In [1]:
```
import warnings    # this module is used to ignore warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# import required modules and libraries

import pandas as pd
import numpy as np

# Below 3 libraries are used for visualization purpose
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

In [3]:
```
#  Loading the dataset
pd.set_option('display.max_columns',None)
burnoutDF=pd.read_excel('C:/Users/Ganes/Downloads/employee_burnout_analysis-AI
burnoutDF
```

Out[3]:

	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
...	...	...	...	...	...	...	...
22745	fffe31003500370039003100	2008-12-30	Female	Service	No	1	3.0
22746	fffe33003000350031003800	2008-01-19	Female	Product	Yes	3	6.0
22747	fffe390032003000	2008-11-05	Male	Service	Yes	3	7.0
22748	fffe33003300320036003900	2008-01-10	Female	Service	No	2	5.0
22749	fffe3400350031003800	2008-01-06	Male	Product	No	3	6.0

22750 rows × 9 columns

In [4]:
```
# Converting Date of Joining column to dateTime Datatype
burnoutDF["Date of Joining"]=pd.to_datetime(burnoutDF["Date of Joining"])
```

In [5]:
```
#  Describing the General ino for from the datase
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
```

In [6]:
```
# Displaying Number of rows and Number of Columns are there in Dataset
burnoutDF.shape
```

Out[6]: (22750, 9)

In [7]:
```
# Displaying top 5 rows
burnoutDF.head()
```

Out[7]:

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation	Me Fati Sc
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	

In [8]:
```
# Displaying last 5 rows
burnoutDF.tail()
```

Out[8]:

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation
22745	fffe31003500370039003100	2008-12-30	Female	Service	No	1	3.0
22746	fffe33003000350031003800	2008-01-19	Female	Product	Yes	3	6.0
22747	fffe390032003000	2008-11-05	Male	Service	Yes	3	7.0
22748	fffe33003300320036003900	2008-01-10	Female	Service	No	2	5.0
22749	fffe3400350031003800	2008-01-06	Male	Product	No	3	6.0

In [9]:
```
# Displaying all the column names present in dataset
burnoutDF.columns
```

Out[9]:
```
Index(['Employee ID', 'Date of Joining', 'Gender', 'Company Type',
       'WFH Setup Available', 'Designation', 'Resource Allocation',
       'Mental Fatigue Score', 'Burn Rate'],
      dtype='object')
```

In [10]:
```
# Chescking how many null vaues are there in each column of dataset
burnoutDF.isna().sum()
```

Out[10]:
```
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
```

In [11]:
```
# Checking whether there is any duplicate values are there in dataset
burnoutDF.duplicated().sum()
```

Out[11]:
```
0
```

In [12]:
```
# Displays the statistical values liek mean,std,min,max,and count of every att
burnoutDF.describe()
```

Out[12]:

	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

In [13]:
```
# Show the unique values
for i,col in enumerate(burnoutDF.columns):
    print(f"\n\n{burnoutDF[col].unique()}")
    print(f"\n\n{burnoutDF[col].value_counts()}\n\n")
```

```
 '2008-02-08T00:00:00.000000000' '2008-11-25T00:00:00.000000000'
 '2008-04-23T00:00:00.000000000' '2008-11-07T00:00:00.000000000'
 '2008-06-20T00:00:00.000000000' '2008-12-23T00:00:00.000000000'
 '2008-11-24T00:00:00.000000000' '2008-06-21T00:00:00.000000000'
 '2008-11-29T00:00:00.000000000' '2008-08-11T00:00:00.000000000'
 '2008-04-29T00:00:00.000000000' '2008-11-19T00:00:00.000000000'
 '2008-12-25T00:00:00.000000000' '2008-02-14T00:00:00.000000000'
 '2008-03-04T00:00:00.000000000' '2008-10-06T00:00:00.000000000'
 '2008-08-16T00:00:00.000000000' '2008-10-29T00:00:00.000000000'
 '2008-07-15T00:00:00.000000000' '2008-04-21T00:00:00.000000000'
 '2008-09-01T00:00:00.000000000' '2008-01-06T00:00:00.000000000'
 '2008-03-20T00:00:00.000000000' '2008-04-14T00:00:00.000000000'
 '2008-02-16T00:00:00.000000000' '2008-10-10T00:00:00.000000000'
 '2008-09-26T00:00:00.000000000' '2008-06-01T00:00:00.000000000'
 '2008-07-11T00:00:00.000000000' '2008-07-23T00:00:00.000000000'
 '2008-07-10T00:00:00.000000000' '2008-10-05T00:00:00.000000000'
 '2008-03-14T00:00:00.000000000' '2008-06-14T00:00:00.000000000'
 '2008-10-23T00:00:00.000000000' '2008-02-22T00:00:00.000000000'
 '2008-05-19T00:00:00.000000000' '2008-09-20T00:00:00.000000000'
 '2008-01-18T00:00:00.000000000' '2008-07-13T00:00:00.000000000'
```

In [14]:
```
# Drop Irrelevant columns
# 1 for columns and 0 for rows
#burnoutDF.drop(['Employee ID'],axis=1)

