Detailed Explanation of the Code (Line by Line)

Introduction

This project is a powerful example of how **multi-modal AI models**—capable of interpreting both images and text—can be applied to real-world tasks like **chart interpretation and report generation**. Built using the **SmolVLM-Instruct** model from Hugging Face, the system is able to **analyze uploaded chart images** (such as bar, pie, or line charts), extract meaningful patterns or trends, and produce a **clear, human-like written explanation**.

What makes this project unique is that it not only interprets data visualizations like a professional analyst but also allows users to **export the entire analysis as a polished PDF report**. This can be especially useful in **educational, business, and research settings**, where quick data interpretation and documentation are vital.

The application is built using **Gradio** for the user interface, **PyTorch + Hugging Face Transformers** for the machine learning backend, and **FPDF** to create dynamic PDF documents. It demonstrates how combining pre-trained models with simple Python libraries can unlock powerful, automated workflows.

Whether you're a **data analyst, educator, student, or business professional**, this tool enables quick and interpretable insights from visual data, making it a valuable companion in any data-driven workflow.

---

1. Importing Required Libraries

```
import gradio as gr
from transformers import AutoProcessor, AutoModelForVision2Seq
from PIL import Image
from fpdf import FPDF
import torch
import textwrap
import re
import os
```

Purpose of Each Library:

- **gradio**: Builds a user interface for interaction.
- **transformers**: Loads and runs pre-trained Hugging Face models.
- **PIL (Image)**: Handles image processing.
- **fpdf**: Used to create PDF reports.
- **torch**: Manages device selection (CPU/GPU) and tensor operations.
- **textwrap**: Wraps long text lines for formatting.
- **re**: Handles regular expressions (used for text cleanup).
- **os**: Performs file system operations (like deleting temp files).

2. Selecting the Execution Device

```
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

- Automatically selects **GPU** (`cuda`) if available, otherwise falls back to **CPU**.

---

3. Loading the Model and Processor

```
processor = AutoProcessor.from_pretrained("HuggingFaceTB/SmolVLM-Instruct")
model = AutoModelForVision2Seq.from_pretrained(
    "HuggingFaceTB/SmolVLM-Instruct",
    torch_dtype=torch.bfloat16 if DEVICE == "cuda" else torch.float32,
).to(DEVICE)
```

- Loads the **SmolVLM-Instruct** model and its associated **processor**.
- Sets the model's precision based on the device (bfloat16 for GPU, float32 for CPU).
- Moves the model to the selected device.

---

4. Default Prompt for Chart Analysis

```
DEFAULT_PROMPT = "Please analyze this chart like a professional data analyst and explain the insights clearly."
```

- This is the **instruction** given to the model for every chart input.

---

5. Main Function: Analyze Image and Generate PDF

```
def analyze_chart_and_generate_pdf(image: Image.Image):
```

Takes a chart image as input, uses the model to generate insights, and produces a PDF report.

5.1 Prepare Message for the Model

```
messages = [
    {
        "role": "user",
        "content": [
            {"type": "image"},
            {"type": "text", "text": DEFAULT_PROMPT}
        ]
```

```
    },
]
```

- Simulates a user message combining the **image** and the **text prompt**.

5.2 Convert Input to Model Format

```
chat_prompt = processor.apply_chat_template(messages,
add_generation_prompt=True)
inputs = processor(text=chat_prompt, images=[image],
return_tensors="pt").to(DEVICE)
```

- Formats the message into a chat-style prompt using the processor.
- Converts the image + text into PyTorch tensors and moves them to the appropriate device.

5.3 Generate Output from the Model

```
generated_ids = model.generate(**inputs, max_new_tokens=500)
raw_output = processor.batch_decode(generated_ids,
skip_special_tokens=True)[0]
```

- Generates a response from the model based on the inputs.
- Decodes the output tensor into a human-readable string.

5.4 Clean Up the Text

```
generated_text = re.split(r"Assistant:\s*", raw_output, maxsplit=1)[-
1].strip()
cleaned_text = generated_text.encode("latin-1",
errors="ignore").decode("latin-1")
```

- Removes any unwanted prefixes like "Assistant:".
- Cleans up special characters for PDF compatibility.

5.5 Save Temporary Image File

