Invoke Bedrock model for code generation

> *This notebook should work well with the* ***`Python 3`*** *kernel in SageMaker Studio*

Introduction

In this notebook we show you how to use a LLM to generate code based on the text prompt. We will use Bedrock's Claude 3 Sonnet using the Boto3 API.

The prompt used in this example is called a zero-shot prompt because we are not providing any examples of text other than the prompt.

Note: *This notebook can be run within or outside of AWS environment.*

Context

To demonstrate the code generation capability of Amazon Bedrock, we will explore the use of Boto3 client to communicate with Amazon Bedrock API. We will demonstrate different configurations available as well as how simple input can lead to desired outputs. We will explore code generation for two use cases:

1. Python code generation for analytical QnA
2. SQL query generation

Pattern

In both use cases, we will simply provide the Amazon Bedrock API with an input consisting of a task, an instruction and an input for the model under the hood to generate an output without providing any additional example. The purpose here is to demonstrate how the powerful LLMs easily understand the task at hand and generate compelling outputs.

Use case 1 - Python code generation for Analytical QnA

To demonstrate the generation capability of models in Amazon Bedrock, let's take the use case of code generation with Python to do some basic analytical QnA.

Persona

You are Moe, a Data Analyst, at AnyCompany. The company wants to understand its sales performance for different products for different products over the past year. You have been provided a dataset named sales.csv. The dataset contains the following columns:

- Date (YYYY-MM-DD) format
- Product_ID (unique identifer for each product)
- Price (price at which each product was sold)

Implementation

To fulfill this use case, in this notebook we will show how to generate code for a given prompt. We will use the Anthropic Claude 3 using the Amazon Bedrock API with Boto3 client.

Setup

In [1]:
```
import json
import os
import sys

import boto3
import botocore

boto3_bedrock = boto3.client('bedrock-runtime')
```

Code Generation

Following on the use case explained above, let's prepare an input for the Amazon Bedrock service to generate python program for our use-case.

Lab setup - create sample sales.csv data for this lab.

In [2]:
```
# create sales.csv file
import csv

data = [
    ["date", "product_id", "price", "units_sold"],
    ["2023-01-01", "P001", 50, 20],
    ["2023-01-02", "P002", 60, 15],
    ["2023-01-03", "P001", 50, 18],
    ["2023-01-04", "P003", 70, 30],
    ["2023-01-05", "P001", 50, 25],
    ["2023-01-06", "P002", 60, 22],
    ["2023-01-07", "P003", 70, 24],
    ["2023-01-08", "P001", 50, 28],
    ["2023-01-09", "P002", 60, 17],
    ["2023-01-10", "P003", 70, 29],
    ["2023-02-11", "P001", 50, 23],
    ["2023-02-12", "P002", 60, 19],
```

```
    ["2023-02-13", "P001", 50, 21],
    ["2023-02-14", "P003", 70, 31],
    ["2023-03-15", "P001", 50, 26],
    ["2023-03-16", "P002", 60, 20],
    ["2023-03-17", "P003", 70, 33],
    ["2023-04-18", "P001", 50, 27],
    ["2023-04-19", "P002", 60, 18],
    ["2023-04-20", "P003", 70, 32],
    ["2023-04-21", "P001", 50, 22],
    ["2023-04-22", "P002", 60, 16],
    ["2023-04-23", "P003", 70, 34],
    ["2023-05-24", "P001", 50, 24],
    ["2023-05-25", "P002", 60, 21]
]

# Write data to sales.csv
with open('sales.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerows(data)

print("sales.csv has been created!")
```

```
sales.csv has been created!
```

Analyzing sales with Amazon Bedrock generated Python program

In [8]:
```
# Create the prompt
# Analyzing sales

prompt_data = """
You have a CSV, sales.csv, with columns:
- date (YYYY-MM-DD)
- product_id
- price
- units_sold

Create a python program to analyze the sales data from a CSV file. The program shou

- Total revenue for the year
- The product with the highest revenue
- The date with the highest revenue
- Visualize monthly sales using a bar chart

Ensure the code is syntactically correct, bug-free, optimized, not span multiple li
Do not use pandas library for the solution.
"""
```

Let's use the Anthropic Claude 3 Sonnet model.

In [9]:
```
body = json.dumps({
    "anthropic_version": "bedrock-2023-05-31",
    "max_tokens": 4096,
    "temperature": 0.1,
    "top_k":250,
    "top_p":0.99,
```

```
    "messages": [
        {
            "role": "user",
            "content": [{"type": "text", "text": prompt_data}]
        }
    ],
})
```

Invoke the Anthropic Claude 3 Sonnet model to generate the code:

In [10]:
```
from IPython.display import clear_output, display, display_markdown, Markdown
modelId = "anthropic.claude-3-sonnet-20240229-v1:0"
accept = 'application/json'
contentType = 'application/json'

response = boto3_bedrock.invoke_model(body=body, modelId=modelId, accept=accept, co
response_body = json.loads(response.get('body').read())

display_markdown(Markdown(print(response_body["content"][0]["text"], end='')))
```

```
import csv
from collections import defaultdict
from datetime import datetime
import matplotlib.pyplot as plt

def parse_date(date_str):
    return datetime.strptime(date_str, '%Y-%m-%d').date()

def analyze_sales(file_path):
    revenue_by_date = defaultdict(int)
    revenue_by_product = defaultdict(int)
    total_revenue = 0

    with open(file_path, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            date = parse_date(row['date'])
            product_id = row['product_id']
            price = float(row['price'])
            units_sold = int(row['units_sold'])

            revenue = price * units_sold
            total_revenue += revenue
            revenue_by_date[date] += revenue
            revenue_by_product[product_id] += revenue

    highest_revenue_date = max(revenue_by_date, key=revenue_by_date.get)
    highest_revenue_product = max(revenue_by_product, key=revenue_by_product.get)

    print(f"Total revenue for the year: ${total_revenue:.2f}")
    print(f"Product with the highest revenue: {highest_revenue_product}")
    print(f"Date with the highest revenue: {highest_revenue_date}")

    visualize_monthly_sales(revenue_by_date)

def visualize_monthly_sales(revenue_by_date):
    monthly_revenue = defaultdict(int)
    for date, revenue in revenue_by_date.items():
        monthly_revenue[date.month] += revenue

    months = sorted(monthly_revenue.keys())
    revenues = [monthly_revenue[month] for month in months]

    plt.figure(figsize=(10, 6))
    plt.bar(months, revenues)
    plt.xlabel('Month')
    plt.ylabel('Revenue')
    plt.title('Monthly Sales')
    plt.xticks(months)
    plt.show()

analyze_sales('sales.csv')
```

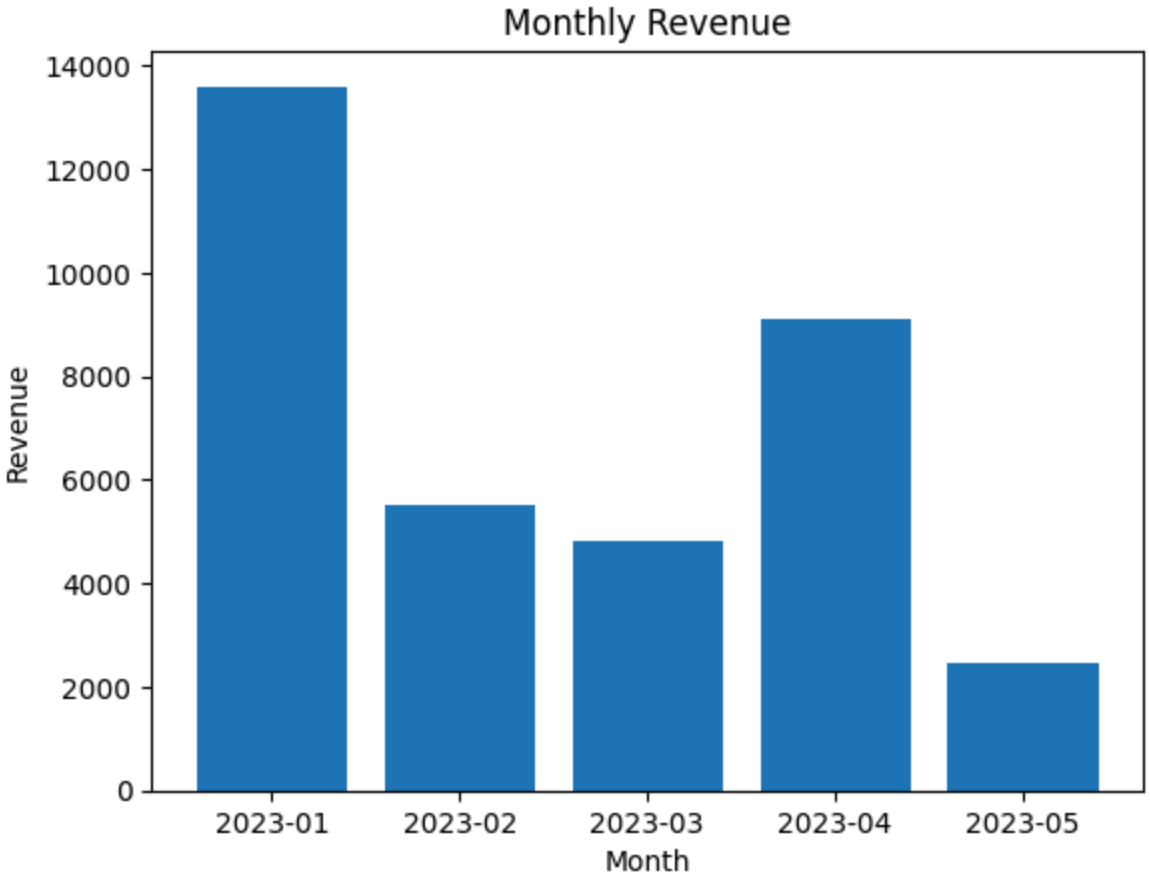
(Optional) Execute the Bedrock generated code for validation. Go to text editor to copy the generated code as printed output can be trucncated. Replace the code in below cell.

In [13]:
```
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /home/studio-lab-user/.conda/envs/defau
lt/lib/python3.9/site-packages (3.9.2)
Requirement already satisfied: fonttools>=4.22.0 in /home/studio-lab-user/.conda/env
s/default/lib/python3.9/site-packages (from matplotlib) (4.55.0)
Requirement already satisfied: pyparsing>=2.3.1 in /home/studio-lab-user/.conda/env
s/default/lib/python3.9/site-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /home/studio-lab-user/.conda/
envs/default/lib/python3.9/site-packages (from matplotlib) (2.9.0)
Requirement already satisfied: importlib-resources>=3.2.0 in /home/studio-lab-user/.
conda/envs/default/lib/python3.9/site-packages (from matplotlib) (6.4.0)
Requirement already satisfied: contourpy>=1.0.1 in /home/studio-lab-user/.conda/env
s/default/lib/python3.9/site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /home/studio-lab-user/.conda/env
s/default/lib/python3.9/site-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /home/studio-lab-user/.conda/envs/
default/lib/python3.9/site-packages (from matplotlib) (24.0)
Requirement already satisfied: cycler>=0.10 in /home/studio-lab-user/.conda/envs/def
ault/lib/python3.9/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: pillow>=8 in /home/studio-lab-user/.conda/envs/defaul
t/lib/python3.9/site-packages (from matplotlib) (11.0.0)
Requirement already satisfied: numpy>=1.23 in /home/studio-lab-user/.conda/envs/defa
ult/lib/python3.9/site-packages (from matplotlib) (2.0.2)
Requirement already satisfied: zipp>=3.1.0 in /home/studio-lab-user/.conda/envs/defa
ult/lib/python3.9/site-packages (from importlib-resources>=3.2.0->matplotlib) (3.17.
0)
Requirement already satisfied: six>=1.5 in /home/studio-lab-user/.conda/envs/defaul
t/lib/python3.9/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

In [14]:
```
# Sample Generated Python Code ( Generated with Amazon Bedrock in previous step)

import csv
from collections import defaultdict
import matplotlib.pyplot as plt

revenue = 0
monthly_revenue = defaultdict(int)
product_revenue = defaultdict(int)
max_revenue = 0
max_revenue_date = ''
max_revenue_product = ''

with open('sales.csv') as f:
    reader = csv.reader(f)
    next(reader)
    for row in reader:
        date = row[0]
        product = row[1]
        price = float(row[2])
        units = int(row[3])

        revenue += price * units
        product_revenue[product] += price * units
        monthly_revenue[date[:7]] += price * units
```

```
        if revenue > max_revenue:
            max_revenue = revenue
            max_revenue_date = date
            max_revenue_product = product

months = list(monthly_revenue.keys())
values = list(monthly_revenue.values())

plt.bar(months, values)
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.title('Monthly Revenue')
plt.show()

print('Total Revenue:', revenue)
print('Product with max revenue:', max_revenue_product)
print('Date with max revenue:', max_revenue_date)
```




```
Total Revenue: 35490.0
Product with max revenue: P002
Date with max revenue: 2023-05-25
```

Use case 2 - SQL query generation

In this section we show you how to use a LLM to generate SQL queries to analyze Sales data. We will use Bedrock's Claude 3 Sonnet model using the Boto3 API.

The prompt used in this example is called a zero-shot prompt because we are not providing any examples of text other than the prompt.

Pattern

We will simply provide the Amazon Bedrock API with an input consisting of a task, an instruction and an input for the model to generate an output without providing any additional examples. The purpose here is to demonstrate how the powerful LLMs easily understand the task at hand and generate compelling outputs.

Use case

Let's take the use case to generate SQL queries to analyze sales data, focusing on top products and average monthly sales.

Persona

Maya is a business analyst, at AnyCompany primarily focusing on sales and inventory data. She is transitioning from Speadsheet analysis to data-driven analysis and want to use SQL to fetch specific data points effectively. She wants to use LLMs to generate SQL queries for her analysis.

Implementation

To fulfill this use case, in this notebook we will show how to generate SQL queries. We will use the Anthropic Claude 3 model using the Amazon Bedrock API with Boto3 client.

Generate SQL Query

Following on the use case explained above, let's prepare an input for the Amazon Bedrock service to generate some SQL queries.

In [15]:
```
# create the prompt to generate SQL query
prompt_data = """

Human: AnyCompany has a database with a table named sales_data containing sales rec
- date (YYYY-MM-DD)
- product_id
- price
- units_sold

Can you generate SQL queries for the below:
- Identify the top 5 best selling products by total sales for the year 2023
- Calculate the average of total monthly sales for the year 2023
```

```
Assistant:
"""
```

Let's use the Claude 3 Sonnet model:

In [16]:
```
body = json.dumps({
    "anthropic_version": "bedrock-2023-05-31",
    "max_tokens": 4096,
    "temperature": 0.1,
    "top_k":250,
    "top_p":0.99,
    "messages": [
        {
            "role": "user",
            "content": [{"type": "text", "text": prompt_data}]
        }
    ],
})
```

In [17]:
```
from IPython.display import clear_output, display, display_markdown, Markdown

modelId = "anthropic.claude-3-sonnet-20240229-v1:0"
accept = 'application/json'
contentType = 'application/json'

response = boto3_bedrock.invoke_model(body=body, modelId=modelId, accept=accept, co
response_body = json.loads(response.get('body').read())

display_markdown(Markdown(print(response_body["content"][0]["text"], end='')))
```

Sure, here are the SQL queries for the given requirements:

1. Identify the top 5 best selling products by total sales for the year 2023:

```sql
SELECT
    product_id,
    SUM(units_sold * price) AS total_sales
FROM
    sales_data
WHERE
    date BETWEEN '2023-01-01' AND '2023-12-31'
GROUP BY
    product_id
ORDER BY
    total_sales DESC
LIMIT 5;
```

This query calculates the total sales for each product by multiplying the units sold with the price, and then sums it up for the year 2023. It groups the results by `product_id`, orders them in descending order of `total_sales`, and limits the output to the top 5 rows.

2. Calculate the average of total monthly sales for the year 2023:

```sql
SELECT
    ROUND(AVG(monthly_sales), 2) AS avg_monthly_sales
FROM
    (SELECT
         DATE_FORMAT(date, '%Y-%m') AS month_year,
         SUM(units_sold * price) AS monthly_sales
     FROM
         sales_data
     WHERE
         date BETWEEN '2023-01-01' AND '2023-12-31'
     GROUP BY
         month_year) AS monthly_sales_subquery;
```

This query first calculates the total monthly sales by summing up the product of `units_sold` and `price` for each month in 2023, using a subquery. It then calculates the average of these monthly sales values using the `AVG` function and rounds the result to 2 decimal places.

Note: The `DATE_FORMAT` function is used to extract the month and year from the `date` column in the subquery. The specific function and its syntax may vary depending on the database system you're using (e.g., MySQL, PostgreSQL, Oracle, etc.).

Conclusion

You have now experimented with using `boto3` SDK which provides a vanilla exposure to Amazon Bedrock API. Using this API you generate a python program to analyze and visualize

given sales data, and generate SQL statements based on an input task and schema.

Take aways

- Adapt this notebook to experiment with different models available through Amazon Bedrock such as Amazon Titan and AI21 Labs Jurassic models!