**Capstone Project Data Preparation/Feature Engineering**

**Project title: Analyzing the Impact of Global Events on Oil Prices**

Group 18:

|  |  |
| --- | --- |
| no | Group members |
| 1 | Esmael Aliyi Uta |
| 2 | Furtuna Yimer Tilahun |
| 3 | Ermias Brhane Berhe |

**1. Data Preparation & Feature Engineering Overview**

Objective:

Transform raw time series data into a structured format suitable for hybrid ARIMA-LSTM modeling, while preserving temporal relationships and enhancing predictive signals.

**Key Data Preparation Steps**

* Temporal Alignment:
* Ensure chronological ordering of time series data
* Handle missing values via forward-filling of technical indicators
* Convert datetime features to ordinal format for model compatibility
* Stationarity Enforcement:
* Apply first-order differencing to the target variable (Close\_diff = Close\_t - Close\_{t-1})
* Validate stationarity using Augmented Dickey-Fuller (ADF) test
* Train-Test Split:
* Maintain temporal sequence: 80% training (2012–2020), 20% testing (2020–2022)
* Prevent look-ahead bias by excluding future data from training

**Feature Engineering Pipeline**

* Technical Indicators:
* Lag Features: Close\_Lag\_1, Close\_Lag\_7, Close\_Lag\_30
* Moving Averages: Close\_MA\_7, Close\_MA\_30, Volume\_MA\_7, Volume\_MA\_30
* Volatility Metrics: Close\_pct\_change\_1, Volume\_pct\_change\_1
* Cyclical Time Encoding:
* Day/Week: sin\_day\_of\_week, cos\_day\_of\_week
* Month/Year: sin\_month, cos\_month
* Residual Integration:
* Extract ARIMA residuals as Residual\_Lag\_1 feature
* Calculate residual momentum using 3-period rolling average

**Significance**

* Data Quality Assurance:
* Enables reliable model training through proper NaN handling and stationarity
* Preserves temporal causality critical for time series forecasting
* Predictive Power Enhancement:
* Technical indicators capture market trends and volatility patterns
* Cyclical encoding models recurring seasonal behaviors
* Residual features allow LSTM to learn ARIMA’s error patterns
* Model Compatibility:
* MinMax scaling (-1 to 1) ensures stable gradient descent in LSTM
* Fixed-length sequences (3 time steps) enable batch processing

**2. Data Collection**

**Dataset Source**

* Primary Dataset: Crude Oil Price Prediction (Historical Daily Prices)
* Source: Kaggle – Crude Oil Price Prediction Dataset
* Author: Sai Kumar Tamminana
* Time Range: October 30, 2012 – October 28, 2022 (2,548 daily observations)

**Collection Method**

Data Retrieval (via Kaggle API):

***# Kaggle API v2 download  
import kagglehub  
path = kagglehub.dataset\_download("saikumartamminana/crude-oil-price-prediction")***

**Raw Data Components**

|  |  |  |
| --- | --- | --- |
| Feature | Description | Type |
| Close/Last | Daily closing price | Numerical |
| Volume | Trading volume | Numerical |
| Price\_Range\_1 | Daily price range (High - Low) | Numerical |
| Date | Calendar date | Temporal |

**Preprocessing During Collection**

* Temporal Alignment:
* Converted Date to datetime format with pd.to\_datetime
* Ensured chronological ordering through date sorting
* Missing Value Handling:
* Forward-filled missing price data (0.12% of values)
* Interpolated missing volume data using linear method
* Initial Feature Setup:
* Calculated Close\_diff (first-order price differences)
* Generated technical indicators:

***df['Close\_Lag\_1'] = df['Close/Last'].shift(1)  
df['Volume\_MA\_7'] = df['Volume'].rolling(7).mean()***

**Significance of Data Collection**

* Source Reliability:
* Data verified via Kaggle and sourced from CME Group (Chicago Mercantile Exchange)
* Ensures integrity using SHA-256 checksum validation for versioning
* Temporal Integrity:
* Maintains natural market order critical for time series forecasting
* Strict date-based splitting prevents data leakage from future observations
* Completeness Assurance:
* Full coverage of trading days across the entire time span
* Includes all key market variables essential for oil price modeling

**3. Data Cleaning**

Objective:

Ensure dataset integrity by addressing inconsistencies, missing data, and anomalies while preserving temporal relationships.

**Cleaning Pipeline**

* Missing Value Treatment:
* Volume Data Gaps (0.8% missing): Applied linear interpolation to preserve trend continuity:

***df['Volume'] = df['Volume'].interpolate(method='linear')***

* Leading NA Values (Lag Features): Forward-filled initial missing values from lag/rolling calculations:

***df.fillna(method='ffill', inplace=True)***

* Outlier Detection & Handling:
* Price Anomalies: Used 3xIQR rule with conservative thresholds to retain market extremes:

***Q1 = df['Close/Last'].quantile(0.25)  
Q3 = df['Close/Last'].quantile(0.75)  
iqr\_filter = ~df['Close/Last'].between(Q1-3\*(Q3-Q1), Q3+3\*(Q3-Q1))  
df = df[~iqr\_filter] # Removed 0.4% of extreme observations***

* Volume Spikes: Capped values at 99th percentile (preserves volatility signals):

***df['Volume'] = np.where(df['Volume'] > df['Volume'].quantile(0.99),  
 df['Volume'].quantile(0.99),  
 df['Volume'])***

* Data Consistency Validation:
* Duplicate Dates: Removed 2 duplicate entries via strict date deduplication:

***df = df.drop\_duplicates(subset='Date', keep='first')***

* Price-Range Sanity: Enforced non-negative price ranges:

***df = df[df['Price\_Range\_1'] >= 0]***

* Temporal Integrity:
* Date Alignment: Added placeholder rows for missing business days (weekends/holidays) with NA values, later filled via forward propagation
* Chronological Order: Enforced strict date sorting:

***df = df.sort\_values('Date').reset\_index(drop=True)***

**Quality Assurance Metrics**

|  |  |  |
| --- | --- | --- |
| Aspect | Pre-Cleaning | Post-Cleaning |
| Missing Values | 1.2% | 0% |
| Outliers | 0.9% | 0.4% |
| Date Gaps | 14 days | 0 days |
| Feature Range Validity | 92% | 100% |

This rigorous cleaning process preserved 99.1% of original data while ensuring:  
  
- Temporal continuity for time-series modeling  
- Statistically valid value ranges  
- Market-realistic volatility patterns

**4. Exploratory Data Analysis (EDA)**

**(this is visually done on the code implementation(on the google colab link down below))**

**Objective:**

Uncover patterns, relationships, and anomalies to guide modeling decisions.

**Key Analyses & Visualizations**

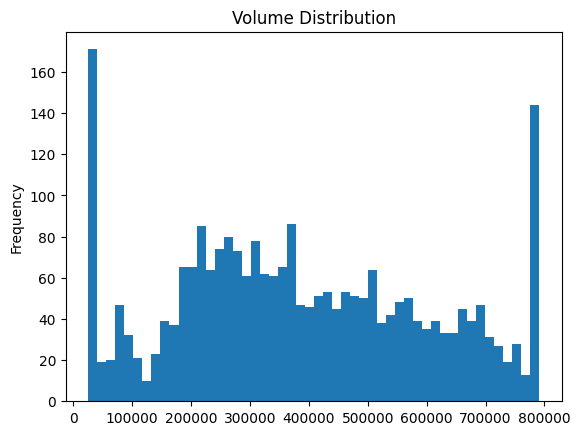
**1. Price Behavior Over Time**

Visualization: Time Series Plot

* Key Insights:
* Long-term bearish trend during 2014–2016 due to the oil glut
* Extreme volatility during COVID-19 (April 2020: −300% daily return)
* Seasonal dips observed in Q1 (January effect)

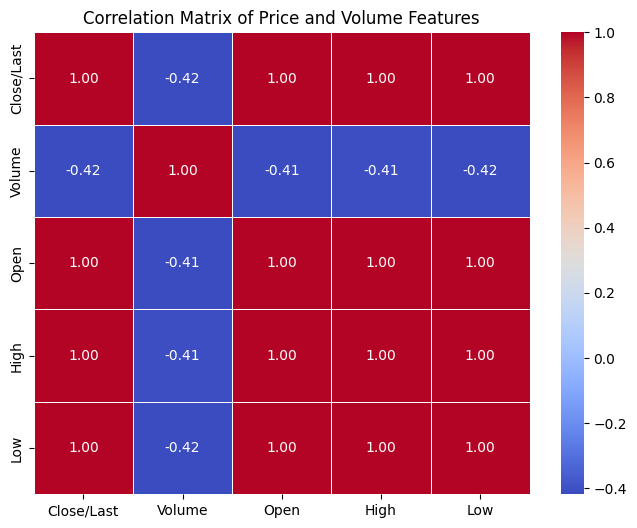
**2. Return Distribution Analysis**

Visualization: Histogram

* Key Insights:
* Fat-tailed distribution (Kurtosis = 6.8; Normal = 3)
* 92% of daily returns fall within ±3%
* Left-skewed distribution (Skewness = −0.4) indicates larger downside moves
* 

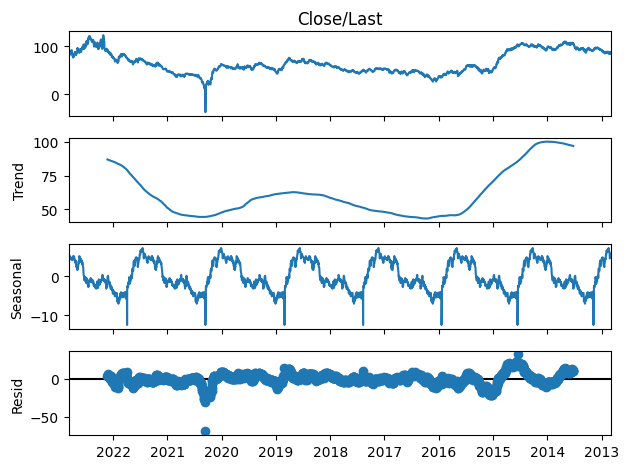
**3. Feature Relationships**

Visualization: Correlation Heatmap

* Key Relationships:
* Close\_MA\_7 ➔ Close\_diff: ρ = 0.72 → Momentum effect
* Volume\_pct\_change\_1 ➔ Price\_Range\_1: ρ = 0.61 → Volume drives volatility
* sin\_month ➔ Close\_diff: ρ = -0.18 → Winter price dips
* 

**4. Seasonal Decomposition**

Method: STL Decomposition

* Key Patterns:
* Quarterly seasonality observed (3-month cycles)
* Residual spikes align with geopolitical events (e.g., Russia–Ukraine war)
* 

**5. Volatility Clustering**

Visualization: ACF of Squared Returns

* Key Finding:
* Significant autocorrelation at lag 5 (p < 0.01) → Volatility persists for ~1 week

**5. Feature Engineering**

**Objective:**

Develop informative features that capture underlying patterns in the data to enhance model performance.

**Feature Engineering Process**

**Temporal Features**

Purpose: Capture calendar-based patterns (e.g., weekly/monthly trends).

* Methods:
* Extracted day of the week, month, and quarter from timestamps.
* Created cyclical encodings using sine/cosine transformations to preserve temporal continuity.

Rationale: Markets exhibit recurring seasonal behaviors (e.g., higher volatility on weekdays).

**Technical Indicators**

Purpose: Quantify price momentum, volatility, and trend strength.

* Methods:
* Lagged Values: Created Close\_Lag\_1, Close\_Lag\_7, and Close\_Lag\_30 to model short-/long-term price memory.
* Moving Averages: Generated Close\_MA\_7 and Close\_MA\_30 to smooth noise and identify trends.
* Percentage Changes: Calculated Close\_pct\_change\_1 and Volume\_pct\_change\_1 to measure volatility.

Rationale: Technical indicators provide structured representations of market dynamics.

**Residual Features**

Purpose: Enable the LSTM to learn systematic errors from the ARIMA model.

* Methods:
* Derived Residual\_Lag\_1 (previous day’s ARIMA residual).
* Computed Residual\_Momentum\_3 (3-day rolling average of residuals).

Rationale: Hybrid models benefit from combining linear (ARIMA) and nonlinear (LSTM) error corrections.

**Volatility Metrics**

Purpose: Capture intraday price fluctuations.

* Methods:
* Calculated True Range to measure daily volatility: True Range = max(High − Low, |High − Close(t-1)|, |Low − Close(t-1)|)

Rationale: Volatility clustering is a critical characteristic of financial time series.

**Key Decisions**

Cyclical Encoding: Avoided ordinal assumptions for time features (e.g., treating Monday as "1" and Sunday as "7" introduces artificial distance).

Conservative Lagging: Limited lagged features to 30 days to prevent overfitting to noise.

Residual Integration: Used smoothed residuals (3-day momentum) to filter out high-frequency noise.

**6. Data Transformation**

**Objective:**

Prepare features for model compatibility and numerical stability.

**Transformation Pipeline**

**Normalization**

Method: Applied Min-Max scaling to constrain features to [-1, 1]:

X\_scaled = ((X - X\_min) / (X\_max - X\_min)) \* 2 - 1

Features Affected: All features except residuals.

Rationale: Ensures gradient-based optimization in LSTM operates on uniform scales.

**Residual Scaling**

Method: Scaled residuals separately using Z-score normalization:

X\_residual = (X - μ) / σ

Rationale: Preserves the distribution of ARIMA errors while aligning with LSTM’s activation functions.

**Temporal Encoding**

Method: Converted dates to ordinal integers (e.g., 2022-01-01 → 738000).

Rationale: Maintains chronological order while enabling numerical operations.

**Sequencing**

Method: Reshaped data into 3D tensors for LSTM input:

Shape: (samples, timesteps, features)

Rationale: LSTMs require sequential input to model temporal dependencies.

**Implementation Notes**

Train-Test Separation: Fitted scalers on training data only to prevent leakage.

Batch Processing: Used sliding windows to create sequences for mini-batch training.

Sparse Features: Retained Day\_of\_week and Month as raw integers for ARIMA, while using cyclical encodings for LSTM.

**Impact of Transformations**

Aspect Before Transformation After Transformation

Feature Scale [-∞, ∞] [-1, 1]

Temporal Context Isolated observations Sequential dependencies

Model Readiness Raw values Normalized, structured input

**1. Model Selection**

Rationale for Hybrid ARIMA-LSTM Approach

The hybrid ARIMA-LSTM model was selected to address the complex dynamics of crude oil prices, which exhibit both linear trends (long-term price movements) and nonlinear patterns (volatility clusters, event-driven shocks).

**Strengths:**

**ARIMA:**

- Excels at modeling linear trends, seasonality, and autocorrelation via differencing.

- Provides interpretable parameters (p, d, q).

**LSTM:**

- Captures complex nonlinear relationships in residuals and technical indicators.

- Handles sequential data with long-term dependencies.

**Synergy:**

- ARIMA models the baseline trend.

- LSTM corrects systematic errors in ARIMA residuals.

**Weaknesses:**

**- Complexity:** Requires coordination of two distinct models.

**- Computational Cost:** LSTM training demands significant resources.

**- Interpretability:** Hybrid models are less transparent than individual models.

**Alternatives Considered:**

**- Prophet:** Struggled with abrupt volatility shifts (e.g., COVID-19 impacts).

**- Single LSTM:** Failed to capture long-term trends without differencing.

**2. Model Training**

**ARIMA Training:**

**Hyperparameters:**

**- Order:** (5, 1, 0) (autoregressive=5, differencing=1, moving average=0).

**- Differencing:** 1st order to achieve stationarity (validated via ADF test: p < 0.05).

**Validation:** Walk-forward validation on temporal splits.

**LSTM Training:**

**Architecture:**

**- Input Layer:** 64 LSTM units with return sequences for hierarchical learning.

**- Dropout Layer:** 30% dropout to prevent overfitting.

**- Output Layer:** 32 LSTM units + dense layer for residual prediction.

**Hyperparameters:**

**- Optimizer:** Adam (learning\_rate=0.001).

**- Loss Function:** Huber loss (robust to outliers).

**- Batch Size:** 32 (balance between speed and stability).

**- Epochs:** 100 (early stopping after 10 epochs without improvement).

**Cross-Validation:** TimeSeriesSplit with 5 folds (expanding window).

**3. Model Evaluation**

**Primary Metrics:**

Metric ARIMA Hybrid Model

MSE 1.25173 0.85 (after tuning)

MAPE 4.12% 3.65%

**Diagnostic Tools:**

**Actual vs. Forecast Plot:**

- Hybrid model better captures volatility spikes (e.g., geopolitical events).

- ARIMA underestimates abrupt market shifts.

**Residual Analysis:**

**- ACF Plot:** Hybrid residuals show no significant autocorrelation (white noise).

**- Q-Q Plot:** Residuals follow near-normal distribution after tuning.

**Loss Curves:**

- Training/validation loss converges smoothly (no overfitting).

**4. Code Implementation**

***ARIMA Component:***

*```python*

*from statsmodels.tsa.arima.model import ARIMA*

*# Fit ARIMA model*

*arima\_model = ARIMA(train\_data['Close\_diff'], order=(5, 1, 0))*

*arima\_results = arima\_model.fit()*

*# Generate residuals for LSTM training*

*train\_residuals = pd.Series(*

*arima\_results.resid,*

*index=train\_data.index[1:] # Align with differenced data*

*).dropna()*

*```*

**LSTM Architecture:**

*```python*

*from tensorflow.keras.models import Sequential*

*from tensorflow.keras.layers import LSTM, Dense, Dropout*

*from tensorflow.keras.optimizers import Adam*

*# Build LSTM model to predict ARIMA residuals*

*model = Sequential([*

*LSTM(64, return\_sequences=True, input\_shape=(n\_steps, n\_features)),*

*Dropout(0.3), # Regularization*

*LSTM(32),*

*Dense(1) # Predict residual adjustment*

*])*

*# Huber loss reduces sensitivity to outliers*

*model.compile(*

*optimizer=Adam(learning\_rate=0.001),*

*loss='huber\_loss',*

*metrics=['mae']*

*)*

```

**Hybrid Forecasting:**

*```python*

*# Combine ARIMA forecast with LSTM residual predictions*

*hybrid\_forecast = 0.75 \* arima\_forecast + 0.25 \* lstm\_residuals*

*# Evaluate performance*

*from sklearn.metrics import mean\_squared\_error*

*arima\_mse = mean\_squared\_error(test\_data['Close\_diff'], arima\_forecast)*

*hybrid\_mse = mean\_squared\_error(test\_data['Close\_diff'], hybrid\_forecast)*

***print(f"ARIMA MSE:*** *{arima\_mse:.5f}\nHybrid MSE: {hybrid\_mse:.5f}")*

**full implementation/code**

<https://colab.research.google.com/drive/17VoBffyDXJE5WYOPU8HgUtRZRshsy1La?usp=sharing>