Big data system term project

411021390 賈俊佑

1.Theme introduction:

I use the stock data from the yfinance to build a linear regression model by apache spark to predict the stock price, and this program has the UI to let the user input some information like the stock symbol to customized the linear regression and there have shown evaluate number that can let the user evaluate the regression model they made, and the last is the UI can let user input today transaction volume and the close price to predict the next day close price.

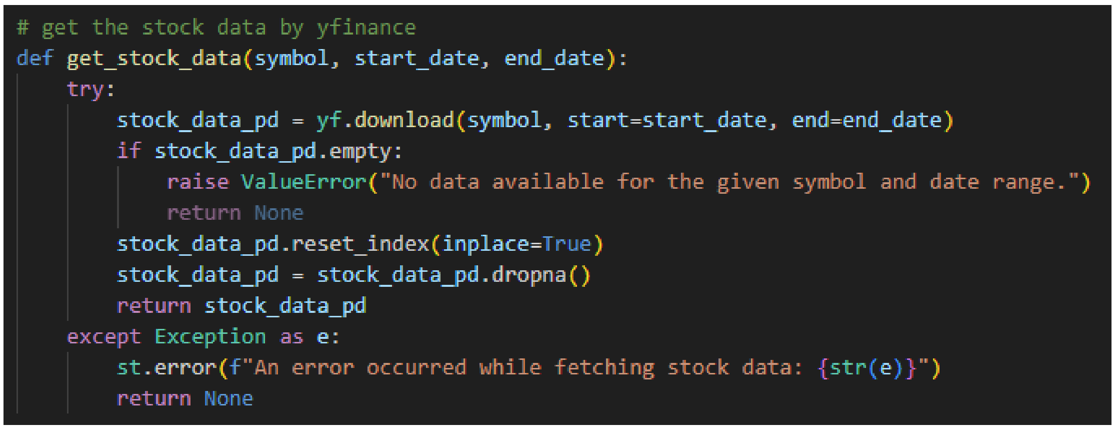
2.Data introduction

The all stock data is from the yfinance .The yfinance is the python library that give user a API to get the history stock data ,and the dataset size is base on the user input, the input which is start day and end day would decide the dataset size and this dataset has 5 columns ,they are

Open、High、Low、Close、Adj Close、Volume.

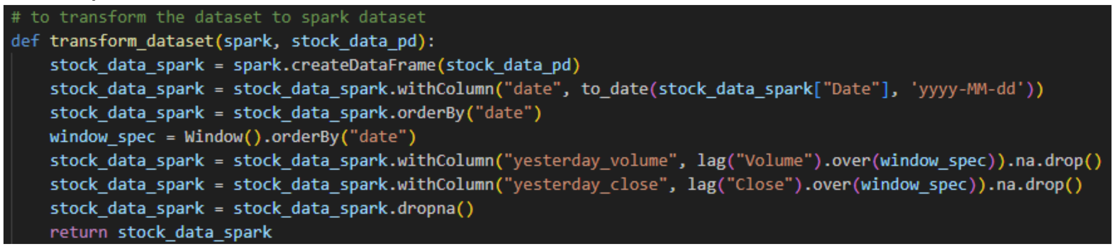
3.Data preprocess

The first step is getting the stock from yfinance and in order to check the data information we get is correct, I would first check the dataset is not empty, and then because initial the index is the date and I want to let it be the column, so I change the index, the last step is dropping the null data.



```python
# get the stock data by yfinance
def get_stock_data(symbol, start_date, end_date):
    try:
        stock_data_pd = yf.download(symbol, start=start_date, end=end_date)
        if stock_data_pd.empty:
            raise ValueError("No data available for the given symbol and date range.")
            return None
        stock_data_pd.reset_index(inplace=True)
        stock_data_pd = stock_data_pd.dropna()
        return stock_data_pd
    except Exception as e:
        st.error(f"An error occurred while fetching stock data: {str(e)}")
        return None
```

