```
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
```

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

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Re Allo
0	fffe32003000360033003200	9/30/2008	Female	Service	No	2	
1	fffe3700360033003500	11/30/2008	Male	Service	Yes	1	
2	fffe31003300320037003900	3/10/2008	Female	Product	Yes	2	
3	fffe32003400380032003900	11/3/2008	Male	Service	Yes	1	
4	fffe31003900340031003600	7/24/2008	Female	Service	No	3	
...	...	...	...	...	...	...	
22745	fffe31003500370039003100	12/30/2008	Female	Service	No	1	
22746	fffe33003000350031003800	1/19/2008	Female	Product	Yes	3	
22747	fffe390032003000	11/5/2008	Male	Service	Yes	3	
22748	fffe33003300320036003900	1/10/2008	Female	Service	No	2	
22749	fffe3400350031003800	1/6/2008	Male	Product	No	3	

22750 rows × 9 columns

```
burnoutDf["Date of Joining"]=pd.to_datetime(burnoutDf["Date of Joining"])
```

```
burnoutDf.shape
```

```
(22750, 9)
```

```
burnoutDf.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
```

```
burnoutDf.head()
```

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation
0	fffe32003000360033003200	2008-09-30	Female	Service	No	2	3.0
1	fffe3700360033003500	2008-11-30	Male	Service	Yes	1	2.0
2	fffe31003300320037003900	2008-03-10	Female	Product	Yes	2	NaN
3	fffe32003400380032003900	2008-11-03	Male	Service	Yes	1	1.0
4	fffe31003900340031003600	2008-07-24	Female	Service	No	3	7.0

```
burnoutDf.columns
```

```
Index(['Employee ID', 'Date of Joining', 'Gender', 'Company Type',
       'WFH Setup Available', 'Designation', 'Resource Allocation',
```

```
      'Mental Fatigue Score', 'Burn Rate'],
     dtype='object')
```

```
burnoutDf.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
```

```
burnoutDf.duplicated().sum()
```

```
0
```

```
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

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

```
Female    11908
Male      10842
Name: Gender, dtype: int64




['Service' 'Product']

Service    14833
Product     7917
Name: Company Type, dtype: int64




['No' 'Yes']

Yes    12290
No     10460
Name: WFH Setup Available, dtype: int64




[2 1 3 0 4 5]

2    7588
3    5985
1    4881
4    2391
0    1507
5     398
Name: Designation, dtype: int64




[ 3.  2. nan  1.  7.  4.  6.  5.  8. 10.  9.]

4.0     3893
5.0     3861
3.0     3192
6.0     2943
2.0     2075
7.0     1965
1.0     1791
8.0     1044
9.0      446
10.0     159
```

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

```
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 Negtively skewed and value is:",intFloatburnoutDf[col].skew()
  else:
    print("\n",col,"feature is Normally Distributed and value is:",intFloatburnoutDf[col].ske
```

```
 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 Negtively skewed and value is: -0.4308950578815428

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

```
burnoutDf['Resource Allocation'].fillna(burnoutDf['Resource Allocation'].mean(),inplace=True)
burnoutDf['Mental Fatigue Score'].fillna(burnoutDf['Mental Fatigue Score'].mean(),inplace=Tru
burnoutDf['Burn Rate'].fillna(burnoutDf['Burn Rate'].mean(),inplace=True)
```

