Predicting Sleep Disorders using Machine Learning Algorithms

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
```

In [2]:

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

In [3]:

```
df = pd.read_csv("sleep_disorder.csv")
```

In [4]:

```
df.head()
```

Out[4]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	E Pres
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	1
1	2	Male	28	Doctor	6.2	6	60	8	Normal	1
2	3	Male	28	Doctor	6.2	6	60	8	Normal	1
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	1
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	1

In [5]:

```
df.tail()
```

Out[5]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	E Pres
369	370	Female	59	Nurse	8.1	9	75	3	Overweight	1
370	371	Female	59	Nurse	8.0	9	75	3	Overweight	1
371	372	Female	59	Nurse	8.1	9	75	3	Overweight	1
372	373	Female	59	Nurse	8.1	9	75	3	Overweight	1
373	374	Female	59	Nurse	8.1	9	75	3	Overweight	1

In [6]:

```
df.shape
```

Out[6]:

```
(374, 13)
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['Person ID', 'Gender', 'Age', 'Occupation', 'Sleep Duration',
       'Quality of Sleep', 'Physical Activity Level', 'Stress Level',
       'BMI Category', 'Blood Pressure', 'Heart Rate', 'Daily Steps',
       'Sleep Disorder'],
      dtype='object')
```

In [8]:

```
df.duplicated().sum()
```

Out[8]:

```
0
```

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
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            0
dtype: int64
```

In [10]:

```
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           374 non-null    object 
dtypes: float64(1), int64(7), object(5)
memory usage: 38.1+ KB
```

In [11]:

```
df.describe()
```

Out[11]:

	Person ID	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate
count	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
std	108.108742	8.673133	0.795657	1.196956	20.830804	1.774526	4.135676
min	1.000000	27.000000	5.800000	4.000000	30.000000	3.000000	65.000000
25%	94.250000	35.250000	6.400000	6.000000	45.000000	4.000000	68.000000
50%	187.500000	43.000000	7.200000	7.000000	60.000000	5.000000	70.000000
75%	280.750000	50.000000	7.800000	8.000000	75.000000	7.000000	72.000000
max	374.000000	59.000000	8.500000	9.000000	90.000000	8.000000	86.000000

In [12]:

```
df.nunique()
```

Out[12]:

```
Person ID                  374
Gender                       2
Age                         31
Occupation                  11
Sleep Duration              27
Quality of Sleep             6
Physical Activity Level     16
Stress Level                 6
BMI Category                 4
Blood Pressure              25
Heart Rate                  19
Daily Steps                 20
Sleep Disorder               3
dtype: int64
```

In [13]:

```
# Check numerical data types
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns

print("Numerical Columns:")
for column in numerical_columns:
    unique_values = df[column].unique()
    print(f"{column}: {unique_values}")

# Check object data types
object_columns = df.select_dtypes(include=['object']).columns

print("\nObject Columns:")
for column in object_columns:
    unique_values = df[column].unique()
    print(f"{column}: {unique_values}")
```

```
Numerical Columns:
Person ID: [  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  1
6  17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270
 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324
 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342
 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
 361 362 363 364 365 366 367 368 369 370 371 372 373 374]
Age: [27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 48 49 50 51
52
 53 54 55 56 57 58 59]
Sleep Duration: [6.1 6.2 5.9 6.3 7.8 6.  6.5 7.6 7.7 7.9 6.4 7.5 7.2 5.8
6.7 7.3 7.4 7.1
 6.6 6.9 8.  6.8 8.1 8.3 8.5 8.4 8.2]
Quality of Sleep: [6 4 7 5 8 9]
Physical Activity Level: [42 60 30 40 75 35 45 50 32 70 80 55 90 47 65 85]
Stress Level: [6 8 7 4 3 5]
Heart Rate: [77 75 85 82 70 80 78 69 72 68 76 81 65 84 74 67 73 83 86]
Daily Steps: [ 4200 10000  3000  3500  8000  4000  4100  6800  5000  7000
5500  5200
  5600  3300  4800  7500  7300  6200  6000  3700]

Object Columns:
Gender: ['Male' 'Female']
Occupation: ['Software Engineer' 'Doctor' 'Sales Representative' 'Teacher'
'Nurse'
 'Engineer' 'Accountant' 'Scientist' 'Lawyer' 'Salesperson' 'Manager']
BMI Category: ['Overweight' 'Normal' 'Obese' 'Normal Weight']
Blood Pressure: ['126/83' '125/80' '140/90' '120/80' '132/87' '130/86' '11
7/76' '118/76'
 '128/85' '131/86' '128/84' '115/75' '135/88' '129/84' '130/85' '115/78'
 '119/77' '121/79' '125/82' '135/90' '122/80' '142/92' '140/95' '139/91'
 '118/75']
Sleep Disorder: ['None' 'Sleep Apnea' 'Insomnia']
```

In [14]:

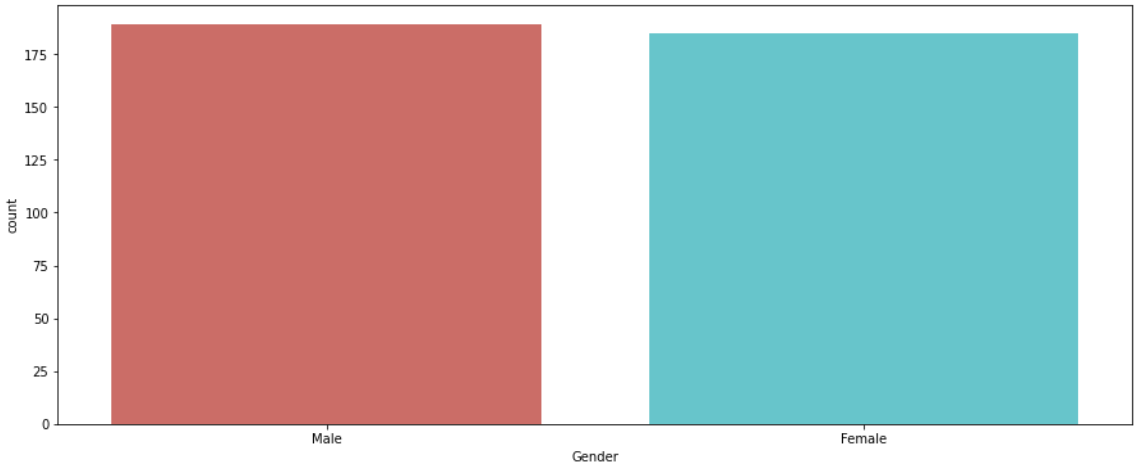
```
df['Gender'].value_counts()
```

Out[14]:

```
Male      189
Female    185
Name: Gender, dtype: int64
```

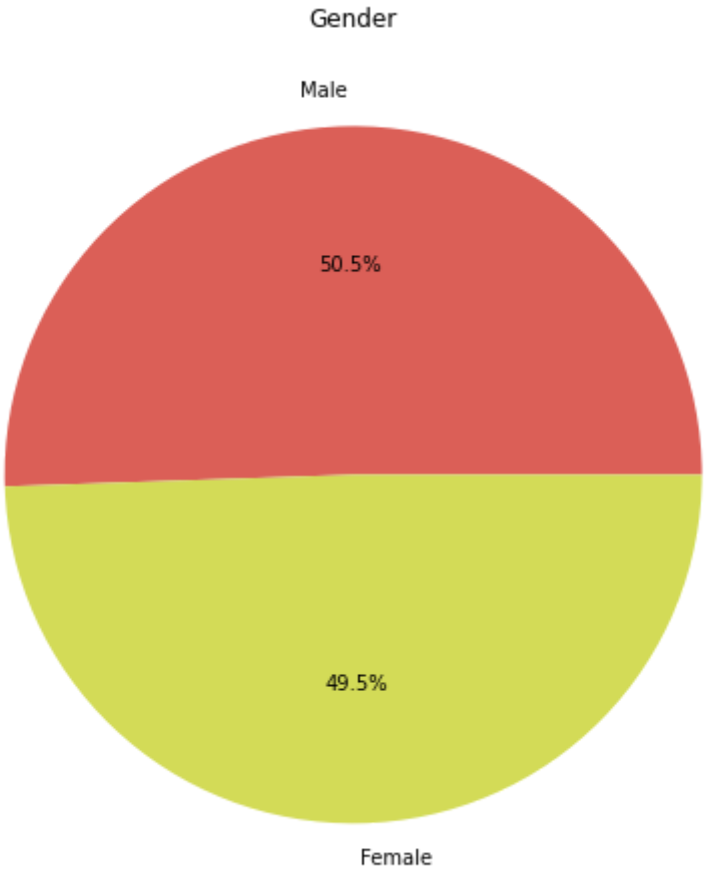
In [15]:

```
plt.figure(figsize=(15,6))
sns.countplot(df['Gender'], data = df, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



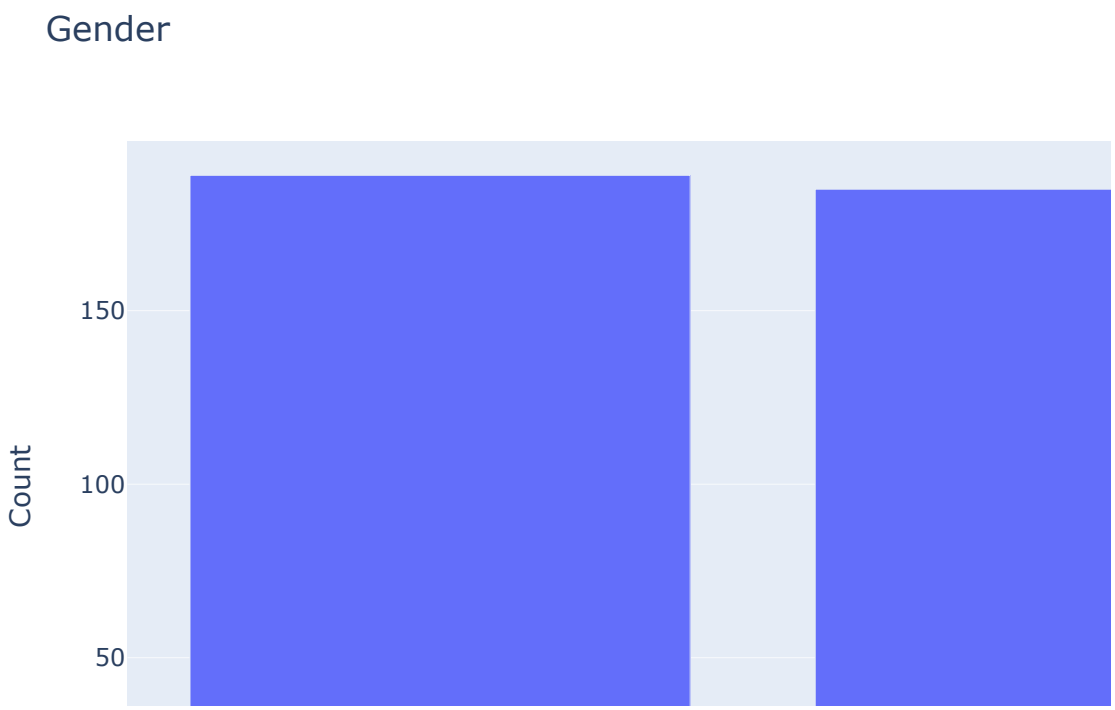
In [16]:

```
plt.figure(figsize=(20, 8))
counts = df['Gender'].value_counts()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', colors=sns.color_palette('hls'))
plt.title('Gender')
plt.show()
```



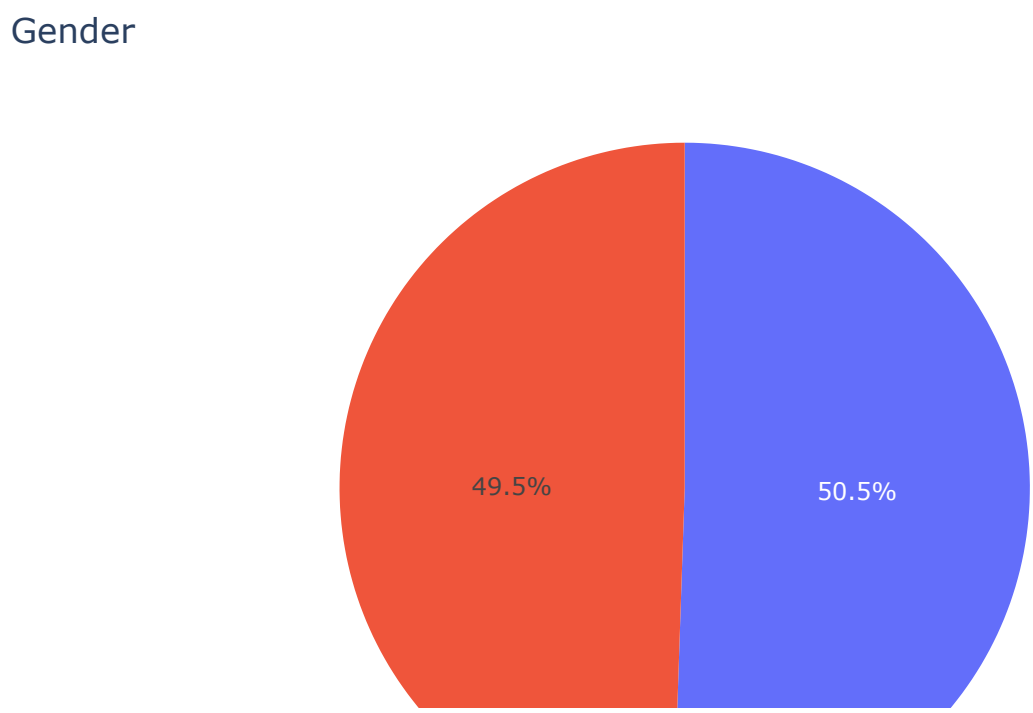
In [17]:

```
fig = go.Figure(data=[go.Bar(x=df['Gender'].value_counts().index, y=df['Gender'].value_c
fig.update_layout(
        title= 'Gender',
        xaxis_title="Gender",
        yaxis_title="Count"
    )
fig.show()
```



In [18]:

```
counts = df['Gender'].value_counts()
fig = go.Figure(data=[go.Pie(labels=counts.index, values=counts)])
fig.update_layout(title= 'Gender')
fig.show()
```



In [19]:

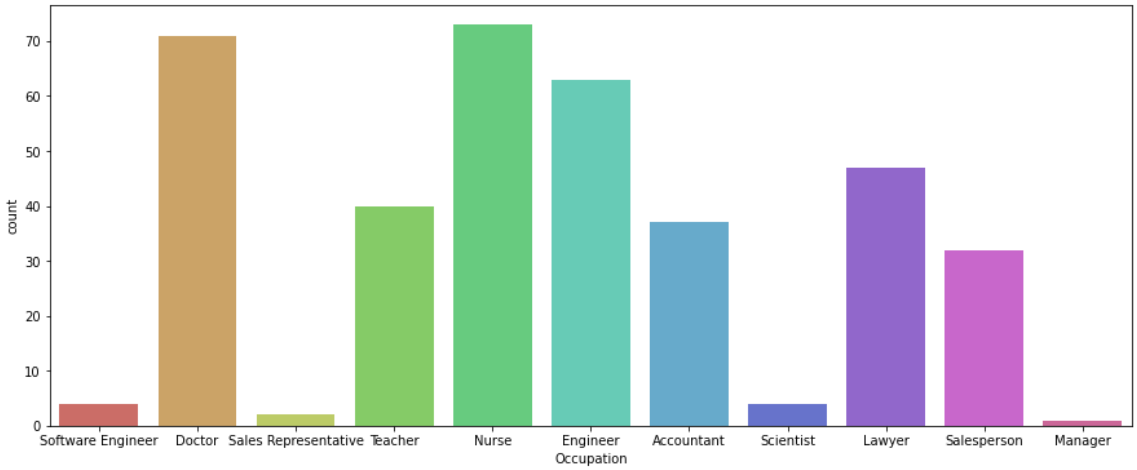
```
df['Occupation'].value_counts()
```

Out[19]:

```
Nurse                    73
Doctor                   71
Engineer                 63
Lawyer                   47
Teacher                  40
Accountant               37
Salesperson              32
Software Engineer         4
Scientist                 4
Sales Representative      2
Manager                   1
Name: Occupation, dtype: int64
```

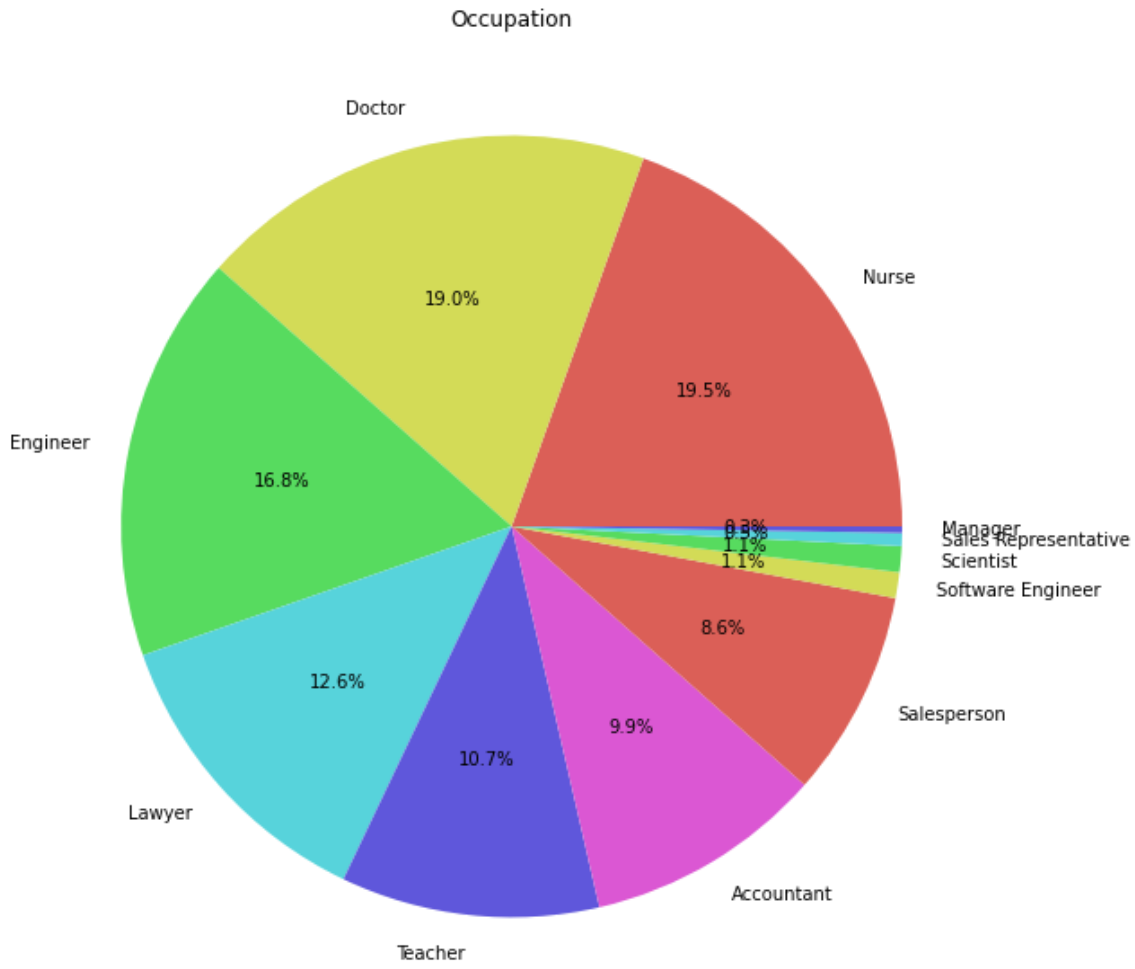
In [20]:

```
plt.figure(figsize=(15,6))
sns.countplot(df['Occupation'], data = df, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



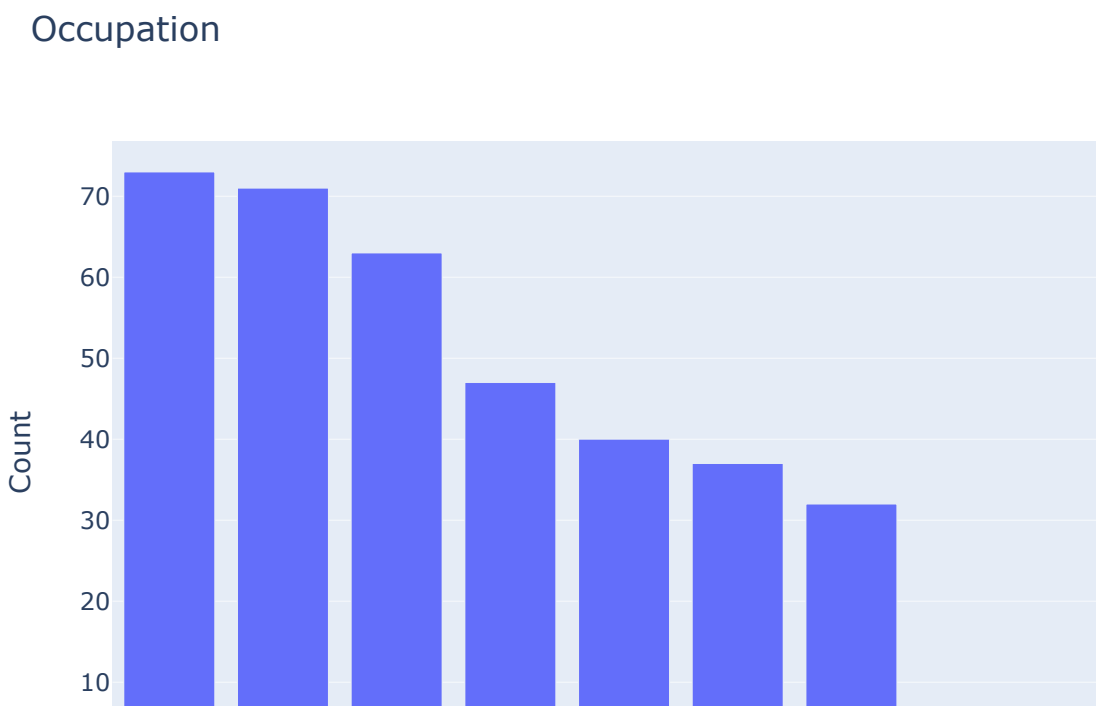
In [21]:

```
plt.figure(figsize=(20, 10))
counts = df['Occupation'].value_counts()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', colors=sns.color_palette('hls'))
plt.title('Occupation')
plt.show()
```



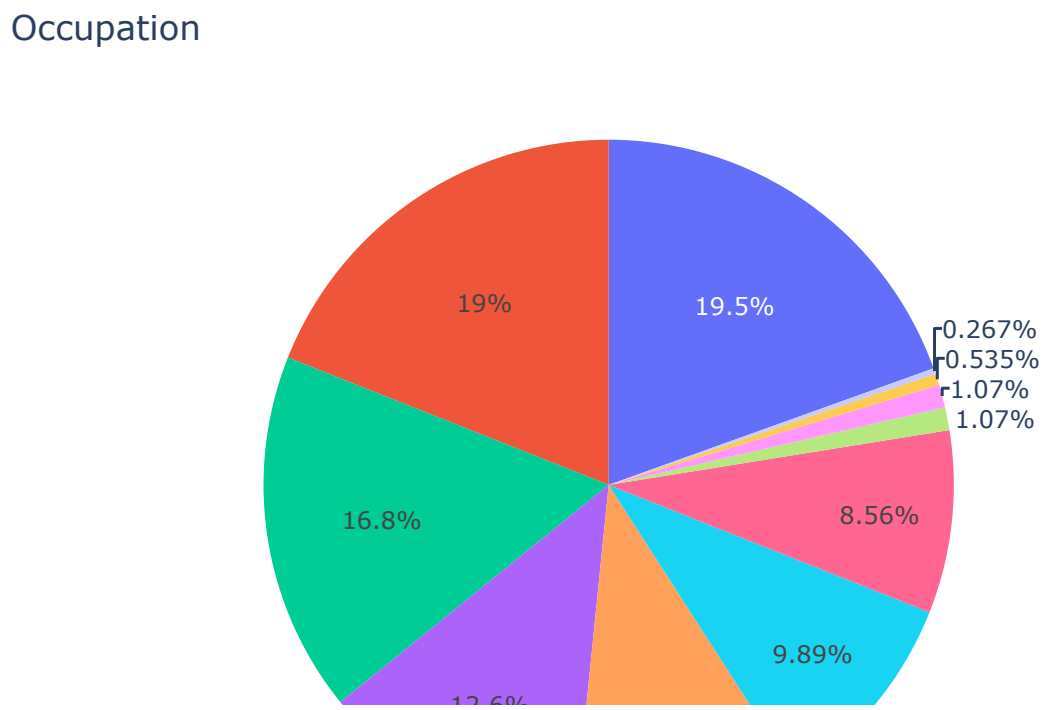
In [22]:

```
fig = go.Figure(data=[go.Bar(x=df['Occupation'].value_counts().index, y=df['Occupation']
fig.update_layout(
        title= 'Occupation',
        xaxis_title="Occupation",
        yaxis_title="Count"
    )
fig.show()
```



In [23]:

```
counts = df['Occupation'].value_counts()
fig = go.Figure(data=[go.Pie(labels=counts.index, values=counts)])
fig.update_layout(title= 'Occupation')
fig.show()
```



In [24]:

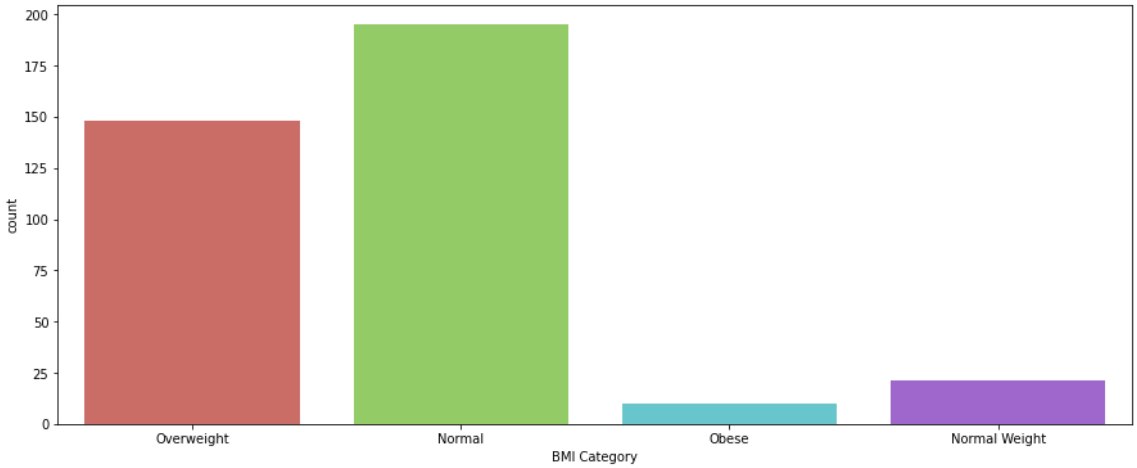
```
df['BMI Category'].value_counts()
```

Out[24]:

```
Normal           195
Overweight       148
Normal Weight     21
Obese             10
Name: BMI Category, dtype: int64
```

