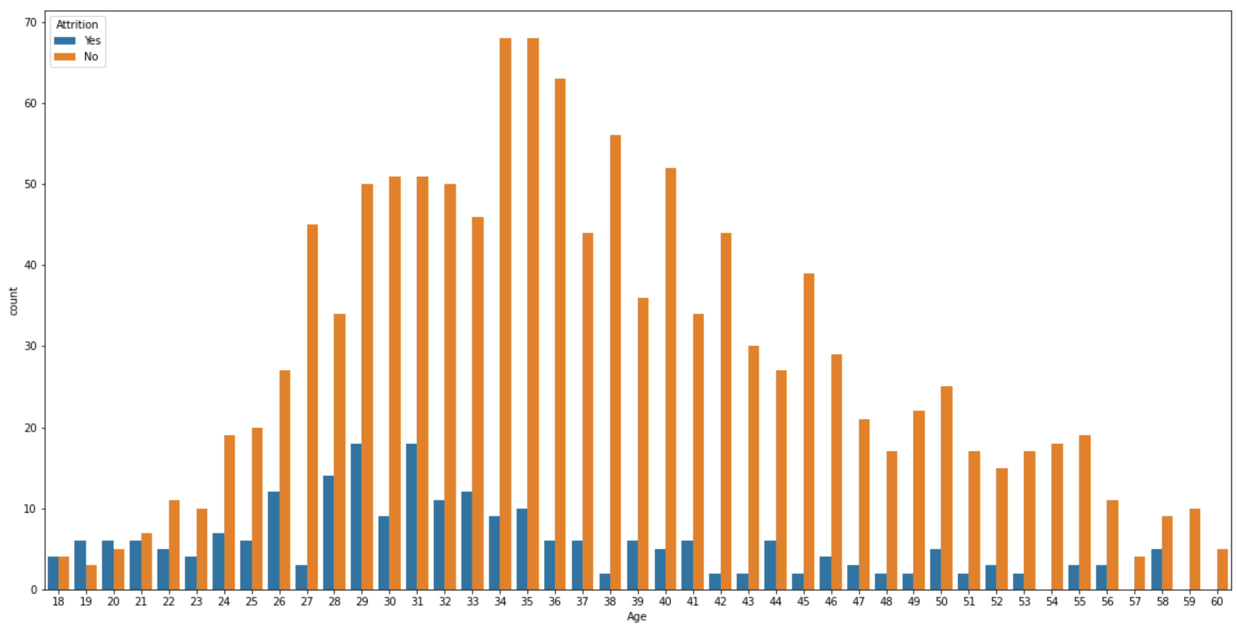
In [218]:
```
# Visualize the churn count for Business Travel
sns.countplot(x='BusinessTravel', hue='Attrition', data= df)
```

Out[218]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6b113a400>



In [219]:
```
# Visualize the churn count for age


fig = plt.gcf()
fig.set_size_inches(20,10)
sns.countplot(x='Age', hue='Attrition', data= df)
```

Out[219]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6aff90820>



In [ ]:

```
import numpy as np; np.random.seed(0)
import matplotlib.pyplot as plt
import pandas as pd

x = np.random.rand(120)
df = pd.DataFrame({"x":x})

bins= [10,20,30,40]
plt.hist(df.values, bins=bins, edgecolor="k")
plt.xticks(bins)

plt.show()
```

