BACKEND TASK

```
[ ]  # models.py
     !pip install Django
     import django
     from django.db import models
     from django.contrib.auth.models import User

     class Alert(models.Model):
         user = models.ForeignKey(User, on_delete=models.CASCADE)
         target_price = models.DecimalField(max_digits=10, decimal_places=2)
         cryptocurrency = models.CharField(max_length=50)
         created_at = models.DateTimeField(auto_now_add=True)
         status = models.CharField(max_length=20, default='created')

         def __str__(self):
             return f"{self.user.username}'s alert for {self.cryptocurrency} at {self.target_price}"
```

```
[ ]  import requests
     response = requests.get("https://api.coingecko.com/api/v3/coins/markets?vs_currency=USD&order=ma")
```

```
[ ]  status_code = response.status_code

     # Get the response content as text
     text = response.text

     # Get the response content as JSON
     json_data = response.json()
```

```
[ ]  # serializers.py

     from rest_framework import serializers
     from .models import Alert

     class AlertSerializer(serializers.ModelSerializer):
         class Meta:
             model = Alert
             fields = '__all__'
```

```python
# views.py

from rest_framework import generics
from rest_framework.permissions import IsAuthenticated
from .models import Alert
from .serializers import AlertSerializer

class AlertCreateView(generics.CreateAPIView):
    permission_classes = [IsAuthenticated]
    serializer_class = AlertSerializer

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

class AlertDeleteView(generics.DestroyAPIView):
    permission_classes = [IsAuthenticated]
    queryset = Alert.objects.all()
    serializer_class = AlertSerializer

class AlertListView(generics.ListAPIView):
    permission_classes = [IsAuthenticated]
    serializer_class = AlertSerializer

    def get_queryset(self):
        status = self.request.query_params.get('status', None)
        if status:
            return Alert.objects.filter(user=self.request.user, status=status)
        return Alert.objects.filter(user=self.request.user)
```

```python
import smtplib
from email.mime.text import MIMEText

# Replace 'your_email@gmail.com' and 'your_password' with your Gmail credentials
sender_email = 'your_email@gmail.com'
password = 'your_password'

# Replace 'recipient@example.com' with the recipient's email address
recipient_email = 'recipient@example.com'

# Create a message
message = MIMEText('This is a test email.')
message['Subject'] = 'Test Email'
message['From'] = sender_email
message['To'] = recipient_email

# Connect to Gmail's SMTP server
with smtplib.SMTP('smtp.gmail.com', 587) as server:
    # Start TLS encryption
    server.starttls()

    # Login to your Gmail account
    server.login(sender_email, password)

    # Send the email
    server.sendmail(sender_email, recipient_email, message.as_string())
```

INFRASTRUCTURE

```
import pandas as pd

# Load the dataset from the CSV file
df = pd.read_csv('/content/orders.csv')

# Convert the 'order_date' column to datetime format
df['order_date'] = pd.to_datetime(df['order_date'])

# Task 1: Compute total revenue by month
df['month'] = df['order_date'].dt.to_period('M')
total_revenue_by_month = df.groupby('month')['product_price'].sum()

# Task 2: Compute total revenue by product
total_revenue_by_product = df.groupby('product_name')['product_price'].sum()

# Task 3: Compute total revenue by customer
total_revenue_by_customer = df.groupby('customer_id')['product_price'].sum()

# Task 4: Identify the top 10 customers by revenue
top_10_customers = total_revenue_by_customer.nlargest(10)

print("Total Revenue by Month:")
print(total_revenue_by_month)
print("\nTotal Revenue by Product:")
print(total_revenue_by_product)
print("\nTotal Revenue by Customer:")
print(total_revenue_by_customer)
print("\nTop 10 Customers by Revenue:")
print(top_10_customers)
```

TEST CASES:

```
import unittest
import pandas as pd
from orders_analysis import compute_total_revenue_by_month,
compute_total_revenue_by_product, compute_total_revenue_by_customer,
identify_top_10_customers

class TestOrdersAnalysis(unittest.TestCase):
```

```
    def setUp(self):
        # Creating a sample DataFrame for testing
        data = {
            'order_id': [1, 2, 3, 4, 5],
            'customer_id': [101, 102, 101, 103, 102],
            'order_date': ['2022-01-01', '2022-01-15', '2022-02-01', '2022-02-10', '2022-03-01'],
            'product_id': [1, 2, 1, 3, 2],
            'product_name': ['ProductA', 'ProductB', 'ProductA', 'ProductC', 'ProductB'],
            'product_price': [10.0, 20.0, 10.0, 15.0, 20.0],
            'quantity': [2, 1, 3, 2, 1]
        }
        self.df = pd.DataFrame(data)
        self.df['order_date'] = pd.to_datetime(self.df['order_date'])

    def test_compute_total_revenue_by_month(self):
        result = compute_total_revenue_by_month(self.df)
        expected_result = pd.Series([50.0, 55.0], index=pd.PeriodIndex(['2022-01', '2022-02'],
freq='M'))
        pd.testing.assert_series_equal(result, expected_result)

    def test_compute_total_revenue_by_product(self):
        result = compute_total_revenue_by_product(self.df)
        expected_result = pd.Series([30.0, 40.0, 15.0], index=['ProductA', 'ProductB', 'ProductC'])
        pd.testing.assert_series_equal(result, expected_result)

    def test_compute_total_revenue_by_customer(self):
        result = compute_total_revenue_by_customer(self.df)
        expected_result = pd.Series([30.0, 40.0, 15.0], index=[101, 102, 103])
        pd.testing.assert_series_equal(result, expected_result)

    def test_identify_top_10_customers(self):
        result = identify_top_10_customers(self.df)
        expected_result = pd.Series([40.0, 30.0, 15.0], index=[102, 101, 103])
        pd.testing.assert_series_equal(result, expected_result)

if __name__ == '__main__':
    unittest.main()
```