IMPORTING NECESSARY LIBRARIES

```
from google.colab import drive
drive.mount('/content/drive')
```

train=70%

test=30%

```
#Importing necessary liabries
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_analysis-AI.xlsx")
```

DATA OVERVIEW

```
data.head()
```

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
0	fffe32003000360033003200	2008-09-30	Female	Service	No	2	3.0	3.8	0.16
1	fffe3700360033003500	2008-11-30	Male	Service	Yes	1	2.0	5.0	0.36
2	fffe31003300320037003900	2008-03-10	Female	Product	Yes	2	NaN	5.8	0.49
3	fffe32003400380032003900	2008-11-03	Male	Service	Yes	1	1.0	2.6	0.20
4	fffe31003900340031003600	2008-07-24	Female	Service	No	3	7.0	6.9	0.52

```
data.describe() #descriptive statistics
```

	Date of Joining	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
count	22750	22750.000000	21369.000000	20633.000000	21626.000000
mean	2008-07-01 09:28:05.274725120	2.178725	4.481398	5.728188	0.452005
min	2008-01-01 00:00:00	0.000000	1.000000	0.000000	0.000000
25%	2008-04-01 00:00:00	1.000000	3.000000	4.600000	0.310000
50%	2008-07-02 00:00:00	2.000000	4.000000	5.900000	0.450000
75%	2008-09-30 00:00:00	3.000000	6.000000	7.100000	0.590000
max	2008-12-31 00:00:00	5.000000	10.000000	10.000000	1.000000
std	NaN	1.135145	2.047211	1.920839	0.198226

```
data.columns.tolist() #column names
```

```
['Employee ID',
 'Date of Joining',
 'Gender',
 'Company Type',
 'WFH Setup Available',
 'Designation',
 'Resource Allocation',
 'Mental Fatigue Score',
 'Burn Rate']
```

```
data.nunique() #number of unique values in each column
```

```
Employee ID             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.info() #information about the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22750 entries, 0 to 22749
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Employee ID           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() #checking for null values
```

```
Employee ID                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()  #total number of null values
```

```
4622
```

