Main Flask App (app.py)

```
import os
import shutil
from flask import Flask, render_template, request, jsonify
from werkzeug.utils import secure_filename

# Text extraction deps
from pdfminer.high_level import extract_text as pdf_extract_text
from pdfminer.layout import LAParams
from PIL import Image, ImageOps, ImageFilter
import pytesseract

ALLOWED_EXTENSIONS = {"pdf", "png", "jpg", "jpeg", "webp", "tif", "tiff", "bmp"}

def allowed_file(filename: str) -> bool:
    return "." in filename and filename.rsplit(".", 1)[1].lower() in ALLOWED_EXTENSIONS

def extract_text_from_pdf(path: str) -> str:
    laparams = LAParams()  # good defaults to keep reasonable layout
    text = pdf_extract_text(path, laparams=laparams) or ""
    return text.strip()

def preprocess_image_for_ocr(img: Image.Image) -> Image.Image:
    # Convert to grayscale, auto-contrast, denoise a touch, sharpen slightly
    gray = ImageOps.grayscale(img)
    gray = ImageOps.autocontrast(gray)
    gray = gray.filter(ImageFilter.MedianFilter(size=3))
    gray = gray.filter(ImageFilter.UnsharpMask(radius=1, percent=100, threshold=3))
    return gray

def extract_text_from_image(path: str) -> str:
    with Image.open(path) as im:
        im = preprocess_image_for_ocr(im)
        # You can tune psm/oem if needed; these are reasonable general defaults
        config = "--oem 3 --psm 3"
        text = pytesseract.image_to_string(im, config=config) or ""
        return text.strip()

def summarize_text_stats(text: str) -> dict:
    words = [w for w in text.replace("\n", " ").split(" ") if w.strip()]
    hashtags = [w for w in words if w.startswith("#")]
    mentions = [w for w in words if w.startswith("@")]
    urls = [w for w in words if w.startswith("http://") or w.startswith("https://")]
    return {
        "chars": len(text),
        "words": len(words),
        "hashtags": len(hashtags),
        "mentions": len(mentions),
        "urls": len(urls),
    }

def quick_engagement_suggestions(text: str) -> list:
    # Lightweight heuristic tips (optional value-add)
    suggestions = []
    if len(text) < 60:
        suggestions.append("Your post is very short—consider adding a detail or hook.")
    if len(text) > 2200:
        suggestions.append("Your post is quite long—consider a TL;DR or break into a thread.")
    if "#" not in text:
        suggestions.append("No hashtags detected—add 1–3 relevant hashtags for discovery.")
    if "http://" in text or "https://" in text:
        suggestions.append("You included a link—add a short call-to-action explaining why to click."
    if "?" not in text and "!" not in text:
        suggestions.append("Add a question or strong call-to-action to invite replies.")
    if not suggestions:
        suggestions.append("Looks good! Consider posting at your audience’s peak time for best reach
    return suggestions

def create_app():
    app = Flask(__name__)
    app.config["MAX_CONTENT_LENGTH"] = 25 * 1024 * 1024  # 25 MB
    app.config["UPLOAD_FOLDER"] = os.path.join(app.root_path, "uploads")
    os.makedirs(app.config["UPLOAD_FOLDER"], exist_ok=True)

    @app.route("/", methods=["GET"])
    def index():
        return render_template("index.html")

    @app.route("/analyze", methods=["POST"])
    def analyze():
        # Basic validation
        if "files" not in request.files:
            return jsonify({"ok": False, "error": "No files part in request."}), 400

        files = request.files.getlist("files")
        if not files:
            return jsonify({"ok": False, "error": "No files provided."}), 400

        results = []
        temp_dir = app.config["UPLOAD_FOLDER"]

        for f in files:
            if not f.filename:
                continue
            filename = secure_filename(f.filename)
            if not allowed_file(filename):
                results.append({
                    "filename": filename or "unknown",
                    "ok": False,
                    "error": "Unsupported file type. Allowed: pdf, png, jpg, jpeg, webp, tif, tiff,
                })
                continue

            save_path = os.path.join(temp_dir, filename)
            f.save(save_path)

            ext = filename.rsplit(".", 1)[1].lower()
            try:
                if ext == "pdf":
                    text = extract_text_from_pdf(save_path)
                    kind = "pdf"
                else:
                    text = extract_text_from_image(save_path)
                    kind = "image"

                stats = summarize_text_stats(text)
                suggestions = quick_engagement_suggestions(text)

                results.append({
                    "filename": filename,
                    "ok": True,
                    "type": kind,
                    "text": text,
                    "stats": stats,
                    "suggestions": suggestions,
                })
            except Exception as e:
                results.append({
                    "filename": filename,
                    "ok": False,
                    "error": f"Failed to process: {str(e)}"
                })
            finally:
                # Clean up individual saved file to keep container/disk tidy
                try:
                    os.remove(save_path)
                except Exception:
                    pass

        return jsonify({"ok": True, "results": results})

    @app.errorhandler(413)
    def file_too_large(_e):
        return jsonify({"ok": False, "error": "File too large. Max 25 MB."}), 413

    return app

app = create_app()

if __name__ == "__main__":
    # For local dev
    app.run(host="0.0.0.0", port=int(os.environ.get("PORT", 5000)), debug=True)
```

