Employee Burnout Prediction

Employee burnout is a state of physical, emotional and mental exhaustion caused by excessive and prolonged stress. It can have serious consequences on an individual's well-being and can lead to decreased productivity and job performance. In today's fast-paced and constantly connected world, it is increasingly important to recognize and address the signs of burnout in order to maintain the health and well-being of employees.

we will be exploring the use of regression techniques to predict employee burnout. By analyzing a dataset containing various factors that may contribute to burnout such as workload, mental fatigue job and work-life balance, we can develop a model to identify individuals who may be at risk of burnout. By proactively addressing these risk factors, organizations can help prevent burnout and promote the well-being of their employees.

Dataset: Are Your Employees Burning Out?

This dataset consists of 9 columns as follows:

• **Employee ID:** The unique ID allocated for each employee **(example: fffe390032003000)**

• **Date of Joining:** The date-time when the employee has joined the organization **(example: 2008-12-30)**

• **Gender:** The gender of the employee **(Male/Female)**

• **Company Type:** The type of company where the employee is working **(Service/Product)**

• **WFH Setup Available:** is the work from home facility available for the employee **(Yes/No)**

• **Designation:** The designation of the employee of work in the organization. In the **range of [0.0, 5.0]** bigger is higher designation.

• **Resource Allocation:** The amount of resource allocated to the employee to work, Ie number of working hours in the **range of [1.0, 10.0]** (higher means more resource)

• **Mental Fatigue Score:** The level of fatigue mentally the employee is facing in the **range of [0.0, 10.0]** where 0.0 means no fatigue and 10.0 means completely fatigue

• **Burn Rate:** The value we need to predict for each employee telling the rate of Bur out while working. In the **range of [0.0, 1.0]** where the higher the value is more is the burn out.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
import pickle as pickle
import os
```

LOADING DATASET

```
data = pd.read_excel("/content/employee_burnout.xlsx")
```

DATA OVERVIEW

```
data.head()
```

	EmployeeID	Date_of_Joining	Gender	Company_Type	WFH_Setup_Available
0	fffe32003000360033003200	2008-09-30	Female	Service	No
1	fffe3700360033003500	2008-11-30	Male	Service	Yes
2	fffe31003300320037003900	2008-03-10	Female	Product	Yes
3	fffe32003400380032003900	2008-11-03	Male	Service	Yes
4	fffe31003900340031003600	2008-07-24	Female	Service	No

```
data.describe()
```

	Date_of_Joining	Designation	Resource_Allocation	Mental_Fatigue_Score	Bur
count	22750	22750.000000	21369.000000	20633.000000	21626
mean	2008-07-01 09:28:05.274725120	2.178725	4.481398	5.728188	0
min	2008-01-01 00:00:00	0.000000	1.000000	0.000000	0
25%	2008-04-01 00:00:00	1.000000	3.000000	4.600000	0
50%	2008-07-02 00:00:00	2.000000	4.000000	5.900000	0

```
data.nunique()
```

```
EmployeeID              22750
Date_of_Joining           366
Gender                      2
Company_Type                2
WFH_Setup_Available         2
Designation                 6
Resource_Allocation        10
Mental_Fatigue_Score      101
Burn_Rate                 101
dtype: int64
```

```
data.columns.tolist()
```

```
['EmployeeID',
 'Date_of_Joining',
 'Gender',
 'Company_Type',
 'WFH_Setup_Available',
 'Designation',
 'Resource_Allocation',
 'Mental_Fatigue_Score',
 'Burn_Rate']
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22750 entries, 0 to 22749
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   EmployeeID            22750 non-null  object
 1   Date_of_Joining       22750 non-null  datetime64[ns]
 2   Gender                22750 non-null  object
```

```
 3   Company_Type          22750 non-null  object
 4   WFH_Setup_Available   22750 non-null  object
 5   Designation           22750 non-null  int64
 6   Resource_Allocation   21369 non-null  float64
 7   Mental_Fatigue_Score  20633 non-null  float64
 8   Burn_Rate             21626 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(1), object(4)
memory usage: 1.6+ MB
```

```
data.isnull().sum()
```

```
EmployeeID                 0
Date_of_Joining            0
Gender                     0
Company_Type               0
WFH_Setup_Available        0
Designation                0
Resource_Allocation     1381
Mental_Fatigue_Score    2117
Burn_Rate               1124
dtype: int64
```

