

```
{
  "cells": [
    {
      "cell_type": "markdown",
      "id": "2a3c15e2",
      "metadata": {
        "_cell_guid": "b1076dfc-b9ad-4769-8c92-a6c4dae69d19",
        "_uuid": "8f2839f25d086af736a60e9eeb907d3b93b6e0e5",
        "execution": {
          "iopub.execute_input": "2025-11-15T07:16:20.018703Z",
          "iopub.status.busy": "2025-11-15T07:16:20.017898Z",
          "iopub.status.idle": "2025-11-15T07:16:20.024932Z",
          "shell.execute_reply": "2025-11-15T07:16:20.023792Z",
          "shell.execute_reply.started": "2025-11-15T07:16:20.018675Z"
        },
        "papermill": {
          "duration": 0.004361,
          "end_time": "2025-11-15T08:51:44.300008",
          "exception": false,
          "start_time": "2025-11-15T08:51:44.295647",
          "status": "completed"
        },
        "tags": []
      },
      "source": [
        "# AutoProposal - Client Proposal Generator Agent (Kaggle Demo)\n",
        "This notebook builds a simple multi-agent pipeline that:\n",
        "- Parses a client brief\n",
        "- Researches competitors (DuckDuckGo search)\n",
        "- Estimates pricing & ROI (mock model)\n",
        "- Generates a proposal PDF\n",
        "\n",
        "***Run each code cell in order (Shift+Enter).**\n",
        "If you don't have API keys, the notebook runs in **MOCK MODE** (no external APIs)."
      ]
    },
    {
      "cell_type": "code",
      "execution_count": 1,
      "id": "c2492aa9",
      "metadata": {
        "execution": {
          "iopub.execute_input": "2025-11-15T08:51:44.308951Z",
          "iopub.status.busy": "2025-11-15T08:51:44.308618Z",
          "iopub.status.idle": "2025-11-15T08:51:54.350699Z",
          "shell.execute_reply": "2025-11-15T08:51:54.349485Z"
        },
        "papermill": {
          "duration": 10.048745,
          "end_time": "2025-11-15T08:51:54.352354",
          "exception": false,
          "start_time": "2025-11-15T08:51:44.303609",
          "status": "completed"
        },
        "tags": []
      },
      "outputs": [
        {
          "name": "stdout",
          "output_type": "stream",
          "text": [
            "\u001b[2K \u001b[90m\u001b[0m \u001b[32m360.7/360.7 kB\u001b[0m \u001b[31m5.9 MB/s\u001b[0m eta \u001b[36m0:00:00\u001b[0m\r\n",
            "\u001b[2K \u001b[90m\u001b[0m \u001b[32m2.0/2.0"
          ]
        }
      ]
    }
  ]
}
```

```
MB\001b[0m \001b[31m33.2 MB/s\001b[0m eta \001b[36m0:00:00\001b[0m\r\n",
    "\001b[2K \001b[90m\u2022\001b[32m3.3/3.3
MB\001b[0m \001b[31m54.0 MB/s\001b[0m eta \001b[36m0:00:00\001b[0m\r\n",
    "\001b[?25h\001b[31mERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following dependency
conflicts.\r\n",
    "litellm 1.76.3 requires openai>=1.99.5, but you have openai 1.40.2 which is
incompatible.\001b[0m\001b[31m\r\n",
    "\001b[0m✓ Dependencies installed\r\n"
]
}
],
{
"source": [
"# Install required packages (run once)\n",
"!pip install --quiet openai==1.40.2 reportlab duckduckgo-search python-dotenv\n",
"print(\"✓ Dependencies installed\")"
]
},
{
"cell_type": "code",
"execution_count": 2,
"id": "3a9f0711",
"metadata": {
"execution": {
"iopub.execute_input": "2025-11-15T08:51:54.361399Z",
"iopub.status.busy": "2025-11-15T08:51:54.361036Z",
"iopub.status.idle": "2025-11-15T08:51:54.500803Z",
"shell.execute_reply": "2025-11-15T08:51:54.499778Z"
},
"papermill": {
"duration": 0.14651,
"end_time": "2025-11-15T08:51:54.502513",
"exception": false,
"start_time": "2025-11-15T08:51:54.356003",
"status": "completed"
},
"tags": []
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"\u26a1 MOCK MODE: No OpenAI key found. LLM calls will be mocked.\n"
]
}
],
"source": [
