In [137]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import MinMaxScaler
import joblib
```

In [138]:
```
df = pd.read_csv("C:/Users/shami/Downloads/archive.zip")

df.head()
```

Out[138]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77
1	2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
2	3	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85

In [139]:
```
df.describe()
```

Out[139]:

	Person ID	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate	Daily Steps
count	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000
mean	187.500000	42.184492	7.132086	7.312834	59.171123	5.385027	70.165775	6816.844920
std	108.108742	8.673133	0.795657	1.196956	20.830804	1.774526	4.135676	1617.915679
min	1.000000	27.000000	5.800000	4.000000	30.000000	3.000000	65.000000	3000.000000
25%	94.250000	35.250000	6.400000	6.000000	45.000000	4.000000	68.000000	5600.000000
50%	187.500000	43.000000	7.200000	7.000000	60.000000	5.000000	70.000000	7000.000000
75%	280.750000	50.000000	7.800000	8.000000	75.000000	7.000000	72.000000	8000.000000
max	374.000000	59.000000	8.500000	9.000000	90.000000	8.000000	86.000000	10000.000000

In [140]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Person ID                374 non-null    int64  
 1   Gender                   374 non-null    object 
 2   Age                      374 non-null    int64  
 3   Occupation               374 non-null    object 
 4   Sleep Duration           374 non-null    float64
 5   Quality of Sleep         374 non-null    int64  
 6   Physical Activity Level  374 non-null    int64  
 7   Stress Level             374 non-null    int64  
 8   BMI Category             374 non-null    object 
 9   Blood Pressure           374 non-null    object 
 10  Heart Rate               374 non-null    int64  
 11  Daily Steps              374 non-null    int64  
 12  Sleep Disorder           155 non-null    object 
dtypes: float64(1), int64(7), object(5)
memory usage: 38.1+ KB
```

In [141]:
```
df.isnull().sum()
```

Out[141]:

```
Person ID                    0
Gender                       0
Age                          0
Occupation                   0
Sleep Duration               0
Quality of Sleep             0
Physical Activity Level      0
Stress Level                 0
BMI Category                 0
Blood Pressure               0
Heart Rate                   0
Daily Steps                  0
Sleep Disorder             219
dtype: int64
```

In [142]:
```
df['Sleep Disorder'].value_counts()
```

Out[142]:
```
Sleep Disorder
Sleep Apnea    78
Insomnia       77
Name: count, dtype: int64
```

In [143]:
```
df['Sleep Disorder'] = df['Sleep Disorder'].fillna('No Disorder')
df
```

Out[143]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77
1	2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
2	3	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85
...	...	...	...	...	...	...	...	...	...	...	...
369	370	Female	59	Nurse	8.1	9	75	3	Overweight	140/95	68
370	371	Female	59	Nurse	8.0	9	75	3	Overweight	140/95	68
371	372	Female	59	Nurse	8.1	9	75	3	Overweight	140/95	68
372	373	Female	59	Nurse	8.1	9	75	3	Overweight	140/95	68
373	374	Female	59	Nurse	8.1	9	75	3	Overweight	140/95	68

374 rows × 13 columns

In [144]:
```
df.drop_duplicates()
```

Out[144]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77
1	2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
2	3	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85
...	...	...	...	...	...	...	...	...	...	...	...
369	370	Female	59	Nurse	8.1	9	75	3	Overweight	140/95	68
370	371	Female	59	Nurse	8.0	9	75	3	Overweight	140/95	68
371	372	Female	59	Nurse	8.1	9	75	3	Overweight	140/95	68
372	373	Female	59	Nurse	8.1	9	75	3	Overweight	140/95	68
373	374	Female	59	Nurse	8.1	9	75	3	Overweight	140/95	68

374 rows × 13 columns

In [145]:
```
df[['SYSTOLIC', 'DIASTOLIC']] = df['Blood Pressure'].str.split('/', expand=True)

df['SYSTOLIC'] = df['SYSTOLIC'].astype(float)
df['DIASTOLIC'] = df['DIASTOLIC'].astype(float)

df.head()
```

Out[145]:

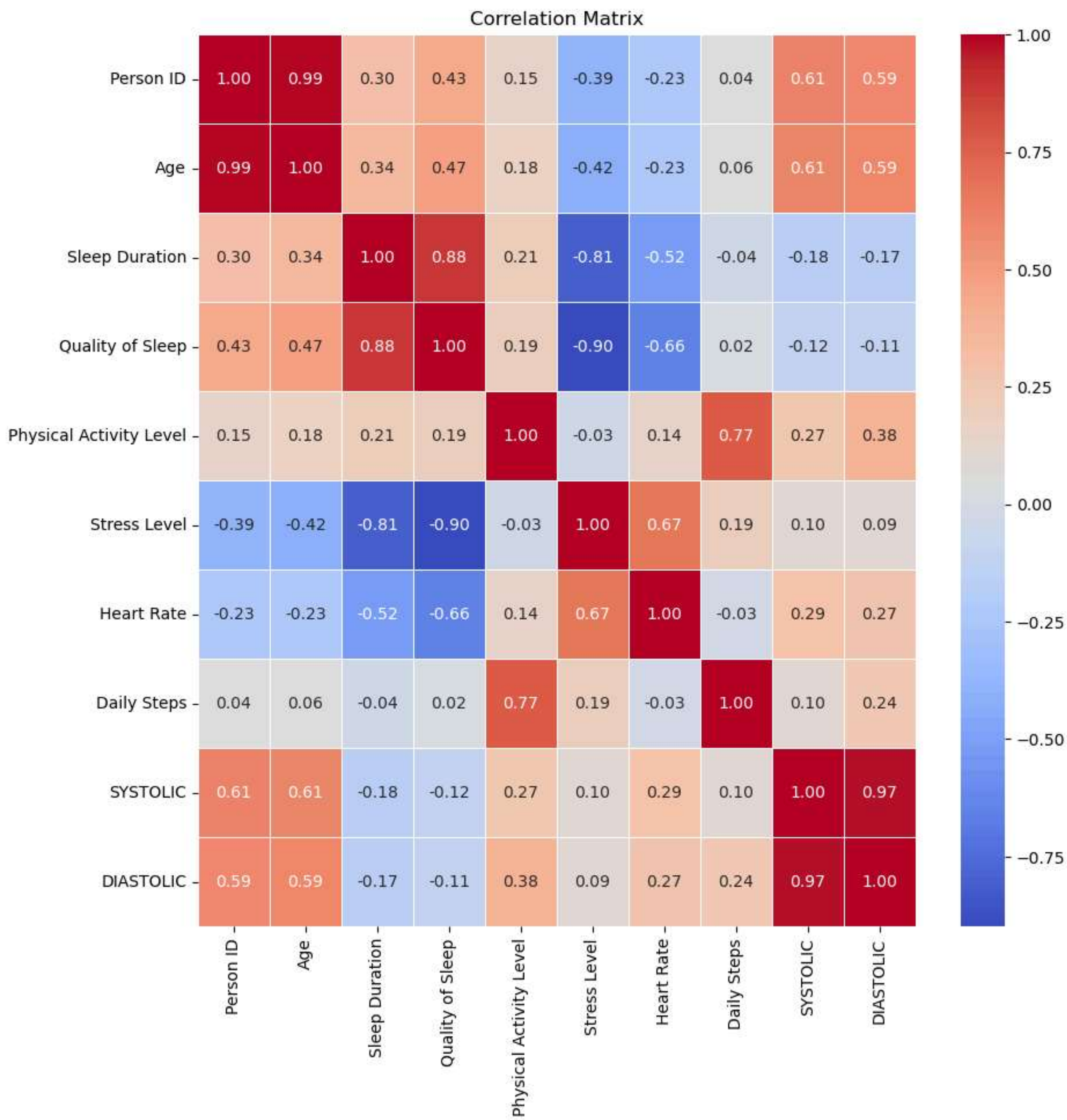
	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77
1	2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
2	3	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85

In [146]:
```
data = df.copy()

data = data.drop(['Person ID', 'Blood Pressure'], axis=1)

data.head()
```

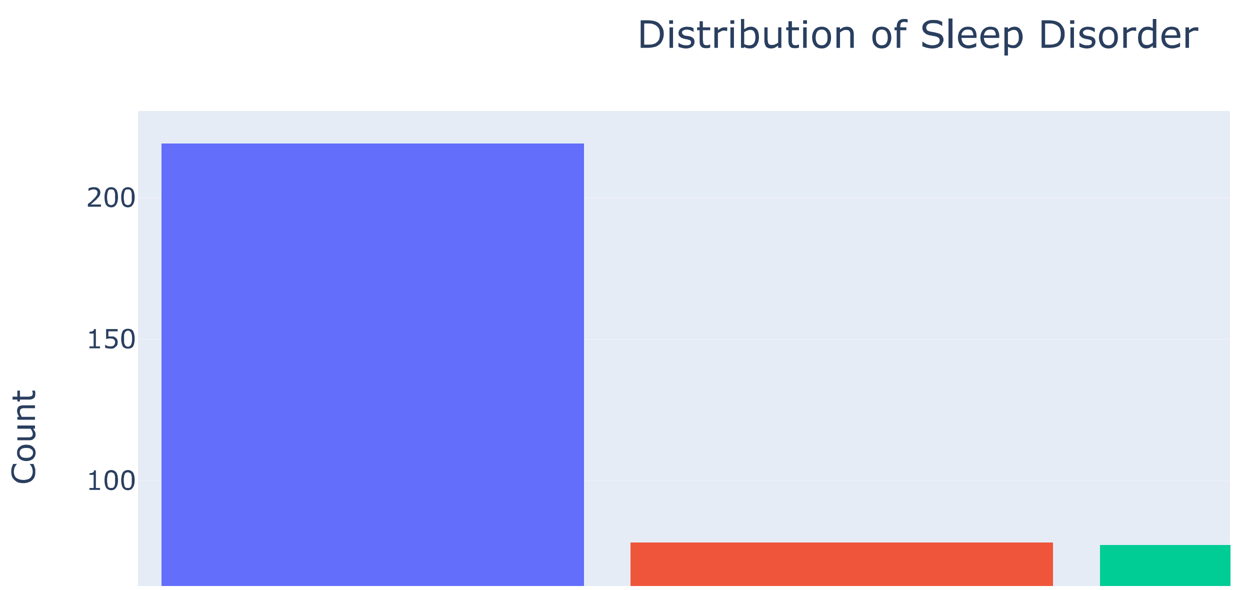
Out[146]:

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Heart Rate	Daily Steps	Sleep Disorder	Sy
0	Male	27	Software Engineer	6.1	6	42	6	Overweight	77	4200	No Disorder	
1	Male	28	Doctor	6.2	6	60	8	Normal	75	10000	No Disorder	
2	Male	28	Doctor	6.2	6	60	8	Normal	75	10000	No Disorder	
3	Male	28	Sales Representative	5.9	4	30	8	Obese	85	3000	Sleep Apnea	
4	Male	28	Sales Representative	5.9	4	30	8	Obese	85	3000	Sleep Apnea	

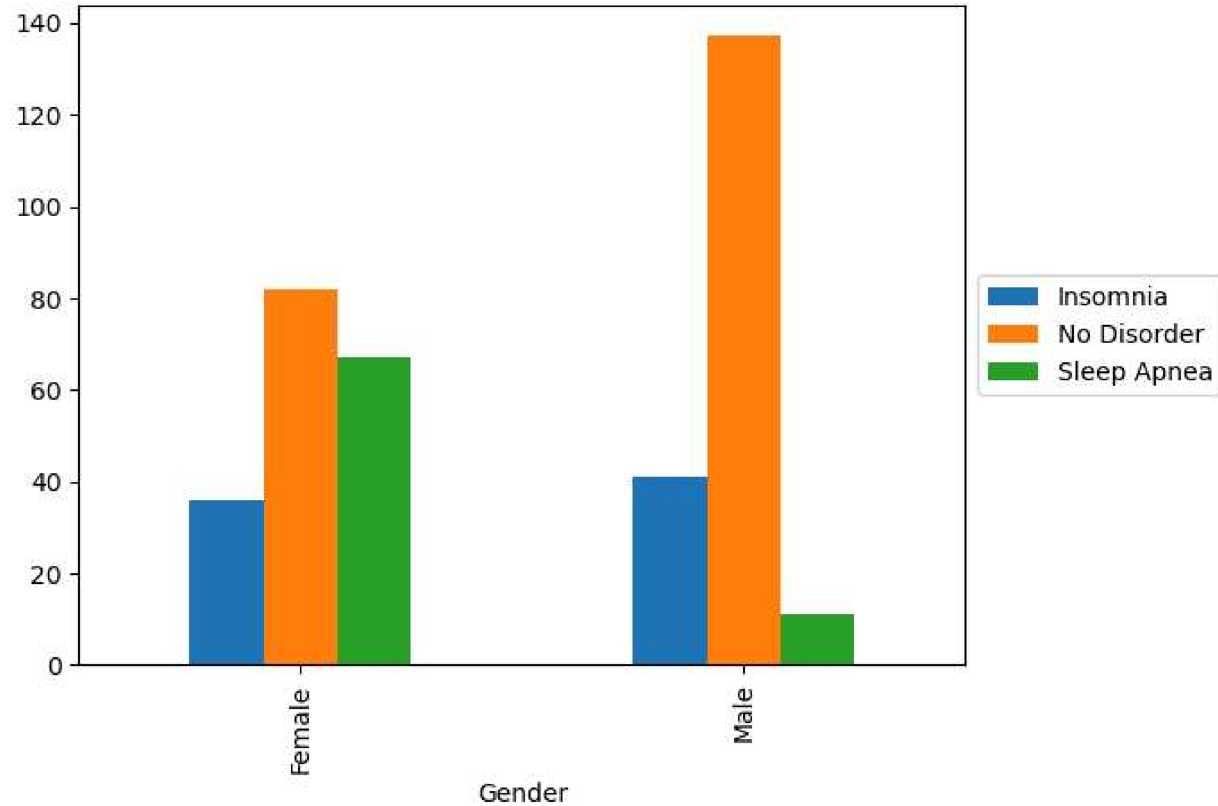
In [147]:
```
num_cols = df.select_dtypes(include=['int64', 'float64']).columns

corr = df[num_cols].corr()

plt.figure(figsize=(10, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



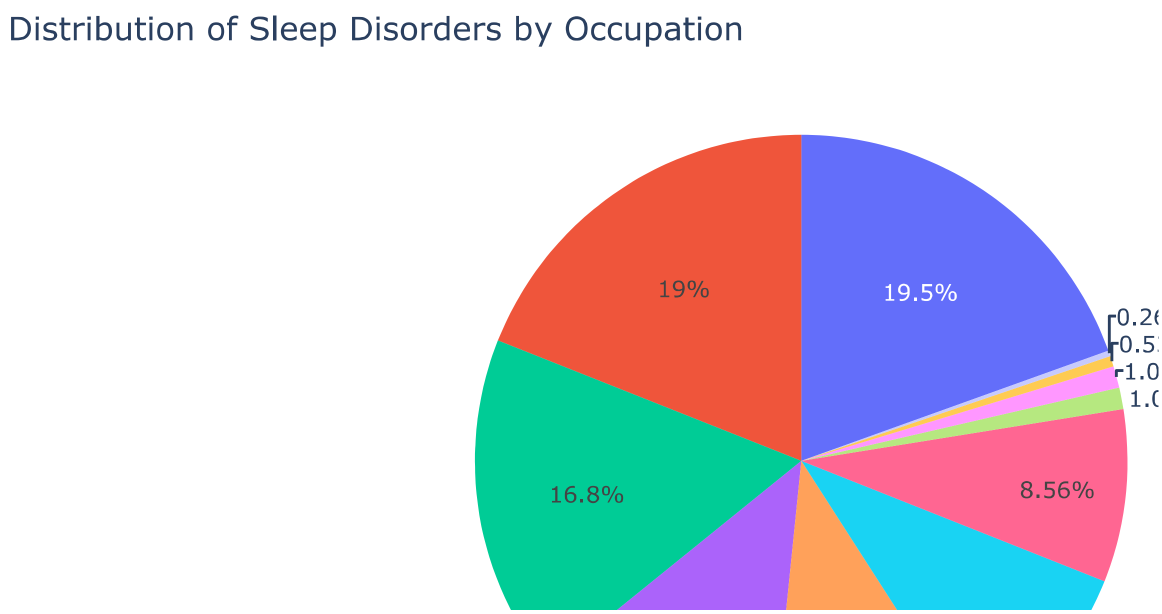
In [148]:
```
# Assuming 'data' is your DataFrame and it contains a column named 'Sleep Disorder'
fig = px.histogram(data, x='Sleep Disorder', title='Distribution of Sleep Disorder',
                   labels={'Sleep Disorder': 'Sleep Disorder'},
                   color='Sleep Disorder',
                   template='plotly')

fig.update_layout(
    xaxis_title='Sleep Disorder',
    yaxis_title='Count',
    title={
        'text': "Distribution of Sleep Disorder",
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'},
    font=dict(size=14),
#     plot_bgcolor='rgba(0,0,0,0)',  # Transparent plot background
#     paper_bgcolor='rgba(0,0,0,0)',  # Transparent paper background
    bargap=0,  # Set the gap between bars to 0
    bargroupgap=0.1  # Set the gap between groups of bars
)

# Set opacity of the bars
# fig.update_traces(opacity=0.75)

fig.show()
```



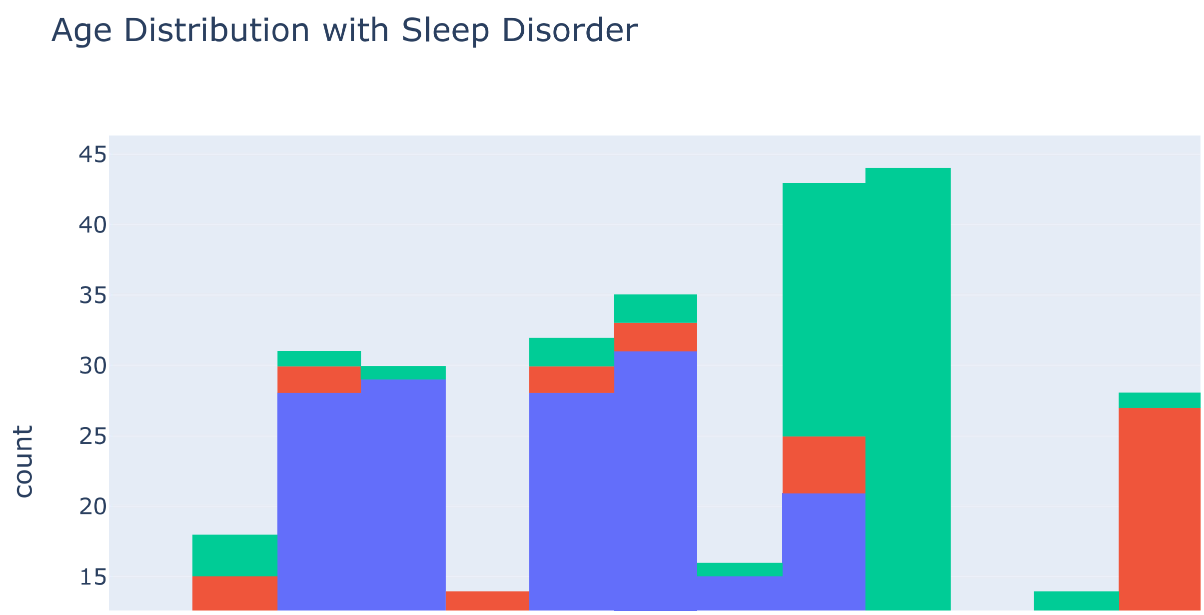
In [149]:
```
pd.crosstab(df["Gender"],df["Sleep Disorder"]).plot(kind="bar")
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



In [150]:
```
occupation_counts = data['Occupation'].value_counts().reset_index()
occupation_counts.columns = ['Occupation', 'Count']

fig = px.pie(occupation_counts, names='Occupation', values='Count',
             title='Distribution of Sleep Disorders by Occupation',
             labels={'Occupation': 'Occupation', 'Count': 'Count'},
             template='plotly')
# fig.update_xaxes(tickangle=45)
fig.show()
```