```
data.isnull().sum().values.sum()
```

```
4622
```

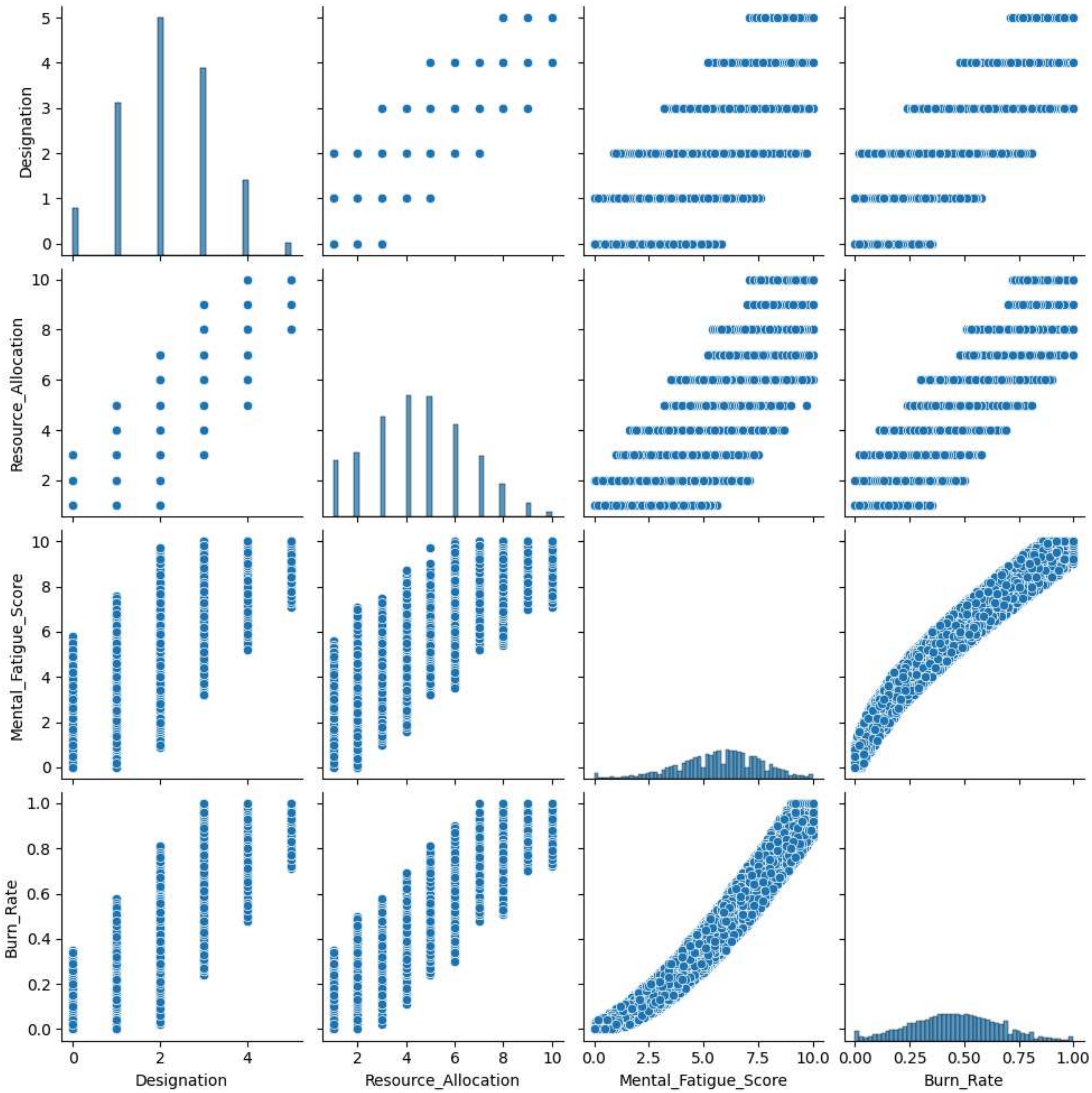
EXPLORATORY DATA ANALYSIS

```
data.corr(numeric_only=True)['Burn_Rate'][:-1]
```

```
Designation             0.737556
Resource_Allocation     0.856278
Mental_Fatigue_Score    0.944546
Name: Burn_Rate, dtype: float64
```

These two variables are strongly collerated with target variable,therefore,important to estimate it

```
sns.pairplot(data)
plt.show()
```


Designation
Resource_Allocation
Mental_Fatigue_Score
Burn_Rate

Drop off all observations with NaN values of our dataframe

```
data=data.dropna()
```

```
data.shape
```

```
(18590, 9)
```

Analsing what type of data is each variable

```
data.dtypes
```

```
EmployeeID                        object
Date_of_Joining           datetime64[ns]
Gender                            object
Company_Type                      object
WFH_Setup_Available               object
Designation                        int64
Resource_Allocation              float64
Mental_Fatigue_Score             float64
Burn_Rate                        float64
dtype: object
```

The values that each variable contains.

The employees ID doesn't provide any useful information and, therefore, they must be dropped.

