Install Stuff

```
[ ] %pip install --upgrade --quiet google-cloud-aiplatform google-cloud-aiplatform[evaluation]
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 62.4 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 118.4/118.4 kB 14.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 739.1/739.1 kB 56.0 MB/s eta 0:00:00
```

import & configuration code below

```
[1] import datetime
    import nest_asyncio
    import pandas as pd
    from IPython.display import display, Markdown, HTML

    import vertexai
    from vertexai.generative_models import GenerativeModel

    pd.set_option('display.max_colwidth', None)
```

Paste the following into a new code block & run it to initialize Vertex AI.

```
[2] PROJECT_ID = "qwiklabs-gcp-04-f4e926e31c52"
    LOCATION = "us-central1"
    import vertexai
    vertexai.init(project=PROJECT_ID, location=LOCATION)
```

Explore example data and use it to generate content

Run this line in a new code cell (by pasting the code and hitting Shift + Return) to download some example data.

```
[3] !gcloud storage cp gs://partner-genai-bucket/genai065/apartment_table.csv .
```

```
Copying gs://partner-genai-bucket/genai065/apartment_table.csv to file://./apartment_table.csv
```

Load it into a Pandas DataFrame, and view the first few rows:

You should see a table like this, presenting data about apartments in New York City, including how many bedrooms they have, how large they are (measured in square feet, identified in this dataset as “Sqft”), whether the building has an elevator, and other attributes of an apartment.

```
[4] apartment_df = pd.read_csv("apartment_table.csv")
    apartment_df.head()
```

	Address	Unit	Sqft	Bedrooms	Elevator	Washer & Dryer in Unit	Pets Allowed	Notable features
0	123 West 14th Street, New York, NY 10014	2E	550	2	yes	no	yes	doorman, pool in the building, shared roof deck with grills
1	456 East 57th Street, New York, NY 10022	1A	789	1	no	yes	no	bike room, package service
2	789 Broadway, New York, NY 10003	C	999	3	yes	yes	yes	excellent laundry room, great city views
3	1011 5th Avenue, New York, NY 10028	30	1024	2	no	no	yes	great view of Central Park, high ceilings
4	2222 Park Avenue, New York, NY 10017	4F	1234	1	no	yes	no	right next to soccer fields at the park

Notice in particular the “Notable features” column, in which unstructured notes from a real estate agent have been captured to describe other selling points of each apartment. You will have a model generate apartment listing descriptions based on these structured and unstructured fields, and then you will measure how well the model sticks to the information in these fields when generating listing descriptions.

Restructure the data into a list of records with keys identified for each record, and view the first record.

```
[5] apartment_records = apartment_df.to_dict(orient='records')
    apartment_records[0]
```

```
{'Address': '123 West 14th Street, New York, NY 10014',
 'Unit': '2E',
 'Sqft': 550,
 'Bedrooms': 2,
 'Elevator': 'yes',
 'Washer & Dryer in Unit': 'no',
 'Pets Allowed': 'yes',
 'Notable features': 'doorman, pool in the building, shared roof deck with grills'}
```

Now it’s time to generate some text. Instantiate a generative model, define a prompt with some instructions for it, and generate content based on the example data:

The model will have generated some text like what you see below. You can compare the content to the record dictionary you saw above to see that features like the number of bedrooms and the square footage are included. Multiple details from the unstructured notes (the doorman, the pool, the grills on the roof deck) are also included.

```
[7] model = GenerativeModel(
        "gemini-2.0-flash-001",
```

```
  generation_config={
      "temperature": 0,
      "top_p": 0.4,
  },
)

prompt = "Write a one paragraph apartment listing to promote this apartment. Make it sound amazing: "

# View the response using Markdown to format it nicely for notebook viewing
Markdown(model.generate_content(prompt + str(apartment_records[0])).text)
```

Live your best NYC life in this charming 2-bedroom apartment at 123 West 14th Street! Unit 2E offers a comfortable 550 sqft of living space in a prime location. Enjoy the convenience of an elevator, the peace of mind of a doorman, and the luxury of a building pool. Plus, your furry friends are welcome! Soak up the sun and city views on the shared roof deck, complete with grills for unforgettable summer evenings. Don't miss out on this incredible opportunity to experience the ultimate urban lifestyle!

You might need to take caution, however, as the description may also include some information that we didn't provide. In the output above, the model suggests that there is an "abundance of natural light" and "modern finishes", but you didn't tell the model those things about this apartment, so that may not be true.

You'll set up an evaluation to attempt to quantify how grounded, or using only provided information, are the model's responses.

