Imports

In [8]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
import numpy as np
```

Data Loading

In [3]:

```
# Load the datasets
labels_df = pd.read_csv('Dataset/Labels.csv')
physiology_df = pd.read_csv('Dataset/Physiology.csv')
sleep_df = pd.read_csv('Dataset/Sleep.csv')
```

Data Preprocessing

In [4]:

```
# Function to extract date-time features
def extract_date_features(df, date_column):
    df[date_column] = pd.to_datetime(df[date_column])
    df['year'] = df[date_column].dt.year
    df['month'] = df[date_column].dt.month
    df['day'] = df[date_column].dt.day
    df['hour'] = df[date_column].dt.hour
    df['minute'] = df[date_column].dt.minute
    df['day_of_week'] = df[date_column].dt.dayofweek
    return df

# Extracting date-time features for each dataset
labels_df = extract_date_features(labels_df, 'date')
physiology_df = extract_date_features(physiology_df, 'date')
sleep_df = extract_date_features(sleep_df, 'date')
```

In [9]:

```
# Data Integration
physiology_df['date_only'] = physiology_df['date'].dt.date
sleep_df['date_only'] = sleep_df['date'].dt.date
numeric_cols_physiology = physiology_df.select_dtypes(include=[np.number]).columns.tolist()
physiology_daily = physiology_df.groupby(['patient_id', 'date_only'])[numeric_cols_physiology].mean().reset_index()
sleep_df['snoring'] = sleep_df['snoring'].astype(int)
numeric_cols_sleep = sleep_df.select_dtypes(include=[np.number]).columns.tolist()
sleep_daily = sleep_df.groupby(['patient_id', 'date_only'])[numeric_cols_sleep].mean().reset_index()
labels_agitation = labels_df[labels_df['type'] == 'Agitation'].copy()
labels_agitation['date_only'] = labels_agitation['date'].dt.date
merged_data = labels_agitation.merge(physiology_daily, on=['patient_id', 'date_only'], how='left')
merged_data = merged_data.merge(sleep_daily, on=['patient_id', 'date_only'], how='left')
```

In [10]:

```
# Preparing Binary Classification Dataset
num_agitated = len(labels_agitation)
physiology_non_agitated_sample = physiology_daily.sample(n=num_agitated, random_state=0)
sleep_non_agitated_sample = sleep_daily.sample(n=num_agitated, random_state=0)
non_agitated_combined = physiology_non_agitated_sample.merge(sleep_non_agitated_sample, o
n=['patient_id', 'date_only'], how='inner')
merged_data['label'] = 1  # Agitated
non_agitated_combined['label'] = 0  # Non-agitated
binary_classification_dataset = pd.concat([merged_data, non_agitated_combined])
binary_classification_dataset = binary_classification_dataset.sample(frac=1, random_state
=0).reset_index(drop=True)
```

Creating various scenarios after noticing data imbalance

In [11]:

```
# Balanced Dataset (1:1 ratio)
agitated = binary_classification_dataset[binary_classification_dataset['label'] == 1]
non_agitated = binary_classification_dataset[binary_classification_dataset['label'] == 0
]
non_agitated_upsampled = resample(non_agitated, replace=True, n_samples=len(agitated), r
andom_state=0)
balanced_dataset = pd.concat([agitated, non_agitated_upsampled])
balanced_dataset = balanced_dataset.sample(frac=1, random_state=0).reset_index(drop=True
)

# Scenario: 1:2 ratio
non_agitated_upsampled_1_2 = resample(non_agitated, replace=True, n_samples=2 * len(agit
ated), random_state=0)
dataset_1_2 = pd.concat([agitated, non_agitated_upsampled_1_2]).sample(frac=1, random_st
ate=0).reset_index(drop=True)

# Scenario: 1:3 ratio
non_agitated_upsampled_1_3 = resample(non_agitated, replace=True, n_samples=3 * len(agit
ated), random_state=0)
dataset_1_3 = pd.concat([agitated, non_agitated_upsampled_1_3]).sample(frac=1, random_st
ate=0).reset_index(drop=True)
```

In [19]:

```
# Data Cleaning and Preprocessing
columns_to_drop = ['year_x', 'month_x', 'day_x', 'hour_x', 'minute_x', 'day_of_week_x',
                   'year_y', 'month_y', 'day_y', 'hour_y', 'minute_y', 'day_of_week_y']
balanced_dataset_cleaned = balanced_dataset.drop(columns=columns_to_drop)
balanced_dataset_cleaned = balanced_dataset_cleaned.fillna(balanced_dataset_cleaned.mean(
numeric_only=True))

# Ensure the features are all numeric for the balanced dataset
X_balanced = balanced_dataset_cleaned.select_dtypes(include=[np.number])
y_balanced = balanced_dataset_cleaned['label']

# Splitting the balanced dataset into training and testing sets
X_train_balanced, X_test_balanced, y_train_balanced, y_test_balanced = train_test_split(
    X_balanced, y_balanced, test_size=0.3, random_state=0)

# ... [Other preprocessing steps, if any]

# Ensure the features are all numeric for the 1:2 ratio dataset
X_1_2 = dataset_1_2.select_dtypes(include=[np.number])
y_1_2 = dataset_1_2['label']

# Splitting the 1:2 ratio dataset into training and testing sets
X_train_1_2, X_test_1_2, y_train_1_2, y_test_1_2 = train_test_split(
    X_1_2, y_1_2, test_size=0.3, random_state=0)

# Ensure the features are all numeric for the 1:3 ratio dataset
X_1_3 = dataset_1_3.select_dtypes(include=[np.number])
y_1_3 = dataset_1_3['label']
```

```
# Splitting the 1:3 ratio dataset into training and testing sets
X_train_1_3, X_test_1_3, y_train_1_3, y_test_1_3 = train_test_split(
    X_1_3, y_1_3, test_size=0.3, random_state=0)
```

Model Training and Evaluation

In [20]:

```
# Function to train and evaluate a model
def train_and_evaluate_model(X_train, y_train, X_test, y_test, model, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred)
    print(f"Classification Report for {model_name}:\n{report}\n")

# Models to be trained
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=0),
    "Random Forest": RandomForestClassifier(random_state=0),
    "Gradient Boosting (XGBoost)": XGBClassifier(use_label_encoder=False, eval_metric='lo
gloss', random_state=0),
    "Neural Network (MLP)": MLPClassifier(max_iter=1000, random_state=0)
}
```

In [21]:

```
# Training and evaluating models
print("Results for Balanced Dataset:")
for model_name, model in models.items():
    train_and_evaluate_model(X_train, y_train, X_test, y_test, model, model_name)

# Training and evaluating models on the 1:2 ratio dataset
print("Results for Dataset with 1:2 ratio:")
for model_name, model in models.items():
    train_and_evaluate_model(X_train_1_2, y_train_1_2, X_test_1_2, y_test_1_2, model, mo
del_name)

# Training and evaluating models on the 1:3 ratio dataset
print("Results for Dataset with 1:3 ratio:")
for model_name, model in models.items():
    train_and_evaluate_model(X_train_1_3, y_train_1_3, X_test_1_3, y_test_1_3, model, mo
del_name)
```

```
Results for Balanced Dataset:

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13840\3416114643.py in ?()
      1 # Training and evaluating models
      2 print("Results for Balanced Dataset:")
      3 for model_name, model in models.items():
----> 4     train_and_evaluate_model(X_train, y_train, X_test, y_test, model, model_name
)
      5 
      6 # Training and evaluating models on the 1:2 ratio dataset
      7 print("Results for Dataset with 1:2 ratio:")

~\AppData\Local\Temp\ipykernel_13840\2224343044.py in ?(X_train, y_train, X_test, y_test,
model, model_name)
      2 def train_and_evaluate_model(X_train, y_train, X_test, y_test, model, model_name)
:
----> 3     model.fit(X_train, y_train)
      4     y_pred = model.predict(X_test)
      5     report = classification_report(y_test, y_pred)
      6     print(f"Classification Report for {model_name}:\n{report}\n")

~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\lo
cal-packages\Python311\site-packages\sklearn\base.py in ?(estimator, *args, **kwargs)
   1148                 skip_parameter_validation=(
   1149                     prefer_skip_nested_validation or global_skip_validation
   1150                 )
   1150                         )
   1151                     ):
-> 1152                         return fit_method(estimator, *args, **kwargs)

~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\lo
cal-packages\Python311\site-packages\sklearn\linear_model\_logistic.py in ?(self, X, y, s
ample_weight)
   1204                 _dtype = np.float64
   1205             else:
   1206                 _dtype = [np.float64, np.float32]
   1207 
-> 1208             X, y = self._validate_data(
   1209                 X,
   1210                 y,
   1211                 accept_sparse="csr",

~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\lo
cal-packages\Python311\site-packages\sklearn\base.py in ?(self, X, y, reset, validate_sep
arately, cast_to_ndarray, **check_params)
    618                     if "estimator" not in check_y_params:
    619                         check_y_params = {**default_check_params, **check_y_params}
    620                     y = check_array(y, input_name="y", **check_y_params)
    621                 else:
--> 622                     X, y = check_X_y(X, y, **check_params)
    623                 out = X, y
    624 
    625             if not no_val_X and check_params.get("ensure_2d", True):

~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\lo
cal-packages\Python311\site-packages\sklearn\utils\validation.py in ?(X, y, accept_sparse
, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_o
utput, ensure_min_samples, ensure_min_features, y_numeric, estimator)
   1142             raise ValueError(
   1143                 f"{estimator_name} requires y to be passed, but the target y is None"
   1144             )
   1145 
-> 1146         X = check_array(
   1147             X,
   1148             accept_sparse=accept_sparse,
   1149             accept_large_sparse=accept_large_sparse,

~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\lo
cal-packages\Python311\site-packages\sklearn\utils\validation.py in ?(array, accept_spars
e, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure
_min_samples, ensure_min_features, estimator, input_name)
    912                         )
    913                     array = xp.astype(array, dtype, copy=False)
    914                 else:
    915                     array = _asarray_with_order(array, order=order, dtype=dtype,
xp=xp)
--> 916             except ComplexWarning as complex_warning:
    917                 raise ValueError(
    918                     "Complex data not supported\n{}\n".format(array)
    919                 ) from complex_warning

~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\lo
cal-packages\Python311\site-packages\sklearn\utils\_array_api.py in ?(array, dtype, order
, copy, xp)
    376         # Use NumPy API to support order
    377         if copy is True:
    378             array = numpy.array(array, order=order, dtype=dtype)
    379         else:
--> 380             array = numpy.asarray(array, order=order, dtype=dtype)
    381 
    382         # At this point array is a NumPy ndarray. We convert it to an array
    383         # container that is consistent with the input's namespace.

~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\lo
cal-packages\Python311\site-packages\pandas\core\generic.py in ?(self, dtype)
   2082     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
   2083         values = self._values
-> 2084         arr = np.asarray(values, dtype=dtype)
   2085         if (
   2085         if (
   2086             astype_is_view(values.dtype, arr.dtype)
   2087             and using_copy_on_write()

ValueError: could not convert string to float: '0d5ef'
```