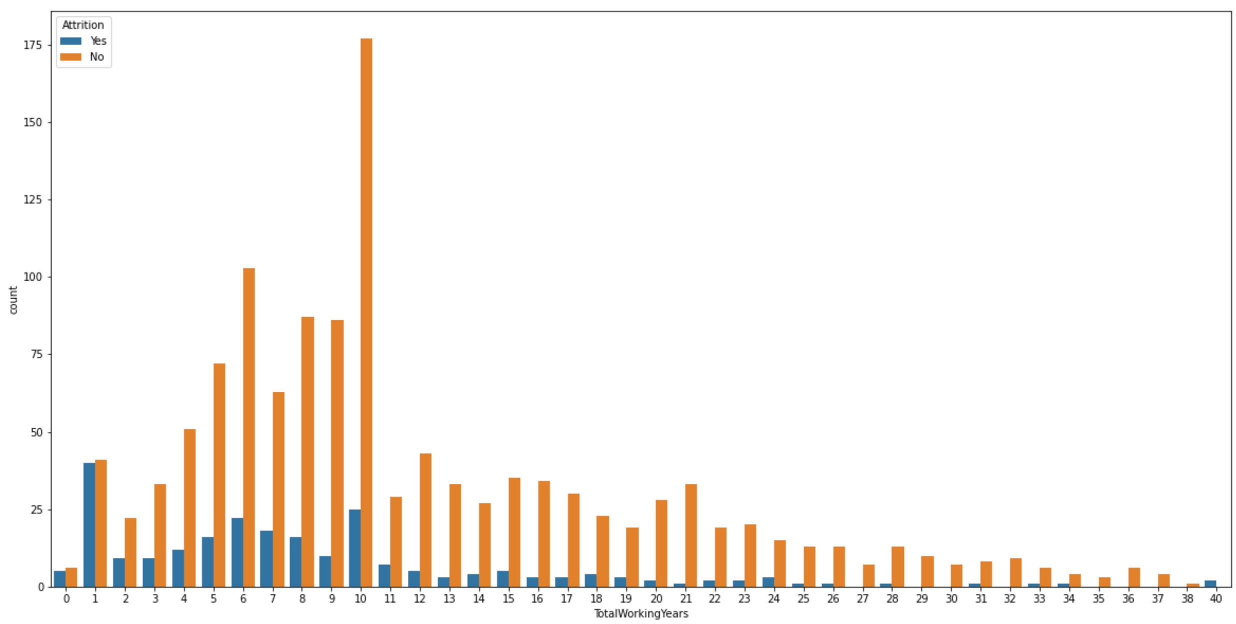
In [178]:

```
#People in their early 30s tend to churn more
```

In [179]:

```
# Visualize the churn count for TotalWorkingYears


fig = plt.gcf()
fig.set_size_inches(20,10)
sns.countplot(x='TotalWorkingYears', hue='Attrition', data= df)
```

Out[179]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6b44aba00>



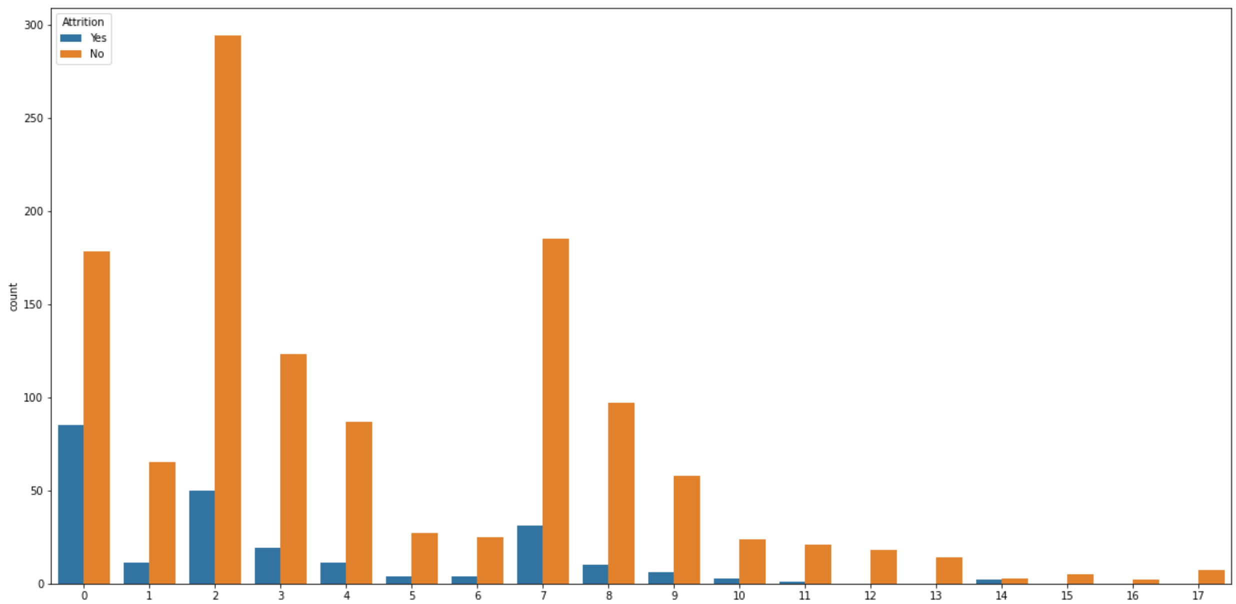
In [180]:

```
#People with less woring years tend to attrition more
```

