```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
import matplotlib.pyplot as plt

def load_and_preprocess_data(Air_quality_data):
    """Load and preprocess air quality data specific to monthly values"""
    # Load the dataset using Pandas
    df = pd.read_csv(Air_quality_data)

    # Convert monthly columns to numeric (some are stored as strings)
    month_cols = ['feb', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct']
    for col in month_cols:
        df[col] = pd.to_numeric(df[col], errors='coerce')

    # Fill missing values
    df.ffill(inplace=True)
    df.bfill(inplace=True)

    # Drop non-numeric columns like 'city' if present
    if 'city' in df.columns:
        df.drop(columns=['city'], inplace=True)

    return df
```

```
def prepare_data(df):
    """Prepare features and target variable"""
    # Assume 'avg' is the target
    target_column = 'avg'

    # All other numeric columns are features
    X = df.drop(columns=[target_column])
    y = df[target_column]

    return X, y, target_column

def train_and_evaluate(X_train, X_test, y_train, y_test):
    """Train and evaluate regression models"""
    models = {
        'Random Forest': RandomForestRegressor(random_state=42),
        'Gradient Boosting': GradientBoostingRegressor(random_state=42),
        'XGBoost': XGBRegressor(random_state=42)
    }

    results = {}
    for name, model in models.items():
        try:
            print(f"Training {name}...")
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)

            mse = mean_squared_error(y_test, y_pred)
```

```
            r2 = r2_score(y_test, y_pred)

            results[name] = {
                'model': model,
                'rmse': np.sqrt(mse),
                'r2': r2
            }

            print(f"{name} - RMSE: {np.sqrt(mse):.2f}, R2: {r2:.2f}")
        except Exception as e:
            print(f"Error training {name}: {str(e)}")

    return results

if __name__ == "__main__":
    try:
        # Load and preprocess the dataset
        df = load_and_preprocess_data('/Air_quality_data.csv')

        # Prepare data
        X, y, target_column = prepare_data(df)

        # Train-test split
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42, shuffle=False
        )

        # Feature scaling
```

```
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train and evaluate models
    results = train_and_evaluate(X_train_scaled, X_test_scaled, y_train, y_test)

    # Plot results for best model
    if results:
        best_model_name = max(results.items(), key=lambda x: x[1]['r2'])[0]
        best_model = results[best_model_name]['model']
        y_pred = best_model.predict(X_test_scaled)

        plt.figure(figsize=(12, 6))
        plt.plot(y_test.values[:100], label='Actual')
        plt.plot(y_pred[:100], label='Predicted')
        plt.title(f'Air Quality Prediction (Target: {target_column})')
        plt.xlabel('Sample Index')
        plt.ylabel('Air Quality (avg)')
        plt.legend()
        plt.tight_layout()
        plt.show()
    else:
        print("No models were successfully trained.")

except Exception as e:
    print(f"An error occurred: {str(e)}")
```