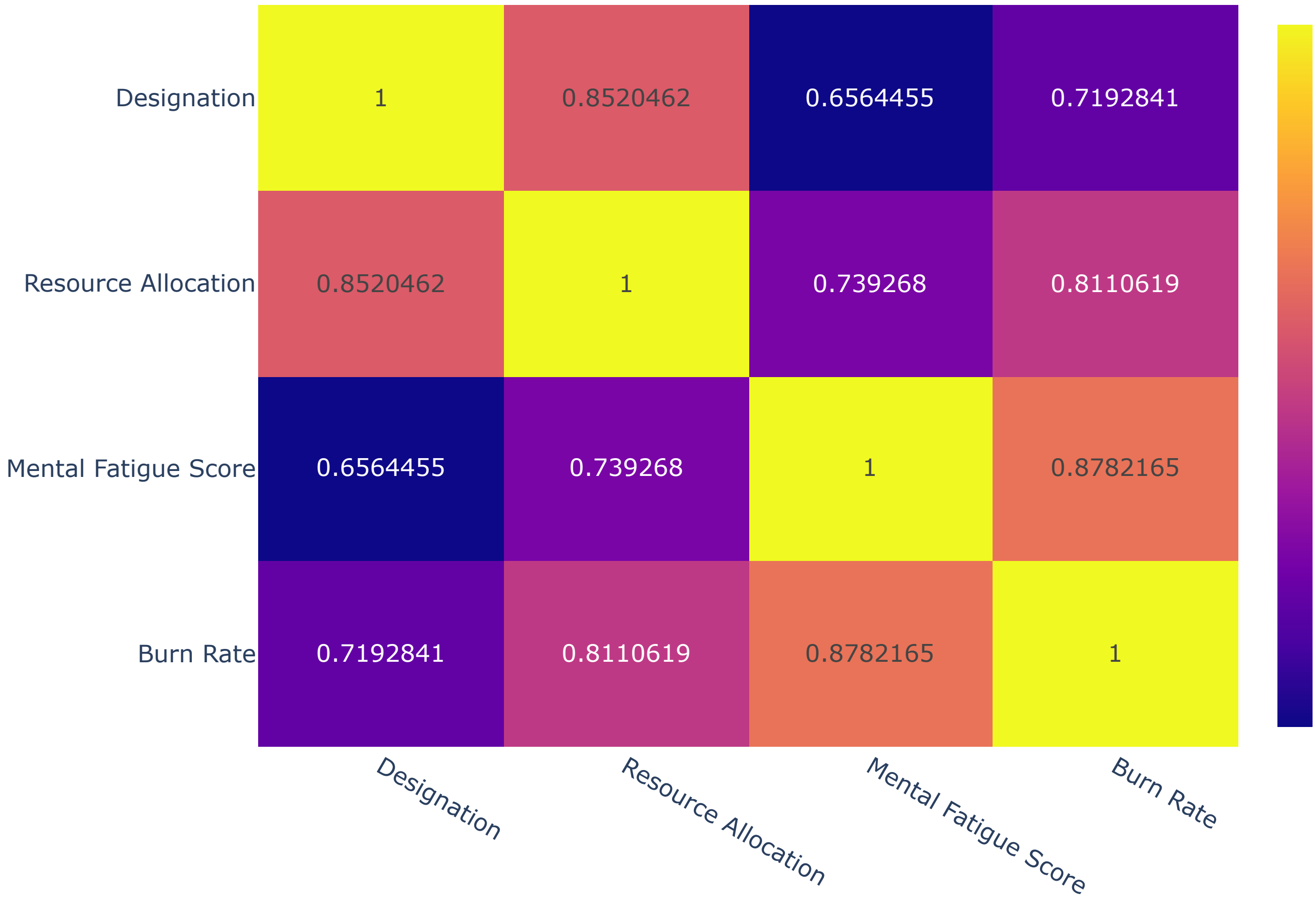
```
burnoutDf.isna().sum()
```

```
Date of Joining         0
Gender                  0
Company Type            0
WFH Setup Available     0
Designation             0
Resource Allocation     0
Mental Fatigue Score    0
Burn Rate               0
dtype: int64
```