Frontend Template (index.html)

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Social Media Content Analyzer</title>
  <link rel="stylesheet" href="/static/styles.css" />
</head>
<body>
  <header>
    <h1>Social Media Content Analyzer</h1>
    <p class="muted">Upload PDFs or images (scanned posts) to extract text and get quick tips.</p>
  </header>

  <main>
    <form id="upload-form" class="card" onsubmit="return false;">
      <div id="dropzone" class="dropzone">
        <input id="file-input" type="file" name="files" accept=".pdf,image/*" multiple />
        <p>Drag &amp; drop files here or <button type="button" id="browse-btn" class="linklike">brows
        <small class="muted">Allowed: PDF, PNG, JPG, JPEG, WEBP, TIFF, BMP (max 25 MB each)</small>
      </div>

      <div id="file-list" class="file-list hidden"></div>

      <div class="actions">
        <button id="analyze-btn" class="primary" disabled>Analyze</button>
        <button id="clear-btn" type="button" class="ghost">Clear</button>
      </div>
    </form>

    <section id="results" class="results"></section>
  </main>

  <div id="loading" class="loading hidden">
    <div class="spinner"></div>
    <p>Extracting text…</p>
  </div>

  <footer>
    <p class="muted">Built with Flask, pdfminer.six and Tesseract OCR.</p>
  </footer>

  <script src="/static/app.js"></script>
</body>
</html>
```

Stylesheet (styles.css)

```css
:root {
  --bg: #0b0d10;
  --panel: #13161a;
  --text: #e8eaf0;
  --muted: #96a0b5;
  --accent: #7c5cff;
  --accent-2: #00d4ff;
  --border: #232832;
}

* { box-sizing: border-box; }
html, body { height: 100%; }
body {
  margin: 0;
  font-family: system-ui, -apple-system, Segoe UI, Roboto, Ubuntu, Cantarell, Noto Sans, "Helvetica
  background: radial-gradient(1200px 800px at 10% -10%, rgba(124,92,255,0.12), transparent),
              radial-gradient(800px 600px at 110% 10%, rgba(0,212,255,0.12), transparent),
              var(--bg);
  color: var(--text);
}

header, footer {
  text-align: center;
  padding: 24px 16px;
}
```

```
h1 { margin: 0 0 8px; font-weight: 700; }
.muted { color: var(--muted); }

main {
  max-width: 920px;
  margin: 0 auto;
  padding: 16px;
}

.card {
  background: linear-gradient(180deg, rgba(255,255,255,0.02), rgba(255,255,255,0.01));
  border: 1px solid var(--border);
  border-radius: 16px;
  padding: 16px;
  box-shadow: 0 10px 30px rgba(0,0,0,0.25);
}

.dropzone {
  position: relative;
  border: 2px dashed var(--border);
  border-radius: 14px;
  padding: 28px;
  text-align: center;
  outline: none;
  transition: border-color 0.2s, background 0.2s;
}

.dropzone.dragover {
  border-color: var(--accent);
  background: rgba(124,92,255,0.06);
}

#file-input {
  position: absolute;
  inset: 0;
  width: 100%;
  height: 100%;
  opacity: 0;
  cursor: pointer;
}

.linklike {
  background: none;
  border: none;
  padding: 0;
  color: var(--accent-2);
  text-decoration: underline;
  cursor: pointer;
  font: inherit;
}

.file-list {
  margin-top: 12px;
  display: grid;
  gap: 8px;
}

.file-pill {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background: var(--panel);
  border: 1px solid var(--border);
  padding: 10px 12px;
  border-radius: 12px;
}

.actions {
  display: flex;
  gap: 10px;
  margin-top: 16px;
}

button.primary {
  background: linear-gradient(90deg, var(--accent), var(--accent-2));
  color: #0b0d10;
  padding: 10px 14px;
  border: none;
  border-radius: 12px;
  font-weight: 700;
  cursor: pointer;
}

button.ghost {
  background: transparent;
  border: 1px solid var(--border);
  color: var(--text);
  padding: 10px 14px;
  border-radius: 12px;
  cursor: pointer;
}

.hidden { display: none; }

.loading {
  position: fixed;
  inset: 0;
  display: grid;
  place-items: center;
  background: rgba(0,0,0,0.45);
  backdrop-filter: blur(3px);
  z-index: 20;
}

.spinner {
  width: 44px;
  height: 44px;
  border-radius: 50%;
  border: 4px solid rgba(255,255,255,0.18);
  border-top-color: white;
  animation: spin 0.9s linear infinite;
  margin: 0 auto 12px;
}

@keyframes spin { to { transform: rotate(360deg); } }

.results {
  display: grid;
  gap: 16px;
  margin: 18px 0 32px;
}

.result-card {
  background: var(--panel);
  border: 1px solid var(--border);
  border-radius: 16px;
  padding: 16px;
}

.result-head {
  display: flex;
  align-items: center;
  gap: 8px;
  justify-content: space-between;
}

.badge {
  background: #1b1f27;
  border: 1px solid var(--border);
  padding: 4px 8px;
  border-radius: 999px;
  font-size: 12px;
  color: var(--muted);
}

pre {
  white-space: pre-wrap;
  background: #0f1217;
  border: 1px solid var(--border);
  border-radius: 12px;
  padding: 12px;
  overflow-x: auto;
}
.kv {
  display: flex;
  gap: 12px;
  flex-wrap: wrap;
  margin: 8px 0 0;
}

.kv span {
  background: #0f1217;
  border: 1px solid var(--border);
  padding: 6px 10px;
  border-radius: 999px;
  font-size: 12px;
}

.copy-btn {
  border: 1px solid var(--border);
  background: transparent;
  color: var(--text);
  border-radius: 10px;
  padding: 6px 10px;
  cursor: pointer;
}
```

Client-side JavaScript (app.js)

```
const dropzone = document.getElementById("dropzone");
const fileInput = document.getElementById("file-input");
const browseBtn = document.getElementById("browse-btn");
const analyzeBtn = document.getElementById("analyze-btn");
const clearBtn = document.getElementById("clear-btn");
const fileList = document.getElementById("file-list");
const loading = document.getElementById("loading");
const results = document.getElementById("results");

let filesSelected = [];

function refreshFileList() {
  fileList.innerHTML = "";
  if (filesSelected.length === 0) {
    fileList.classList.add("hidden");
    analyzeBtn.disabled = true;
    return;
  }
  fileList.classList.remove("hidden");
  analyzeBtn.disabled = false;
  filesSelected.forEach((f, idx) => {
    const pill = document.createElement("div");
    pill.className = "file-pill";
    pill.innerHTML = `<span>${f.name}</span><button class="ghost" data-idx="${idx}">Remove</button>`
    pill.querySelector("button").addEventListener("click", (e) => {
      const i = Number(e.currentTarget.dataset.idx);
      filesSelected.splice(i, 1);
      refreshFileList();
    });
    fileList.appendChild(pill);
  });
}

function addFiles(fs) {
  for (const f of fs) {
    filesSelected.push(f);
  }
  refreshFileList();
}

browseBtn.addEventListener("click", () => fileInput.click());
fileInput.addEventListener("change", (e) => addFiles(e.target.files));

dropzone.addEventListener("dragover", (e) => {
  e.preventDefault();
  dropzone.classList.add("dragover");
});
dropzone.addEventListener("dragleave", () => dropzone.classList.remove("dragover"));
dropzone.addEventListener("drop", (e) => {
  e.preventDefault();
  dropzone.classList.remove("dragover");
  if (e.dataTransfer?.files?.length) {
    addFiles(e.dataTransfer.files);
  }
});

clearBtn.addEventListener("click", () => {
  filesSelected = [];
  refreshFileList();
  results.innerHTML = "";
});

analyzeBtn.addEventListener("click", async () => {
  const fd = new FormData();
  filesSelected.forEach((f) => fd.append("files", f, f.name));

  loading.classList.remove("hidden");
  analyzeBtn.disabled = true;

  try {
    const res = await fetch("/analyze", {
      method: "POST",
      body: fd
    });
    const data = await res.json();
    results.innerHTML = "";
    if (!data.ok) {
      const div = document.createElement("div");
      div.className = "result-card";
      div.innerHTML = `<p><strong>Error:</strong> ${data.error || "Unknown error"}</p>`;
      results.appendChild(div);
    } else {
      data.results.forEach(renderResult);
    }
  } catch (err) {
    const div = document.createElement("div");
    div.className = "result-card";
    div.innerHTML = `<p><strong>Network error:</strong> ${err.message}</p>`;
    results.appendChild(div);
  } finally {
    loading.classList.add("hidden");
    analyzeBtn.disabled = false;
  }
});

function renderResult(r) {
  const wrap = document.createElement("div");
  wrap.className = "result-card";

  const head = document.createElement("div");
  head.className = "result-head";
  head.innerHTML = `
    <div>
      <strong>${r.filename}</strong>
      <span class="badge">${r.ok ? r.type.toUpperCase() : "ERROR"}</span>
    </div>
    <div>
      <button class="copy-btn">Copy text</button>
    </div>
  `;

  const pre = document.createElement("pre");
  pre.textContent = r.text || r.error || "";

  const kv = document.createElement("div");
  kv.className = "kv";
  if (r.ok) {
    kv.innerHTML = `
      <span>Words: ${r.stats.words}</span>
      <span>Chars: ${r.stats.chars}</span>
      <span>#Hashtags: ${r.stats.hashtags}</span>
      <span>@Mentions: ${r.stats.mentions}</span>
      <span>Links: ${r.stats.urls}</span>
    `;
  }

  const tips = document.createElement("ul");
  if (r.ok && Array.isArray(r.suggestions)) {
    for (const s of r.suggestions) {
      const li = document.createElement("li");
      li.textContent = s;
      tips.appendChild(li);
    }
  }

  wrap.appendChild(head);
  wrap.appendChild(kv);
  wrap.appendChild(pre);
  if (tips.children.length) {
    const h = document.createElement("p");
    h.innerHTML = "<strong>Quick tips:</strong>";
    wrap.appendChild(h);
    wrap.appendChild(tips);
  }

  head.querySelector(".copy-btn").addEventListener("click", async () => {
    try {
      await navigator.clipboard.writeText(r.text || "");
      head.querySelector(".copy-btn").textContent = "Copied!";
      setTimeout(() => (head.querySelector(".copy-btn").textContent = "Copy text"), 1200);
    } catch {}
  });

  results.appendChild(wrap);
}
```

Dependencies (requirements.txt)

```
Flask==3.0.3
pdfminer.six==20231228
pytesseract==0.3.13
Pillow==10.3.0
Werkzeug==3.0.3
gunicorn==22.0.0
```

Documentation (README.md)

~~~
# Social Media Content Analyzer (Flask)

A small Flask web app that lets you **upload PDFs and images**, extracts text using **pdfminer.six**

## Features
- Drag & drop or file picker uploads (multiple files).
- PDF text extraction that keeps reasonable layout via `pdfminer.six`.
- OCR for images (PNG, JPG/JPEG, WEBP, TIFF, BMP) via `pytesseract`.
- Loading overlay and clear error messages.
- Word/character counts, hashtags/mentions/links detection, and quick tips.

## Prerequisites
- **Python 3.10+**
- **Tesseract OCR** binary installed on your system:
  - Windows: Download installer from https://github.com/tesseract-ocr/tesseract
  - macOS: `brew install tesseract`
  - Ubuntu/Debian: `sudo apt update && sudo apt install -y tesseract-ocr`
- Optionally set `TESSDATA_PREFIX` if you installed language packs elsewhere.

If `pytesseract` can't find Tesseract, set the path in code:
```python
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
```

## Setup
```bash
python -m venv .venv
source .venv/bin/activate  # Windows: .venv\Scripts\activate
pip install -r requirements.txt
python app.py
```
Open http://localhost:5000
~~~

## Deploy (Gunicorn example)
```bash
gunicorn -w 2 -b 0.0.0.0:5000 app:app
```

## Notes
- Max upload size is 25 MB per file (configurable).
- PDFs are parsed with `pdfminer.six`. For image■heavy PDFs, consider adding pdf2image + OCR (not re
- Basic suggestions are heuristic; tailor for your use case.