Configure and trigger a model-based evaluation

1. View the prompt templates for model-based evaluation in the documentation. Select the Pointwise text use case template for "Groundedness" to read the prompt.
2. Notice that the model will score a response with a 1 if it appears to be fully grounded, or a 0 otherwise.

Also notice that the prompt contains a couple of placeholders in curly braces: {prompt} and {response}. You must provide at least the prompt data in your evaluation dataset, but you can choose whether to provide the response data yourself or let the Evaluation Task run the prompts through a model you provide to generate the responses before evaluating them.

Additionally, pairwise metrics (which compare two models) require the response of the other model you are comparing against, which must be provided in a field named baseline_model_response. For chat evaluations, the chat history will also need to be provided as history.

3. To create an evaluation dataset, create the prompt for each example, which will consist of the prompt instructions you defined earlier and the context data for each apartment.

```
[8] # Context is the supplemental information you provide the
    # model, usually specific to a given query or example,
    # that it needs to fulfill your instructions.
    # In this case, the context is each apartment record.
    contexts = [str(record) for record in apartment_records]
    # The full prompt combines the prompt instructions you
    # created earlier with the context for each apartment.
    full_prompts = [prompt + str(record) for record in apartment_records]

    print(full_prompts[0])
```

```
Write a one paragraph apartment listing to promote this apartment. Make it sound amazing: {'Address': '123 West 14th Street, New York, NY 10014', 'Unit': '2E', 'Sqft': 55
```

Instead of generating the responses, yourself, create an evaluation dataset of just the prompts, and the evaluation service will generate responses for you as part of the evaluation task.

Note:Because of Qwiklabs quota limitations, you will limit your evaluation dataset to 5 examples. But the best practice recommendation would be to include around 100 examples covering the types of inputs your model might see.

```
[9] eval_dataset = pd.DataFrame({
        "prompt": full_prompts[0:5],
    })
```

Now you'll explore some of the classes available for evaluation. Run the following imports and print the list of available MetricPromptTemplateExamples:

```
[10] from vertexai.evaluation import (
         MetricPromptTemplateExamples,
         EvalTask,
         PairwiseMetric,
         PairwiseMetricPromptTemplate,
         PointwiseMetric,
         PointwiseMetricPromptTemplate,
     )

     MetricPromptTemplateExamples.list_example_metric_names()
```

BROWSE PRE-BUILT METRICS

```
['coherence',
 'fluency',
 'safety',
 'groundedness',
 'instruction_following',
 'verbosity',
 'text_quality',
 'summarization_quality',
 'question_answering_quality',
 'multi_turn_chat_quality',
 'multi_turn_safety',
 'pairwise_coherence',
 'pairwise_fluency',
 'pairwise_safety',
 'pairwise_groundedness',
 'pairwise_instruction_following',
 'pairwise_verbosity',
 'pairwise_text_quality',
```

```
    'pairwise_summarization_quality',
    'pairwise_question_answering_quality',
    'pairwise_multi_turn_chat_quality',
    'pairwise_multi_turn_safety']
```

Instead of using the documentation to review the prompt as you did earlier, you can view the criteria and ratings for each metric within your code like so:

```
[11] print(MetricPromptTemplateExamples.get_prompt_template('groundedness'))
```

```
# Instruction
You are an expert evaluator. Your task is to evaluate the quality of the responses generated by AI models.
We will provide you with the user input and an AI-generated response.
You should first read the user input carefully for analyzing the task, and then evaluate the quality of the responses based on the criteria provided in the Evaluation sec
You will assign the response a rating following the Rating Rubric and Evaluation Steps. Give step by step explanations for your rating, and only choose ratings from the R


# Evaluation
## Metric Definition
You will be assessing groundedness, which measures the ability to provide or reference information included only in the user prompt.

## Criteria
Groundedness: The response contains information included only in the user prompt. The response does not reference any outside information.

## Rating Rubric
1: (Fully grounded). All aspects of the response are attributable to the context.
0: (Not fully grounded). The entire response or a portion of the response is not attributable to the context provided by the user prompt.

## Evaluation Steps
STEP 1: Assess the response in aspects of Groundedness. Identify any information in the response not present in the prompt and provide assessment according to the criteri
STEP 2: Score based on the rating rubric. Give a brief rationale to explain your evaluation considering Groundedness.


# User Inputs and AI-generated Response
## User Inputs
### Prompt
{prompt}

## AI-generated Response
{response}
```

