Phase 1 in Visual Studio (VS)

Table of Contents

To install and set up the final version of Phase 1 into Visual Studio, you'll need to create a setup script that creates a virtual environment, installs the necessary dependencies, and prepares your environment for development. Below is the detailed script for setting up Phase 1 in Visual Studio.

Batch Script for Windows (`setup_phase_1.bat`)

```
@echo off
REM Create virtual environment
python -m venv venv

REM Activate virtual environment
call venv\\Scripts\\activate

REM Install dependencies
pip install pandas scikit-learn xgboost Flask joblib

REM Install additional tools for development
pip install pytest pylint

echo Virtual environment and dependencies setup complete.
echo To activate the virtual environment, run: call venv\\Scr
ipts\\activate
pause
```

Shell Script for macOS/Linux (`setup_phase_1.sh`)

```
#!/bin/bash
# Create virtual environment
python3 -m venv venv

# Activate virtual environment
source venv/bin/activate

# Install dependencies
pip install pandas scikit-learn xgboost Flask joblib

# Install additional tools for development
pip install pytest pylint

echo "Virtual environment and dependencies setup complete."
echo "To activate the virtual environment, run: source venv/b
in/activate"
```

Instructions for Use

1. **Save the Script:**
   - Save the script as `setup_phase_1.bat` if you are using Windows.
   - Save the script as `setup_phase_1.sh` if you are using macOS/Linux.
2. **Place the Script:**
   - Place the script in the root directory of your Visual Studio project.
3. **Run the Script:**
   - On Windows: Double-click the `setup_phase_1.bat` file or run it from the command prompt.
   - On macOS/Linux: Open a terminal and run `bash setup_phase_1.sh`.
4. **Open Project in Visual Studio:**
   - Open Visual Studio.
   - Open the project or solution where the Phase 1 scripts are located.

5. **Set Up Python Environment in Visual Studio:**
    - Go to `Tools` > `Python` > `Python Environments`.
    - Select `Add Environment`.
    - Choose `Existing environment` and point to the `venv` folder created by the script.
    - Set it as the default environment for your project.

Example Project Structure

Ensure your project structure looks something like this:

```
MyProject
|-- venv/
|-- scripts/
|   |-- analyze_model.py
|   |-- data_preparation.py
|   |-- model_development.py
|   |-- update_api.py
|-- setup_phase_1.bat (or setup_phase_1.sh)
|-- run_pipeline.py
|-- test/
|   |-- test_analyze_model.py
|   |-- test_data_preparation.py
|   |-- test_model_development.py
|   |-- test_pipeline.py
|-- README.md
```

Script Contents for Phase 1

analyze_model.py

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, mean_absolute_error
import joblib
import pandas as pd
```

```
import logging

logging.basicConfig(level=logging.INFO)

def evaluate_model(model, X_test, y_test):
    try:
        predictions = model.predict(X_test)
        metrics = {
            'accuracy': accuracy_score(y_test, predictions),
            'precision': precision_score(y_test, predictions,
average='macro'),
            'recall': recall_score(y_test, predictions, avera
ge='macro'),
            'f1_score': f1_score(y_test, predictions, average
='macro'),
            'mean_absolute_error': mean_absolute_error(y_tes
t, predictions)
        }
        logging.info("Model evaluation successful.")
        return metrics
    except Exception as e:
        logging.error(f"Error evaluating model: {e}")
        raise

if __name__ == "__main__":
    try:
        model = joblib.load('property_valuation_model.pkl')
        X_test = pd.read_csv('X_test.csv')
        y_test = pd.read_csv('y_test.csv')
        metrics = evaluate_model(model, X_test, y_test)
        print(metrics)
    except Exception as e:
        logging.error(f"Error running analysis: {e}")
```

data_preparation.py

```
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO)

def load_data():
    try:
        demographics = pd.read_csv('neighborhood_demographic
s.csv')
        school_info = pd.read_csv('school_district_info.csv')
        market_trends = pd.read_csv('real_estate_market_trend
s.csv')
        property_records = pd.read_csv('property_records.cs
v')
        public_records = pd.read_csv('public_records.csv')
        economic_indicators = pd.read_csv('economic_indicator
s.csv')
        social_data = pd.read_csv('social_data.csv')
        geospatial_data = pd.read_csv('geospatial_data.csv')
        logging.info("Data loaded successfully.")
        return demographics, school_info, market_trends, prop
erty_records, public_records, economic_indicators, social_dat
a, geospatial_data
    except Exception as e:
        logging.error(f"Error loading data: {e}")
        raise

def feature_engineering(df):
    try:
        df['price_per_sqft'] = df['price'] / df['sqft']
        df['age_of_property'] = 2024 - df['year_built']
        logging.info("Feature engineering successful.")
        return df
    except Exception as e:
        logging.error(f"Error in feature engineering: {e}")
```

```
        raise

if __name__ == "__main__":
    try:
        demographics, school_info, market_trends, property_re
cords, public_records, economic_indicators, social_data, geos
patial_data = load_data()
        all_data = pd.concat([demographics, school_info, mark
et_trends, property_records, public_records, economic_indicat
ors, social_data, geospatial_data], axis=1)
        all_data = feature_engineering(all_data)
        all_data.to_csv('prepared_data.csv', index=False)
        logging.info("Data preparation complete.")
    except Exception as e:
        logging.error(f"Error in data preparation: {e}")
```

model_development.py

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO)

def train_model(X, y):
    try:
        model = xgb.XGBRegressor()
        model.fit(X, y)
        logging.info("Model training successful.")
        return model
    except Exception as e:
        logging.error(f"Error training model: {e}")
        raise
```

```
if __name__ == "__main__":
    try:
        data = pd.read_csv('prepared_data.csv')
        X = data.drop(columns=['price'])
        y = data['price']
        X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size=0.2, random_state=42)
        model = train_model(X_train, y_train)
        model.save_model('new_property_valuation_model.json')
        metrics = evaluate_model(model, X_test, y_test)
        print(metrics)
    except Exception as e:
        logging.error(f"Error in model development: {e}")
```

update_api.py

```
from flask import Flask, request, jsonify
import xgboost as xgb
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO)

app = Flask(__name__)

try:
    model = xgb.XGBRegressor()
    model.load_model('new_property_valuation_model.json')
    logging.info("Model loaded successfully.")
except Exception as e:
    logging.error(f"Error loading model: {e}")
    raise

@app.route('/predict', methods=['POST'])
def predict():
```

```
    try:
        data = request.get_json()
        df = pd.DataFrame(data)
        prediction = model.predict(df)
        return jsonify({'prediction': prediction.tolist()})
    except Exception as e:
        logging.error(f"Error in prediction: {e}")
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

Summary

This setup ensures that your environment for Phase 1 is properly configured and ready for development. The provided scripts install necessary dependencies, set up the virtual environment, and prepare the codebase for execution in Visual Studio. Remember to replace any placeholder paths or filenames with the actual paths or filenames used in your project.

Step-by-Step Guide

1. Open Your Project in Visual Studio

- Launch Visual Studio.
- Open your existing project or create a new Python project.

2. Open the Terminal in Visual Studio

- Go to `View` > `Terminal` or use the shortcut `Ctrl +` (backtick) to open the integrated terminal.

3. Create a Virtual Environment

In the terminal, navigate to your project directory if you are not already there. Then, run the following commands:

```
# Create a virtual environment
```

```
Here's a detailed guide to set up Phase 1 in Visual Studio, i
ncluding creating a virtual environment, installing dependenc
ies, and setting up your project structure:

### Step-by-Step Guide

#### 1. Open Your Project in Visual Studio
- Launch Visual Studio.
- Open your existing project or create a new Python project.

#### 2. Open the Terminal in Visual Studio
- Go to `View` > `Terminal` or use the shortcut `Ctrl + ` (ba
cktick) to open the integrated terminal.

#### 3. Create a Virtual Environment
In the terminal, navigate to your project directory if you ar
e not already there. Then, run the following commands:

```bash
# Create a virtual environment
python -m venv venv

# Activate the virtual environment
# On Windows
venv\\Scripts\\activate
# On macOS/Linux
source venv/bin/activate
```

4. Install Dependencies

With the virtual environment activated, run the following command to install the required dependencies:

```
pip install pandas scikit-learn xgboost Flask joblib
```
```

5. Verify Installation

Run the following command to verify that all dependencies have been installed correctly:

```
pip freeze
```

6. Setup Project Structure and Files

Create the necessary project files and directories as outlined in Phase 1.

Create `analyze_model.py`

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, mean_absolute_error
import joblib
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO)

def evaluate_model(model, X_test, y_test):
    try:
        predictions = model.predict(X_test)
        metrics = {
            'accuracy': accuracy_score(y_test, predictions),
            'precision': precision_score(y_test, predictions,
average='macro'),
            'recall': recall_score(y_test, predictions, avera
ge='macro'),
            'f1_score': f1_score(y_test, predictions, average
='macro'),
            'mean_absolute_error': mean_absolute_error(y_tes
t, predictions)
        }
        logging.info("Model evaluation successful.")
        return metrics
```

```
    except Exception as e:
        logging.error(f"Error evaluating model: {e}")
        raise

if __name__ == "__main__":
    try:
        model = joblib.load('property_valuation_model.pkl')
        X_test = pd.read_csv('X_test.csv')
        y_test = pd.read_csv('y_test.csv')
        metrics = evaluate_model(model, X_test, y_test)
        print(metrics)
    except Exception as e:
        logging.error(f"Error running analysis: {e}")
```

Create `data_preparation.py`

```
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO)

def load_data():
    try:
        demographics = pd.read_csv('neighborhood_demographic
s.csv')
        school_info = pd.read_csv('school_district_info.csv')
        market_trends = pd.read_csv('real_estate_market_trend
s.csv')
        property_records = pd.read_csv('property_records.cs
v')
        public_records = pd.read_csv('public_records.csv')
        economic_indicators = pd.read_csv('economic_indicator
s.csv')
        social_data = pd.read_csv('social_data.csv')
        geospatial_data = pd.read_csv('geospatial_data.csv')
```

```
        logging.info("Data loaded successfully.")
        return demographics, school_info, market_trends, prop
erty_records, public_records, economic_indicators, social_dat
a, geospatial_data
    except Exception as e:
        logging.error(f"Error loading data: {e}")
        raise

def feature_engineering(df):
    try:
        df['price_per_sqft'] = df['price'] / df['sqft']
        df['age_of_property'] = 2024 - df['year_built']
        logging.info("Feature engineering successful.")
        return df
    except Exception as e:
        logging.error(f"Error in feature engineering: {e}")
        raise

if __name__ == "__main__":
    try:
        demographics, school_info, market_trends, property_re
cords, public_records, economic_indicators, social_data, geos
patial_data = load_data()
        all_data = pd.concat([demographics, school_info, mark
et_trends, property_records, public_records, economic_indicat
ors, social_data, geospatial_data], axis=1)
        all_data = feature_engineering(all_data)
        all_data.to_csv('prepared_data.csv', index=False)
        logging.info("Data preparation complete.")
    except Exception as e:
        logging.error(f"Error in data preparation: {e}")
```

Create `model_development.py`

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO)

def train_model(X, y):
    try:
        model = xgb.XGBRegressor()
        model.fit(X, y)
        logging.info("Model training successful.")
        return model
    except Exception as e:
        logging.error(f"Error training model: {e}")
        raise

if __name__ == "__main__":
    try:
        data = pd.read_csv('prepared_data.csv')
        X = data.drop(columns=['price'])
        y = data['price']
        X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size=0.2, random_state=42)
        model = train_model(X_train, y_train)
        model.save_model('new_property_valuation_model.json')
        metrics = evaluate_model(model, X_test, y_test)
        print(metrics)
    except Exception as e:
        logging.error(f"Error in model development: {e}")
```

Create `update_api.py`

```
from flask import Flask, request, jsonify
import xgboost as xgb
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO)

app = Flask(__name__)

try:
    model = xgb.XGBRegressor()
    model.load_model('new_property_valuation_model.json')
    logging.info("Model loaded successfully.")
except Exception as e:
    logging.error(f"Error loading model: {e}")
    raise

@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.get_json()
        df = pd.DataFrame(data)
        prediction = model.predict(df)
        return jsonify({'prediction': prediction.tolist()})
    except Exception as e:
        logging.error(f"Error in prediction: {e}")
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

Script to Install Dependencies and Setup Environment

To automate the setup process in Visual Studio, create a batch script for Windows or a shell script for macOS/Linux:

Batch Script for Windows (`setup_phase1.bat`):

```
@echo off

:: Create a virtual environment
echo Creating virtual environment...
python -m venv venv

:: Activate the virtual environment
echo Activating virtual environment...
call venv\\Scripts\\activate

:: Install dependencies
echo Installing dependencies...
pip install pandas scikit-learn xgboost Flask joblib

:: Verify installation
echo Verifying installation...
pip freeze

echo Setup complete. To activate the virtual environment in t
he future, use 'venv\\Scripts\\activate'.
pause
```

Shell Script for macOS/Linux (`setup_phase1.sh`):

```
#!/bin/bash

# Create a virtual environment
echo "Creating virtual environment..."
python -m venv venv

# Activate the virtual environment
echo "Activating virtual environment..."
source venv/bin/activate
```

```
# Install dependencies
echo "Installing dependencies..."
pip install pandas scikit-learn xgboost Flask joblib

# Verify installation
echo "Verifying installation..."
pip freeze

echo "Setup complete. To activate the virtual environment, us
e 'source venv/bin/activate' on macOS/Linux or 'venv\\Scripts
\\activate' on Windows."
```

Running the Setup Script in Visual Studio

1. **Create the Script File:**
   - Save the script as `setup_phase1.bat` for Windows or `setup_phase1.sh` for macOS/Linux in your project directory.
2. **Run the Script:**
   - Open the terminal in Visual Studio (`View` > `Terminal`).
   - Navigate to your project directory.
   - Run the script:
     - For Windows:

       ```
       setup_phase1.bat
       ```

     - For macOS/Linux:

       ```
       chmod +x setup_phase1.sh
       ./setup_phase1.sh
       ```

This setup will prepare your environment, install all necessary dependencies, and ensure that your project is ready for Phase 1 development in Visual Studio.