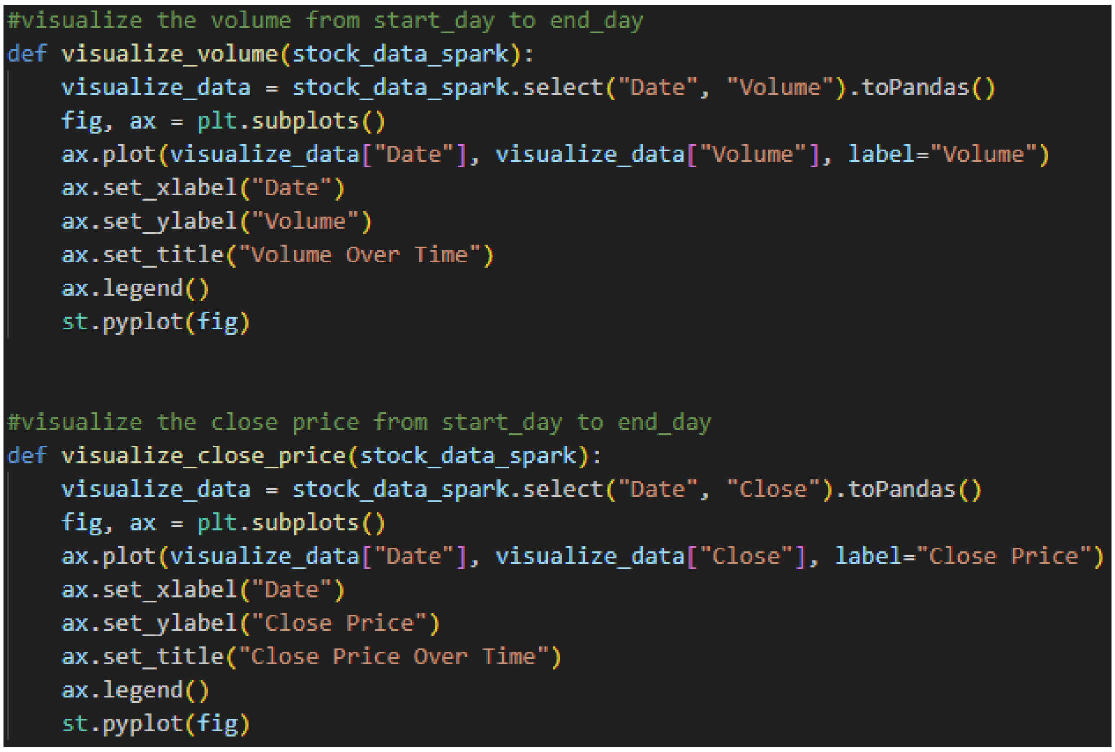
And the next step is transforming the dataset to spark data frame, because initial the dataset is original python data frame and I addition two column they are the yesterday transaction volume and the yesterday close price.



```
# to transform the dataset to spark dataset
def transform_dataset(spark, stock_data_pd):
    stock_data_spark = spark.createDataFrame(stock_data_pd)
    stock_data_spark = stock_data_spark.withColumn("date", to_date(stock_data_spark["Date"], 'yyyy-MM-dd'))
    stock_data_spark = stock_data_spark.orderBy("date")
    window_spec = Window().orderBy("date")
    stock_data_spark = stock_data_spark.withColumn("yesterday_volume", lag("Volume").over(window_spec)).na.drop()
    stock_data_spark = stock_data_spark.withColumn("yesterday_close", lag("Close").over(window_spec)).na.drop()
    stock_data_spark = stock_data_spark.dropna()
    return stock_data_spark
```

4.Data visualization

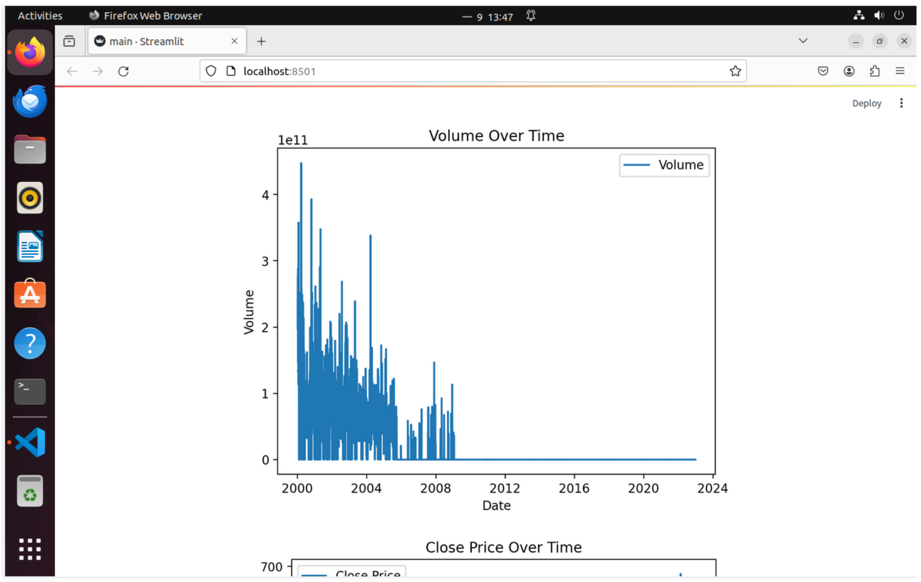
This step visualization the data of close price and volume in the dataset over time.

