Importing Libraries

```
import warnings
warnings.filterwarnings('ignore')
```

```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

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

```
Mounted at /content/drive
```

Taking The Dataset

```
pd.set_option('display.max_columns', None)
burnoutDf=pd.read_csv('/content/employee_burnout_analysis.csv')
burnoutDf
```

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation
0	fffe32003000360033003200	30-09-2008	Female	Service	No	2
1	fffe3700360033003500	30-11-2008	Male	Service	Yes	1
2	fffe31003300320037003900	10-03-2008	Female	Product	Yes	2
3	fffe32003400380032003900	03-11-2008	Male	Service	Yes	1
4	fffe31003900340031003600	24-07-2008	Female	Service	No	3
...	...	...	...	...	...	...
22745	fffe31003500370039003100	30-12-2008	Female	Service	No	1

```
# Convert into datetime ddataType
burnoutDf["Date of Joiniing"]= pd.to_datetime(burnoutDf["Date of Joining"])
```

```
#give the number of rows and columns
burnoutDf.shape
```

```
(22750, 10)
```

```
# general iinformation
burnoutDf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22750 entries, 0 to 22749
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Employee ID           22750 non-null  object
 1   Date of Joining       22750 non-null  object
 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
 9   Date of Joiniing      22750 non-null  datetime64[ns]
```

```
dtypes: datetime64[ns](1), float64(3), int64(1), object(5)
memory usage: 1.7+ MB
```

```
# show top 5 rows
burnoutDf.head()
```

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Re: Allo
0	fffe32003000360033003200	30-09-2008	Female	Service	No	2	
1	fffe3700360033003500	30-11-2008	Male	Service	Yes	1	

```
# extract all columns of the dataset
burnoutDf.columns
```

```
Index(['Employee ID', 'Date of Joining', 'Gender', 'Company Type',
       'WFH Setup Available', 'Designation', 'Resource Allocation',
       'Mental Fatigue Score', 'Burn Rate', 'Date of Joiniing'],
      dtype='object')
```

```
# check for null values
burnoutDf.isna().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
Date of Joiniing           0
dtype: int64
```

```
# check the duplicate values
burnoutDf.duplicated().sum()
```

```
0
```

```
# calculate the mean , std , min , max and count of every attrubutes
burnoutDf.describe()
```

	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
count	22750.000000	21369.000000	20633.000000	21626.000000
mean	2.178725	4.481398	5.728188	0.452005
std	1.135145	2.047211	1.920839	0.198226
min	0.000000	1.000000	0.000000	0.000000
25%	1.000000	3.000000	4.600000	0.310000
50%	2.000000	4.000000	5.900000	0.450000
75%	3.000000	6.000000	7.100000	0.590000
max	5.000000	10.000000	10.000000	1.000000

```
# show the unique values
for i, col in enumerate(burnoutDf.columns):
  print(f"\n\n\{burnoutDf[col].unique()}")
  print(f"\n{burnoutDf[col].value_counts()}\n\n")
```

```
 '2008-05-11T00:00:00.000000000' '2008-08-19T00:00:00.000000000'
 '2008-04-17T00:00:00.000000000' '2008-07-08T00:00:00.000000000'
 '2008-12-31T00:00:00.000000000' '2008-05-27T00:00:00.000000000'
 '2008-09-29T00:00:00.000000000' '2008-05-30T00:00:00.000000000'
 '2008-12-18T00:00:00.000000000' '2008-02-20T00:00:00.000000000'
 '2008-11-12T00:00:00.000000000' '2008-11-27T00:00:00.000000000'
 '2008-07-20T00:00:00.000000000' '2008-11-28T00:00:00.000000000'
 '2008-03-08T00:00:00.000000000' '2008-10-20T00:00:00.000000000'
 '2008-07-07T00:00:00.000000000' '2008-08-06T00:00:00.000000000'
 '2008-03-24T00:00:00.000000000' '2008-12-21T00:00:00.000000000'
 '2008-09-04T00:00:00.000000000' '2008-05-05T00:00:00.000000000'
 '2008-12-06T00:00:00.000000000' '2008-04-18T00:00:00.000000000'
 '2008-01-27T00:00:00.000000000' '2008-10-17T00:00:00.000000000'
 '2008-09-05T00:00:00.000000000' '2008-03-29T00:00:00.000000000'
 '2008-12-09T00:00:00.000000000' '2008-07-25T00:00:00.000000000'
 '2008-07-04T00:00:00.000000000' '2008-02-05T00:00:00.000000000'
 '2008-02-06T00:00:00.000000000' '2008-02-10T00:00:00.000000000'
 '2008-02-26T00:00:00.000000000' '2008-12-07T00:00:00.000000000'
 '2008-06-02T00:00:00.000000000' '2008-06-23T00:00:00.000000000'
 '2008-06-11T00:00:00.000000000' '2008-07-16T00:00:00.000000000'
 '2008-06-25T00:00:00.000000000' '2008-01-29T00:00:00.000000000'
 '2008-02-29T00:00:00.000000000' '2008-03-25T00:00:00.000000000'
 '2008-08-18T00:00:00.000000000' '2008-05-04T00:00:00.000000000'
 '2008-05-15T00:00:00.000000000' '2008-12-12T00:00:00.000000000'
 '2008-10-25T00:00:00.000000000' '2008-06-04T00:00:00.000000000'
 '2008-11-13T00:00:00.000000000' '2008-04-09T00:00:00.000000000'
 '2008-05-24T00:00:00.000000000' '2008-10-06T00:00:00.000000000'
 '2008-03-31T00:00:00.000000000' '2008-01-12T00:00:00.000000000'
 '2008-05-01T00:00:00.000000000' '2008-09-15T00:00:00.000000000'
 '2008-10-12T00:00:00.000000000' '2008-10-02T00:00:00.000000000'
 '2008-03-12T00:00:00.000000000' '2008-01-02T00:00:00.000000000']

2008-06-01    86
2008-05-21    85
2008-04-02    82
2008-07-16    81
2008-07-13    80
              ..
2008-06-27    44
2008-06-07    44
2008-04-07    43
2008-12-24    43
2008-07-12    39
Name: Date of Joiniing, Length: 366, dtype: int64
```

```
# drop irrelevant column
burnoutDf=burnoutDf.drop(['Employee ID'],axis=1)
```

```
# check the skewness of the attributes
intFloatburnoutDf=burnoutDf.select_dtypes([np.int, np.float])
for i, col in enumerate(intFloatburnoutDf.columns):
  if (intFloatburnoutDf[col].skew() >= 0.1):
    print("\n",col, "feature is Positively skewed and value is; ", intFloatburnoutDf[col].skew())
  elif (intFloatburnoutDf[col].skew() <= -0.1):
    print("\n",col, "feature is Negatively skewed and value is; ", intFloatburnoutDf[col].skew())
  else:
    print("\n",col, "feature is Normally Distributed and value is; ", intFloatburnoutDf[col].skew())
```

```
 Designation feature is Normally Distributed and value is;  0.09242138478903683

 Resource Allocation feature is Positively skewed and value is;  0.20457273454318103

 Mental Fatigue Score feature is Negatively skewed and value is;  -0.4308950578815428

 Burn Rate feature is Normally Distributed and value is;  0.045737370909640515
```

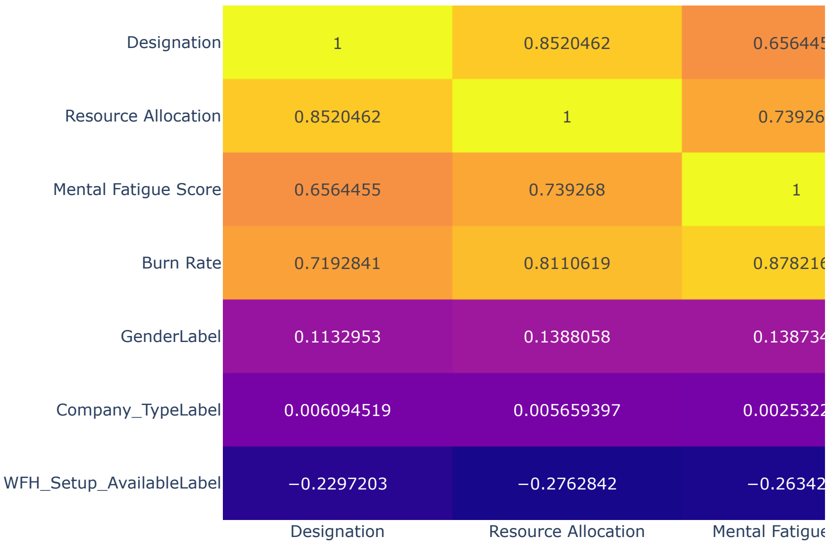
```
# Replace the null values with mean
burnoutDf['Resource Allocation'].fillna(burnoutDf['Resource Allocation'].mean(),inplace=True)
burnoutDf['Mental Fatigue Score'].fillna(burnoutDf['Mental Fatigue Score'].mean(),inplace=True)
burnoutDf['Burn Rate'].fillna(burnoutDf['Burn Rate'].mean(),inplace=True)
```