In [181]:

```
# Visualize the churn count for YearsWithCurrManager


fig = plt.gcf()
fig.set_size_inches(20,10)
sns.countplot(x='YearsWithCurrManager', hue='Attrition', data= df)
```

Out[181]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd6b1b55f70>



Feature Engineering

In [112]:

```
### One-hot Encoding
```

In [182]:

```
#Split teh data into train and test
y = df['Attrition'].reset_index(drop=True)
X = df.drop(columns='Attrition')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

```

In [184]:

```
# Defining a function for OneHotEncoding that retains feature names (S
# Reference: https://github.com/gdiepen/PythonScripts/blob/master/data

class DataFrameOneHotEncoder(BaseEstimator, TransformerMixin):
    """Specialized version of OneHotEncoder that plays nice with panda
    will automatically set the feature/column names after fit/transfor
    """

    def __init__(
        self,
        categories="auto",
        drop=None,
        sparse=None,
        dtype=np.float64,
        handle_unknown="error",
        col_overrule_params={},
    ):
        """Create DataFrameOneHotEncoder that can be fitted to and tra
        and that will set up the column/feature names automatically to
        original_column_name[categorical_value]
        If you provide the same arguments as you would for the sklearn
        OneHotEncoder, these parameters will apply for all of the colu
        to have specific overrides for some of the columns, provide th
        argument col_overrule_params.

        For example:
            DataFrameOneHotEncoder(col_overrule_params={"col2":{"drop"
        will create a OneHotEncoder for each of the columns with defau
        uses a drop=first argument for columns with the name col2
        Args:
            categories'auto' or a list of array-like, default='auto'
                'auto' : Determine categories automatically from the t
                list : categories[i] holds the categories expected in 
                The passed categories should not mix strings and numer
                within a single feature, and should be sorted in case 
                values.
            drop: {'first', 'if_binary'} or a array-like of shape (n_f
                default=None
                See OneHotEncoder documentation
            sparse: Ignored, since we always will work with dense data
            dtype: number type, default=float
                Desired dtype of output.
            handle_unknown: {'error', 'ignore'}, default='error'
                Whether to raise an error or ignore if an unknown cate
                is present during transform (default is to raise). Whe
                is set to 'ignore' and an unknown category is encounte
                transform, the resulting one-hot encoded columns for t
                be all zeros. In the inverse transform, an unknown cat
                denoted as None.
            col_overrule_params: dict of {column_name: dict_params} wh
                are exactly the options cateogires,drop,sparse,dtype,h
                For the column given by the key, these values will ove
                parameters
        """
        self.categories = categories
        self.drop = drop
```

```
        self.sparse = sparse
        self.dtype = dtype
        self.handle_unknown = handle_unknown
        self.col_overrule_params = col_overrule_params
        pass

    def fit(self, X, y=None):
        """Fit a separate OneHotEncoder for each of the columns in the
        Args:
            X: dataframe
            y: None, ignored. This parameter exists only for compatibil
                Pipeline
        Returns
            self
        Raises
            TypeError if X is not of type DataFrame
        """
        if type(X) != pd.DataFrame:
            raise TypeError(f"X should be of type dataframe, not {type

        self.onehotencoders_ = []
        self.column_names_ = []

        for c in X.columns:
            # Construct the OHE parameters using the arguments
            ohe_params = {
                "categories": self.categories,
                "drop": self.drop,
                "sparse": False,
                "dtype": self.dtype,
                "handle_unknown": self.handle_unknown,
            }
            # and update it with potential overrule parameters for the
            ohe_params.update(self.col_overrule_params.get(c, {}))

            # Regardless of how we got the parameters, make sure we al
            # sparsity to False
            ohe_params["sparse"] = False

            # Now create, fit, and store the onehotencoder for current
            ohe = OneHotEncoder(**ohe_params)
            self.onehotencoders_.append(ohe.fit(X.loc[:, [c]]))

            # Get the feature names and replace each x0_ with empty an
            # surround the categorical value with [] and prefix it wit
            # column name
            feature_names = ohe.get_feature_names()
            feature_names = [x.replace("x0_", "") for x in feature_name
            feature_names = [f"{c}[{x}]" for x in feature_names]

            self.column_names_.append(feature_names)

        return self

    def transform(self, X):
        """Transform X using the one-hot-encoding per column
        Args:
```

```
            X: Dataframe that is to be one hot encoded
        Returns:
            Dataframe with onehotencoded data
        Raises
            NotFittedError if the transformer is not yet fitted
            TypeError if X is not of type DataFrame
        """
        if type(X) != pd.DataFrame:
            raise TypeError(f"X should be of type dataframe, not {type

        if not hasattr(self, "onehotencoders_"):
            raise NotFittedError(f"{type(self).__name__} is not fitted

        all_df = []

        for i, c in enumerate(X.columns):
            ohe = self.onehotencoders_[i]

            transformed_col = ohe.transform(X.loc[:, [c]])

            df_col = pd.DataFrame(transformed_col, columns=self.column_
            all_df.append(df_col)

        return pd.concat(all_df, axis=1)
```