Exporing Data Analysis

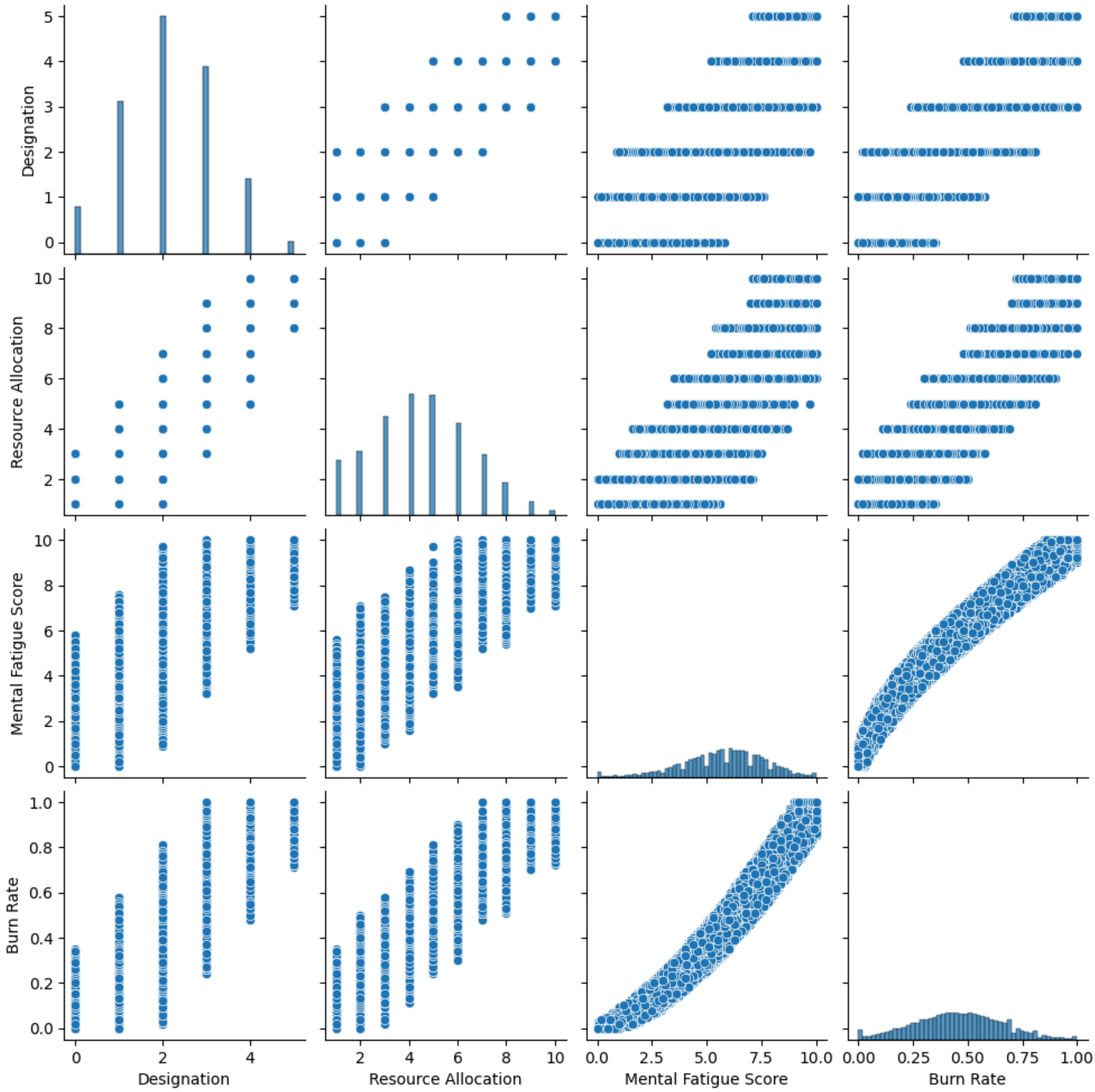
There are NaN values on our target("burn rate") and also in Resource Allocation and Mental Fatigue Score coulmns.As we are going to perform supervised linear regression ,our target variablesa is needed to do so.Therfore,this 1124 rows with NaN vlaues must be dropped off of our dataset

```
data.corr(numeric_only=True)['Burn Rate'][:-1] #checking for correlation
```

```
Designation             0.737556
Resource Allocation     0.856278
Mental Fatigue Score    0.944546
Name: Burn Rate, dtype: float64
```

These two varibles are stongly correlated with target variables ,therfore,important to estimate it.

```
sns.pairplot(data) #checking for linear relationship
plt.show()  #checking for linear relationship
```




```
data=data.dropna() #dropping rows with NaN values
```

```
data.shape  #checking the shape of the dataset after dropping rows with NaN values
```

```
(18590, 9)
```

```
data.dtypes #checking the data types of each column
```

```
Employee ID                     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 employee ID doesn't provide any useful and therfore,they must be dropped.

```
data=data.drop(['Employee ID'],axis=1) #dropping the column
```

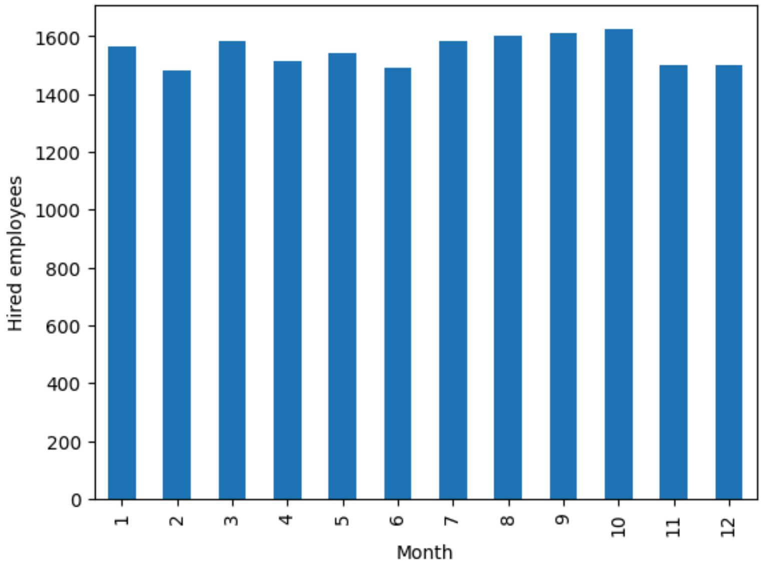
Unsupportd cell type double-click to inspect/edit the content

Checkin the correlation of Date of joining with target varibles.

```
print(f"Min date {data['Date of Joining'].min()}")  # Removed extra '
print(f"Max date {data['Date of Joining'].max()}")  # Removed extra '
data_month = data.copy()  # Create a copy of the original DataFrame

# Corrected column name, removed extra space
data_month['Date of Joining'] = data_month['Date of Joining'].astype("datetime64[ns]")  # Specify time unit as nanoseconds
data_month['Date of Joining'].groupby(data_month['Date of Joining'].dt.month).count().plot(kind="bar", xlabel="Month", ylabel="Hired emp
plt.show()
```

```
Min date 2008-01-01 00:00:00
Max date 2008-12-31 00:00:00
```



The date joining id uniform distribute with values between 2008-01-01 and 2008-12-31.so in order to create a new feature which represents labour sensority.We could create a variable with days work

```
data_2008=pd.to_datetime(["2008-01-01"]*len(data))
#specify time unit as nanoseconds when converting to datetime64
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'][:] #checking for 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
```

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

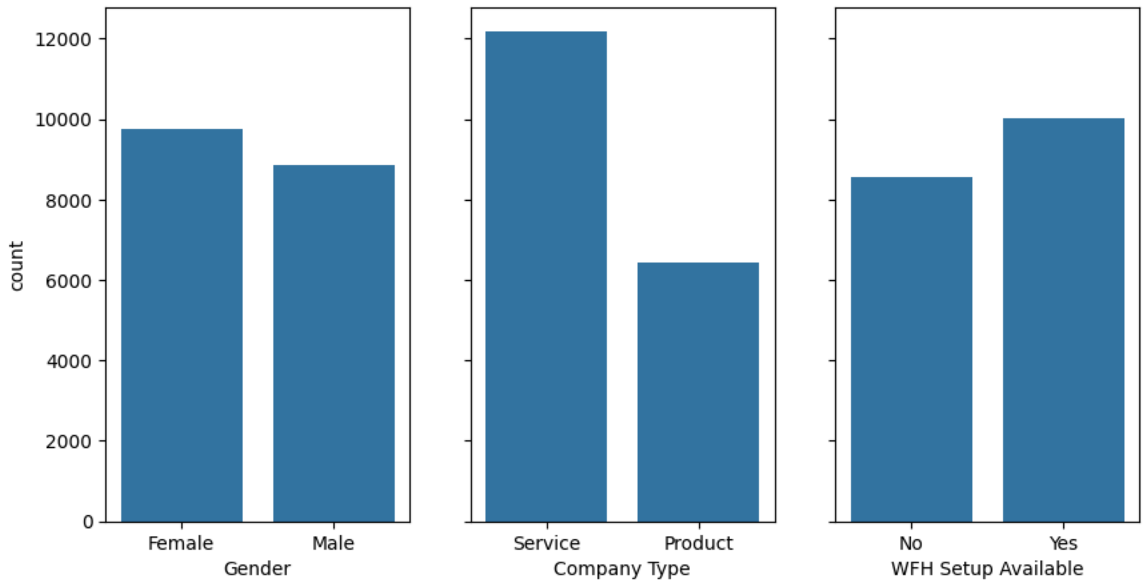
```
data=data.drop(['Date of Joining','Days'],axis=1) #dropping the column
```