```
image_path = "/tmp/temp_chart_image.png"
image.save(image_path)
```

- Saves the uploaded image temporarily for use in the PDF.

5.6 Generate the PDF Report

```
pdf_path = "/tmp/chart_analysis_result.pdf"
pdf = FPDF()
pdf.add_page()
```

- Initializes a new PDF document and adds a page.

Add Report Title

```
pdf.set_font("Arial", 'B', 18)
pdf.cell(0, 15, "Chart Analysis Report", ln=True, align='C')
```

Add Chart Image Section

```
pdf.set_font("Arial", 'B', 14)
pdf.cell(0, 10, "Uploaded Chart", ln=True, align='C')
```

- Adds section titles.

Insert Image to PDF

```
img_width = 180
img_ratio = image.height / image.width
img_height = img_width * img_ratio
pdf.image(image_path, x=(210 - img_width) / 2, y=35, w=img_width,
h=img_height)
```

- Calculates image height to maintain aspect ratio.
- Centers the image horizontally.

Add AI Insights Title

```
pdf.set_xy(10, y_after_image)
pdf.set_font("Arial", 'B', 14)
pdf.cell(0, 10, "AI-Generated Insights", ln=True)
```

Add Wrapped Insight Text

```
pdf.set_font("Arial", size=12)
wrapped_lines = textwrap.wrap(cleaned_text, width=90)
for line in wrapped_lines:
    pdf.multi_cell(0, 8, line, align='L')
```

- Wraps the insight text so it fits nicely within the PDF page width.

5.7 Save PDF and Clean Up Temp Image

```
pdf.output(pdf_path)
os.remove(image_path)
```

- Saves the final PDF and deletes the temporary chart image.

5.8 Return Output

```
return generated_text, pdf_path
```

- Returns the analysis as text and the path to the generated PDF.

---

6. Clear Function for UI Reset

```
def clear_fields():
    return None, "", None
```

- Resets the image input, text output, and PDF file in the UI.

---

7. Gradio User Interface

```
with gr.Blocks() as demo:
```

- Creates a block-based Gradio interface.

7.1 Collapsible Info Section

```
with gr.Accordion("□ About This Project", open=False):
    gr.Markdown("""..."""")
```

- Contains project info, features, and usage suggestions in a collapsible accordion.

7.2 Input/Output Elements

```
image_input = gr.Image(type="pil", label="Upload Chart Image")
analyze_button = gr.Button("Analyze Chart")
clear_button = gr.Button("Clear")
output_text = gr.Textbox(label="Analysis Result", lines=12,
interactive=False)
pdf_output = gr.File(label="Download PDF")
```

- Components for user interaction:
  - Upload image
  - Run analysis
  - Clear results
  - Show text output
  - Offer PDF download

7.3 Connect Buttons to Functions

```
analyze_button.click(
    fn=analyze_chart_and_generate_pdf,
    inputs=[image_input],
    outputs=[output_text, pdf_output]
)

clear_button.click(
    fn=clear_fields,
    inputs=[],
    outputs=[image_input, output_text, pdf_output]
)
```

- Defines what happens when the **Analyze** or **Clear** button is clicked.

7.4 Launch the App

```
demo.launch()
```

- Starts the Gradio interface in the browser.

Summary

Functionality	Description
Upload Chart	Users can upload an image of a chart.
AI Analysis	SmolVLM model generates a detailed explanation.
PDF Report	Includes both chart and explanation.
Clear Button	Resets all fields in the UI.
Gradio Interface	Web-based interface for easy access and use.

Conclusion

This application is a seamless integration of **vision-language modeling**, **natural language generation**, and **interactive UI design**. It automates the once-manual task of reading and explaining charts by leveraging the latest advances in artificial intelligence.

Key Benefits:

- **Saves time** by automatically analyzing complex charts.
- **Reduces human error** in interpreting visual data.
- **Creates shareable documentation** in the form of PDF reports.
- **User-friendly** interface with no coding required.

Real-World Applications:

- **Business reports** and executive summaries.
- **Academic research** and educational assignments.
- **Training materials** and AI-assisted learning.

In short, this project bridges the gap between raw data visualization and understandable insights—empowering users to **extract meaning from data without needing deep analytical expertise**. With further improvements (like multilingual support or integration into dashboards), it can become an indispensable assistant in data-centric roles.