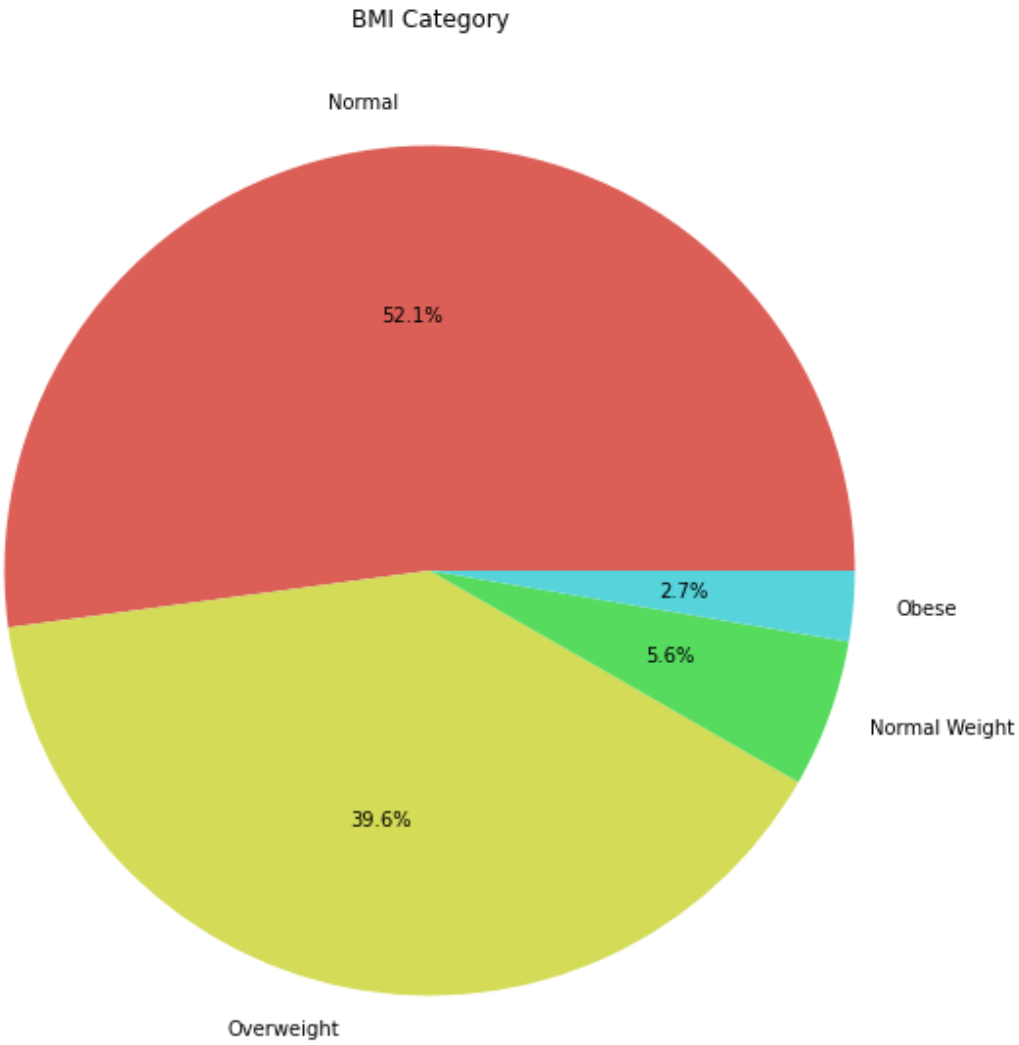
In [25]:

```
plt.figure(figsize=(15,6))
sns.countplot(df['BMI Category'], data = df, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



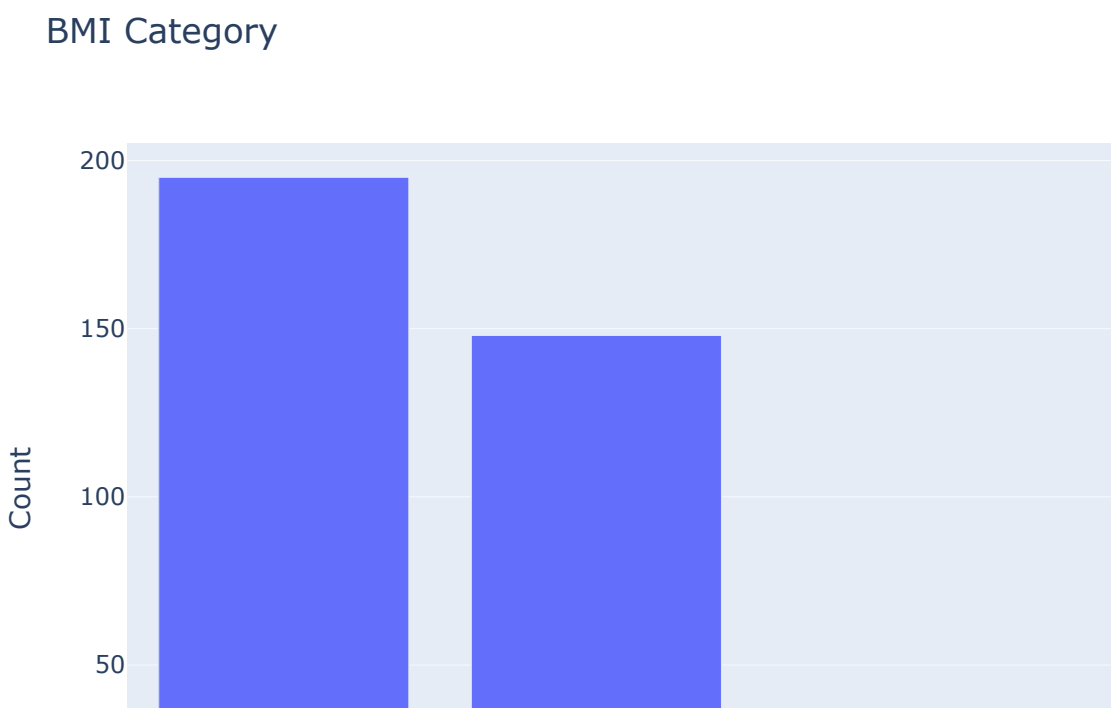
In [26]:

```
plt.figure(figsize=(20, 10))
counts = df['BMI Category'].value_counts()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', colors=sns.color_palette('hls'))
plt.title('BMI Category')
plt.show()
```



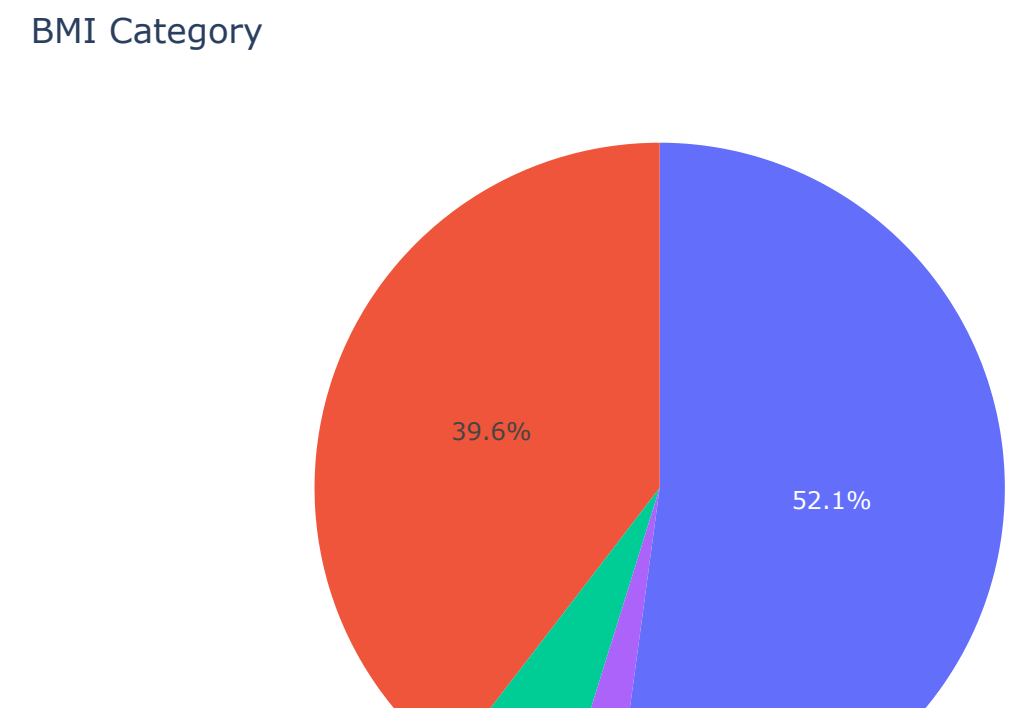
In [27]:

```
fig = go.Figure(data=[go.Bar(x=df['BMI Category'].value_counts().index, y=df['BMI Catego
fig.update_layout(
        title= 'BMI Category',
        xaxis_title="BMI Category",
        yaxis_title="Count"
    )
fig.show()
```



In [28]:

```
counts = df['BMI Category'].value_counts()
fig = go.Figure(data=[go.Pie(labels=counts.index, values=counts)])
fig.update_layout(title= 'BMI Category')
fig.show()
```



In [29]:

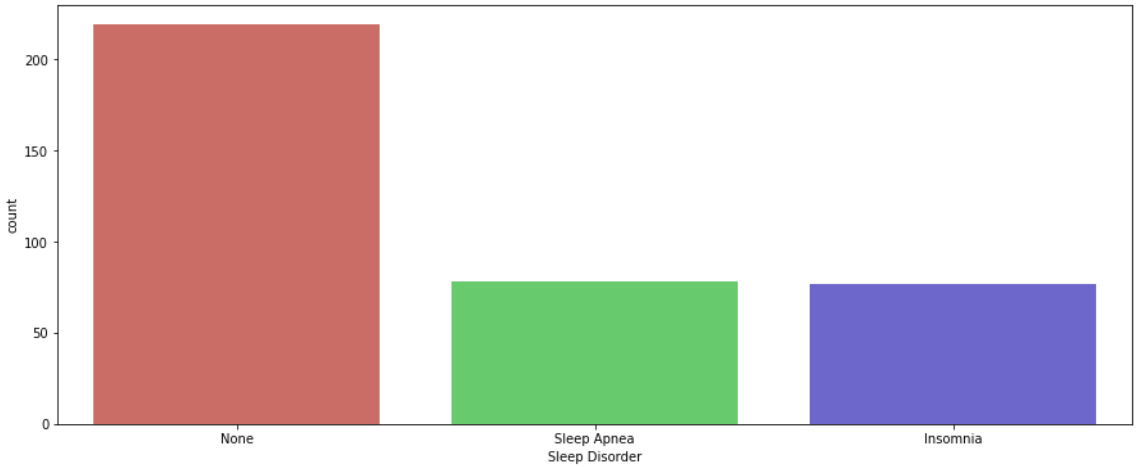
```
df['Sleep Disorder'].value_counts()
```

Out[29]:

```
None           219
Sleep Apnea     78
Insomnia        77
Name: Sleep Disorder, dtype: int64
```

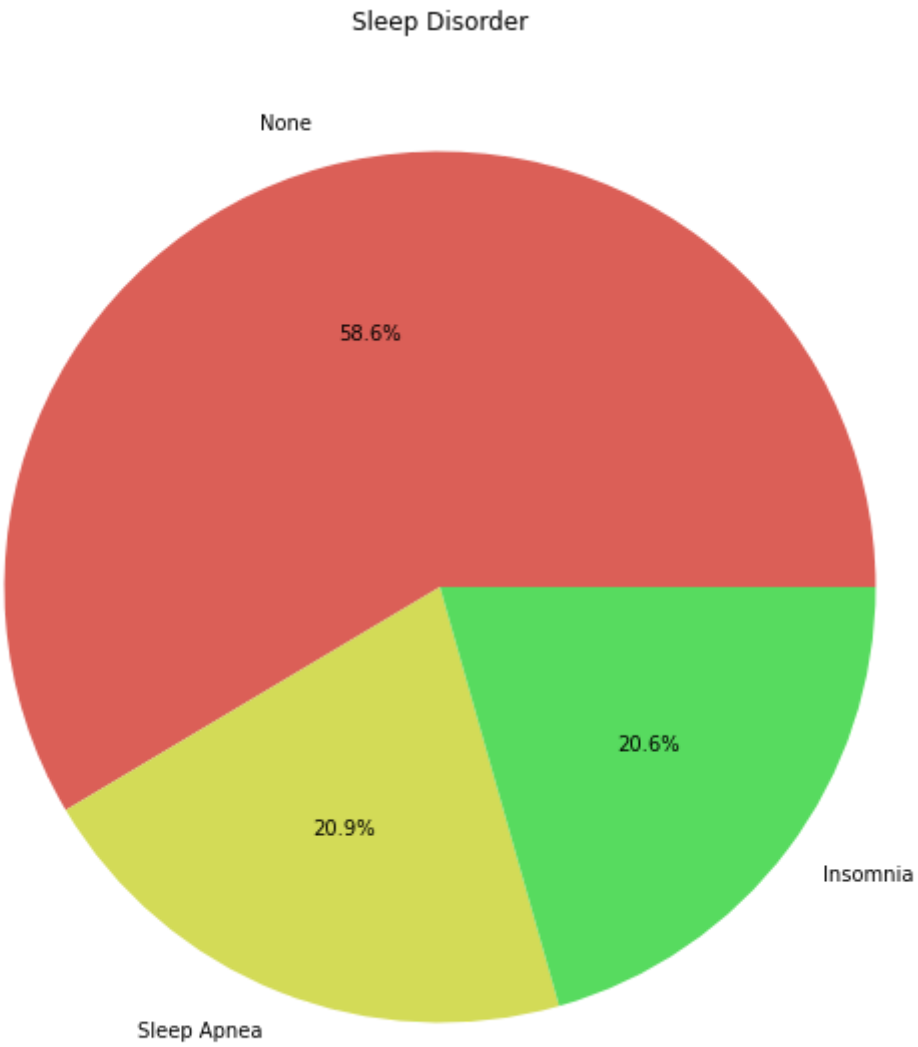
In [30]:

```
plt.figure(figsize=(15,6))
sns.countplot(df['Sleep Disorder'], data = df, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



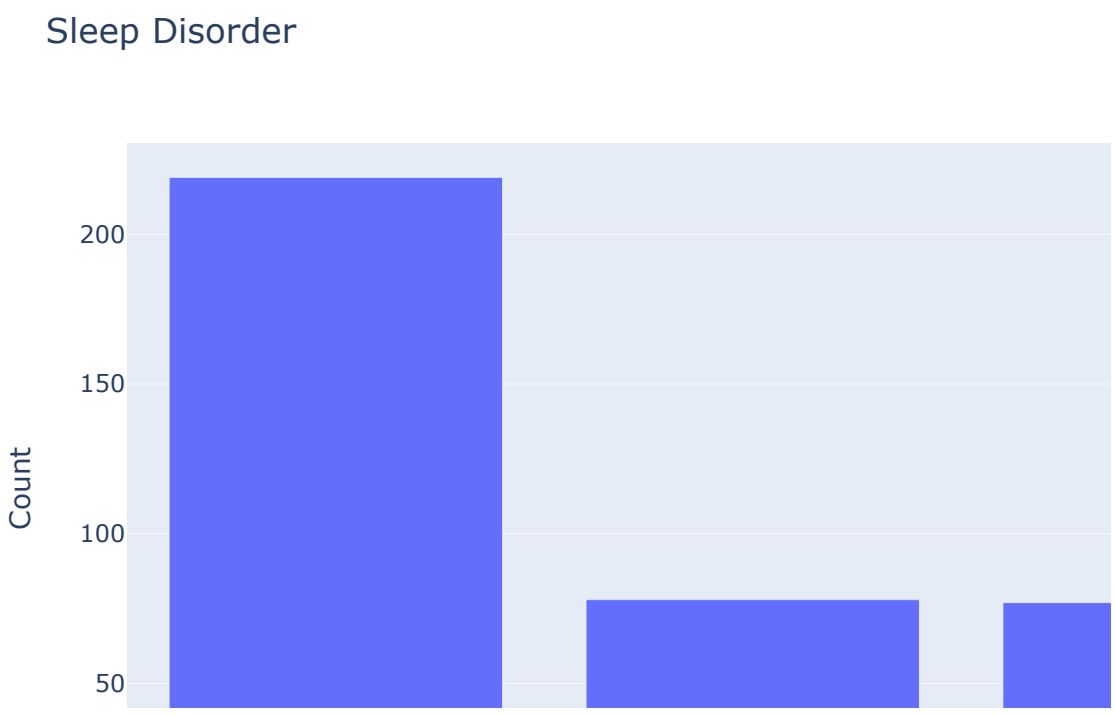
In [31]:

```
plt.figure(figsize=(20, 10))
counts = df['Sleep Disorder'].value_counts()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', colors=sns.color_palette('hls'))
plt.title('Sleep Disorder')
plt.show()
```



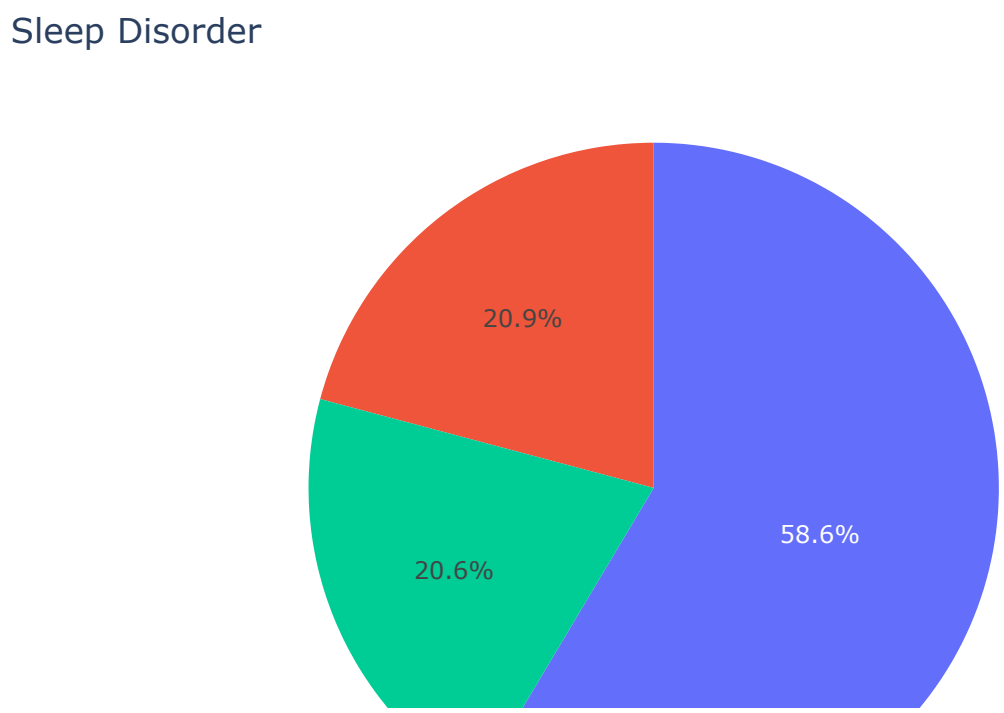
In [32]:

```
fig = go.Figure(data=[go.Bar(x=df['Sleep Disorder'].value_counts().index, y=df['Sleep Di
fig.update_layout(
        title= 'Sleep Disorder',
        xaxis_title="Sleep Disorder",
        yaxis_title="Count"
    )
fig.show()
```



In [33]:

```
counts = df['Sleep Disorder'].value_counts()
fig = go.Figure(data=[go.Pie(labels=counts.index, values=counts)])
fig.update_layout(title= 'Sleep Disorder')
fig.show()
```

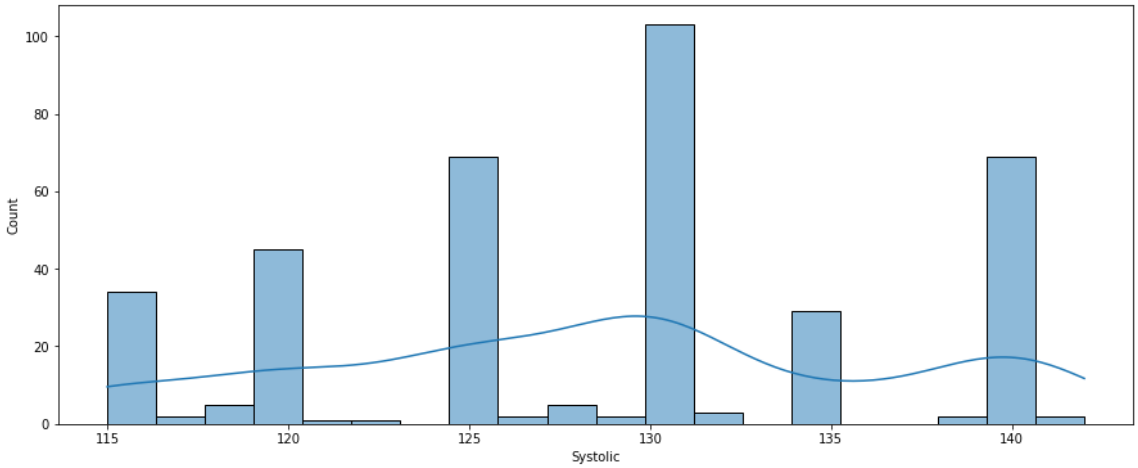


In [34]:

```
# Split the blood pressure values into systolic and diastolic columns
df[['Systolic', 'Diastolic']] = df['Blood Pressure'].str.split('/', expand=True).astype(
```

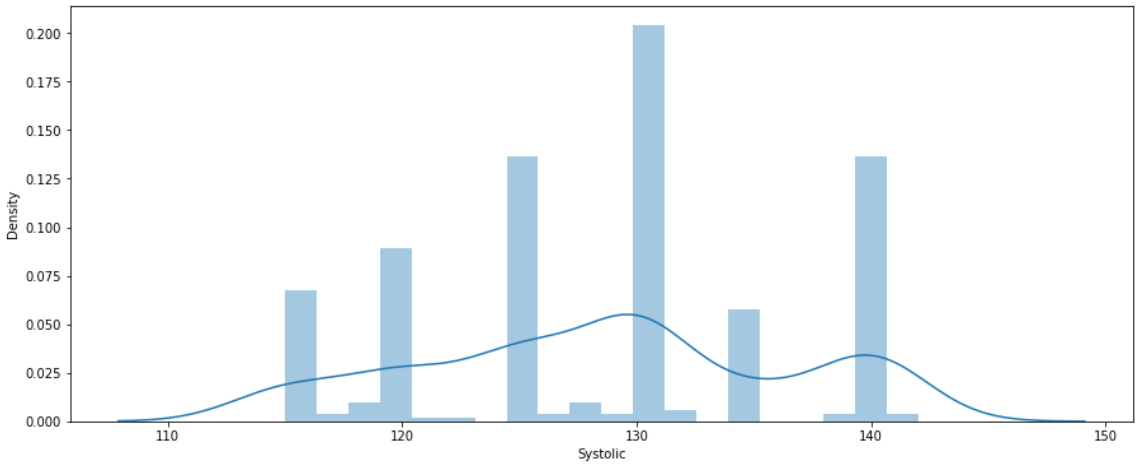
In [35]:

```
plt.figure(figsize=(15,6))
sns.histplot(df['Systolic'], kde = True, bins = 20, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



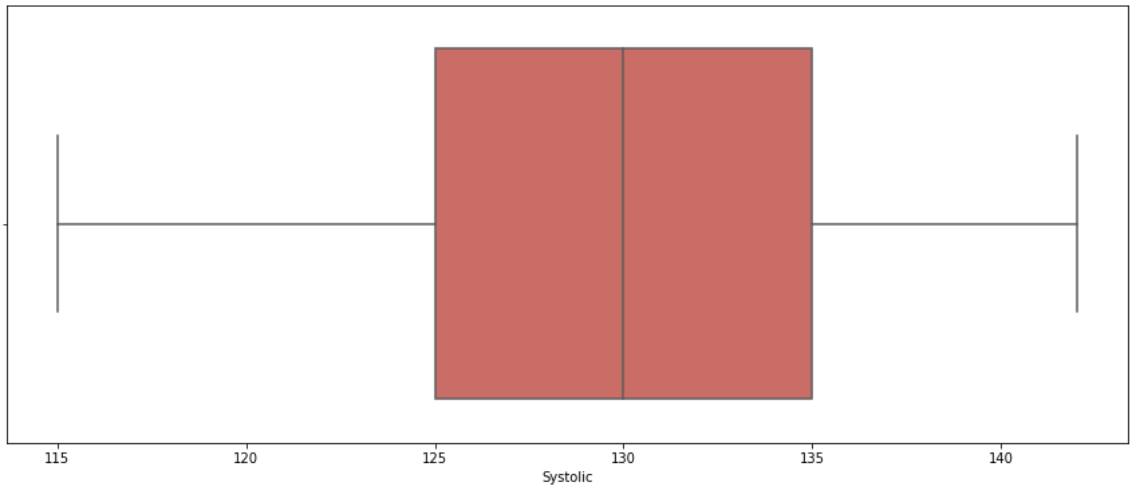
In [36]:

```
plt.figure(figsize=(15,6))
sns.distplot(df['Systolic'], kde = True, bins = 20)
plt.xticks(rotation = 0)
plt.show()
```



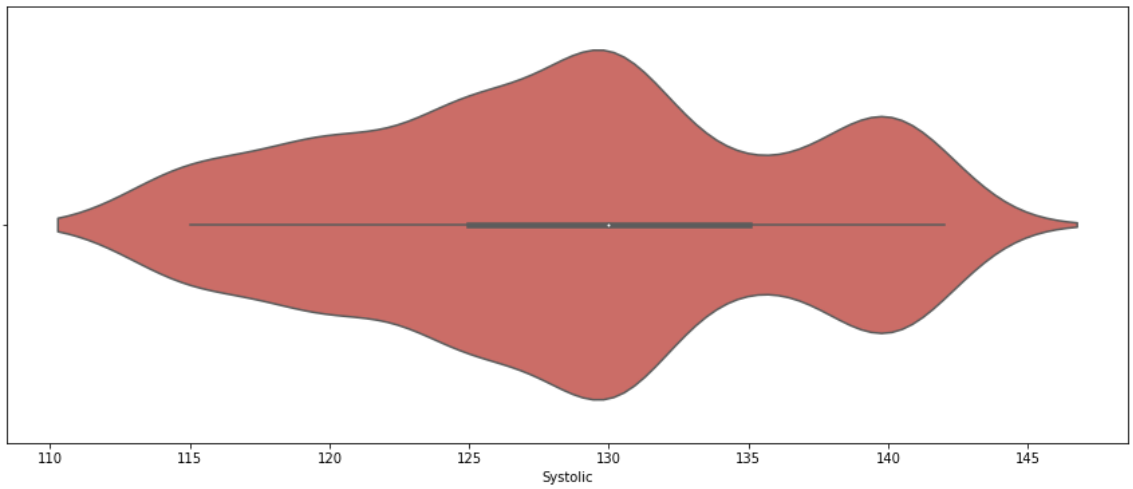
In [37]:

