In [ ]:
```
import numpy as np
import pandas as pd
```

In [10]:
```
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder, FunctionTransformer, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from lightgbm import LGBMClassifier
from sklearn.neural_network import MLPClassifier
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE, RandomOverSampler, BorderlineSMOTE, ADASYN
from imblearn.under_sampling import RandomUnderSampler

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report
```

Experiments to find best model and parameters

In [12]:
```
# Load data
df = pd.read_csv('../data/depression_data.csv')
df = df.drop(columns=['Name'])

# Splitting features and target
X = df.drop(['History of Mental Illness'], axis=1)
y = df['History of Mental Illness'].map({'Yes': 1, 'No': 0})

# Columns setup
categorical_cols = ['Marital Status', 'Education Level', 'Smoking Status', 'Physical Activity Level',
                    'Employment Status', 'Alcohol Consumption', 'Dietary Habits', 'Sleep Patterns',
                    'History of Substance Abuse', 'Family History of Depression', 'Chronic Medical Conditions']
numeric_cols = ['Age', 'Number of Children']

# Log scaling for Income
df['Income'] = df['Income'].apply(lambda x: np.log(x + 1))

# One hot encoding and scaling
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_cols),
        ('cat', OneHotEncoder(drop=None), categorical_cols)
    ])

# Transform the data
X_transformed = preprocessor.fit_transform(X)

# Train-test split (80-20)
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y, test_size=0.2, random_state=42)

# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Define models and hyperparameters for GridSearch
model_params = {
    'DecisionTree': {
        'model': DecisionTreeClassifier(random_state=42),
        'params': {
            'classifier__max_depth': [10, 20, 30, None],
            'classifier__min_samples_split': [2, 5, 10]
        }
    },
    'LogisticRegression': {
        'model': LogisticRegression(random_state=42),
        'params': {
            'classifier__max_iter': [100, 250, 500],
            'classifier__C': [0.01, 0.1, 1, 10],
            'classifier__penalty': ['l1', 'l2'],
            'classifier__solver': ['liblinear', 'saga']
        }
    },
    'LightGBM': {
        'model': LGBMClassifier(random_state=42),
        'params': {
            'classifier__n_estimators': [50, 100, 200],
            'classifier__max_depth': [10, 20, 30, None],
            'classifier__learning_rate': [0.01, 0.05, 0.1]
        }
    },
    'RandomForest': {
        'model': RandomForestClassifier(random_state=42),
        'params': {
            'classifier__n_estimators': [50, 100, 150],
            'classifier__max_depth': [10, 20, 30, None],
            'classifier__min_samples_split': [2, 5, 10]
        }
    }
}


# Iterate through each model, apply GridSearchCV and evaluate
for model_name, mp in model_params.items():
    print(f"\nModel: {model_name}")

    # Create pipeline for model with classifier (preprocessor is already applied)
    clf = Pipeline(steps=[('classifier', mp['model'])])

    # Perform Grid Search with Cross-Validation
    grid_search = GridSearchCV(clf, mp['params'], cv=5, scoring='f1', n_jobs=-1)
    grid_search.fit(X_resampled, y_resampled)

    # Get the best model from grid search
    best_model = grid_search.best_estimator_
    print("--------------- Best Model: ------------------")
    print(best_model)

    # Predictions on the test set
    y_pred = best_model.predict(X_test)

    # Print evaluation metrics
    print(f"Best parameters found: {grid_search.best_params_}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred):.4f}")
    print(f"Recall: {recall_score(y_test, y_pred):.4f}")
    print(f"F1 Score: {f1_score(y_test, y_pred):.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
```

```
Model: DecisionTree
--------------- Best Model: ------------------
Pipeline(steps=[('classifier', DecisionTreeClassifier(random_state=42))])
Best parameters found: {'classifier__max_depth': None, 'classifier__min_samples_split': 2}
Accuracy: 0.5933
Precision: 0.3287
Recall: 0.3178
F1 Score: 0.3232
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.71      0.71     57471
           1       0.33      0.32      0.32     25283

    accuracy                           0.59     82754
   macro avg       0.52      0.52      0.52     82754
weighted avg       0.59      0.59      0.59     82754


Model: LogisticRegression
--------------- Best Model: ------------------
Pipeline(steps=[('classifier',
                 LogisticRegression(C=0.01, random_state=42,
                                    solver='liblinear'))])
Best parameters found: {'classifier__C': 0.01, 'classifier__max_iter': 100, 'classifier__penalty': 'l2', 'classifier__solver': 'liblinear'}
Accuracy: 0.6172
Precision: 0.3919
Recall: 0.4586
F1 Score: 0.4226
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.69      0.71     57471
           1       0.39      0.46      0.42     25283

    accuracy                           0.62     82754
   macro avg       0.57      0.57      0.57     82754
weighted avg       0.64      0.62      0.62     82754


Model: LightGBM
[LightGBM] [Info] Number of positive: 230472, number of negative: 230472
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.030919 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 8670
[LightGBM] [Info] Number of data points in the train set: 460944, number of used features: 34
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
--------------- Best Model: ------------------
Pipeline(steps=[('classifier',
                 LGBMClassifier(learning_rate=0.05, max_depth=30,
                                n_estimators=50, random_state=42))])
Best parameters found: {'classifier__learning_rate': 0.05, 'classifier__max_depth': 30, 'classifier__n_estimators': 50}
Accuracy: 0.6392
Precision: 0.4000
Recall: 0.3621
F1 Score: 0.3801
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.76      0.75     57471
           1       0.40      0.36      0.38     25283

    accuracy                           0.64     82754
   macro avg       0.57      0.56      0.56     82754
weighted avg       0.63      0.64      0.63     82754


Model: RandomForest
--------------- Best Model: ------------------
Pipeline(steps=[('classifier',
                 RandomForestClassifier(max_depth=30, min_samples_split=5,
                                        n_estimators=150, random_state=42))])
Best parameters found: {'classifier__max_depth': 30, 'classifier__min_samples_split': 5, 'classifier__n_estimators': 150}
Accuracy: 0.6061
Precision: 0.3403
Recall: 0.3082
F1 Score: 0.3234
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.74      0.72     57471
           1       0.34      0.31      0.32     25283

    accuracy                           0.61     82754
   macro avg       0.52      0.52      0.52     82754
weighted avg       0.60      0.61      0.60     82754
```

Experiment Insights and Key Takeaways

In this experiment, we tested four models: decision tree, logistic regression, lightGBM, and random forest. The aim was to find the best-performing model in terms of accuracy, precision, recall, and F1 score, as well as model explainability and robustness.

Conclusion

For our final model, we will use **Logistic Regression** with the best settings from this experiment ('C = 0.01, max_iter = 100, penalty = 'l2', solver = 'liblinear''). This model provides the best balance of performance, simplicity, and explainability for predicting if a person is likely to suffer from mental illness.

In [ ]: