This GitHub repository contains all relevant info

Between Artificial and Human Intelligence

Anchoring Effect

Examining Anchoring Bias in Language Models

Overview

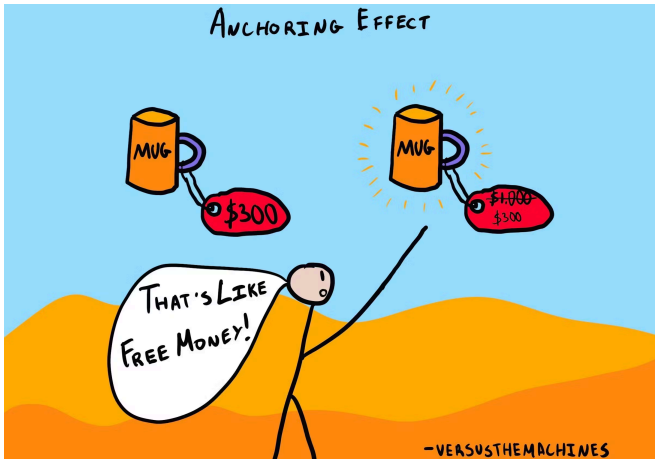
This project investigates whether large language models (LLMs) exhibit the cognitive bias known as the anchoring effect. Originally identified by Professors Amos Tversky and Daniel Kahneman, this bias influences human judgment such that initial information, even if incorrect or irrelevant, can significantly affect decision-making. This effect persists even in cases where the individual's awareness of the anchor's irrelevance. For instance, if someone claims that "the average price of an iPhone is $1," we intuitively know this to be extremely improbable based on our general knowledge of the market. However, this statement might still subconsciously influence us to lower our estimate of what an iPhone should cost, compared to our assessment in the absence of such an anchor.

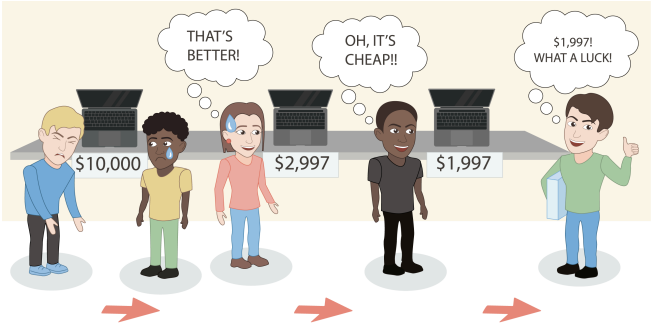
The Anchoring Effect

Anchoring bias occurs when an initial piece of information provided at the time of decision-making influences perceptions. For example, if one group is informed that Kennedy was 38 at the time of his assassination, and another group is told he was 55, the group exposed to the higher age will likely estimate his age at death to be greater. Tversky posited that the anchor establishes a mental baseline, which is then insufficiently adjusted. Kahneman suggested that the anchor primes associated thoughts, which then influence subsequent judgments.

Relevance to Language Models

This research aims to assess if LLMs, similar to humans, can be influenced by anchors in situations such as estimating product prices. Given that LLMs process inputs based on both the provided data and their extensive pre-training on human-generated text, they might also reflect human biases. This expectation is based on the understanding that their training data typically embodies these biases.






```
!pip install transformers --upgrade
!pip install -q -U accelerate bitsandbytes
```

```
import pandas as pd
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig
import bitsandbytes as bnb
from accelerate import Accelerator
import re
from scipy.stats import ttest_rel
from google.colab import drive
import numpy as np
import os
import random
```

```
# Constants

NUMBER_OF_EXAMPLES = 120
PRICE_RANGE = (0, 1000)

# DATA_FILE_PATH = '/content/drive/My Drive/between human cognition to intelliger
DATA_FILE_PATH = '/content/data.xlsx'
FEW_SHOT_FILE_PATH = '/content/few_shot_data.xlsx'
# RESULT_FILE_PREFIX = '/content/drive/My Drive/between human cognition to intell
RESULT_FILE_PREFIX = '/content/results.xlsx'
# ANALYSIS_FILE_PREFIX = '/content/drive/My Drive/between human cognition to inte
ANALYSIS_FILE_PREFIX = '/content/results_analysis.xlsx'
MODEL_NAME = 'mrm8488/mistral-7b-ft-h4-no_robots_instructions'

MODELS_NAMES = ['facebook/opt-350m',  'EleutherAI/gpt-j-6B', 'mrm8488/mistral-7b-
```

To re-establish access and interact with Google Drive, use the following command to authorize the session:

```
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Benchmark Development

The benchmark comprises over 100 examples designed to investigate how anchoring prices can influence the judgments of large language models (LLMs). These examples cover a variety of contexts where initial price information might act as an anchor, potentially affecting the model outputs.

Product	Anchor Price	Basic Prompt	Anchored Prompt	Guided Prompt
smartphone	165	A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 165 dollars. A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 165 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a smartphone is typically
smartphone	794	A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 794 dollars. A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 794 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a smartphone is typically
smartphone	502	A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 502 dollars. A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 502 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a smartphone is typically
laptop	105	A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 105 dollars. A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 105 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a laptop is typically
laptop	178	A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 178 dollars. A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 178 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a laptop is typically
laptop	71	A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 71 dollars. A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 71 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a laptop is typically
headphones	709	A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 709 dollars. A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 709 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a headphones is typically
headphones	160	A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 160 dollars. A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 160 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a headphones is typically
headphones	815	A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 815 dollars. A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 815 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a headphones is typically
bicycle	818	A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 818 dollars. A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 818 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a bicycle is typically
bicycle	742	A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 742 dollars. A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 742 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a bicycle is typically
bicycle	931	A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 931 dollars. A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 931 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a bicycle is typically
watch	298	A fair market price for a watch is typically	In some sale, the price for a high-quality watch was 298 dollars. A fair market price for a watch is typically	In some sale, the price for a high-quality watch was 298 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a watch is typically
watch	974	A fair market price for a watch is typically	In some sale, the price for a high-quality watch was 974 dollars. A fair market price for a watch is typically	In some sale, the price for a high-quality watch was 974 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a watch is typically
watch	851	A fair market price for a watch is typically	In some sale, the price for a high-quality watch was 851 dollars. A fair market price for a watch is typically	In some sale, the price for a high-quality watch was 851 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a watch is typically
backpack	397	A fair market price for a backpack is typically	In some sale, the price for a high-quality backpack was 397 dollars. A fair market price for a backpack is typically	In some sale, the price for a high-quality backpack was 397 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a backpack is typically
backpack	865	A fair market price for a backpack is typically	In some sale, the price for a high-quality backpack was 865 dollars. A fair market price for a backpack is typically	In some sale, the price for a high-quality backpack was 865 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a backpack is typically
backpack	176	A fair market price for a backpack is typically	In some sale, the price for a high-quality backpack was 176 dollars. A fair market price for a backpack is typically	In some sale, the price for a high-quality backpack was 176 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a backpack is typically
espresso machine	505	A fair market price for a espresso machine is typically	In some sale, the price for a high-quality espresso machine was 505 dollars. A fair market price for a espresso machine is typically	In some sale, the price for a high-quality espresso machine was 505 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a espresso machine is typically
espresso machine	155	A fair market price for a espresso machine is typically	In some sale, the price for a high-quality espresso machine was 155 dollars. A fair market price for a espresso machine is typically	In some sale, the price for a high-quality espresso machine was 155 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a espresso machine is typically
espresso machine	253	A fair market price for a espresso machine is typically	In some sale, the price for a high-quality espresso machine was 253 dollars. A fair market price for a espresso machine is typically	In some sale, the price for a high-quality espresso machine was 253 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a espresso machine is typically
running shoes	291	A fair market price for a running shoes is typically	In some sale, the price for a high-quality running shoes was 291 dollars. A fair market price for a running shoes is typically	In some sale, the price for a high-quality running shoes was 291 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a running shoes is typically
running shoes	195	A fair market price for a running shoes is typically	In some sale, the price for a high-quality running shoes was 195 dollars. A fair market price for a running shoes is typically	In some sale, the price for a high-quality running shoes was 195 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a running shoes is typically
running shoes	68	A fair market price for a running shoes is typically	In some sale, the price for a high-quality running shoes was 68 dollars. A fair market price for a running shoes is typically	In some sale, the price for a high-quality running shoes was 68 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a running shoes is typically
gaming console	783	A fair market price for a gaming console is typically	In some sale, the price for a high-quality gaming console was 783 dollars. A fair market price for a gaming console is typically	In some sale, the price for a high-quality gaming console was 783 dollars. Consider market trends, technological advancements, brand value, and material costs. A fair market price for a gaming console is typically

Structure of the Benchmark:

Benchmark Structure:

- Product: The benchmark uses a predefined list of products. Each product appears 2 to 3 times with different anchor prices.
- Anchor Price: Each product is associated with a randomly generated price within a range of $ [0, 1000].

Prompt Structure Overview

Each scenario in the benchmark employs a structured approach to prompts, designed to test the influence of anchoring bias and the effectiveness of additional contextual information. Here is a detailed breakdown of the prompt templates:

- **Basic Prompt**: This prompt sets a neutral ground for price estimation, devoid of any anchoring bias.

```
basic_prompt = f"A fair market price for a {product} is typically."
```

- **Anchored Prompt**: Introduces a specific anchor price that might influence the model's price estimation.

```
anchored_prompt = f"In some sale, the price for a high-quality {product}
```

- **Guided Prompt**: Combines the basic prompt with additional context to counteract the anchoring bias, enhanced by few-shot learning examples:

```
guided_prompt = f"A fair market price for a {product} is typically. Consi
few_shot_examples = "Example 1: In some store, the price of a smartphone
```

Application in Benchmark

- **Diverse Scenarios**: Utilizes prompt variations across a wide array of products and anchor prices to explore the anchoring effect in different contexts. Randomly assigned anchor prices simulate the variability consumers might encounter in real-life scenarios.
- **Contrasting Anchors**: Each product scenario features multiple variations with different, randomly chosen anchor prices. This randomness is intended to simulate diverse pricing information that consumers encounter and to test the model's responses even when the anchor is clearly irrelevant or unlikely to be reliable. This approach aims to evaluate the model's resilience to such cognitive biases under varied conditions of awareness, checking whether anchoring effects persist despite general knowledge suggesting the irrelevance of the anchor.
- **Controlled Comparisons**: Comparisons between scenarios with anchors and those without help quantify the influence of anchoring on decision-making processes. Using random anchor prices enriches the dataset, providing a clearer picture of how anchoring affects model behavior across a spectrum of initial cues.

This benchmark serves as a strategic tool for systematically examining the susceptibility of LLMs to cognitive biases. It highlights the need for developing technologies that can recognize and mitigate biases analogous to those in human judgment. All prompts conclude with a basic prompt at the end, inviting the model to complete it with a price estimate.

For a comprehensive exploration and to access the complete dataset, visit the benchmark repository.

This structured approach not only identifies the presence of anchoring bias but also explores debiasing techniques to enhance the accuracy and impartiality of AI judgments.

Below are the scripts used to create the benchmark:

```
def create_full_data_with_prompts(product, price):
    """
    Generate statements with price context and different types of prompts for eac
    """
    # Generate different types of prompts
    basic_prompt = f"A fair market price for a {product} is typically"
    anchored_prompt = f"In some sale, the price for a high-quality {product} was
    guided_prompt = (f"In some sale, the price for a high-quality {product} was {
                     "Consider market trends, technological advancements, brand v
                     f"A fair market price for a {product} is typically")

    return product, price, basic_prompt, anchored_prompt, guided_prompt

# List of products to use
products = [
    'smartphone', 'laptop', 'headphones', 'bicycle', 'watch',
    'backpack', 'espresso machine', 'running shoes', 'gaming console',
    'car', 'blender', 'desk chair', 'sunglasses', 'bookshelf',
    'kitchen mixer', 'digital camera', 'fitness tracker', 'tent',
    'suitcase', 'grill', 'snowboard', 'hiking boots', 'drone',
    'graphic tablet', 'electric kettle', 'wireless charger', 'yoga mat',
    'board game', 'action camera', 'water bottle', 'office desk', 'guitar',
    'puzzle', 'wine rack', 'coffee table', 'novel', 'skateboard',
    'sleeping bag', 'lamp', 'flower pot', 'photo frame', 'wall art',
    'tablecloth', 'candle holder', 'throw pillow', 'garden tools',
    'bath towel', 'cutlery set', 'wine glasses', 'yarn', 'paint set',
    'notebook', 'pen', 'desk organizer', 'alarm clock', 'calculator'
]

# Generate all data, including statements and prompts for each product
data = [create_full_data_with_prompts(product, random.randint(*PRICE_RANGE)) for
data = data[:NUMBER_OF_EXAMPLES]  # Limit to NUMBER_OF_EXAMPLES

# Create a DataFrame
column_names = ['Product', 'Anchor Price', 'Basic Prompt', 'Anchored Prompt', 'Gu
df = pd.DataFrame(data, columns=column_names)

# Save the DataFrame to an Excel file
df.to_excel(DATA_FILE_PATH, index=False)

print(f"Excel file created at {DATA_FILE_PATH}")
```

```
def update_guided_prompts_with_few_shot_learning(df):
    """
    Updates the 'Guided Prompt' column of the provided DataFrame with few-shot le
    """
    # Define few-shot learning examples
    few_shot_examples = [
    "Example 1: In some store, the price of a smartphone is $200. considering tec
    "Example 2: Even though a bicycle was previously sold for $9000, taking into
    "Example 3: Despite the high-quality laptop being advertised at $40 during re
    ]

    # Update the 'Guided Prompt' column by appending few-shot examples before the
    df['Guided Prompt'] = df.apply(lambda row: "".join(few_shot_examples) + row['

    return df

# Load the DataFrame from the specified path
df = pd.read_excel(DATA_FILE_PATH)

# Update the DataFrame with few-shot learning examples in the 'Guided Prompt'
updated_df = update_guided_prompts_with_few_shot_learning(df)

# Save the updated DataFrame to a new Excel file
updated_df.to_excel(FEW_SHOT_FILE_PATH, index=False)

print(f"Updated Excel file with few-shot learning prompts created at {FEW_SHOT_FI
```

```
Updated Excel file with few-shot learning prompts created at /content/few_sho
```

Utility Functions

This section details the utility functions that were used in executing the benchmark. These functions are designed to streamline various aspects of setting up and running the scenarios, ensuring the process is efficient and consistent. Below, I provide a comprehensive explanation of each function, including its purpose and how it integrates into the overall benchmark execution workflow.

```
def set_seed(seed):
    """
    Sets the seed for generating random numbers to ensure reproducibility.
    This sets the seed for both the CPU and GPU (if available).

    Parameters:
    - seed (int): The seed value for random number generators.

    """
    torch.manual_seed(seed)  # Set the seed for CPU
    if torch.cuda.is_available():  # Check if GPU is available
        torch.cuda.manual_seed_all(seed)  # Set the seed for all GPUs
```

```
def initialize_model(model_name):
    """
    Initializes and returns a quantized transformer model, its tokenizer, and an
    This function is specifically tailored for causal language models and include
    settings using the bitsandbytes library.

    Parameters:
    - model_name (str): The name of the pretrained model.

    Returns:
    - tuple: Contains the model, tokenizer, and accelerator objects.
    """
    set_seed(42)  # Ensure reproducibility

    # Load the tokenizer for the specified model
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    # Configure the model for quantization
    quant_config = BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_use_double_quar

    # Load the model with the specified quantization config and map it to the app
    model = AutoModelForCausalLM.from_pretrained(model_name, quantization_config=

    # Initialize the accelerator for distributed or optimized single device execu
    accelerator = Accelerator()

    # Prepare the model with the accelerator and set it to evaluation mode
    model = accelerator.prepare(model)
    model.eval()

    return model, tokenizer, accelerator
```

```
def generate_text(model, tokenizer, accelerator, prompt, base_length=10):
    """Generate text based on the provided prompt using the model.

    Parameters:
        model: The language model for text generation.
        tokenizer: Tokenizer for encoding and decoding the prompt.
        accelerator: Accelerator object for device placement.
        prompt: The input text prompt for the model.
        base_length: The additional maximum length for generated text.

    Returns:
        str: The generated text.
    """
    input_ids = tokenizer.encode(prompt, return_tensors="pt")
    input_ids = input_ids.to(accelerator.device)
    total_length = input_ids.shape[1] + base_length
    with torch.no_grad():
        output_ids = model.generate(input_ids, max_length=total_length, num_retur
    generated_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    return generated_text[len(tokenizer.decode(input_ids[0], skip_special_tokens=
```

Post Processing

Evaluating the responses from large language models (LLMs) is essential to extract optimal price estimations effectively. The functions listed below have proven to be particularly effective in processing the responses generated by the models I chose to use in the study. This section will detail each function's purpose and how they contribute to refining and optimizing the extraction of price estimates.

```
# Post processing for facebook/opt-350m model

def extract_and_validate_price(response):
    """Extract price from the response and validate if it falls within the predef

    Parameters:
        response (str): The text response from the model.

    Returns:
        int or None: The extracted price if valid, or None if no valid price is f
    """
    # Extract prices using regex. The pattern covers numbers with commas
    prices = re.findall(r'\b\d{1,3}(?:,\d{3})*(?:\.\d+)?\b', response)
    # Convert extracted price strings to float, remove commas, and validate agair
    valid_prices = [float(price.replace(',', '')) for price in prices if PRICE_RA
    if valid_prices:
        # Return the first valid price found
        return valid_prices[0]
    else:
        # Return None if no valid price is found
        return None


# Post processing for 'EleutherAI/gpt-j-6B', 'mrm8488/mistral-7b-ft-h4-no_robots_

def extract_and_average_validate_price(response):
    """Extract and validate the average price from a response containing price ra

    Parameters:
        response (str): The text response that might contain price ranges.

    Returns:
        float or None: The average of the extracted price range if it falls withi
    """
    # Extract prices using a pattern that captures typical expressions of ranges
    prices = re.findall(r'\b\d{1,3}(?:,\d{3})*(?:\.\d+)?\b', response.replace(','
    numbers = [float(price) for price in prices]
    if numbers:
        average_price = sum(numbers) / len(numbers)
        # Validate if the average price falls within the predefined range
        if PRICE_RANGE[0] <= average_price <= PRICE_RANGE[1]:
            return average_price
    return None



# Initialize model and tokenizer
model, tokenizer, accelerator = initialize_model(MODEL_NAME)
```

Executing or Running the Benchmark

This code segment starts with demonstration of the process of runningthe benchmark on a single dataset example with the chosen model. It involves generating responses to various

prompt types and extracting validated price estimates from those responses. This provides insight into how the model processes input and produces outputs under different conditions.

Overview of `execute_and_process_benchmark` Function

The `execute_and_process_benchmark` function is designed to process a dataset by generating and analyzing text responses from a pre-trained model. It handles different types of prompts—Basic, Anchored, and Guided—to evaluate how each influences the model's output, specifically focusing on price estimations.

Key Steps:

1. **Generate Responses:** For each data entry, the model generates responses based on Basic, Anchored, and Guided prompts.
2. **Extract Prices:** Prices are extracted and validated from each model response, ensuring they reflect accurate estimations.
3. **Calculate Differences:** The function calculates differences between the price estimates derived from various prompt types to measure the influence of anchoring and guidance.
4. **Save Results:** All relevant data, including product details, model responses, price estimates, and comparisons, are systematically saved to an Excel file for further analysis.

```
basic_prompt = "A fair market price for a smartphone product is typically"
anchored_prompt = "In some sale, the price for a high-quality smartphone was 386
guided_prompt = "In some sale, the price for a high-quality smartphone was 386 do

basic_response = generate_text(model, tokenizer, accelerator, basic_prompt)
anchored_response = generate_text(model, tokenizer, accelerator, anchored_prompt)
guided_response = generate_text(model, tokenizer, accelerator, guided_prompt)

processed_basic_response = extract_and_validate_price(basic_response)
processed_anchored_response = extract_and_validate_price(basic_response)
processed_guided_response = extract_and_validate_price(guided_response)

print("Response for prompt:", basic_response)
print("Processed Response for prompt:", processed_basic_response)
print("Response without Guidance:", anchored_response)
print("Processed Response without Guidance:", processed_anchored_response)
print("Response with Guidance:", guided_response)
print("Processed Response with Guidance:", processed_guided_response)
```

```
    def execute_and_process_benchmark(model_name, result_file_path):
        """
        Executes the benchmark scenarios for a given model and processes the response
        This function generates responses using basic, anchored, and guided prompts,
        these prices, and saves the results in an Excel file.

        Parameters:
        - model_name (str): The name of the model being used. This is used for loggir
        - result_file_path (str): The file path where the Excel file containing the r

        Returns:
        - result_df (DataFrame): A pandas DataFrame containing all the results from t
            - Anchor Price: The anchor price used in the benchmark.
            - Product: The product name for which the benchmark was executed.
            - Basic Response: The model's response to the basic prompt.
            - Basic Estimated Price: The price extracted from the basic response.
            - Anchored Response: The model's response to the anchored prompt.
            - Anchored Estimated Price: The price extracted from the anchored respons
            - Guided Response: The model's response to the guided prompt.
            - Guided Estimated Price: The price extracted from the guided response.
            - Basic-Anchored Diff: The absolute difference in estimated prices betwee
            - Anchored-Guided Diff: The absolute difference in estimated prices betwe
            - Basic-Guided Diff: The absolute difference in estimated prices between

        Notes:
        - Ensure that the DataFrame `df` is available in your scope with the necessar
          'Guided Prompt', 'Anchor Price', and 'Product'.
        """
        results = []
        # Generate text and extract prices
        for index, row in df.iterrows():
            basic_response = generate_text(model, tokenizer, accelerator, row['Basic
            anchored_response = generate_text(model, tokenizer, accelerator, row['Anc
            guided_response = generate_text(model, tokenizer, accelerator, row['Guide

            basic_price = extract_and_average_validate_price(basic_response)  # extra
            anchored_price = extract_and_average_validate_price(anchored_response)  #
            guided_price = extract_and_average_validate_price(guided_response)  # ext

            # Calculate differences, using np.nan for missing values if any prices ar
            basic_anchored_diff = np.nan if (basic_price is None or anchored_price is
            anchored_guided_diff = np.nan if (anchored_price is None or guided_price
            basic_guided_diff = np.nan if (basic_price is None or guided_price is Nor

            results.append({
                'Anchor Price': row['Anchor Price'],
                'Product': row['Product'],
                'Basic Response': basic_response,
                'Basic Estimated Price': basic_price,
                'Anchored Response': anchored_response,
                'Anchored Estimated Price': anchored_price,
                'Guided Response': guided_response,
                'Guided Estimated Price': guided_price,
                'Basic-Anchored Diff': basic_anchored_diff,
                'Anchored-Guided Diff': anchored_guided_diff,
```

```
            'Basic-Guided Diff': basic_guided_diff
        })

    # Save results to DataFrame and then to Excel
    result_df = pd.DataFrame(results)
    result_df.to_excel(result_file_path, index=False)
    print(f"Results have been saved to {result_file_path}")
    return result_df


# Load data from Excel file
df = pd.read_excel(DATA_FILE_PATH)

result_df = execute_and_process_benchmark(MODEL_NAME, RESULT_FILE_PREFIX)
```

Overview of perform_statistical_analysis Function

This function conducts statistical analysis on a dataset containing price estimates from different prompting strategies. It assesses potential biases and the effectiveness of debiasing measures through paired t-tests.

Key Steps:

- Prepare Data: Drops rows with missing price data and ensures there are enough data points for analysis.
- Statistical Testing: Performs paired t-tests between:
    1. Basic and Anchored estimates to check initial biases.
    2. Anchored and Guided estimates to evaluate debiasing effects.
    3. Save Results: Compiles t-statistics and p-values into a DataFrame and saves it as an Excel file.

Usage:

```
perform_statistical_analysis(result_df, '/path/to/analysis_results.xlsx')
```

This function is crucial for validating the reliability of price estimations and ensuring that any adjustments to the model's prompting strategies are statistically significant.

Statistical Analysis of Anchoring Bias

This function assesses anchoring bias in price estimations by conducting paired t-tests on differences from the anchor price across different types of prompts: basic, anchored, and guided.

Key Steps:

1. **Data Filtering**: Rows missing any necessary price or anchor information are excluded to ensure the analysis uses only complete data sets.
2. **Difference Calculation**: Absolute differences from the anchor price are calculated for each type of prompt. This quantifies how closely each prompt's price estimation adheres to the anchor.
3. **Statistical Testing**:
    - **Basic vs. Anchored**: Compares how basic (control) and anchored prompts differ in their proximity to the anchor price, testing the direct influence of the anchor.
    - **Anchored vs. Guided**: Evaluates if guided prompts, which include additional contextual information, lead to price estimates further from the anchor compared to anchored prompts, assessing the effectiveness of debiasing strategies.

Appropriateness of the Tests:

- **Paired T-Tests** are ideal here due to the related nature of the data sets—each product is assessed under all three conditions, allowing us to directly compare their responses within the same experimental framework. These tests help determine if significant statistical differences exist between the groups, indicating the presence of anchoring bias and the efficacy of any debiasing interventions.

The results, including T-Statistics and P-Values, are displayed in a DataFrame and saved to an Excel file, facilitating a clear and accessible presentation of findings. This rigorous approach ensures that the conclusions about anchoring bias and the potential for debiasing are grounded in statistically valid comparisons.

```
def perform_anchor_bias_analysis(result_df, analysis_file_path):
    """
    Performs a statistical analysis to assess anchoring bias in pricing estimatic

    This function calculates the absolute differences from the anchor price for p
    basic, anchored, and guided prompts. It then conducts paired t-tests to compa
    the presence and mitigation of anchoring bias.

    Parameters:
        result_df (pd.DataFrame): DataFrame containing columns for 'Basic Estimat
                                  'Guided Estimated Price', and 'Anchor Price'.
        analysis_file_path (str): Path where the Excel report of the analysis res

    Outputs:
        Excel file: Saves the statistical analysis results to an Excel file speci
        Console output: Prints the number of examples used and the t-test results
    """

    # Drop rows where any necessary price information or anchor information is mi
    filtered_df = result_df.dropna(subset=['Basic Estimated Price', 'Anchored Est
    print(f"The statistical analysis is using {len(filtered_df)} examples")

    # Check if there are enough data points to perform statistical tests
    if len(filtered_df) >= 2:
        # Calculate absolute differences from the anchor for Basic, Anchored, anc
        basic_diffs = abs(filtered_df['Basic Estimated Price'] - filtered_df['Anc
        anchored_diffs = abs(filtered_df['Anchored Estimated Price'] - filtered_d
        guided_diffs = abs(filtered_df['Guided Estimated Price'] - filtered_df['A

        # Perform t-tests between the differences
        basic_anchored_diff_t_stat, basic_anchored_diff_p_value = ttest_rel(basic
        anchored_guided_diff_t_stat, anchored_guided_diff_p_value = ttest_rel(anc

        # Prepare the results in a DataFrame
        analysis_results = pd.DataFrame({
            'Comparison': ['Basic vs. Anchored Distance to Anchor', 'Anchored vs.
            'T-Statistic': [basic_anchored_diff_t_stat, anchored_guided_diff_t_st
            'P-Value': [basic_anchored_diff_p_value, anchored_guided_diff_p_value
        })

        # Print and save the results
        print("Statistical Analysis Results:")
        print(analysis_results)
        analysis_results.to_excel(analysis_file_path, sheet_name='Anchor Bias Ana
        print(f"Analysis has been saved to {analysis_file_path}")

    else:
        print("Not enough data points for statistical tests.")
```

```
def perform_mean_difference_analysis(result_df, analysis_file_path):
    """
    Performs an analysis to assess the mean differences in price estimations from
    across different types of prompts: basic, anchored, and guided.

    This function calculates the mean differences from the anchor price for each
    providing insight into how much the anchoring effect influences the price est
    made under different prompting conditions.

    The function first filters out any rows missing necessary price or anchor inf
    It calculates the mean differences from the anchor price for each type of prc
    and outputs these results both in the console and in an Excel file.

    Parameters:
        result_df (pd.DataFrame): DataFrame containing columns for 'Basic Estimat
                                  'Anchored Estimated Price', 'Guided Estimated F
                                  and 'Anchor Price'.
        analysis_file_path (str): Path where the Excel report of the analysis res

    Outputs:
        Excel file: Saves the mean differences analysis results to an Excel file
                    analysis_file_path.
        Console output: Prints the number of examples used and the mean differenc
    """

    # Drop rows where any necessary price information or anchor information is mi
    filtered_df = result_df.dropna(subset=['Basic Estimated Price', 'Anchored Est
    print(f"The analysis is using {len(filtered_df)} examples")

    # Calculate mean differences from the anchor for Basic, Anchored, and Guided
    mean_diff_basic = (filtered_df['Basic Estimated Price'] - filtered_df['Anchor
    mean_diff_anchored = (filtered_df['Anchored Estimated Price'] - filtered_df['
    mean_diff_guided = (filtered_df['Guided Estimated Price'] - filtered_df['Anch

    # Prepare the results in a DataFrame
    analysis_results = pd.DataFrame({
        'Prompt Type': ['Basic', 'Anchored', 'Guided'],
        'Mean Difference from Anchor': [mean_diff_basic, mean_diff_anchored, mear
    })

    # Print and save the results
    print("Mean Difference Analysis Results:")
    print(analysis_results)
    analysis_results.to_excel(analysis_file_path, sheet_name='Mean Difference Ana
    print(f"Analysis has been saved to {analysis_file_path}")


result_df = pd.read_excel('/content/results.xlsx')
perform_mean_difference_analysis(result_df, '/content/simpler_analysis_results.xl


result_df = pd.read_excel('/content/results.xlsx')
perform_anchor_bias_analysis(result_df, '/content/analysis_results.xlsx')
```

Results

Analysis and Debiasing Techniques

Model Response Analysis

In analyzing the responses from various models, I encountered two types of patterns

1. The model's responses typically led with a valid price, which corresponded accurately to the intended estimations. Thus, I chose to extract the first numerical value generated by the model as the estimated price. This straightforward approach provided a contrast to other models, which required more complex post-processing strategies to identify and extract the most relevant numerical information.
2. Price range representations. I calculated the average of all numerical values mentioned in the response, utilizing the `extract_and_average_validate_price` function. This allowed for a more balanced estimation, factoring in the entire range of numbers provided by the model.

Additionally, during the statistical analysis, any rows associated with prompts that failed to generate numeric estimations were excluded from the dataset.

Bias Evaluation

To counteract the anchoring bias, I employed a strategy of prompt engineering that provided the models with context-rich guided prompts. This technique steered the focus away from the potentially misleading anchor price, redirecting attention to a holistic analysis that accounts general market knowledge. The prompts were designed to remind the model to consider rational factors—like market trends and material costs—in its estimations. The goal was to weaken the impact of the anchoring bias on the model's output, fostering more grounded and unbiased price assessments.

Debiasing Strategies

The debiasing process involved presenting models with context-enriched guided prompts, employing prompt engenireeing technique, aiming to shift their focus from the anchor price to a more comprehensive price analysis, using general knowledge and reminding that the estimation process should be based on reasonable factors such as market trends, material costs etc. This approach sought to dilute the anchoring bias's influence on the generated estimates.

Comparative Results

The statistical assessment, conducted via paired t-tests, revealed:

- **Basic vs. Anchored**: A positive T-Statistic indicated a significant anchoring effect, where anchored prompts led to estimates closer to the anchor price.

- **Anchored vs. Guided**: A negative T-Statistic suggested the guided prompts' effectiveness in reducing anchoring bias, evidenced by estimates moving away from the anchor price.

The P-Values affirmed these results' statistical significance (<0.05), solidifying the impact of anchoring and the corrective power of guided information.

facebook/opt-350m

Model Overview

The `facebook/opt-350m` model, part of Facebook AI's OPT series, excels in parsing and generating text that closely resembles human language. It's celebrated for its efficient design and scalability, making it a powerful tool for natural language processing tasks.

Post Processing

- first approach

Comparative Results

```
Comparison,T-Statistic,P-Value
Basic vs. Anchored Distance to Anchor,6.910979397,0.0000000009101749835
Anchored vs. Guided Distance to Anchor,-2.829062906,0.005852024966
```

```
Prompt Type,Mean Difference from Anchor
Basic,-236.3333333
Anchored,-9.380952381
Guided,111.3333333
```