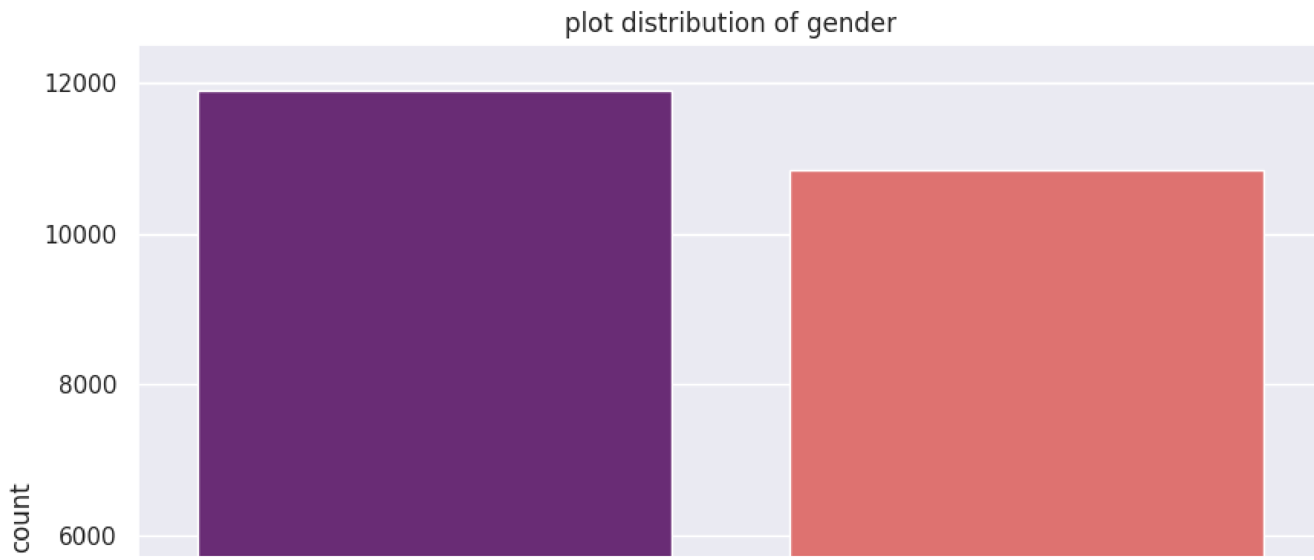
```
burnoutDf.corr()
```

	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
Designation	1.000000	0.852046	0.656445	0.719284
Resource Allocation	0.852046	1.000000	0.739268	0.811062
Mental Fatigue Score	0.656445	0.739268	1.000000	0.878217
Burn Rate	0.719284	0.811062	0.878217	1.000000

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



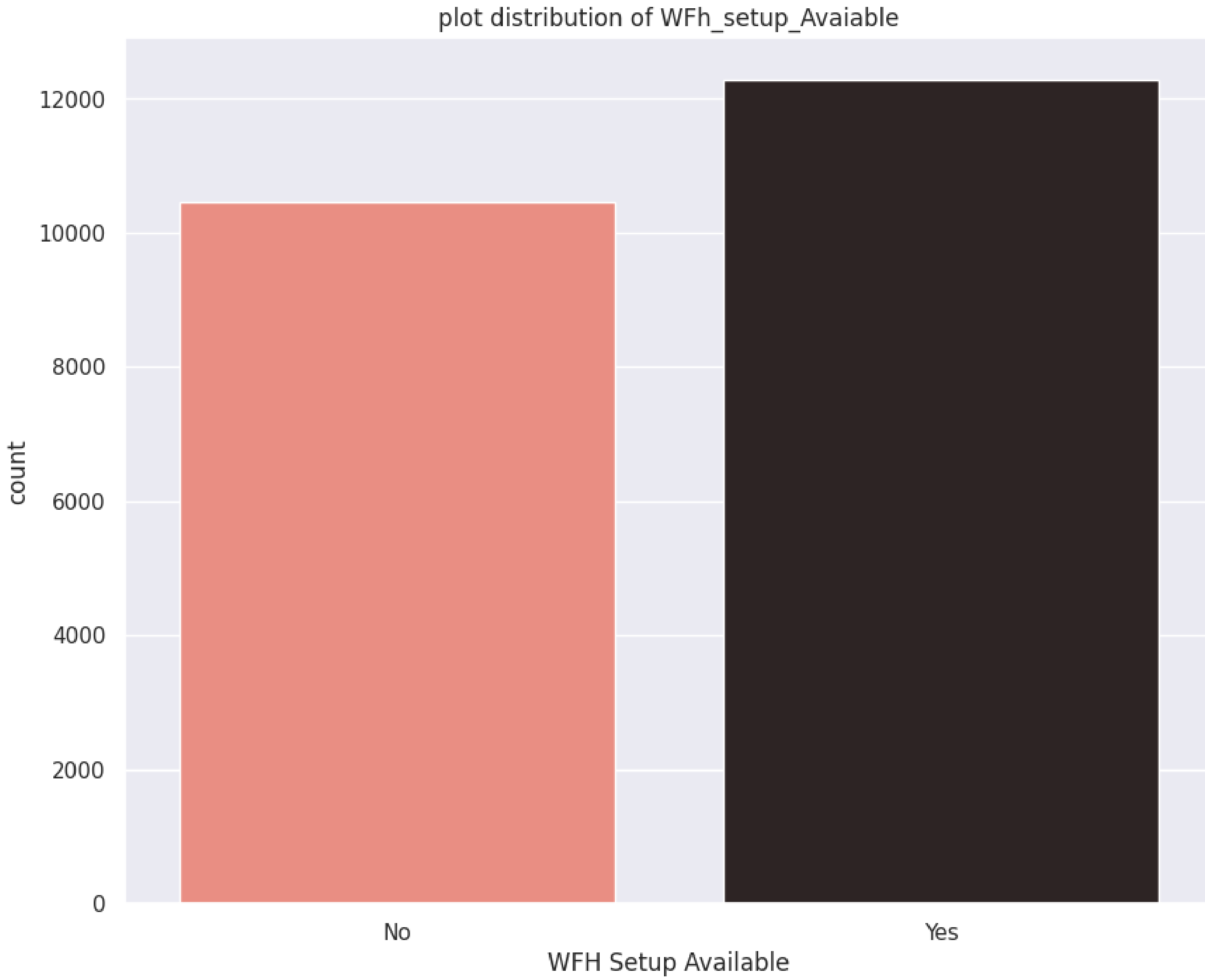

```
plt.figure(figsize=(10,8))
sns.countplot(x='Gender',data=burnoutDf,palette="magma")
plt.title("plot distribution of gender")
plt.show()
```



