FINAL PROJECT REPORT

1. INTRODUCTION

1.1 PROJECT OVERVIEW

This project aims to develop a machine learning model to accurately forecast natural gas prices. By utilizing historical price data and advanced machine learning techniques, the project addresses the challenges of market volatility and the limitations of current forecasting methods. The goal is to provide a reliable tool for energy company planners and financial advisors, enabling them to make informed, confident, and cost-effective decisions.

1.2 OBJECTIVES

The objective of this project is to develop a machine learning model capable of accurately predicting future natural gas prices. By Utilizing historical price data and advanced predictive algorithms, the model aims to provide valuable insights for stakeholders in the energy sector, enabling them to make informed decisions regarding pricing, procurement, and investment strategies.

2. PROJECT INITIALIZATION AND PLANNING PHASE

2.1 DEFINE PROBLEM STATEMENT

Develop a Machine Learning Model to accurately forecast natural gas prices, addressing the challenges of market volatility and unreliable current methods, to aid energy planners, Natural gas traders and financial advisors in making informed, confident, and cost-effective decisions.

2.2 PROJECT PROPOSAL(PROPOSED PROPOSAL)

1.**Data Collection**

2.**Data Preprocessing**

• Reading the Dataset

• Handling Missing Values

• Label Encoding and One Hot Encoding

• Data Visualization

• Splitting Dataset into Dependent and Independent Variable.

• Splitting Dataset into Train Set and Test Set

3.**Model Building**

• Train the Model with Descision Tree Algorithm

• Test the Model

4.**Application Building**

• Build HTML Page

• WEB Page

• Build Python Code

• Run the App

• Output

Advanced Feature Engineering:

• **Time-Based Features**: Utilize detailed time-based features (day, month, year, day of the week) to capture seasonal and temporal patterns in natural gas prices.
• **Lagged Features and Rolling Statistics**: Incorporate past price data through lagged features and rolling statistics (e.g., moving averages, standard deviations) to detect trends and volatility, enhancing predictive accuracy.

User-Friendly Deployment:

• **API and Interface Development**: Develop an intuitive API and user interface using frameworks like Flask allowing users to easily access and interpret predictions in real-time.

• **Interactive Visualization**: Provide interactive visualizations for users to explore historical data and forecast results, aiding in better decision-making.

Practical Utility for Stakeholders:

• **Customized Insights for Different Users**: Tailor the solution to provide specific insights for different stakeholders, such as energy company planners and financial advisors, enhancing its practical utility and relevance.

• **Risk Mitigation and Opportunity Identification**: Enable users to better navigate market volatility by providing predictive insights that help in mitigating risks and identifying profitable opportunities.

2.3 INITIAL PROJECT PLANNING

The initial planning phase involved setting up a detailed project roadmap with defined milestones and deliverables:

Phase 1: Data collection and preprocessing (1 day)

Phase 2: Studying Data (1 day)

Phase 3: Model Building (2 day)

Phase 4: Application Building (1 day)

Phase 5: Final Reports (1 day)

3. DATA COLLECTION AND PREPROCESSING PHASE

3.1 DATA COLLECTION PLAN AND RAW DATA SOURCES IDENTIFIED

Data Collection Plan

1.Skill Wallet Platform

2.Kaggle

Raw Data Sources Identified

Monthly and Daily Prices of Natural gas, starting from January 1997.

monthly_csv.csv (17.05 kB), Kaggle.

daily_csv.csv (85kB), Skill Wallet.

3.2 DATA QUALITY REPORT

A thorough data quality report was generated to assess the initial dataset's completeness, consistency:

Missing values in the dataset: Use imputation techniques like mean or median to fill missing values

Inconsistent data formats (dates, numerical values): Standardize data formats during preprocessing

3.3 DATA EXPLORATION AND PREPROCESSING

Data Overview

```
#dimensions and structure
data.shape
```

```
(5953, 2)
```

```
data.head()
```

	Date	Price
0	1997-01-07	3.82
1	1997-01-08	3.80
2	1997-01-09	3.61
3	1997-01-10	3.92
4	1997-01-13	4.00

```
data.tail()
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5953 entries, 0 to 5952
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5953 non-null   object
 1   Price   5952 non-null   float64
dtypes: float64(1), object(1)
memory usage: 93.1+ KB
```

```
#Basic statistics
data.describe()
```

Univariate Analysis

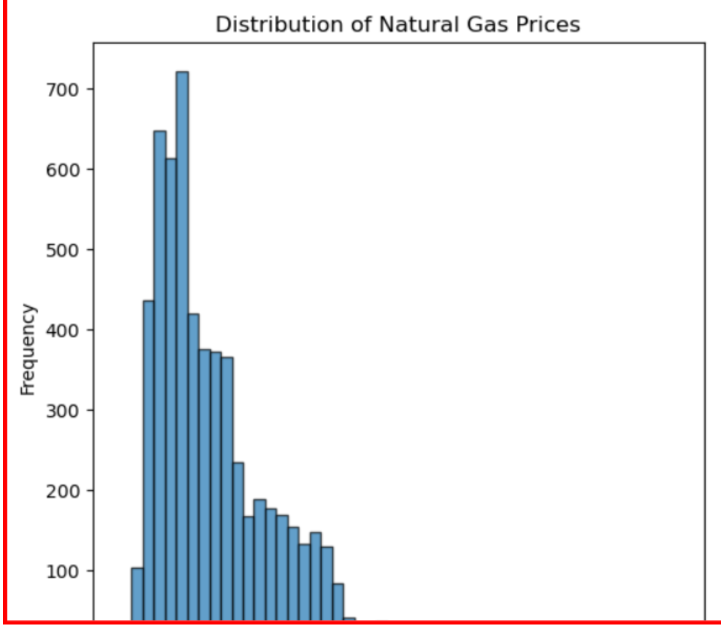
```
#Exploration of individual varibles
print("Mean:", data['Price'].mean())
print("Median:", data['Price'].median())
print("Mode:", data['Price'].mode())
```

```
Mean: 4.184643817204301
Median: 3.53
Mode: 0    2.75
Name: Price, dtype: float64
```

Bivariate Analysis

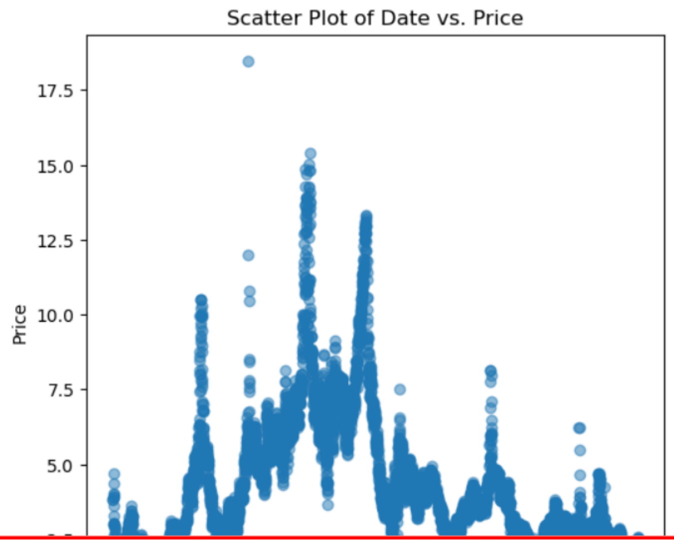
```
import matplotlib.pyplot as plt

# Distribution of natural gas prices
plt.figure(figsize=(6, 6))
plt.hist(data['Price'], bins=50, edgecolor='k', alpha=0.7)
plt.title('Distribution of Natural Gas Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

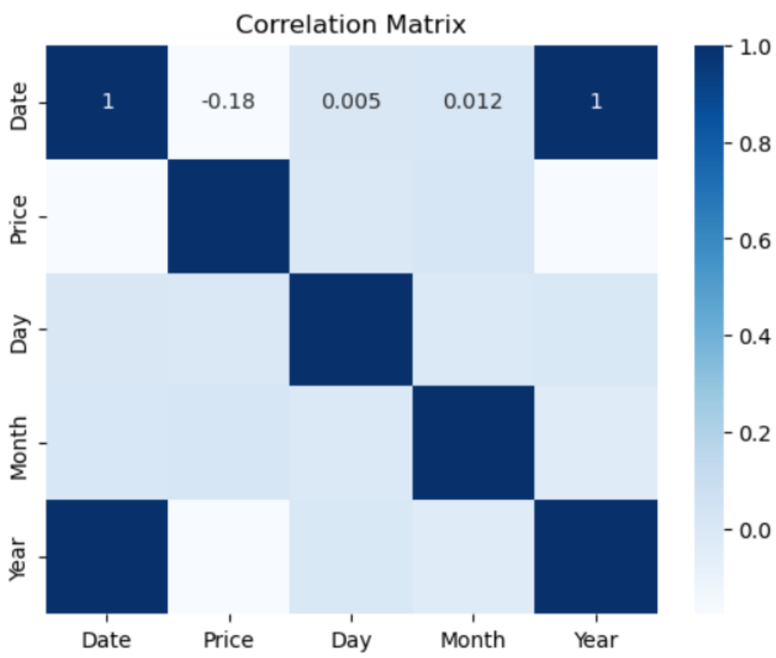


```python
import seaborn as sns

# Scatter plot between Date and Price
plt.figure(figsize=(6, 6))
plt.scatter(data['Date'], data['Price'], alpha=0.5)
plt.title('Scatter Plot of Date vs. Price')
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```



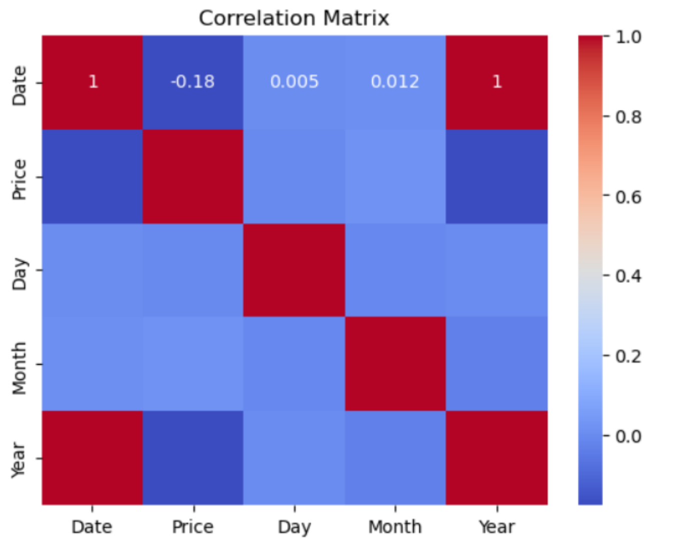

```python
# Correlation matrix
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='Blues')
plt.title('Correlation Matrix')
plt.show()
```



Multivariate Analysis

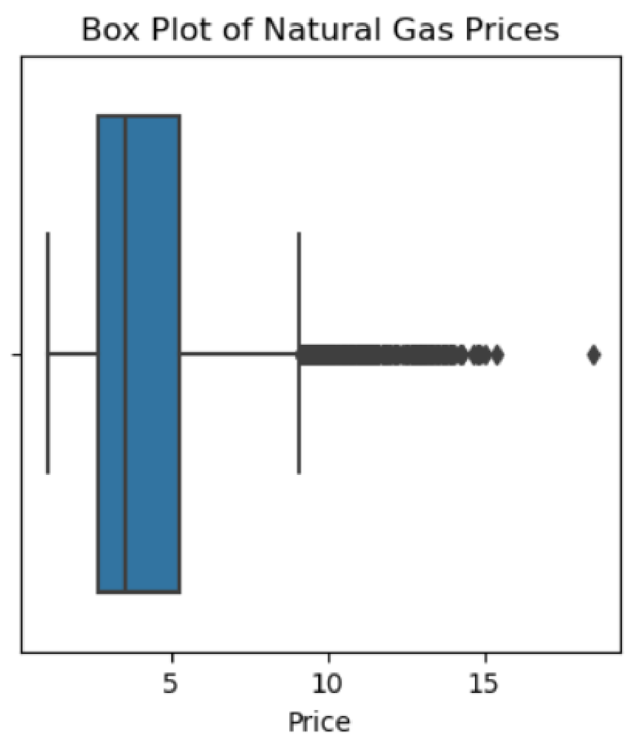
```python
# Pair plot to see pairwise relationships between multiple variables
sns.pairplot(data)
plt.show()
```

```python
# Correlation matrix for multivariate analysis
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Outliers and Anomalies

```python
# Box plot to identify outliers in price
plt.figure(figsize=(4, 4))
sns.boxplot(x=data['Price'])
plt.title('Box Plot of Natural Gas Prices')
plt.show()
```



```
# Removing outliers
Q1 = data['Price'].quantile(0.25)
Q3 = data['Price'].quantile(0.75)
IQR = Q3 - Q1
```

```
ata = data[(data['Price'] >= Q1 - 1.5 * IQR) & (data['Price'] <= Q3 + 1.5 * IQR)]
print("Shape after removing outliers:", data.shape)
```

```
Shape after removing outliers: (5953, 5)
```

Data Preprocessing Code Screenshots

```
#importing the libraries
import numpy as np
import pandas as pd
```

```
#loading the dataset
data=pd.read_csv(r"C:\Users\vyshn\Downloads\daily_csv.csv")
data
```

```
data.sort_values('Date', inplace=True)
```

```
# Convert the 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'])
```

```
# Extract day, month, and year into separate columns
data['Day'] = data['Date'].dt.day
data['Month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year
```

```
# Identify missing values
print(data.isnull().sum())
```

```
Date     0
Price    1
Day      0
Month    0
Year     0
dtype: int64
```

```
print(data.isnull().any())
```

```
Date     False
Price     True
Day      False
Month    False
Year     False
dtype: bool
```

```
data['Price'].fillna(data['Price'].mean(),inplace=True)
```

```python
print(data.isnull().any())
```

```
Date      False
Price     False
Day       False
Month     False
Year      False
dtype: bool
```

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Initialize scaler
minmax_scaler = MinMaxScaler()
```

```python
# Assuming 'Price' is the column to be sca
data['Price'] = minmax_scaler.fit_transform(data[['Price']])
```

```python
data
```

	Date	Price	Day	Month	Year
0	1997-01-07	0.158921	7	1	1997
1	1997-01-08	0.157774	8	1	1997
2	1997-01-09	0.146873	9	1	1997
3	1997-01-10	0.164659	10	1	1997
4	1997-01-13	0.169248	13	1	1997

```python
#feature engineering
# Create lagged features
data['Price_lag1'] = data['Price'].shift(1)
data['Price_lag7'] = data['Price'].shift(7)
```

```python
# Create rolling mean features
data['Price_rolling_mean7'] = data['Price'].rolling(window=7).mean()
```

```python
# Drop rows with NaN values generated by lagging
data.dropna(inplace=True)
```

```python
#Save Processed Data
data.to_csv('preprocessed_natural_gas_prices.csv', index=False)
```

```python
# Verify by loading the saved file
processed_data = pd.read_csv('preprocessed_natural_gas_prices.csv')
print(processed_data.head())
```

```
      Price  Day  Month  Year  Price_lag1  Price_lag7  Price_rolling_mean7
0  0.209983   16      1  1997    0.188755    0.158921             0.172445
1  0.164085   17      1  1997    0.209983    0.157774             0.173346
2  0.126793   20      1  1997    0.164085    0.146873             0.170478
3  0.111302   21      1  1997    0.126793    0.164659             0.162856
4  0.114745   22      1  1997    0.111302    0.169248             0.155069
```

```python
pd.read_csv(r'preprocessed_natural_gas_prices.csv')
```

	Price	Day	Month	Year	Price_lag1	Price_lag7	Price_rolling_mean7
0	0.209983	16	1	1997	0.188755	0.158921	0.172445
1	0.164085	17	1	1997	0.209983	0.157774	0.173346
2	0.126793	20	1	1997	0.164085	0.146873	0.170478
3	0.111302	21	1	1997	0.126793	0.164659	0.162856
4	0.114745	22	1	1997	0.111302	0.169248	0.155069
...	...	...	...	...	...	...	...

4.Model Development Phase

4.1. Feature Selection Report

Price_lag1

The price one day before.

Captures the immediate past value of the price, which can be useful in predicting the next day's price.

Price_lag7

The price seven days before.

Captures weekly patterns and trends in the data, providing a longer-term perspective.

Price_rolling_mean7

The 7-day rolling mean of the price.

Helps smooth out short-term fluctuations and highlight longer-term trends.

4.2 Model Selection Report

Model	Description	Hyperparameters	Performance Metric (e.g., Accuracy, F1 Score)

Linear Regression	Linear Regression predicts a continuous target variable by fitting a linear relationship between input features and the target variable.	null	`from sklearn.linear_model import LinearRegression`
			`# Initialize and train the Linear Regression model`
			`lr_model = LinearRegression()`
			`lr_model.fit(x_train, y_train)`
			LinearRegression
			LinearRegression()
			`# Make predictions on the test set`
			`y_pred = lr_model.predict(x_test)`
			`# Calculate evaluation metrics`
			`rmse = np.sqrt(mean_squared_error(y_test, y_pred))`
			`print(metrics.r2_score(y_test, y_pred))`
			0.9871686856443018
			`rmse`
			0.014620445192057996

Decision Tree Regressor	Decision Tree Regressor predicts continuous target values by splitting data into branches based on feature thresholds, optimizing	max_depth=5, min_samples_split=10 random_state=2	

	prediction accuracy.		# Split data into training and testing sets
			x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)
			print(x.shape,x_train.shape,x_test.shape)
			(5946, 6) (4756, 6) (1190, 6)
			# Train a Decision Tree Regressor
			model_dt = DecisionTreeRegressor(max_depth=5, min_samples_split=10, random_state=2)
			model_dt.fit(x_train, y_train)
			DecisionTreeRegressor
			DecisionTreeRegressor(max_depth=5, min_samples_split=10, random_state=2)
			#predictions
			y_pred_dt = model_dt.predict(x_test)
			# Evaluating the model
			mse_dt = mean_squared_error(y_test, y_pred_dt)
			rmse_dt = np.sqrt(mse_dt)
			rmse_dt
			0.017540079972366295
			from sklearn import metrics
			print(metrics.r2_score(y_test, y_pred_dt))
			0.9815322881744076

Random Forest Regressor	Random Forest Regressor predicts continuous values by averaging predictions from multiple decision trees, reducing overfitting and improving accuracy.	n_estimators =100, random_state = 2	# Train a Random Forest Regressor
			model_rf = RandomForestRegressor(n_estimators=100, random_state=2)
			model_rf.fit(x_train, y_train)
			RandomForestRegressor
			RandomForestRegressor(random_state=2)
			y_pred_rf = model_rf.predict(x_test)
			# Evaluate the model
			mse_rf = mean_squared_error(y_test, y_pred_rf)
			rmse_rf = np.sqrt(mse_rf)
			rmse_rf
			0.016233192522670137
			print(metrics.r2_score(y_test, y_pred_rf))
			0.9841817720586651

SVM	Support Vector Regression (SVR) uses support vectors and kernel functions to predict continuous target values with high accuracy and robustness.	kernel='rbf', C=100, gamma=0.1	# Train a Support Vector Regressor
			model_svm = SVR(kernel='rbf', C=100, gamma=0.1)
			model_svm.fit(x_train, y_train)
			SVR
			SVR(C=100, gamma=0.1)
			# Make predictions
			y_pred_svm = model_svm.predict(x_test)
			# Evaluate the model
			mse_svm = mean_squared_error(y_test, y_pred_svm)
			rmse_svm = np.sqrt(mse_svm)
			rmse_svm
			0.06461344492463574
			print(metrics.r2_score(y_test, y_pred_svm))
			0.7493915307908854

4.3 Initial Model Training Code, Model Validation and Evaluation Report

Model	Classification Report	Accuracy	Confusion Matrix

Descision Tree Classifier	data=pd.read_csv(r'preprocessed_natural_gas_prices.csv')
	
	# Create a binary target: 1 if the price went up, 0 if it went down
	data['Price_Up'] = (data['Price'].shift(-1) > data['Price']).astype(int)
	
	# Drop the last row with NaN target
	data.dropna(inplace=True)
	
	# Features and target
	features = ['Day', 'Month', 'Year', 'Price_lag1', 'Price_lag7', 'Price_rolling_mean7']
	x = data[features]
	y = data['Price_Up']
	
	from sklearn.model_selection import train_test_split, cross_val_score
	x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)
	
	# Train a Decision Tree Classifier
	from sklearn.tree import DecisionTreeClassifier
	dt_model = DecisionTreeClassifier(max_depth=5, min_samples_split=10, random_state=2)
	dt_model.fit(x_train, y_train)
	
	DecisionTreeClassifier
	DecisionTreeClassifier(max_depth=5, min_samples_split=10, random_state=2)
	
	report_dt = metrics.classification_report(y_test, y_pred)
	report_dt
	' precision recall f1-score support\n\n 0 0.55 0.75 0.63 646\n 1 0.47 0.26 0.34 544\n\n accuracy 0.53 1190\n macro avg 0.51 0.51 0.49 1190\nweighted avg 0.51 0.53 0.50 1190\n'
		# Make predictions
		y_pred = dt_model.predict(x_test)
		
		# Calculate accuracy score
		from sklearn import metrics
		accuracy = metrics.accuracy_score(y_test, y_pred)
		accuracy
		
		0.5285714285714286
			# Calculate confusion matrix
			conf_matrix = metrics.confusion_matrix(y_test, y_pred)
			conf_matrix
			
			array([[487, 159],
			[402, 142]], dtype=int64)

Random Forest Classifier	from sklearn.ensemble import RandomForestClassifier rf_model = RandomForestClassifier(n_estimators=100, random_state=2) rf_model.fit(x_train, y_train) RandomForestClassifier RandomForestClassifier(random_state=2) y_pred_rf = rf_model.predict(x_test) report_rf = metrics.classification_report(y_test, y_pred_rf) report_rf ' precision recall f1-score support\n\n 0 0.55 0.61 0.58 646\n 1 0.47 0.41 0.44 544\n\n accuracy 0.52 1190\n macro avg 0.51 0.51 0.51 1190\nweighted avg 0.52 0.52 0.52 1190\n'	accuracy_rf = metrics.accuracy_score(y_test, y_pred_rf) accuracy_rf 0.5218487394957984	conf_matrix_rf = metrics.confusion_matrix(y_test, y_pred_rf) conf_matrix_rf array([[397, 249], [320, 224]], dtype=int64)
SVM	from sklearn.svm import SVC model_svm = SVC(kernel='rbf', C=100, gamma=0.1) model_svm.fit(x_train, y_train) SVC SVC(C=100, gamma=0.1) y_pred_svc = model_svm.predict(x_test) report_svc = metrics.classification_report(y_test, y_pred_svc) report_svc ' precision recall f1-score support\n\n 0 0.53 0.55 0.54 646\n 1 0.44 0.42 0.43 544\n\n accuracy 0.49 1190\n macro avg 0.48 0.48 0.48 1190\nweighted avg 0.49 0.49 0.49 1190\n'	accuracy_svc = metrics.accuracy_score(y_test, y_pred_svc) accuracy_svc 0.4899159663865546	conf_matrix_svc = metrics.confusion_matrix(y_test, y_pred_svc) conf_matrix_svc array([[357, 289], [318, 226]], dtype=int64)

5.Model Optimization and Tuning Phase

5.1 Hyperparameter Tuning Documentation

Model	Tuned Hyperparameters	Optimal Values

Decision Tree Regressor	max_depth, min_samples_split, min_samples_leaf	# Define the model
		model = DecisionTreeRegressor()
		
		# Define the hyperparameters grid
		param_grid = {
		'max_depth': [3, 5, 7, 10],
		'min_samples_split': [2, 5, 10],
		'min_samples_leaf': [1, 2, 5]
		}
		
		# Perform grid search
		grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error')
		grid_search.fit(x, y)
		
		GridSearchCV
		estimator: DecisionTreeRegressor
		DecisionTreeRegressor
		
		# Best hyperparameters
		print(f'Best parameters: {grid_search.best_params_}')
		
		Best parameters: {'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 2}

SVR	c, epsilon, kernel	from sklearn.svm import SVR
		from sklearn.model_selection import GridSearchCV
		
		# Define the model
		svr_model = SVR()
		
		# Define the hyperparameters grid
		param_grid_svr = {
		'C': [0.1, 1, 10, 100],
		'epsilon': [0.01, 0.1, 0.2],
		'kernel': ['linear', 'poly', 'rbf', 'sigmoid']
		}
		
		# Perform grid search
		grid_search_svr = GridSearchCV(svr_model, param_grid_svr, cv=5, scoring='neg_mean_squared_error')
		grid_search_svr.fit(x, y)
		
		# Best hyperparameters
		print(f'Best parameters for SVR: {grid_search_svr.best_params_}')
		
		Best parameters for SVR: {'C': 0.1, 'epsilon': 0.2, 'kernel': 'linear'}

Random Forest
 	
Regressor
	n_estimators, max_depth, min_samples_split,
	
	min_samples_leaf
		`from sklearn.ensemble import RandomForestRegressor`
		
		`# Define the model`
		`rf_model = RandomForestRegressor()`
		
		`# Define the hyperparameters grid`
		`param_grid_rf = {`
		`    'n_estimators': [50, 100, 200],`
		`    'max_depth': [None, 10, 20, 30],`
		`    'min_samples_split': [2, 5, 10],`
		`    'min_samples_leaf': [1, 2, 4]`
		`}`
		
		`# Perform grid search`
		`grid_search_rf = GridSearchCV(rf_model, param_grid_rf, cv=5, scoring='neg_mean_squared_error')`
		`grid_search_rf.fit(x, y)`
		
		`# Best hyperparameters`
		`print(f'Best parameters for Random Forest: {grid_search_rf.best_params_}')`
		`Best parameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 100}`

5.2 Performance Metrics Comparison Report

Model	Baseline Metric	Optimized Metric

Decision Tree
 	
Regressor
	`# Train and evaluate Decision Tree Regressor`
	`dt_model.fit(x_train, y_train)`
	`y_pred_dt = dt_model.predict(x_test)`
	`baseline_dt = mean_squared_error(y_test, y_pred_dt, squared=False)`
	
	Baseline RMSE for Decision Tree: 0.020540231956418322
		`# Perform hyperparameter tuning and evaluation`
		`# Decision Tree Regressor`
		`param_grid_dt = {`
		`    'max_depth': [3, 5, 7, 10],`
		`    'min_samples_split': [2, 5, 10],`
		`    'min_samples_leaf': [1, 2, 5]`
		`}`
		`grid_search_dt = GridSearchCV(DecisionTreeRegressor(), param_grid_dt, cv=5, scoring='neg_mean_squared_error')`
		`grid_search_dt.fit(x_train, y_train)`
		`y_pred_dt_opt = grid_search_dt.predict(x_test)`
		`optimized_dt = mean_squared_error(y_test, y_pred_dt_opt, squared=False)`
		
		Optimized RMSE for Decision Tree: 0.017249806930300513

SVR	`# Train and evaluate SVR`
	`svr_model.fit(x_train, y_train)`
	`y_pred_svr = svr_model.predict(x_test)`
	`baseline_svr = mean_squared_error(y_test, y_pred_svr, squared=False)`
	
	Baseline RMSE for SVR: 0.12873558875775146
		`# SVR`
		`param_grid_svr = {`
		`    'C': [0.1, 1, 10, 100],`
		`    'epsilon': [0.01, 0.1, 0.2],`
		`    'kernel': ['linear', 'poly', 'rbf', 'sigmoid']`
		`}`
		`grid_search_svr = GridSearchCV(SVR(), param_grid_svr, cv=5, scoring='neg_mean_squared_error')`
		`grid_search_svr.fit(x_train, y_train)`
		`y_pred_svr_opt = grid_search_svr.predict(x_test)`
		`optimized_svr = mean_squared_error(y_test, y_pred_svr_opt, squared=False)`
		
		Optimized RMSE for SVR: 0.10890416589873322

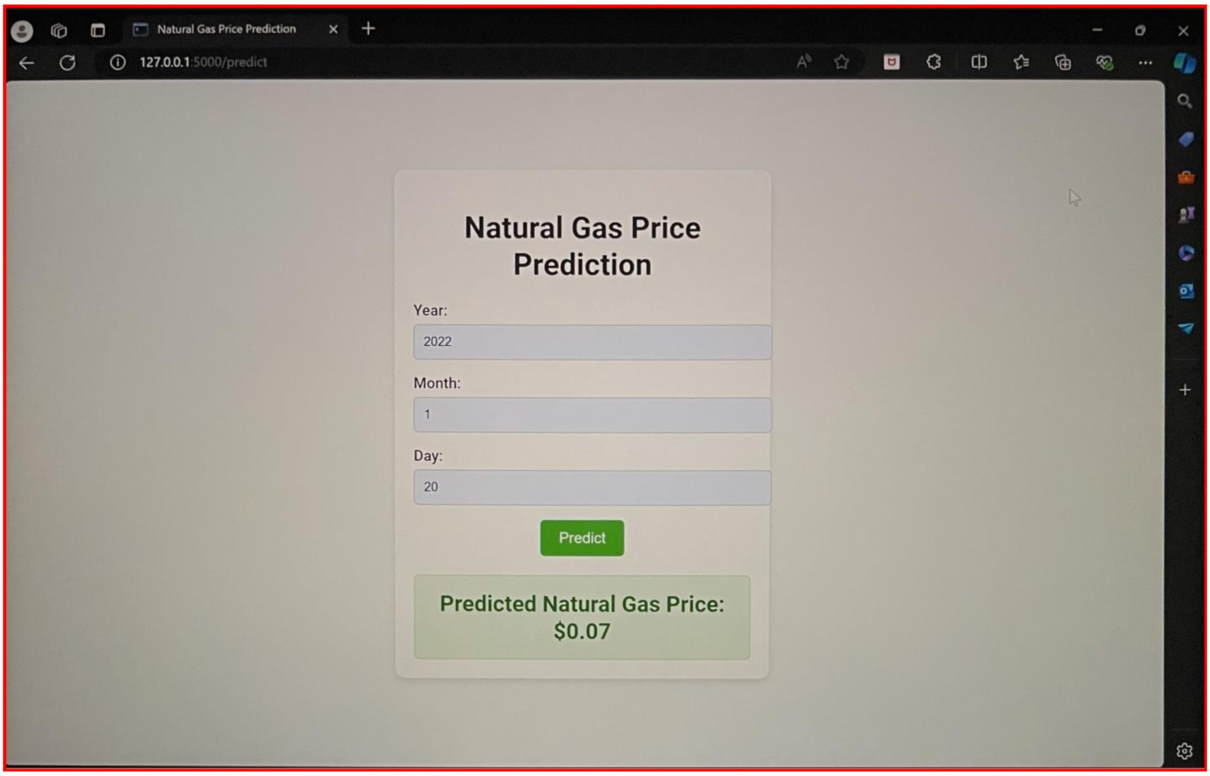
Random Forest
 	
Regressor
	`# Train and evaluate Random Forest Regressor`
	`rf_model.fit(x_train, y_train)`
	`y_pred_rf = rf_model.predict(x_test)`
	`baseline_rf = mean_squared_error(y_test, y_pred_rf, squared=False)`
	
	Baseline RMSE for Random Forest: 0.016116414736557 59
		`# Random Forest Regressor`
		`param_grid_rf = {`
		`    'n_estimators': [50, 100, 200],`
		`    'max_depth': [None, 10, 20, 30],`
		`    'min_samples_split': [2, 5, 10],`
		`    'min_samples_leaf': [1, 2, 4]`
		`}`
		`grid_search_rf = GridSearchCV(RandomForestRegressor(), param_grid_rf, cv=5, scoring='neg_mean_squared_error')`
		`grid_search_rf.fit(x_train, y_train)`
		`y_pred_rf_opt = grid_search_rf.predict(x_test)`
		`optimized_rf = mean_squared_error(y_test, y_pred_rf_opt, squared=False)`
		
		Optimized RMSE for Random Forest: 0.01622791715705109

5.3 Final Model Selection Justification

I chose the Decision Tree Regressor for predicting natural gas prices because it handles non-linear relationships well, is easy to interpret, and requires minimal data preprocessing. Its ability to model complex patterns in data and provide clear visualizations makes it a suitable choice for this regression task.

6.Results

6.1 Output Screenshots



7.Advantages & Disadvantages

Advantages:

1. Accurate Forecasting:
    - Informed Decision-Making: Reliable predictions enable stakeholders to make informed decisions regarding procurement, pricing, and investment strategies.
    - Risk Management: Better predictions help in managing risks associated with price volatility.
2. Efficiency:
    - Automated Process: Once deployed, the model can provide continuous, real-time predictions without manual intervention.
    - Scalability: The solution can be scaled to incorporate additional data sources and features, enhancing its accuracy and robustness.
3. Cost Savings:
    - Optimized Operations: Companies can optimize their operations by timing their buying and selling based on accurate price forecasts.

- Reduced Overhead: Automating the forecasting process reduces the need for extensive manual analysis and labor.

4. Competitive Advantage:
   - Market Insights: Access to advanced predictive analytics can give companies a competitive edge in the market.
   - Strategic Planning: Companies can plan their long-term strategies based on reliable future price trends.
5. Enhanced Understanding:
   - Feature Analysis: By understanding the key factors influencing natural gas prices, companies can better understand market dynamics.

Disadvantages:

1. Data Dependency:
   - Data Quality: The accuracy of the model is heavily dependent on the quality and completeness of historical price data.
   - Limited Data: In the absence of sufficient historical data, the model's predictions may be less reliable.
2. Complexity:
   - Model Complexity: Advanced models like LSTM or GBM can be complex to implement and require expertise in machine learning and data science.
   - Resource Intensive: High computational resources may be required for training complex models, especially with large datasets.
3. Maintenance:
   - Regular Updates: The model needs to be regularly updated with new data to maintain its accuracy over time.
   - Monitoring: Continuous monitoring is necessary to detect and address any performance degradation.
4. Uncertainty in Predictions:
   - External Factors: Unpredictable external factors (e.g., geopolitical events, natural disasters) can affect natural gas prices, which the model may not fully account for.
   - Model Limitations: All models have inherent limitations and may not capture every nuance of the market.

5. Implementation Costs:
    - Initial Investment: The initial development and implementation of the model can be costly.
    - Technical Expertise: Requires skilled personnel for model development, implementation, and maintenance.

8. Conclusion

The Natural Gas Price Prediction project successfully demonstrates the potential of machine learning in forecasting commodity prices. By providing accurate and timely predictions, the project empowers stakeholders to make data-driven decisions, optimize operations, and strategically navigate the complexities of the natural gas market. While the project addresses several challenges, continuous efforts in data quality management, model maintenance, and scalability will ensure its long-term success and relevance. The insights gained from this project lay the foundation for further advancements in predictive analytics within the energy sector, fostering innovation and improved market efficiencies.

9.Future Scope

The future scope of this project includes integrating additional data sources such as weather patterns, geopolitical events, and economic indicators to improve model accuracy. Expanding the model to predict prices for other energy commodities like oil and electricity can enhance its utility. Implementing real-time data feeds and deploying the model in a cloud-based environment will ensure scalability and accessibility. Advanced techniques like ensemble learning and deep learning can be explored for further accuracy improvements. Additionally, developing user-friendly applications for stakeholders to interact with the model's predictions can broaden its impact and adoption in the industry.

10.Appendix

10.1 Source Code

Model Training and Testing

```python
# Split data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)
```

```python
print(x.shape,x_train.shape,x_test.shape)
```

```
(5946, 6) (4756, 6) (1190, 6)
```

```python
# Train a Decision Tree Regressor
model_dt = DecisionTreeRegressor(max_depth=5, min_samples_split=10, random_state=2)
model_dt.fit(x_train, y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, min_samples_split=10, random_state=2)
```

```python
#predictions
y_pred_dt = model_dt.predict(x_test)
```

```python
# Evaluating the model
mse_dt = mean_squared_error(y_test, y_pred_dt)
rmse_dt = np.sqrt(mse_dt)
```

```python
rmse_dt
```

```
0.017540079972366295
```

```python
from sklearn import metrics
print(metrics.r2_score(y_test, y_pred_dt))
```

```
0.9815322881744076
```

Flask

```python
from flask import Flask, request, render_template
import pandas as pd
Click to collapse the range.
import pickle
app = Flask(__name__)
with open('model_dt.pkl', 'rb') as f:
    model = pickle.load(f)
data = pd.read_csv('preprocessed_natural_gas_prices.csv')
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/predict', methods=['POST'])
def predict():
    year = int(request.form['year'])
    month = int(request.form['month'])
    day = int(request.form['day'])

    date_index = pd.Timestamp(year=year, month=month, day=day)

    data['Date'] = pd.to_datetime(data[['Year', 'Month', 'Day']])
    past_data = data[data['Date'] <= date_index].sort_values(by='Date').tail(7)

    price_lag1 = past_data.iloc[-2]['Price'] if len(past_data) > 1 else np.nan
    price_lag7 = past_data.iloc[0]['Price'] if len(past_data) == 7 else np.nan
    price_rolling_mean7 = past_data['Price'].mean()

    features = [price_lag1, price_lag7, price_rolling_mean7]
    final_features = [np.array(features)]

    prediction = model.predict(final_features)

    return render_template('index.html', prediction_text=f'Predicted Natural Gas Price: ${prediction[0]

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

HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Natural Gas Price Prediction</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap" rel="stylesheet">
</head>
<body>
    <div class="container">
        <div class="card">
            <h1>Natural Gas Price Prediction</h1>
            <form action="/predict" method="post">
                <div class="input-group">
                    <label for="year">Year:</label>
                    <input type="text" name="year" required>
                </div>
                <div class="input-group">
                    <label for="month">Month:</label>
                    <input type="text" name="month" required>
                </div>
                <div class="input-group">
                    <label for="day">Day:</label>
```

```html
                <div class="input-group">
                    <label for="day">Day:</label>
                    <input type="text" name="day" required>
                </div>
                <button type="submit">Predict</button>
            </form>
            {% if prediction_text %}
                <div class="result">
                    <h2>{{ prediction_text }}</h2>
                </div>
            {% endif %}
        </div>
    </div>
</body>
</html>
```

10.2. GitHub & Project Demo Link

Project Demo Link:

https://drive.google.com/file/d/1OXtiroboSRIkLCxQ_8GRmO2U29TzeU-p/view?usp=drivesdk

GitHub Link:

https://github.com/vyshnavikonakalla/machinelearning-approach-for-predicting-the-price-of-natural-gas