```
data = data.drop('EmployeeID', axis = 1)
```

Checking the correlation of Date of Joining with Target variable

```
print(f"Min date {data['Date_of_Joining'].min()}")
```

```
Min date 2008-01-01 00:00:00
```

```
print(f"Max date {data['Date_of_Joining'].max()}")
```

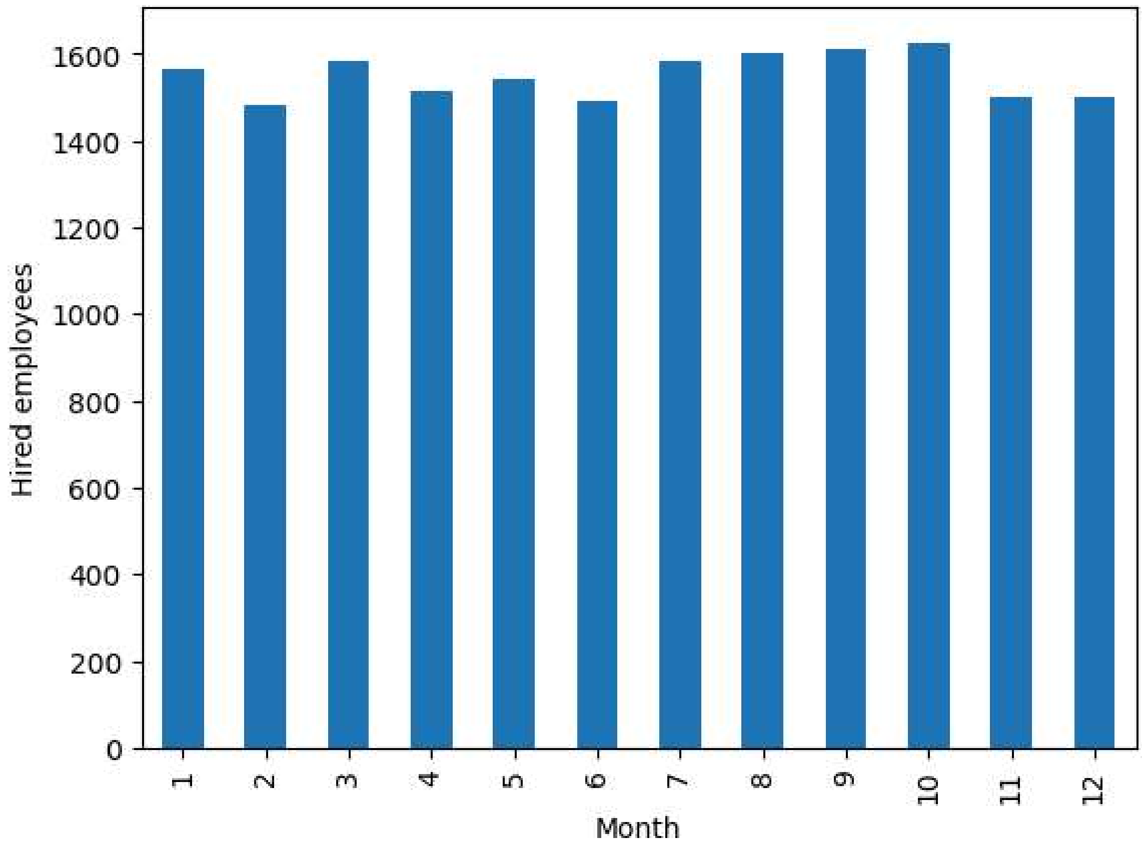
```
Max date 2008-12-31 00:00:00
```

```
data_month = data.copy()
```

```
data_month["Date_of_Joining"] = data_month['Date_of_Joining'].astype("datetime64[ns]")  # Sp
```

```
data_month["Date_of_Joining"].groupby(data_month['Date_of_Joining'].dt.month).count().plot(k
```

