**Automated Valuation Model (AVM) - Project Documentation**

**Prepared by:** Mahmoud Ayman Kharoof

**Project Overview**

The Automated Valuation Model (AVM) leverages AI and machine learning to estimate real estate property values in the Dubai market. This project integrates data preprocessing, feature selection, model implementation, meta-learning, and Docker containerization to streamline property valuation using cutting-edge AI algorithms.

**Environment Setup**

**1. Install WSL**

* Ensure you have WSL2 with Ubuntu installed.
* Update the WSL environment:

sudo apt update && sudo apt upgrade -y

**2. Install Python**

* Install Python 3.10 or higher:

sudo apt install python3 python3-venv python3-pip -y

**3. Install Docker**

* Install Docker CE:

sudo apt install docker-ce docker-ce-cli containerd.io -y

* Configure Docker to run without sudo:

sudo usermod -aG docker $USER

newgrp docker

**Folder Structure**

The project adheres to the following structure for clarity and modularity:

plaintext

notebooks/

├── preprocess.py          # Data preprocessing pipeline

├── feature\_selection.py   # Feature selection logic

├── base\_models.py         # Base model implementation

├── meta\_learner.py        # Meta-learner implementation

├── main.py               # Orchestrates the entire pipeline

├── evaluation.py         # Model evaluation metrics

├── config.py             # Configuration parameters

├── Dockerfile            # Docker container setup

├── requirements.txt      # Python dependencies

└── test.py              # Unit tests

data/

├── snp\_dld\_2024\_transactions.parquet       # Sales data raw I changed it from csv to parquet to reduce storage consumption

├── snp\_dld\_2024\_transactions\_cleaned.parquet      # Sales data cleaned

├── snp\_dld\_2024\_rents.parquet      # Rent data raw I changed it from csv to parquet to reduce storage consumption

└── snp\_dld\_2024\_rents\_cleaned.parquet # Rent data cleaned

Dockerfile            # Docker container setup

README.md            # Documentation

**Data Preprocessing**

**1. Workflow Overview**

The preprocess.py script manages two primary data types:

* **Rental Data**
* **Sales Data**

**Steps:**

1. Load raw data (CSV/Parquet).
2. Clean and standardize the data.
3. Handle missing values.
4. Process special columns (e.g., rooms, parking).
5. Engineer features (e.g., dates, durations).
6. Encode categorical variables.
7. Scale numerical features.
8. Save processed data.

**2. Transition from CSV to Parquet**

Due to the large size of the original CSV files, I switched to the Parquet format, which offers the following advantages:

* **Compression:** Parquet supports built-in compression, reducing file size.
* **Columnar Storage:** Facilitates faster query performance, especially for analytical tasks.
* **Scalability:** Handles large datasets more efficiently in both storage and computation.
* **Interoperability:** Well-supported by modern data science libraries such as Pandas and PySpark.

The data pipeline was updated to seamlessly convert CSV files to Parquet during initial ingestion.

def load\_and\_convert\_csv\_to\_parquet(csv\_path, parquet\_path):

data = pd.read\_csv(csv\_path)

data.to\_parquet(parquet\_path, index=False)

return parquet\_path

**Feature Selection**

The feature selection pipeline integrates statistical methods, machine learning models, and recursive elimination.

**Key Methodologies and Technical Insights:**

**1. Correlation Analysis**

* Identifies highly correlated features that can cause multicollinearity in models.
* Removes features where the correlation coefficient exceeds a configurable threshold.
* Ensures model stability and interpretability.

**2. Tree-Based Importance (Random Forest)**

* Leverages feature importance scores from a Random Forest model.
* Handles non-linear relationships effectively.
* Provides ranked features based on contribution to predictive performance.

**3. Recursive Feature Elimination (RFE)**

* Iteratively removes the least important features to identify the optimal subset.
* Uses Random Forest regressors for feature ranking, ensuring robustness.

**4. Univariate Selection**

* Evaluates individual features using statistical tests like ANOVA (F-test).
* Identifies features with the strongest relationship to the target variable.

**5. Combined Strategy**

* Combines outputs from all methods, prioritizing Random Forest importance.
* Removes redundancy from correlated features while preserving meaningful ones.

**Base Models**

The base\_models.py script includes implementations of **XGBoost**, **Random Forest**, and **Support Vector Regression (SVR)**.

**Technical Details and Model Explanations:**

**1. XGBoost**

* **Type:** Gradient-boosted decision trees.
* **Advantages:**
  + Handles missing data internally.
  + Highly customizable with options like tree depth, learning rate, and subsampling.
* **Use Case:** Provides robust performance on structured tabular data.

model = XGBRegressor(

max\_depth=10,

learning\_rate=0.05,

n\_estimators=200,

objective='reg:squarederror'

)

**2. Random Forest**

* **Type:** Ensemble of decision trees trained on bootstrapped samples.
* **Advantages:**
  + Reduces overfitting compared to individual trees.
  + Provides feature importance out-of-the-box.
* **Use Case:** Effective for handling non-linear interactions.

model = RandomForestRegressor(

n\_estimators=100,

max\_depth=20,

min\_samples\_leaf=5

)

**3. Support Vector Regression (SVR)**

* **Type:** Kernel-based method for regression tasks.
* **Advantages:**
  + Works well with smaller datasets or high-dimensional spaces.
  + Provides flexibility via kernel functions (linear, RBF).
* **Use Case:** Suitable for capturing complex patterns in rental price data.

python

model = SVR(kernel='rbf', C=1.0, epsilon=0.1)

**Meta-Learner Implementation**

The meta\_learner.py combines predictions from base models into a single output using a **Neural Network**.

**Technical Insights:**

1. **Ensemble Learning:**
   * Takes predictions from XGBoost, Random Forest, and SVR.
   * Combines them using a meta-model for improved generalization.
2. **Neural Network Architecture:**
   * Input: Predictions from base models.
   * Hidden Layers: Configurable with dropout for regularization.
   * Optimizer: Adam, with Mean Squared Error (MSE) as the loss function.
3. **Hyperparameter Optimization:**
   * Uses Bayesian optimization to tune architecture (hidden units, learning rate, dropout).
   * Ensures high performance without manual trial-and-error.

def \_create\_meta\_model(num\_hidden\_layers, hidden\_units, dropout\_rate):

model = Sequential()

model.add(Input(shape=(3,)))

for \_ in range(num\_hidden\_layers):

model.add(Dense(hidden\_units, activation='relu'))

model.add(Dropout(dropout\_rate))

model.add(Dense(1)) # Single output

model.compile(optimizer='adam', loss='mse')

return model

**Evaluation Metrics**

The evaluation.py script assesses model performance using:

**1. Metrics:**

* **RMSE (Root Mean Square Error):** Penalizes larger errors more heavily.
* **R² Score:** Measures the proportion of variance explained.
* **MAE (Mean Absolute Error):** Provides an interpretable average error.

**2. Visualization:**

* **Residual Plots:** Highlight systematic errors.
* **Actual vs Predicted Scatter Plots:** Show model fit quality.

**Pipeline Orchestration**

The main.py script orchestrates all components, from preprocessing to evaluation.

**Pipeline Highlights:**

1. **Dual Processing:** Handles rental and sale datasets separately for targeted models.
2. **Logging:** Captures errors, execution times, and key events.
3. **Meta-Learning:** Finalizes predictions through ensemble strategies.

**Docker Setup**

**1. Create Dockerfile**

Dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

CMD ["python", "main.py"]

**2. Build Image**

docker build -t avm-app .

**3. Run Container**

docker run -it avm-app

**Key Achievements**

1. **Scalability:** Transitioned to Parquet for handling large datasets.
2. **Robustness:** Incorporated ensemble learning with hyperparameter optimization.
3. **Comprehensiveness:** Combined advanced models with detailed evaluation for reliable predictions.