```
# show the correlation
burnoutDf.corr()
```

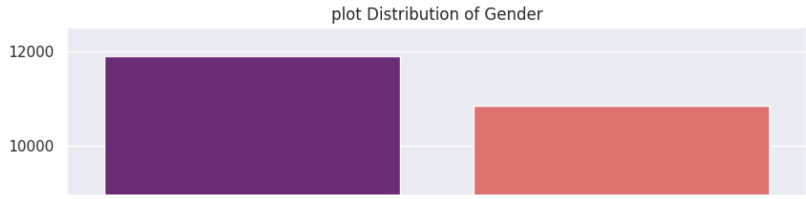
	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate	GenderLabe
Designation	1.000000	0.852046	0.656445	0.719284	0.11329

Data Visualization

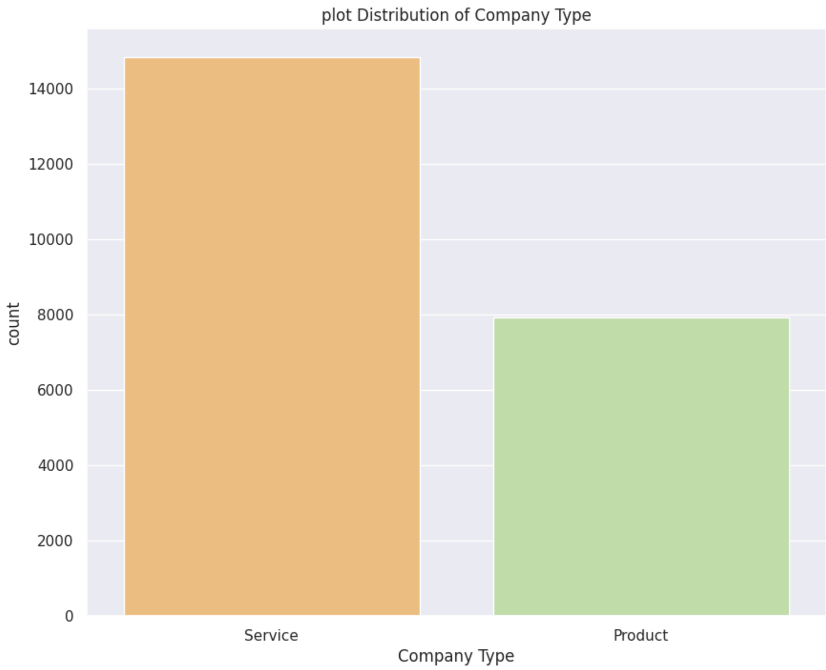
```
# plotting Heat map to check Correlation
Corr=burnoutDf.corr()
sns.set(rc={'figure.figsize':(14,12)})
fig = px.imshow(Corr, text_auto=True, aspect="auto")
fig.show()
```



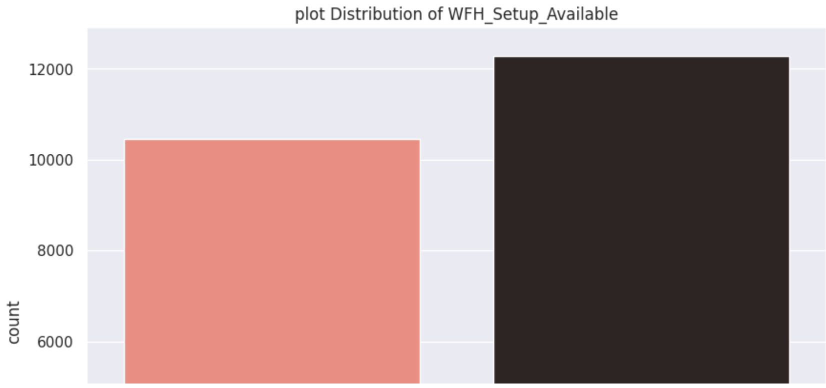