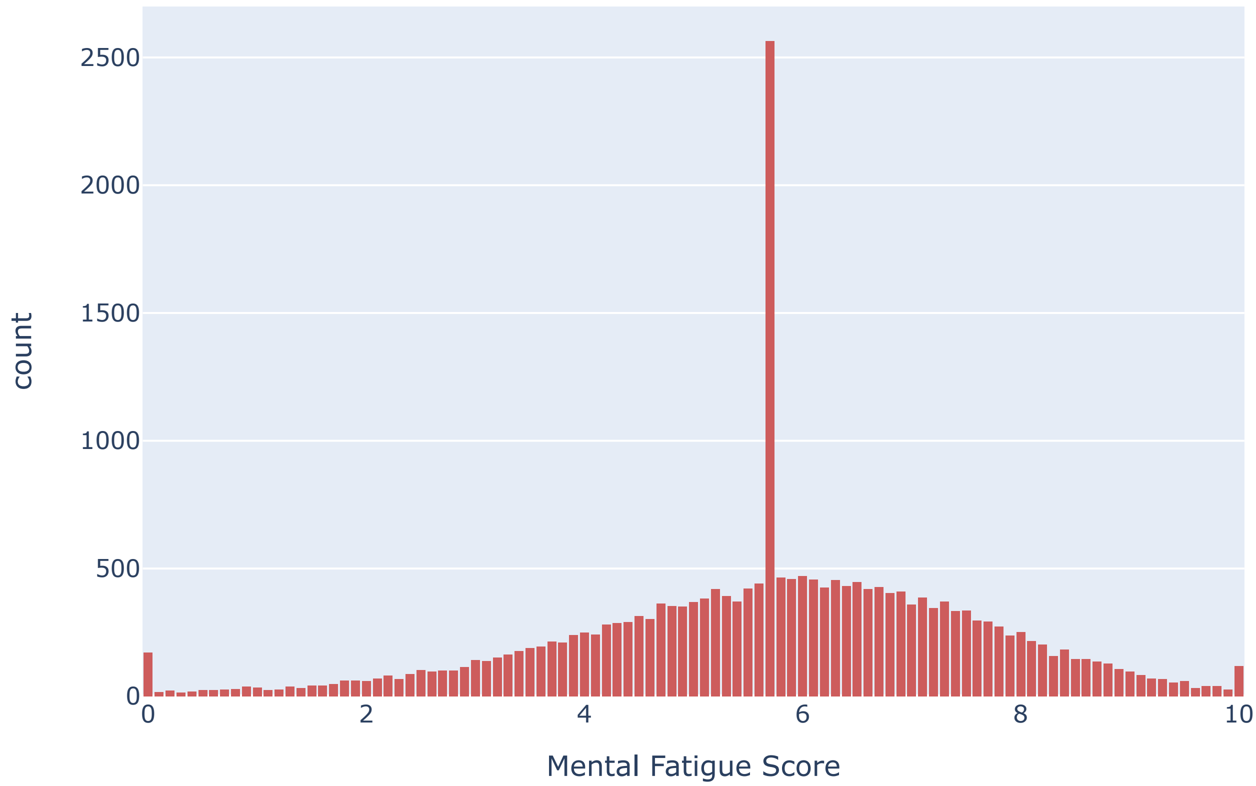

```
plt.figure(figsize=(10,8))
sns.countplot(x='Company Type',data=burnoutDf,palette="Spectral")
plt.title("plot distribution of Company Type")
plt.show()
```

