BRIGHT STORE

Sales Analysis

Introduction

The purpose of this presentation is to discuss the metrics and insights derived from the sales analysis of the daily trading of one of the products sold at Bright Retail Store.

The following metrics were derived

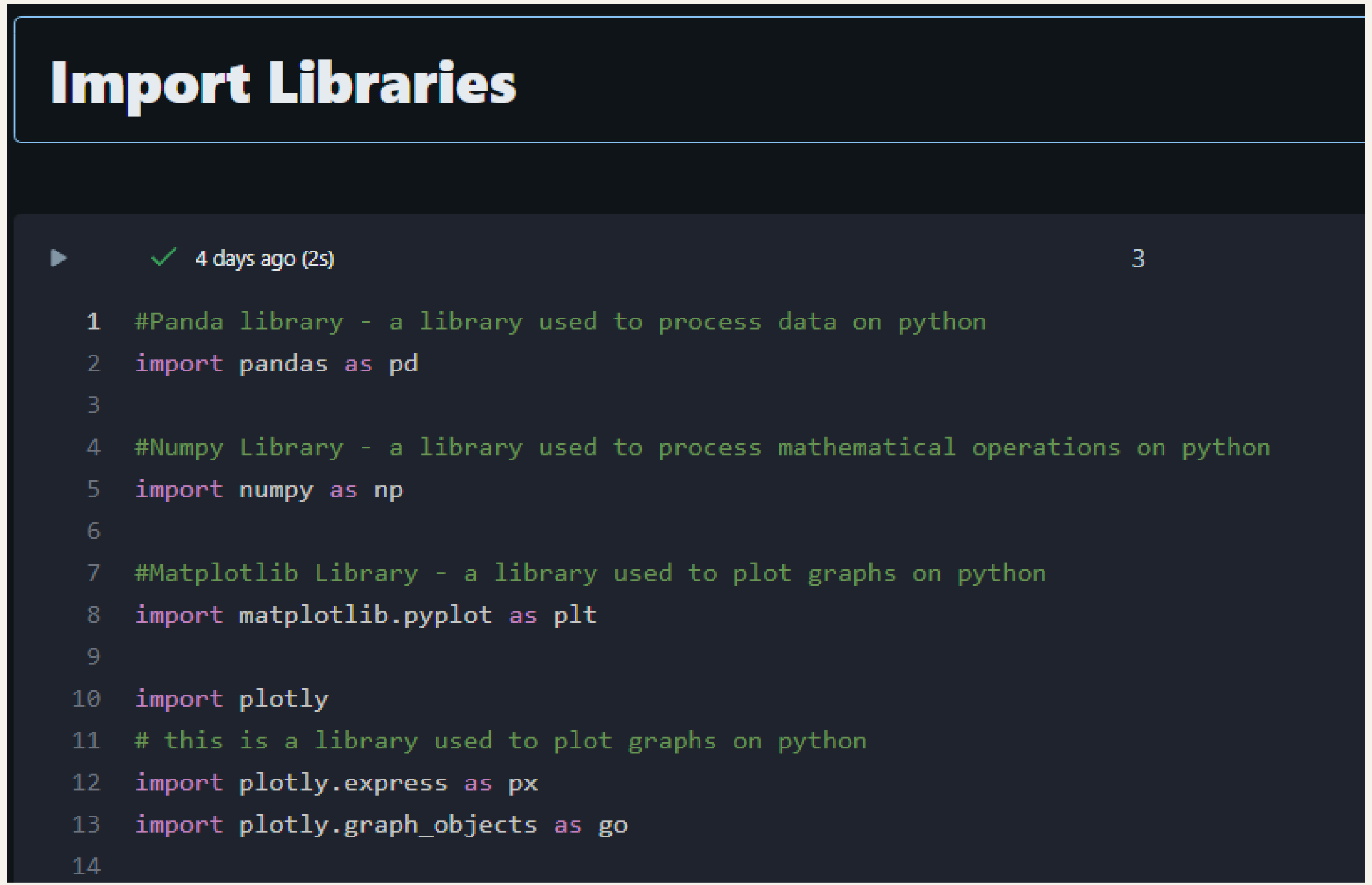
- Daily sales price per unit
- Average unit sales price
- % Gross profit per unit
- Price Elasticity of Demand

Loading data

The step after loading data to Databricks was to load Python libraries to help me process data

Followed by loading libraries that will allow me to process the given data:

- Pandas - to process data on python
- Numpy - to process mathematical operations python
- Matplotlib - to plot graphs on python
- Plotly - to plot graphs on python

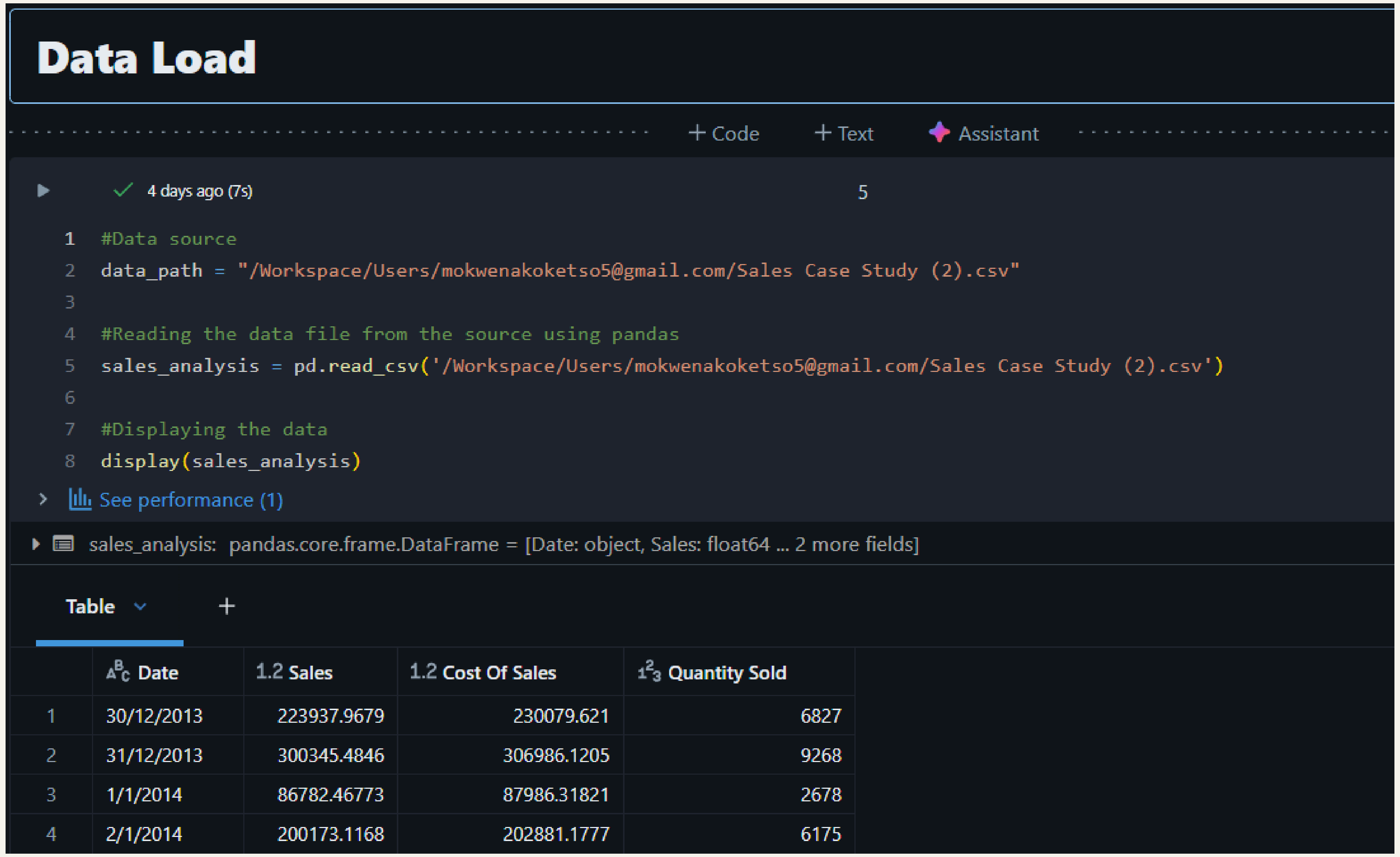


Import Libraries

```
#Panda library - a library used to process data on python
import pandas as pd

#Numpy Library - a library used to process mathematical operations on python
import numpy as np

#Matplotlib Library - a library used to plot graphs on python
import matplotlib.pyplot as plt

import plotly
# this is a library used to plot graphs on python
import plotly.express as px
import plotly.graph_objects as go

```

Data Load

- Assigned a data path for the given dataset
- Used python pandas pd.read_csv to read the file from the assigned data path



Data Load

```python
#Data source
data_path = "/Workspace/Users/mokwenakoketso5@gmail.com/Sales Case Study (2).csv"

#Reading the data file from the source using pandas
sales_analysis = pd.read_csv('/Workspace/Users/mokwenakoketso5@gmail.com/Sales Case Study (2).csv')

#Displaying the data
display(sales_analysis)
```

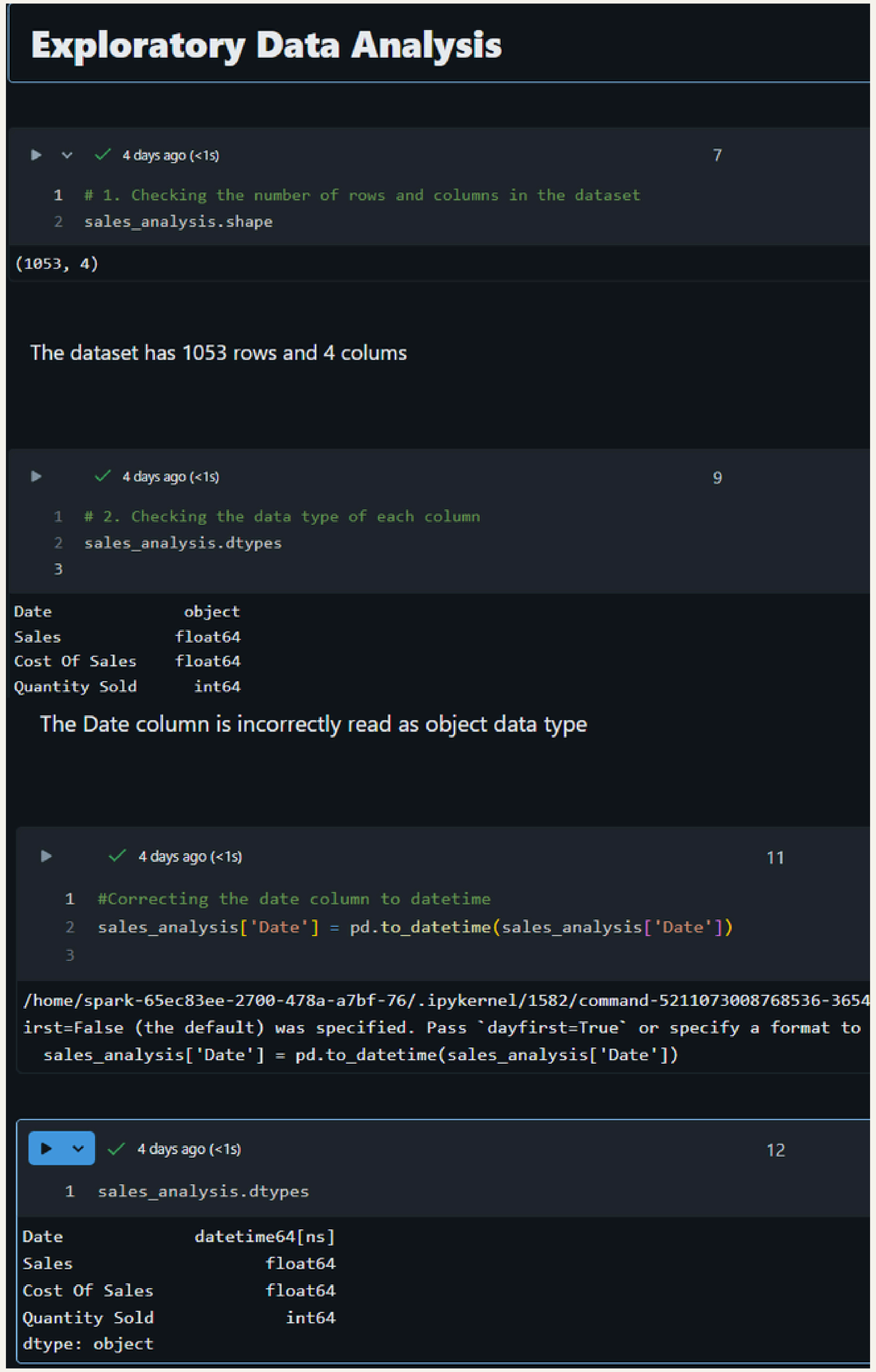
```
sales_analysis:  pandas.core.frame.DataFrame = [Date: object, Sales: float64 ... 2 more fields]
```

	Date	Sales	Cost Of Sales	Quantity Sold
1	30/12/2013	223937.9679	230079.621	6827
2	31/12/2013	300345.4846	306986.1205	9268
3	1/1/2014	86782.46773	87986.31821	2678
4	2/1/2014	200173.1168	202881.1777	6175

Exploratory Data Analysis

Used the following python functions to explore the data:

- .shape - to check the number of rows and columns in the dataset
- .dtypes - to checking the data type of each column
- pd.to_date - to correct the date from object to datetime data type
- pd.duplicated().sum() - to check for duplicates
- .isnull().sum() - to check for missing values



Exploratory Data Analysis

4 days ago (<1s) 7

```
# 1. Checking the number of rows and columns in the dataset
sales_analysis.shape
```

```
(1053, 4)
```

The dataset has 1053 rows and 4 colums

4 days ago (<1s) 9

```
# 2. Checking the data type of each column
sales_analysis.dtypes
```

```
Date               object
Sales             float64
Cost Of Sales     float64
Quantity Sold       int64
```

The Date column is incorrectly read as object data type

4 days ago (<1s) 11

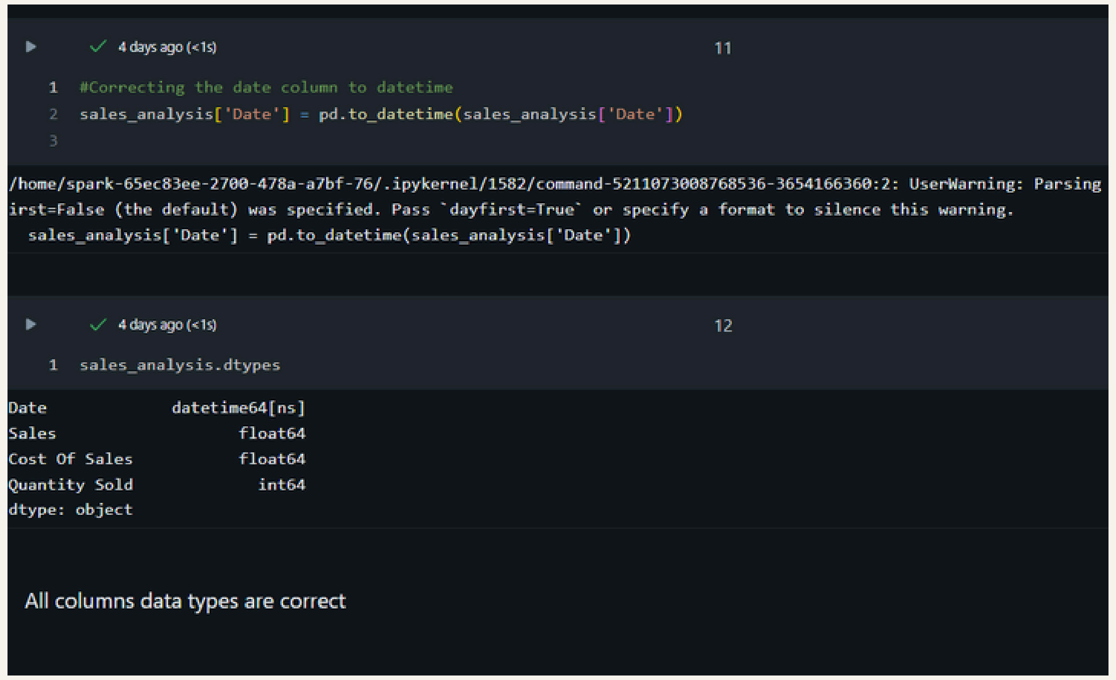
```
#Correcting the date column to datetime
sales_analysis['Date'] = pd.to_datetime(sales_analysis['Date'])
```

```
/home/spark-65ec83ee-2700-478a-a7bf-76/.ipykernel/1582/command-5211073008768536-3654
irst=False (the default) was specified. Pass `dayfirst=True` or specify a format to
  sales_analysis['Date'] = pd.to_datetime(sales_analysis['Date'])
```

4 days ago (<1s) 12

```
sales_analysis.dtypes
```

```
Date             datetime64[ns]
Sales                   float64
Cost Of Sales           float64
Quantity Sold             int64
dtype: object
```



4 days ago (<1s) 11

```
#Correcting the date column to datetime
sales_analysis['Date'] = pd.to_datetime(sales_analysis['Date'])
```

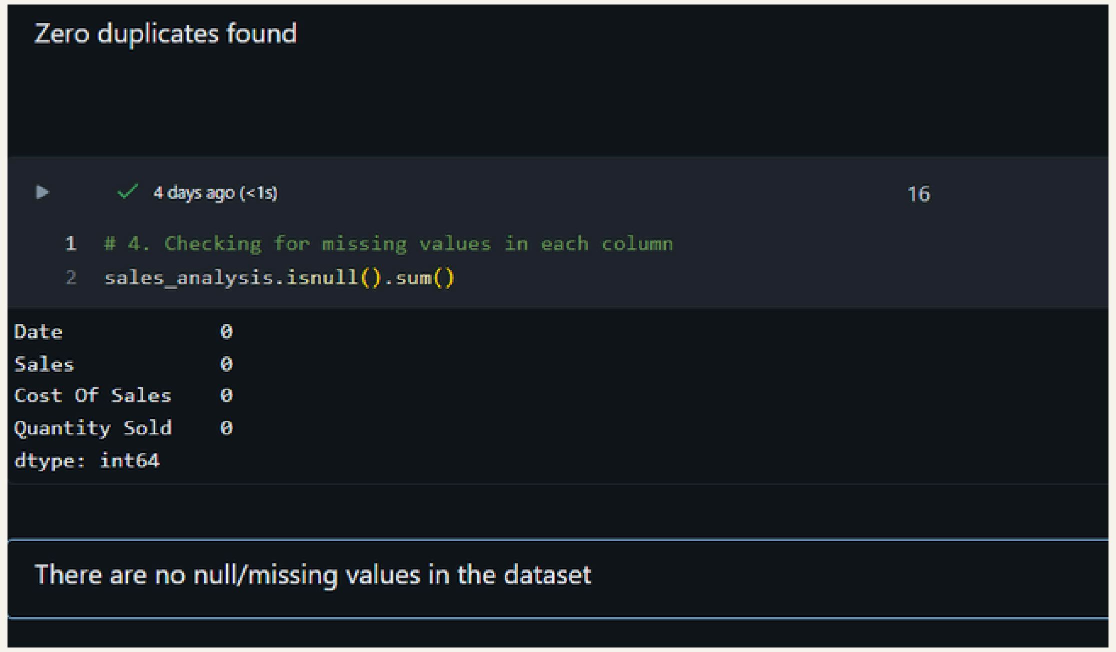
```
/home/spark-65ec83ee-2700-478a-a7bf-76/.ipykernel/1582/command-5211073008768536-3654166360:2: UserWarning: Parsing
irst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence this warning.
  sales_analysis['Date'] = pd.to_datetime(sales_analysis['Date'])
```

4 days ago (<1s) 12

```
sales_analysis.dtypes
```

```
Date             datetime64[ns]
Sales                   float64
Cost Of Sales           float64
Quantity Sold             int64
dtype: object
```

All columns data types are correct



Zero duplicates found

4 days ago (<1s) 16

```
# 4. Checking for missing values in each column
sales_analysis.isnull().sum()
```

```
Date             0
Sales            0
Cost Of Sales    0
Quantity Sold    0
dtype: int64
```

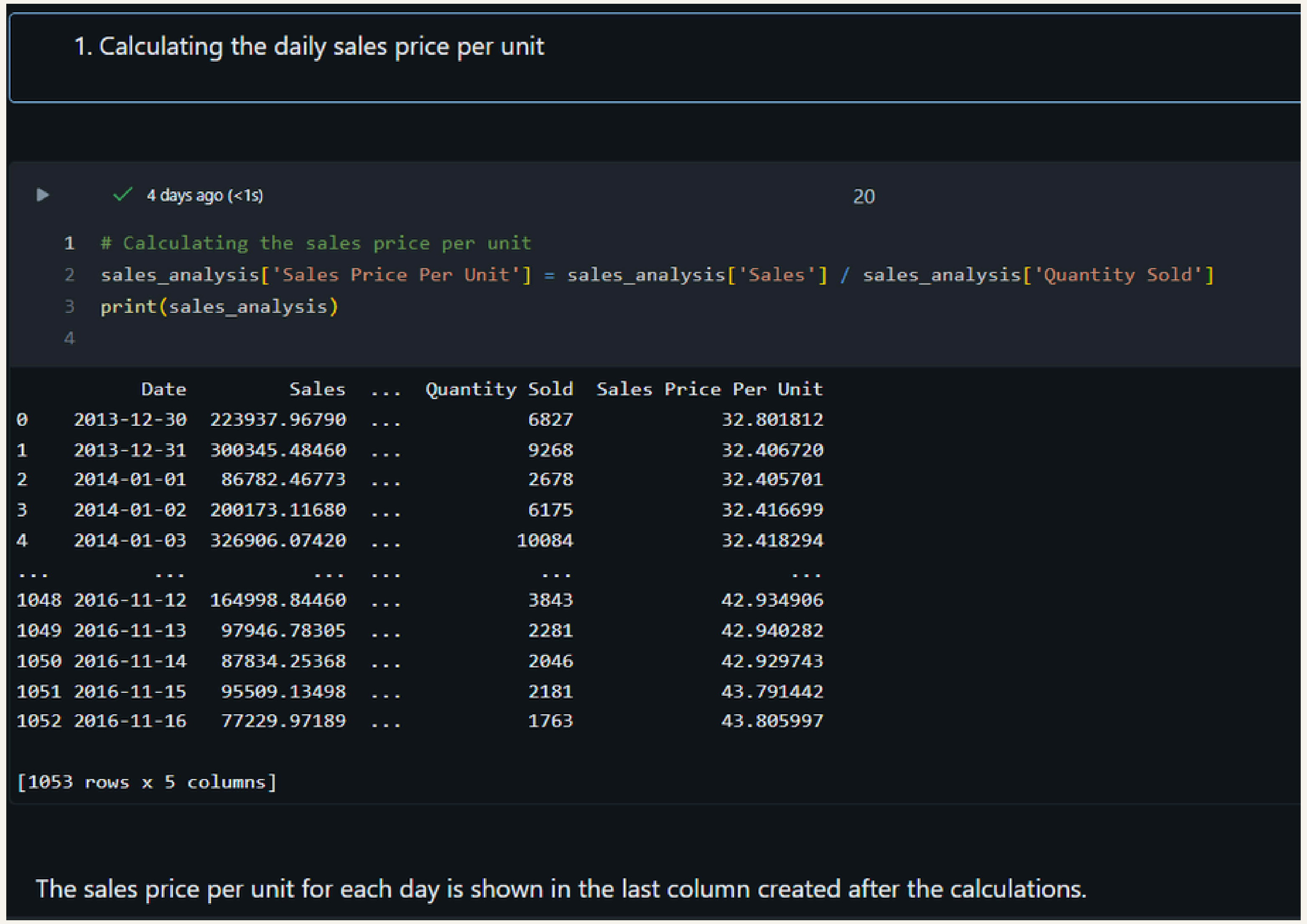
There are no null/missing values in the dataset

Data metrics and insights

1. **Calculating the daily sales price per unit**

- The sales price per unit was calculated from the formula as shown;

$$\text{Sales Price Per Unit} = \frac{\text{Total Sales}}{\text{Total Quantity Sold}}$$



1. Calculating the daily sales price per unit

```python
# Calculating the sales price per unit
sales_analysis['Sales Price Per Unit'] = sales_analysis['Sales'] / sales_analysis['Quantity Sold']
print(sales_analysis)
```

```
          Date         Sales  ...  Quantity Sold  Sales Price Per Unit
0    2013-12-30  223937.96790  ...           6827             32.801812
1    2013-12-31  300345.48460  ...           9268             32.406720
2    2014-01-01   86782.46773  ...           2678             32.405701
3    2014-01-02  200173.11680  ...           6175             32.416699
4    2014-01-03  326906.07420  ...          10084             32.418294
...         ...           ...  ...            ...                   ...
1048 2016-11-12  164998.84460  ...           3843             42.934906
1049 2016-11-13   97946.78305  ...           2281             42.940282
1050 2016-11-14   87834.25368  ...           2046             42.929743
1051 2016-11-15   95509.13498  ...           2181             43.791442
1052 2016-11-16   77229.97189  ...           1763             43.805997

[1053 rows x 5 columns]
```

The sales price per unit for each day is shown in the last column created after the calculations.

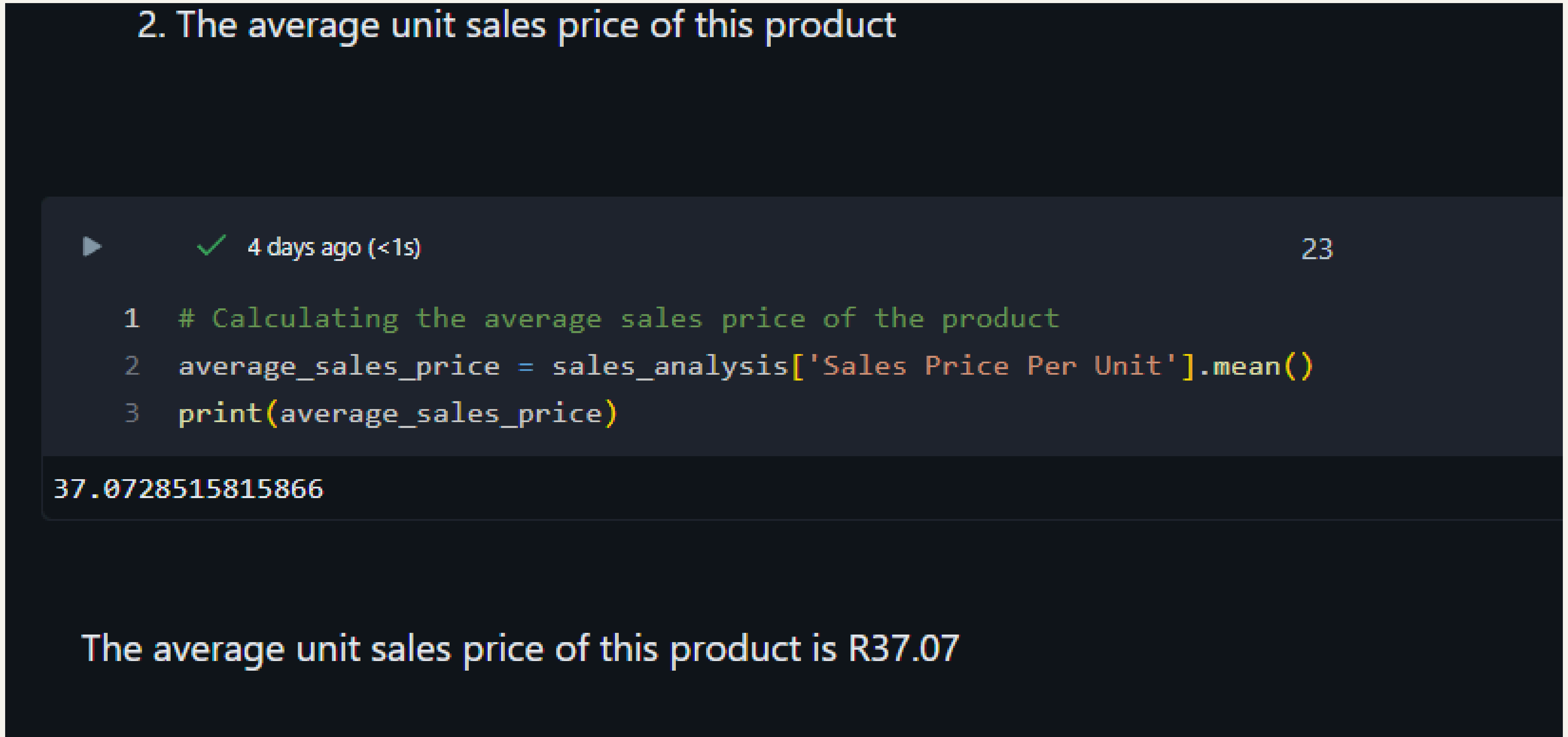
Data metrics and insights

2. **What is the average unt sales price of this product**

- The average unit sales price was calculated from the formula as shown;

$$\text{Sales Price Per Unit} = \frac{\text{Product Revenue}}{\text{Total Quantity Sold}}$$

- The average sales price of this product is R37.07



2. The average unit sales price of this product

```
# Calculating the average sales price of the product
average_sales_price = sales_analysis['Sales Price Per Unit'].mean()
print(average_sales_price)
```

```
37.0728515815866
```

The average unit sales price of this product is R37.07

Data metrics and insights

3. What is the daily % gross profit

- The daily % gross profit was calculated from the formulas as shown;

$$\%Gross = (\frac{GrossProfit}{Revenue})(100)$$

$$Revenue = Sales \times QuantitySold$$

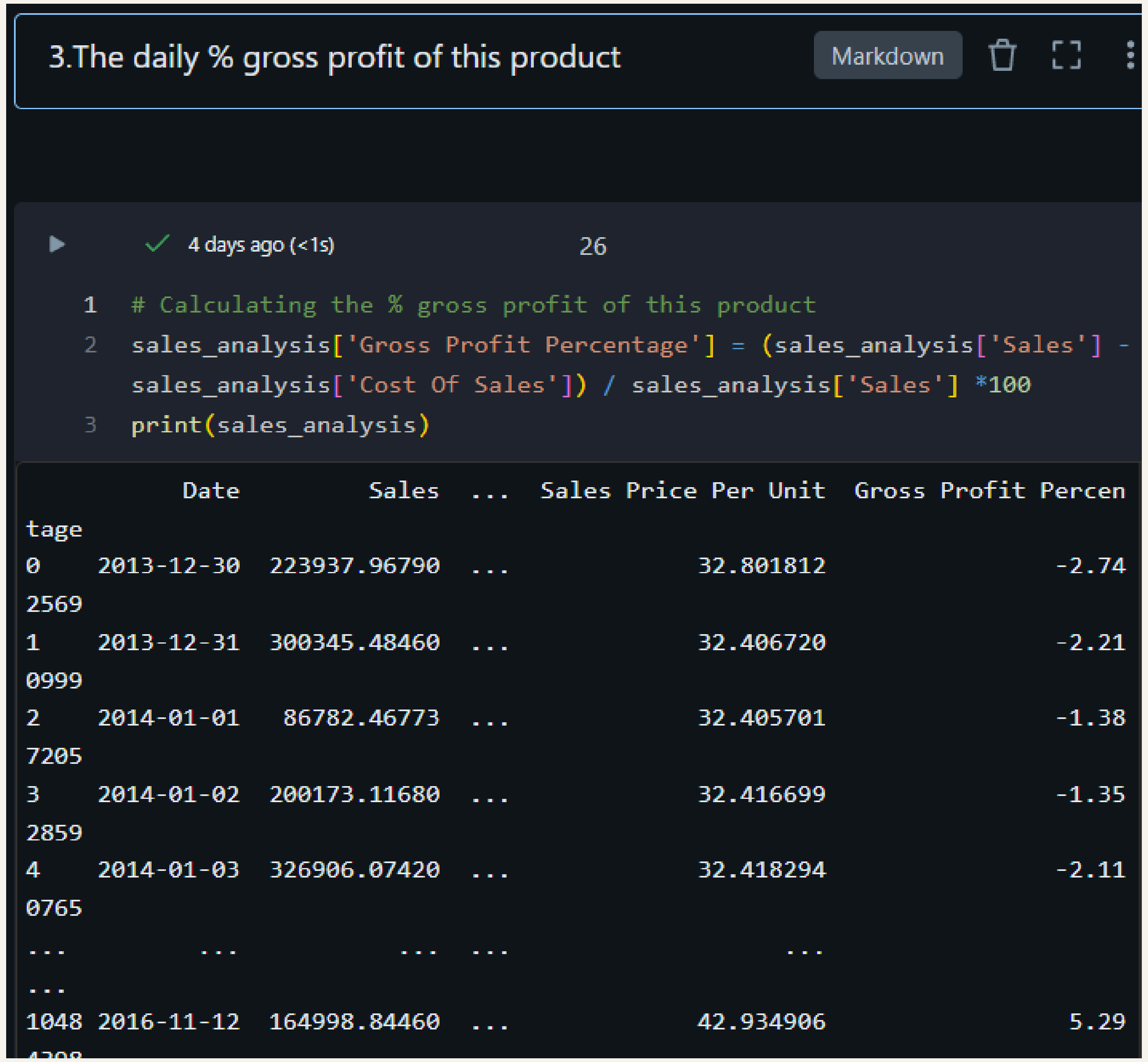
$$GrossProfit = Revenue - CostofSales$$

- The formula required revenue and gross profit and their formulas are provided

- Interpretation:

% Gross profit and Gross profit both depend on the Sales and Cost. If the Cost of sales is greater than the Sales, the gross profit and percentage becomes a negative and means there was a loss of profit.

A positive % Gross Profit means the Sales were greater than the cost of sales, meaning profit was gained



3.The daily % gross profit of this product

```
# Calculating the % gross profit of this product
sales_analysis['Gross Profit Percentage'] = (sales_analysis['Sales'] -
sales_analysis['Cost Of Sales']) / sales_analysis['Sales'] *100
print(sales_analysis)
```

```
           Date         Sales  ...  Sales Price Per Unit  Gross Profit Percen
tage
0    2013-12-30  223937.96790  ...             32.801812                 -2.74
2569
1    2013-12-31  300345.48460  ...             32.406720                 -2.21
0999
2    2014-01-01   86782.46773  ...             32.405701                 -1.38
7205
3    2014-01-02  200173.11680  ...             32.416699                 -1.35
2859
4    2014-01-03  326906.07420  ...             32.418294                 -2.11
0765
...         ...           ...  ...                   ...
...
1048 2016-11-12  164998.84460  ...             42.934906                  5.29
```

Data metrics and insights

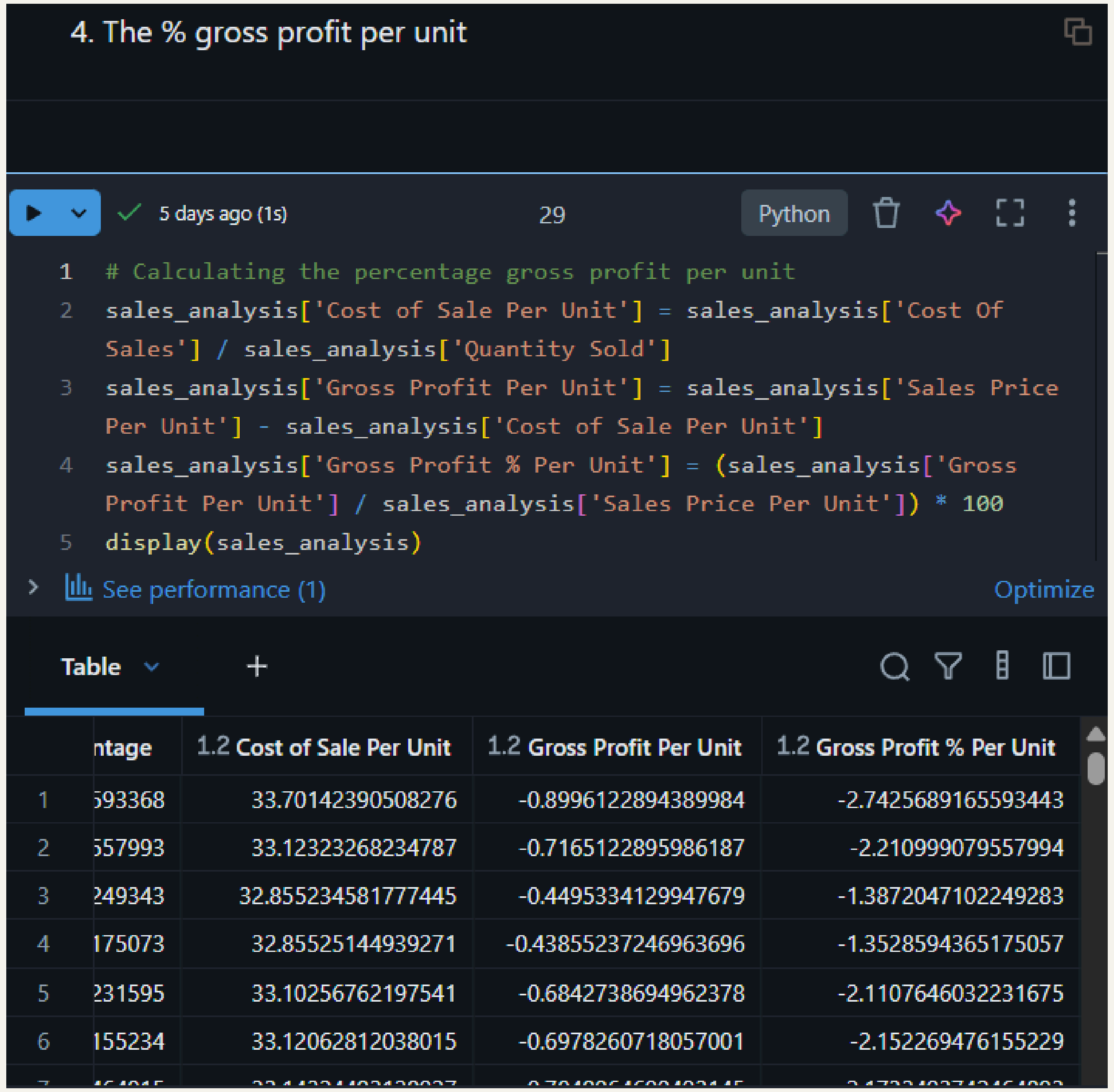
4. What is the daily % gross profit per unit

- The % gross profit per unit is obtained from calculating the revenue and gross profit per unit

$$\%Gross = (\frac{GrossProfit}{Revenue})(100)$$

$$Revenue = Sales \times QuantitySold$$

$$GrossProfit = Revenue - CostofSales$$



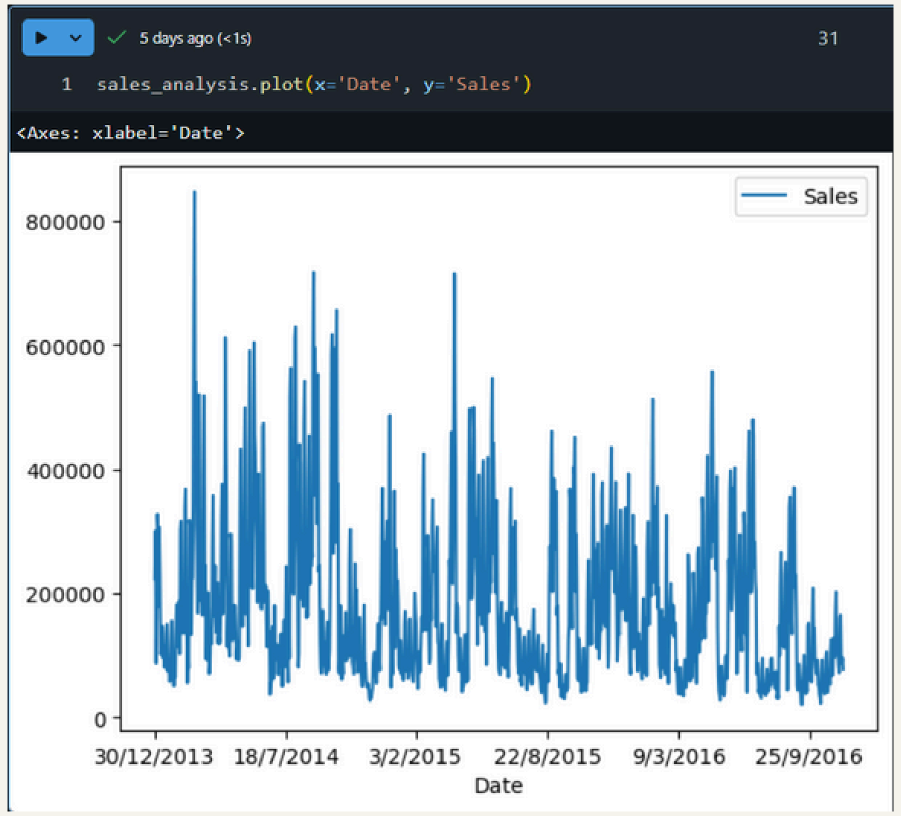
4. The % gross profit per unit

```python
# Calculating the percentage gross profit per unit
sales_analysis['Cost of Sale Per Unit'] = sales_analysis['Cost Of Sales'] / sales_analysis['Quantity Sold']
sales_analysis['Gross Profit Per Unit'] = sales_analysis['Sales Price Per Unit'] - sales_analysis['Cost of Sale Per Unit']
sales_analysis['Gross Profit % Per Unit'] = (sales_analysis['Gross Profit Per Unit'] / sales_analysis['Sales Price Per Unit']) * 100
display(sales_analysis)
```

	ntage	1.2 Cost of Sale Per Unit	1.2 Gross Profit Per Unit	1.2 Gross Profit % Per Unit
1	593368	33.70142390508276	-0.8996122894389984	-2.7425689165593443
2	557993	33.12323268234787	-0.7165122895986187	-2.210999079557994
3	249343	32.855234581777445	-0.4495334129947679	-1.3872047102249283
4	175073	32.85525144939271	-0.43855237246963696	-1.3528594365175057
5	231595	33.10256762197541	-0.6842738694962378	-2.1107646032231675
6	155234	33.12062812038015	-0.6978260718057001	-2.152269476155229

Data metrics and insights

5. **Pick any 5 periods during which this product was on promotion. How was the Price Elasticity of Demand.**



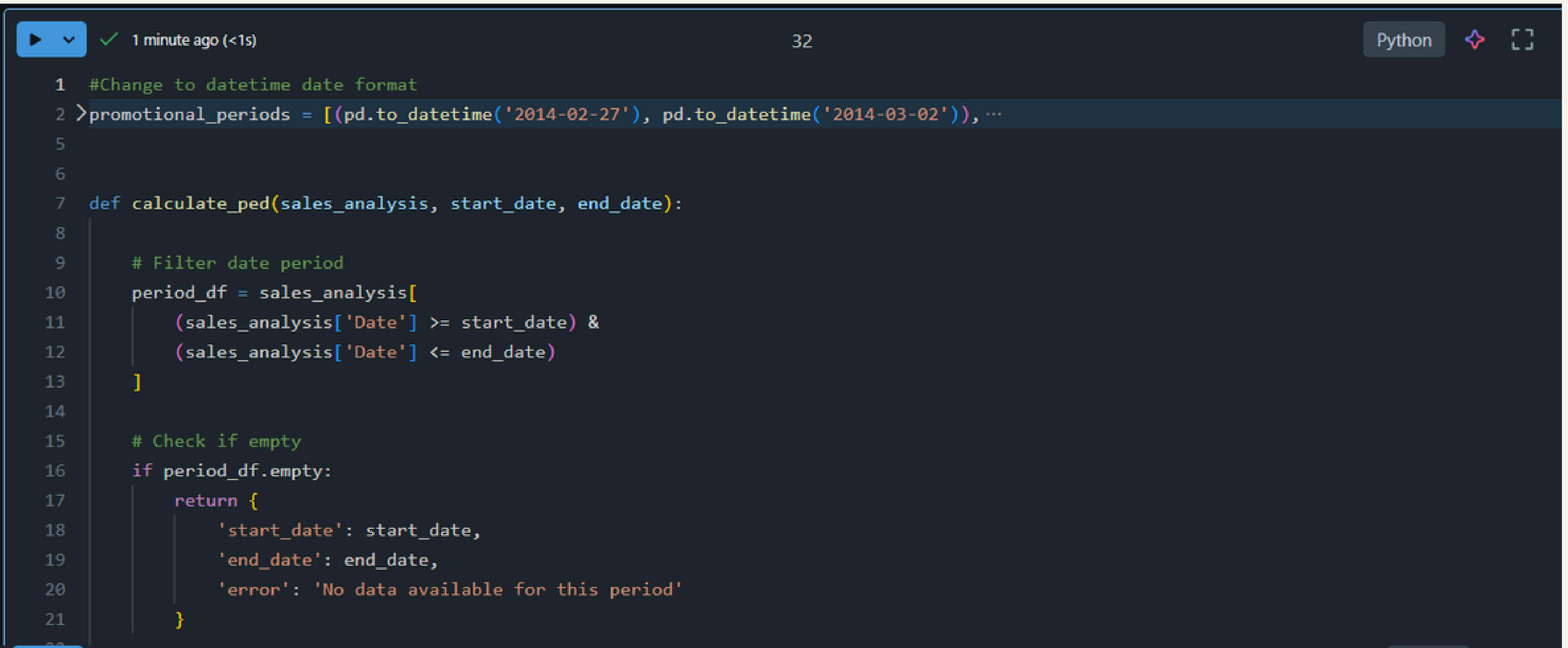
```
sales_analysis.plot(x='Date', y='Sales')
```

```
<Axes: xlabel='Date'>
```

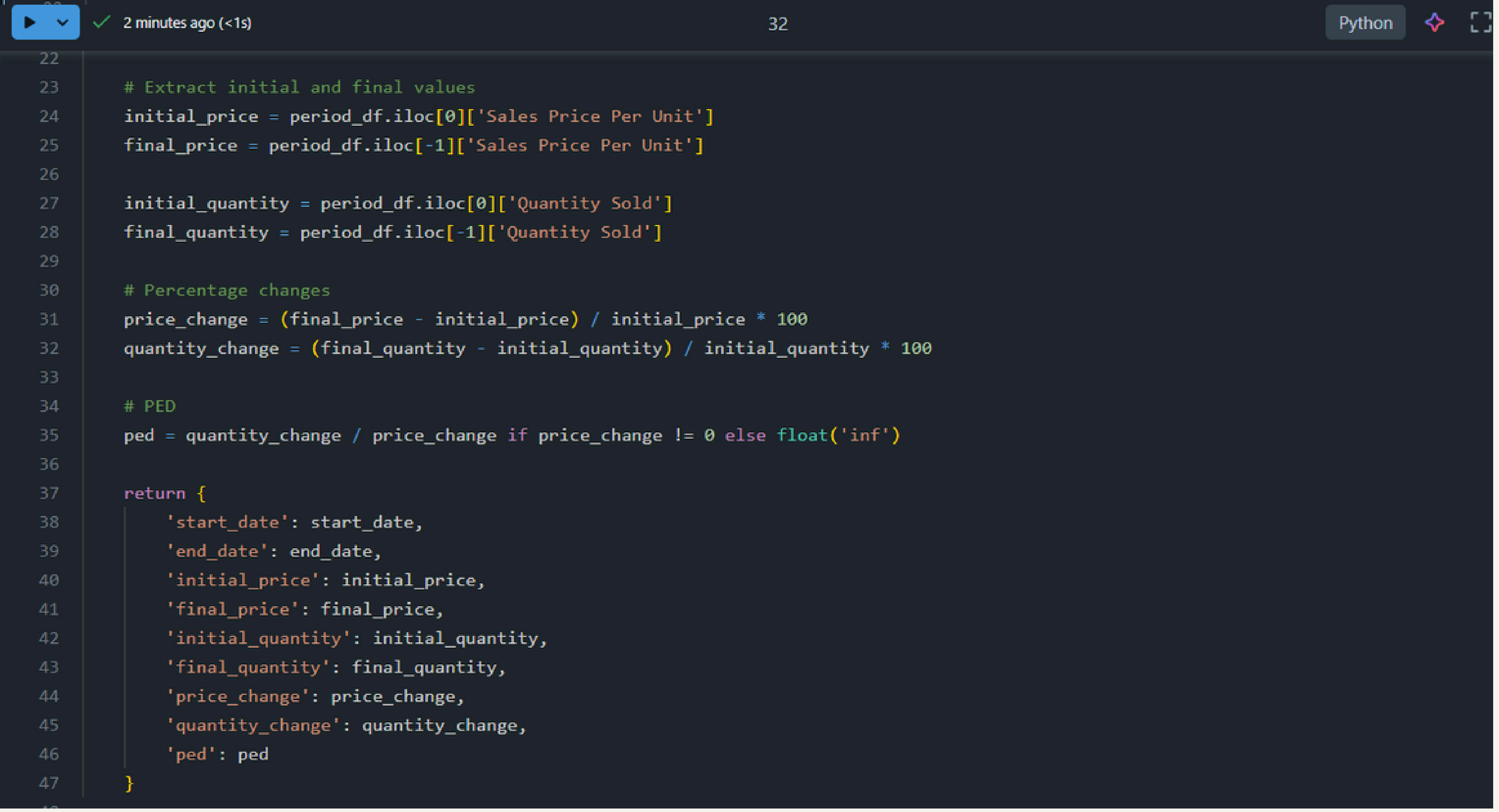
- Promotional periods are shown by the high peaks on the above, which is shown on 27/02/2014 -02/03/2014, 28/08/2014-31/08/2014 and 30/02/2015-05/04/2015

$$PriceElasticityofDemand = \frac{\%\backslash changeinquantitydemanded}{\%\backslash changeinprice}$$

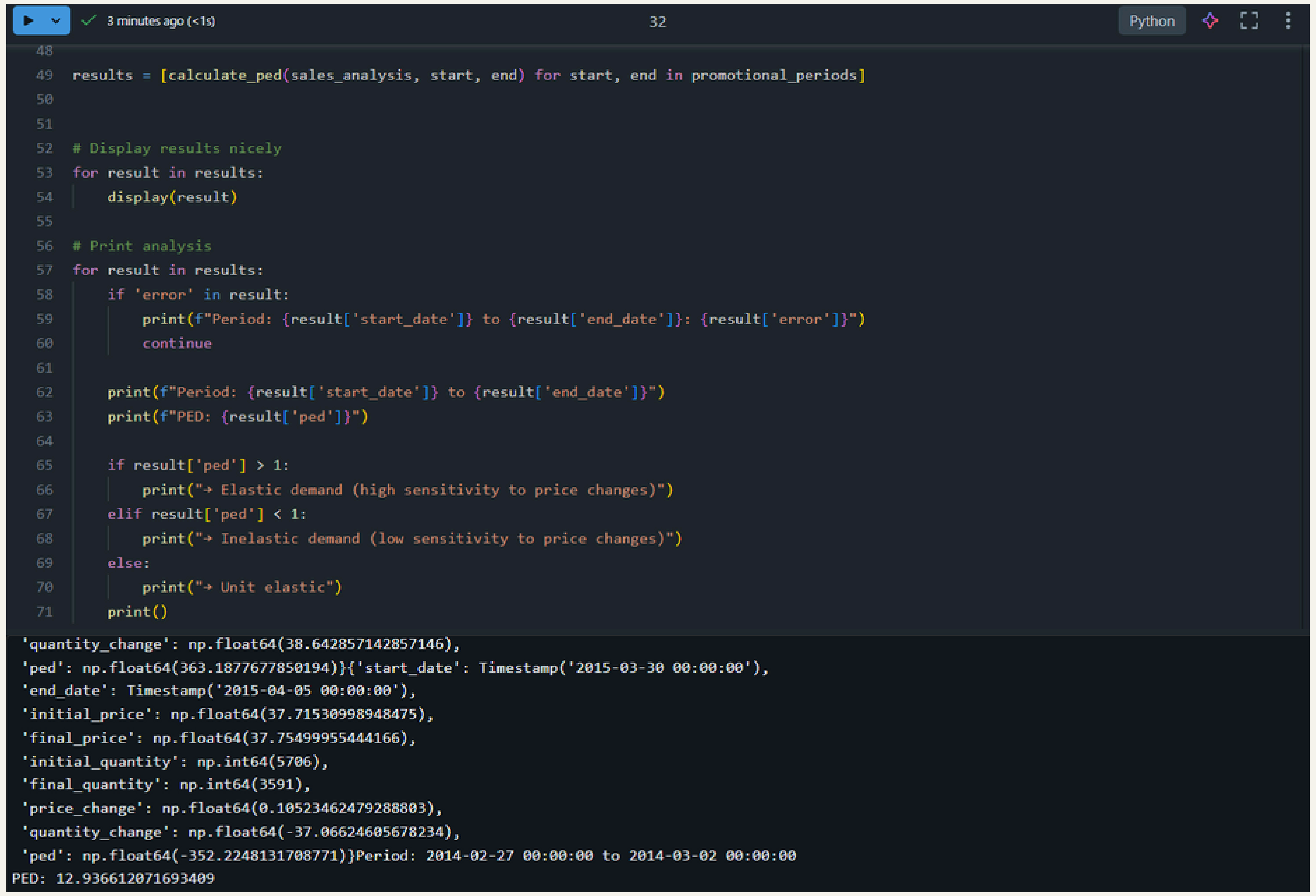
- The following shows how the PED was calculated using python functions



```python
#Change to datetime date format
promotional_periods = [(pd.to_datetime('2014-02-27'), pd.to_datetime('2014-03-02')), ⋯


def calculate_ped(sales_analysis, start_date, end_date):

    # Filter date period
    period_df = sales_analysis[
        (sales_analysis['Date'] >= start_date) &
        (sales_analysis['Date'] <= end_date)
    ]

    # Check if empty
    if period_df.empty:
        return {
            'start_date': start_date,
            'end_date': end_date,
            'error': 'No data available for this period'
        }
```



```python

    # Extract initial and final values
    initial_price = period_df.iloc[0]['Sales Price Per Unit']
    final_price = period_df.iloc[-1]['Sales Price Per Unit']

    initial_quantity = period_df.iloc[0]['Quantity Sold']
    final_quantity = period_df.iloc[-1]['Quantity Sold']

    # Percentage changes
    price_change = (final_price - initial_price) / initial_price * 100
    quantity_change = (final_quantity - initial_quantity) / initial_quantity * 100

    # PED
    ped = quantity_change / price_change if price_change != 0 else float('inf')

    return {
        'start_date': start_date,
        'end_date': end_date,
        'initial_price': initial_price,
        'final_price': final_price,
        'initial_quantity': initial_quantity,
        'final_quantity': final_quantity,
        'price_change': price_change,
        'quantity_change': quantity_change,
        'ped': ped
    }
```

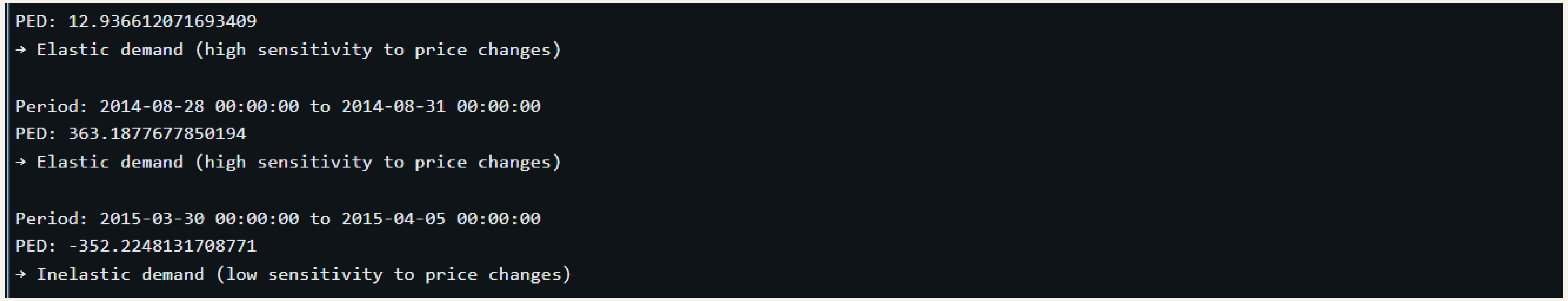


```python

results = [calculate_ped(sales_analysis, start, end) for start, end in promotional_periods]


# Display results nicely
for result in results:
    display(result)

# Print analysis
for result in results:
    if 'error' in result:
        print(f"Period: {result['start_date']} to {result['end_date']}: {result['error']}")
        continue

    print(f"Period: {result['start_date']} to {result['end_date']}")
    print(f"PED: {result['ped']}")

    if result['ped'] > 1:
        print("→ Elastic demand (high sensitivity to price changes)")
    elif result['ped'] < 1:
        print("→ Inelastic demand (low sensitivity to price changes)")
    else:
        print("→ Unit elastic")
    print()
```

```
 'quantity_change': np.float64(38.642857142857146),
 'ped': np.float64(363.1877677850194)}{'start_date': Timestamp('2015-03-30 00:00:00'),
 'end_date': Timestamp('2015-04-05 00:00:00'),
 'initial_price': np.float64(37.71530998948475),
 'final_price': np.float64(37.75499955444166),
 'initial_quantity': np.int64(5706),
 'final_quantity': np.int64(3591),
 'price_change': np.float64(0.10523462479288803),
 'quantity_change': np.float64(-37.06624605678234),
 'ped': np.float64(-352.2248131708771)}Period: 2014-02-27 00:00:00 to 2014-03-02 00:00:00
PED: 12.936612071693409
```

Price Elasticity of Demand Output

- The formula required revenue and gross profit and their formulas are provided



```
PED: 12.936612071693409
→ Elastic demand (high sensitivity to price changes)

Period: 2014-08-28 00:00:00 to 2014-08-31 00:00:00
PED: 363.1877677850194
→ Elastic demand (high sensitivity to price changes)

Period: 2015-03-30 00:00:00 to 2015-04-05 00:00:00
PED: -352.2248131708771
→ Inelastic demand (low sensitivity to price changes)
```

Interpretations and Insights

- Price Elasticity of Demand measures how customers are sensitive to price changes
- It also measures how quantity demands change with price change
- Elastic it is when the PED>1, which means quantity demands drop when prices increase because customers are sensitive to price changes
- Inelastic it is when the PED<1, which means quantity demands do not change when prices increase because customers are not sensitive to change
- From the output we can see that PED of the product is mostly elastic, meaning the customers are sensitive to its price changes

The End