"# Imports and environment loader\n",
"import os\n",
"import json\n",
"import random\n",
"from duckduckgo_search import DDGS\n",
"from reportlab.lib.pagesizes import A4\n",
"from reportlab.pdfgen import canvas\n",
"from dotenv import load_dotenv\n",
"\n",
"load_dotenv() # loads .env if present\n",
"\n",
"# Check for OpenAI key (optional)\n",
"OPENAI_API_KEY = os.getenv(\"OPENAI_API_KEY\")\n",
"if OPENAI_API_KEY:\n",
"    print(\"OpenAI key found - real LLM mode enabled (optional).\")\n",
"else:\n"
]
]
```

```
"    print(\" MOCK MODE: No OpenAI key found. LLM calls will be mocked.\")"
]
},
{
"cell_type": "code",
"execution_count": 3,
"id": "9891e5bc",
"metadata": {
"execution": {
"iopub.execute_input": "2025-11-15T08:51:54.511166Z",
"iopub.status.busy": "2025-11-15T08:51:54.510728Z",
"iopub.status.idle": "2025-11-15T08:51:54.517595Z",
"shell.execute_reply": "2025-11-15T08:51:54.516693Z"
},
"papermill": {
"duration": 0.013157,
"end_time": "2025-11-15T08:51:54.519205",
"exception": false,
"start_time": "2025-11-15T08:51:54.506048",
"status": "completed"
},
"tags": []
},
"outputs": [],
"source": [
"# Simple LLM wrapper: uses OpenAI if key present, otherwise returns mock text\n",
"def llm(prompt):\n",
"    \"\"\"\n",
"    If OPENAI_API_KEY present, attempt a Chat completion.\n",
"    Otherwise return a deterministic MOCK response.\n",
"    \"\"\"\n",
"    if OPENAI_API_KEY is None:\n",
"        # deterministic mock: keep output short but useful\n",
"        sample = (\n",
"            \"MOCK SUMMARY:\n",
"            + prompt.split(\"\\n\")[0][:150]\\n",
"            + \" ...\\n\\n(Use real API key to enable full LLM responses.)\\n",
"        )\\n",
"        return sample\\n",
"    \"\"\"\n",
"    # Real API usage (if you add key later)\n",
"    try:\n",
"        from openai import OpenAI\n",
"        client = OpenAI(api_key=OPENAI_API_KEY)\n",
"        completion = client.chat.completions.create(\n",
"            model=\"gpt-4o-mini\",\n",
"            messages=[{\"role\": \"user\", \"content\": prompt}],\n",
"            max_tokens=600\\n",
"        )\\n",
"        return completion.choices[0].message.content\\n",
"    except Exception as e:\n",
"        return f\"LLM ERROR (falling back to MOCK): {e}\""
]
},
{
"cell_type": "code",
"execution_count": 4,
"id": "b7bfeb06",
"metadata": {
"execution": {
"iopub.execute_input": "2025-11-15T08:51:54.527916Z",
"iopub.status.busy": "2025-11-15T08:51:54.527613Z",
"iopub.status.idle": "2025-11-15T08:51:54.534538Z",
"shell.execute_reply": "2025-11-15T08:51:54.533697Z"
}
},
```

```

"papermill": {
    "duration": 0.013259,
    "end_time": "2025-11-15T08:51:54.536104",
    "exception": false,
    "start_time": "2025-11-15T08:51:54.522845",
    "status": "completed"
},
"tags": []
},
"outputs": [],
"source": [
"def competitor_research(industry, keywords=\"\", max_results=5):\n",
"    \"\"\"\n",
"        Use DuckDuckGo to fetch top pages for competitor research.\n",
"        Returns list of title & url strings. In mock mode returns sample list.\n",
"    \"\"\"\n",
"    print(\"🔍 Running competitor research...\")\n",
"    try:\n",
"        results = []\n",
"        with DDGS() as ddgs:\n",
"            query = f'{industry} {keywords} competitors'\n",
"            for r in ddgs.text(query, max_results=max_results):\n",
"                # r has 'title' and 'href'\n",
"                title = r.get('title', '')[:200]\n",
"                href = r.get('href', '')\n",
"                results.append(f'{title} - {href}')\n",
"        if not results:\n",
"            raise Exception('No results')\n",
"        return results\n",
"    except Exception as e:\n",