```
<Axes: xlabel='Month', ylabel='Hired employees'>
```



The date of joining is uniform distributed with values between 2008-01-01 and 2008-12-31. So in order to create a new feature which represents the labor seniority, we could create a variable with de days worked

```
data_2008 = pd.to_datetime(["2008-01-01"]*len(data)) # Specify time unit as nanoseconds wher
data["Days"] = data['Date_of_Joining'].astype("datetime64[ns]").sub(data_2008).dt.days
data.Days
```

```
0        273
1        334
3        307
4        205
5        330
        ...
22743    349
22744    147
22746     18
22748      9
22749      5
Name: Days, Length: 18590, dtype: int64
```

```
# Select only numeric columns before calculating correlation
numeric_data = data.select_dtypes(include=['number'])
correlation = numeric_data.corr()['Burn_Rate']
print(correlation)
```

```
Designation             0.736412
Resource_Allocation     0.855005
Mental_Fatigue_Score    0.944389
Burn_Rate               1.000000
Days                    0.000309
Name: Burn_Rate, dtype: float64
```

```
data.corr(numeric_only=True)['Burn_Rate'][:]
```

```
Designation             0.736412
Resource_Allocation     0.855005
Mental_Fatigue_Score    0.944389
Burn_Rate               1.000000
Days                    0.000309
Name: Burn_Rate, dtype: float64
```

We observed that there is no strong correlation between Date of Joining and Burn Rate.So, we are dropping the column Date of Joining.

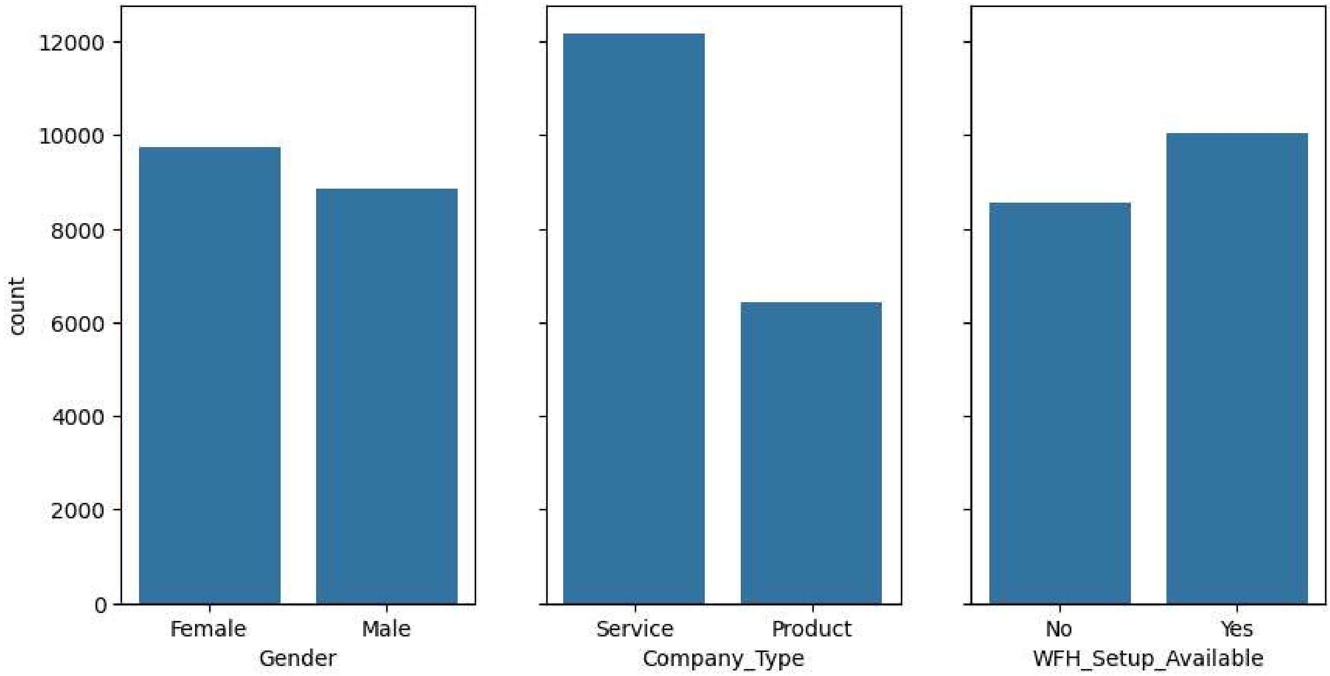
```
data = data.drop(['Date_of_Joining','Days'], axis = 1)
```

