Identifying Risk Factors for Individual Somatic Symptoms

Miao Yu, Jessica Tanchone

2025-04-10

Table of contents

1 Introduction

The present study investigates psychosocial and demographic correlates of 13 somatic symptoms that were selected and assessed by the authors of the publicly available EAMMi2 dataset (Grahe et al., 2018). These symptoms—such as fatigue, dizziness, chest pain, shortness of breath, and fainting spells—were measured using a standardized self-report scale in which participants rated how much they were bothered by each symptom in the past week. While we do not know the rationale behind the selection of these specific symptoms, our goal is to build predictive models using the available data to explore how psychological and demographic factors may be associated with each symptom. Some of these symptoms, such as shortness of breath or chest pain, are often linked to well-established physical causes, particularly in clinical contexts. However, other symptoms—such as fatigue, sleep problems, and headaches—may be more closely tied to psychosocial stressors. We hypothesize that the strength of association with psychological predictors will vary by symptom, with fatigue showing stronger psychosocial correlates and fainting spells showing weaker ones.

2 Literature Reivew

2.1 Somatic Symptoms

Somatic symptoms—such as fatigue, dizziness, chest pain, and gastrointestinal discomfort—are common physical complaints that may not have a clear biomedical cause but are often associated with psychological

distress. These symptoms are frequently observed in both clinical and community populations and are a frequent reason why individuals seek medical care. However, their presence can complicate diagnosis and treatment, especially when they are misattributed solely to physical illness.

Recent research emphasizes the importance of identifying persistent somatic symptoms (PSS) early, particularly in primary care settings. Kitselaar et al. (2023) reported that approximately 10% of adults experience PSS, which are not fully explained by a medical condition. In general practice, up to 50% of consultations involve symptoms that cannot be clearly linked to physical disease. These symptoms are often self-limiting, but for some individuals, they become chronic and burdensome, both personally and within healthcare systems. Patients with PSS tend to have higher consultation rates and are more likely to experience mental health conditions, such as anxiety and depression (Kitselaar et al., 2023).

The fifth edition of the Diagnostic and Statistical Manual of Mental Disorders (DSM-5) redefined these conditions under the category of somatic symptom and related disorders, shifting focus from the absence of medical explanation to the presence of excessive thoughts, feelings, or behaviors related to the symptoms (Kurlansik & Maffei, 2016). This change helps reduce stigma and better aligns with how these symptoms are experienced in real-world clinical settings. Research shows that individuals with somatic symptom disorder often have a heightened awareness of bodily sensations and may interpret them as signs of serious illness, especially under stress. Risk factors for persistent symptoms include childhood adversity, substance use, and comorbid mental health issues (Kurlansik & Maffei, 2016).

Somatic symptoms are also closely tied to depression. In a large international study, Simon et al. (1999) found that 69% of patients with major depression presented with only physical symptoms, such as back pain or fatigue, rather than reporting emotional distress. This was especially common in settings where patients did not have a regular relationship with a physician. The study suggested that cultural and healthcare system differences influence how psychological distress is expressed, with somatic symptoms often serving as a socially acceptable way to seek help.

2.1.0.1 Symptom 1: Stomach Pain

2.1.0.2 Symptom 2: Back pain

2.1.0.3 Symptom 3: Pain in your arms, legs, or joints (knees, hips, etc).

2.1.0.4 Symptom 4: Headaches

2.1.0.5 Symptom 5: Chest pain

2.1.0.6 Symptom 6: Dizziness

2.1.0.7 Symptom 7: Fainting spell

Fainting spells (syncope) represent transient episodes of loss of consciousness and have historically been interpreted as potential indicators of underlying neurological or cardiovascular dysfunction. Early psychiatric research suggested no significant association between electroencephalographic (EEG) abnormalities and the frequency, duration, or severity of fainting episodes (American Journal of Psychiatry, 1945). Similarly, clinical evidence indicates that syncope does not consistently predict hemodynamic instability or poor cardiac outcomes among patients with pulmonary embolism (Pawar et al., 2025). Specifically, syncope was not associated with lower cardiac index or elevated pulmonary artery pressures, challenging its role as a reliable clinical marker of severity.

In addition, earlier research has emphasized that psychological factors may play a critical role in fainting spells, particularly in vasovagal (neurocardiogenic) syncope. Gracie et al. (1995) highlighted that anxiety, health-related worry, and depressive symptoms are common among patients with vasovagal syncope.

Psychological distress can increase the frequency of episodes and lead to avoidance behaviors that negatively impact quality of life. Moreover, addressing these psychological factors through interventions such as cognitive-behavioral therapy has been shown to improve symptom management and reduce fear of recurrence.

Collectively, these findings suggest that fainting spells may not be fully explained by physiological or structural abnormalities alone, underscoring the potential relevance of psychosocial and behavioral factors. In this context, the present study aims to explore whether psychological and demographic characteristics are associated with self-reported fainting spells among young adults. Although prior literature indicates that such associations may be weak or non-significant, investigating these relationships may provide novel insights into the broader psychosomatic mechanisms underlying somatic symptom experiences.

Given the small number of participants endorsing fainting spells, and supported by recent evidence suggesting a shared underlying sensitivity across related symptoms, we combined dizziness and fainting spells into a single outcome. Lukacova et al. (2023) found that susceptibility to visually induced motion sickness was significantly correlated with both dizziness and syncope, along with migraine and vertigo. These symptoms loaded onto a single latent factor reflecting a common sensitivity construct, supporting the view that dizziness and fainting may reflect a continuum of perceptual or vestibular vulnerability rather than discrete clinical categories.

2.1.0.8 Symptom 8: Heart palpitations

Recent evidence from coronary heart disease patients highlights that psychosocial risk profiles, shaped by both gender traits and demographic factors, play a critical role in symptom experience and reporting (van den Houdt et al., 2024).

symptom 9: Shortness of breath

Previous research has shown that shortness of breath is more commonly associated with identifiable physical conditions, especially in emergency department settings. For instance, Zachariah et al. (2017) reported that the most common diagnosis for patients aged 18–44 presenting with dyspnea was acute asthma exacerbation (14.8%), while obstructive chronic bronchitis predominated in middle-aged and older adults (11.1% for ages 45–64; 12.4% for ages 65–79). Among patients aged 80 and over, congestive heart failure was the most frequently diagnosed condition (15.9%) Zachariah et al., 2017.

Shortness of breath is a common somatic symptom that may signal serious underlying medical conditions, particularly cardiac dysfunction. In a prospective emergency department study of 251 patients presenting with dyspnea, Dieplinger et al. (2009) evaluated ten candidate biomarkers and found that BNP and MR-proANP were the only independent diagnostic indicators of acute destabilized heart failure. These two markers demonstrated high diagnostic accuracy (AUCs = 0.92 and 0.88, respectively), outperforming other biomarkers such as copeptin, ST2, and adiponectin. This finding underscores the clinical relevance of shortness of breath as a potential signal of cardiac pathology and highlights the importance of distinguishing physiologically-based symptoms from those potentially influenced by psychological or behavioral factors.

2.1.0.9 symptom 10: Constipation, loose bowels, or diarrhea

Research suggests that cognitive and behavioral factors differ meaningfully across irritable bowel syndrome (IBS) subtypes. In a study of over 500 patients with refractory IBS, Windgassen et al. (2018) found that avoidance behaviors and unhelpful gastrointestinal cognitions were significantly associated with loose/watery stools, whereas control behaviors were associated with hard/lumpy stools. Importantly, these cognitive-behavioral differences emerged even when anxiety, depression, and symptom severity were held constant, highlighting the potential for targeted psychological interventions based on symptom subtype.

2.1.0.10 symptom 11: Nausea, gas, or indigestion

Nausea and vomiting are key somatic symptoms frequently observed in both medical and non-medical populations. In their comprehensive work, Nausea: Mechanisms and Management, Stern, Koch, and Andrews (2011) detail a wide range of physiological causes including gastrointestinal disturbances (e.g., diabetic gastroparesis), postoperative complications (e.g., surgery and anesthesia), cancer treatments (e.g., chemotherapy),

Table 1: This table desplays random 2 rows of the data and transpose for eacy to read

```
data = pd.read_excel("data/EAMMi2-Data1.2.xlsx", sheet_name="EAMMi2_Data")

data.sample(2).transpose()
```

pregnancy, and motion-related triggers. Notably, the authors describe how nausea and vomiting during early pregnancy may be associated with a reduced risk of miscarriage, suggesting a possible adaptive function.

2.1.0.11 symptom 13: sleeping problem

In a large survey of 1,925 women over 50 with chronic health conditions, Meredith et al. (2020) found that 43% reported concurrent sleep problems, and that financial strain significantly increased their odds (OR = 1.61–2.84), while higher levels of physical activity were protective (OR = 0.63). This underscores the interplay between economic stress, lifestyle behaviors, and sleep-related somatic complaints in older adults.

```
!pip install pingouin
```

```
!pip install shap
```

```
!pip install imbalanced-learn
```

```
# Core packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pingouin as pg
import shap

# Display tools
from IPython.display import Markdown

# preprocessing and pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer

# models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from imblearn.over_sampling import SMOTE

# feature selection
from sklearn.feature_selection import SelectKBest, f_classif

# metrics
from sklearn.metrics import (
    classification_report, roc_auc_score, confusion_matrix,
    RocCurveDisplay, accuracy_score, roc_curve, auc
)
```

```
data.shape
```

```
codebook = pd.read_excel("data/EAMMi2-Data1.2-Codebook.xlsx").set_index('Variable Name')
```

```
# Recode sibling variable
data['sibling_c'] = data['sibling'].apply(lambda x: -0.5 if x == 1 else 0.5)

# Rename and process marriage variables
data = data.rename(columns={'marriage2': 'marriage_importance', 'marriage5': 'parental_marriage'})
data = pd.concat([data, pd.get_dummies(data['parental_marriage'].astype('category'),
                                       prefix='parental_marriage', drop_first=True)], axis=1)

# Compute scale means
scales = {
    'idea_m': [f'IDEA_{i}' for i in range(1, 9)],
    'moa_achievement_m': [f'moa1#2_{i}' for i in range(1, 11)] + [f'moa2#1_{i}' for i in range(1, 11)],
    'moa_importance_m': [f'moa2#1_{i}' for i in range(1, 11)] + [f'moa2#2_{i}' for i in range(1, 11)],
    'stress_m': [f'stress_{i}' for i in range(1, 11)],
    'support_m': [f'support_{i}' for i in range(1, 13)],
    'belong_m': [f'belong_{i}' for i in range(1, 11)],
    'mindful_m': [f'mindful_{i}' for i in range(1, 16)],
    'efficacy_m': [f'efficacy_{i}' for i in range(1, 11)],
    'npi_m': [f'NPI{i}' for i in range(1, 14)],
    'exploit_m': [f'exploit_{i}' for i in range(1, 4)],
    'disability_m': [f'Q10_{i}' for i in range(1, 16)] + ['Q11'] + [f'Q14_{i}' for i in range(1, 7)],
    'social_conn_m': [f'SocMedia_{i}' for i in range(1, 6)],
    'social_new_m': [f'SocMedia_{i}' for i in range(6, 10)],
    'social_info_m': [f'SocMedia_{i}' for i in range(10, 12)],
    'swb_m': [f'swb_{i}' for i in range(1, 7)],
    'transgres_m': [f'transgres_{i}' for i in range(1, 5)],
    'usdream_m': ['usdream_1', 'usdream_2']
}

for name, items in scales.items():
    valid_items = [item for item in items if item in data.columns]
    data[name] = data[valid_items].mean(axis=1, skipna=True)

# Select final columns
final_data = data[[col for col in data.columns if
                   col.endswith(('_m', '_c')) or
                   col in ['marriage_importance', 'parental_marriage'] or
                   col.startswith(('parental_marriage_', 'sex_'))]]

print("Final columns:\n", final_data.columns.tolist())
```

2.2 Variables and Measures

2.2.1 Outcome Variable

The outcome variable were assessed using item 12 from the **Patient Health Questionnaire (PHQ)**. Responses were recoded into a binary variable, where a score of 2 or higher was considered symptomatic (coded as 1), and a score of 1 indicated no significant fatigue symptoms (coded as 0). This binary formulation was chosen due to its moderate class balance and clinical relevance.

Table 2: This is the codebook for all the variables I created.

```
codebook_created = {
    'idea_m': "Mean score: Identity Exploration and Development Assessment (IDEA-8)",
    'moa_achievement_m': "Mean score: Markers of Adulthood - Achievement subscale",
    'moa_importance_m': "Mean score: Markers of Adulthood - Importance subscale",
    'stress_m': "Mean score: Perceived Stress Scale",
    'support_m': "Mean score: Perceived Social Support",
    'belong_m': "Mean score: Need to Belong Scale",
    'mindful_m': "Mean score: Mindfulness Scale",
    'efficacy_m': "Mean score: Efficacy/Competence Scale",
    'npi_m': "Mean score: Narcissistic Personality Inventory (NPI-13)",
    'exploit_m': "Mean score: Interpersonal Exploitativeness Scale",
    'disability_m': "Mean score: Disability Identity & Status items",
    'social_conn_m': "Mean score: Social Media Use - Maintaining Connections",
    'social_new_m': "Mean score: Social Media Use - Making New Connections",
    'social_info_m': "Mean score: Social Media Use - Information Seeking",
    'swb_m': "Mean score: Subjective Well-Being",
    'transgres_m': "Mean score: Interpersonal Transgressions",
    'usdream_m': "Mean score: American Dream (Importance and Belief in Achievability)",
    'sibling_c': "Recode: -0.5 = no siblings, +0.5 = at least one sibling",
    'marriage_importance': "Importance of getting married (1-5 scale)",
    'parental_marriage': "Parental marriage status (categorical)"

}


codebook_df = pd.DataFrame({
    'Variable': list(codebook_created.keys()),
    'Description': list(codebook_created.values())
})


Markdown(codebook_df.to_markdown())
```

Table 3: This is the codebook of the variables retained without modification from the raw dataset.

```
# Get all columns that start with 'physSx_'
physSx_cols = codebook.index[codebook.index.str.startswith('physSx_')].tolist()

# Combine with other columns
selected_cols = ['sex', 'edu', 'race', 'income'] + physSx_cols

Markdown(
    codebook
    .loc[selected_cols]
    [['Question text', 'responses']]
    .to_markdown()
)
```

Table 4: "Summary Statistics"

```
stats = final_data.describe().transpose().round(3)
stats['missing'] = final_data.isnull().sum()
stats = stats[['count', 'missing', 'mean', 'std', 'min', '50%', 'max']]
stats
```

2.2.2 Psychological Predictors

All psychological variables were computed as mean scores across item-level responses, provided that a sufficient proportion of items (70%) were present. The following scales were included:

- **IDEA-8**: Identity exploration and development (8 items)
- **Markers of Adulthood**:
  - Achievement subscale (20 items)
  - Importance subscale (20 items)
- **Perceived Stress Scale** (10 items)
- **Perceived Social Support** (12 items)
- **Need to Belong Scale** (10 items)
- **Mindfulness Scale** (15 items)
- **Self-Efficacy/Competence** (10 items)
- **Narcissistic Personality Inventory (NPI-13)** (13 items)
- **Interpersonal Exploitativeness** (3 items)
- **Disability Identity and Status** (combination of 22 items from Q10, Q11, and Q14)
- **Social Media Use**:
  - Maintaining connections (5 items)
  - Making new connections (4 items)
  - Seeking information (2 items)
- **Subjective Well-Being** (6 items)
- **Interpersonal Transgressions** (4 items)
- **Belief in and importance of the American Dream** (2 items)

2.2.3 Demographic and Attitudinal Predictors

- **Sex** (male, female, other; categorical)
- **Education level**
- **Race/Ethnicity**
- **Household income**
- **School attended**
- **Parental marriage status** (recoded into categorical dummy variables)
- **Siblings** (recoded: -0.5 = no siblings, 0.5 = at least one sibling)
- **Importance of marriage** (single-item rating)

These variables represent a theoretically grounded and diverse set of predictors for modeling somatic symptom risk, with particular focus on psychological and social functioning within a young adult population.

```
# Get physSx_ columns
physSx_data = data.filter(regex='^physSx_')

value_counts_dict = {}
for col in physSx_data.columns:
    counts = data[col].value_counts()
    percentages = data[col].value_counts(normalize=True) * 100
    value_counts_dict[col] = counts.astype(str) + ' (' + percentages.round(1).astype(str) + '%)'

table = pd.DataFrame(value_counts_dict).fillna('')
```

```
table
```

3 EDA

```
# Histograms for numeric variables
numeric_vars = final_data.select_dtypes(include=['float64', 'int64']).columns
final_data[numeric_vars].hist(figsize=(16, 14), bins=20)
plt.suptitle("Distribution of Numeric Variables", fontsize=16)
plt.tight_layout()
plt.show()
```

Figure 1

As shown in Figure 1, the dataset includes both continuous and ordinal variables stored in numeric format. Several psychological scales—such as mindfulness, self-efficacy, and subjective well-being—exhibit approximately normal or symmetric distributions, supporting their use in regression-based modeling.

In contrast, other variables demonstrate notable skew. For example, exploitative tendencies, interpersonal transgressions, and social information-seeking show strong right-skew, indicating a concentration of lower scores across participants. These distributions may require transformation or interpretation with caution in downstream analysis.

Importantly, several variables—such as education level, household income, and marriage importance—are ordinal Likert-type items encoded as numeric. While represented as continuous values in the dataset, these variables reflect discrete, ordered categories and may not satisfy assumptions of linearity or interval scale interpretation.

Together, the histograms highlight the variability in scale characteristics, underscoring the need to carefully consider variable type and distribution when specifying predictive models.

```
# heatmap
# Add all physSx_ columns to final_data
for col in physSx_data.columns:
    final_data[col] = data[col]

# Create selected_vars with numeric_vars + all physSx_ columns
selected_vars = list(numeric_vars) + list(physSx_data.columns)

plt.figure(figsize=(24,20))
sns.heatmap(final_data[selected_vars].corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix (Numeric Predictors)", fontsize=14)
plt.show()
```

Figure 2

```
moa1_cols = [col for col in data.columns if col.startswith('moa1')]
alpha = pg.cronbach_alpha(data=data[moa1_cols].dropna())
print(f"Markers of Adulthood Importance scale Cronbach's Alpha: {alpha[0]:.3f}")

moa2_cols = [col for col in data.columns if col.startswith('moa2')]
alpha = pg.cronbach_alpha(data=data[moa2_cols].dropna())
print(f"Markers of Adulthood Achievement scale Cronbach's Alpha: {alpha[0]:.3f}")

IDEA_cols = [col for col in data.columns if col.startswith('IDEA_')]
```

```
alpha = pg.cronbach_alpha(data=data[IDEA_cols].dropna())
print(f"IDEA-8 Cronbach's Alpha: {alpha[0]:.3f}")

swb_cols = [col for col in data.columns if col.startswith('swb_')]
alpha = pg.cronbach_alpha(data=data[swb_cols].dropna())
print(f"Subjective Wellbeing Scale Cronbach's Alpha: {alpha[0]:.3f}")

mindful_cols = [col for col in data.columns if col.startswith('mindful_')]
alpha = pg.cronbach_alpha(data=data[mindful_cols].dropna())
print(f"Mindfulness Scale Cronbach's Alpha: {alpha[0]:.3f}")

belong_cols = [col for col in data.columns if col.startswith('belong_')]
alpha = pg.cronbach_alpha(data=data[belong_cols].dropna())
print(f"Need to Belong Scale Cronbach's Alpha: {alpha[0]:.3f}")

efficacy_cols = [col for col in data.columns if col.startswith('efficacy_')]
alpha = pg.cronbach_alpha(data=data[efficacy_cols].dropna())
print(f"Efficacy Scale Cronbach's Alpha: {alpha[0]:.3f}")

support_cols = [col for col in data.columns if col.startswith('support_')]
alpha = pg.cronbach_alpha(data=data[support_cols].dropna())
print(f"Perceived Social Support Scale Cronbach's Alpha: {alpha[0]:.3f}")

SocMedia_cols = [col for col in data.columns if col.startswith('SocMedia_')]
alpha = pg.cronbach_alpha(data=data[SocMedia_cols].dropna())
print(f"Social Media Use Cronbach's Alpha: {alpha[0]:.3f}")

usdream_cols = [col for col in data.columns if col.startswith('usdream_')]
alpha = pg.cronbach_alpha(data=data[usdream_cols].dropna())
print(f"US Dream Scale Cronbach's Alpha: {alpha[0]:.3f}")

transgres_cols = [col for col in data.columns if col.startswith('transgres_')]
alpha = pg.cronbach_alpha(data=data[transgres_cols].dropna())
print(f"Interpersonal Transgressions  Scale Cronbach's Alpha: {alpha[0]:.3f}")

NPI_cols = [col for col in data.columns if col.startswith('NPI')]
alpha = pg.cronbach_alpha(data=data[NPI_cols].dropna())
print(f"Narcissistic Personality Inventory-13 Cronbach's Alpha: {alpha[0]:.3f}")

stress_cols = [col for col in data.columns if col.startswith('stress')]
alpha = pg.cronbach_alpha(data=data[stress_cols].dropna())
print(f"Perceived Stress Scale Cronbach's Alpha: {alpha[0]:.3f}")

marriage_cols = [col for col in data.columns if col.startswith('marriage')]
alpha = pg.cronbach_alpha(data=data[marriage_cols].dropna())
print(f"Marriage Opinions Scale Cronbach's Alpha: {alpha[0]:.3f}")

phys_cols = [col for col in data.columns if col.startswith('physSx_')]
alpha = pg.cronbach_alpha(data=data[phys_cols].dropna())
print(f"Physical Symptom Cronbach's Alpha: {alpha[0]:.3f}")

exploit_cols = [col for col in data.columns if col.startswith('exploit')]
alpha = pg.cronbach_alpha(data=data[exploit_cols].dropna())
```

```
print(f"Interpersonal Exploitativeness Scale Cronbach's Alpha: {alpha[0]:.3f}")
```

3.1 Split Data

```
# Get physSx_ column names
physSx_cols = [col for col in final_data.columns if col.startswith('physSx_')]

# Create X by dropping all physSx_ columns
X = final_data.drop(columns=physSx_cols, errors='ignore')

# Store all train/test splits
train_test_data = {}

# Loop through each physSx_ variable as dependent variable
for physSx_var in physSx_cols:
    print(f"\nPreparing data for: {physSx_var}")

    # Create y for current physSx_ variable
    y = data[physSx_var]

    # Combine X and y, drop missing values
    model_data = X.join(y.rename(physSx_var)).dropna()

    # Separate X and y for modeling
    X_clean = model_data.drop(columns=[physSx_var])
    y_clean = model_data[physSx_var]

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        X_clean, y_clean,
        test_size=0.3,
        random_state=45,
        stratify=y_clean
    )

    # Store the splits
    train_test_data[physSx_var] = {
        'X_train': X_train,
        'X_test': X_test,
        'y_train': y_train,
        'y_test': y_test,
        'X_clean': X_clean
    }

    print(f"Train shape: {X_train.shape}")
    print(f"Test shape: {X_test.shape}")
    print("Target distribution:")
    print(y_clean.value_counts(normalize=True))

print(f"\nData preparation complete for {len(train_test_data)} physSx variables")
```

4 Result

4.1 Model Accuracy and Comparison

```
# Define features once
numeric_features = [col for col in X.columns if col.endswith('_m') or col.endswith('_c')]
categorical_features = [col for col in ['sex', 'edu', 'race', 'income', 'parental_marriage'] if col in X
categorical_features = [col for col in categorical_features if col in X_clean.columns]

# Convert categorical variables to string (important!)
for col in categorical_features:
  X_train[col] = X_train[col].astype(str)
  X_test[col] = X_test[col].astype(str)

# Preprocessing pipelines
scaled_preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
])

unscaled_preprocessor = ColumnTransformer(transformers=[
    ('num', 'passthrough', numeric_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
])

# Updated models with class_weight='balanced'
pipelines = {
    'Logistic Regression': Pipeline([
        ('preprocessor', scaled_preprocessor),
        ('classifier', LogisticRegression(max_iter=1000, class_weight='balanced'))
    ]),
    'KNN': Pipeline([
        ('preprocessor', scaled_preprocessor),
        ('classifier', KNeighborsClassifier(n_neighbors=5))
    ]),
    'Random Forest': Pipeline([
        ('preprocessor', unscaled_preprocessor),
        ('classifier', RandomForestClassifier(
            n_estimators=100,
            class_weight='balanced',
            random_state=42
        ))
    ])
}
```

```
# Loop through each physSx_ variable for modeling
for physSx_var, data_splits in train_test_data.items():
    print(f"\n{'='*60}")
    print(f"Running models for: {physSx_var}")
    print(f"{'='*60}")
    # Fit & evaluate
    for name, pipeline in pipelines.items():
        pipeline.fit(X_train, y_train)
        y_pred = pipeline.predict(X_test)
```

```
        acc = accuracy_score(y_test, y_pred)
        print(f"\n{name} Accuracy: {acc:.3f}")
        print(classification_report(y_test, y_pred))
```

4.2 Logistic Regression Model

```
# Initialize results storage
results_list = []
fitted_models = {}

for physSx_var, data_splits in train_test_data.items():
    # Extract the data splits for this specific physSx variable
    X_train = data_splits['X_train'].copy()
    X_test = data_splits['X_test'].copy()
    y_train = data_splits['y_train']
    y_test = data_splits['y_test']

    # Convert categorical variables to string
    for col in categorical_features:
        X_train[col] = X_train[col].astype(str)
        X_test[col] = X_test[col].astype(str)

    grid = GridSearchCV(
        Pipeline([
            ('preprocessor', scaled_preprocessor),
            ('classifier', LogisticRegression(max_iter=1000, class_weight='balanced', solver='lbfgs'))
        ]),
        param_grid={'classifier__C': [0.01, 0.1, 1, 10]},
        cv=5,
        scoring='f1_macro'
    )
    grid.fit(X_train, y_train)
    # Store the fitted model
    fitted_models[physSx_var] = grid

    results_list.append({
        'Variable': physSx_var,
        'Best_C': grid.best_params_['classifier__C'],
        'Best_F1_Score': round(grid.best_score_, 3)
    })

# Create DataFrame after the loop
lr_results = pd.DataFrame(results_list)
lr_results = lr_results.sort_values('Best_F1_Score', ascending=False)
lr_results
```

```
eval_results = []

for physSx_var, data_splits in train_test_data.items():
   X_test = data_splits['X_test'].copy()
   y_test = data_splits['y_test']

   # Convert categorical to string
```

```
    for col in categorical_features:
        X_test[col] = X_test[col].astype(str)

    # Predict and evaluate
    y_pred = fitted_models[physSx_var].best_estimator_.predict(X_test)
    report_dict = classification_report(y_test, y_pred, output_dict=True)

    eval_results.append({
        'Variable': physSx_var,
        'Precision': round(report_dict['macro avg']['precision'], 3),
        'Recall': round(report_dict['macro avg']['recall'], 3),
        'F1_Score': round(report_dict['macro avg']['f1-score'], 3)
    })

eval_df = pd.DataFrame(eval_results).sort_values('F1_Score', ascending=False)
eval_df
```

```
for physSx_var in fitted_models.keys():
    best_model = fitted_models[physSx_var].best_estimator_

    # Get feature names and coefficients
    feature_names = (numeric_features +
                     list(best_model.named_steps['preprocessor']
                          .named_transformers_['cat']
                          .get_feature_names_out(categorical_features)))

    coef = best_model.named_steps['classifier'].coef_
    if coef.shape[0] > 1:
        importance = np.mean(np.abs(coef), axis=0)
    else:
        importance = np.abs(coef[0])

    # Create dataframe and get top 15
    coef_df = pd.DataFrame({'Feature': feature_names, 'Importance': importance})
    coef_df = coef_df.nlargest(15, 'Importance')

    # Plot
    plt.figure(figsize=(10, 6))
    plt.barh(coef_df['Feature'][::-1], coef_df['Importance'][::-1])
    plt.xlabel("Feature Importance")
    plt.title(f"Top 15 Predictors for {physSx_var}")
    plt.tight_layout()
    plt.show()
```

Figure 3

4.3 Random Forest Model (Tuned with Grid Search)

```
# Initialize results storage
tree_list = []
tree_models = {}

for physSx_var, data_splits in train_test_data.items():
```

```
    # Extract the data splits for this specific physSx variable
    X_train = data_splits['X_train'].copy()
    X_test = data_splits['X_test'].copy()
    y_train = data_splits['y_train']
    y_test = data_splits['y_test']

    # Convert categorical variables to string
    for col in categorical_features:
        X_train[col] = X_train[col].astype(str)
        X_test[col] = X_test[col].astype(str)

    param_grid_rf = {
        'classifier__n_estimators': [100, 200],
        'classifier__max_depth': [3, 5, None],
        'classifier__min_samples_split': [2, 5]
        }

    grid_rf = GridSearchCV(
        Pipeline([
            ('preprocessor', unscaled_preprocessor),
             ('classifier', RandomForestClassifier(
                 random_state=42,
                 class_weight='balanced'
                 ))
             ]),
        param_grid=param_grid_rf,
        cv=5,
        scoring='f1_macro'
        )
    grid_rf.fit(X_train, y_train)

    # Store the fitted model
    tree_models[physSx_var] = grid_rf

    tree_list.append({
        'Variable': physSx_var,
        'Best_RF': grid_rf.best_params_,
        'Best_F1_Macro': round(grid_rf.best_score_, 3)
    })

# Create DataFrame after the loop
tree_results = pd.DataFrame(tree_list)
tree_results = lr_results.sort_values('Best_F1_Score', ascending=False)
tree_results
```

```
from sklearn.metrics import balanced_accuracy_score
from imblearn.pipeline import Pipeline as ImbPipeline

# Loop through each physSx_ variable for modeling
for physSx_var, data_splits in train_test_data.items():
    print(f"\n{'='*60}")
    print(f"Running models for: {physSx_var}")
    print(f"{'='*60}")
```

```
    # Extract data
    X_train = data_splits['X_train'].copy()
    X_test = data_splits['X_test'].copy()
    y_train = data_splits['y_train']
    y_test = data_splits['y_test']

    # Convert categorical to string (important!)
    for col in categorical_features:
      X_train[col] = X_train[col].astype(str)
      X_test[col] = X_test[col].astype(str)

    # Updated pipelines with SMOTE
    smote_pipelines = {
        'Logistic Regression': ImbPipeline([
            ('preprocessor', scaled_preprocessor),
            ('smote', SMOTE(random_state=42)),
            ('classifier', LogisticRegression(max_iter=1000, class_weight='balanced'))
        ]),
        'KNN': ImbPipeline([
            ('preprocessor', scaled_preprocessor),
            ('smote', SMOTE(random_state=42)),
            ('classifier', KNeighborsClassifier(n_neighbors=5))
        ]),
        'Random Forest': ImbPipeline([
            ('preprocessor', unscaled_preprocessor), # Use unscaled for Random Forest
            ('smote', SMOTE(random_state=42)),
            ('classifier', RandomForestClassifier(
                n_estimators=100,
                class_weight='balanced',
                random_state=42
            ))
        ])
    }

    markdown_outputs = []

    # Fit & evaluate
    for name, pipeline in smote_pipelines.items():
        pipeline.fit(X_train, y_train)
        y_pred = pipeline.predict(X_test)

        # Evaluate
        acc = accuracy_score(y_test, y_pred)
        bal_acc = balanced_accuracy_score(y_test, y_pred)
        report = classification_report(y_test, y_pred, output_dict=True)
        report_df = pd.DataFrame(report).transpose().round(2)
        report_table = report_df[['precision', 'recall', 'f1-score', 'support']].to_markdown()

        model_md = f"""
### {name} Evaluation for {physSx_var}
**Accuracy**: {acc:.3f}
**Balanced Accuracy**: {bal_acc:.3f}
```

```
{report_table}
"""
        markdown_outputs.append(model_md)

    for md in markdown_outputs:
        display(Markdown(md))
```

```
import matplotlib.pyplot as plt
import numpy as np

for physSx_var in tree_models.keys():
    best_model_rf = tree_models[physSx_var].best_estimator_

    # Get feature names
    feature_names = (numeric_features +
                     list(best_model_rf.named_steps['preprocessor']
                          .named_transformers_['cat']
                          .get_feature_names_out(categorical_features)))

    # Get feature importance (not coefficients!)
    importance = best_model_rf.named_steps['classifier'].feature_importances_

    # Create dataframe and get top 15
    coef_df = pd.DataFrame({'Feature': feature_names, 'Importance': importance})
    coef_df = coef_df.nlargest(15, 'Importance')

    # Plot
    plt.figure(figsize=(10, 6))
    plt.barh(coef_df['Feature'][::-1], coef_df['Importance'][::-1])
    plt.xlabel("Feature Importance")
    plt.title(f"Top 15 Predictors for {physSx_var} (Random Forest)")
    plt.tight_layout()
    plt.show()
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline # Import Pipeline for clarity

# Loop through each symptom
for physSx_var, data_splits in train_test_data.items():
    print(f"\n{'='*80}")
    print(f"SHAP IMPORTANCE COMPARISON FOR: {physSx_var}")
    print(f"{'='*80}")

    # Extract data
    X_train = data_splits['X_train'].copy()
    X_test = data_splits['X_test'].copy()
    y_train = data_splits['y_train']
    y_test = data_splits['y_test']

    # Convert categorical to string
    for col in categorical_features:
        X_train[col] = X_train[col].astype(str)
        X_test[col] = X_test[col].astype(str)
```

```
# Use the preprocessor from a fitted model (e.g., Logistic Regression)
# to transform the data before calculating SHAP values.
# We can get the preprocessor from any of the fitted pipelines as they use the same one.
preprocessor = fitted_models[physSx_var].best_estimator_.named_steps['preprocessor']

X_train_processed = preprocessor.transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# Get feature names after preprocessing for SHAP output
feature_names_processed = (numeric_features +
                           list(preprocessor.named_transformers_['cat']
                                .get_feature_names_out(categorical_features)))


# --- Random Forest ---
# Re-fit the Random Forest model on the preprocessed data
# Note: The pipeline for RF in the previous step didn't scale numeric features.
# For SHAP comparison with LR (which uses scaled features),
# it's more appropriate to use scaled data for both, or fit RF on unscaled data here
# consistently with the RF pipeline definition. Let's use the unscaled preprocessor
# output for RF consistency with the previous RF tuning.
unscaled_preprocessor_shap = unscaled_preprocessor # Use the unscaled preprocessor defined earlier
X_train_unscaled_processed = unscaled_preprocessor_shap.transform(X_train)
X_test_unscaled_processed = unscaled_preprocessor_shap.transform(X_test)
feature_names_unscaled_processed = (numeric_features +
                                    list(unscaled_preprocessor_shap.named_transformers_['cat']
                                         .get_feature_names_out(categorical_features)))


rf_model = RandomForestClassifier(class_weight='balanced', random_state=42, n_estimators=100) # Use
rf_model.fit(X_train_unscaled_processed, y_train)

# SHAP for Random Forest (uses TreeExplainer with unscaled processed data)
rf_explainer = shap.TreeExplainer(rf_model, X_train_unscaled_processed, feature_perturbation="inter
rf_shap_values = rf_explainer.shap_values(X_test_unscaled_processed, check_additivity=False)

# Pick one class (here: last class)
if isinstance(rf_shap_values, list):
    class_idx = len(rf_shap_values) - 1
    rf_class_shap = rf_shap_values[class_idx]
else:
     # For binary classification, shap_values might not be a list
    if rf_shap_values.ndim == 3:
         class_idx = rf_shap_values.shape[2] - 1
         rf_class_shap = rf_shap_values[:, :, class_idx]
    else:
         rf_class_shap = rf_shap_values # Assuming it's already in the correct shape for a single cl

rf_mean_abs = np.abs(rf_class_shap).mean(axis=0)
rf_feature_names = feature_names_unscaled_processed # Use unscaled feature names


# --- Logistic Regression ---
```

```
# Use the scaled processed data for Logistic Regression consistent with its pipeline
log_model = LogisticRegression(max_iter=500, class_weight='balanced', multi_class='multinomial', sol
log_model.fit(X_train_processed, y_train)

# SHAP for Logistic Regression (uses Explainer with scaled processed data)
log_explainer = shap.Explainer(log_model, X_train_processed)
log_shap_vals = log_explainer(X_test_processed).values  # shape: (n_samples, n_features, n_classes)

# Pick same class
if log_shap_vals.ndim == 3:
    class_idx_log = log_shap_vals.shape[2] - 1
    log_class_shap = log_shap_vals[:, :, class_idx_log]
else:
    log_class_shap = log_shap_vals # Assuming it's already in the correct shape for a single class

log_mean_abs = np.abs(log_class_shap).mean(axis=0)
log_feature_names = feature_names_processed # Use scaled feature names (they should be the same name


# --- Create dataframes ---
rf_df = pd.DataFrame({'feature': rf_feature_names, 'mean_abs_shap': rf_mean_abs, 'model': 'Random Fo
log_df = pd.DataFrame({'feature': log_feature_names, 'mean_abs_shap': log_mean_abs, 'model': 'Logist

# Ensure consistent feature names by merging or reindexing if necessary
# Given both preprocessors use the same categorical features and get_feature_names_out,
# the feature names should align. Let's assume they do for now.
shap_summary_df = pd.concat([rf_df, log_df])

# Top 20 features (based on mean across both models)
top20 = shap_summary_df.groupby('feature')['mean_abs_shap'].mean().nlargest(20).index
plot_df = shap_summary_df[shap_summary_df['feature'].isin(top20)]

# --- Plot ---
plt.figure(figsize=(10, 7))
sns.barplot(data=plot_df.sort_values(by='mean_abs_shap', ascending=False), x='mean_abs_shap', y='fea
plt.xlabel("Mean |SHAP value|")
plt.ylabel("Feature")
plt.title(f"Top 20 SHAP Importance for {physSx_var}: RF vs Logistic Regression (Last Class)")
plt.tight_layout()
plt.show()
```

5 Reference

1. American Journal of Psychiatry (1945) Original EEG and syncope study

Bickford, R. G., & Doty, L. G. (1945). The relation of fainting spells to electroencephalographic abnormalities. American Journal of Psychiatry, 102(3), 301–305. https://doi.org/10.1176/ajp.102.3.301

2. The American Journal of Cardiology (2025) Pawar et al. syncope and pulmonary embolism

Pawar, S., Khan, A., Wu, D., Waraich, H., & Menda, J. (2025, March 8). Fainting spells and flow: Revisiting the role of syncope in pulmonary embolism. The American Journal of Cardiology, 249, 85–86. https://doi.org/10.1016/j.amjcard.2025.03.003

3. Gracie, J., Baker, C., & Newton, J. L. (2006). The role of psychological factors in response to treatment

in neurocardiogenic (vasovagal) syncope. Europace, 8(8), 636–643. https://doi.org/10.1093/europace/eul073

4. van den Houdt, S. C. M., Mommersteeg, P. M. C., Widdershoven, J., & Kupper, N. (2024). Sex and gender differences in psychosocial risk profiles among patients with coronary heart disease — The THORESCI-Gender study. International Journal of Behavioral Medicine, 31, 130–144. https://doi.org/10.1007/s12529-023-10170-5

5. Zachariah, J., Leva, E., Ganti, L., Puleo, A., & Ganti, L. (2017). Etiologies of shortness of breath in the emergency department. Academic Emergency Medicine, 24(3), 276–285. https://doi.org/10.1111/acem.13448

6.Dieplinger B, Gegenhuber A, Haltmayer M, et al Evaluation of novel biomarkers for the diagnosis of acute destabilised heart failure in patients with shortness of breath Heart 2009;95:1508-1513

7. Windgassen, S., Moss-Morris, R., Everitt, H., Sibelli, A., Goldsmith, K., & Chalder, T. (2018). Cognitive and behavioral differences between subtypes in refractory irritable bowel syndrome. Behaviour Therapy, 49(5), 726–740. https://doi.org/10.1016/j.beth.2018.09.006

8.Lukacova, I., Keshavarz, B., & Golding, J. F. (2023). Measuring the susceptibility to visually induced motion sickness and its relationship with vertigo, dizziness, migraine, syncope and personality traits. Experimental Brain Research, 241(6), 1381–1391. https://doi.org/10.1007/s00221-023-06603-y

9. Stern, R. M., Koch, K. L., & Andrews, P. L. R. (2011). Nausea: Mechanisms and management. Oxford University Press.

10. Meredith, S., Sibbritt, D., Adams, J., & Frawley, J. (2020). Risk factors for developing comorbid sleeping problems: Results of a survey of 1,925 women over 50 with a chronic health condition. Journal of Aging and Health, 32(5–6), 472–480. https://doi.org/10.1177/0898264319832134

11. Kitselaar, W. M., Büchner, F. L., van der Vaart, R., Sutch, S. P., Bennis, F. C., Evers, A. W., & Numans, M. E. (2023). Early identification of persistent somatic symptoms in primary care: Data-driven and theory-driven predictive modelling based on electronic medical records of Dutch general practices. BMJ Open, 13(5), e066183. https://doi.org/10.1136/bmjopen-2022-066183

12. Kurlansik, S. L., & Maffei, M. S. (2016). Somatic symptom disorder. American Family Physician, 93(1), 49–54.

13. Simon, G. E., VonKorff, M., Piccinelli, M., Fullerton, C., & Ormel, J. (1999). An international study of the relation between somatic symptoms and depression. New England Journal of Medicine, 341(18), 1329–1335. https://doi.org/10.1056/NEJM199910283411801