"        print(\"⚠ Research tool fallback (mock).\", e)\n",
"        # Mock list\n",
"        return [\n",
"            f'{industry} Competitor A - https://example.com/a',\n",
"            f'{industry} Competitor B - https://example.com/b',\n",
"            f'{industry} Competitor C - https://example.com/c',\n",
"        ]\n"
},
{
"cell_type": "code",
"execution_count": 5,
"id": "f8c6ae2c",
"metadata": {
    "execution": {
        "iopub.execute_input": "2025-11-15T08:51:54.544747Z",
        "iopub.status.busy": "2025-11-15T08:51:54.544399Z",
        "iopub.status.idle": "2025-11-15T08:51:54.550155Z",
        "shell.execute_reply": "2025-11-15T08:51:54.549179Z"
    },
    "papermill": {
        "duration": 0.01218,
        "end_time": "2025-11-15T08:51:54.551887",
        "exception": false,
        "start_time": "2025-11-15T08:51:54.539707",
        "status": "completed"
    },
    "tags": []
},
"outputs": [],
"source": [
"def calculate_pricing(deliverables_count, complexity=1.5, base_price=150):\n",
"    \"\"\"\n",
"        Simple pricing model:\n",
"        price = base_price * complexity * deliverables_count\n",
"
```

```
"    ROI: random multiplier to simulate business value\n",
"    \"\"\"\n",
"    price = base_price * complexity * max(1, int(deliverables_count))\n",
"    roi_multiplier = random.uniform(2.0, 4.0)\n",
"    roi_value = round(price * roi_multiplier, 2)\n",
"    return {\n",
"        \"estimated_price\": round(price, 2),\n",
"        \"expected_roi_value\": roi_value,\n",
"        \"details\": {\n",
"            \"base_price\": base_price,\n",
"            \"complexity\": complexity,\n",
"            \"deliverable_count\": deliverables_count\n",
"        }\n",
"    }\n"
},
{
"cell_type": "code",
"execution_count": 6,
"id": "c2c48345",
"metadata": {
"execution": {
"iopub.execute_input": "2025-11-15T08:51:54.560373Z",
"iopub.status.busy": "2025-11-15T08:51:54.560002Z",
"iopub.status.idle": "2025-11-15T08:51:54.567303Z",
"shell.execute_reply": "2025-11-15T08:51:54.566382Z"
},
"papermill": {
"duration": 0.013483,
"end_time": "2025-11-15T08:51:54.568900",
"exception": false,
"start_time": "2025-11-15T08:51:54.555417",
"status": "completed"
},
"tags": []
},
"outputs": [],
"source": [
"def generate_pdf(client_name, proposal_text, out_dir=\"/kaggle/working\"):\\n",
"    \"\"\"\n",
"    Very simple PDF generator using reportlab.\n",
"    Saves PDF to Kaggle working directory for easy download.\n",
"    \"\"\"\n",
"    safe_name = client_name.replace(\" \", \"_\")\\n",
"    file = os.path.join(out_dir, f\"{safe_name}_proposal.pdf\")\\n",
"    c = canvas.Canvas(file, pagesize=A4)\\n",
"    width, height = A4\\n",
"\\n",
"    # Simple header\\n",
"    c.setFont(\"Helvetica-Bold\", 16)\\n",
"    c.drawString(40, height - 60, f\"Proposal for {client_name}\")\\n",
"\\n",
"    # Body text\\n",
"    text = c.beginText(40, height - 100)\\n",
"    text.setFont(\"Helvetica\", 10)\\n",
"    max_width_chars = 110\\n",
"    for paragraph in proposal_text.split(\"\\n\"):\\n",
"        # naive wrap\\n",
"        while len(paragraph) > max_width_chars:\\n",
"            text.textLine(paragraph[:max_width_chars])\\n",
"            paragraph = paragraph[max_width_chars:]\\n",
"            text.textLine(paragraph)\\n",
"            text.textLine(\"\") # blank line\\n",
"\\n",
"        c.drawText(text)\\n",
"    
```

```

    "c.showPage()\n",
    "c.save()\n",
    "print(f"📄 PDF generated at: {file}\")\n",
    "return file\n"
]
},
{
"cell_type": "code",
"execution_count": 7,
"id": "f9e840a8",
"metadata": {
"execution": {
"iopub.execute_input": "2025-11-15T08:51:54.577723Z",
"iopub.status.busy": "2025-11-15T08:51:54.577085Z",
"iopub.status.idle": "2025-11-15T08:51:54.583727Z",
"shell.execute_reply": "2025-11-15T08:51:54.582184Z"
},
"papermill": {
"duration": 0.013483,
"end_time": "2025-11-15T08:51:54.585946",
"exception": false,
"start_time": "2025-11-15T08:51:54.572463",
"status": "completed"
},
"tags": []
},
"outputs": [],
"source": [
"def requirements_agent(client_brief):\n",
"    \"\"\"\n",
"    Use LLM (or mock) to extract structured fields from the brief.\n",
"    Returns dict with industry, goals, pain_points, deliverables (list), audience.\n",
"    \"\"\"\n",
"    prompt = f\"\"\"\n",
"    Extract these items from the client brief (as JSON):\n",
"    - industry\n",
"    - goals\n",
"    - pain_points\n",
"    - deliverables (list; up to 7 items)\n",
"    - target_audience\n",
"\n",
"    Client brief:\n",
"    {client_brief}\n",
"\n",
"    Return JSON only.\n",
"    \"\"\"\n",
"    raw = llm(prompt)\n",
"    # Try to parse JSON from LLM; if mock, create fallback structure\n",
"    try:\n",
"        # Attempt to find first '{' for JSON\n",
"        start = raw.find("{")\n",
"        if start != -1:\n",
"            data = json.loads(raw[start:])\n",
"            return data\n",
"    except Exception:\n",
"        pass\n",
"\n",
"    # Fallback (mock)\n",
"    return {\n",
"        \"industry\": \"Digital Marketing\", \n",
"        \"goals\": \"Increase leads and brand awareness by 50% in 6 months\", \n",
"        \"pain_points\": \"Low lead volume; inconsistent messaging\", \n",
"        \"deliverables\": [\"SEO optimization\", \"PPC campaigns\", \"Landing page\", \"Email automation\"], \n",
"        \"target_audience\": \"SMB owners and marketing managers\"\n"
}

```

```
"    }\n"]\n},\n{\n    "cell_type": "code",\n    "execution_count": 8,\n    "id": "7a88e7a2",\n    "metadata": {\n        "execution": {\n            "iopub.execute_input": "2025-11-15T08:51:54.594539Z",\n            "iopub.status.busy": "2025-11-15T08:51:54.594162Z",\n            "iopub.status.idle": "2025-11-15T08:51:54.599906Z",\n            "shell.execute_reply": "2025-11-15T08:51:54.598599Z"\n        },\n        "papermill": {\n            "duration": 0.01229,\n            "end_time": "2025-11-15T08:51:54.601895",\n            "exception": false,\n            "start_time": "2025-11-15T08:51:54.589605",\n            "status": "completed"\n        },\n        "tags": []\n    },\n    "outputs": [],\n    "source": [\n        "def research_agent(industry):\n            competitors = competitor_research(industry, keywords=\"digital marketing\")\n            # Summarize using LLM (mock if no key)\n            summary_prompt = f\"Summarize key insights from these\ncompetitors:\\n\\n{json.dumps(competitors, indent=2)}\\n\\nGive 5 bullet points.\\n\",\n            summary = llm(summary_prompt)\n            return {\"summary\": summary, \"competitors\": competitors}\n    ]\n},\n{\n    "cell_type": "code",\n    "execution_count": 9,\n    "id": "d9996e2f",\n    "metadata": {\n        "execution": {\n            "iopub.execute_input": "2025-11-15T08:51:54.611932Z",\n            "iopub.status.busy": "2025-11-15T08:51:54.610938Z",\n            "iopub.status.idle": "2025-11-15T08:51:54.616703Z",\n            "shell.execute_reply": "2025-11-15T08:51:54.615666Z"\n        },\n        "papermill": {\n            "duration": 0.012328,\n            "end_time": "2025-11-15T08:51:54.618157",\n            "exception": false,\n            "start_time": "2025-11-15T08:51:54.605829",\n            "status": "completed"\n        },\n        "tags": []\n    },\n    "outputs": [],\n    "source": [\n        "def pricing_agent(deliverables):\n            count = len(deliverables)\n            # complexity estimate heuristic: longer deliverable names => higher complexity\n            avg_len = sum(len(d) for d in deliverables) / max(1, count)\n            complexity = 1.0 + min(2.0, avg_len / 30.0)\n            pricing = calculate_pricing(count, complexity=complexity)\n            return pricing\n    ]\n},
```

```
{  
  "cell_type": "code",  
  "execution_count": 10,  
  "id": "9cebadd7",  
  "metadata": {  
    "execution": {  
      "iopub.execute_input": "2025-11-15T08:51:54.626569Z",  
      "iopub.status.busy": "2025-11-15T08:51:54.626211Z",  
      "iopub.status.idle": "2025-11-15T08:51:54.633885Z",  
      "shell.execute_reply": "2025-11-15T08:51:54.632901Z"  
    },  
    "papermill": {  
      "duration": 0.013735,  
      "end_time": "2025-11-15T08:51:54.635466",  
      "exception": false,  
      "start_time": "2025-11-15T08:51:54.621731",  
      "status": "completed"  
    },  
    "tags": []  
  },  
  "outputs": [],  
  "source": [  
    "def proposal_writer(parsed, research, pricing):\n",  
    "  \"\"\"\n",  
    "  Create a professional proposal body (string).\n",  
    "  \"\"\",  
    "  prompt = f\"\"\n",  
    "  Write a professional B2B proposal using the fields below.\n",  
    "  Include: Executive summary, Proposed Scope (with deliverables),\n",  
    "  Timeline (3 milestones), Pricing summary, ROI estimate, Risks & Next steps.\n",  
    "  \n",  
    "  Parsed Info:\n",  
    "  {json.dumps(parsed, indent=2)}\n",  
    "\n",  
    "  Research Summary:\n",  
    "  {research['summary']}\n",  
    "\n",  
    "  Competitors:\n",  
    "  {json.dumps(research['competitors']), indent=2})\n",  
    "\n",  
    "  Pricing:\n",  
    "  {json.dumps(pricing, indent=2)}\n",  
    "  \"\"\",  
    "  proposal_text = llm(prompt)\n",  
    "  # If mock output is short, augment with template\n",  
    "  if proposal_text.startswith(\"MOCK\"):\n",  
    "    # naive template build\n",  
    "    lines = []\n",  
    "    lines.append(f\"Executive Summary\\nWe propose a {parsed['industry']} engagement to  
achieve: {parsed['goals']}.\")\n",  
    "    lines.append(\"\\nProposed Scope:\")\n",  
    "    for d in parsed['deliverables']:\n",  
    "      lines.append(f\"- {d}\")\n",  
    "    lines.append(\"\\nTimeline:\")\n",  
    "    lines.append(\"1. Discovery (2 weeks)\\n\",  
    "    lines.append(\"2. Implementation (8 weeks)\\n\",  
    "    lines.append(\"3. Optimization (4 weeks)\\n\",  
    "    lines.append(\"\\nPricing Summary:\")\n",  
    "    lines.append(f\"Total estimated price: ${pricing['estimated_price']}\\n\",  
    "    lines.append(f\"Expected ROI (estimated): ${pricing['expected_roi_value']}\\n\",  
    "    lines.append(\"\\nRisks & Next Steps:\\n\",  
    "    lines.append(\"- Risk: availability of client assets\\n\",  
    "    lines.append(\"- Next step: sign SOW and schedule kickoff\\n\",  
    "    proposal_text = \"\\n\".join(lines)\n",  
    "    return proposal_text\n"
```

```
        ],
    },
    {
        "cell_type": "code",
        "execution_count": 11,
        "id": "6f74e7d8",
        "metadata": {
            "execution": {
                "iopub.execute_input": "2025-11-15T08:51:54.644107Z",
                "iopub.status.busy": "2025-11-15T08:51:54.643478Z",
                "iopub.status.idle": "2025-11-15T08:51:54.649832Z",
                "shell.execute_reply": "2025-11-15T08:51:54.648907Z"
            },
            "papermill": {
                "duration": 0.012425,
                "end_time": "2025-11-15T08:51:54.651448",
                "exception": false,
                "start_time": "2025-11-15T08:51:54.639023",
                "status": "completed"
            },
            "tags": []
        },
        "outputs": [],
        "source": [
            "def coordinator(client_brief, client_name=\"Client\"):\n",
            "    print(\"== STEP 1: Parse requirements ==\")\n",
            "    parsed = requirements_agent(client_brief)\n",
            "    print(\"Parsed:\", parsed)\n",
            "\n",
            "    print(\"\\n== STEP 2: Research competitors ==\")\n",
            "    research = research_agent(parsed.get(\"industry\", \"\"))\n",
            "    print(\"Research summary (truncated):\", research['summary'][:300])\n",
            "\n",
            "    print(\"\\n== STEP 3: Pricing & ROI ==\")\n",
            "    pricing = pricing_agent(parsed.get(\"deliverables\", []))\n",
            "    print(\"Pricing:\", pricing)\n",
            "\n",
            "    print(\"\\n== STEP 4: Write proposal ==\")\n",
            "    proposal_text = proposal_writer(parsed, research, pricing)\n",
            "    print(\"Proposal length:\", len(proposal_text))\n",
            "\n",
            "    print(\"\\n== STEP 5: Generate PDF ==\")\n",
            "    pdf_path = generate_pdf(client_name, proposal_text)\n",
            "\n",
            "    # Return structured result\n",
            "    return {\n",
            "        \"parsed\": parsed,\n",
            "        \"research\": research,\n",
            "        \"pricing\": pricing,\n",
            "        \"proposal_text\": proposal_text,\n",
            "        \"pdf_path\": pdf_path\n",
            "    }\n"
        ]
    },
    {
        "cell_type": "code",
        "execution_count": 12,
        "id": "7a11ce74",
        "metadata": {
            "execution": {
                "iopub.execute_input": "2025-11-15T08:51:54.659968Z",
                "iopub.status.busy": "2025-11-15T08:51:54.659619Z",
                "iopub.status.idle": "2025-11-15T08:51:54.903690Z",
                "shell.execute_reply": "2025-11-15T08:51:54.902671Z"
            },
        }
    }
```

```
"papermill": {
    "duration": 0.250422,
    "end_time": "2025-11-15T08:51:54.905375",
    "exception": false,
    "start_time": "2025-11-15T08:51:54.654953",
    "status": "completed"
},
"tags": [],
},
"outputs": [
{
    "name": "stdout",
    "output_type": "stream",
    "text": [
        "==== STEP 1: Parse requirements ===\n",
        "Parsed: {'industry': 'Digital Marketing', 'goals': 'Increase leads and brand awareness by 50% in 6 months', 'pain_points': 'Low lead volume; inconsistent messaging', 'deliverables': ['SEO optimization', 'PPC campaigns', 'Landing page', 'Email automation'], 'target_audience': 'SMB owners and marketing managers'}\n",
        "\n",
        "==== STEP 2: Research competitors ===\n",
        "🔍 Running competitor research...\n"
    ]
},
{
    "name": "stderr",
    "output_type": "stream",
    "text": [
        "/tmp/ipykernel_13/386215183.py:9: RuntimeWarning: This package (`duckduckgo_search`) has been renamed to `ddgs`! Use `pip install ddgs` instead.\n",
        "    with DDGS() as ddgs:\n"
    ]
},
{
    "name": "stdout",
    "output_type": "stream",
    "text": [
        "⚠️ Research tool fallback (mock). No results\n",
        "Research summary (truncated): MOCK SUMMARY: Summarize key insights from these competitors:\n...",
        "\n",
        "(Use real API key to enable full LLM responses.)\n",
        "\n",
        "==== STEP 3: Pricing & ROI ===\n",
        "Pricing: {'estimated_price': 885.0, 'expected_roi_value': 2259.96, 'details': {'base_price': 150, 'complexity': 1.475, 'deliverable_count': 4}}\n",
        "\n",
        "==== STEP 4: Write proposal ===\n",
        "Proposal length: 485\n",
