Stress Level Prediction Project (OOP with ML)

This notebook uses Stress level prediction using **Machine Learning** and **Object-Oriented Programming (OOP)** principles.

Step 1: Import Required Libraries

Below are the essential libraries and their purposes:

- `pandas` : Data manipulation
- `matplotlib` & `seaborn` : Visualization
- `pickle` : Save/load trained ML models
- `sklearn` : Machine learning tools (SVM, train-test split)

In [6]:
```
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

DATA LOADING AND UNDERSTANDING

In [7]:
```
# --- DATA LOADING & UNDERSTANDING ---
class DataLoader:
    """
    A class to load data from a CSV file using pandas.
    """
    def __init__(self, filepath):
        """
        Constructor that loads data from the given CSV file path.

        Parameters:
        filepath (str): The path to the CSV file.
        """
        self.df = pd.read_csv(filepath)
        print("Data Loaded.\n")

    def get_data(self):
        """
        Returns the loaded DataFrame.
```

```
        Returns:
        pd.DataFrame: The data loaded from the CSV file.
        """
        return self.df

    def display_head(self, n=5):
        """
        Returns the first n rows of the DataFrame.

        Parameters:
        n (int): Number of rows to return (default is 5).

        Returns:
        pd.DataFrame: First n rows of the dataset.
        """
        return self.df.head(n)

    def display_tail(self, n=5):
        """
        Returns the last n rows of the DataFrame.

        Parameters:
        n (int): Number of rows to return (default is 5).

        Returns:
        pd.DataFrame: Last n rows of the dataset.
        """
        return self.df.tail(n)

    def description(self):
        """
        Returns descriptive statistics of the DataFrame.

        Returns:
        pd.DataFrame: Summary including count, mean, std, min, max, etc.
        """
        return self.df.describe()

    def display_info(self):
        """
        Displays information about the DataFrame (columns, data types, memory usage
        """
        self.df.info()
```

DATA PREPROCESSING

In [8]:
```
# --- DATA PREPROCESSING ---
class DataPreprocessor:
    """
    A class to preprocess a pandas DataFrame.
    """
    def __init__(self, dF):
        """
        Initializes with a DataFrame.
```

```
        Parameters:
        dF (pd.DataFrame): The data to be preprocessed.
        """
        self.df = dF

    def check_missing_values(self):
        """
        Checks for missing (null) values in the DataFrame.

        Returns:
        pd.Series: Count of missing values per column.
        """
        return self.df.isnull().sum()

    def fill_missing(self):
        """
        Fills missing (NaN) values in numeric columns with the column mean.
        This modifies the DataFrame in place and does not return anything.
        """
        # Ensure only numeric columns are selected for mean calculation
        numeric_cols = self.df.select_dtypes(include=np.number).columns
        self.df[numeric_cols] = self.df[numeric_cols].fillna(self.df[numeric_cols].
        print("Missing values filled (numeric columns by mean).\n")


    def encode_categorical(self):
        """
        Encodes categorical (object type) columns using one-hot encoding.
        Replaces object columns with dummy variables and returns the updated DataFr
        If no categorical columns are present, returns the original DataFrame.

        Returns:
        pd.DataFrame: DataFrame with encoded categorical variables.
        """
        object_cols = self.df.select_dtypes(include='object').columns
        if len(object_cols) > 0:
            self.df = pd.get_dummies(self.df, columns=object_cols, drop_first=True)
            print("Categorical columns encoded.\n")
        else:
            print("No categorical columns to encode.\n")
        return self.df
```

UNIVARIATE AND BIVARIATE ANALYSIS

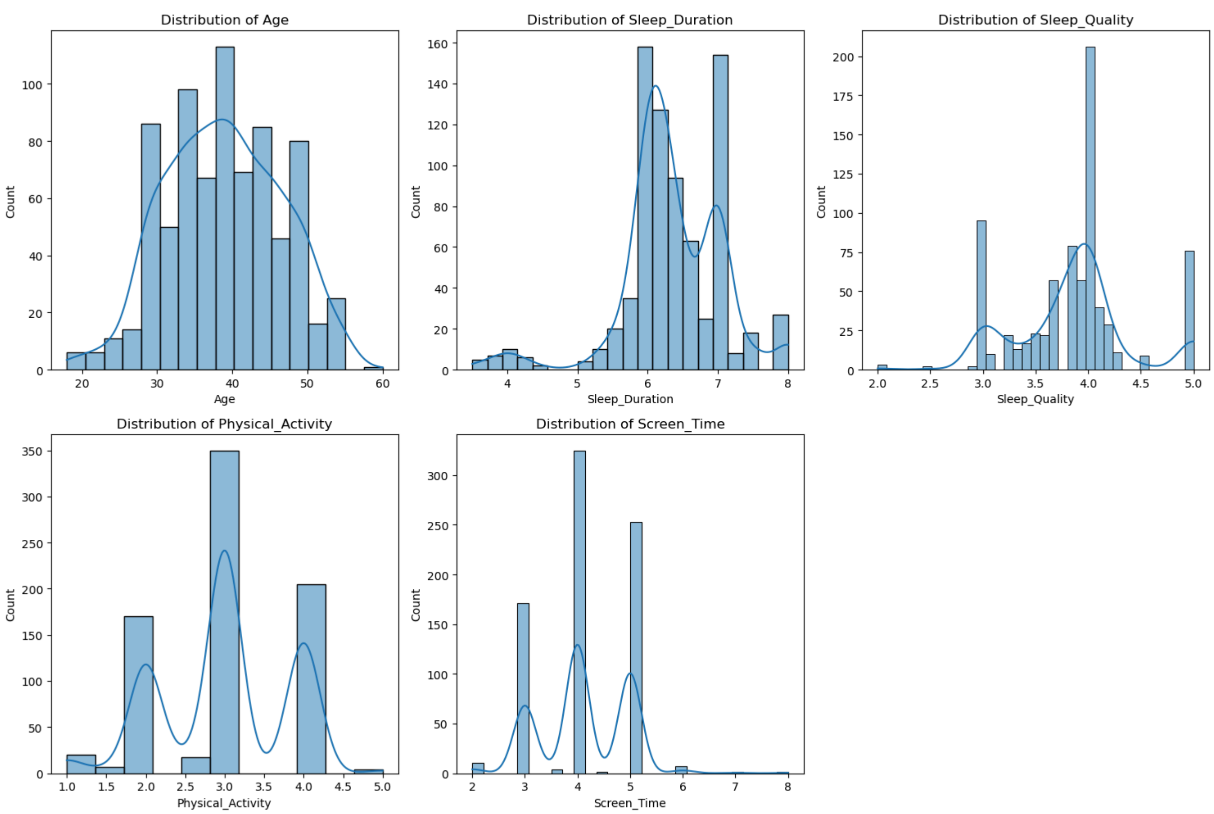
In [9]:
```
# --- UNIVARIATE & BIVARIATE ANALYSIS ---
class Analyzer:
    """
    A class for performing data analysis and visualization.
    """
    def __init__(self, df):
        self.df = df

    def univariate(self, column):
```

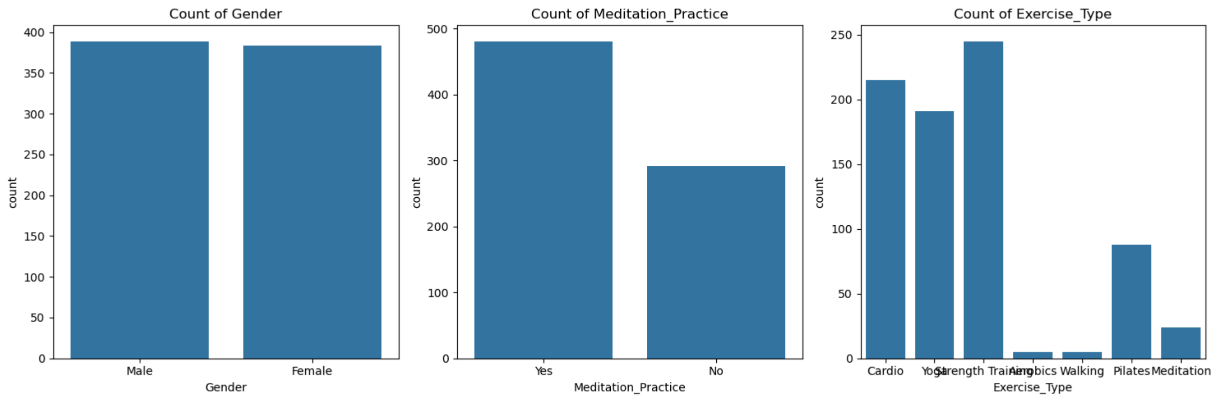
```
        """
        Performs univariate analysis on a given column.
        Displays value counts and a histogram with KDE.

        Parameters:
        column (str): The column to analyze.
        """
        if column not in self.df.columns:
            print(f"Column '{column}' not found in the DataFrame.")
            return

        print(f"Univariate Analysis for: {column}")
        print(self.df[column].value_counts())

        plt.figure(figsize=(8, 5))
        if pd.api.types.is_numeric_dtype(self.df[column]):
            sns.histplot(self.df[column], kde=True)
        else:
            sns.countplot(y=self.df[column], order=self.df[column].value_counts().i

        plt.title(f"Univariate Analysis of {column}")
        plt.xlabel(column)
        plt.ylabel("Count")
        plt.tight_layout()
        plt.show()

    def correlation_matrix(self, annot=True, cmap='coolwarm'):
        """
        Plots the correlation matrix heatmap for numerical features.

        Parameters:
        annot (bool): If True, write the data value into each cell.
        cmap (str): Colormap to use for the heatmap.
        """
        # Select only numeric columns for correlation matrix
        numeric_df = self.df.select_dtypes(include=np.number)
        if numeric_df.empty:
            print("No numeric columns found for correlation matrix.")
            return