In [185]:

```

# Let's use the new function for OneHotEncoding
# Reference: https://www.guidodiepen.nl/2021/02/keeping-column-names-wh

df = X_train[['BusinessTravel', 'Department', 'EducationField', 'Gender
df_ohe = DataFrameOneHotEncoder()
ohe_done = pd.DataFrame(df_ohe.fit_transform(df))
# Merge both back into one dataframe
X_train.reset_index(inplace=True)
X_train = pd.concat([X_train, ohe_done], axis=1)
# Drop old columns
X_train = X_train.drop(['BusinessTravel', 'Department', 'EducationField
X_train.reset_index(inplace=True)
X_train.drop('index', axis=1, inplace=True)
X_train.drop('level_0', axis=1, inplace=True)

```

In [186]:
```
X_train.head()
```

Out[186]:

JobRole[Manufacturing Director]	JobRole[Research Director]	JobRole[Research Scientist]	JobRole[Sales Executive]	JobRole[Sales Representative]	MaritalS
1.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	
0.0	0.0	1.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	
0.0	1.0	0.0	0.0	0.0	

In [187]:
```
# Let's not forget to encode the Target Variable using LabelEncoder!
# But to avoid data leakage, let's split the dataset
```

In [188]:
```
# Now we encode the target variable
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
y_train = pd.DataFrame(le.fit_transform(y_train))
y_test = pd.DataFrame(le.transform(y_test))
```

In [189]:
```
# Reset indices for easier interpretation
pd.options.mode.chained_assignment = None  # Disables harmless 'Setting
X_train.reset_index(inplace=True)
X_test.reset_index(inplace=True)
X_train.drop('index', axis=1, inplace=True)
X_test.drop('index', axis=1, inplace=True)
```

In [190]:
```
X_train.drop('Over18',axis='columns', inplace=True)
```

In [191]:
```
X_train.drop('OverTime',axis='columns', inplace=True)
```

In [192]:
```
# How does our revised train dataset look?
X_train.head()
```

Out[192]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentS
0	37	370	10	4	1	1809	
1	40	543	1	4	1	2012	
2	32	977	2	3	1	1671	
3	28	791	1	4	1	1286	
4	34	419	7	4	1	28	