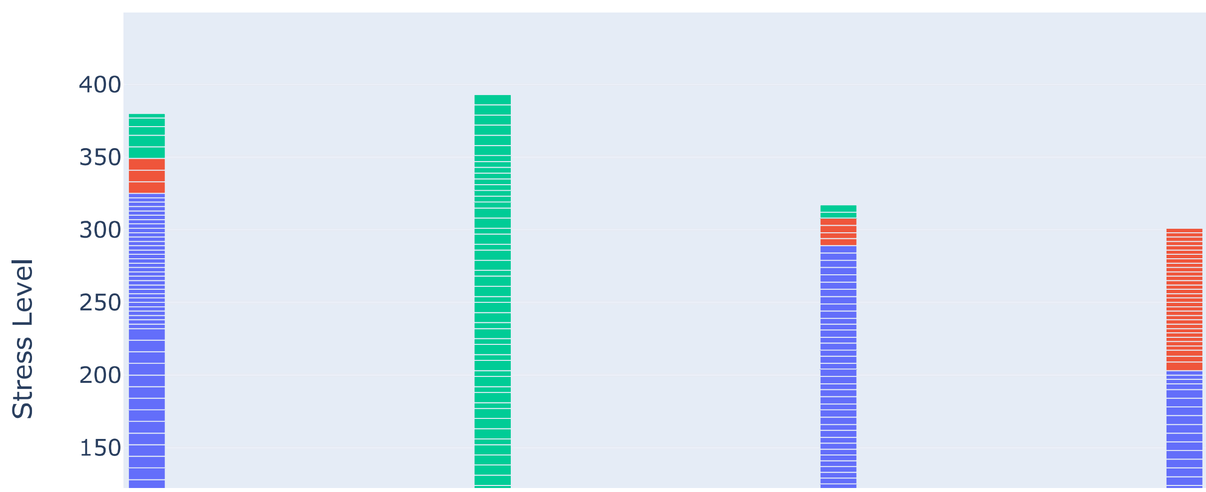
In [151]:
```
fig = px.histogram(data, x='Age', color='Sleep Disorder',
                   title='Age Distribution with Sleep Disorder',
                   labels={'Age': 'Age', 'count': 'Count'},
                   template='plotly')
fig.show()
```



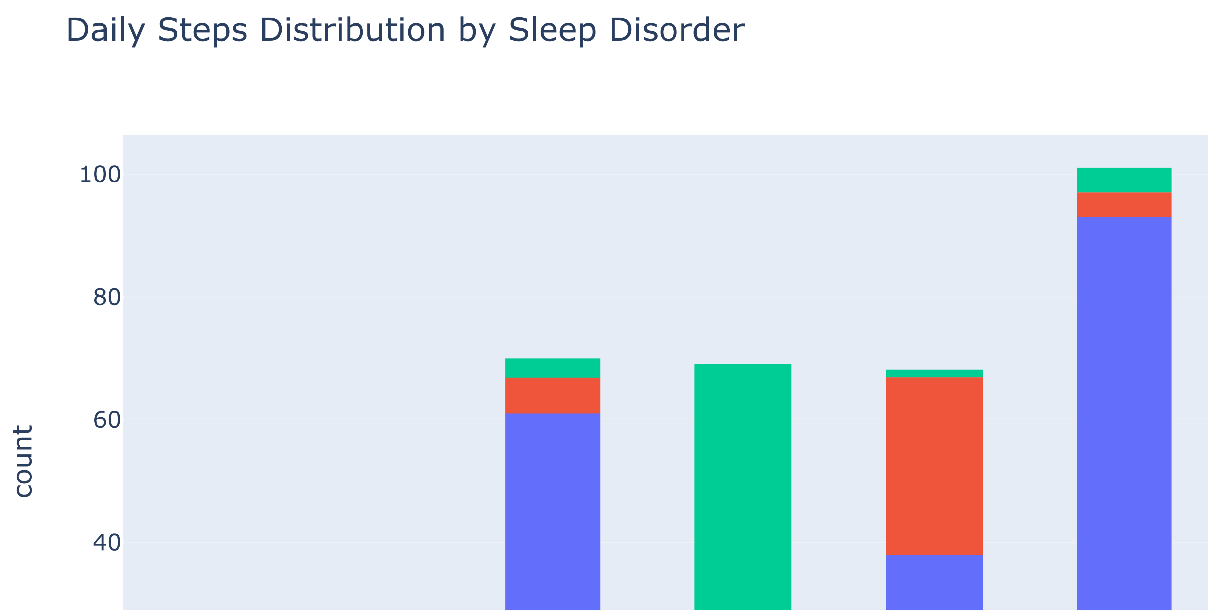
In [152]:
```
fig = px.scatter(data, x='Sleep Duration', y='Quality of Sleep', color='Sleep Disorder
                 title='Sleep Duration vs. Quality of Sleep',
                 labels={'Sleep Duration': 'Sleep Duration', 'Quality of Sleep': 'Qual
                 template='plotly')
fig.show()
```

In [153]:
```
fig = px.bar(data, x='Physical Activity Level', y='Stress Level', color='Sleep Disorde
                  title='Stress Level vs. Physical Activity Level',
                  labels={'Physical Activity Level': 'Physical Activity Level', 'Stress
fig.show()
```