Instantiate an EvalTask by associating your dataset & selected metric. You can also provide an experiment name to track your evaluations in Vertex AI Experiments.

```
[12] eval_task = EvalTask(
       dataset=eval_dataset,
       metrics=[MetricPromptTemplateExamples.Pointwise.GROUNDEDNESS],
       experiment="apartment-listing-generation",
     )
```

Run the evaluate() method on the task by giving it a unique run name. By passing the model you would like to evaluate, the EvalTask can generate the responses needed to complete the evaluation dataset.

```
[13] run_ts = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
     eval_result = eval_task.evaluate(
       model=model,
       experiment_run_name=f"apt-gen-{run_ts}"
       )

     # You might want to keep track of your results in a list
     # which you will use to plot your results later on
     eval_results_to_compare = []
     eval_results_to_compare.append(eval_result)
```

VIEW EXPERIMENT

VIEW EXPERIMENT RUN

```
INFO:vertexai.evaluation.eval_task:Logging Eval Experiment metadata: {'model_name': 'publishers/google/models/gemini-2.0-flash-001', 'temperature': 0, 'top_p': 0.4}
INFO:vertexai.evaluation._evaluation:Generating a total of 5 responses from Gemini model gemini-2.0-flash-001.
100%|██████████| 5/5 [00:01<00:00,  3.99it/s]
INFO:vertexai.evaluation._evaluation:All 5 responses are successfully generated from Gemini model gemini-2.0-flash-001.
INFO:vertexai.evaluation._evaluation:Multithreaded Batch Inference took: 1.2640455259999044 seconds.
INFO:vertexai.evaluation._evaluation:Computing metrics with a total of 5 Vertex Gen AI Evaluation Service API requests.
100%|██████████| 5/5 [00:04<00:00,  1.02it/s]
INFO:vertexai.evaluation._evaluation:All 5 metric requests are successfully computed.
INFO:vertexai.evaluation._evaluation:Evaluation Took:4.914633108999965 seconds
```

VIEW EVALUATION RESULTS

The evaluation service's results consist of three parts: a complete table of results (metrics_table), some summary metrics with averages and standard deviation of the metrics (summary_metrics), and some metadata related to the run. First look at the summary metrics:

```
[14] eval_result.summary_metrics
```

```
{'row_count': 5, 'groundedness/mean': np.float64(1.0), 'groundedness/std': 0.0}
```

Viewing the outputs, you'll see that the groundedness was likely not perfect. From the Groundedness prompt descriptions we saw above, we know the model scores a 1 for grounded or a 0 for grounded, so a perfect result would have returned a mean of 1.0 and a std (or standard deviation) of 0 suggesting no deviation from that mean.

Next, view the complete results table:

You should see a table like this containing your results. Look for examples where the groundedness/score is 0.0, meaning the results were not grounded. Read the response and then the groundedness/explanation to see why the model rated the response as not grounded. You may see something like:

```
[15] eval_result.metrics_table
```