        corr = numeric_df.corr()
        plt.figure(figsize=(10, 8))
        sns.heatmap(corr, annot=annot, cmap=cmap, fmt=".2f", linewidths=.5)
        plt.title("Correlation Matrix")
        plt.tight_layout()
        plt.show()
```

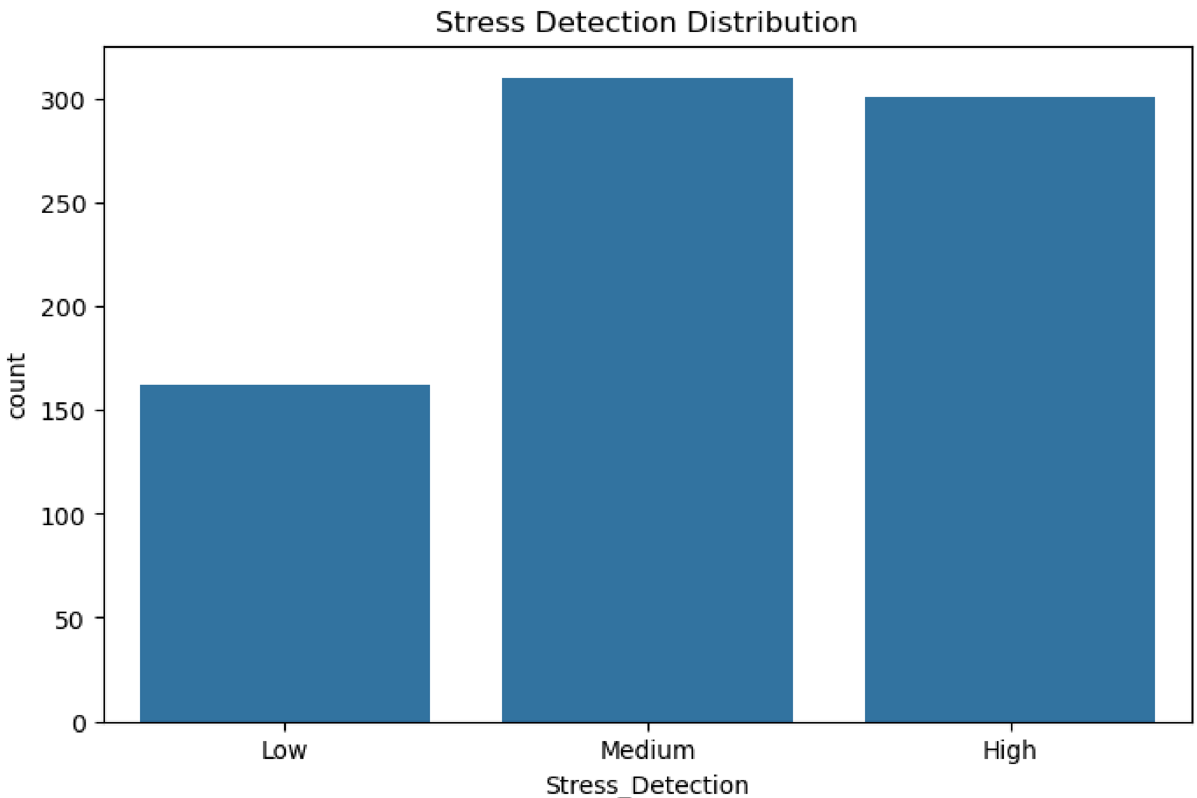
GRAPHS

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset here
```

```python
data = pd.read_csv("stress_detection_data.csv")

# --- Univariate Analysis ---

# Numeric columns
num_cols = ['Age', 'Sleep_Duration', 'Sleep_Quality', 'Physical_Activity', 'Screen_
plt.figure(figsize=(15, 10))
for i, col in enumerate(num_cols, 1):
    plt.subplot(2, 3, i)
    sns.histplot(data=data, x=col, kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()

# Categorical columns
cat_cols = ['Gender', 'Meditation_Practice', 'Exercise_Type']
plt.figure(figsize=(15, 5))
for i, col in enumerate(cat_cols, 1):
    plt.subplot(1, 3, i)
    sns.countplot(data=data, x=col)
    plt.title(f'Count of {col}')
plt.tight_layout()
plt.show()

# Target variable
plt.figure(figsize=(8, 5))
sns.countplot(x=data['Stress_Detection'])
plt.title('Stress Detection Distribution')
plt.show()
```







In [11]:
```
# --- GRAPHS (ADAPTED FOR STRESS DATA) ---
class StressGraphs:
    """
    A class for plotting various graphs related to stress detection data.
    """
    def __init__(self, df):
        self.df = df
        sns.set_theme(style="whitegrid")

    def plot_anxiety_vs_stress(self, x='Anxiety_Level', y='Stress_Level', hue=None)
        """
        Displays a scatter plot showing the relationship between Anxiety Level and
        """
        if x not in self.df.columns or y not in self.df.columns:
            print(f"One or more columns ({x}, {y}) not found for plot_anxiety_vs_st
            return

        plt.figure(figsize=(8, 5))
        sns.scatterplot(data=self.df, x=x, y=y, hue=hue)
```

```
        plt.title('Anxiety Level vs Stress Level')
        plt.xlabel('Anxiety Level')
        plt.ylabel('Stress Level')
        plt.tight_layout()
        plt.show()

    def plot_depression_vs_stress(self, x='Depression_Score', y='Stress_Level', hue
        """
        Displays a scatter plot showing the relationship between Depression Score a
        """
        if x not in self.df.columns or y not in self.df.columns:
            print(f"One or more columns ({x}, {y}) not found for plot_depression_vs
            return

        plt.figure(figsize=(8, 5))
        sns.scatterplot(data=self.df, x=x, y=y, hue=hue)
        plt.title('Depression Score vs Stress Level')
        plt.xlabel('Depression Score')
        plt.ylabel('Stress Level')
        plt.tight_layout()
        plt.show()

    def plot_sleep_quality_vs_stress(self, x='Sleep_Quality', y='Stress_Level', hue
        """
        Displays a scatter plot showing the relationship between Sleep Quality and
        """
        if x not in self.df.columns or y not in self.df.columns:
            print(f"One or more columns ({x}, {y}) not found for plot_sleep_quality
            return

        plt.figure(figsize=(8, 5))
        sns.scatterplot(data=self.df, x=x, y=y, hue=hue)
        plt.title('Sleep Quality vs Stress Level')
        plt.xlabel('Sleep Quality')
        plt.ylabel('Stress Level')
        plt.tight_layout()
        plt.show()

    def plot_stress_distribution(self, x='Stress_Level', kde=True, color='lightcora
        """
        Displays a histogram showing the distribution of Stress Levels in the datas
        with a KDE curve overlay to visualize the distribution shape.
        """
        if x not in self.df.columns:
            print(f"Column '{x}' not found for plot_stress_distribution.")
            return

        plt.figure(figsize=(8, 5))
        sns.histplot(data=self.df, x=x, kde=kde, color=color)
        plt.title('Stress Level Distribution')
        plt.xlabel('Stress Level')
        plt.ylabel('Count')
        plt.tight_layout()
        plt.show()

    def plot_categorical_vs_stress(self, categorical_col, y='Stress_Level'):
```

```
        """
        Displays a box plot comparing the stress distribution across different cate
        Useful for identifying median, spread, and outliers for categorical feature
        """
        if categorical_col not in self.df.columns or y not in self.df.columns:
            print(f"One or more columns ({categorical_col}, {y}) not found for plot
            return

        if pd.api.types.is_numeric_dtype(self.df[categorical_col]):
            print(f"Column '{categorical_col}' is numeric, not suitable for this pl
            return

        plt.figure(figsize=(10, 6))
        sns.boxplot(data=self.df, x=categorical_col, y=y)
        plt.title(f'Stress Level by {categorical_col}')
        plt.xlabel(categorical_col)
        plt.ylabel('Stress Level')
        plt.xticks(rotation=45, ha='right') # Rotate labels for better readability
        plt.tight_layout()
        plt.show()
```

DATA SPLITTING

In [12]:
```
# --- DATA SPLITTING ---
class DataSplitter:
    """
    A class to split the dataset into training and testing sets.
    """
    def __init__(self, df):
        self.df = df

    def split(self, target_col, test_size=0.2, random_state=42):
        """
        Splits the data into features (X) and target (y), then into training and te

        Parameters:
        target_col (str): The name of the target column.
        test_size (float): Proportion of the dataset to include in the test split.
        random_state (int): Seed for random number generator. Default is 42.

        Returns:
        tuple: X_train, X_test, y_train, y_test
        """
        if target_col not in self.df.columns:
            raise ValueError(f"Target column '{target_col}' not found in the DataFr

        X = self.df.drop(target_col, axis=1)
        y = self.df[target_col]

        # Handle cases where target variable might have a single unique value
        if y.nunique() == 1:
            print(f"Warning: Target column '{target_col}' has only one unique value
            # Forcing a split even with one unique value, but it's important to not
            # In a real scenario, this might indicate an issue with the data or tar
```

```
        return train_test_split(X, y, test_size=test_size, random_state=random_stat
```

MODELTRAIN SVM

In [13]:
```
# --- MODEL TRAIN (SVM) ---
class StressModel:
    """
    A class to build and train a stress prediction model using Support Vector Regre
    """
    def __init__(self):
        # Initializing SVR with a linear kernel as per the original logic
        self.model = SVR(kernel='linear')

    def train(self, X_train, y_train):
        """
        Trains the SVR model on the provided training data.

        Parameters:
        X_train (pd.DataFrame): Training features.
        y_train (pd.Series): Target values for training.

        Returns:
        model: The trained SVR model.
        """
        print("Training model...")
        self.model.fit(X_train, y_train)
        print("Model training complete.\n")
        return self.model
```

EVALUATE MODEL

In [14]:
```
# --- EVALUATE MODEL ---
class ModelEvaluator:
    """
    A class to evaluate the performance of a regression model.
    """
    def evaluate(self, model, X_test, y_test):
        """
        Evaluates the model using R² Score, MAE, and MSE.

        Parameters:
        model: Trained regression model.
        X_test (pd.DataFrame): Test features.
        y_test (pd.Series): True target values.

        Returns:
        dict: Evaluation metrics including R², MAE, and MSE.
        """
        print("Evaluating model...")
        y_pred = model.predict(X_test)

        evaluation_metrics = {
```

```
            "R2 Score": round(r2_score(y_test, y_pred), 4),
            "MAE": round(mean_absolute_error(y_test, y_pred), 2),
            "MSE": round(mean_squared_error(y_test, y_pred), 2)
        }
        print("Model evaluation complete.\n")
        return evaluation_metrics
```

SAVE AS PICKLE

In [15]:
```
# --- MODEL SAVE AS PICKLE ---
class ModelSaver:
    """
    A class to save a trained model to a file using pickle.
    """
    def __init__(self, model, filename='stress_model.pkl'):
        """
        Initializes the model saver.

        Parameters:
        model: Trained model to be saved.
        filename (str): Name of the file to save the model. Default is 'stress_mode
        """
        self.model = model
        self.filename = filename

    def save(self):
        """
        Saves the model to the specified file using pickle.

        Returns:
        str: The filename where the model was saved.
        """
        try:
            with open(self.filename, 'wb') as f:
                pickle.dump(self.model, f)
            print(f"Model saved as {self.filename}\n")
            return self.filename
        except Exception as e:
            print(f"Error saving model: {e}")
            return None
```

OBJECTS OF ALL CLASSES

In [11]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('stress_detection_data.csv')

# Display basic information
print("=== Dataset Information ===")
```

```
print(f"Shape: {df.shape}")
print("\nFirst 5 rows:")
print(df.head())
print("\nData types and non-null counts:")
print(df.info())
print("\nDescriptive statistics:")
print(df.describe(include='all'))

# Data preprocessing
# Convert time columns to datetime and extract hours
df['Wake_Up_Time'] = pd.to_datetime(df['Wake_Up_Time'], format='%I:%M %p').dt.hour
df['Bed_Time'] = pd.to_datetime(df['Bed_Time'], format='%I:%M %p').dt.hour

# Encode categorical variables
label_encoders = {}
categorical_cols = ['Gender', 'Occupation', 'Marital_Status', 'Smoking_Habit',
                    'Meditation_Practice', 'Exercise_Type', 'Stress_Detection']
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Function to show value counts for categorical columns
def show_value_counts(df, cols):
    for col in cols:
        print(f"\nValue counts for {col}:")
        print(df[col].value_counts())

show_value_counts(df, categorical_cols)

# Correlation analysis
print("\n=== Correlation with Stress Level ===")
correlations = df.corr()['Stress_Detection'].sort_values(ascending=False)
print(correlations)

# Visualization 1: Stress distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='Stress_Detection', data=df)
plt.title('Distribution of Stress Levels')
plt.xticks(ticks=[0, 1, 2], labels=['Low', 'Medium', 'High'])
plt.show()

# Visualization 2: Top correlated features
top_features = correlations.index[1:6]  # Exclude Stress_Detection itself
plt.figure(figsize=(10, 6))
sns.barplot(x=top_features, y=correlations[1:6])
plt.title('Top Features Correlated with Stress Level')
plt.xticks(rotation=45)
plt.show()

# Visualization 3: Stress by Occupation (top 10)
top_occupations = df['Occupation'].value_counts().index[:10]
plt.figure(figsize=(12, 6))
sns.boxplot(x='Occupation', y='Stress_Detection',
            data=df[df['Occupation'].isin(top_occupations)])
plt.title('Stress Levels by Top 10 Occupations')
```

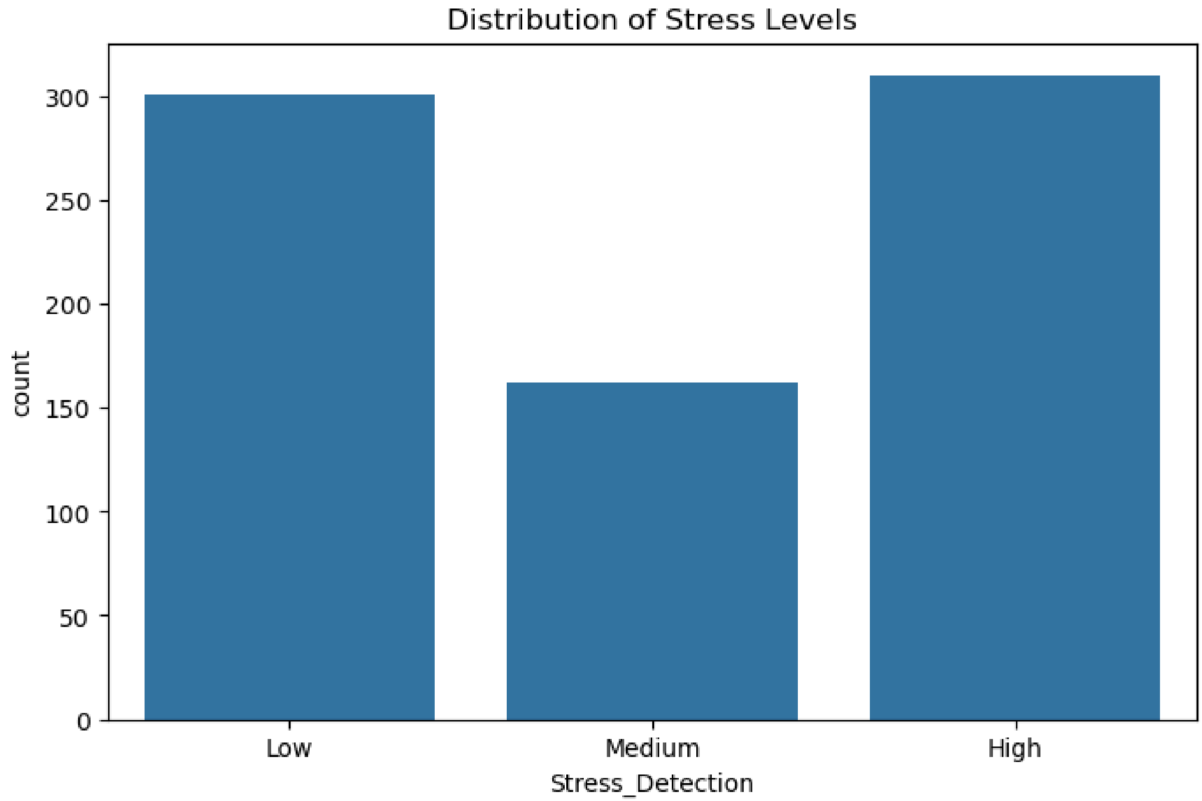
```
plt.xticks(rotation=45)
plt.show()

# Visualization 4: Pairplot of top correlated features
sns.pairplot(df[['Stress_Detection', 'Blood_Pressure', 'Cholesterol_Level',
                'Blood_Sugar_Level', 'Work_Hours']])
plt.suptitle('Pairplot of Top Correlated Features with Stress', y=1.02)
plt.show()

# Group analysis by stress level
print("\n=== Average Values by Stress Level ===")
stress_groups = df.groupby('Stress_Detection').mean()
print(stress_groups[['Age', 'Sleep_Duration', 'Sleep_Quality', 'Physical_Activity',
                    'Screen_Time', 'Work_Hours', 'Blood_Pressure',
                    'Cholesterol_Level', 'Blood_Sugar_Level']])

# Function to decode encoded values for interpretation
def decode_value(col, val):
    if col in label_encoders:
        return label_encoders[col].inverse_transform([val])[0]
    return val
```

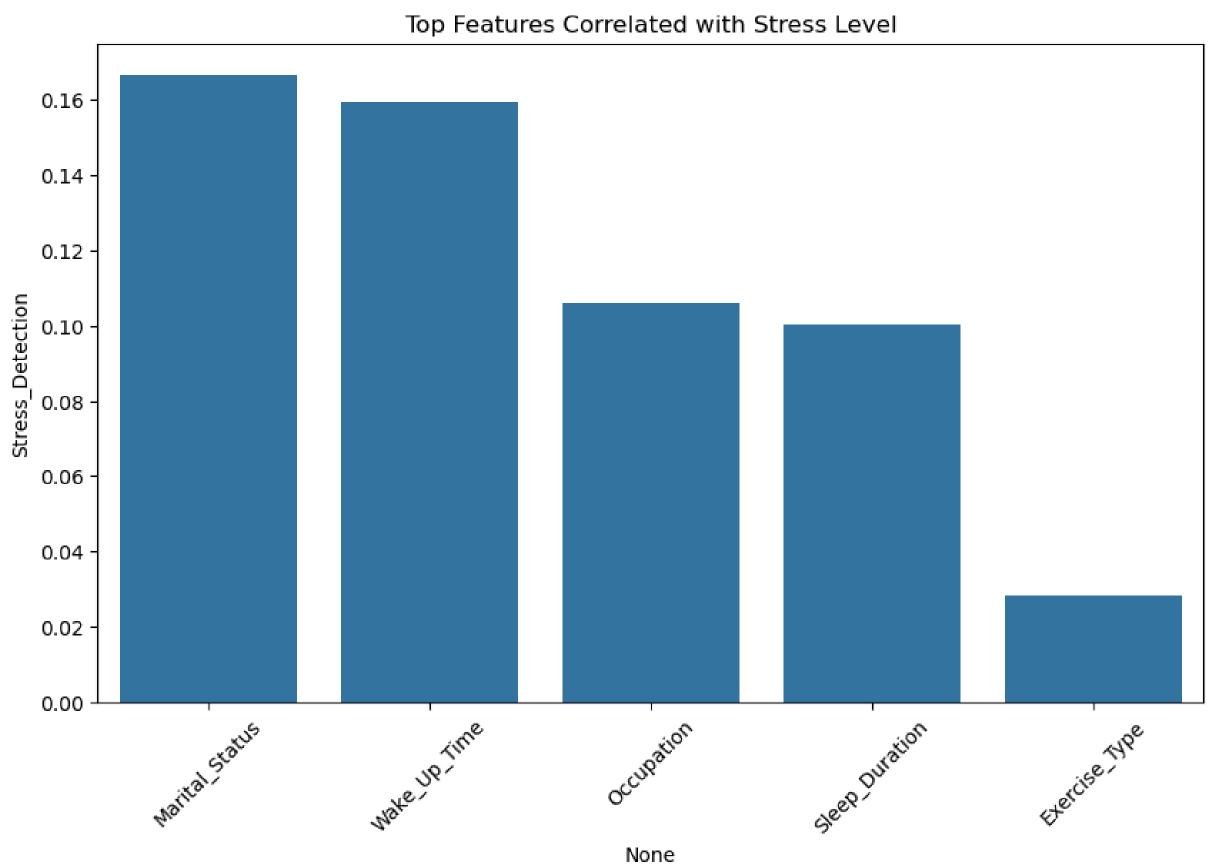
```
=== Dataset Information ===
Shape: (773, 22)

First 5 rows:
   Age  Gender         Occupation Marital_Status  Sleep_Duration  \
0   30    Male  Software Engineer         Single             7.0
1   35  Female  Marketing Manager        Married             6.0
2   40    Male     Data Scientist       Divorced             7.0
3   35    Male  Software Engineer         Single             7.0
4   29  Female            Teacher         Single             8.0

   Sleep_Quality Wake_Up_Time  Bed_Time  Physical_Activity  Screen_Time  ...  \
0            4.0      7:00 AM  10:00 PM                2.0          4.0  ...
1            3.0      6:00 AM  11:00 PM                1.0          3.0  ...
2            4.0      7:00 AM  10:00 PM                2.0          4.0  ...
3            4.0      7:00 AM  10:00 PM                2.0          4.0  ...
4            5.0      6:30 AM  10:30 PM                3.0          2.0  ...

   Smoking_Habit  Work_Hours Travel_Time  Social_Interactions  \
0             No           8         1.0                    5
1             No           9         2.0                    3
2             No           8         1.0                    5
3             No           8         1.0                    5
4             No           7         1.0                    4

   Meditation_Practice      Exercise_Type Blood_Pressure Cholesterol_Level  \
0                  Yes             Cardio            120               180
1                   No               Yoga            110               160
2                  Yes  Strength Training            130               200
3                  Yes             Cardio            120               180
4                  Yes               Yoga            110               180

   Blood_Sugar_Level  Stress_Detection
0                 90               Low
1                 80            Medium
2                100              High
3                 90               Low
4                 90               Low

[5 rows x 22 columns]

Data types and non-null counts:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 773 entries, 0 to 772
Data columns (total 22 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Age                  773 non-null    int64
 1   Gender               773 non-null    object
 2   Occupation           773 non-null    object
 3   Marital_Status       773 non-null    object
 4   Sleep_Duration       773 non-null    float64
 5   Sleep_Quality        773 non-null    float64
 6   Wake_Up_Time         773 non-null    object
 7   Bed_Time             773 non-null    object
 8   Physical_Activity    773 non-null    float64
```

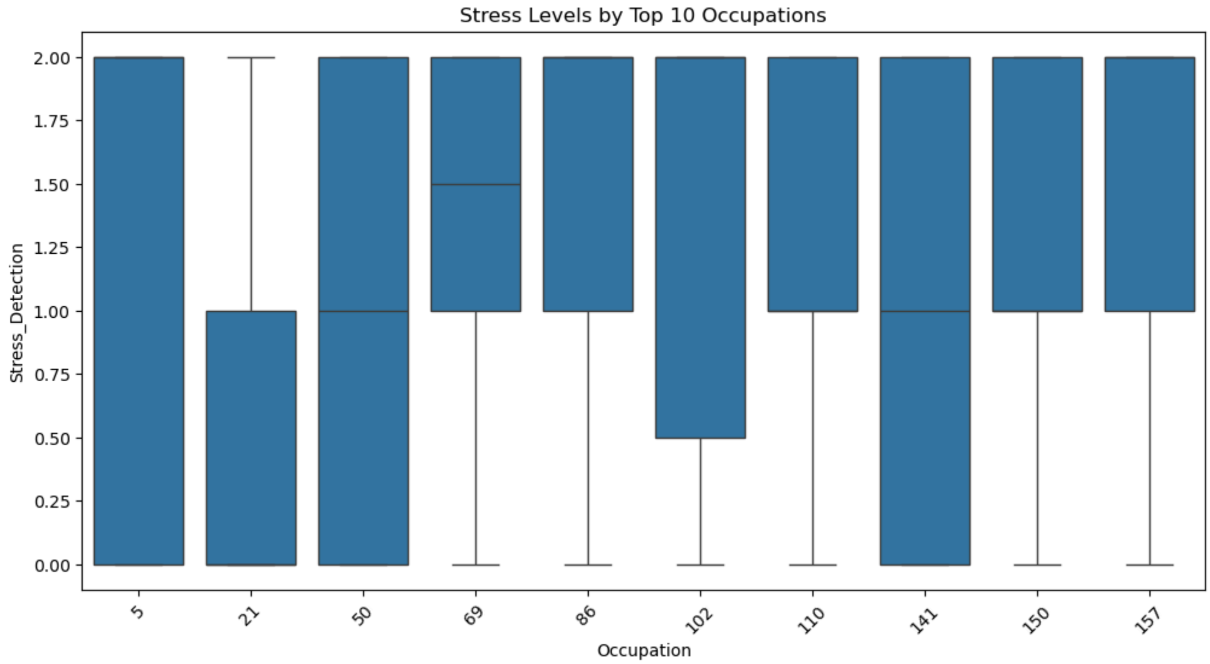
```
 9   Screen_Time          773 non-null    float64
 10  Caffeine_Intake      773 non-null    int64
 11  Alcohol_Intake       773 non-null    int64
 12  Smoking_Habit        773 non-null    object
 13  Work_Hours           773 non-null    int64
 14  Travel_Time          773 non-null    float64
 15  Social_Interactions  773 non-null    int64
 16  Meditation_Practice  773 non-null    object
 17  Exercise_Type        773 non-null    object
 18  Blood_Pressure       773 non-null    int64
 19  Cholesterol_Level    773 non-null    int64
 20  Blood_Sugar_Level    773 non-null    int64
 21  Stress_Detection     773 non-null    object
dtypes: float64(5), int64(8), object(9)
memory usage: 133.0+ KB
None

Descriptive statistics:
               Age Gender Occupation Marital_Status  Sleep_Duration  \
count   773.000000    773        773            773      773.000000
unique         NaN      2        169              3             NaN
top            NaN   Male    Teacher        Married             NaN
freq           NaN    389         37            360             NaN
mean     38.887451    NaN        NaN            NaN        6.338422
std       7.686642    NaN        NaN            NaN        0.733584
min      18.000000    NaN        NaN            NaN        3.500000
25%      33.000000    NaN        NaN            NaN        6.000000
50%      39.000000    NaN        NaN            NaN        6.300000
75%      45.000000    NaN        NaN            NaN        7.000000
max      60.000000    NaN        NaN            NaN        8.000000

        Sleep_Quality Wake_Up_Time Bed_Time  Physical_Activity  Screen_Time  \
count      773.000000          773      773         773.000000   773.000000
unique            NaN           10       19                NaN          NaN
top               NaN      7:00 AM  6:00 PM                NaN          NaN
freq              NaN          192      143                NaN          NaN
mean         3.848124          NaN      NaN           2.979301     4.105433
std          0.545459          NaN      NaN           0.797234     0.812513
min          2.000000          NaN      NaN           1.000000     2.000000
25%          3.600000          NaN      NaN           2.000000     4.000000
50%          3.900000          NaN      NaN           3.000000     4.000000
75%          4.000000          NaN      NaN           4.000000     5.000000
max          5.000000          NaN      NaN           5.000000     8.000000

        ...  Smoking_Habit  Work_Hours Travel_Time  Social_Interactions  \
count   ...            773  773.000000  773.000000           773.000000
unique  ...              2         NaN         NaN                  NaN
top     ...            Yes         NaN         NaN                  NaN
freq    ...            415         NaN         NaN                  NaN
mean    ...            NaN    8.258732    2.858344             3.196636
std     ...            NaN    1.064168    1.083758             0.856332
min     ...            NaN    6.000000    0.500000             1.000000
25%     ...            NaN    8.000000    2.000000             3.000000
50%     ...            NaN    8.000000    3.000000             3.000000
75%     ...            NaN    9.000000    4.000000             4.000000
max     ...            NaN   14.000000    5.000000             5.000000
```

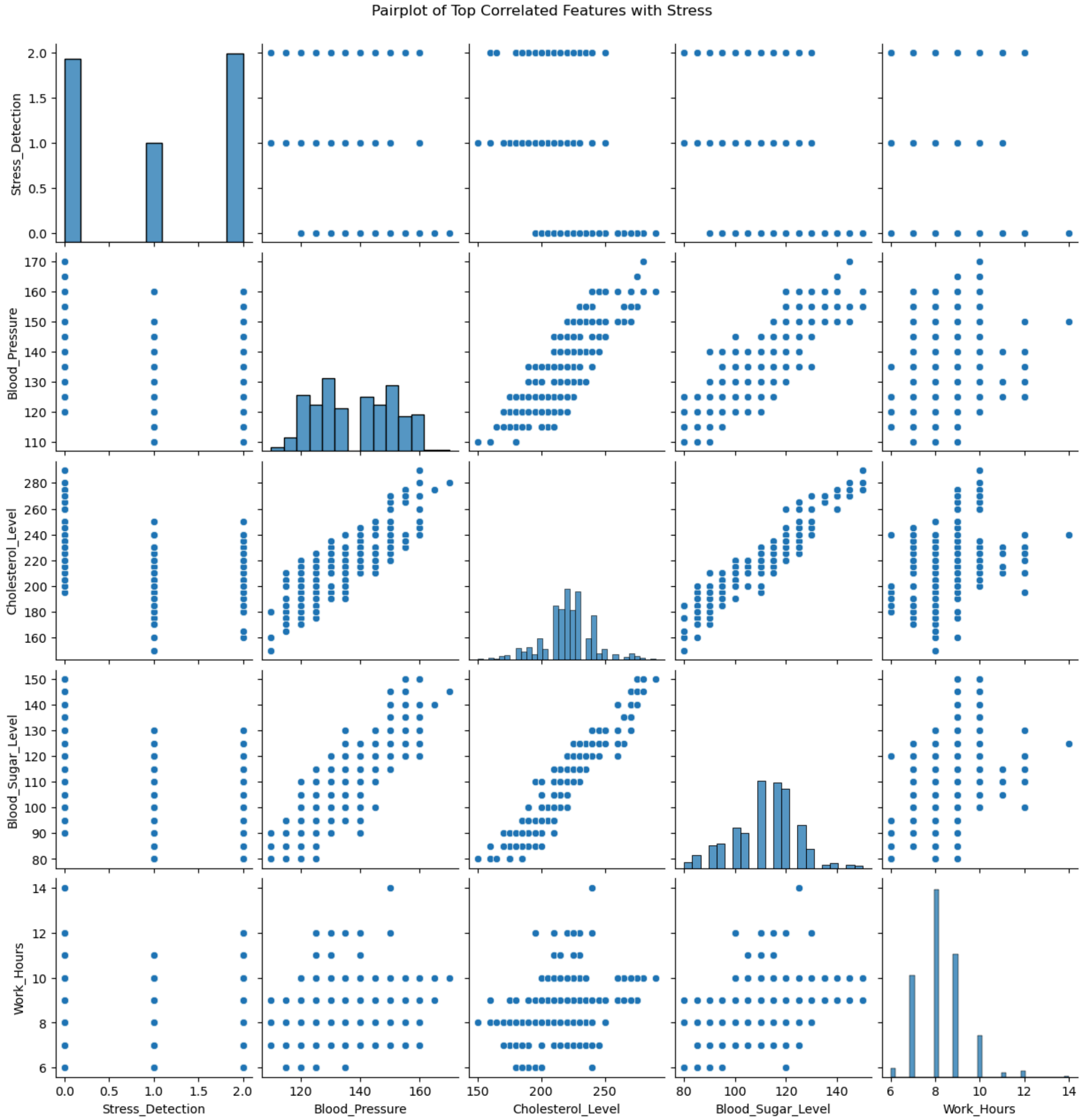
```
        Meditation_Practice      Exercise_Type Blood_Pressure  \
count                   773                773     773.000000   
unique                    2                  7            NaN   
top                     Yes  Strength Training            NaN   
freq                    481                245            NaN   
mean                    NaN                NaN     137.943079   
std                     NaN                NaN      13.122060   
min                     NaN                NaN     110.000000   
25%                     NaN                NaN     130.000000   
50%                     NaN                NaN     140.000000   
75%                     NaN                NaN     150.000000   
max                     NaN                NaN     170.000000   

       Cholesterol_Level  Blood_Sugar_Level  Stress_Detection  
count         773.000000         773.000000               773  
unique               NaN                NaN                 3  
top                  NaN                NaN            Medium  
freq                 NaN                NaN               310  
mean          220.834411         111.765847               NaN  
std            19.322622          12.533097               NaN  
min           150.000000          80.000000               NaN  
25%           210.000000         105.000000               NaN  
50%           220.000000         115.000000               NaN  
75%           230.000000         120.000000               NaN  
max           290.000000         150.000000               NaN  

[11 rows x 22 columns]

Value counts for Gender:
Gender
1    389
0    384
Name: count, dtype: int64

Value counts for Occupation:
Occupation
157    37
21     26
102    23
110    22
69     18
       ..
57      1
79      1
135     1
127     1
121     1
Name: count, Length: 169, dtype: int64

Value counts for Marital_Status:
Marital_Status
1    360
2    353
0     60
Name: count, dtype: int64
```

```
Value counts for Smoking_Habit:
Smoking_Habit
1    415
0    358
Name: count, dtype: int64

Value counts for Meditation_Practice:
Meditation_Practice
1    481
0    292
Name: count, dtype: int64

Value counts for Exercise_Type:
Exercise_Type
4    245
1    215
6    191
3     88
2     24
0      5
5      5
Name: count, dtype: int64

Value counts for Stress_Detection:
Stress_Detection
2    310
0    301
1    162
Name: count, dtype: int64

=== Correlation with Stress Level ===
Stress_Detection       1.000000
Marital_Status         0.166686
Wake_Up_Time           0.159234
Occupation             0.106155
Sleep_Duration         0.100349
Exercise_Type          0.028427
Bed_Time               0.011710
Sleep_Quality         -0.063091
Gender                -0.129601
Social_Interactions   -0.181557
Age                   -0.211412
Smoking_Habit         -0.262163
Work_Hours            -0.280962
Alcohol_Intake        -0.306902
Meditation_Practice   -0.313961
Blood_Pressure        -0.316430
Travel_Time           -0.317396
Caffeine_Intake       -0.349747
Blood_Sugar_Level     -0.371895
Cholesterol_Level     -0.373976
Screen_Time           -0.379847
Physical_Activity     -0.380487
Name: Stress_Detection, dtype: float64
```


Distribution of Stress Levels
count
300
250
200
150
100
50
0
Low
Medium
High
Stress_Detection


Top Features Correlated with Stress Level
Stress_Detection
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Marital_Status
Wake_Up_Time
Occupation
Sleep_Duration
Exercise_Type
None


Stress Levels by Top 10 Occupations
Stress_Detection
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
5
21
50
69
86
102
110
141
150
157
Occupation




```
=== Average Values by Stress Level ===
                        Age  Sleep_Duration  Sleep_Quality  Physical_Activity  \
Stress_Detection
0                 41.485050        6.164784       3.878738           3.435216
1                 36.123457        6.671605       3.880247           2.570988
2                 37.809677        6.332903       3.801613           2.750000

                  Screen_Time  Work_Hours  Blood_Pressure  Cholesterol_Level  \
Stress_Detection
0                    4.599668    8.647841      145.348837         232.956811
1                    3.577160    8.080247      128.024691         206.419753
2                    3.901613    7.974194      135.935484         216.596774

                  Blood_Sugar_Level
Stress_Detection
0                        119.335548
1                        103.395062
2                        108.790323
```