        "\n",
        "==== STEP 5: Generate PDF ===\n",
        "📄 PDF generated at: /kaggle/working/BrightShop_proposal.pdf\n"
    ]
},
{
    "data": {
        "text/plain": [
            "{'parsed': {'industry': 'Digital Marketing',\n            'goals': 'Increase leads and brand awareness by 50% in 6 months',\n            'pain_points': 'Low lead volume; inconsistent messaging',\n            'deliverables': ['SEO optimization',\n                'PPC campaigns',\n                'Landing page',\n                'Email automation'],\n            'target_audience': 'SMB owners and marketing managers'}}\n"
        ]
    }
}
```

```
"research": {"summary": "MOCK SUMMARY: Summarize key insights from these competitors:\n...\\n\\n(Use real API key to enable full LLM responses.)",\n    "competitors": ["Digital Marketing Competitor A - https://example.com/a",\n        "Digital Marketing Competitor B - https://example.com/b",\n        "Digital Marketing Competitor C - https://example.com/c"]},\n    "pricing": {"estimated_price": 885.0,\n        "expected_roi_value": 2259.96},\n    "details": {"base_price": 150, "complexity": 1.475, "deliverable_count": 4}},\n    "proposal_text": "Executive Summary\\nWe propose a Digital Marketing engagement to achieve:\nIncrease leads and brand awareness by 50% in 6 months.\\n\\nProposed Scope:\\n- SEO optimization\\n-\nPPC campaigns\\n- Landing page\\n- Email automation\\n\\nTimeline:\\n1. Discovery (2 weeks)\\n2.\nImplementation (8 weeks)\\n3. Optimization (4 weeks)\\n\\nPricing Summary:\\nTotal estimated price:\n$885.0\\nExpected ROI (estimated): $2259.96\\n\\nRisks & Next Steps:\\n- Risk: availability of client\nassets\\n- Next step: sign SOW and schedule kickoff",\n    "pdf_path": "/kaggle/working/BrightShop_proposal.pdf"}\n]\n},\n"execution_count": 12,\n"metadata": {},\n"output_type": "execute_result"
}\n],\n"source": [\n    "# Example client brief – edit this to try different inputs\\n",\n    "client_brief = \"\"\"\n",\n    "We are BrightShop, an e-commerce startup selling eco-friendly home goods.\\n",\n    "Goal: Increase monthly leads by 50% and double online conversion in 6 months.\\n",\n    "Budget range: $5,000 - $12,000 for initial 3-month engagement.\\n",\n    "We need SEO, Google Ads management, landing page optimization, and email flows.\\n",\n    "Target audience: urban 25-45 eco-conscious buyers.\\n",\n    "\"\"\"",\n    "\n",\n    "result = coordinator(client_brief, client_name=\"BrightShop\")\\n",\n    "result # shows dict output; PDF path is in result['pdf_path']\\n"
]\n}\n],\n"metadata": {\n    "kaggle": {\n        "accelerator": "none",\n        "dataSources": [],\n        "dockerImageVersionId": 31192,\n        "isGpuEnabled": false,\n        "isInternetEnabled": true,\n        "language": "python",\n        "sourceType": "notebook"
},\n    "kernelspec": {\n        "display_name": "Python 3",\n        "language": "python",\n        "name": "python3"
},\n    "language_info": {\n        "codemirror_mode": {\n            "name": "ipython",\n            "version": 3
},\n        "file_extension": ".py",\n        "mimetype": "text/x-python",\n        "name": "python",\n        "nbconvert_exporter": "python",\n        "pygments_lexer": "ipython3",\n        "version": "3.11.13"
},\n    "papermill": {
}
```

```
"default_parameters": {},  
"duration": 16.414755,  
"end_time": "2025-11-15T08:51:55.328686",  
"environment_variables": {},  
"exception": null,  
"input_path": "__notebook__.ipynb",  
"output_path": "__notebook__.ipynb",  
"parameters": {},  
"start_time": "2025-11-15T08:51:38.913931",  
"version": "2.6.0"  
}  
},  
"nbformat": 4,  
"nbformat_minor": 5  
}
```