	prompt	response	groundedness/explanation	groundedness/score
0	Write a one paragraph apartment listing to promote this apartment. Make it sound amazing: {'Address': '123 West 14th Street, New York, NY 10014', 'Unit': '2E', 'Sqft': 550, 'Bedrooms': 2, 'Elevator': 'yes', 'Washer & Dryer in Unit': 'no', 'Pets Allowed': 'yes', 'Notable features': 'doorman, pool in the building, shared roof deck with grills'}	Live the quintessential New York City lifestyle in this charming 2-bedroom apartment at 123 West 14th Street! Unit 2E offers a comfortable 550 sqft of living space in a prime location. Enjoy the convenience of an elevator, the peace of mind of a doorman, and the luxury of a building pool. Pet owners rejoice, your furry friends are welcome here! Plus, soak up breathtaking city views and host unforgettable gatherings on the shared roof deck, complete with grills. Don't miss out on this incredible opportunity to experience the best of urban living!\n	The response only uses information provided in the prompt and does not include any outside information.	1.0
1	Write a one paragraph apartment listing to promote this apartment. Make it sound amazing: {'Address': '456 East 57th Street, New York, NY 10022', 'Unit': '1A', 'Sqft': 789, 'Bedrooms': 1, 'Elevator': 'no', 'Washer & Dryer in Unit': 'yes', 'Pets Allowed': 'no', 'Notable features': 'bike room, package service'}	Nestled in the heart of Manhattan's vibrant East Side at 456 East 57th Street, Unit 1A offers an unparalleled urban living experience. This charming 789 sqft one-bedroom apartment boasts the ultimate convenience with an in-unit washer and dryer, eliminating trips to the laundromat. Enjoy the ease of city living with package service and a dedicated bike room, perfect for storing your wheels after exploring the city. While furry friends will have to stay behind, you'll find peace and quiet in this meticulously maintained building. Experience the best of New York City right outside your door!\n	The response only contains information provided in the prompt and does not include any outside information.	1.0
2	Write a one paragraph apartment listing to promote this apartment. Make it sound amazing: {'Address': '789 Broadway, New York, NY 10003', 'Unit': 'C', 'Sqft': 999, 'Bedrooms': 3, 'Elevator': 'yes', 'Washer & Dryer in Unit': 'yes', 'Pets Allowed': 'yes', 'Notable features': 'excellent laundry room, great city views'}	Live the quintessential New York City dream in this stunning 3-bedroom apartment at 789 Broadway, Unit C! Boasting a generous 999 sqft of living space, this meticulously maintained residence offers breathtaking city views and the ultimate in convenience. Enjoy the ease of an in-unit washer and dryer, a modern elevator, and a truly excellent laundry room. Plus, bring your furry friends along – pets are welcome! Don't miss your chance to experience unparalleled comfort and style in the heart of it all.\n	The response only contains information from the prompt; it mentions the address, unit, square footage, number of bedrooms, elevator, in-unit washer and dryer, pet allowance, and notable features like the excellent laundry room and city views.	1.0
3	Write a one paragraph apartment listing to promote this apartment. Make it sound amazing: {'Address': '1011 5th Avenue, New York, NY 10028', 'Unit': '30', 'Sqft': 1024, 'Bedrooms': 2, 'Elevator': 'no', 'Washer & Dryer in Unit': 'no', 'Pets Allowed': 'yes', 'Notable features': 'great view of Central Park, high ceilings'}	Live in unparalleled luxury at 1011 5th Avenue, in unit 30! This stunning 2-bedroom apartment boasts 1024 sqft of living space with breathtaking, unobstructed views of Central Park from your windows. Enjoy the grandeur of high ceilings that amplify the natural light, creating an airy and sophisticated ambiance. While this unit doesn't have an elevator or in-unit washer and dryer, and pets are welcome, the unbeatable location and spectacular views more than compensate. Experience the best of New York City living in this exceptional residence!\n	The response only uses information provided in the prompt, and does not include any outside information.	1.0
4	Write a one paragraph apartment listing to promote this apartment. Make it sound amazing: {'Address': '2222 Park Avenue, New York, NY 10017', 'Unit': '4F', 'Sqft': 1234, 'Bedrooms': 1, 'Elevator': 'no', 'Washer & Dryer in Unit': 'yes', 'Pets Allowed': 'no', 'Notable features': 'right next to soccer fields at the park'}	Live the ultimate New York City dream in this stunning one-bedroom apartment at 2222 Park Avenue, Unit 4F! Boasting a generous 1234 square feet of living space and the unparalleled convenience of an in-unit washer and dryer, this residence offers both comfort and style. Imagine waking up and strolling directly to the park's soccer fields for a morning workout or leisurely afternoon game. While this pristine space is best suited for those without furry friends, the location and amenities make it an absolute must-see for anyone seeking a vibrant and convenient urban lifestyle. Please note that this unit is located in a walk-up building.\n	All information in the response is attributable to the context provided in the user prompt, including the address, unit number, square footage, number of bedrooms, in-unit washer and dryer, pet policy, and proximity to soccer fields, and the response does not include outside information.	1.0

Improve the prompt to see scores improve

In this task, you'll make a tweak to address the hallucination concerns and improve your application's groundedness score.

Improve the prompt to improve your groundedness score. With the new prompt, also create a new evaluation dataset.

```
[16] # This is a minor update, but should stop the model from
     # inventing as many details about each apartment.
     updated_prompt = "Write a one paragraph apartment listing highlighting the best known features of this apartment. Use only the details included in the following informat

     updated_full_prompts = [updated_prompt + str(record) for record in apartment_records]

     updated_eval_dataset = pd.DataFrame(
       {
           "prompt": updated_full_prompts[0:5]
       }
     )
```

Create a new EvalTask and run its evaluate() method to generate evaluations. Preview them in a table.

```
[17] updated_eval_task = EvalTask(
       dataset=updated_eval_dataset,
       metrics=[MetricPromptTemplateExamples.Pointwise.GROUNDEDNESS],
       experiment="apartment-listing-generation",
     )

     run_ts = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
     updated_result = updated_eval_task.evaluate(
       model=model,
       experiment_run_name=f"apt-gen-{run_ts}"
     )

     # Append the new result to your results
     eval_results_to_compare.append(updated_result)

     # Preview the summary
     print(updated_result.summary_metrics)
```