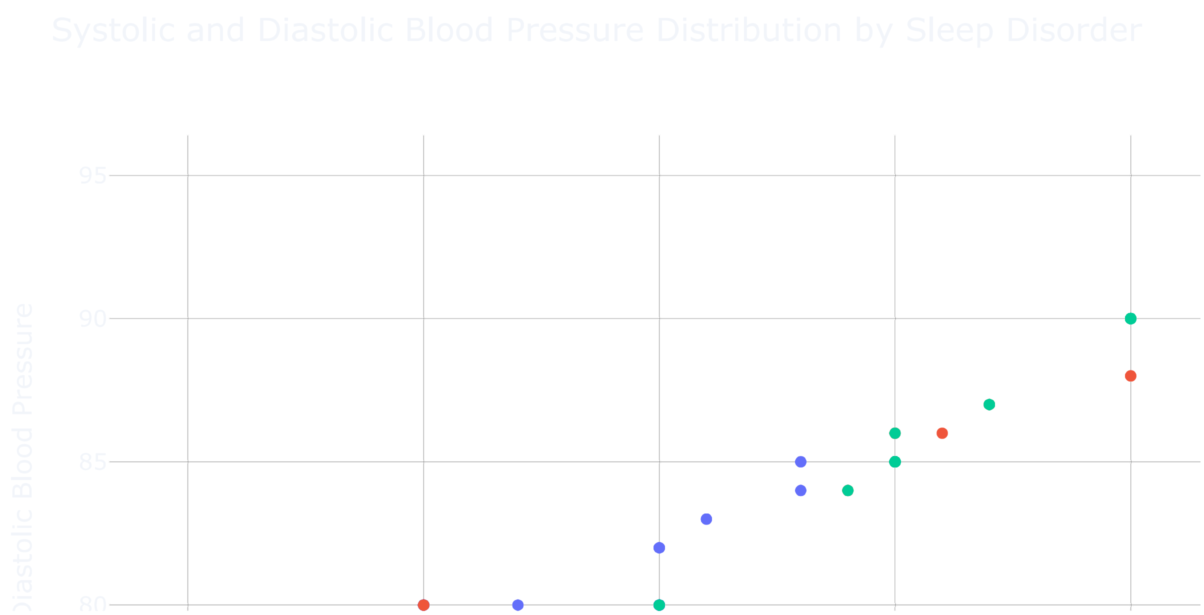
Stress Level vs. Physical Activity Level



In [154]:
```
fig = px.histogram(data, x='Daily Steps', color='Sleep Disorder',
                   title='Daily Steps Distribution by Sleep Disorder',
                   labels={'Daily Steps': 'Daily Steps', 'count': 'Count'},
                   template='plotly')
fig.show()
```



In [155]:
```
fig = px.scatter(data, x='SYSTOLIC', y='DIASTOLIC', color='Sleep Disorder',
                 title='Systolic and Diastolic Blood Pressure Distribution by Sleep Di
                 labels={'SYSTOLIC': 'Systolic Blood Pressure', 'DIASTOLIC': 'Diastoli
                 template='plotly_dark')
fig.show()
```

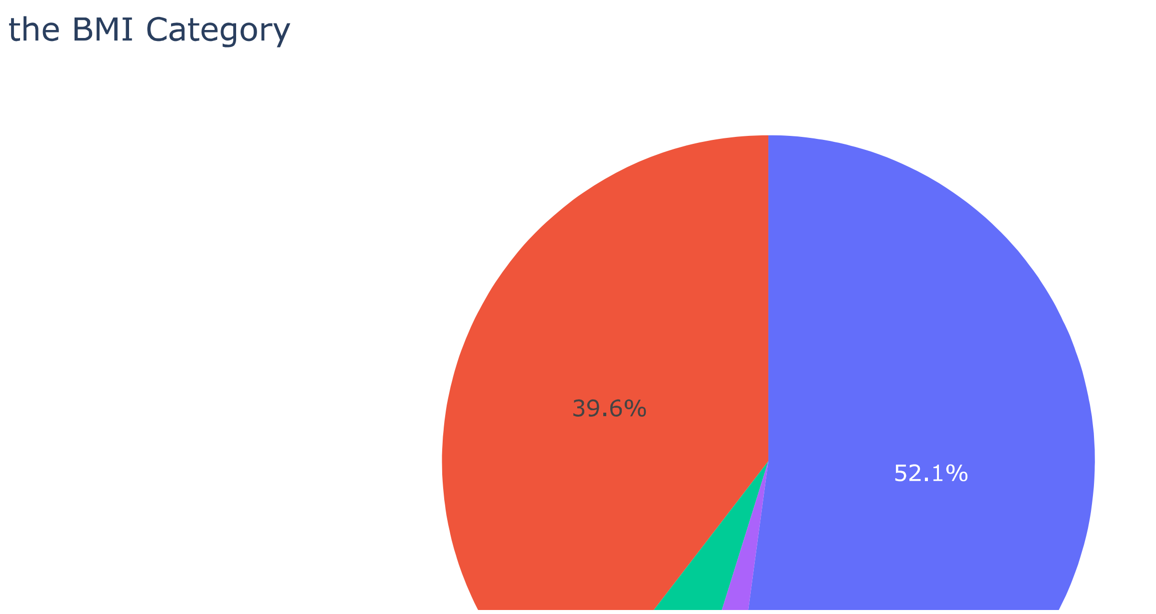


In [156]:
```
BMI_Category_count=data['BMI Category'].value_counts().reset_index()
BMI_Category_count
```

Out[156]:

	BMI Category	count
0	Normal	195
1	Overweight	148
2	Normal Weight	21
3	Obese	10

In [157]:
```
fig=px.pie(BMI_Category_count,values='count',names='BMI Category',title="the BMI Categ
fig.show()
```

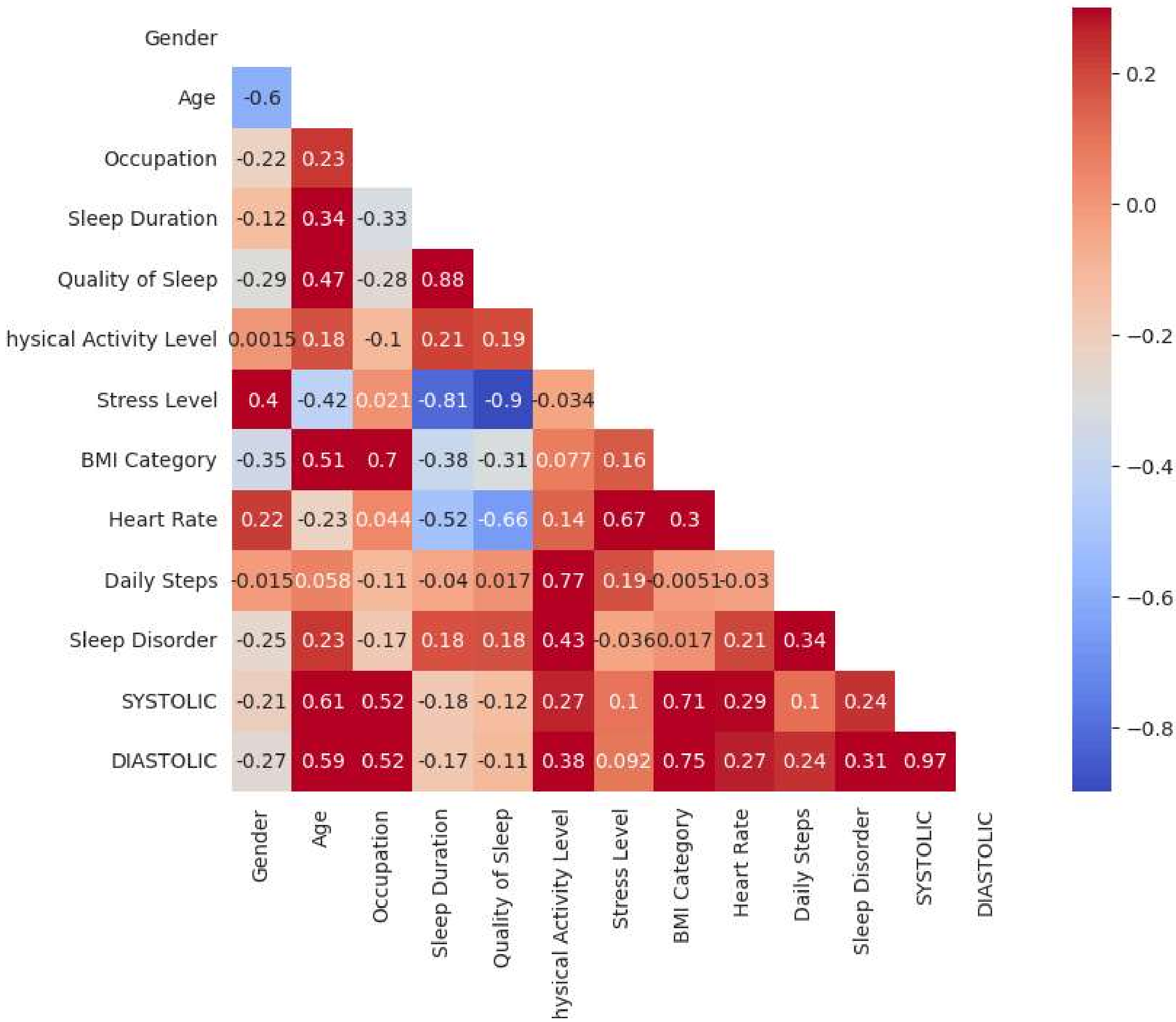


In [158]:
```
label_encoder = preprocessing.LabelEncoder()
data['Gender'] = label_encoder.fit_transform(data['Gender'])
data['Occupation'] = label_encoder.fit_transform(data['Occupation'])
data['BMI Category'] = label_encoder.fit_transform(data['BMI Category'])
data['Sleep Disorder'] = label_encoder.fit_transform(data['Sleep Disorder'])
df.head()
```

Out[158]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Blood Pressure	Heart Rate
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	126/83	77
1	2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
2	3	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	140/90	85

In [159]:
```
def corr_vis(corr) :
    mask = np.zeros_like(corr)
    mask[np.triu_indices_from(mask)] = True
    with sns.axes_style("white"):
        f, ax = plt.subplots(figsize=(10, 7))
        g = sns.heatmap(corr, mask=mask, vmax=.3, square=True, annot=True, cmap='coolw
        g.set_xticklabels(g.get_xticklabels(), rotation = 90, fontsize = 10)

num_corr = data.corr()
corr_vis(data.corr())
```



In [160]:
```python
label_encoders = {}
cat_columns = ['Occupation', 'BMI Category', 'Sleep Disorder', 'Gender']

for col in cat_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

# # Save label encoders
# for col, le in label_encoders.items():
#     joblib.dump(le, f'{col}_label_encoder.pkl')

data.head()
```