```
# Count plot distribution of "Gender"
plt.figure(figsize=(10,8))
sns.countplot(x="Gender", data=burnoutDf, palette="magma")
plt.title("plot Distribution of Gender")
plt.show()
```



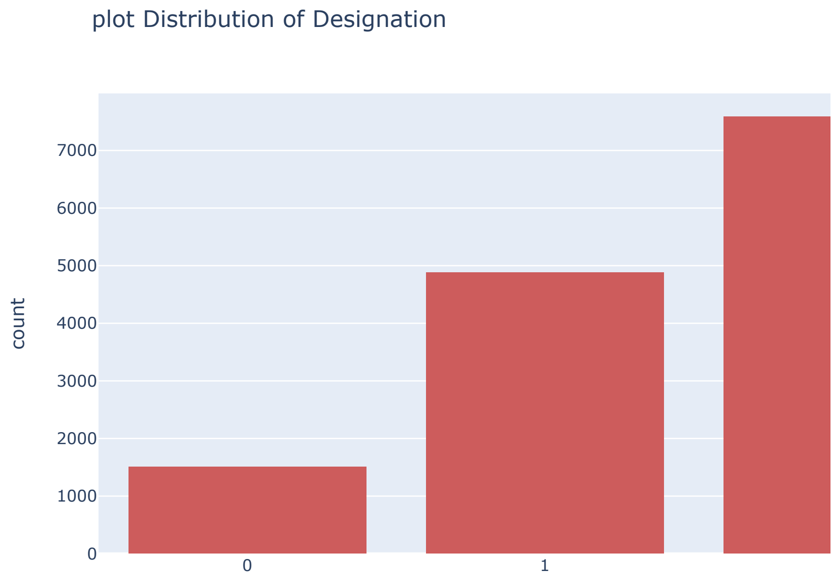

```
# Count plot distribution of "Company Type"
plt.figure(figsize=(10,8))
sns.countplot(x="Company Type", data=burnoutDf, palette="Spectral")
plt.title("plot Distribution of Company Type")
plt.show()
```

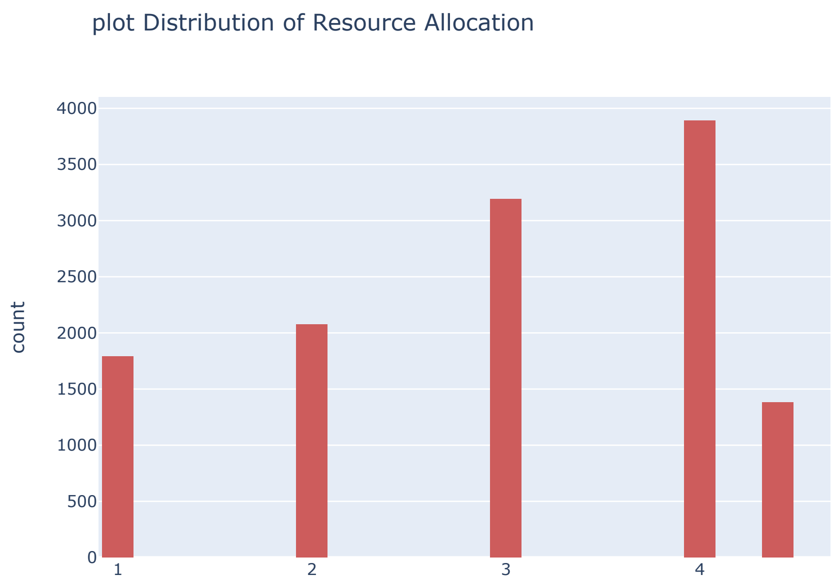



```
# Count plot distribution of "Company Type"
plt.figure(figsize=(10,8))
sns.countplot(x="WFH Setup Available", data=burnoutDf, palette="dark:salmon_r")
plt.title("plot Distribution of WFH_Setup_Available")
plt.show()
```