burnoutDF
```

Out[14]:

	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
...	...	...	...	...	...	...	...
22745	fffe31003500370039003100	2008-12-30	Female	Service	No	1	3.0
22746	fffe33003000350031003800	2008-01-19	Female	Product	Yes	3	6.0
22747	fffe390032003000	2008-11-05	Male	Service	Yes	3	7.0
22748	fffe33003300320036003900	2008-01-10	Female	Service	No	2	5.0
22749	fffe3400350031003800	2008-01-06	Male	Product	No	3	6.0

22750 rows × 9 columns

In [15]:
```
# 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 :",intFloat
    elif (intFloatburnoutDF[col].skew()<=-0.1):
         print("\n",col," feature is negitively skewed and value is :",intFloa
    else:
         print("\n",col," feature is Normally skewed and value is :",intFloatb
```

```
 Designation  feature is Normally skewed and value is : 0.09242138478903683

 Resource Allocation  feature is positively skewed and value is : 0.204572734
54318103

 Mental Fatigue Score  feature is negitively skewed and value is : -0.4308950
578815428

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

In [16]:
```
# Replacing the null values with mean value
burnoutDF['Resource Allocation'].fillna(burnoutDF['Resource Allocation'].mean(
burnoutDF['Mental Fatigue Score'].fillna(burnoutDF['Mental Fatigue Score'].mea
burnoutDF['Burn Rate'].fillna(burnoutDF['Burn Rate'].mean(),inplace=True)
```

In [17]:
```
# Check for null Values
burnoutDF.isna().sum()
```

Out[17]:
```
Employee ID             0
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
```

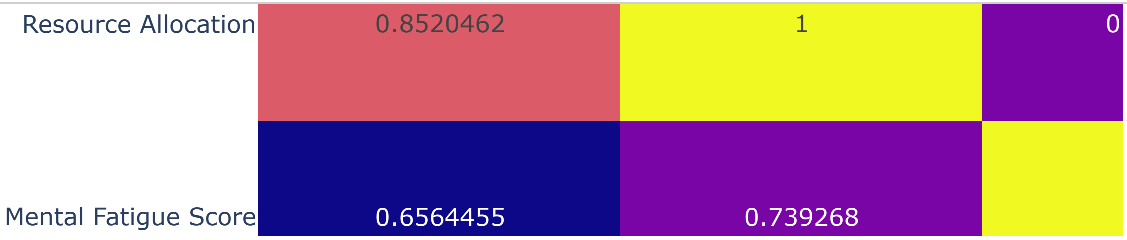
In [18]:
```
# Display the correlation
burnoutDF.corr()
```

Out[18]:

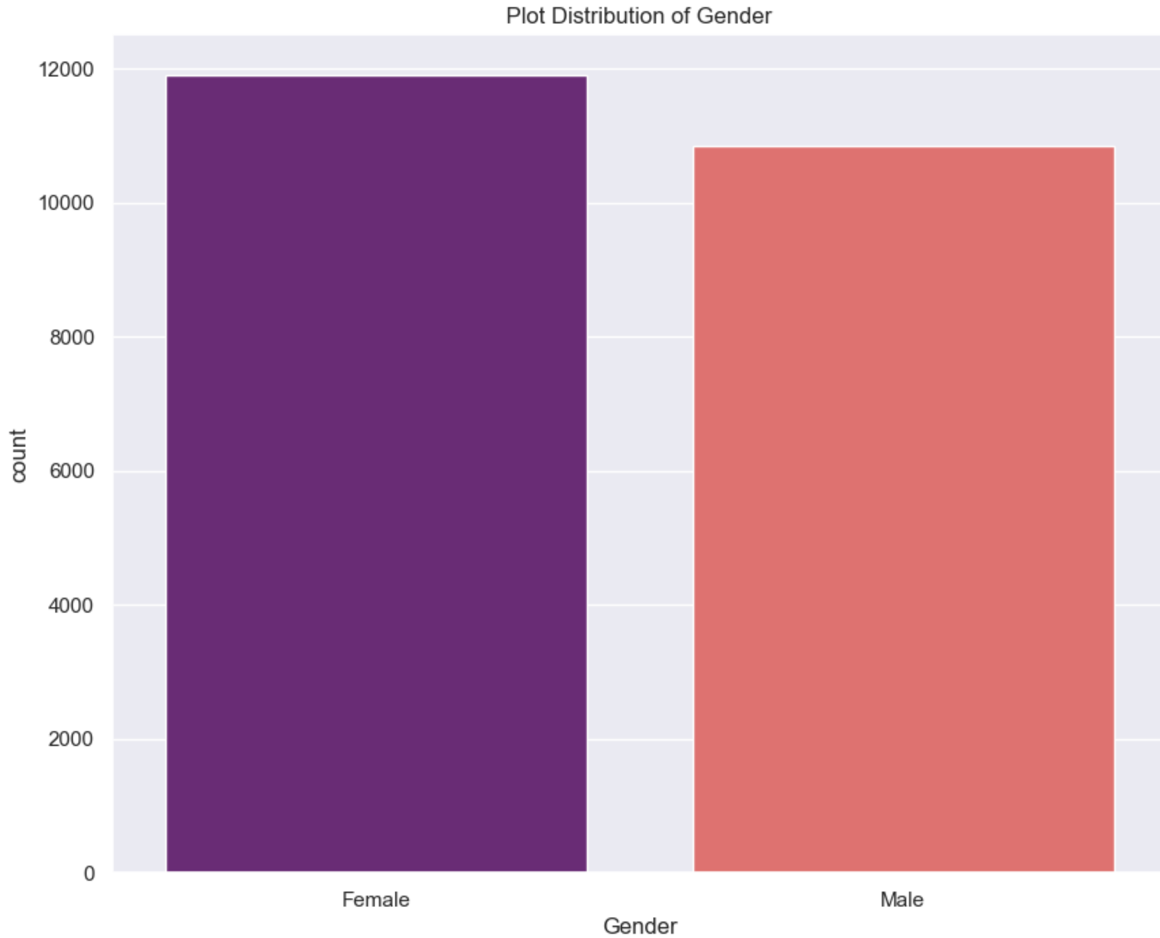
	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
Designation	1.000000	0.852046	0.656445	0.719284
Resource Allocation	0.852046	1.000000	0.739268	0.811062
Mental Fatigue Score	0.656445	0.739268	1.000000	0.878217
Burn Rate	0.719284	0.811062	0.878217	1.000000