```
plt.figure(figsize=(15,6))
sns.boxplot(df['Systolic'], data=df, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



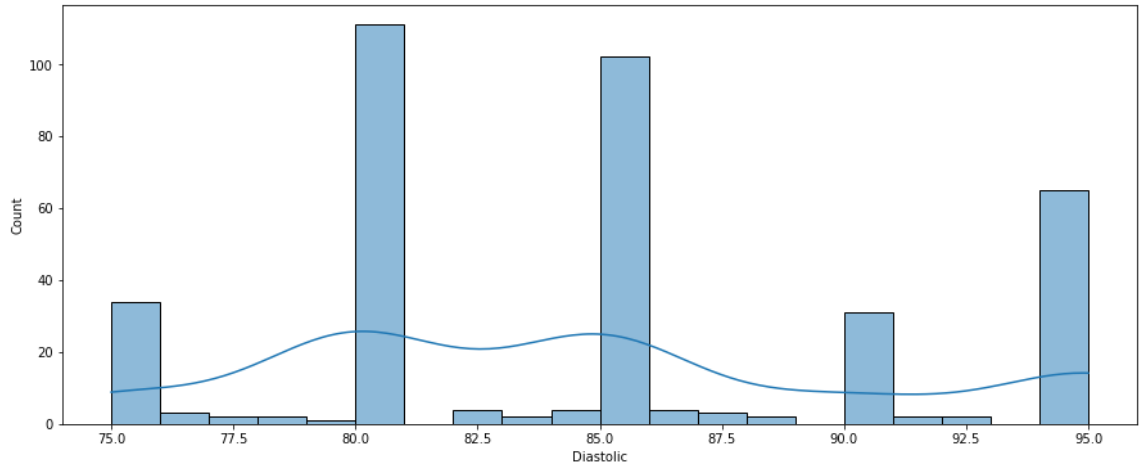
In [38]:

```
plt.figure(figsize=(15,6))
sns.violinplot(df['Systolic'], data=df, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



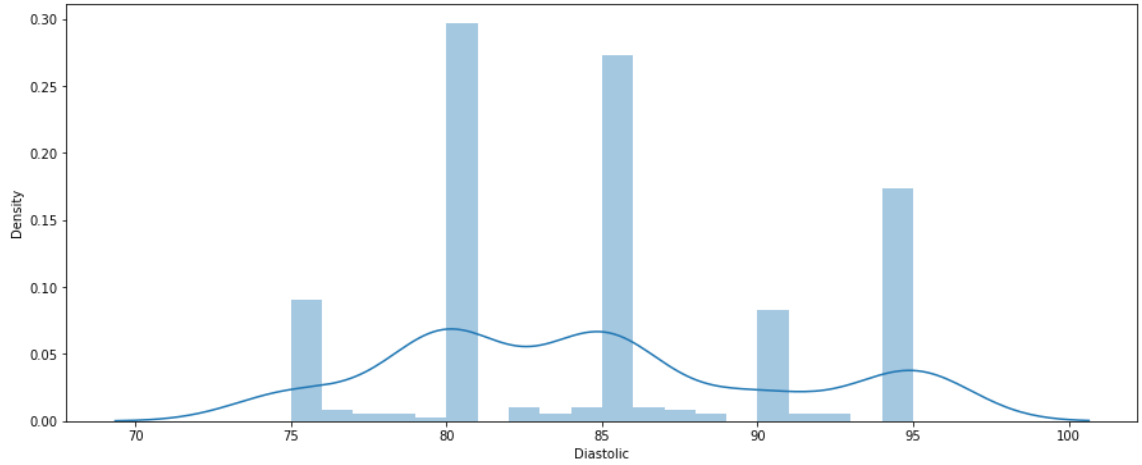
In [39]:

```
plt.figure(figsize=(15,6))
sns.histplot(df['Diastolic'], kde = True, bins = 20, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



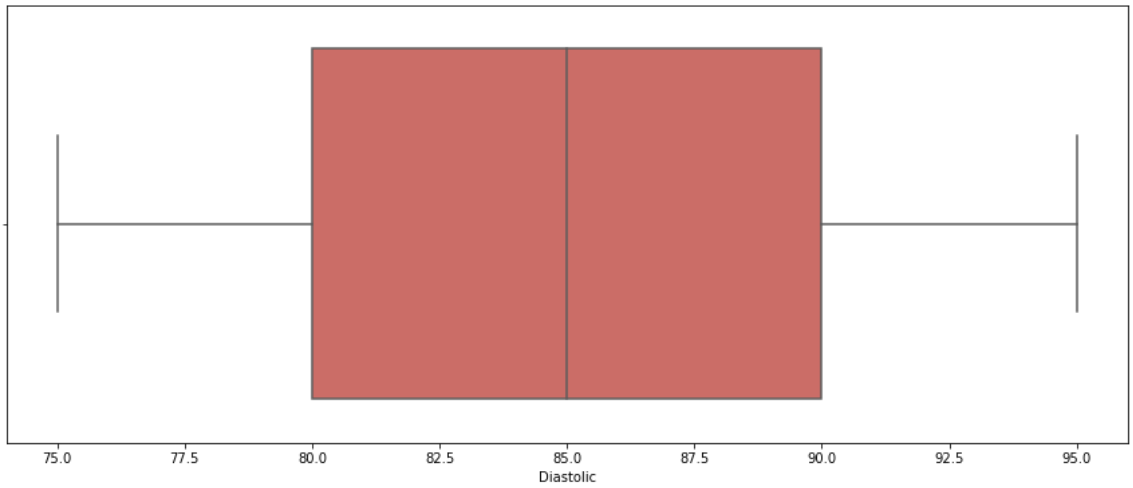
In [40]:

```
plt.figure(figsize=(15,6))
sns.distplot(df['Diastolic'], kde = True, bins = 20)
plt.xticks(rotation = 0)
plt.show()
```



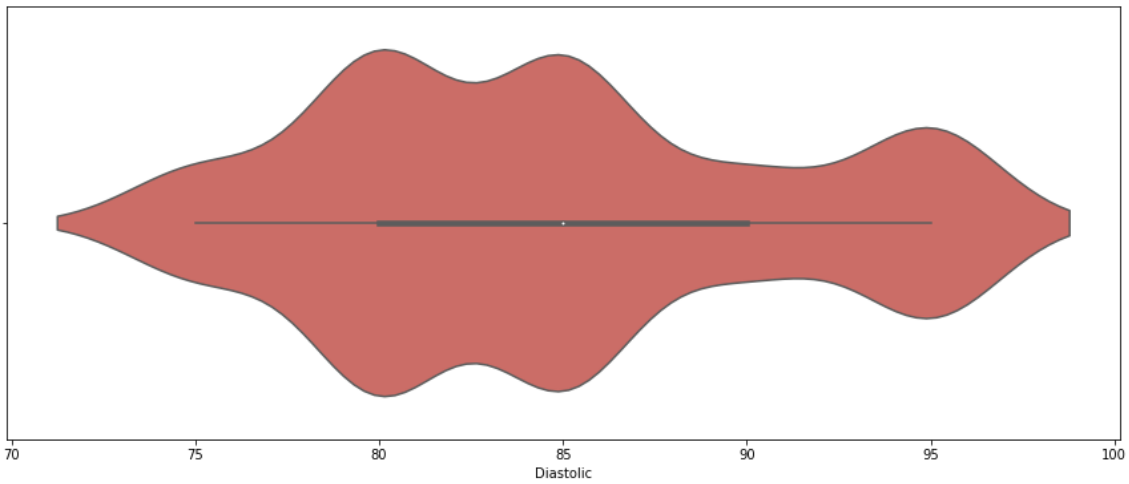
In [41]:

```
plt.figure(figsize=(15,6))
sns.boxplot(df['Diastolic'], data=df, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



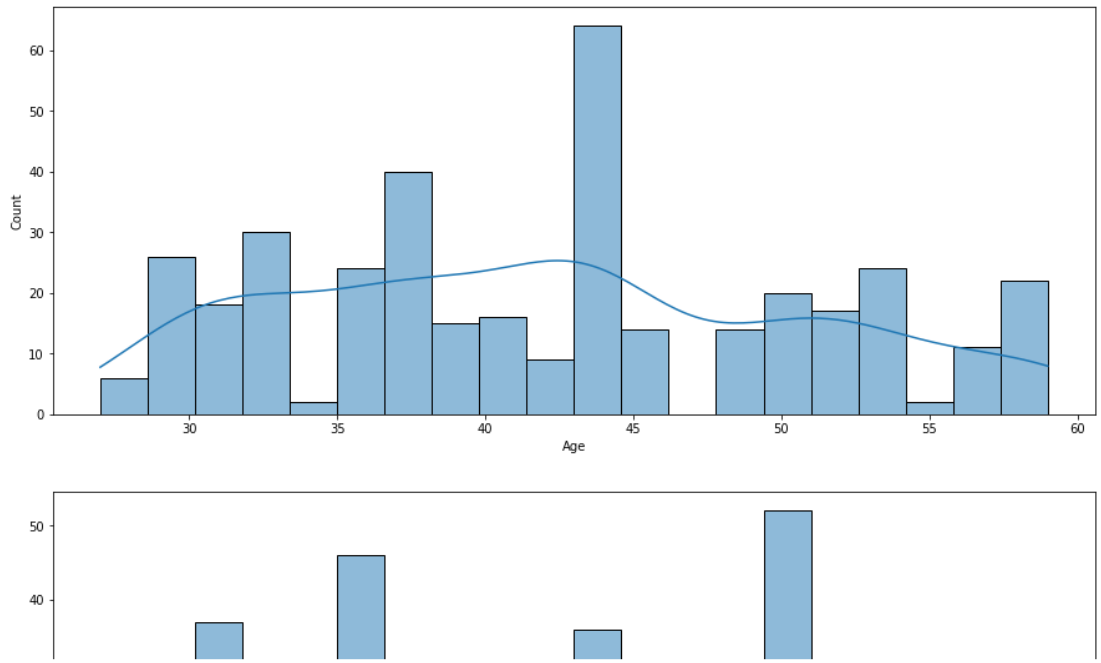
In [42]:

```
plt.figure(figsize=(15,6))
sns.violinplot(df['Diastolic'], data=df, palette = 'hls')
plt.xticks(rotation = 0)
plt.show()
```



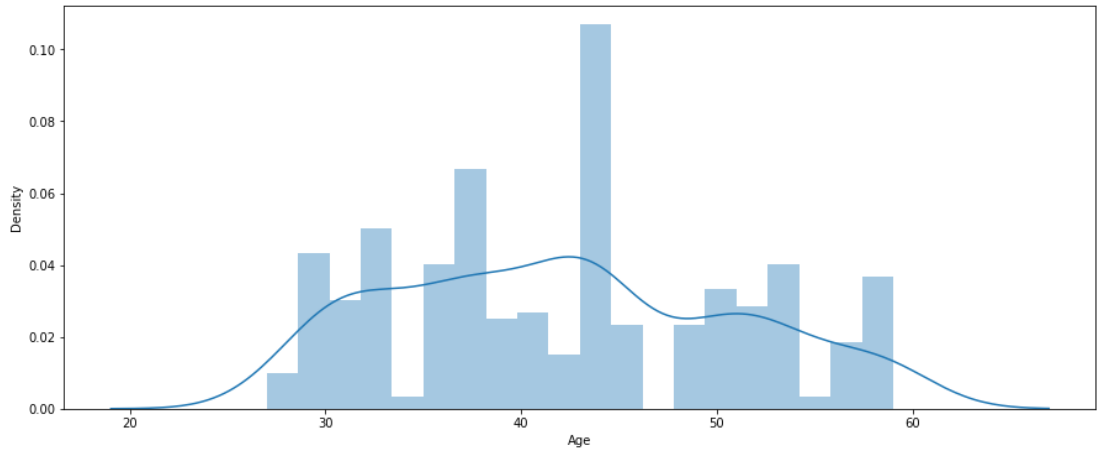
In [43]:

```
for i in numerical_columns:
    if i != 'Person ID':
        plt.figure(figsize=(15,6))
        sns.histplot(df[i], kde = True, bins = 20, palette = 'hls')
        plt.xticks(rotation = 0)
        plt.show()
```



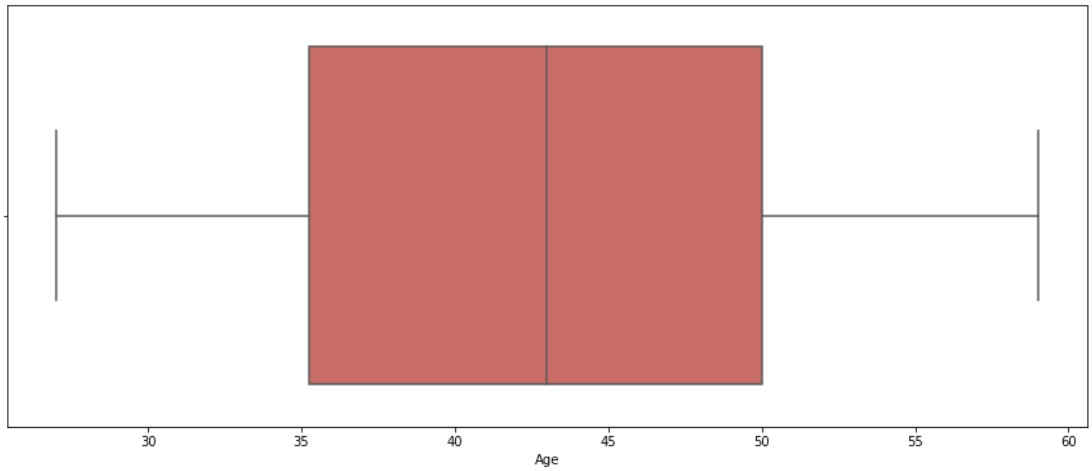
In [44]:

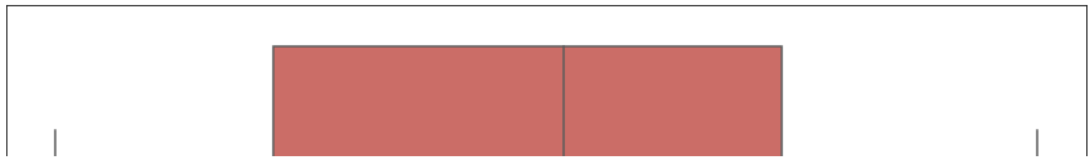
```
for i in numerical_columns:
    if i != 'Person ID':
        plt.figure(figsize=(15,6))
        sns.distplot(df[i], kde = True, bins = 20)
        plt.xticks(rotation = 0)
        plt.show()
```



In [45]:

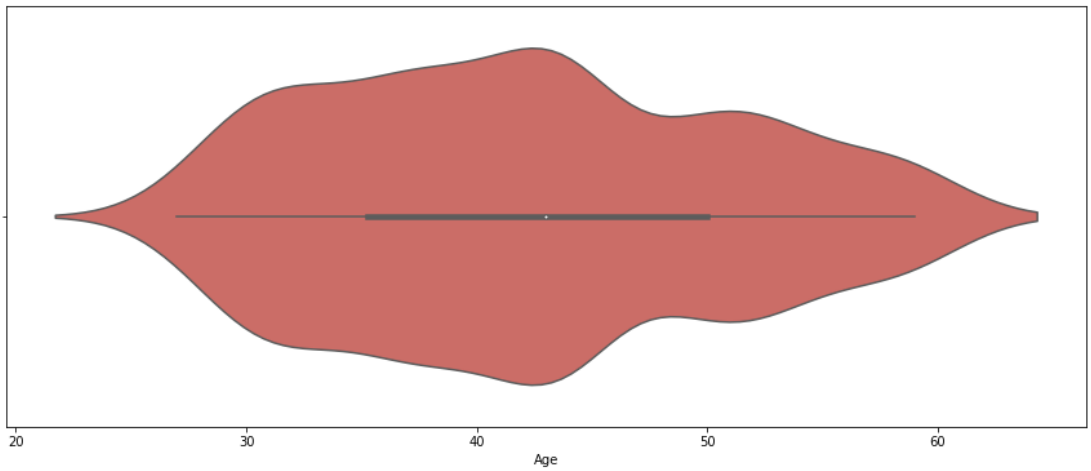
```
for i in numerical_columns:
    if i != 'Person ID':
        plt.figure(figsize=(15,6))
        sns.boxplot(df[i], data = df, palette = 'hls')
        plt.xticks(rotation = 0)
        plt.show()
```

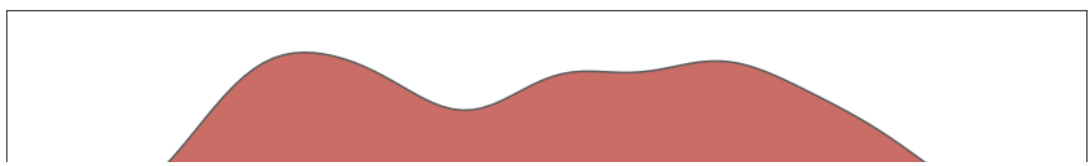




In [46]:

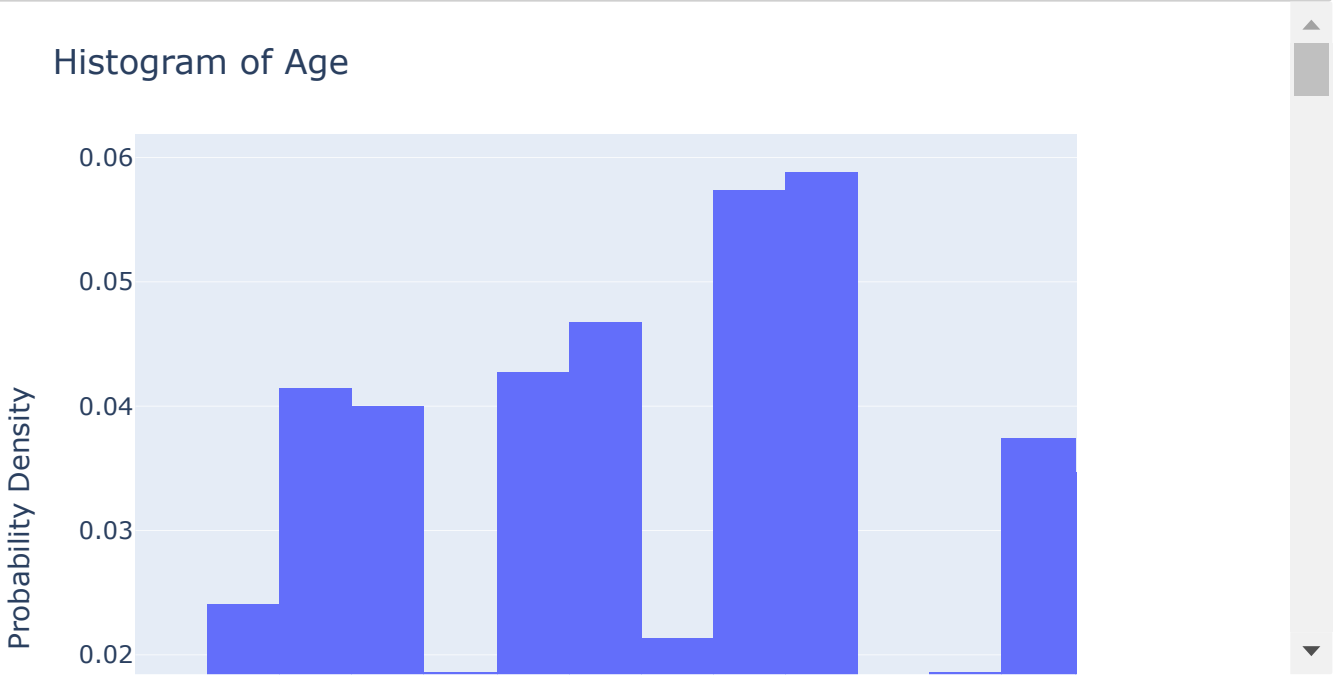
```
for i in numerical_columns:
    if i != 'Person ID':
        plt.figure(figsize=(15,6))
        sns.violinplot(df[i], data = df, palette = 'hls')
        plt.xticks(rotation = 0)
        plt.show()
```





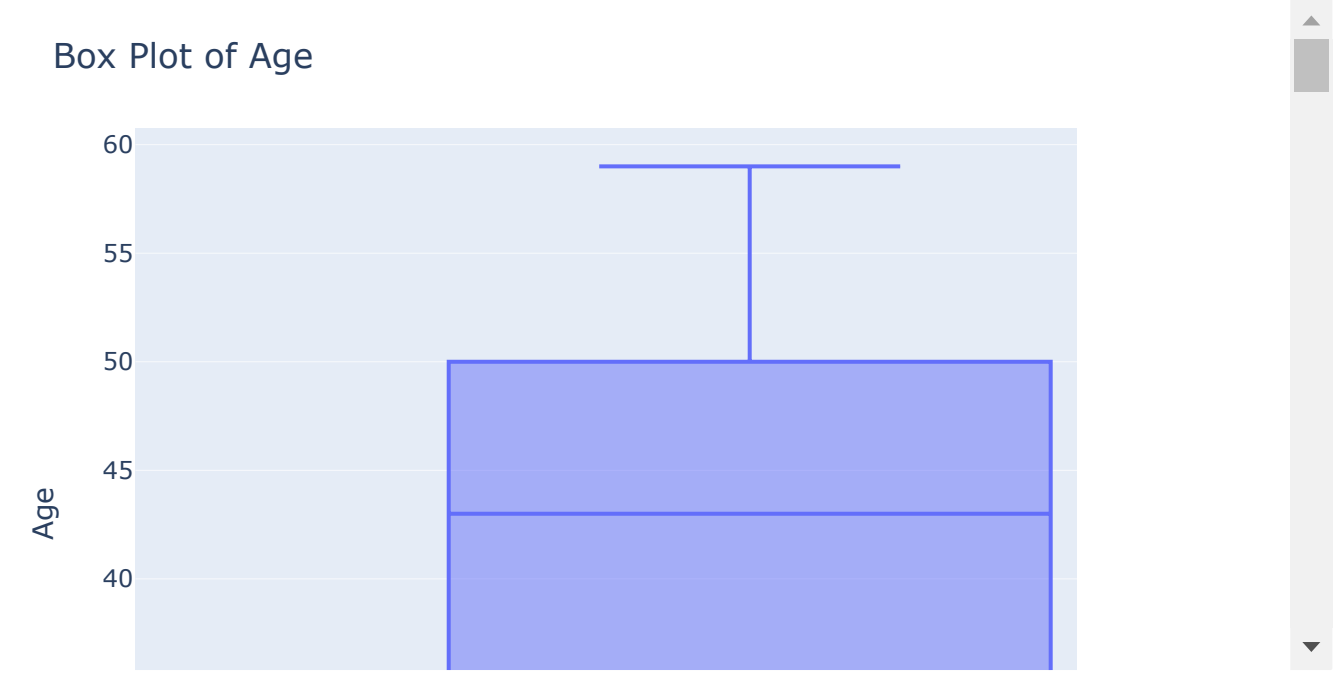
In [47]:

```
for column in numerical_columns:
    if column != 'Person ID':
        fig = px.histogram(df, x=column, nbins=20, histnorm='probability density')
        fig.update_layout(title=f"Histogram of {column}", xaxis_title=column, yaxis_titl
        fig.show()
```



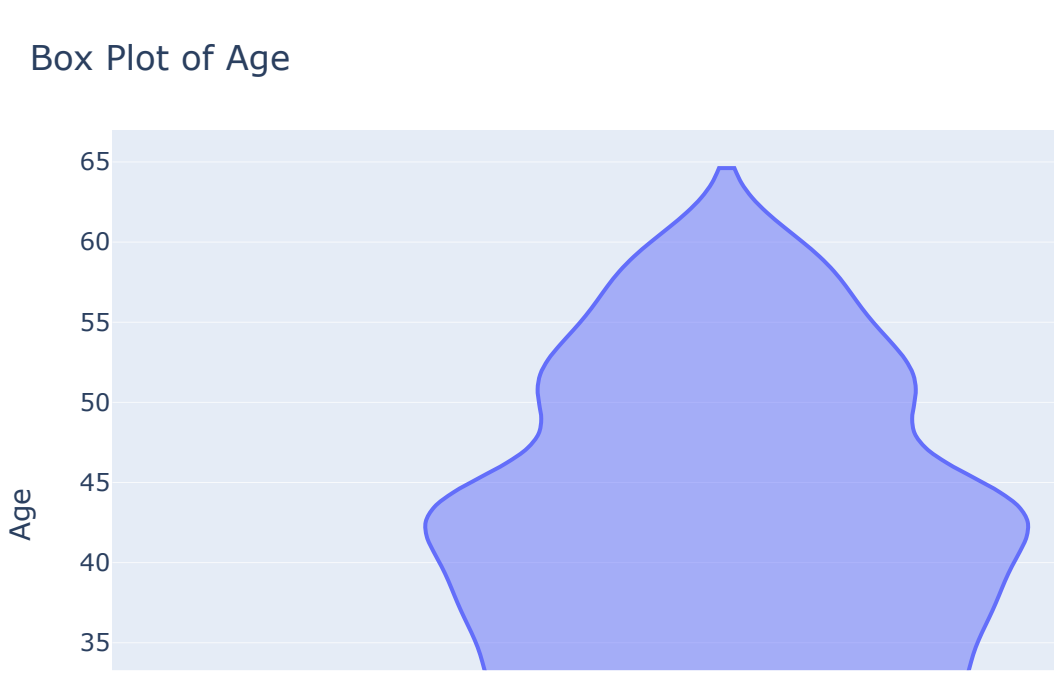
In [48]:

```
for column in numerical_columns:
    if column != 'Person ID':
        fig = px.box(df, y=column)
        fig.update_layout(title=f"Box Plot of {column}", yaxis_title=column)
        fig.show()
```



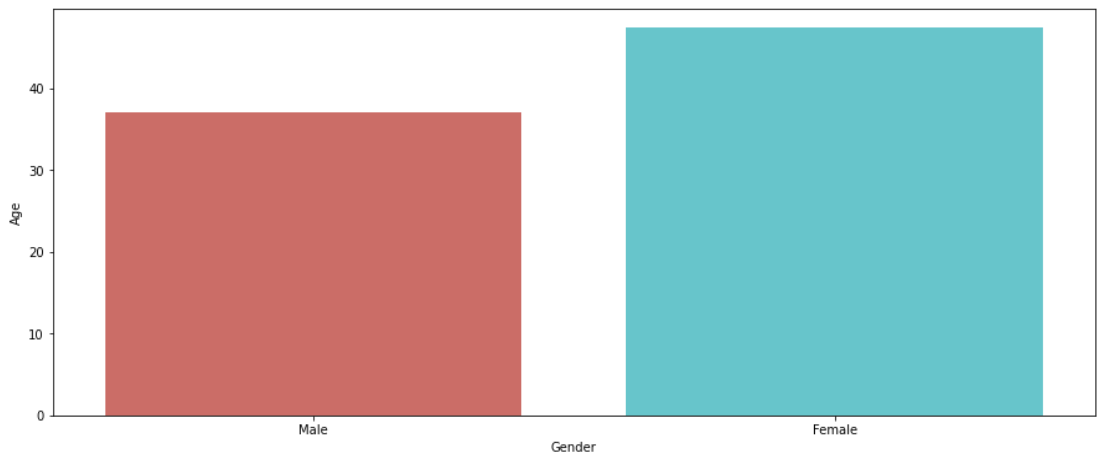
In [49]:

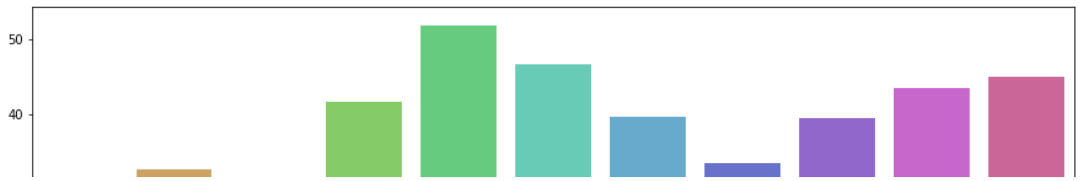
```
for column in numerical_columns:
    if column != 'Person ID':
        fig = px.violin(df, y=column)
        fig.update_layout(title=f"Box Plot of {column}", yaxis_title=column)
        fig.show()
```



In [50]:

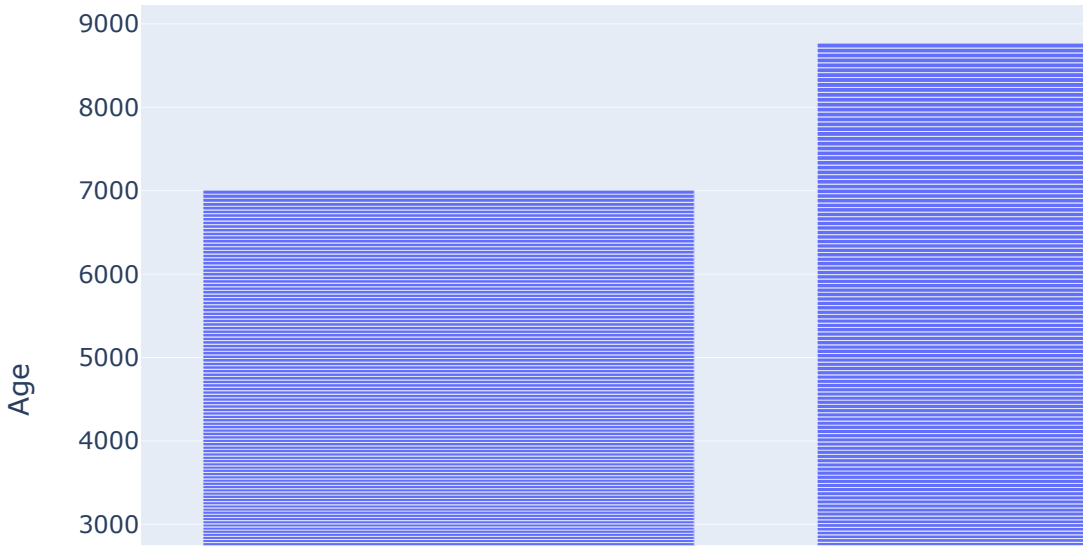
```
for i in numerical_columns:
    for j in object_columns:
        if i != 'Person ID':
            if j != 'Blood Pressure':
                plt.figure(figsize=(15,6))
                sns.barplot(x = df[j], y = df[i], data = df, ci = None, palette = 'hls')
                plt.show()
```





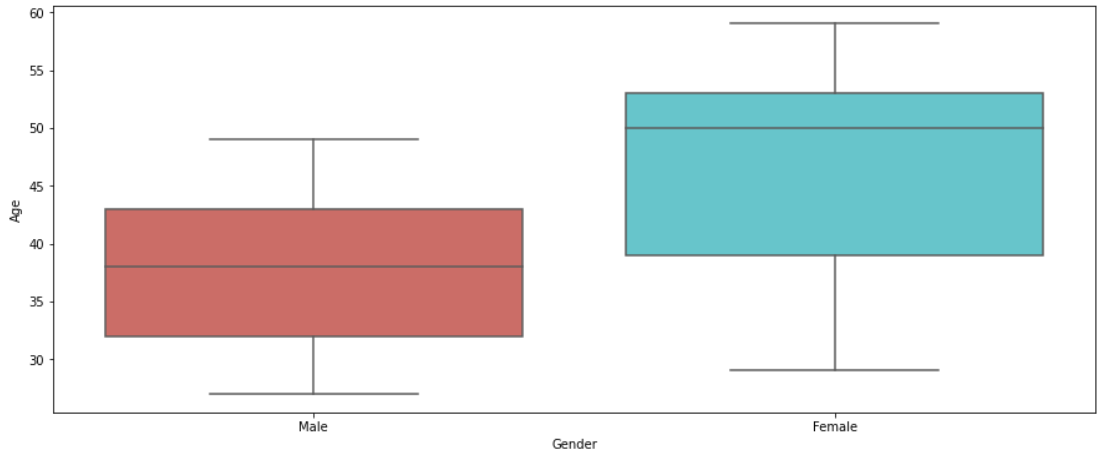
In [51]:

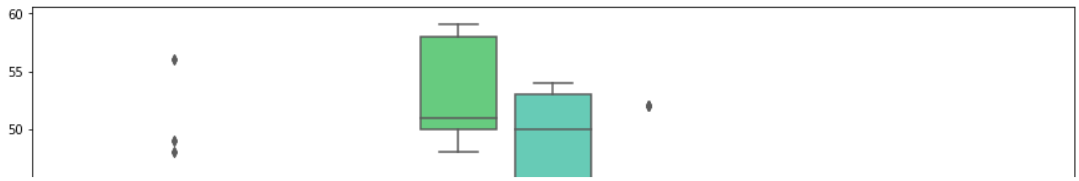
```
for i in numerical_columns:
    for j in object_columns:
        if i != 'Person ID' and j != 'Blood Pressure':
            fig = px.bar(df, x=j, y=i)
            fig.show()
```



In [52]:

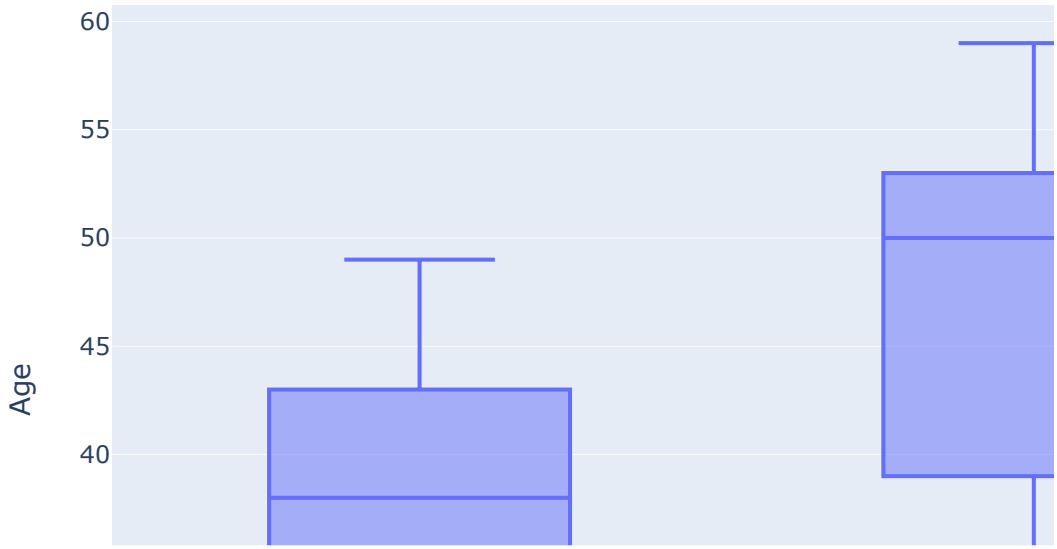
```
for i in numerical_columns:
    for j in object_columns:
        if i != 'Person ID':
            if j != 'Blood Pressure':
                plt.figure(figsize=(15,6))
                sns.boxplot(x = df[j], y = df[i], data = df, palette = 'hls')
                plt.show()
```





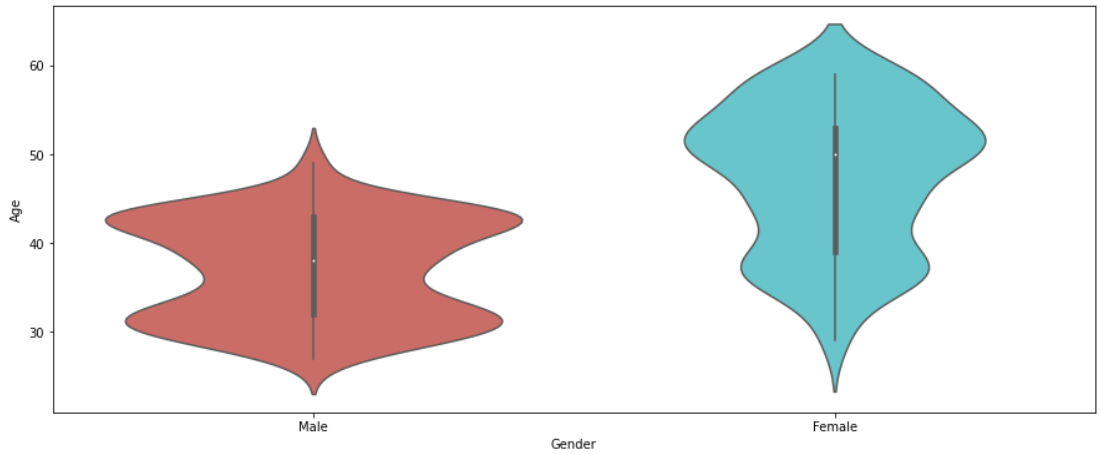
In [53]:

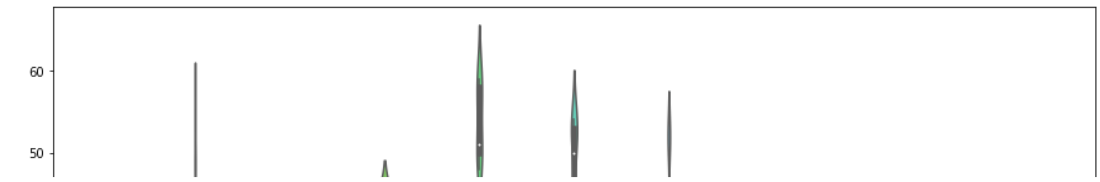
```
for i in numerical_columns:
    for j in object_columns:
        if i != 'Person ID' and j != 'Blood Pressure':
            fig = px.box(df, x=j, y=i)
            fig.show()
```



In [54]:

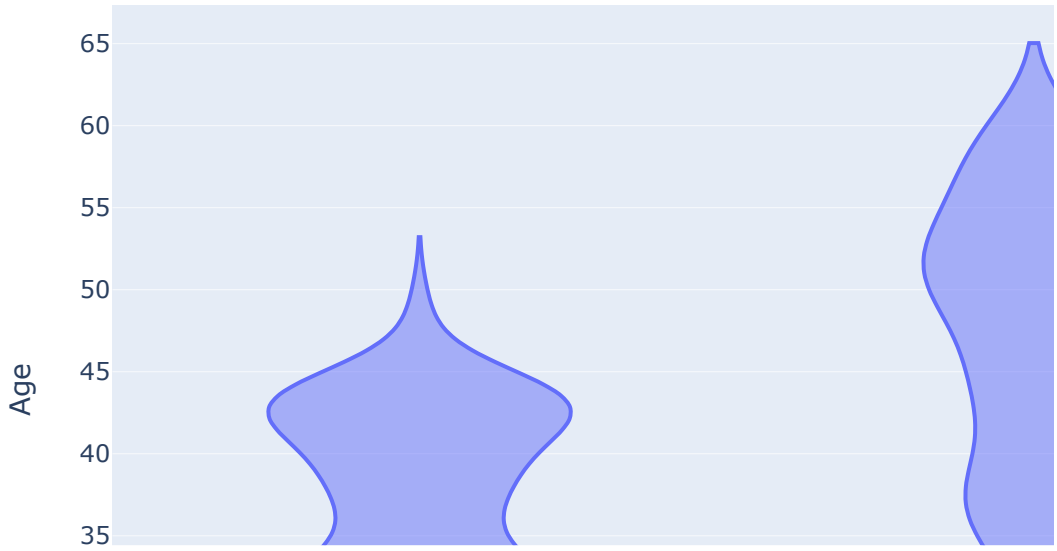
```
for i in numerical_columns:
    for j in object_columns:
        if i != 'Person ID':
            if j != 'Blood Pressure':
                plt.figure(figsize=(15,6))
                sns.violinplot(x = df[j], y = df[i], data = df, palette = 'hls')
                plt.show()
```





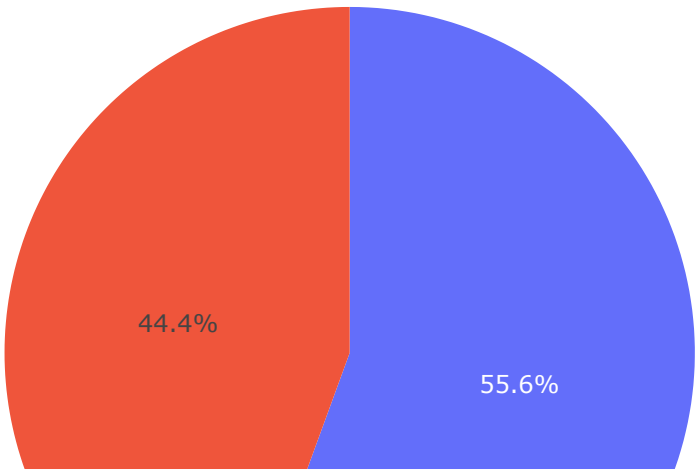
In [55]:

```
for i in numerical_columns:
    for j in object_columns:
        if i != 'Person ID' and j != 'Blood Pressure':
            fig = px.violin(df, x=j, y=i)
            fig.show()
```



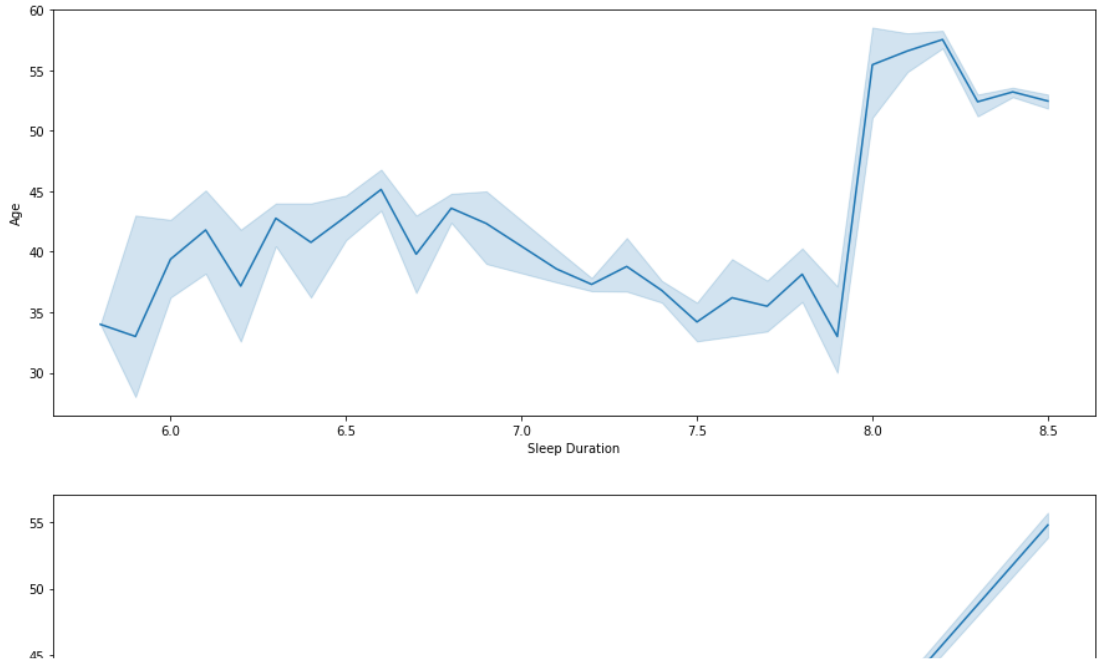
In [56]:

```
for i in numerical_columns:
    for j in object_columns:
        if i != 'Person ID' and j != 'Blood Pressure':
            fig = go.Figure(data=[go.Pie(labels=df[j], values=df[i])])
            fig.show()
```



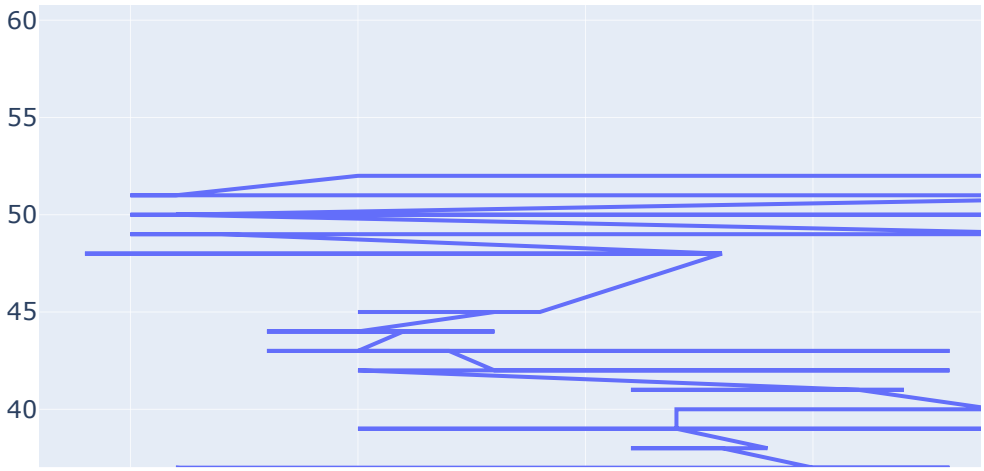
In [57]:

```
for i in numerical_columns:
    for j in numerical_columns:
        if i != 'Person ID':
            if j != 'Person ID':
                if i != j:
                    plt.figure(figsize=(15,6))
                    sns.lineplot(x = df[j], y = df[i], data = df, palette = 'hls')
                    plt.show()
```



In [58]:

```
for i in numerical_columns:
    for j in numerical_columns:
        if i != 'Person ID' and j != 'Person ID' and i != j:
            fig = go.Figure(data=go.Scatter(x=df[j], y=df[i], mode='lines'))
            fig.show()
```



In [59]:

```
df.columns
```

Out[59]:

```
Index(['Person ID', 'Gender', 'Age', 'Occupation', 'Sleep Duration',
       'Quality of Sleep', 'Physical Activity Level', 'Stress Level',
       'BMI Category', 'Blood Pressure', 'Heart Rate', 'Daily Steps',
       'Sleep Disorder', 'Systolic', 'Diastolic'],
      dtype='object')
```

In [60]:

```
average_systolic = df['Systolic'].mean()
average_diastolic = df['Diastolic'].mean()

print(f"Average Systolic Pressure: {average_systolic}")
print(f"Average Diastolic Pressure: {average_diastolic}")
```

```
Average Systolic Pressure: 128.55347593582889
Average Diastolic Pressure: 84.64973262032086
```

In [61]:

```
# Create a copy of the DataFrame with only the selected columns
df_selected = df.copy()
```

In [62]:

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

In [63]:

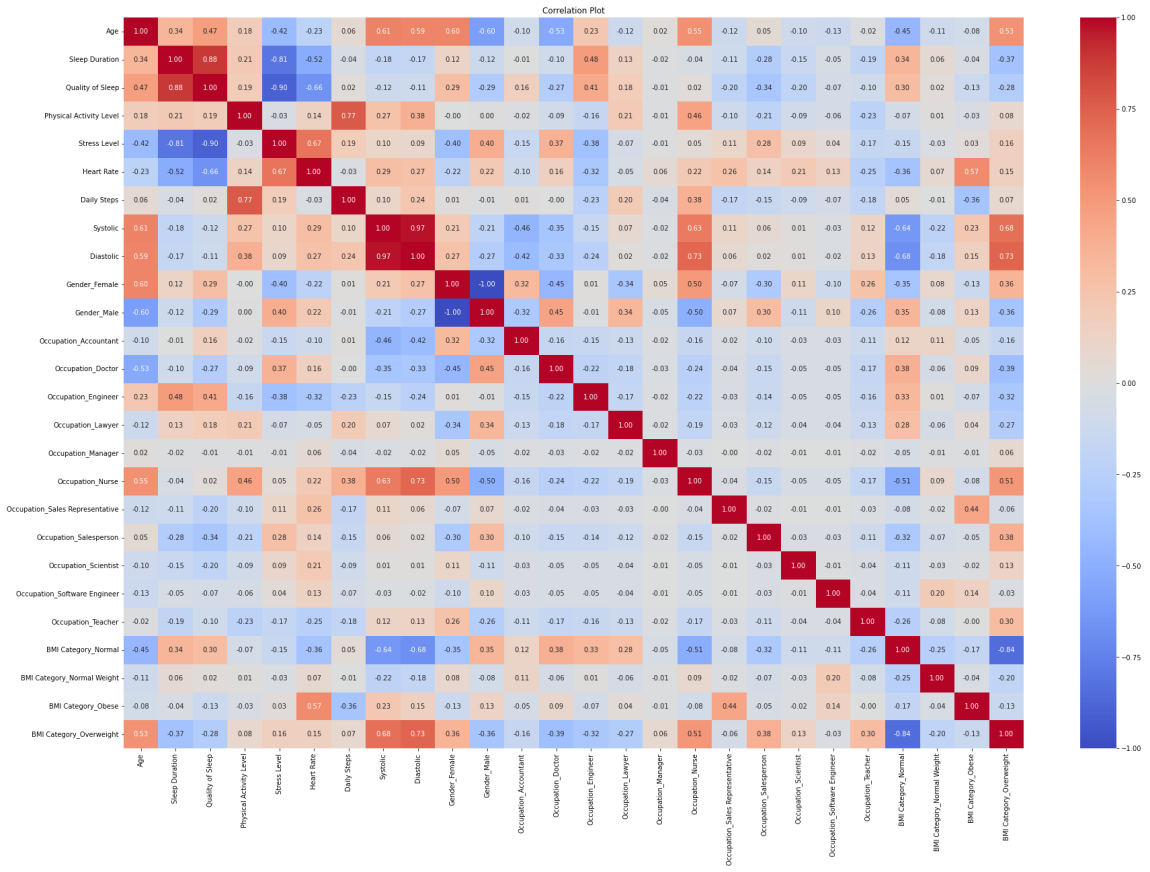
```
# Encode categorical features using one-hot encoding
df_encoded = pd.get_dummies(df_selected, columns=['Gender', 'Occupation', 'BMI Category'
```

In [64]:

```
# Calculate the correlation matrix
correlation_matrix = df_encoded.corr()
```

In [65]:

```
plt.figure(figsize=(30, 20))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Plot')
plt.show()
```



In [66]:

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df_encoded['Sleep Disorder'] = label_encoder.fit_transform(df_encoded['Sleep Disorder'])
```

In [67]:

```
df_encoded.columns
```

Out[67]:

```
Index(['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Lev
el',
       'Stress Level', 'Heart Rate', 'Daily Steps', 'Sleep Disorder',
       'Systolic', 'Diastolic', 'Gender_Female', 'Gender_Male',
       'Occupation_Accountant', 'Occupation_Doctor', 'Occupation_Enginee
r',
       'Occupation_Lawyer', 'Occupation_Manager', 'Occupation_Nurse',
       'Occupation_Sales Representative', 'Occupation_Salesperson',
       'Occupation_Scientist', 'Occupation_Software Engineer',
       'Occupation_Teacher', 'BMI Category_Normal',
       'BMI Category_Normal Weight', 'BMI Category_Obese',
       'BMI Category_Overweight'],
      dtype='object')
```

In [68]:

```
from sklearn.ensemble import RandomForestClassifier

# Assuming X is the feature matrix and y is the target variable
X = df_encoded.drop('Sleep Disorder', axis=1)  # Drop the target variable from the featu
y = df_encoded['Sleep Disorder']

# Create a random forest classifier
clf = RandomForestClassifier()

# Fit the classifier to the data
clf.fit(X, y)

# Get feature importance scores
feature_importance = clf.feature_importances_

# Create a DataFrame to display feature importance scores
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_import
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=Fal

# Print the feature importance scores
print(feature_importance_df)
```

```
                          Feature  Importance
25          BMI Category_Overweight    0.136686
7                         Systolic    0.134527
8                        Diastolic    0.131742
22              BMI Category_Normal    0.103014
1                    Sleep Duration    0.091207
0                               Age    0.065515
16                 Occupation_Nurse    0.062296
3           Physical Activity Level    0.051061
5                        Heart Rate    0.050716
6                       Daily Steps    0.049116
4                     Stress Level    0.025062
2                  Quality of Sleep    0.021485
18           Occupation_Salesperson    0.018330
10                     Gender_Male    0.009689
21               Occupation_Teacher    0.008291
13              Occupation_Engineer    0.006060
9                    Gender_Female    0.005971
12                Occupation_Doctor    0.005480
23       BMI Category_Normal Weight    0.005454
24                BMI Category_Obese    0.004698
14                 Occupation_Lawyer    0.003395
17  Occupation_Sales Representative    0.002632
11            Occupation_Accountant    0.002516
20     Occupation_Software Engineer    0.002411
19              Occupation_Scientist    0.002351
15                Occupation_Manager    0.000296
```

In [69]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

In [70]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

In [71]:

```
logreg = LogisticRegression()
```

In [72]:

```
logreg.fit(X_train, y_train)
```

Out[72]:

```
▾ LogisticRegression
LogisticRegression()
```

In [73]:

```
y_pred = logreg.predict(X_test)
```

In [74]:

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.72
```

In [75]:

```
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.40      0.27      0.32        15
           1       0.72      0.86      0.78        44
           2       1.00      0.75      0.86        16

    accuracy                           0.72        75
   macro avg       0.71      0.63      0.65        75
weighted avg       0.71      0.72      0.71        75
```

In [76]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [77]:

```
dt = DecisionTreeClassifier()
```

In [78]:

```
dt.fit(X_train, y_train)
```

Out[78]:

DecisionTreeClassifier

DecisionTreeClassifier()

In [79]:

```
y_pred = dt.predict(X_test)
```

In [80]:

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9333333333333333
```

In [81]:

```
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.87      0.87        15
           1       1.00      0.95      0.98        44
           2       0.83      0.94      0.88        16

    accuracy                           0.93        75
   macro avg       0.90      0.92      0.91        75
weighted avg       0.94      0.93      0.93        75
```

In [82]:

```
import xgboost as xgb
```

In [83]:

```
xgb_model = xgb.XGBClassifier()
```

In [84]:

```
xgb_model.fit(X_train, y_train)
```

Out[84]:

```
                                XGBClassifier
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytre
e=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depth
wise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_oneho
t=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_w
```

In [85]:

```
y_pred = xgb_model.predict(X_test)
```

In [86]:

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.92
```

In [87]:

```
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.87      0.81        15
           1       1.00      0.98      0.99        44
           2       0.87      0.81      0.84        16

    accuracy                           0.92        75
   macro avg       0.88      0.89      0.88        75
weighted avg       0.92      0.92      0.92        75
```