```
data.head()
```

	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
0	Female	Service	No	2	3.0	3.8	0.16
1	Male	Service	Yes	1	2.0	5.0	0.36
3	Male	Service	Yes	1	1.0	2.6	0.20
4	Female	Service	No	3	7.0	6.9	0.52
5	Male	Product	Yes	2	4.0	3.6	0.29

Now analysing the categorical variables

```
cat_columns=data.select_dtypes(include=['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 observation of each category on each variable is equally distrubuted,expect to the company_Type where the number of service joba its almost twice that of product ones.

one-Hot Encoding for categorical features

```
data=pd.get_dummies(data,columns=['Company Type','WFH Setup Available','Gender'],drop_first=True) #one-Hot Encoding for categorical feat
data.head()
encoded_columns=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,test_size=0.7,shuffle=True,random_state=1) #train-test split
#scale x
scaler=StandardScaler()
scaler.fit(x_train)
x_train=pd.DataFrame(scaler.transform(x_train),index=x_train.index,columns=x_train.columns)
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 model 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 Setup Available_Yes	Gender_Male
3249	0.729794	0.733212	0.501849	0.714251	-1.104601	1.049455
7925	0.729794	0.733212	0.398570	-1.400068	0.905304	1.049455
16635	-0.155002	-0.730675	-1.925209	0.714251	-1.104601	1.049455
3456	1.614590	0.733212	1.224802	-1.400068	-1.104601	-0.952875
17562	0.729794	0.245250	-0.169465	0.714251	-1.104601	1.049455
...	...	...	...	...	...	...
13453	0.729794	1.221175	1.637919	-1.400068	0.905304	-0.952875
21179	0.729794	0.245250	-1.047337	0.714251	0.905304	1.049455
6327	0.729794	0.245250	0.088733	0.714251	-1.104601	1.049455
14933	-0.155002	0.245250	0.708407	0.714251	-1.104601	1.049455
288	-0.155002	0.245250	1.069884	-1.400068	-1.104601	-0.952875

5577 rows × 6 columns

```
y_train
```

```
3249     0.62
7925     0.50
16635    0.13
3456     0.66
17562    0.42
         ...
13453    0.78
21179    0.30
6327     0.42
14933    0.54
288      0.57
Name: Burn Rate, Length: 5577, dtype: float64
```

```
import os
import pickle
#saving the processed data
path='../models/processed_data.pkl'
#create the directory if it doesn't exist
os.makedirs(os.path.dirname(path),exist_ok=True)

x_train.to_csv(path+'x_train_prosessed.csv',index=False)
y_train.to_csv(path+'y_train_processed.csv',index=False)
```

Model building

Linear regression

```
#create an instance of the LinearRegression
linear_regression_model=LinearRegression()
#train the model
linear_regression_model.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
#linear Regression Model performance metrices
print("Linear Regression Model performance metrices:\n")
#make prediction on test set
y_pred=linear_regression_model.predict(x_test)
#calculate mean square 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 metrices:

Mean Squared Error: 0.0031265186703183815
Root Mean Squared Error: 0.05591528118786833
Mean Absolute Error: 0.045748186923994995
R-squared Score: 0.9199255887706531
```

based on the evalution metrices,the linear Regression model appears to be best model for pedicting burnout analysis.

It has the lowest mean squaes error,root mean squared error,and mean absolute error,indicating better accuracy and precision in its prediction.Additionally,it has the highest R-square score,indicating a good fir to the data and explaining a highest proportion of variance.

so we are choosing this model for deployment.