```
# count-plot Distribution of attributes with the help of Histogram
burn_st=burnoutDf.loc[:,'Date of Joining':'Burn Rate']
burn_st=burn_st.select_dtypes([int, float])
for i, col in enumerate(burn_st.columns):
  fig = px.histogram(burn_st, x=col, title="plot Distribution of " +col, color_discrete_sequence=['indianred'])
  fig.update_layout(bargap=0.2)
  fig.show()
```



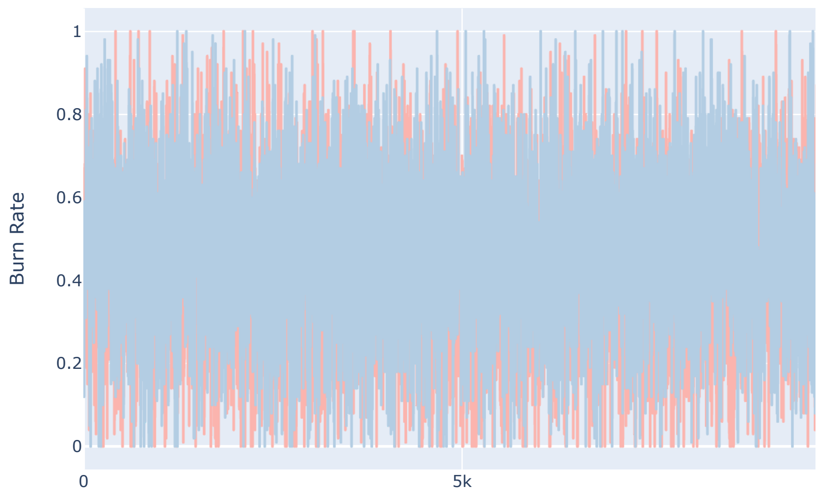



```
# plot distribution of burn rate on the basis of Designation
fig = px.line(burnoutDf, y="Burn Rate", color="Designation", title="Burn rate on the basis of Designation", color_discrete_sequence=px.c
fig.update_layout(bargap=0.1)
fig.show()
```

Burn rate on the basis of Designation

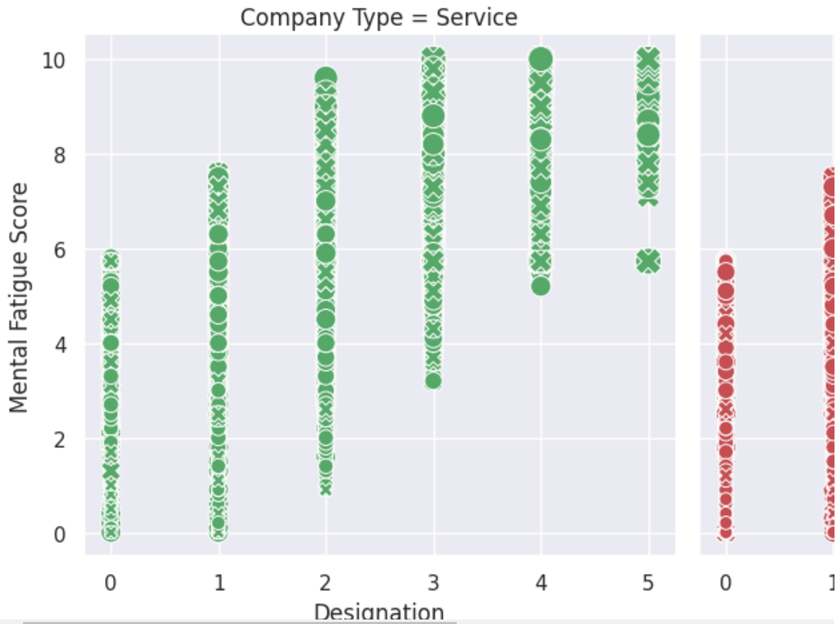
```
# plot distribution of burn rate on the basis of Gender
fig = px.line(burnoutDf, y="Burn Rate", color="Gender", title="Burn rate on the basis of Gender", color_discrete_sequence=px.colors.qual
fig.update_layout(bargap=0.2)
fig.show()
```

Burn rate on the basis of Gender




```
# plot Distribution of " Designation vs mental fatigue" as per company type , Burn rate and Gender
sns.relplot(
    data=burnoutDf, x="Designation", y="Mental Fatigue Score", col="Company Type",
    hue="Company Type", size="Burn Rate", style="Gender",
    palette=["g", "r"], sizes=(50, 200)
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f6141c85690>
```



Label Encoding

```
# label encoding and assign in new variable
from sklearn import preprocessing
Label_encode = preprocessing.LabelEncoder()


