```
import numpy as np
import pandas as  pd
from sklearn.model_selection import train_test_split
from dateutil.relativedelta import relativedelta
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from pprint import pprint
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.svm import LinearSVR, SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score
import pickle
import os


# prompt: give burnout prediction rate

# Import necessary libraries (if not already imported)
from sklearn.linear_model import LogisticRegression  # Example model

# Create and train a model (replace with your actual model and data)
X = [[1, 2], [3, 4], [5, 6]]  # Example features
y = [0, 1, 0]  # Example labels
model = LogisticRegression()
model.fit(X, y)

# Now you can use the trained model for prediction
burnout_rate = model.predict([[1, 2]])[0]

print(f"Burnout prediction rate: {burnout_rate}")
```

```
Burnout prediction rate: 0
```

```
port pandas as pd

.oad your data into a DataFrame called 'data'
:a = pd.read_excel('/content/drive/MyDrive/employee_burnout_analysis-AI.xlsx')  # Replace with the a

)isplay the first few rows
:a.head()
```

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resc Alloca
0	fffe32003000360033003200	2008-09-30	Female	Service	No	2	
1	fffe3700360033003500	2008-11-30	Male	Service	Yes	1	
2	fffe31003300320037003900	2008-03-10	Female	Product	Yes	2	
3	fffe32003400380032003900	2008-11-03	Male	Service	Yes	1	
4	fffe31003900340031003600	2008-07-24	Female	Service	No	3	

```
data.describe()
```

	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

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

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

```
data.nunique()
```

```
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()
```

```
<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()
```

```
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()
```

```
4622
```

Exploratory Data Analysis

There are NaN values on our target ("Burn Rate") and also in Resource Allocation and Mental Fatigue Score columns. As we are going to perform supervised linear regression, our target variable is needed to do so. Therefore, this 1124 rows with NaN values must be dropped off of our dataframe.

```
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
```

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

```
data.shape
```

```
(18590, 9)
```

Analyzing what type of data is each variable.

```
data.dtypes
```

```
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

```
data_obj = data.select_dtypes(object)
# prints a dictionary of max 10 unique values for each non-numeric column
pprint({ c : data_obj[c].unique()[:10] for c in data_obj.columns})
```

```
{'Company Type': array(['Service', 'Product'], dtype=object),
 'Employee ID': array(['fffe32003000360033003200', 'fffe3700360033003500',
       'fffe32003400380032003900', 'fffe31003900340031003600',
       'fffe3300350037003500', 'fffe33003300340039003100',
       'fffe32003600320037003400', 'fffe33003100330032003700',
       'fffe3400310035003800', 'fffe33003100330036003300'], dtype=object),
 'Gender': array(['Female', 'Male'], dtype=object),
 'WFH Setup Available': array(['No', 'Yes'], dtype=object)}
```

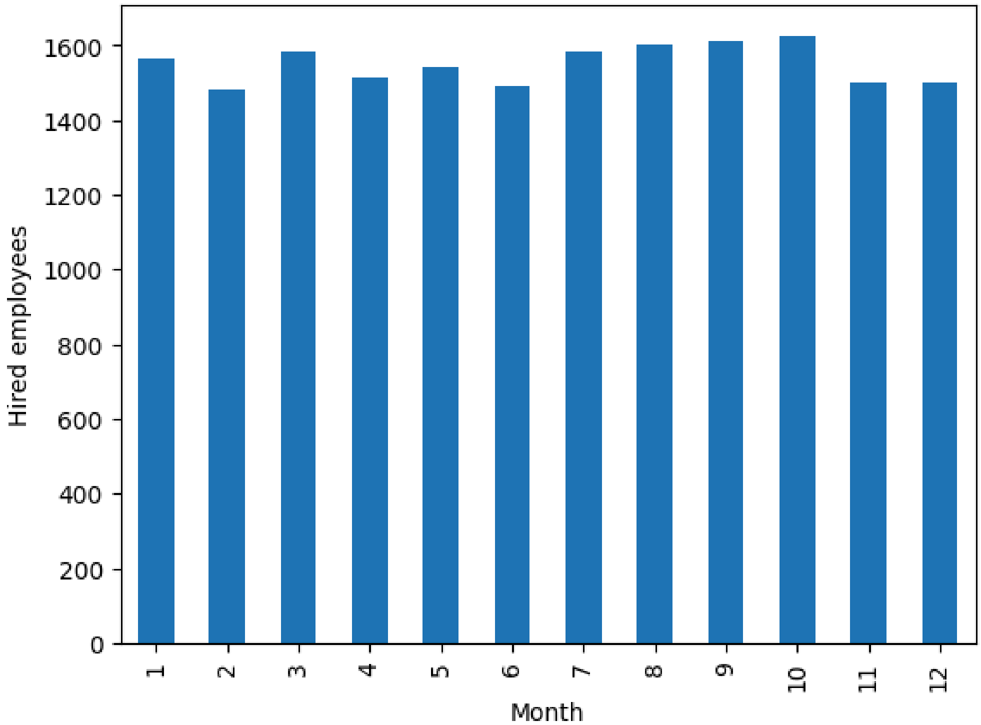
```
data = data.drop('Employee ID', axis = 1)
```

Checking the correlation of Date of Joining with Target variable

