```
# create a class for the depression data
import pandas as pd
import optuna
import imblearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt



class DepressionTask:
    def __init__(self, file_path):
        self.file_path = file_path

    def load_data(self):
        data = pd.read_csv(self.file_path)
        return data

    def explore_data(self, data):
        # Summary statistics for numerical columns
        summary_stats = data.describe()
        print("Summary Statistics for Numerical Data:")
        print(summary_stats)

        # Categorical data overview
```

```
                categorical_overview = {col: data[col].unique() for col in
data.select_dtypes(include='object').columns}
    print("\nUnique Values in Categorical Columns:")
    for col, values in categorical_overview.items():
        print(f"{col}: {values}")

    # Function to create a bar chart for categorical data
    def plot_categorical_distribution(column, title):
        data[column].value_counts().plot(kind='bar', figsize=(8, 6))
        plt.title(title)
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.savefig('data/depression/figures/' + column + '.png')

    # Function to plot numerical data distributions
    def plot_numerical_distribution(column, title):
        data[column].plot(kind='hist', bins=20, figsize=(8, 6), alpha=0.7)
        plt.title(title)
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.savefig('data/depression/figures/' + column + '.png')

    # Plot distributions of some key categorical columns
    plot_categorical_distribution('Marital Status', 'Distribution of Marital Status')
    plot_categorical_distribution('Smoking Status', 'Distribution of Smoking Status')
    plot_categorical_distribution('Alcohol Consumption', 'Distribution of Alcohol Consumption')
```

```
        # Plot numerical data distributions
        plot_numerical_distribution('Age', 'Distribution of Age')
        plot_numerical_distribution('Income', 'Distribution of Income')

        # Analyzing relationships: Family history of depression and history of mental illness
            family_history_vs_mental_illness = data.groupby('Family History of Depression')['History of Mental Illness'].value_counts(normalize=True).unstack()
        family_history_vs_mental_illness.plot(kind='bar', stacked=True, figsize=(10, 6))
        plt.title('Mental Illness Based on Family History of Depression')
        plt.ylabel('Proportion')
        plt.savefig('data/depression/figures/family_history_vs_mental_illness.png')

        # Analyzing the relationship between alcohol consumption and mental illness
                alcohol_vs_mental_illness = data.groupby('Alcohol Consumption')['History of Mental Illness'].value_counts(normalize=True).unstack()
        alcohol_vs_mental_illness.plot(kind='bar', stacked=True, figsize=(10, 6))
        plt.title('Mental Illness Based on Alcohol Consumption')
        plt.ylabel('Proportion')
        plt.savefig('data/depression/figures/alcohol_vs_mental_illness.png')

    def preprocess_data(self, data):
        # Encode categorical variables using LabelEncoder
        label_encoders = {}
        for column in data.select_dtypes(include='object').columns:
            if column != 'History of Mental Illness':  # Target variable handled separately
                label_encoders[column] = LabelEncoder()
                data[column] = label_encoders[column].fit_transform(data[column])
```

```
        # Encode the target variable (History of Mental Illness)
        target_encoder = LabelEncoder()
        data['History of Mental Illness'] = target_encoder.fit_transform(data['History of Mental Illness'])

        # Separate features (X) and target variable (y)
        X = data.drop(['History of Mental Illness', 'Name'], axis=1)  # Drop Name as it's not a feature
        y = data['History of Mental Illness']

        return X, y

    def split_data(self, X, y):
        # Split data into training and testing sets
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,
stratify=y)
        return X_train, X_test, y_train, y_test

    def standardize_data(self, X_train, X_test):
        # Standardise the numerical features
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
        return X_train, X_test

    def balance_data(self, X_train, y_train):
        # Apply SMOTE to balance the training data
        smote = imblearn.over_sampling.SMOTE(random_state=42)
```

```
    X_train, y_train = smote.fit_resample(X_train, y_train)
    return X_train, y_train

def train_rf_model(self, X_train, y_train, X_test, y_test):
    # Train a Random Forest Classifier
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict_proba(X_test)

    # Evaluate the model using ROC AUC score
    roc_auc = roc_auc_score(y_test, y_pred[:, 1])
    return roc_auc

def train_lr_model(self, X_train, y_train, X_test, y_test):
    # Train a Logistic Regression model
    model = LogisticRegression(random_state=42)
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict_proba(X_test)

    # Evaluate the model using ROC AUC score
    roc_auc = roc_auc_score(y_test, y_pred[:, 1])
    return roc_auc
```

```
def optimize_rf_hyperparameters(self, X_train, y_train, X_test, y_test):
    def objective(trial):
        param = {
            "n_estimators": trial.suggest_int("n_estimators", 50, 500),
            "max_depth": trial.suggest_int("max_depth", 3, 10),
            "min_samples_split": trial.suggest_int("min_samples_split", 2, 20),
            "min_samples_leaf": trial.suggest_int("min_samples_leaf", 1, 10),
            "max_features": trial.suggest_categorical("max_features", ["auto", "sqrt", "log2"]),
        }

        rf = RandomForestClassifier(**param)
        rf.fit(X_train, y_train)
        y_pred = rf.predict_proba(X_test)
        return roc_auc_score(y_test, y_pred[:, 1])

    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=100)
    return study.best_params

def optimize_lr_hyperparameters(self, X_train, y_train, X_test, y_test):
    def objective(trial):
        param = {
            "C": trial.suggest_loguniform("C", 1e-5, 1e5),
            "penalty": trial.suggest_categorical("penalty", ["l1", "l2"]),
            "max_iter": trial.suggest_int("max_iter", 100, 1000),
        }
```

```
    lr = LogisticRegression(**param, solver="saga")
    lr.fit(X_train, y_train)
    y_pred = lr.predict_proba(X_test)
    return roc_auc_score(y_test, y_pred[:, 1])

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100)
return study.best_params
```