plot distribution of Company Type

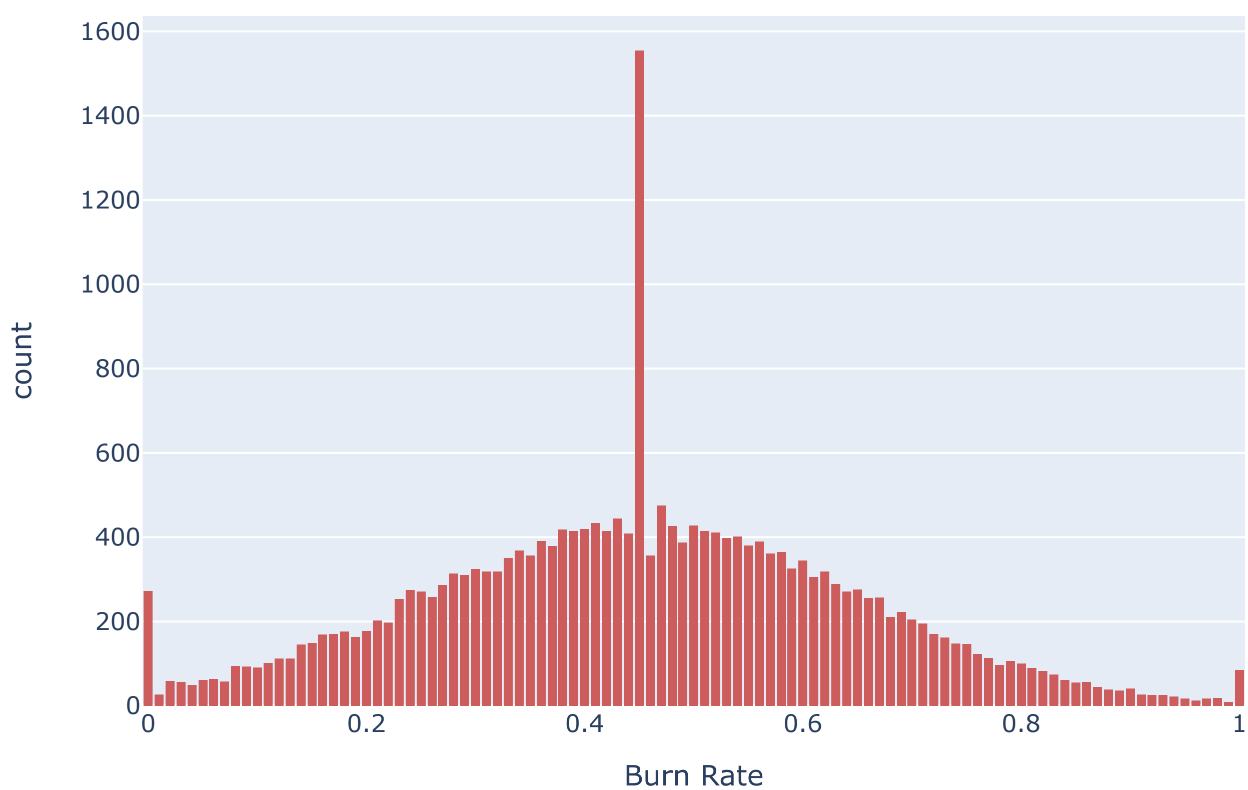
```
plt.figure(figsize=(10,8))
sns.countplot(x='WFH Setup Available',data=burnoutDf,palette="dark:salmon_r")
plt.title("plot distribution of WFh_setup_Avaiable")
plt.show()
```