Data Visualization

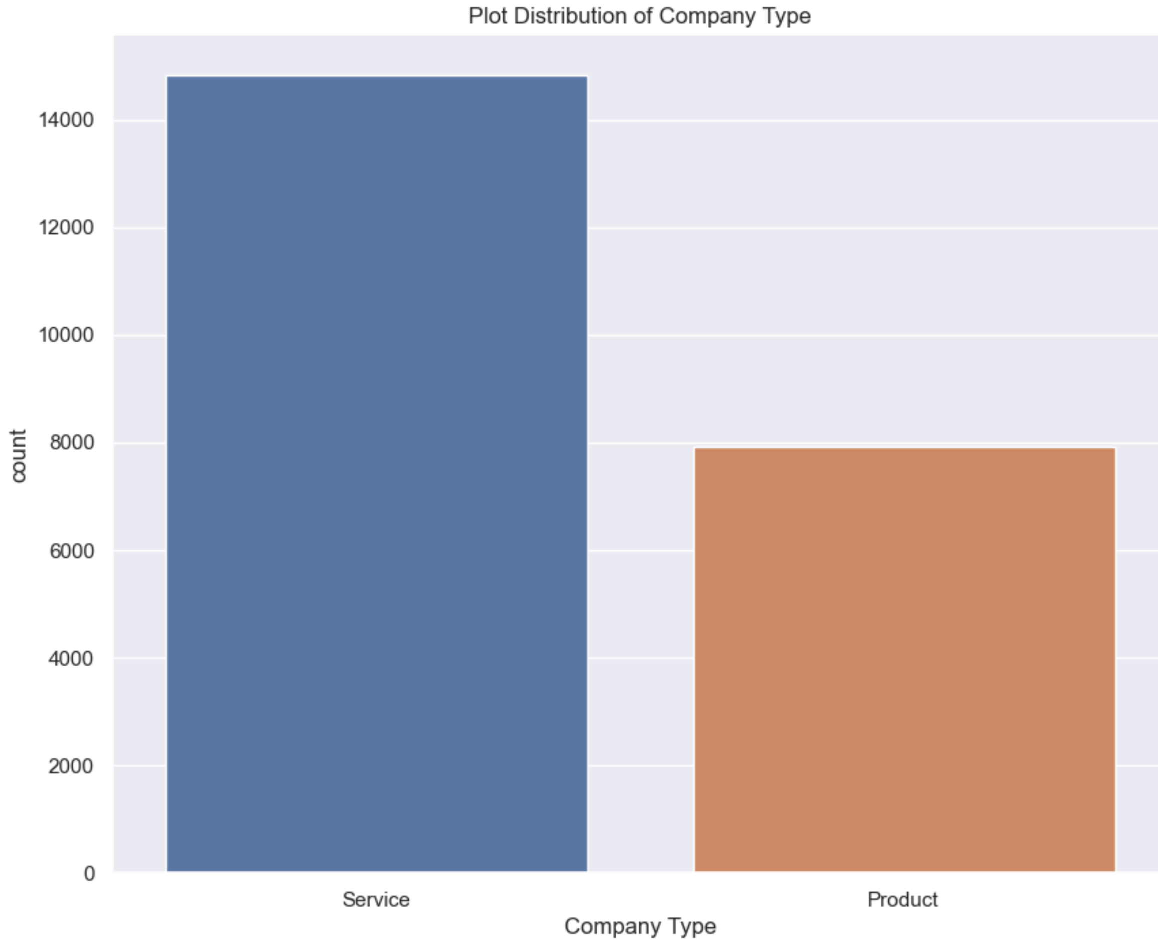
In [20]:
```
# Ploting a Heat Map to check correlation
corr=burnoutDF.corr()
sns.set(rc={'figure.figsize':(14,12)})
fig=px.imshow(corr,text_auto=True,aspect="auto")
fig.show()
```



In [21]:
```
# Count Plot Distribution of Gender
plt.figure(figsize=(10,8))
sns.countplot(x="Gender",data=burnoutDF,palette="magma") # Palette means color
plt.title(" Plot Distribution of Gender ")
plt.show()
```

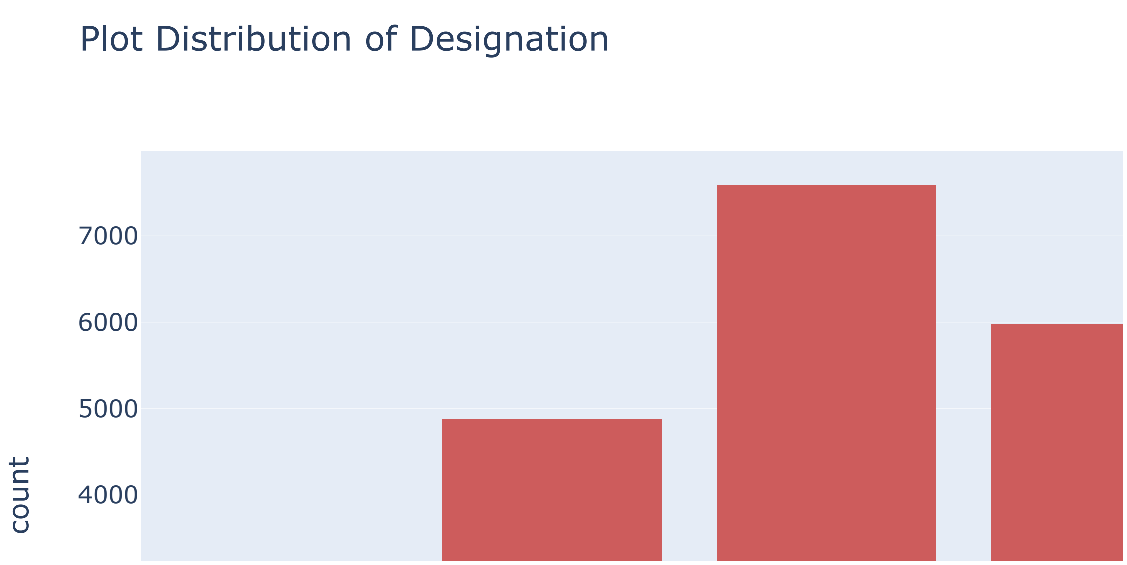


In [22]:
```
# Count Plot Distribution of " Company Type"
plt.figure(figsize=(10,8))
sns.countplot(x="Company Type",data=burnoutDF) # Palette means color
plt.title(" Plot Distribution of Company Type ")
plt.show()
```