5 rows × 52 columns

In [193]:

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1176 entries, 0 to 1175
Data columns (total 52 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   Age                                 1176 non-null   int64  
 1   DailyRate                           1176 non-null   int64  
 2   DistanceFromHome                    1176 non-null   int64  
 3   Education                           1176 non-null   int64  
 4   EmployeeCount                       1176 non-null   int64  
 5   EmployeeNumber                      1176 non-null   int64  
 6   EnvironmentSatisfaction             1176 non-null   int64  
 7   HourlyRate                          1176 non-null   int64  
 8   JobInvolvement                      1176 non-null   int64  
 9   JobLevel                            1176 non-null   int64  
 10  JobSatisfaction                     1176 non-null   int64  
 11  MonthlyIncome                       1176 non-null   int64  
 12  MonthlyRate                         1176 non-null   int64  
 13  NumCompaniesWorked                  1176 non-null   int64  
 14  PercentSalaryHike                   1176 non-null   int64  
 15  PerformanceRating                   1176 non-null   int64  
 16  RelationshipSatisfaction            1176 non-null   int64  
 17  StandardHours                       1176 non-null   int64  
 18  StockOptionLevel                    1176 non-null   int64  
 19  TotalWorkingYears                   1176 non-null   int64  
 20  TrainingTimesLastYear               1176 non-null   int64  
 21  WorkLifeBalance                     1176 non-null   int64  
 22  YearsAtCompany                      1176 non-null   int64  
 23  YearsInCurrentRole                  1176 non-null   int64  
 24  YearsSinceLastPromotion             1176 non-null   int64  
 25  YearsWithCurrManager                1176 non-null   int64  
 26  BusinessTravel[Non-Travel]          1176 non-null   float64
 27  BusinessTravel[Travel_Frequently]   1176 non-null   float64
 28  BusinessTravel[Travel_Rarely]       1176 non-null   float64
 29  Department[Human Resources]         1176 non-null   float64
 30  Department[Research & Development]  1176 non-null   float64
 31  Department[Sales]                   1176 non-null   float64
 32  EducationField[Human Resources]     1176 non-null   float64
 33  EducationField[Life Sciences]       1176 non-null   float64
 34  EducationField[Marketing]           1176 non-null   float64
 35  EducationField[Medical]             1176 non-null   float64
 36  EducationField[Other]               1176 non-null   float64
 37  EducationField[Technical Degree]    1176 non-null   float64
 38  Gender[Female]                      1176 non-null   float64
 39  Gender[Male]                        1176 non-null   float64
 40  JobRole[Healthcare Representative]  1176 non-null   float64
 41  JobRole[Human Resources]            1176 non-null   float64
 42  JobRole[Laboratory Technician]      1176 non-null   float64
 43  JobRole[Manager]                    1176 non-null   float64
 44  JobRole[Manufacturing Director]     1176 non-null   float64
 45  JobRole[Research Director]          1176 non-null   float64
 46  JobRole[Research Scientist]         1176 non-null   float64
 47  JobRole[Sales Executive]            1176 non-null   float64
 48  JobRole[Sales Representative]       1176 non-null   float64
 49  MaritalStatus[Divorced]             1176 non-null   float64
```

```
 50  MaritalStatus[Married]            1176 non-null   float64
 51  MaritalStatus[Single]             1176 non-null   float64
dtypes: float64(26), int64(26)
memory usage: 477.9 KB
```

Model Optimization and Selection

In [194]:

```python
#Support vetctor Machine
svr = SVR(kernel='rbf')
svr.fit(X_train, y_train.values.ravel())

svr_t_predictions = svr.predict(X_train)
svr_train_rmse = mean_squared_error(y_train, svr_t_predictions, squared
print(f"RMSE for testing: {svr_train_rmse:.1f}")

```