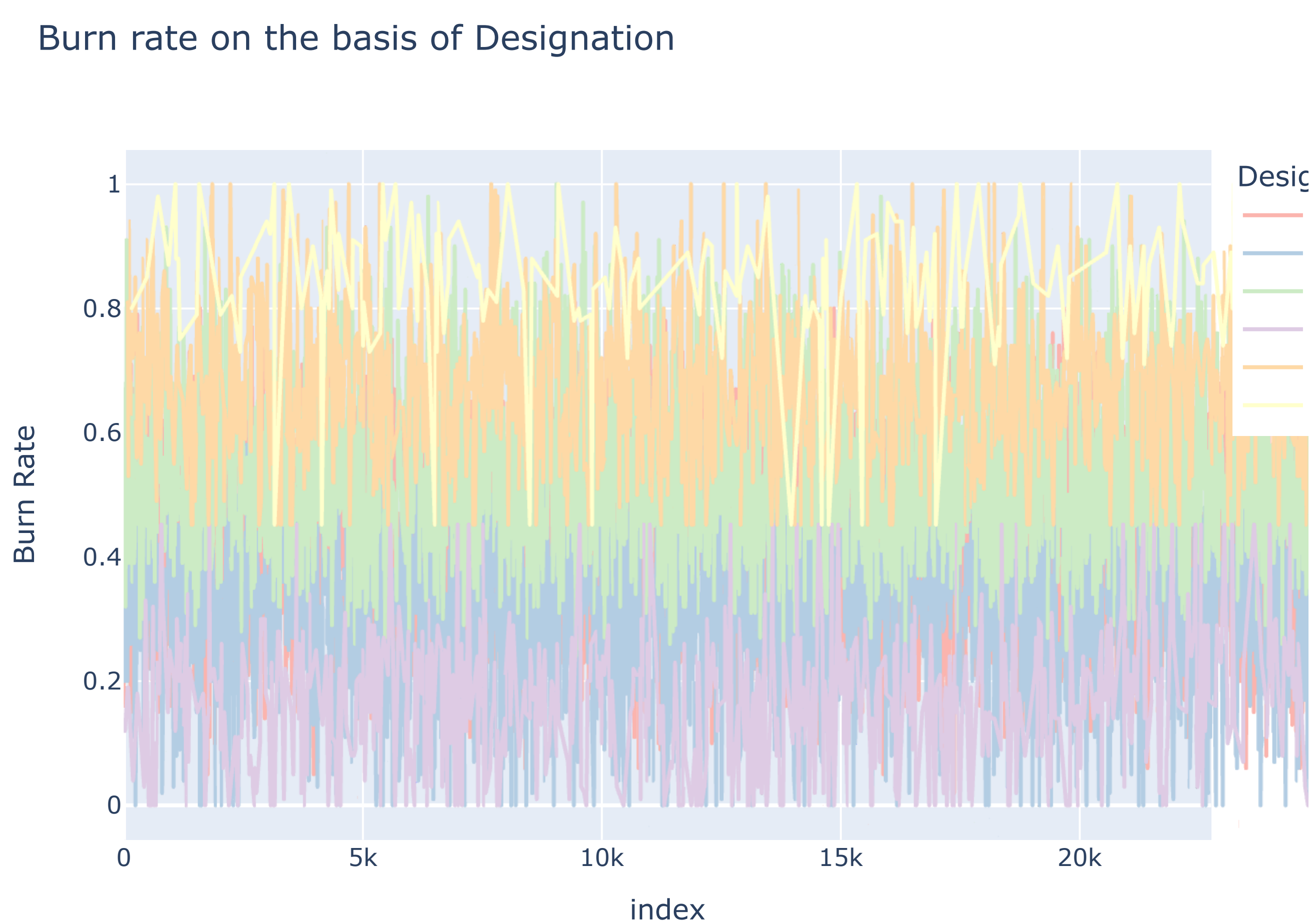

```
burn_st=burnoutDf.loc[:,'Date of Joining':'Burn Rate']
burn_st=burn_st.select_dtypes([int,float])
for i,col in enumerate(burn_st.columns):
  fig=px.histogram(burn_st,x=col,title="plot Distribution of "+col,color_discrete_sequence=['
  fig.update_layout(bargap=0.2)
  fig.show()
```



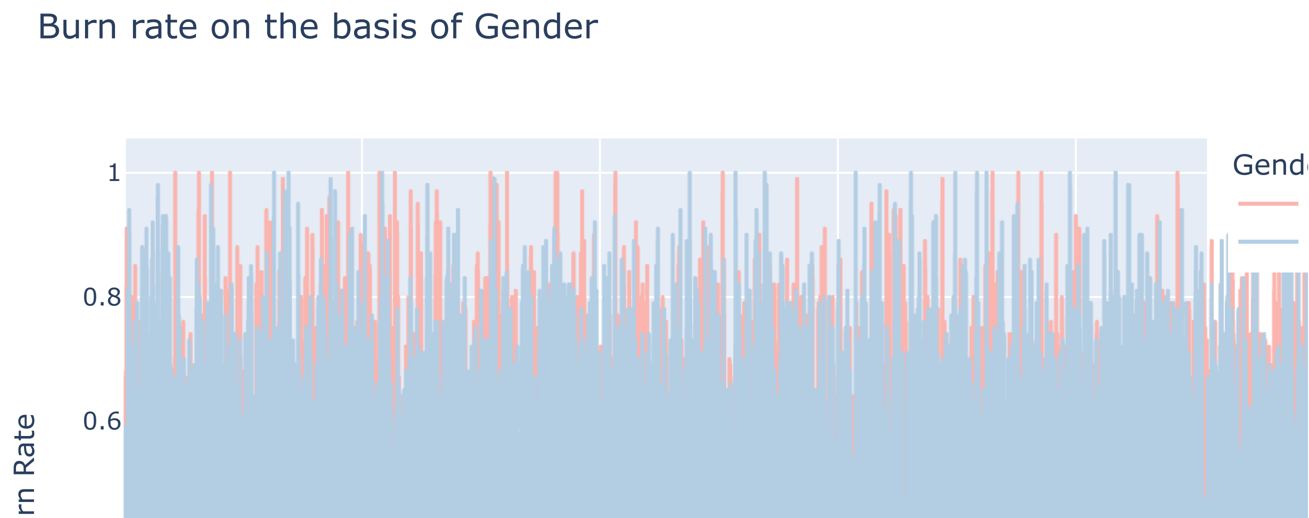
plot Distribution of Burn Rate



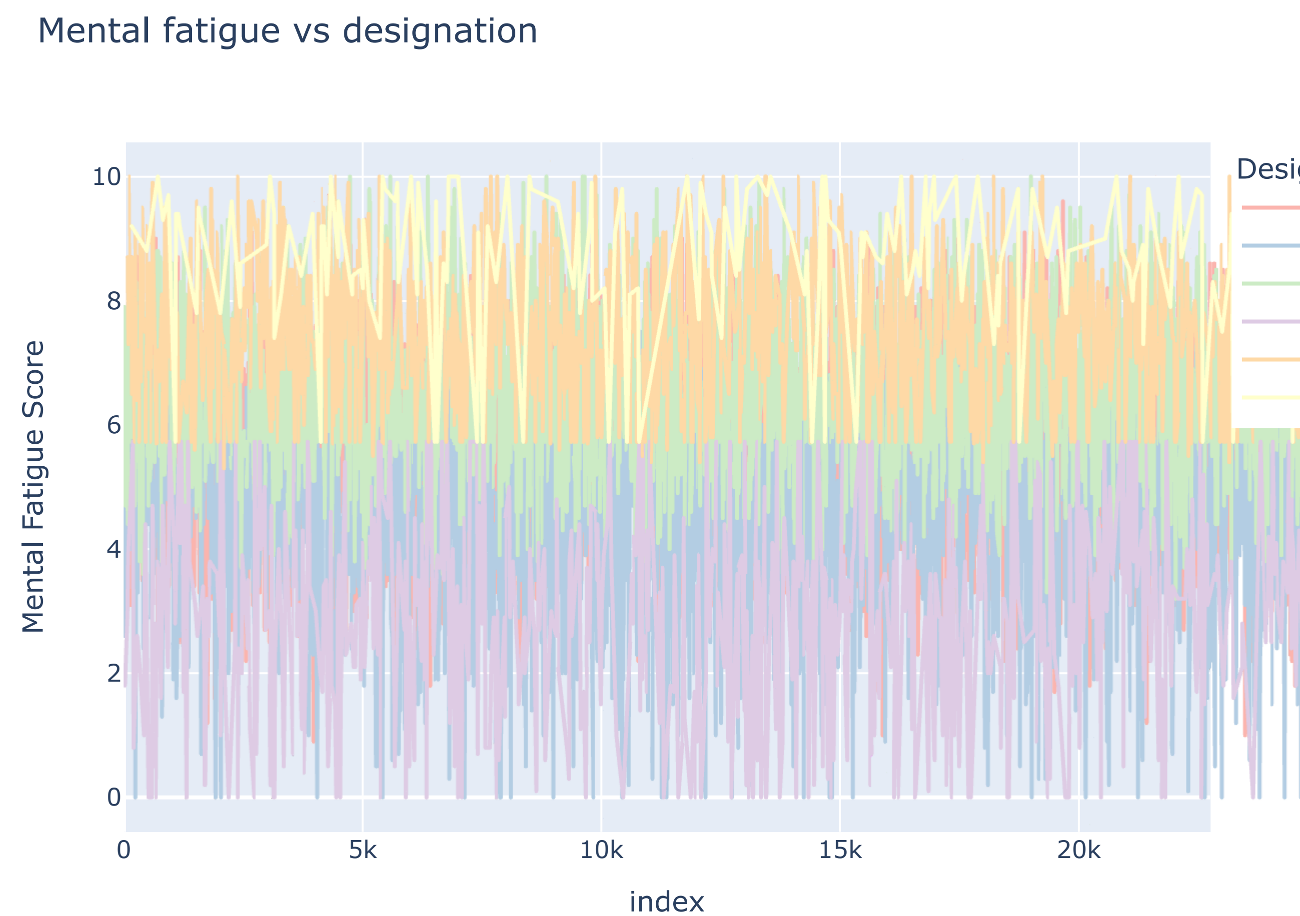
```
fig=px.line(burnoutDf,y="Burn Rate",color="Designation",title="Burn rate on the basis of Desi
fig.update_layout(bargap=0.1)
fig.show()
```




```
fig=px.line(burnoutDf,y="Burn Rate",color="Gender",title="Burn rate on the basis of Gender",c
fig.update_layout(bargap=0.2)
fig.show()
```




```
fig=px.line(burnoutDf,y="Mental Fatigue Score",color="Designation",title="Mental fatigue vs d
fig.update_layout(bargap=0.2)
fig.show()
```




```
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 0x7f0e25ec5fc0>
```


```
from sklearn import preprocessing
Label_encode=preprocessing.LabelEncoder()
```

```
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 Availab
```

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

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

```
ct = burnoutDf.groupby('Company Type')
ct = ct['Company_TypeLabel']
ct.first()
```

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

```
wsa=burnoutDf.groupby('WFH Setup Available')
wsa=wsa['WFH Setup AvailableLabel']
```

```
wsa=wsa['WFH_Setup_AvailableLabel']
wsa.first()
```

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

```
burnoutDf.tail(10)
```

	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
22740	2008-09-05	Female	Product	No	3	6.0	7.300000	0.550000
22741	2008-01-07	Male	Product	No	2	5.0	6.000000	0.452005
22742	2008-07-28	Male	Product	No	3	5.0	8.100000	0.690000
22743	2008-12-15	Female	Product	Yes	1	3.0	6.000000	0.480000
22744	2008-05-27	Male	Product	No	3	7.0	6.200000	0.540000
22745	2008-12-30	Female	Service	No	1	3.0	5.728188	0.410000
22746	2008-01-19	Female	Product	Yes	3	6.0	6.700000	0.590000
22747	2008-11-05	Male	Service	Yes	3	7.0	5.728188	0.720000
22748	2008-01-10	Female	Service	No	2	5.0	5.900000	0.520000
22749	2008-01-06	Male	Product	No	3	6.0	7.800000	0.610000

```
Columns=['Designation','Resource Allocation','Mental Fatigue Score','GenderLabel','Company_Ty
x=burnoutDf[Columns]
y=burnoutDf['Burn Rate']
```

Double-click (or enter) to edit

```
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

```
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
```

```
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(x_train_pca.shape,x_test.shape,y_train.shape,y_test.shape)
```

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

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

```
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 tarin data: "+str(round(100*train_r2, 4))+" %")
print("Accuracy score of test data: "+str(round(100*test_r2, 4))+" %")
```

```
Accuracy score of tarin data: 91.1793 %
Accuracy score of test data: 83.8979 %
```

```
from sklearn.ensemble import AdaBoostRegressor
abr_model=AdaBoostRegressor()
abr_model.fit(x_train_pca,y_train)

train_pred_adboost=abr_model.predict(x_train_pca)
train_r2=r2_score(y_train, train_pred_adboost)
test_pred_adaboost=abr_model.predict(x_test)
test_r2 = r2_score(y_test, test_pred_adaboost)
print("Accuracy score of tarin data: "+str(round(100*train_r2, 4))+" %")
print("Accuracy score of test data: "+str(round (100*test_r2, 4))+" %")
```

```
Accuracy score of tarin data: 77.6596 %
Accuracy score of test data: 77.0145 %
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_pca,y,test_size=0.25,random_state=10)
print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
```

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

```
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor
```

```
rf_model = RandomForestRegressor()
rf_model.fit(x_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
rf=rf_model.predict(x_train)
_score(y_train, train_pred_rf)
f=rf_model.predict(x_test)
score(y_test, test_pred_rf)
racy score of train data after random forest regression: "+str(round(100*train_r2, 4))+" %")
racy score of test data after random forest regression: "+str(round(100*test_r2, 4))+" %")
```

```
Accuracy score of train data after random forest regression: 91.1976 %
Accuracy score of test data after random forest regression: 83.8699 %
```

```
from sklearn.ensemble import AdaBoostRegressor
abr_model=AdaBoostRegressor()
abr_model.fit(x_train,y_train)
```

```
▾ AdaBoostRegressor
AdaBoostRegressor()
```