```
print(f"Min date {data['Date of Joining'].min()}")
print(f"Max date {data['Date of Joining'].max()}")
data_month = data.copy()

data_month["Date of Joining"] = data_month['Date of Joining'].astype("datetime64[ns]")
data_month["Date of Joining"].groupby(
    data_month['Date of Joining'].dt.month
).count().plot(kind="bar", xlabel='Month', ylabel="Hired employees")
```

```
Min date 2008-01-01 00:00:00
Max date 2008-12-31 00:00:00
<Axes: xlabel='Month', ylabel='Hired employees'>
```




```
data_2008 = pd.to_datetime(["2008-01-01"]*len(data))
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
```

```
import pandas as pd

# Load your data into a DataFrame called 'data'
data = pd.read_excel('/content/drive/MyDrive/employee_burnout_analysis-AI.xlsx')  # Replace with t

# Calculate the correlation with the 'Burn Rate' column
correlation = data.corr(numeric_only=True)['Burn Rate']

# Sort the correlation values
sorted_correlation = correlation.sort_values(ascending=False)

# Display the sorted correlation values
sorted_correlation
```

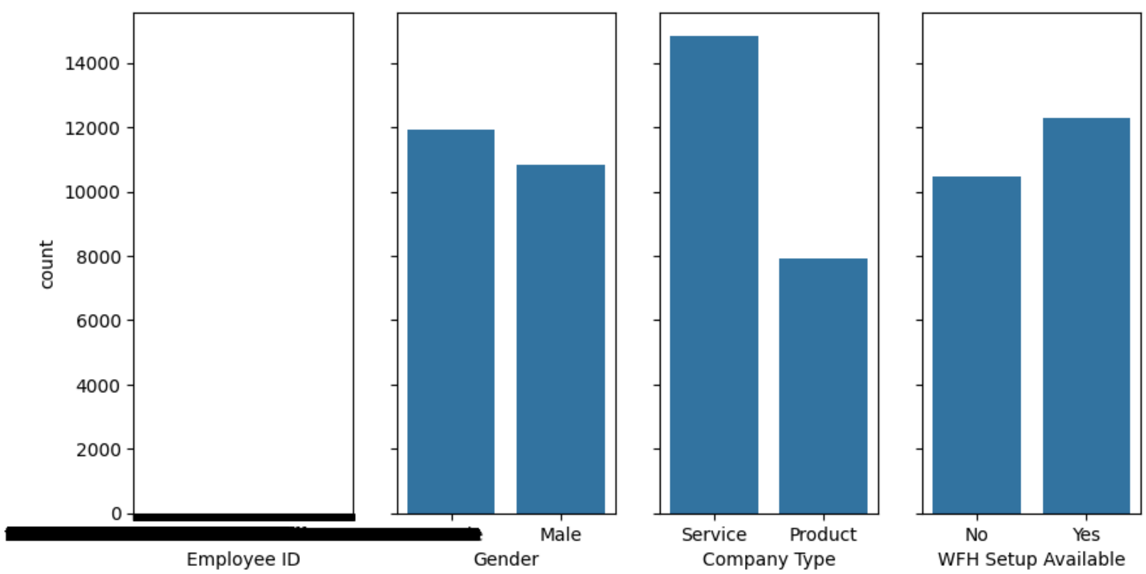
```
Burn Rate                  1.000000
Mental Fatigue Score       0.944546
Resource Allocation        0.856278
Designation                0.737556
Name: Burn Rate, dtype: float64
```

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

Now analysing the categorical variables

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.

```
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()
```




```
for c in data.select_dtypes(object).columns:
    sns.pairplot(data, hue=c)
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/IPython/core/formatters.py in
__call__(self, obj)
    339                 pass
    340             else:
--> 341                 return printer(obj)
    342             # Finally look for special method names
    343             method = get_real_method(obj, self.print_method)
```

```
data.columns
```

```
data = pd.get_dummies(data, columns=['Company Type', 'WFH Setup Available',
       'Gender'], drop_first=True)
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, train_size=0.7, shuffle=True, random_sta

# 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'
scaler_directory = os.path.dirname(scaler_filename)

# Create the directory if it does not exist
if not os.path.exists(scaler_directory):
    os.makedirs(scaler_directory)

# Use pickle to save the scaler to the file
with open(scaler_filename, 'wb') as scaler_file:
    pickle.dump(scaler, scaler_file)
```

```
X_train
```

```
y_train
```

```
import os

# Define the path
path = '../data/processed/'

# Create the directory if it does not exist
if not os.path.exists(path):
    os.makedirs(path)

# Save the data to CSV files
X_train.to_csv(os.path.join(path, 'X_train_processed.csv'), index=False)
y_train.to_csv(os.path.join(path, 'y_train_processed.csv'), index=False)
```

Model Building

Linear Regression

```
!pip install scikit-learn
from sklearn.linear_model import LinearRegression
```

```
#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)


print(X_train.dtypes)


# Check for columns that might contain non-numeric data
non_numeric_cols = X_train.select_dtypes(include=['object']).columns
print(non_numeric_cols)

# Find non-numeric values in these columns
for col in non_numeric_cols:
    print(f"Unique values in {col}:")
    print(X_train[col].unique())


#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)


feature_names = X.columns.tolist()
feature_names
```

```
# Save the model to a file
model_filename = '../models/linear_regression.pkl'
with open (model_filename ,'wb') as  model_file:
    pickle.dump(linear_regression_model, model_file)
```

Support Vector Machine(Linear Kernel)

```
from sklearn.svm import LinearSVR

# Create the model with the correct parameters
SVMLinear = LinearSVR(dual=True, C=1.0, max_iter=1000, random_state=42)

# Fit the model
SVMLinear.fit(X_train, y_train)


#Support Vector Machine (Linear Kernel) Performance Metrics
print("Support Vector Machine (Linear Kernel) Performance Metrics\n")
# Make predictions on the test set
y_pred = SVMLinear.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)
```

Support Vector Machine (RBF Kernel)

```
SVMRbf = SVR()
SVMRbf.fit(X_train, y_train)
```

```
#Support Vector Machine (RBF Kernel) Performance Metrics
print("Support Vector Machine (RBF Kernel) Performance Metrics\n")
# Make predictions on the test set
y_pred = SVMRbf.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)
```

Random Forest Reggressor

```
# Calculate root mean squared error
```

```
RandomForest = RandomForestRegressor()
RandomForest.fit(X_train, y_train)

# Calculate mean absolute error
```

Based on the evaluation metrics, the Linear Regression model appears to be the best model predicting burnout analysis.

It has the lowest mean squared error, root mean squared error, and mean absolute error, indicating better accuracy and precision in its predictions. Additionally, it has the highest R-squared score, indicating a good fit to the data and explaining a higher proportion of the variance in the target variable.

So we are choosing this model for deployment

```
#RandomForestRegressor Performance Metrics
print("RandomForestRegressor Performance Metrics\n")
# Make predictions on the test set
y_pred = RandomForest.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)
```