Out[160]:

	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Heart Rate	Daily Steps	Sleep Disorder	SYST(
0	1	27	9	6.1	6	42	6	3	77	4200	1	1
1	1	28	1	6.2	6	60	8	0	75	10000	1	1
2	1	28	1	6.2	6	60	8	0	75	10000	1	1
3	1	28	6	5.9	4	30	8	2	85	3000	2	1
4	1	28	6	5.9	4	30	8	2	85	3000	2	1

In [161]:
```python
X = data.drop(['Sleep Disorder'], axis=1)
y = data['Sleep Disorder']

scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)
scaled_features
```

Out[161]:
```
array([[ 0.9893614 , -1.75309569,  1.7127411 , ..., -1.61958404,
        -0.33000229, -0.26810236],
       [ 0.9893614 , -1.63764266, -0.90849745, ...,  1.97007745,
        -0.45923879, -0.7556402 ],
       [ 0.9893614 , -1.63764266, -0.90849745, ...,  1.97007745,
        -0.45923879, -0.7556402 ],
       ...,
       [-1.010753  ,  1.94140144,  0.40212182, ...,  0.11335599,
         1.47930869,  1.68204901],
       [-1.010753  ,  1.94140144,  0.40212182, ...,  0.11335599,
         1.47930869,  1.68204901],
       [-1.010753  ,  1.94140144,  0.40212182, ...,  0.11335599,
         1.47930869,  1.68204901]])
```

In [162]:
```
X_train, X_test, y_train, y_test = train_test_split(scaled_features, y, test_size=0.3,
# Classification algorithms
classifiers = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Support Vector Machine': SVC(),
    'Naive Bayes': GaussianNB(),
    'K-Nearest Neighbours': KNeighborsClassifier()
}
```

In [163]:
```
results = {}
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix for", name ,": \n",cm)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy
    print(f'{name} Accuracy: {accuracy *100:.2f} %')
    print(classification_report(y_test, y_pred))
    print('...........................................................................
```

```
Confusion Matrix for Logistic Regression :
 [[18  2  3]
 [ 4 61  1]
 [ 1  3 20]]
Logistic Regression Accuracy: 87.61 %
              precision    recall  f1-score   support

           0       0.78      0.78      0.78        23
           1       0.92      0.92      0.92        66
           2       0.83      0.83      0.83        24

    accuracy                           0.88       113
   macro avg       0.85      0.85      0.85       113
weighted avg       0.88      0.88      0.88       113

.....................................................................................
.........................
Confusion Matrix for Decision Tree :
 [[19  2  2]
 [ 3 60  3]
 [ 1  3 20]]
Decision Tree Accuracy: 87.61 %
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        23
           1       0.92      0.91      0.92        66
           2       0.80      0.83      0.82        24

    accuracy                           0.88       113
   macro avg       0.85      0.86      0.85       113
weighted avg       0.88      0.88      0.88       113

.....................................................................................
.........................
Confusion Matrix for Random Forest :
 [[19  2  2]
 [ 1 62  3]
 [ 1  3 20]]
Random Forest Accuracy: 89.38 %
              precision    recall  f1-score   support

           0       0.90      0.83      0.86        23
           1       0.93      0.94      0.93        66
           2       0.80      0.83      0.82        24

    accuracy                           0.89       113
   macro avg       0.88      0.87      0.87       113
weighted avg       0.89      0.89      0.89       113

.....................................................................................
.........................
Confusion Matrix for Support Vector Machine :
 [[18  2  3]
 [ 5 60  1]
 [ 1  3 20]]
Support Vector Machine Accuracy: 86.73 %
              precision    recall  f1-score   support

           0       0.75      0.78      0.77        23
           1       0.92      0.91      0.92        66
```

```
           2       0.83      0.83      0.83        24

    accuracy                           0.87       113
   macro avg       0.84      0.84      0.84       113
weighted avg       0.87      0.87      0.87       113

....................................................................................
........................
Confusion Matrix for Naive Bayes :
 [[19  2  2]
 [ 6 60  0]
 [ 2  3 19]]
Naive Bayes Accuracy: 86.73 %
              precision    recall  f1-score   support

           0       0.70      0.83      0.76        23
           1       0.92      0.91      0.92        66
           2       0.90      0.79      0.84        24

    accuracy                           0.87       113
   macro avg       0.84      0.84      0.84       113
weighted avg       0.87      0.87      0.87       113

....................................................................................
........................
Confusion Matrix for K-Nearest Neighbours :
 [[19  2  2]
 [ 5 59  2]
 [ 2  3 19]]
K-Nearest Neighbours Accuracy: 85.84 %
              precision    recall  f1-score   support

           0       0.73      0.83      0.78        23
           1       0.92      0.89      0.91        66
           2       0.83      0.79      0.81        24

    accuracy                           0.86       113
   macro avg       0.83      0.84      0.83       113
weighted avg       0.86      0.86      0.86       113

....................................................................................
........................
```

In [164]:

```python
best_classifier = max(results, key=results.get)
print(f'Best Classifier: {best_classifier} with Accuracy: {results[best_classifier]:.4
```

```
Best Classifier: Random Forest with Accuracy: 0.8938
```

In [ ]: