

KGISL INSTITUTE OF TECHNOLOGY

**NAAN MUDHALVAN**

**DIVISION:** APPLIED DATA SCIENCE

**PROJECT TITLE: ELECTRICITY PRICE PREDICTION**

**PHASE 4**

**PROBLEM STATEMENT:**

The problem is to develop a predictive model that uses historical electricity prices and relevant factors to forecast future electricity prices. The objective is to create a tool that assists both energy providers and consumers in making informed decisions regarding consumption and investment by predicting future electricity prices. This project involves data preprocessing, feature engineering, model selection, training, and evaluation.

**TEAM MEMBERS:**

Shreevarsha G

Shereena A

Nishanthi Akshayaa S

Guhapriyan K

**TEAM MENTOR:**

Indu Poornima R

**PROBLEM STATEMENT:**

The energy industry faces significant challenges in efficiently managing the supply and demand of electricity, particularly in a rapidly changing landscape marked by fluctuating energy prices and evolving consumer preferences. Energy providers and consumers both need a reliable tool to predict future electricity prices and make informed decisions about consumption and investments. To address this critical issue, we aim to develop a predictive model for electricity price forecasting that incorporates customer behavior analysis, serving as a powerful solution for stakeholders in the energy sector.

**PROPOSED SOLUTION:**

**Our innovative solution involves the following steps:**

**1. Project Planning:**

Define Objectives: Clearly articulate the goals of the project. For electricity price forecasting, specify the time frame (e.g., hourly, daily) and the geographic region of interest.

Stakeholder Requirements: Understand the specific needs and expectations of energy providers and consumers. Determine what kind of insights or predictions will be most valuable to them.

**2. Data Collection:**

Historical Electricity Price Data: Collect historical electricity price data from reliable sources. These sources might include energy market databases, government agencies, or utilities. Tools such as Python libraries (Pandas, NumPy) can help with data collection.

Associated Factors Data: Gather data on factors that influence electricity prices, such as weather data, demand, supply, and economic indicators. APIs or web scraping tools can be used to acquire this data.

**3. Data Preprocessing:**

Data Cleaning: Clean the data by removing missing values, outliers, and inconsistencies using Python libraries like Pandas and NumPy.

Data Transformation: Convert the data into a suitable format. For time series data, you might need to decompose it into trend, seasonality, and residuals.

Data Normalization or Scaling: Ensure all input features are on the same scale, which can help with model training.

**4. Feature Engineering:**

Create Relevant Features: Extract or engineer features that could provide meaningful insights into electricity price prediction. For instance, create lag features to capture historical trends.

Feature Selection: Use techniques like correlation analysis or feature importance from machine learning models to select the most relevant features. This step aims to reduce dimensionality and improve model performance.

**5. Model Selection:**

Choose Algorithms: Select the appropriate machine learning algorithms for time series forecasting. Common choices include ARIMA, LSTM (for deep learning), XGBoost, Random Forest, or regression models.

Hyperparameter Tuning: Optimize the model's hyperparameters using tools like scikit-learn's GridSearchCV or RandomizedSearchCV.

**6. Model Training:**

Split Data: Divide the dataset into training, validation, and test sets. Common splits might include 70% for training, 15% for validation, and 15% for testing.

Train Models: Train selected models on the training data using Python libraries like scikit-learn, Keras, or TensorFlow.

**7. Model Evaluation:**

Performance Metrics: Evaluate the models using appropriate metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).

Cross-Validation: Implement cross-validation techniques (e.g., k-fold cross-validation) to assess the model's robustness and generalization performance.

Visualizations: Create visualizations to help stakeholders understand model performance and predictions. This can include time series plots and error distribution plots.

**8. Model Deployment:**

Deploy the model in a production environment. This may involve using cloud services (e.g., AWS, Azure, or Google Cloud), containerization (e.g., Docker), and creating APIs for real-time predictions.

Set up automated pipelines for regular model updates and retraining, as the model's accuracy may degrade over time.

**9. Documentation and Reporting:**

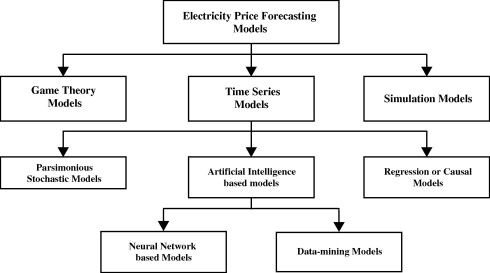
Document the entire process, including data sources, preprocessing steps, feature engineering, model selection, and evaluation metrics. This documentation is crucial for reproducibility and collaboration.

Prepare clear and concise reports for stakeholders, explaining the model's capabilities and limitations.

**10. Maintenance and Monitoring:**

Continuously monitor the model's performance. Implement alerting systems to notify stakeholders of issues or significant changes in predictions.

Update the model as new data becomes available or as its accuracy degrades over time. Regular maintenance is essential to keep the model relevant.



**Dataset Description:**

Source: Specify the source of your dataset, whether it's a publicly available dataset, a company's internal data, or any other source.

Data Format: Describe the format of the data. Common formats include CSV, JSON, Excel, or database formats.

Features: List the features (variables) present in the dataset. Include information about both input features and target variables for prediction tasks.

Data Size: Mention the number of samples or records in the dataset.

Data Description: Provide a brief description of the data, including the data types, possible value ranges, and any unique characteristics.

**Data Preprocessing Steps:**

Data Cleaning: Handle missing values, outliers, or any inconsistencies in the data. This could involve imputation, removal, or advanced techniques like interpolation.

Feature Selection/Engineering: Choose relevant features for the model. This might include transforming variables, creating new features, or selecting a subset of features based on domain knowledge or feature importance analysis.

Normalization/Standardization: Scale the features to a similar range to prevent any particular feature from dominating the learning process.

Encoding: Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding.

Train-Test Split: Divide the dataset into training and testing sets for model evaluation. A common split ratio is 80% for training and 20% for testing.

Validation Set (Optional): Create a validation set from the training data for hyperparameter tuning and model selection.

Data Augmentation (For Image Data): Generate new training samples by applying random transformations like rotation, scaling, and flipping to increase the diversity of the dataset.

**Model Training Process:**

Model Selection: Choose an appropriate machine learning algorithm or deep learning architecture based on the nature of the problem (classification, regression, etc.) and the dataset.

Initialization: Initialize the model parameters. For deep learning, this might involve using pre-trained weights from models trained on large datasets (transfer learning).

Loss Function: Define a loss function appropriate for the task (e.g., mean squared error for regression, cross-entropy for classification).

Optimizer: Choose an optimization algorithm (e.g., SGD, Adam) to minimize the loss function during training.

Training: Feed the training data into the model and adjust the model parameters iteratively to minimize the loss on the training data. Monitor the model's performance on the validation set to prevent overfitting.

Hyperparameter Tuning: Fine-tune hyperparameters like learning rate, batch size, and the number of layers/neurons through experimentation and validation set performance.

Evaluation: Evaluate the trained model on the test dataset to assess its performance. Common evaluation metrics include accuracy, mean squared error, precision, recall, and F1-score, depending on the problem type.

Deployment (Optional): If the model performs satisfactorily, deploy it for predictions on new, unseen data in a real-world setting.

**Time Series Forecasting Algorithms:**

Autoregressive Integrated Moving Average (ARIMA):

Suitable for: Univariate time series data with a linear trend and stationary patterns.

Advantages: Simple and interpretable, captures linear dependencies in the data.

Considerations: May not perform well for complex or nonlinear relationships.

Seasonal-Trend decomposition using LOESS (STL):

Suitable for: Time series data with strong seasonal patterns and trends.

Advantages: Handles nonlinearity and seasonality well.

Considerations: Can be computationally intensive for large datasets.

Prophet:

Suitable for: Time series data with daily observations and missing data points.

Advantages: Robust to missing data, outliers, and shifts in trend.

Considerations: Requires careful tuning of parameters for optimal performance.

Long Short-Term Memory (LSTM) Networks:

Suitable for: Complex, nonlinear time series data with long-term dependencies.

Advantages: Can capture intricate patterns, suitable for long-range predictions.

Considerations: Requires a larger amount of data for training and careful tuning of hyperparameters.

Gated Recurrent Unit (GRU) Networks:

Similar to LSTM but computationally more efficient: Suitable for cases where computational resources are limited.

**Evaluation Metrics for Time Series Forecasting:**

Mean Absolute Error (MAE):

Definition: Average of the absolute differences between predicted and actual values.

Interpretation: Measures the average magnitude of errors in the forecasts.

Use case: Useful when errors need to be reported in the same units as the data.

Mean Squared Error (MSE):

Definition: Average of the squared differences between predicted and actual values.

Interpretation: Punishes larger errors more severely than MAE.

Use case: Emphasizes larger errors and is sensitive to outliers.

Root Mean Squared Error (RMSE):

Definition: Square root of MSE.

Interpretation: Provides an interpretable scale (same as the original data) and penalizes large errors.

Use case: Commonly used when a comprehensible error metric is required.

Mean Absolute Percentage Error (MAPE):

Definition: Average of the percentage differences between predicted and actual values.

Interpretation: Represents the percentage error relative to the actual values.

Use case: Useful when errors need to be expressed as a percentage of the actual values.

Symmetric Mean Absolute Percentage Error (SMAPE):

Definition: A symmetric version of MAPE that handles zero values in the data.

Interpretation: Provides a symmetric, percentage-based error metric.

Use case: Particularly useful when dealing with intermittent demand or sporadic data.

R-squared (R²) Score:

Definition: Measures the proportion of the variance in the dependent variable that is predictable from the independent variables.

Interpretation: Indicates the goodness of fit of the model.

Use case: Provides an overall assessment of the model's explanatory power.

Factors Influencing Choice:

Data Characteristics: Consider the presence of trends, seasonality, and other patterns in the data. Choose models that can capture these characteristics effectively.

Model Complexity: Choose models that balance complexity with interpretability. Simpler models like ARIMA might be preferred for straightforward forecasting tasks.

Computational Resources: Deep learning models like LSTM and GRU are powerful but require substantial computational resources and larger datasets for training.

Business Requirements: Consider the impact of forecast accuracy on business decisions. Some metrics, like MAPE, offer a clear understanding of prediction errors in percentage terms.

Data Availability: Models like Prophet are designed to handle missing data and outliers, making them suitable for real-world datasets with data gaps.

Forecast Horizon: Some models perform better for short-term forecasts, while others, like LSTM, can handle longer forecast horizons effectively.

**DEVELOPMENT PART – 1**

Importing Required Libraries

The code begins by importing several essential libraries:

PANDAS (as pd):

This library is used for data manipulation and analysis. It provides data structures and functions for working with structured data.

NUMPY (as np):

NumPy is used for numerical operations and arrays. It is often used for mathematical and numerical computations.

MATPLOTLIB.PYPLOT (as plt):

Matplotlib is a data visualization library, and pyplot is a sub-library that provides a convenient interface for creating various types of plots and charts.

SEABORN (as sns):

Seaborn is another data visualization library that enhances the aesthetics and visual appeal of data visualizations.

LOADING THE DATASETS:

The code loads a dataset from a CSV file named "MSFT.csv" into a Pandas DataFrame, which is essentially a structured table of data. This DataFrame is named df. The dataset likely contains historical Microsoft stock data, and it's important for the subsequent data analysis.

**UNDERSTANDING THE DATASETS:**

To better understand the data, the code performs the following operations:

**df.describe() (Electricity) :**

This function provides summary statistics for numerical columns in the DataFrame. It gives information such as the mean, standard deviation, minimum, maximum, and quartiles for each numeric attribute.

**df.info() (Electricity):**

This function provides information about the DataFrame, including the data types of each column (e.g., integer, float, string).

**df.isnull().sum() (Electricity):**

This code counts the number of missing values (NaN) in each column of the DataFrame. Identifying missing data is crucial for data cleaning and imputation.

**EXCESS:**

* Electricity.shape()
* Electricity.mean()
* Electricity.median()
* Electricity.mode()
* Electricity.std()
* Electricity.var()
* Electricity.skew()
* Electricity.describe(include=’all’)
* Electricity.kurt()
* Sum(Electricity.duplicated())

**VISUALIZING THE DATASET:**

The code proceeds to visualize the data to gain insights into its distribution and relationships between variables.

**HISTOGRAMS FOR NUMERICAL COLUMNS:**

The code creates histograms for a set of specified numerical columns. A histogram is a graphical representation of the distribution of data. It helps visualize how values are spread across the range of each attribute. Each histogram is displayed with 20 bins, and it uses a blue color with black edges for aesthetics. Titles, x-axis labels, and y-axis labels are set to provide context for each histogram.

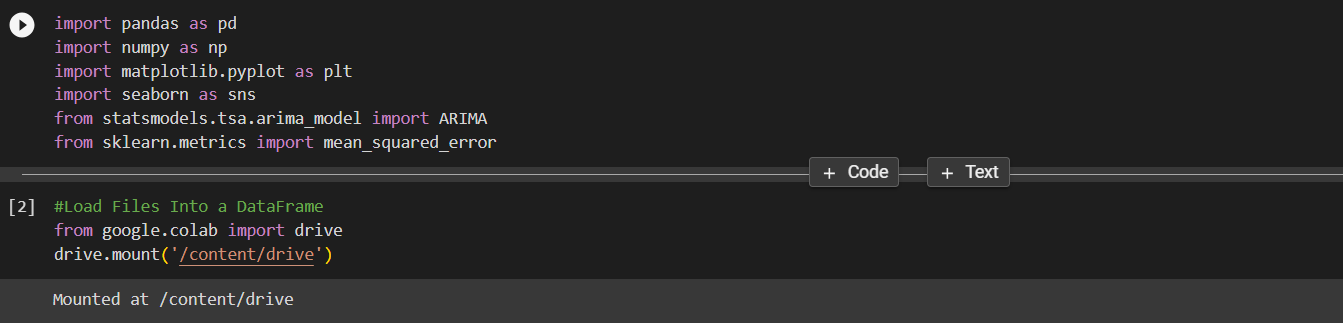
**Histogram of the 'Close' Column (Target):**

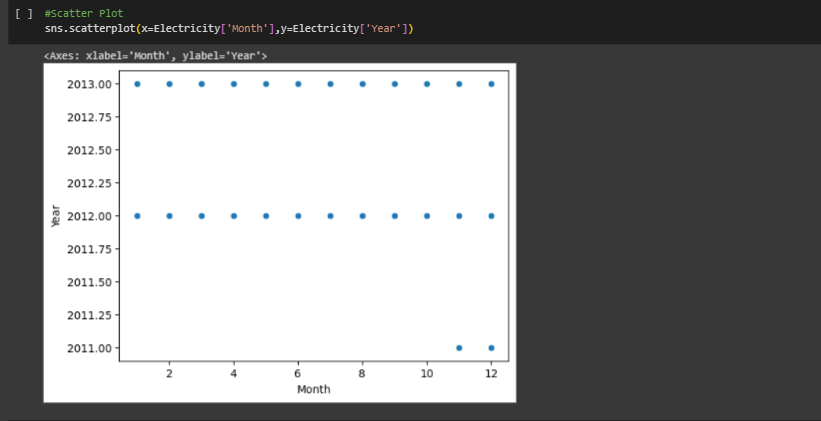
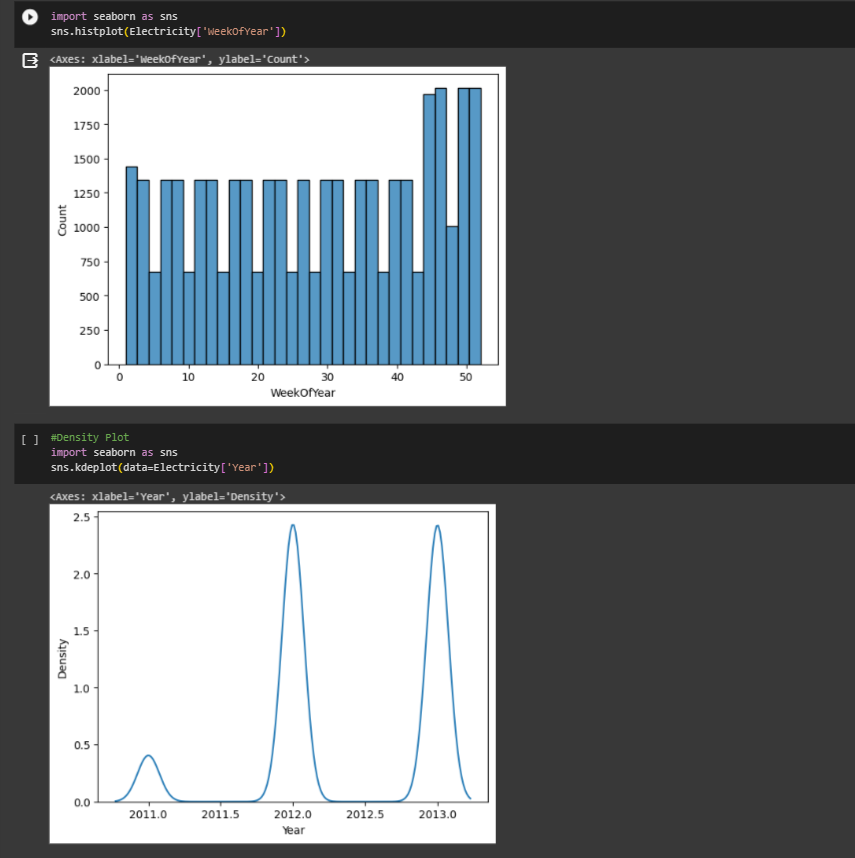
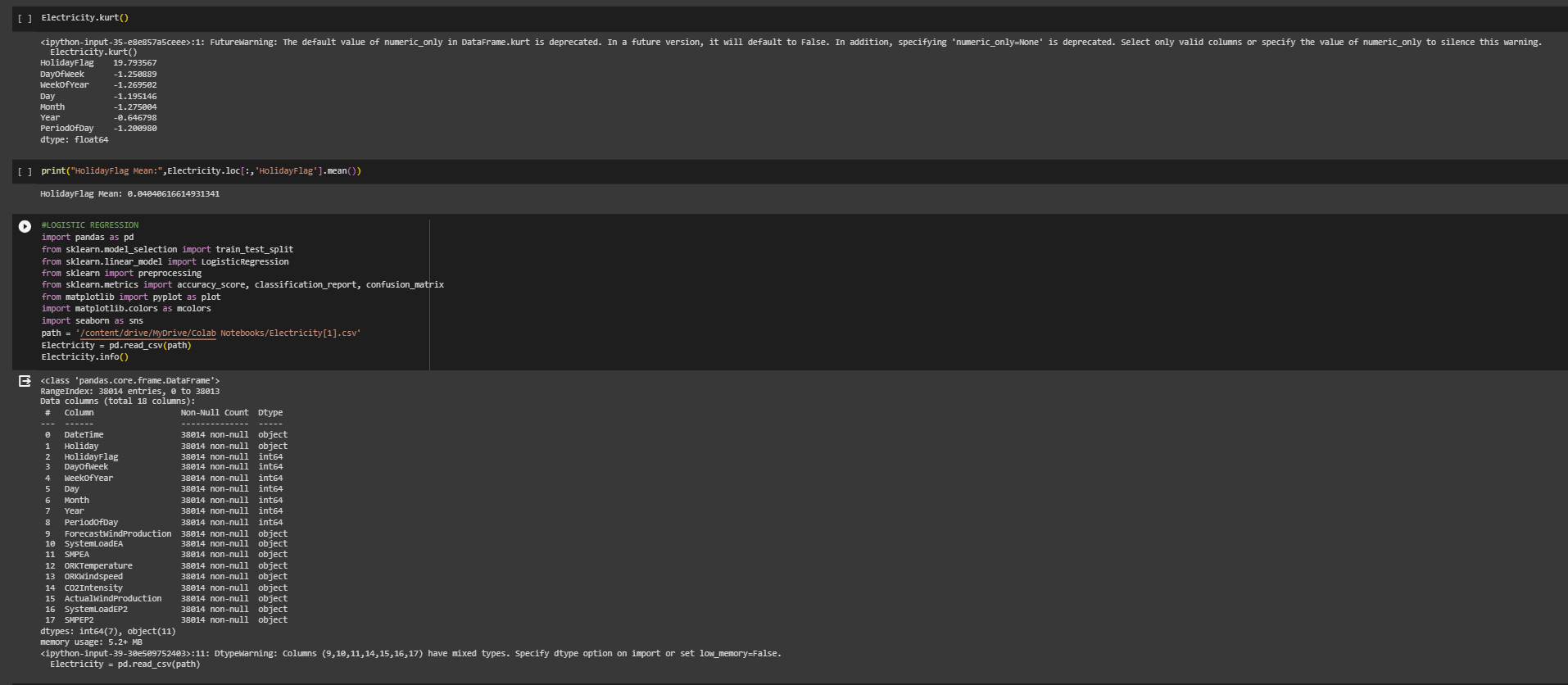
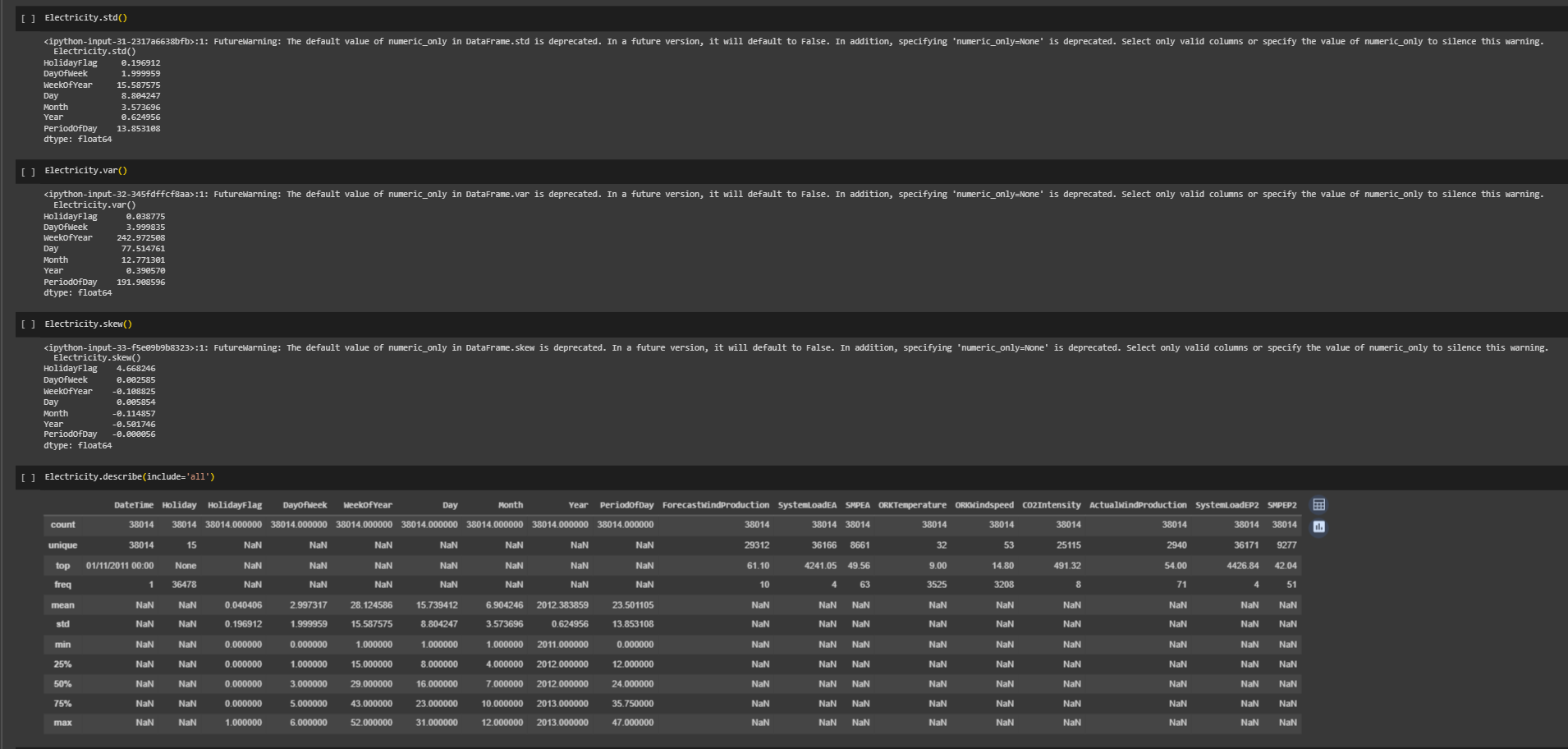
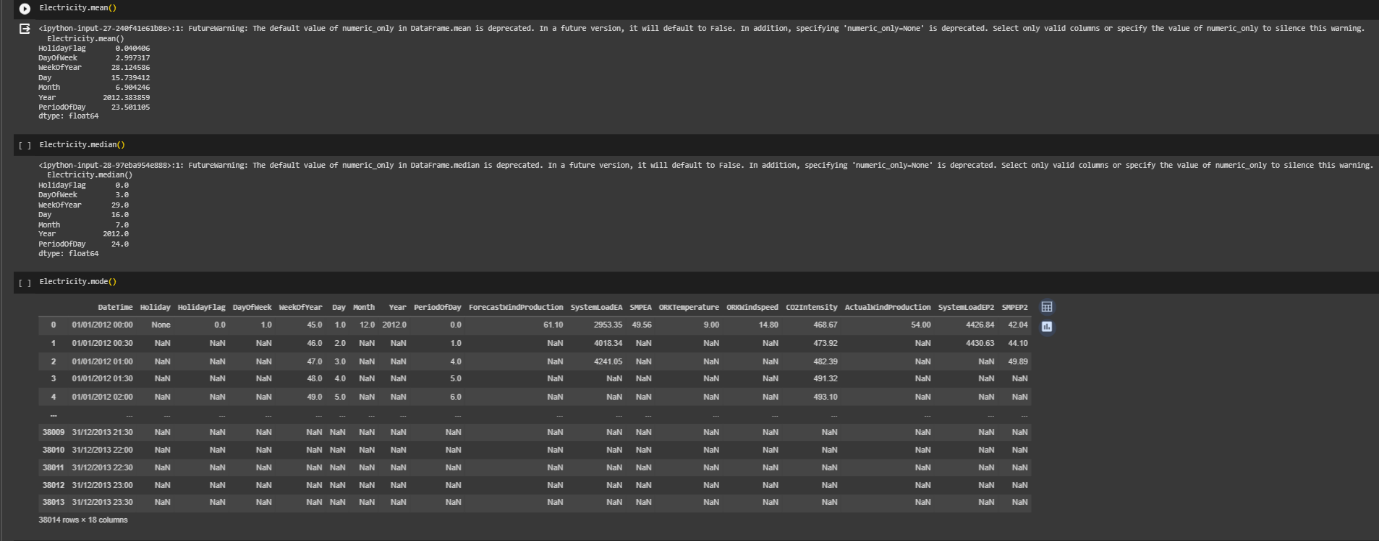
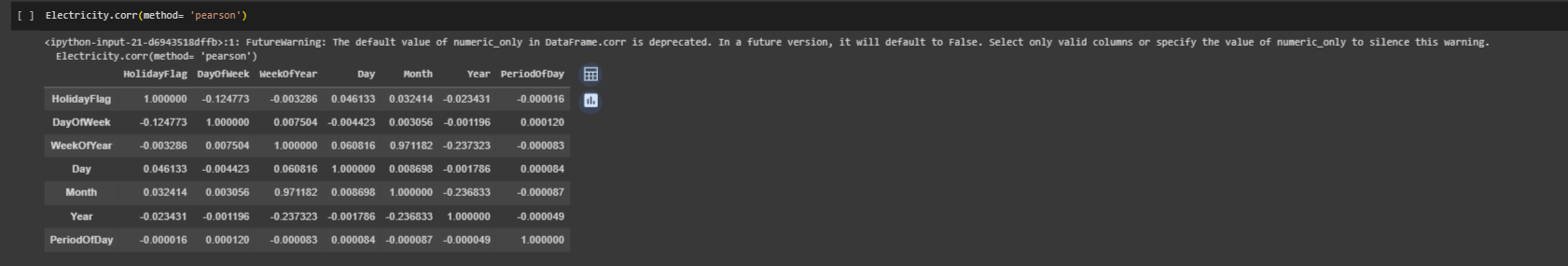
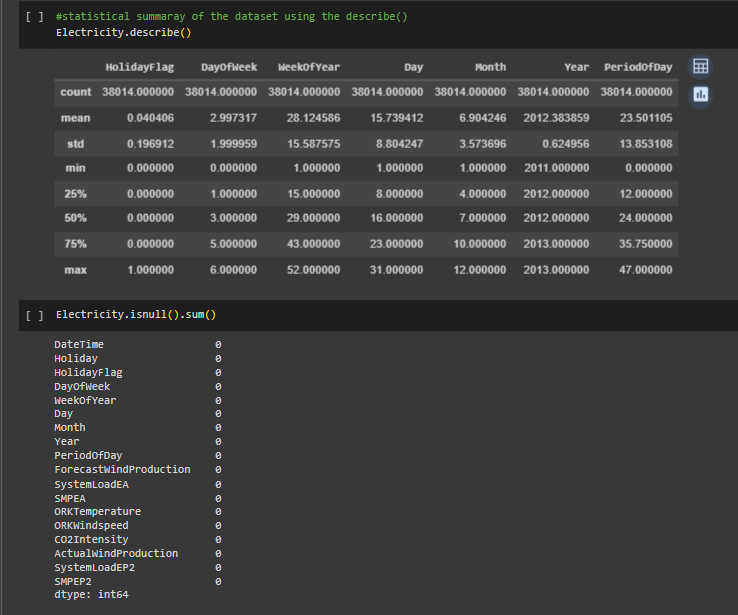
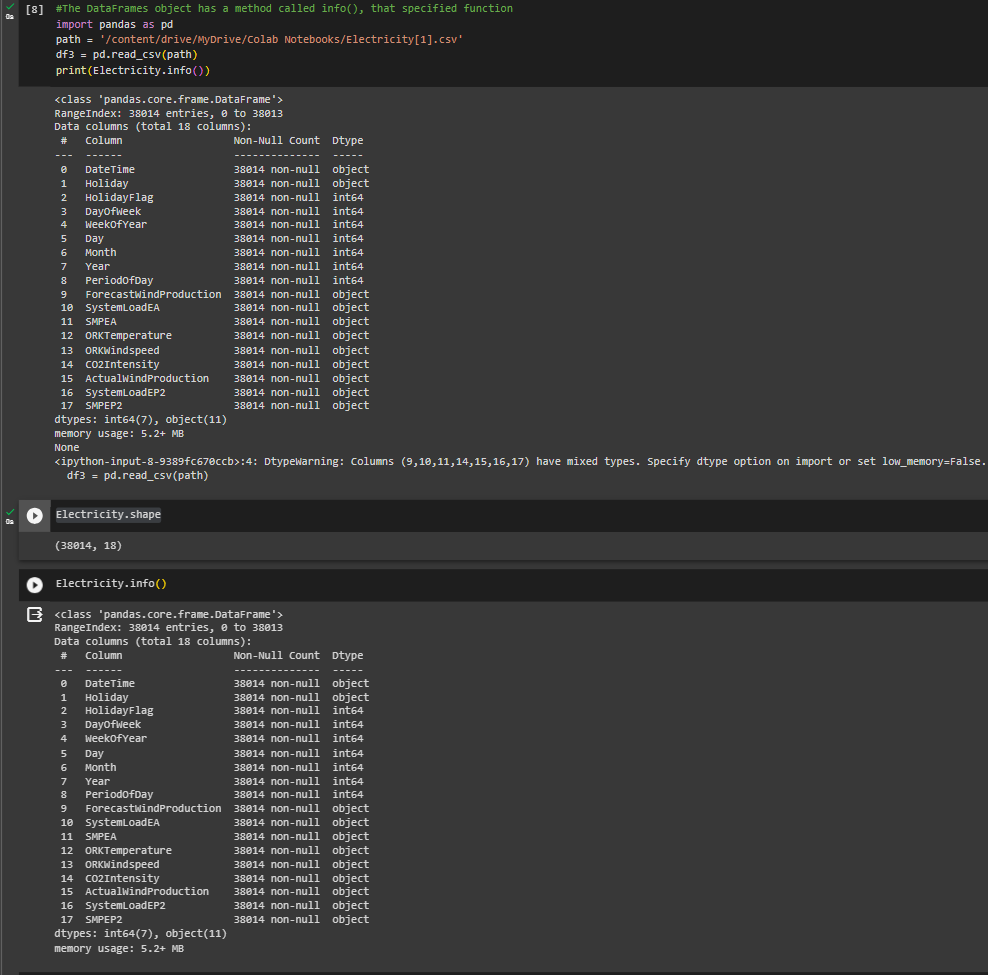
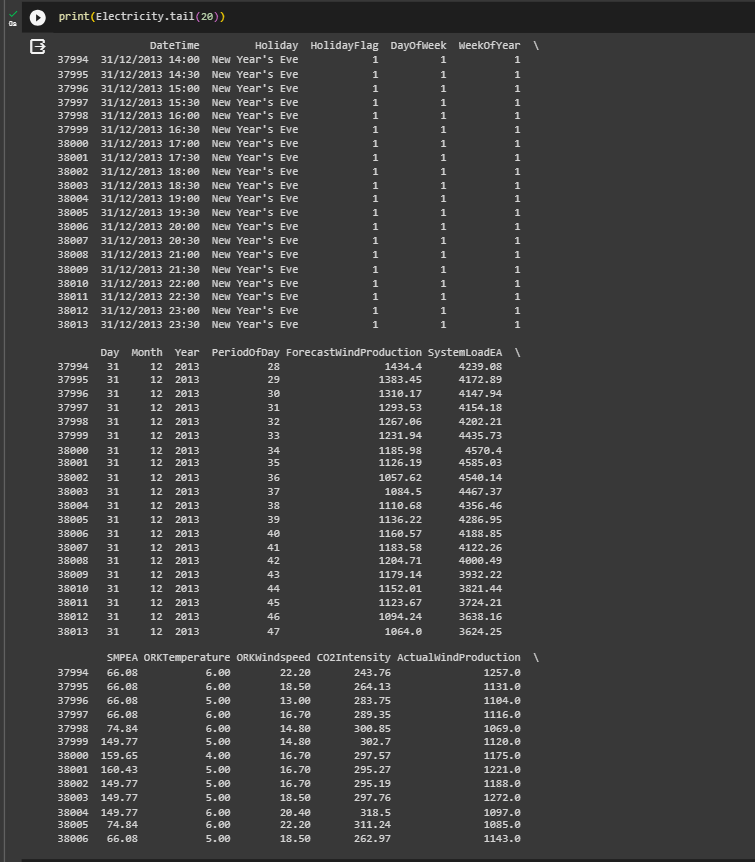
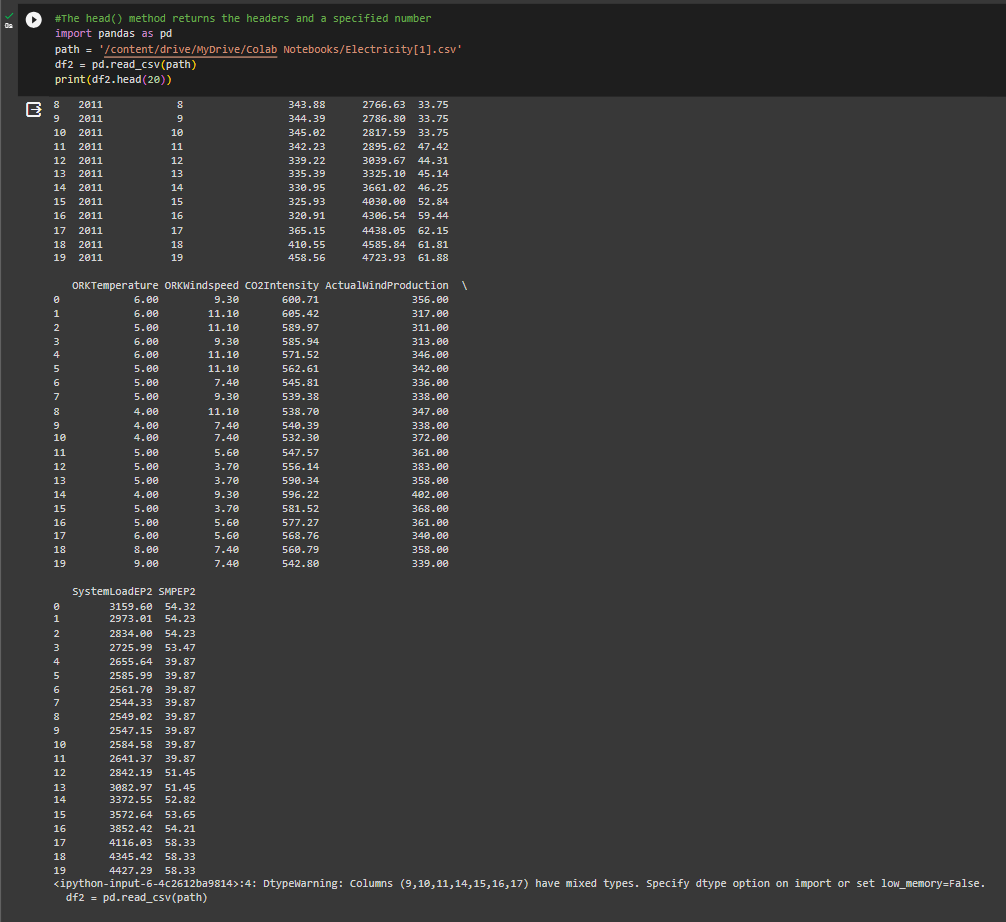
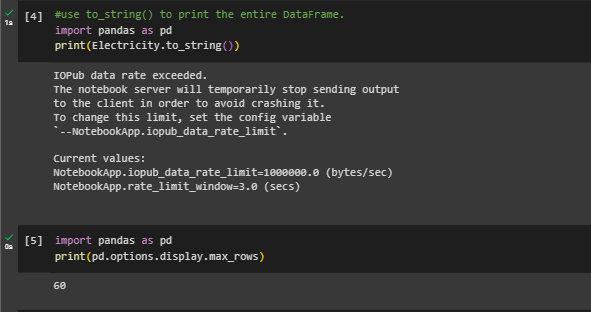
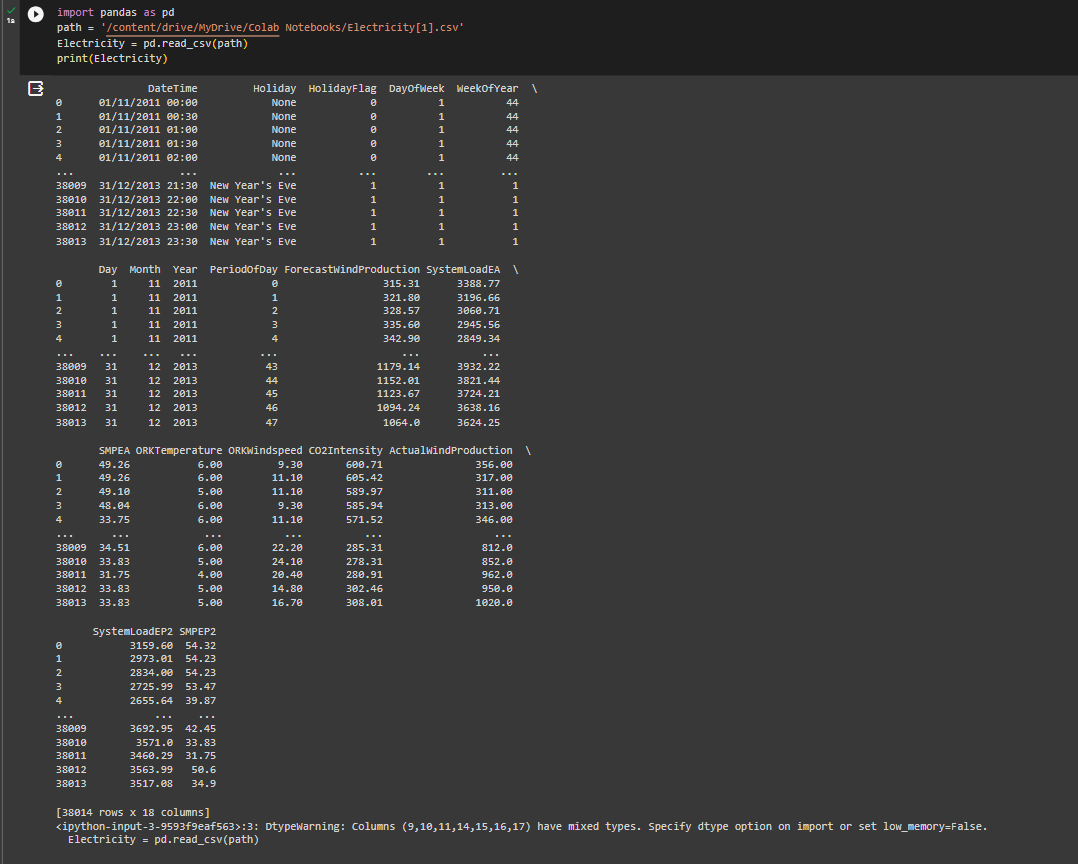
A separate histogram is created specifically for the 'Close' column, which is likely the target variable of interest. This histogram visualizes how the closing prices are distributed.

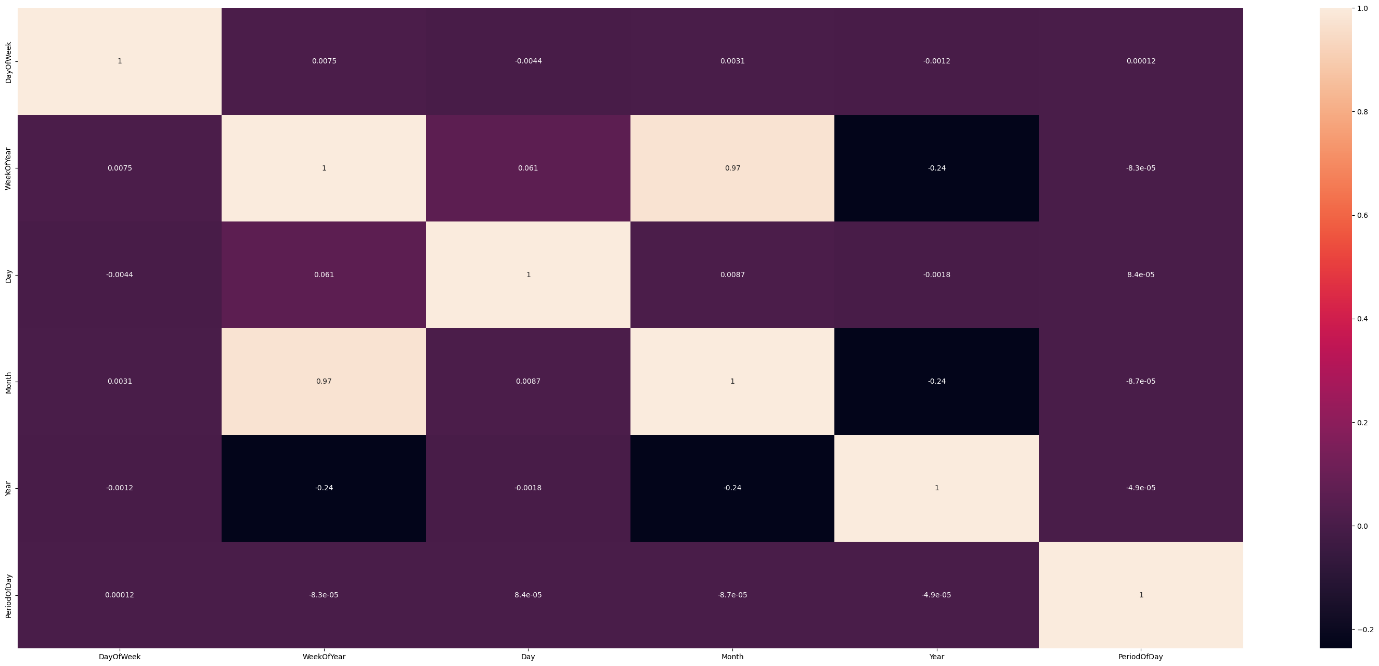
**Pairplot:**

A pairplot is generated using Seaborn. This visualization creates scatterplots for combinations of numerical columns in the dataset. It helps identify potential relationships and correlations between different attributes. Pairplots are a useful tool for exploring multivariate data.

In conclusion, this code is a basic but essential data analysis pipeline for exploring a dataset containing Microsoft stock data. It includes data loading, summary statistics, and visualizations to gain initial insights and prepare the data for more in-depth analysis and modeling.

**MODEL: **

****

****

**COLAB LINK:**

**https://colab.research.google.com/drive/1931heJS5wtqcO8HQeeEdSRmoOFMss9Ze?usp=share\_link**