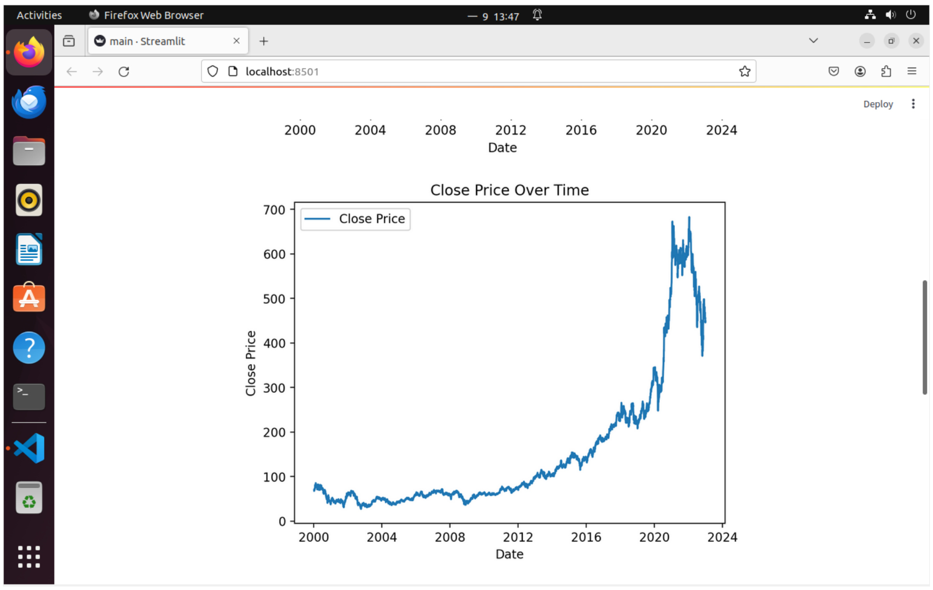


```
#visualize the volume from start_day to end_day
def visualize_volume(stock_data_spark):
    visualize_data = stock_data_spark.select("Date", "Volume").toPandas()
    fig, ax = plt.subplots()
    ax.plot(visualize_data["Date"], visualize_data["Volume"], label="Volume")
    ax.set_xlabel("Date")
    ax.set_ylabel("Volume")
    ax.set_title("Volume Over Time")
    ax.legend()
    st.pyplot(fig)


#visualize the close price from start_day to end_day
def visualize_close_price(stock_data_spark):
    visualize_data = stock_data_spark.select("Date", "Close").toPandas()
    fig, ax = plt.subplots()
    ax.plot(visualize_data["Date"], visualize_data["Close"], label="Close Price")
    ax.set_xlabel("Date")
    ax.set_ylabel("Close Price")
    ax.set_title("Close Price Over Time")
    ax.legend()
    st.pyplot(fig)
```

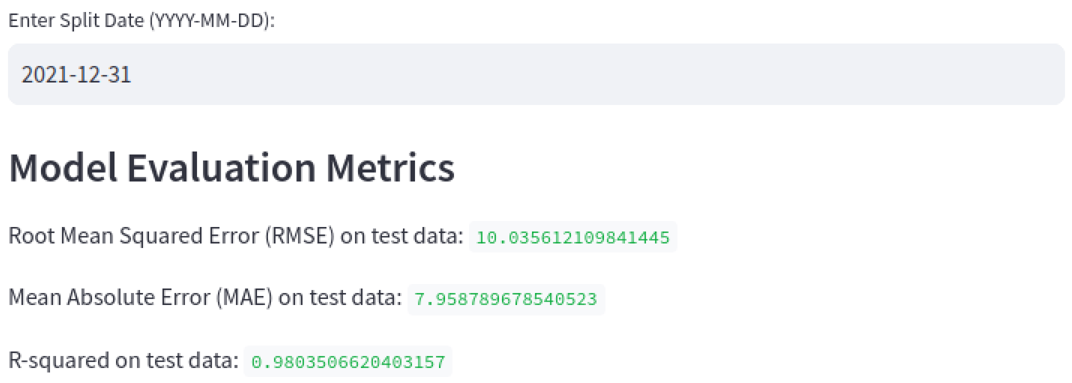
This the example shows the output of the two function.