```
RMSE for testing: 0.4
```

In [195]:

```python
#Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train.values.ravel())

lr_tr_predictions = lr.predict(X_train)
lr_train_rmse = mean_squared_error(y_train, lr_tr_predictions, squared=
print(f"RMSE for testing: {lr_train_rmse:.1f}")


```

```
RMSE for testing: 0.3
```

In [196]:

```python
#random Forest Regressor
rfr = RandomForestRegressor(n_estimators=10, random_state=1111)
rfr.fit(X_train, y_train.values.ravel())

rfr_tr_predictions = rfr.predict(X_train)
rfr_train_rmse = mean_squared_error(y_train, rfr_tr_predictions, square
print(f"RMSE for testing: {rfr_train_rmse:.1f}")


```

```
RMSE for testing: 0.2
```

Note

Random Forrest Regresosr Model has the least error among the other Machine Learning models. Thus, we go with Random Forrest Regressor

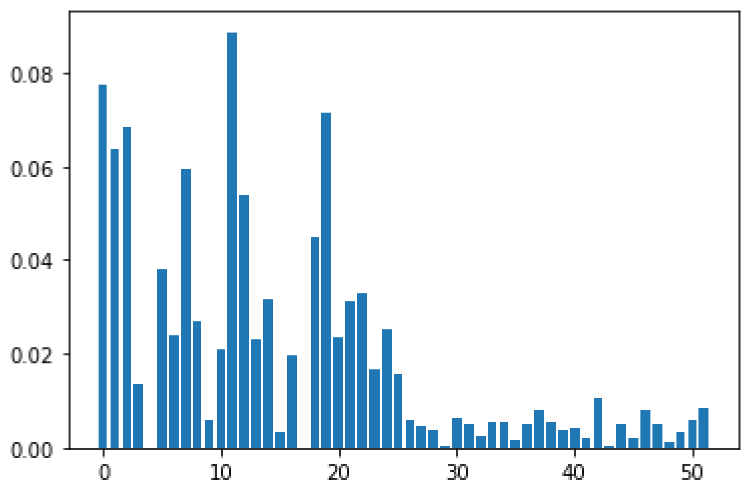
Parameter Optimization

In [197]:

```
#Paraemter Optimization
# Optimize the parameters using GridSearchCV
parameters = {'n_estimators': [4, 6, 9],
              'max_features': ['log2', 'sqrt','auto'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1,5,8]
             }
rfr_cv = GridSearchCV(estimator= rfr, param_grid = parameters, scoring
rfr_cv = rfr_cv.fit(X_train, y_train.values.ravel())
```

In [198]:

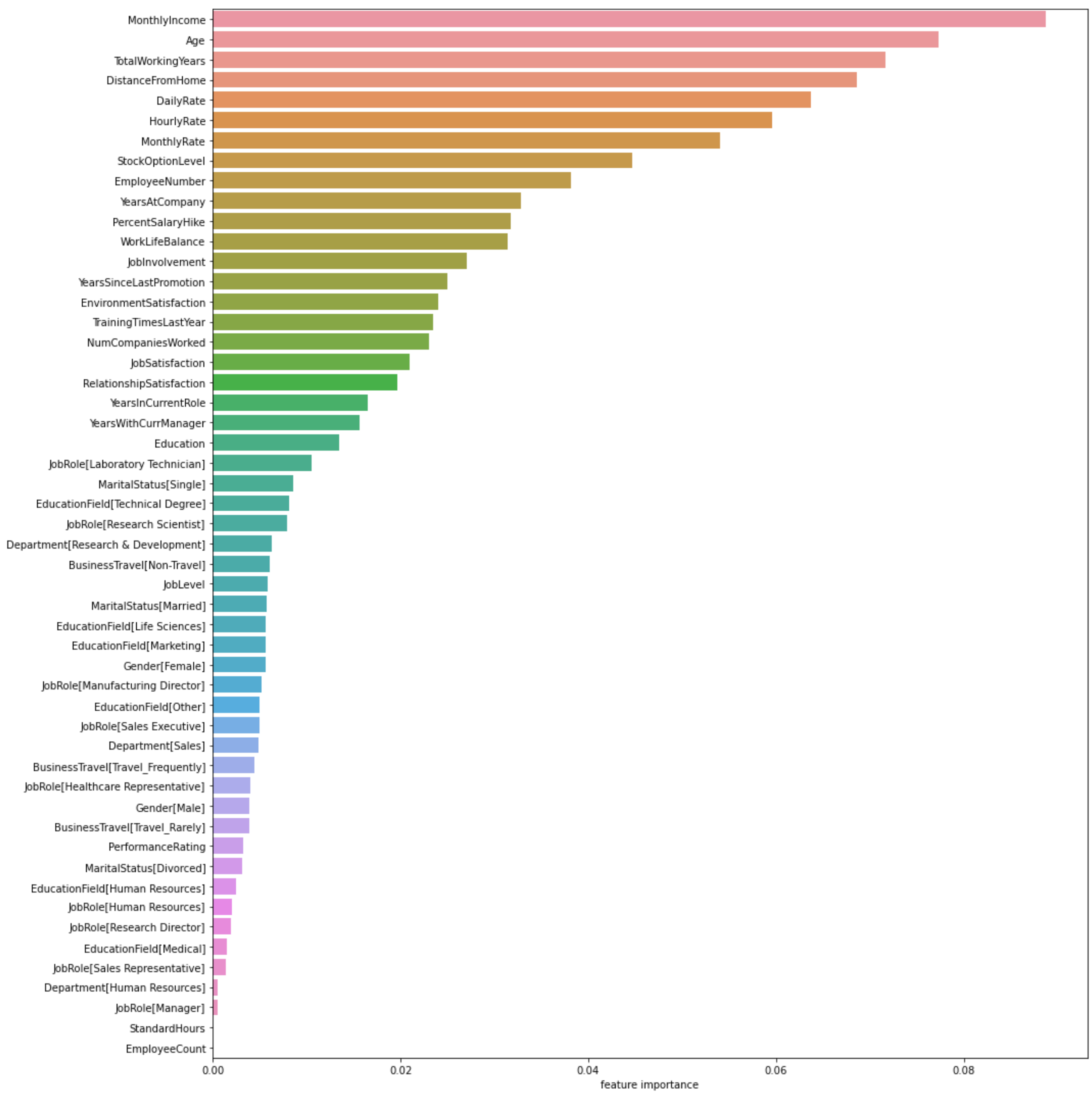
```
#Feature Selection

from matplotlib import pyplot
# get importance
importance = rfr.feature_importances_

## summarize feature importance (commented to save space)
#for i,v in enumerate(importance):
    #print('Feature: %0d, Score: %.5f' % (i,v))

# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```



Feature Selection

In [200]:

```
ranking = np.argsort(-rfr.feature_importances_)
f, ax = plt.subplots(figsize=(15, 15))
sns.barplot(x=rfr.feature_importances_[ranking], y=X_train.columns.valu
ax.set_xlabel("feature importance")
plt.tight_layout()
plt.show()
```



Note:

Based on the model feature importance, the top 3 features are (Monthly Income, Age and Total Working years). These are the features that affect the willingness of the employee to leave the company or not.

This seems logical as an employee's salary is one of the biggest reasons for an employee to leave their job or not.

The second affecting variable in employee attrition is the age of an employee. So as it shows that age plays a significant role in the attrition of the employee. So here we can inform the HR department to put some strategies depending on ages and especially the employees in their late 20s and early 30s as we have seen earlier in the explatory analysis that they tend to leave the company more than others. SO the HR can target theur strategy more towards employees of these ages and focus on them more so that they won't leave the company.

The third affecting variable in employee attrition is total working years. The longer an employee stays in the company the less likely they tend to leave the company. Here the HR department could work on strategies so that the employee would stay longer in the company.

In [ ]:

```
```