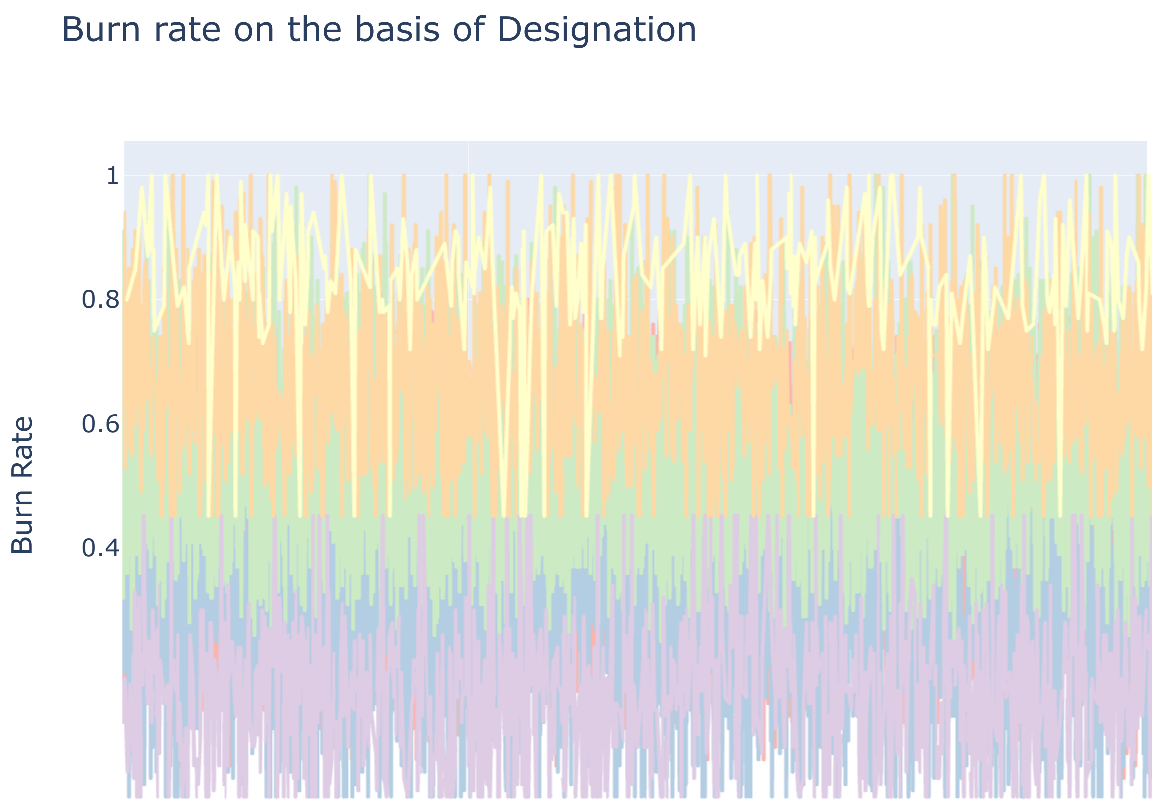


In [23]:
```
# Count Plot Distribution of " WFH setUp Available"
plt.figure(figsize=(10,8))
sns.countplot(x="WFH Setup Available",data=burnoutDF,palette="dark:salmon_r")
plt.title(" Plot Distribution of WFH setUp Available ")
plt.show()
```