5.Stock prediction

In the first I would let the user input the spilt day, the function would base on split day to split the dataset into training dataset and test dataset.


Enter Split Date (YYYY-MM-DD):

2021-12-31

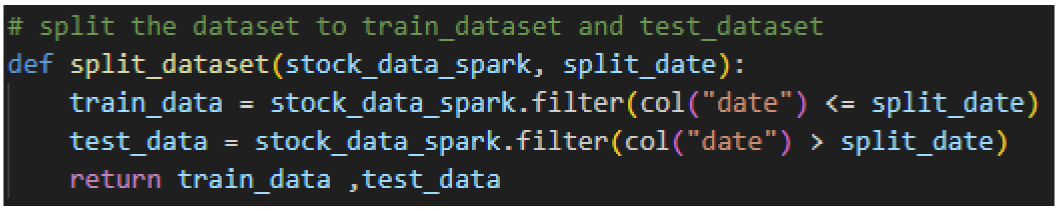
Model Evaluation Metrics

Root Mean Squared Error (RMSE) on test data: `10.035612109841445`

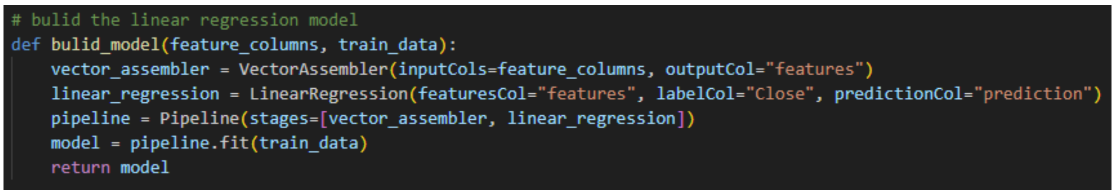
Mean Absolute Error (MAE) on test data: `7.958789678540523`

R-squared on test data: `0.9803506620403157`

This is code implementation

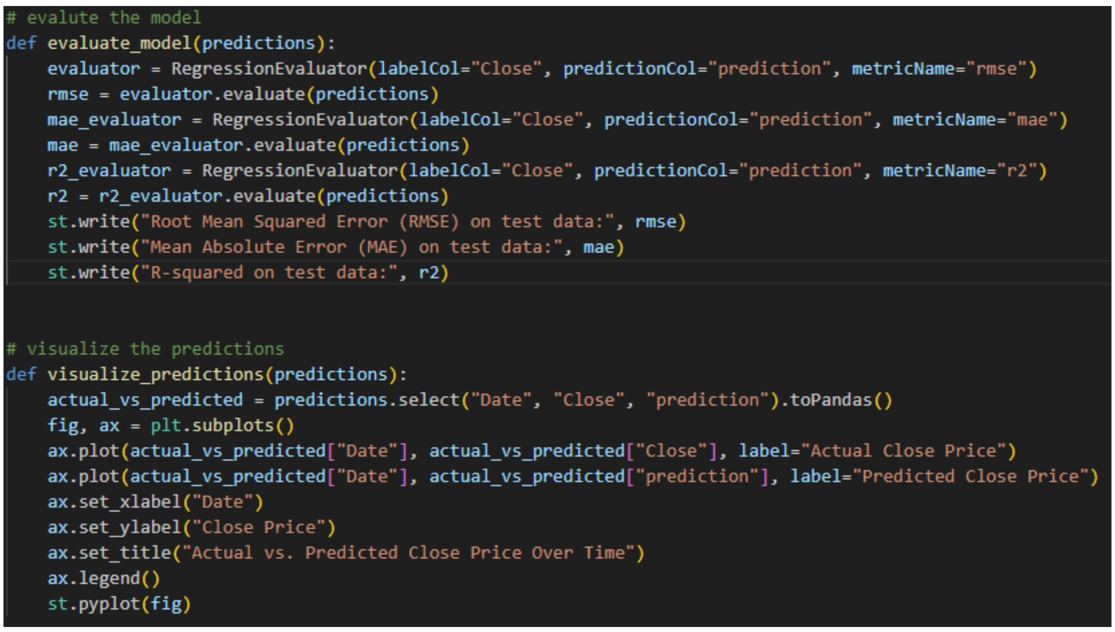

```python
# split the dataset to train_dataset and test_dataset
def split_dataset(stock_data_spark, split_date):
    train_data = stock_data_spark.filter(col("date") <= split_date)
    test_data = stock_data_spark.filter(col("date") > split_date)
    return train_data ,test_data
```

And then use the training dataset to make a linear regression model.


```python
# bulid the linear regression model
def bulid_model(feature_columns, train_data):
    vector_assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
    linear_regression = LinearRegression(featuresCol="features", labelCol="Close", predictionCol="prediction")
    pipeline = Pipeline(stages=[vector_assembler, linear_regression])
    model = pipeline.fit(train_data)
    return model
```

The next step is using the model we bullied to evaluate the result and visualize the prediction base on test dataset.

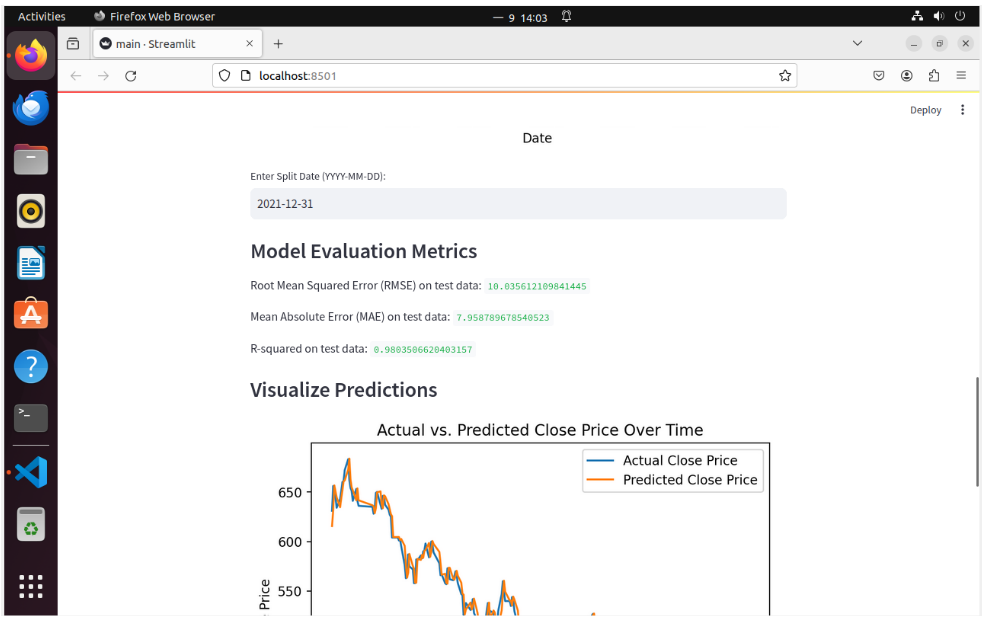
There are three evaluate number in the program ,they are root mean square error(RMSE) 、mean absolute error(MAE)and the R squared.



```python
# evalute the model
def evaluate_model(predictions):
    evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction", metricName="rmse")
    rmse = evaluator.evaluate(predictions)
    mae_evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction", metricName="mae")
    mae = mae_evaluator.evaluate(predictions)
    r2_evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction", metricName="r2")
    r2 = r2_evaluator.evaluate(predictions)
    st.write("Root Mean Squared Error (RMSE) on test data:", rmse)
    st.write("Mean Absolute Error (MAE) on test data:", mae)
    st.write("R-squared on test data:", r2)


# visualize the predictions
def visualize_predictions(predictions):
    actual_vs_predicted = predictions.select("Date", "Close", "prediction").toPandas()
    fig, ax = plt.subplots()
    ax.plot(actual_vs_predicted["Date"], actual_vs_predicted["Close"], label="Actual Close Price")
    ax.plot(actual_vs_predicted["Date"], actual_vs_predicted["prediction"], label="Predicted Close Price")
    ax.set_xlabel("Date")
    ax.set_ylabel("Close Price")
    ax.set_title("Actual vs. Predicted Close Price Over Time")
    ax.legend()
    st.pyplot(fig)
```

This is the example output from above function.



Date

Enter Split Date (YYYY-MM-DD):

2021-12-31

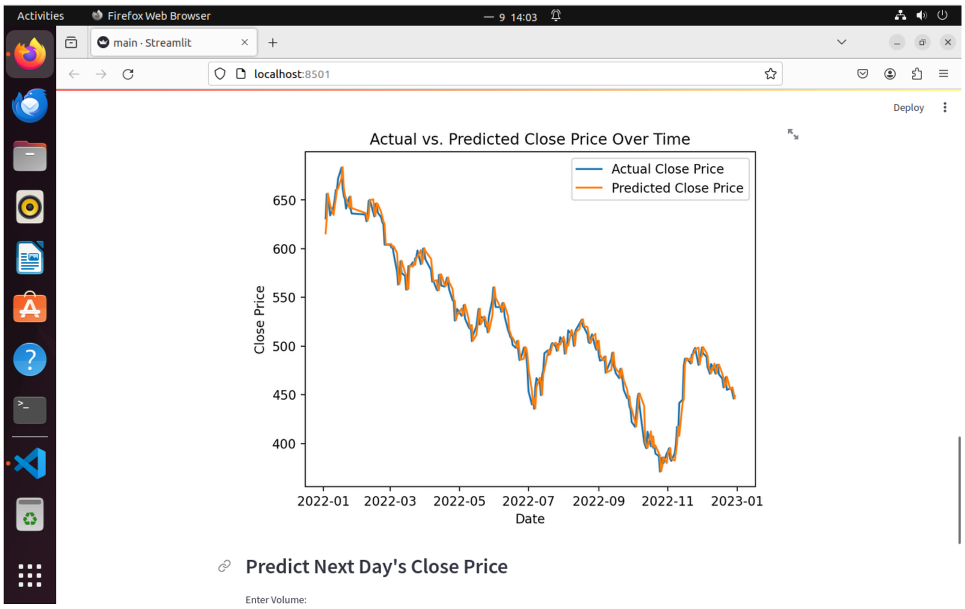
Model Evaluation Metrics

Root Mean Squared Error (RMSE) on test data: `10.035612109841445`

Mean Absolute Error (MAE) on test data: `7.958789678540523`

R-squared on test data: `0.9803506620403157`

Visualize Predictions



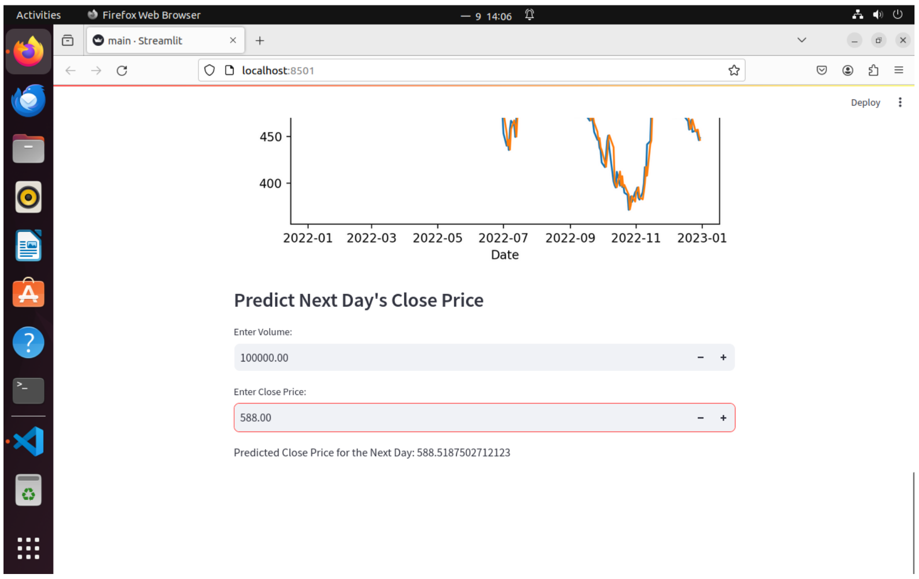
Predict Next Day's Close Price

Enter Volume:

The last step is let the user enter the close price and the transaction today to predict tomorrow close price.

```python
# input the volume and close price to predit the close price in the next day
def predict_next_day_value(model, spark):
    st.subheader("Predict Next Day's Close Price")
    user_volume = st.number_input("Enter Volume:", min_value=0.0)
    user_close = st.number_input("Enter Close Price:", min_value=0.0)
    user_data = pd.DataFrame({"yesterday_volume": [user_volume], "yesterday_close": [user_close]})
    user_spark_data = spark.createDataFrame(user_data)
    user_predictions = model.transform(user_spark_data)
    predicted_close = user_predictions.select("prediction").collect()[0]["prediction"]
    st.write(f"Predicted Close Price for the Next Day: {predicted_close}")
```

And below is an example of the output.



Predict Next Day's Close Price

Enter Volume:

100000.00

Enter Close Price:

588.00

```
Predicted Close Price for the Next Day: 588.5187502712123
```

6.Full code.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, lag
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql import Window
import matplotlib.pyplot as plt
import logging
import yfinance as yf
import streamlit as st
import pandas as pd


# get the stock data by yfinance
def get_stock_data(symbol, start_date, end_date):
    try:
        stock_data_pd = yf.download(symbol, start=start_date, end=end_date)
        if stock_data_pd.empty:
            raise ValueError("No data available for the given symbol and date range.")
            return None
        stock_data_pd.reset_index(inplace=True)
        stock_data_pd = stock_data_pd.dropna()
        return stock_data_pd
    except Exception as e:
        st.error(f"An error occurred while fetching stock data: {str(e)}")
        return None


# to transform the dataset to spark dataset
def transform_dataset(spark, stock_data_pd):
    stock_data_spark = spark.createDataFrame(stock_data_pd)
    stock_data_spark = stock_data_spark.withColumn("date", to_date(stock_data_spark["Date"],
'yyyy-MM-dd'))
    stock_data_spark = stock_data_spark.orderBy("date")
    window_spec = Window().orderBy("date")
    stock_data_spark = stock_data_spark.withColumn("yesterday_volume",
lag("Volume").over(window_spec)).na.drop()
    stock_data_spark = stock_data_spark.withColumn("yesterday_close",
lag("Close").over(window_spec)).na.drop()
```

```
    stock_data_spark = stock_data_spark.dropna()
    return stock_data_spark


#visualize the volume from start_day to end_day
def visualize_volume(stock_data_spark):
    visualize_data = stock_data_spark.select("Date", "Volume").toPandas()
    fig, ax = plt.subplots()
    ax.plot(visualize_data["Date"], visualize_data["Volume"], label="Volume")
    ax.set_xlabel("Date")
    ax.set_ylabel("Volume")
    ax.set_title("Volume Over Time")
    ax.legend()
    st.pyplot(fig)


#visualize the close price from start_day to end_day
def visualize_close_price(stock_data_spark):
    visualize_data = stock_data_spark.select("Date", "Close").toPandas()
    fig, ax = plt.subplots()
    ax.plot(visualize_data["Date"], visualize_data["Close"], label="Close Price")
    ax.set_xlabel("Date")
    ax.set_ylabel("Close Price")
    ax.set_title("Close Price Over Time")
    ax.legend()
    st.pyplot(fig)


# split the dataset to train_dataset and test_dataset
def split_dataset(stock_data_spark, split_date):
    train_data = stock_data_spark.filter(col("date") <= split_date)
    test_data = stock_data_spark.filter(col("date") > split_date)
    return train_data ,test_data


# bulid the linear regression model
def bulid_model(feature_columns, train_data):
    vector_assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
    linear_regression = LinearRegression(featuresCol="features", labelCol="Close",
predictionCol="prediction")
```

```
    pipeline = Pipeline(stages=[vector_assembler, linear_regression])
    model = pipeline.fit(train_data)
    return model


# evalute the model
def evaluate_model(predictions):
    evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction",
metricName="rmse")
    rmse = evaluator.evaluate(predictions)
    mae_evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction",
metricName="mae")
    mae = mae_evaluator.evaluate(predictions)
    r2_evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction",
metricName="r2")
    r2 = r2_evaluator.evaluate(predictions)
    st.write("Root Mean Squared Error (RMSE) on test data:", rmse)
    st.write("Mean Absolute Error (MAE) on test data:", mae)
    st.write("R-squared on test data:", r2)


# visualize the predictions
def visualize_predictions(predictions):
    actual_vs_predicted = predictions.select("Date", "Close", "prediction").toPandas()
    fig, ax = plt.subplots()
    ax.plot(actual_vs_predicted["Date"], actual_vs_predicted["Close"], label="Actual Close
Price")
    ax.plot(actual_vs_predicted["Date"], actual_vs_predicted["prediction"], label="Predicted
Close Price")
    ax.set_xlabel("Date")
    ax.set_ylabel("Close Price")
    ax.set_title("Actual vs. Predicted Close Price Over Time")
    ax.legend()
    st.pyplot(fig)


# input the volume and close price to predit the close price in the next day
def predict_next_day_value(model, spark):
    st.subheader("Predict Next Day's Close Price")
```

```python
    user_volume = st.number_input("Enter Volume:", min_value=0.0)
    user_close = st.number_input("Enter Close Price:", min_value=0.0)
    user_data = pd.DataFrame({"yesterday_volume": [user_volume], "yesterday_close":
[user_close]})
    user_spark_data = spark.createDataFrame(user_data)
    user_predictions = model.transform(user_spark_data)
    predicted_close = user_predictions.select("prediction").collect()[0]["prediction"]
    st.write(f"Predicted Close Price for the Next Day: {predicted_close}")


def main():
    st.title("Stock Price Prediction App")
    st.subheader("Enter the information")
    symbol = st.text_input("Enter Stock Symbol (e.g., 2330.TW):", "2330.TW")
    start_date = st.text_input("Enter Start Date (YYYY-MM-DD):", "2000-01-01")
    end_date = st.text_input("Enter End Date (YYYY-MM-DD):", "2023-01-01")
    spark = SparkSession.builder.appName("StockAnalyze").getOrCreate()
    logger = spark._jvm.org.apache.log4j
    logger.LogManager.getLogger("org.apache.spark.sql.execution.window.WindowExec").setLev
el(logger.Level.ERROR)

    stock_data_pd = get_stock_data(symbol, start_date, end_date)
    if stock_data_pd is not None:
        stock_data_spark = transform_dataset(spark, stock_data_pd)
        st.subheader("Visualize Dataset")
        visualize_volume(stock_data_spark)
        visualize_close_price(stock_data_spark)

        split_date = st.text_input("Enter Split Date (YYYY-MM-DD):", "2021-12-31")
        train_data ,test_data= split_dataset(stock_data_spark, split_date)
        feature_columns = ["yesterday_volume", "yesterday_close"]
        model = bulid_model(feature_columns, train_data)
        predictions = model.transform(test_data)

        st.subheader("Model Evaluation Metrics")
        evaluate_model(predictions)
        st.subheader("Visualize Predictions")
        visualize_predictions(predictions)
        predict_next_day_value(model, spark)
    spark.stop()

if __name__ == "__main__":
    main()
```

7. How to operate

First ,install the spark in the local machine, and then install the necessary python library include yfinance、pyspark、pandas、matplotlib and the streamlit, the last step is go to the code directory in cmd and typing streamlit run main.py(example base on your file name)

8. conclusion

In conclusion, this stock price prediction application leverages Apache Spark and yfinance to create a robust and user-friendly platform for forecasting stock prices. The project incorporates various key components, including preprocessing, visualization, model building, evaluation, and user interaction through a Streamlit-based UI.

The utilization of yfinance facilitates the seamless acquisition of historical stock data, which is then transformed into a Spark DataFrame for efficient processing and analysis. The data preprocessing steps ensure that the dataset is well-structured, with additional features such as yesterday's transaction volume and close price included for enhanced predictive modeling.

The heart of the application lies in the implementation of a linear regression model using Apache Spark's machine learning capabilities. The model is trained on a user-defined dataset split, and its performance is evaluated using metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared. These metrics provide a quantitative assessment of the model's accuracy and reliability.

The Streamlit-based user interface enables users to interactively input stock symbols, date ranges, and other parameters to customize their linear regression model. The application also allows users to input current transaction volume and close price, providing them with a predicted close price for the following day.

In summary, this project not only showcases the technical prowess of leveraging big data tools like Apache Spark for predictive analytics but also emphasizes user engagement through a well-designed and intuitive interface