```
data.head()
```

	Gender	Company_Type	WFH_Setup_Available	Designation	Resource_Allocation	Mental_F
0	Female	Service	No	2	3.0	
1	Male	Service	Yes	1	2.0	
3	Male	Service	Yes	1	1.0	
4	Female	Service	No	3	7.0	
5	Male	Product	Yes	2	4.0	

Now analysing the categorical variables

```
cat_columns = data.select_dtypes(object).columns
fig, ax = plt.subplots(nrows=1, ncols=len(cat_columns), sharey=True, figsize=(10, 5))
for i, c in enumerate(cat_columns):
    sns.countplot(x=c, data=data, ax=ax[i])
plt.show()
```



The number of observations of each category on each variable is equally distributed, except to the Company_Type where the number of service jobs its almost twice that of product ones.

One-Hot Encoding for categorical features

```
# Check if the columns exist before applying get_dummies
if all(col in data.columns for col in ['Company_Type', 'WFH_Setup_Available', 'Gender']):
    data = pd.get_dummies(data, columns=['Company_Type', 'WFH_Setup_Available','Gender'], dr
    data.head()
    encoded_columns = data.columns
else:
    print("Error: One or more of the specified columns are not present in the DataFrame.")
    # Add debugging steps here to investigate why the columns are missing.
    # For example, print the existing columns:
    print(data.columns)
```

Preprocessing

```
# Split df into X and y
y = data['Burn_Rate']
X = data.drop('Burn_Rate', axis=1)
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=True, ranc
# Scale X
scaler = StandardScaler()
scaler.fit(X_train)
X_train = pd.DataFrame(scaler.transform(X_train), index=X_train.index, columns=X_train.colum
X_test = pd.DataFrame(scaler.transform(X_test), index=X_test.index, columns=X_test.columns )
```

```
import os
import pickle
scaler_filename = '../models/scaler.pkl'
# Create the 'models' directory if it doesn't exist
os.makedirs(os.path.dirname(scaler_filename), exist_ok=True)
# Use pickle to save the scaler to the file
with open(scaler_filename, 'wb') as scaler_file:
    pickle.dump(scaler, scaler_file)
```

```
X_train
```

	Designation	Resource_Allocation	Mental_Fatigue_Score	Company_Type_Service	WFH
8977	0.723327	0.250185	-0.061773	0.724706	
14115	-0.159330	0.250185	-0.941481	0.724706	
8797	0.723327	0.250185	0.973179	0.724706	
1173	-1.041987	-1.214568	-0.579248	-1.379869	
1941	-0.159330	0.738436	1.180169	-1.379869	
...	...	...	...	...	
13453	0.723327	1.226687	1.645897	-1.379869	
21179	0.723327	0.250185	-1.044976	0.724706	
6327	0.723327	0.250185	0.093470	0.724706	
14933	-0.159330	0.250185	0.714441	0.724706	
288	-0.159330	0.250185	1.076674	-1.379869	

13013 rows × 6 columns

```
y_train
```

```
8977     0.41
14115    0.34
8797     0.61
1173     0.35
1941     0.61
         ...
```

```
13453    0.78
21179    0.30
6327     0.42
14933    0.54
288      0.57
Name: Burn_Rate, Length: 13013, dtype: float64
```

```
import os
import pickle
#saving the processed data
path = '../data/processed/'
# Create the directory if it doesn't exist
os.makedirs(path, exist_ok=True)
X_train.to_csv(path + 'X_train_processed.csv', index=False)
y_train.to_csv(path + 'y_train_processed.csv', index=False)
```

MODEL BUILDING

Linear Regression

```
#from sklearn.linear_model import LinearRegression
# Create an instance of the LinearRegression class
linear_regression_model = LinearRegression()
# Train the model
linear_regression_model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
#Linear Regressing Model Performance Metrics
print("Linear Regression Model Performance Metrics:\n")
# Make predictions on the test set
y_pred = linear_regression_model.predict(X_test)
# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
# Calculate root mean squared error
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error:", rmse)
# Calculate mean absolute error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)
# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared Score:", r2)
```

```
Linear Regression Model Performance Metrics:

Mean Squared Error: 0.0031569779113610717
Root Mean Squared Error: 0.0561869905882231
Mean Absolute Error: 0.04595032032644773
R-squared Score: 0.918822674247248
```