Phase 1 implementation with all improvements included.

Outline and Table of Contents

- Unit Test for `model_development.py`

5. Integration Test for the Entire Pipeline
6. Final Script to Run the Entire Pipeline
7. Environment Setup Scripts
    - Batch Script: `setup_phase1.bat`
    - Shell Script: `setup_phase1.sh`
8. Documentation
    - `README.md`
9. Implementation Steps
    - Set Up Virtual Environment and Install Dependencies
    - Execute the Setup Script
    - Run the Pipeline
10. Conclusion

1 implementation with all improvements included. The following is the comprehensive set of scripts and instructions for setting up, running, and validating Phase 1 of the property valuation AI model enhancement.

Phase 1 requirements install

1. Introduction
2. Project Structure
3. Detailed Explanation of Scripts:
    - `analyze_model.py`
    - `data_preparation.py`

- `model_development.py`
- `update_api.py`

4. Unit Tests:
   - Unit Test for `analyze_model.py`
   - Unit Test for `data_preparation.py`
   - Unit Test for `model_development.py`
5. Integration Test for the Entire Pipeline
6. Final Script to Run the Entire Pipeline
7. Environment Setup Scripts:
   - Batch Script: `setup_phase1.bat`
   - Shell Script: `setup_phase1.sh`
8. Documentation:
   - `README.md`
9. Implementation Steps:
   - Set Up Virtual Environment and Install Dependencies
   - Execute the Setup Script
   - Run the Pipeline
10. Conclusion
11. Introduction
    In this document, we present the comprehensive set of scripts and instructions for setting up, running, and validating Phase 1 of the property valuation AI model enhancement.
12. Project Structure
    The project is organized into several Python scripts that handle different aspects of the pipeline - from data preparation to model evaluation.
13. Detailed Explanation of Scripts:
    - `analyze_model.py`: Evaluates the current model's performance using various evaluation metrics.

- `data_preparation.py`: Loads additional data sources and performs feature engineering.
- `model_development.py`: Trains new models with advanced algorithms and evaluates their performance.
- `update_api.py`: Sets up a Flask application with updated API endpoints to utilize the new model.

14. Unit Tests:
    - Unit Test for `analyze_model.py`: Checks if the model evaluation script is working as expected.
    - Unit Test for `data_preparation.py`: Checks if the data loading and feature engineering scripts are working as expected.
    - Unit Test for `model_development.py`: Checks if the model training and evaluation scripts are working as expected.
15. Integration Test for the Entire Pipeline
    - Checks if all the scripts in the pipeline are working together seamlessly.
16. Final Script to Run the Entire Pipeline
    - `run_pipeline.py`: A script that runs the entire pipeline from data preparation to model deployment.
17. Environment Setup Scripts:
    - Batch Script: `setup_phase1.bat`: A script for setting up the project environment on Windows.
    - Shell Script: `setup_phase1.sh`: A script for setting up the project environment on macOS/Linux.
18. Documentation:
    - `README.`MD is a document that provides an overview of the project and instructions on how to use the scripts.
19. Implementation Steps:
    - Set Up Virtual Environment and Install Dependencies: Instructions on how to set up the project environment.

- Execute the Setup Script: Instructions on how to run the setup script to install the necessary dependencies.
- Run the Pipeline: Instructions on how to run the entire pipeline using the `run_pipeline.py` script.

20. Conclusion
    This finalized Phase 1 implementation ensures a comprehensive and robust approach to enhancing the property valuation AI model. All scripts have error handling, logging, unit tests, integration tests, and a detailed setup process. This will facilitate the enhanced model's smooth development, testing, and deployment.

As an Azure Full Stack Developer, I would suggest making the following changes in order to follow best practices for using Azure services:

`data_preparation.py`

Replace the local file reading with Azure Blob Storage service. This would allow for better scalability and centralized data management:

```
from azure.storage.blob import BlobServiceClient
import io

def load_data():
    try:
        blob_service_client = BlobServiceClient.from_connecti
on_string("<your_connection_string>")
        container_name = "<your_container_name>"

        data_sources = []
        for file_name in ['neighborhood_demographics.csv', 's
chool_district_info.csv', 'real_estate_market_trends.csv', 'p
roperty_records.csv', 'public_records.csv', 'economic_indicat
ors.csv', 'social_data.csv', 'geospatial_data.csv']:
            blob_client = blob_service_client.get_blob_client
(container_name, file_name)
            data = blob_client.download_blob().readall()
```

```
            df = pd.read_csv(io.BytesIO(data))
            data_sources.append(df)

        logging.info("Data loaded successfully.")
        return data_sources
    except Exception as e:
        logging.error(f"Error loading data: {e}")
        raise
```

Be sure to replace `<your_connection_string>` and `<your_container_name>` with your actual connection string and container name.

`model_development.py`

Use Azure Machine Learning service to train the model:

```
from azureml.core import Workspace, Experiment, Run
from azureml.train.automl import AutoMLConfig

def train_model(X, y):
    try:
        ws = Workspace.from_config()

        automl_config = AutoMLConfig(task='regression',
                                     X=X,
                                     y=y,
                                     iterations=50,
                                     primary_metric='spearman
_correlation',
                                     n_cross_validations=5)

        experiment = Experiment(ws, 'property_valuation_exper
iment')
        run = experiment.submit(automl_config, show_output=Tr
ue)
```

```
        best_run, fitted_model = run.get_output()

        logging.info("Model training successful.")
        return fitted_model
    except Exception as e:
        logging.error(f"Error training model: {e}")
        raise
```

`update_api.py`

Deploy the model as a web service on Azure Container Instance:

```
from azureml.core.model import Model, InferenceConfig
from azureml.core.webservice import AciWebservice, Webservice
from azureml.core import Workspace
from azureml.core.environment import Environment

def deploy_model():
    try:
        ws = Workspace.from_config()
        model = Model.register(model_path="<path_to_model>",
                               model_name="property_valuation
_model",
                               workspace=ws)

        myenv = Environment.get(workspace=ws, name="myenv")
        inference_config = InferenceConfig(entry_script="scor
e.py", environment=myenv)

        deployment_config = AciWebservice.deploy_configuratio
n(cpu_cores=1, memory_gb=1)
        service = Model.deploy(ws, "myservice", [model], infe
rence_config, deployment_config)
        service.wait_for_deployment(show_output=True)

        logging.info("Model deployed successfully.")
```

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

Note:

In the above code, replace `<path_to_model>` with the actual path to your model. Also, you need to create a scoring script (`score.py`) that includes two functions: `init()` to initialize your model, and `run(raw_data)` to run predictions on your model.

Remember to always use secure practices while handling connection strings and sensitive information. Never hardcode them into your code. Use Azure Key Vault to securely store and manage sensitive information.

Project Structure

```
Phase1/
|-- analyze_model.py
|-- data_preparation.py
|-- model_development.py
|-- update_api.py
|-- test_analyze_model.py
|-- test_data_preparation.py
|-- test_model_development.py
|-- test_pipeline.py
|-- run_pipeline.py
|-- setup_phase1.bat
|-- setup_phase1.sh
|-- README.md
```

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}")
```

2. `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}")
```

3. `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
```

```
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:
        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}")
```

4. `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)
```

Unit Tests

Unit Test for `analyze_model.py`

```
# test_analyze_model.py

import unittest
from analyze_model import evaluate_model
import joblib
import pandas as pd

class TestAnalyzeModel(unittest.TestCase):
    def setUp(self):
        self.model = joblib.load('property_valuation_model.pk
l')
        self.X_test = pd.read_csv('X_test.csv')
        self.y_test = pd.read_csv('y_test.csv')

    def test_evaluate_model(self):
        metrics = evaluate_model(self.model, self.X_test, sel
f.y_test)
        self.assertIn('accuracy', metrics)
        self.assertIn('precision', metrics)
        self.assertIn('recall', metrics)
        self.assertIn('f1_score', metrics)
        self.assertIn('mean_absolute_error', metrics)

if __name__ == '__main__':
    unittest.main()
```

Unit Test for `data_preparation.py`

```
# test_data_preparation.py

import unittest
from data_preparation import load_data, feature_engineering
import pandas as pd

class TestDataPreparation(unittest.TestCase):
```

```
    def test_load_data(self):
        data_sources = load_data()
        self.assertEqual(len(data_sources), 8)

    def test_feature_engineering(self):
        df = pd.DataFrame({'price': [100, 200], 'sqft': [100
0, 1500], 'year_built': [2000, 2010]})
        engineered_df = feature_engineering(df)
        self.assertIn('price_per_sqft', engineered_df.column
s)
        self.assertIn('age_of_property', engineered_df.column
s)

if __name__ == '__main__':
    unittest.main()
```

Unit Test for `model_development.py`

```
# test_model_development.py

import unittest
from model_development import train_model, evaluate_model
import pandas as pd
from sklearn.model_selection import train_test_split

class TestModelDevelopment(unittest.TestCase):
    def setUp(self):
        self.data = pd.read_csv('prepared_data.csv')
        self.X = self.data.drop(columns=['price'])
        self.y = self.data['price']

    def test_train_model(self):
        model = train_model(self.X, self.y)
        self.assertIsNotNone(model)

    def test_evaluate_model(self):
```

```
        X_train, X_test, y_train, y_test = train_test_split(s
elf.X, self.y, test_size=0.2, random_state=42)
        model = train_model(X_train, y_train)
        metrics = evaluate_model(model, X_test, y_test)
        self.assertIn('accuracy', metrics)
        self.assertIn('precision', metrics)
        self.assertIn('recall', metrics)
        self.assertIn('f1_score', metrics)
        self.assertIn('mean_absolute_error', metrics)

if __name__ == '__main__':
    unittest.main()
```

Integration Test for the Entire Pipeline

```
# test_pipeline.py

import unittest
import subprocess

class TestPipeline(unittest.TestCase):
    def test_pipeline(self):
        scripts = ['analyze_model.py', 'data_preparation.py',
'

model_development.py', 'update_api.py']
        for script in scripts:
            result = subprocess.run(['python', script], captu
re_output=True, text=True)
            self.assertEqual(result.returncode, 0, f"Failed r
unning {script}")
```

```
if __name__ == '__main__':
    unittest.main()
```

Final Script to Run the Entire Pipeline

```
# run_pipeline.py

import subprocess

def run_script(script_name):
    result = subprocess.run(['python', script_name], capture_
output=True, text=True)
    if result.returncode != 0:
        print(f"Error running {script_name}: {result.stder
r}")
    else:
        print(f"Successfully ran {script_name}: {result.stdou
t}")

if __name__ == "__main__":
    scripts = ['analyze_model.py', 'data_preparation.py', 'mo
del_development.py', 'update_api.py']
    for script in scripts:
        run_script(script)
```

Environment Setup Scripts

Batch Script: `setup_phase1.bat`

```
@echo off
python -m venv venv
call venv\\Scripts\\activate
pip install pandas scikit-learn xgboost Flask joblib
echo Virtual environment and dependencies setup complete.
pause
```

Shell Script: `setup_phase1.sh`

```
#!/bin/bash
python3 -m venv venv
source venv/bin/activate
pip install pandas scikit-learn xgboost Flask joblib
echo "Virtual environment and dependencies setup complete."
```

Documentation

`README.md`

```
# Property Valuation Model Enhancement

## Scripts

### 1. analyze_model.py
- Reviews the current model's performance using various evalu
ation metrics.
- **Usage:** `python analyze_model.py`

### 2. data_preparation.py
- Loads additional data sources and performs feature engineer
ing.
- **Usage:** `python data_preparation.py`

### 3. model_development.py
- Trains new models with advanced algorithms and evaluates th
eir performance.
- **Usage:** `python model_development.py`

### 4. update_api.py
- Sets up a Flask application with updated API endpoints to u
tilize the new model.
- **Usage:** `python update_api.py`
```

```
### 5. test_analyze_model.py
- Unit test for `analyze_model.py`.
- **Usage:** `python -m unittest test_analyze_model.py`

### 6. test_data_preparation.py
- Unit test for `data_preparation.py`.
- **Usage:** `python -m unittest test_data_preparation.py`

### 7. test_model_development.py
- Unit test for `model_development.py`.
- **Usage:** `python -m unittest test_model_development.py`

### 8. test_pipeline.py
- Integration test for the entire pipeline.
- **Usage:** `python -m unittest test_pipeline.py`

### 9. run_pipeline.py
- Runs the entire pipeline.
- **Usage:** `python run_pipeline.py`

## Dependencies
- pandas
- scikit-learn
- xgboost
- Flask
- joblib

## Environment Setup

1. Create a virtual environment:

```bash
python -m venv venv
```

1. Activate the virtual environment:
```

- On Windows:

```
venv\\\\Scripts\\\\activate
```

- On macOS/Linux:

```
source venv/bin/activate
```

1. Install dependencies:

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

1. Run the pipeline:

```
python run_pipeline.py
```

````
### Implementation Steps

1. **Set Up Virtual Environment and Install Dependencies**

**Batch Script: `setup_phase1.bat`**

```batch
@echo off
python -m venv venv
call venv\\Scripts\\activate
pip install pandas scikit-learn xgboost Flask joblib
echo Virtual environment and dependencies setup complete.
pause
````

Shell Script: `setup_phase1.sh`
````

```
#!/bin/bash
python3 -m venv venv
source venv/bin/activate
pip install pandas scikit-learn xgboost Flask joblib
echo "Virtual environment and dependencies setup complete."
```

1. **Execute the Setup Script**

- On Windows, run `setup_phase1.bat`.
- On macOS/Linux, run `setup_phase1.sh`.

1. **Run the Pipeline**

```
python run_pipeline.py
```

This finalized Phase 1 implementation ensures a comprehensive and robust approach to enhancing the property valuation AI model. All scripts are equipped with error handling, logging, unit tests, integration tests, and a detailed setup process. This will facilitate smooth development, testing, and deployment of the enhanced model.