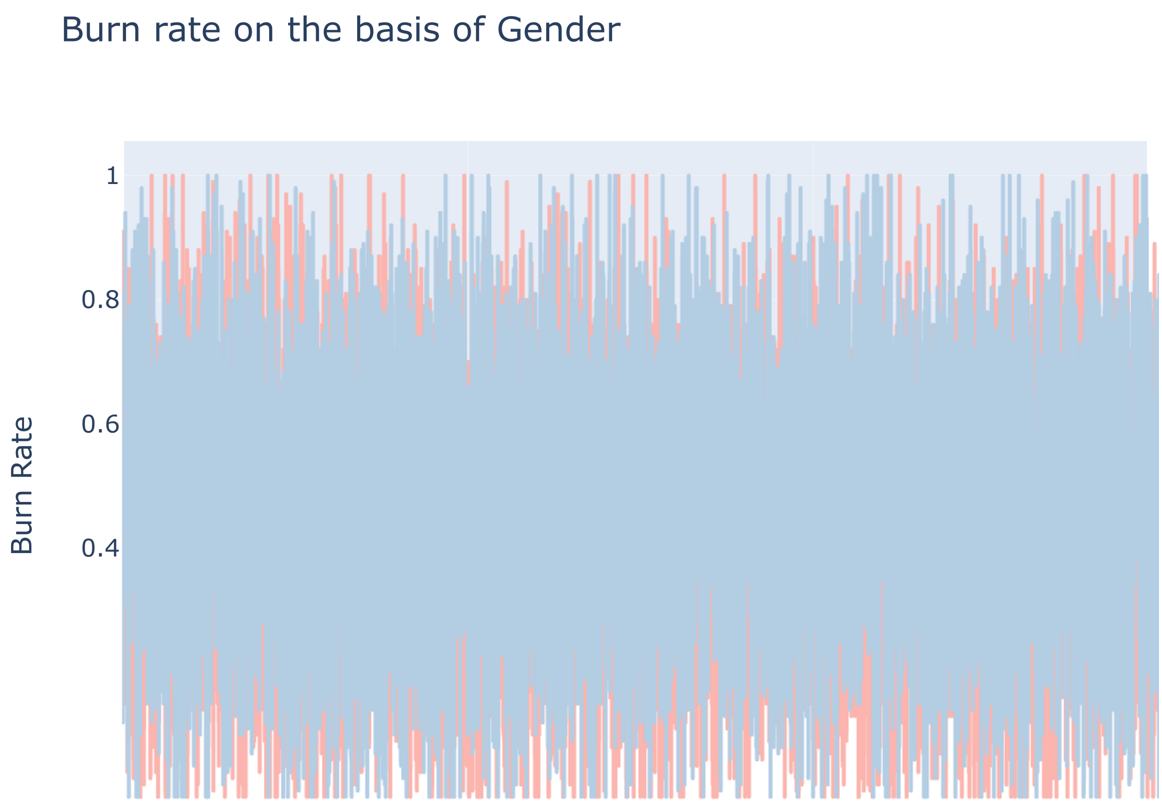
In [24]:
```
# Count Plot Distribution of attributes with the help of Histogram
burn_st=burnoutDF.loc[:,'Date of Joining':'Burn Rate']
burn_st=burn_st.select_dtypes([int,float])
for i,col in enumerate(burn_st.columns):
    fig=px.histogram(burn_st,x=col,title="Plot Distribution of "+col,color_dis
    fig.update_layout(bargap=0.2)
    fig.show()
```