# Assign in new variable
burnoutDf['GenderLabel'] = Label_encode.fit_transform(burnoutDf['Gender'].values)
burnoutDf['Company_TypeLabel'] = Label_encode.fit_transform(burnoutDf['Company Type'].values)
burnoutDf['WFH_Setup_AvailableLabel'] = Label_encode.fit_transform(burnoutDf['WFH Setup Available'].values)
```

```
# Check assigned values

gn = burnoutDf.groupby('Gender')
gn = gn['GenderLabel']
gn.first()
```

```
Gender
Female    0
Male      1
Name: GenderLabel, dtype: int64
```

```
# Check assigned values
ct = burnoutDf.groupby('Company Type')
ct = ct['Company_TypeLabel']
ct.first()
```

```
Company Type
Product    0
Service    1
Name: Company_TypeLabel, dtype: int64
```

```
# check assigned values
wsa = burnoutDf.groupby('WFH Setup Available')
wsa = wsa['WFH_Setup_AvailableLabel']
wsa.first()
```

```
WFH Setup Available
No     0
Yes    1
Name: WFH_Setup_AvailableLabel, dtype: int64
```

```
# show last 10 rows
burnoutDf.tail(22)
```

	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation	Mental Fatigue Score	
22728	26-08-2008	Male	Product	No	2	6.0	6.000000	0.52

Feature Selection

```
# Feature Selection
Columns=['Designation','Resource Allocation','Mental Fatigue Score','GenderLabel', 'Company_TypeLabel', 'WFH_Setup_AvailableLabel']
x=burnoutDf[Columns]
y=burnoutDf['Burn Rate']
```

```
print(x)
```

```
       Designation  Resource Allocation  Mental Fatigue Score  GenderLabel  \
0                2             3.000000              3.800000            0
1                1             2.000000              5.000000            1
2                2             4.481398              5.800000            0
3                1             1.000000              2.600000            1
4                3             7.000000              6.900000            0
...            ...                  ...                   ...          ...
22745            1             3.000000              5.728188            0
22746            3             6.000000              6.700000            0
22747            3             7.000000              5.728188            1
22748            2             5.000000              5.900000            0
22749            3             6.000000              7.800000            1

       Company_TypeLabel  WFH_Setup_AvailableLabel
0                      1                         0
1                      1                         1
2                      0                         1
3                      1                         1
4                      1                         0
...                  ...                       ...
22745                  1                         0
22746                  0                         1
22747                  1                         1
22748                  1                         0
22749                  0                         0

[22750 rows x 6 columns]
```

```
print(y)
```

```
0        0.16
1        0.36
2        0.49
3        0.20
4        0.52
         ...
22745    0.41
22746    0.59
22747    0.72
22748    0.52
22749    0.61
Name: Burn Rate, Length: 22750, dtype: float64
```

Implementing PCA

```
# Principle component Analysis
from sklearn.decomposition import PCA

pca = PCA(0.95)
x_pca = pca.fit_transform(x)

print("PCA shape of X is :",x_pca.shape, "and original shape is :", x.shape)
print("% of importance of selected features is:", pca.explained_variance_ratio_)
print("the number of features selected through PCA is:", pca.n_components_)
```

```
PCA shape of X is : (22750, 4) and original shape is : (22750, 6)
% of importance of selected features is: [0.78371089 0.11113597 0.03044541 0.02632422]
the number of features selected through PCA is: 4
```

Data Splitting

```
from sklearn.model_selection import train_test_split
x_train_pca, X_test, Y_train, Y_test = train_test_split(x_pca,y, test_size = 0.25, random_state=10)
```

```
# print the shape of splitted data

print(x_train_pca.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(17062, 4) (5688, 4) (17062,) (5688,)
```

Model Implementation

Random Forst Regressor

```
from sklearn.metrics import r2_score
```

```
# Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor()
rf_model.fit(x_train_pca, Y_train)

train_pred_rf = rf_model.predict(x_train_pca)
train_r2 = r2_score(Y_train, train_pred_rf)
test_pred_rf = rf_model.predict(X_test)
test_r2 = r2_score(Y_test, test_pred_rf)

print("Accuracy score of train data: "+str(round(100*train_r2, 4))+"%")
print("Accuracy score of test data: "+str(round(100*train_r2, 4))+"%")
```

```
Accuracy score of train data: 91.1887%
Accuracy score of test data: 91.1887%
```

✓ 6s completed at 10:53