VIEW EXPERIMENT

VIEW EXPERIMENT RUN

```
INFO:vertexai.evaluation.eval_task:Logging Eval Experiment metadata: {'model_name': 'publishers/google/models/gemini-2.0-flash-001', 'temperature': 0, 'top_p': 0.4}
INFO:vertexai.evaluation._evaluation:Generating a total of 5 responses from Gemini model gemini-2.0-flash-001.
100%|██████████| 5/5 [00:01<00:00,  4.54it/s]
INFO:vertexai.evaluation._evaluation:All 5 responses are successfully generated from Gemini model gemini-2.0-flash-001.
INFO:vertexai.evaluation._evaluation:Multithreaded Batch Inference took: 1.1069973549999759 seconds.
INFO:vertexai.evaluation._evaluation:Computing metrics with a total of 5 Vertex Gen AI Evaluation Service API requests.
100%|██████████| 5/5 [00:01<00:00,  4.57it/s]
INFO:vertexai.evaluation._evaluation:All 5 metric requests are successfully computed.
INFO:vertexai.evaluation._evaluation:Evaluation Took:1.1016453570000522 seconds
```

VIEW EVALUATION RESULTS

```
{'row_count': 5, 'groundedness/mean': np.float64(1.0), 'groundedness/std': 0.0}
```

Your groundedness/mean score should now have increased.

Use the provided helper function to visualize your evaluation runs against each other. See whether your application improved on the metrics:

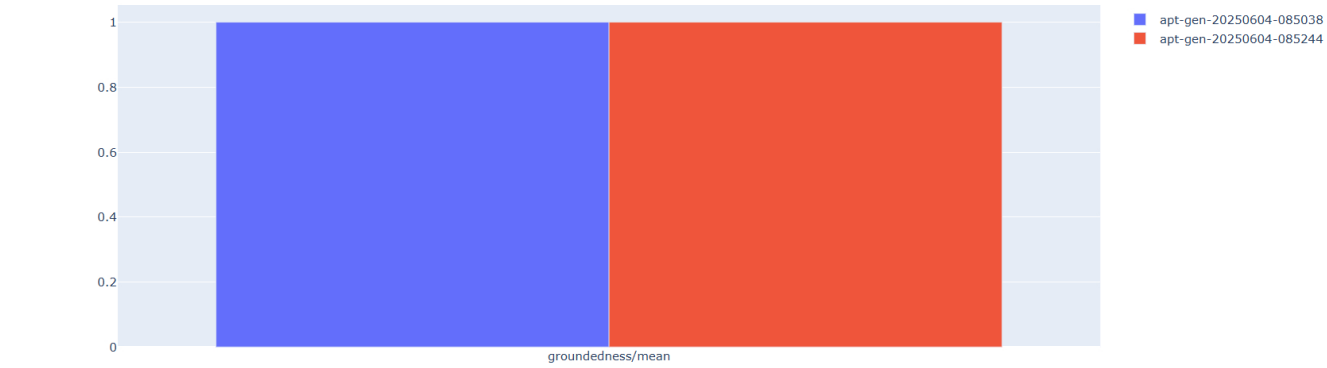
You should see that you've made a measurable improvement on groundedness!

```
[18] import plotly.graph_objects as go
     def plot_bar_plot(eval_results, metrics=None):
       fig = go.Figure()
       data = []
       for eval_result in eval_results:
           summary_metrics = eval_result.summary_metrics
           if metrics:
               summary_metrics = {
                   k: summary_metrics[k]
                   for k, v in summary_metrics.items()
                   if any(selected_metric in k for selected_metric in metrics)
               }


           data.append(
               go.Bar(
                   x=list(summary_metrics.keys()),
                   y=list(summary_metrics.values()),
                   name=eval_result.metadata["experiment_run"]
               )
           )
       fig = go.Figure(data=data)


       # Change the bar mode
       fig.update_layout(barmode="group")
       fig.show()


     plot_bar_plot(eval_results_to_compare, metrics=["groundedness/mean"])
```



Within the **Google Cloud Console**, navigate to **Vertex AI** and then Experiments within the left navigation bar. If you click on your experiment name and select a run name, you can select the **Metrics** and **Parameters** tabs to see that the Generative AI Evaluation Service has documented your runs.

```
[ ]
```