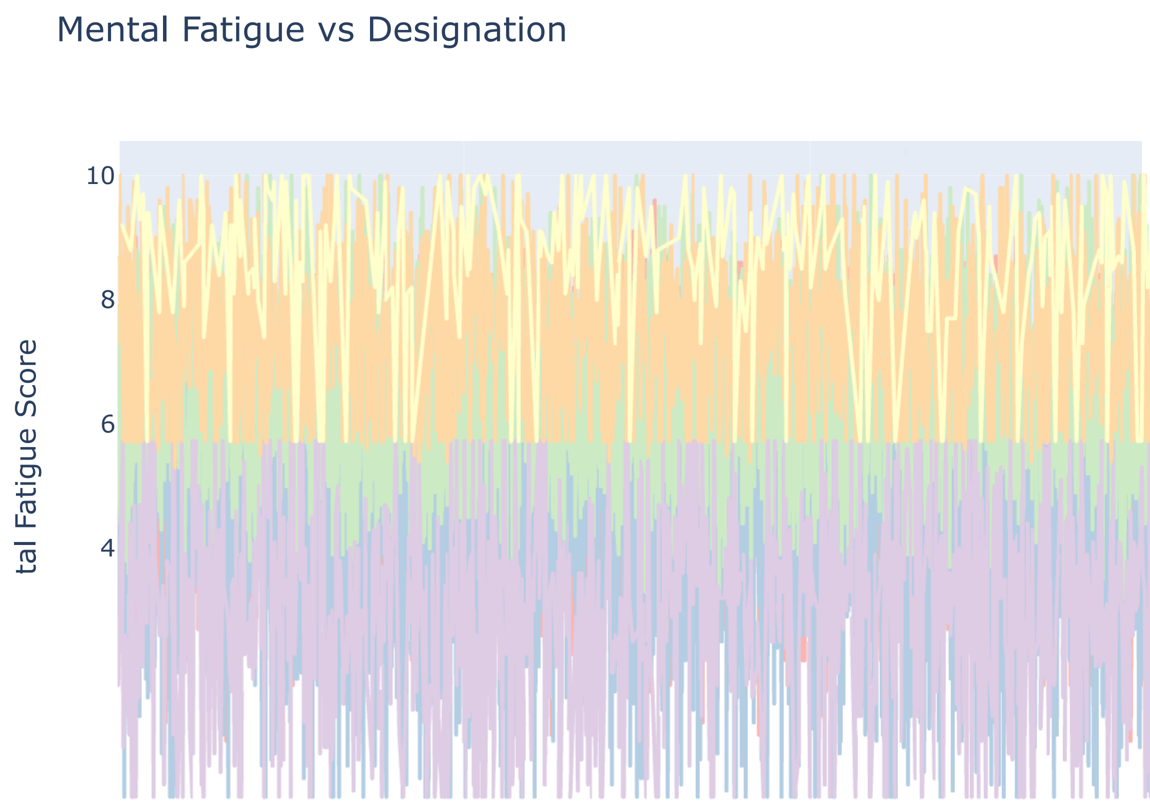
In [25]:
```
# Plot distribution of Burn rate on the basis of Designation
fig=px.line(burnoutDF,y="Burn Rate",color="Designation",title="Burn rate on th
fig.update_layout(bargap=0.1)
fig.show()
```



In [26]:
```
# Plot distribution of Burn rate on the basis of Gender
fig=px.line(burnoutDF,y="Burn Rate",color="Gender",title="Burn rate on the bas
fig.update_layout(bargap=0.2)
fig.show()
```



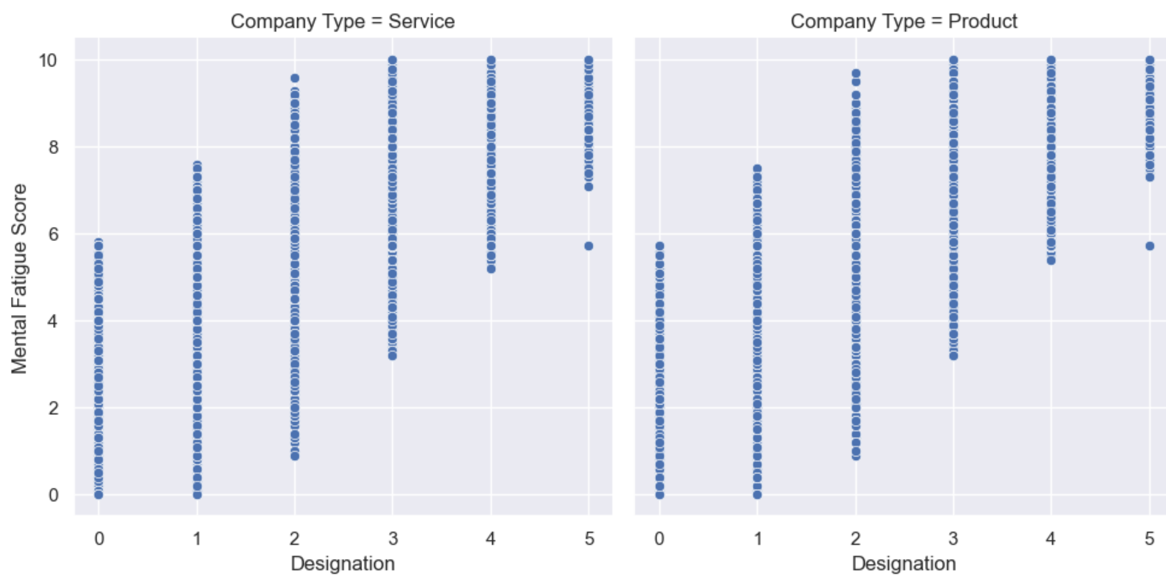
In [27]:
```
# Plot Distribution of mental fatigue score on the basis of designation
fig=px.line(burnoutDF,y="Mental Fatigue Score",color="Designation",title="Ment
fig.update_layout(bargap=0.2)
fig.show()
```



In [28]:
```
# Plot Distribution of " Designation vs Mental Fatigue " as per company type ,
sns.relplot(
    data=burnoutDF,x="Designation",y="Mental Fatigue Score",col="Company Type"
    palette=["g","r"],sizes=(50,200)

)
```

Out[28]: <seaborn.axisgrid.FacetGrid at 0x1fd553ef370>



Label Encoding

In [29]:
```
# Label encoding and assign in a new variable
from sklearn import preprocessing
Label_encode=preprocessing.LabelEncoder()
```

In [30]:
```
# Assigning a new variable  ( or ) Renaming the column names
burnoutDF['GenderLabel']=Label_encode.fit_transform(burnoutDF['Gender'].values
burnoutDF['Company_TypeLabel']=Label_encode.fit_transform(burnoutDF['Company T
burnoutDF['WFH_Setup_AvailableLabel']=Label_encode.fit_transform(burnoutDF['WF
```

In [31]:
```
# Checked Assigned Values
# It replaces Female value with " 0 " and Male Value with " 1 "
gn=burnoutDF.groupby('Gender')
gn=gn['GenderLabel']
gn.first()
```

Out[31]:
```
Gender
Female    0
Male      1
Name: GenderLabel, dtype: int32
```

In [32]:
```
# Check Assigned Values
# It replaces Product value with " 0 " and Service Value with " 1 "
ct=burnoutDF.groupby('Company Type')
ct=ct['Company_TypeLabel']
ct.first()
```

Out[32]:
```
Company Type
Product    0
Service    1
Name: Company_TypeLabel, dtype: int32
```

In [33]:
```
# Check assigned values
# It replaces No value with " 0 " and YES Value with " 1 "
wsa=burnoutDF.groupby('WFH Setup Available')
wsa=wsa['WFH_Setup_AvailableLabel']
wsa.first()
```

Out[33]:
```
WFH Setup Available
No     0
Yes    1
Name: WFH_Setup_AvailableLabel, dtype: int32
```

In [34]:
```
# Show last 10 rows
burnoutDF.tail(10)
```

Out[34]:

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation
22740	fffe33003300380031003100	2008-09-05	Female	Product	No	3	6.0
22741	fffe31003600350034003800	2008-01-07	Male	Product	No	2	5.0
22742	fffe33003200310039003000	2008-07-28	Male	Product	No	3	5.0
22743	fffe3300390030003600	2008-12-15	Female	Product	Yes	1	3.0
22744	fffe32003500370033003200	2008-05-27	Male	Product	No	3	7.0
22745	fffe31003500370039003100	2008-12-30	Female	Service	No	1	3.0
22746	fffe33003000350031003800	2008-01-19	Female	Product	Yes	3	6.0
22747	fffe390032003000	2008-11-05	Male	Service	Yes	3	7.0
22748	fffe33003300320036003900	2008-01-10	Female	Service	No	2	5.0
22749	fffe3400350031003800	2008-01-06	Male	Product	No	3	6.0

Feature Selection

In [35]:
```python
# These are the columns that are required to train Model and produce result
Columns=['Designation','Resource Allocation','Mental Fatigue Score',
        'GenderLabel','Company_TypeLabel','WFH_Setup_AvailableLabel']
x=burnoutDF[Columns]
y=burnoutDF['Burn Rate']
```

In [36]:
```python
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]
```

In [37]:
```python
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 (Principal Component Analysis)

In [38]:
```
# The Principal Component Analysis is a popular unsupervised learning techniqu
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_rati
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

In [39]:
```
# Data Splitting in train and test
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,ranc
```

In [40]:
```
# 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 Forest Regression

In [45]:
```
from sklearn.metrics import r2_score
```

In [49]:
```
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)
# Accuracy Score
print("Accuracy score of train data: "+str(round(100*train_r2,4))+" %")
print("Accuracy score of test data: "+str(round(100*test_r2,4))+" %")
```

```
Accuracy score of train data: 91.1819 %
Accuracy score of test data: 83.9133 %
```

AdaBoost Regressor

In [44]:
```
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)

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

```
Accuracy score of train data: 77.6044 %
Accuracy score of test data: 76.9392 %
```

In [ ]:
```

```