```
# To be installed in the command prompt along with Tesseract OCR
pip install pillow pymupdf translatepy pytesseract numpy


from PIL import Image, ImageDraw, ImageFont, ImageOps, ImageFilter
import textwrap
import fitz  # PyMuPDF
import os
from translatepy import Translator
import pytesseract
from glob import glob
import random
import pathlib
import numpy as np

# Tesseract OCR path
pytesseract.pytesseract.tesseract_cmd = r"C:\\Program Files\\Tesseract-OCR\\tesseract.exe"

# Page layout
PAGE_WIDTH = 2230
PAGE_HEIGHT = 2849
MARGIN_LEFT = 170
MARGIN_RIGHT = 100
MARGIN_TOP = 340
MARGIN_BOTTOM = 200
LINE_SPACING = 10
FONT_SCALE = 0.9

# Font and lines
FONT_SIZE = int(((PAGE_HEIGHT - MARGIN_TOP - MARGIN_BOTTOM) // 40) * FONT_SCALE)
LINES_PER_PAGE = (PAGE_HEIGHT - MARGIN_TOP - MARGIN_BOTTOM) // (FONT_SIZE + LINE_SPACING)
CHARS_PER_LINE = None

# Ink color presets
COLOR_PRESETS = {
    "black": (0, 0, 0),
    "red": (255, 0, 0),
    "green": (0, 128, 0),
    "blue": (0, 0, 255),
    "darkblue": (0, 0, 139),
    "purple": (128, 0, 128),
    "brown": (139, 69, 19),
    "orange": (255, 165, 0),
    "grey": (128, 128, 128),
    "pink": (255, 20, 147)
}

# Handwriting effects
JITTER_ENABLED = False
CHARACTER_JITTER = True
MAX_X_JITTER = 3
MAX_Y_JITTER = 2
MAX_FADE_VARIATION = 40
PRESSURE_SIMULATION = False
SMUDGE_ENABLED = False
SMUDGE_RADIUS = 1

# Handwriting effect toggles
```

```python
def get_effect_toggles():
    global JITTER_ENABLED, PRESSURE_SIMULATION, SMUDGE_ENABLED
    print("\n=== Handwriting Effects Configuration ===")
    if input("Enable handwriting jitter/fading? (Y/N): ").strip().lower() == 'y':
        JITTER_ENABLED = True
        print("Jitter and fading enabled")
    if input("Enable pressure variation simulation? (Y/N): ").strip().lower() == 'y':
        PRESSURE_SIMULATION = True
        print("Pressure simulation enabled")
    if input("Enable smudging effect? (Y/N): ").strip().lower() == 'y':
        SMUDGE_ENABLED = True
        print("Smudging effect enabled")

# List .ttf fonts in current directory
def list_fonts():
    return [f for f in os.listdir() if f.lower().endswith(".ttf")]

# Generate preview images for each font
def generate_font_previews(fonts):
    print("\nGenerating font previews...")
    for font_file in fonts:
        try:
            text = ("ನಮಸ್ಕಾರ! ನಾನು ಪೈತಾನ್ನಲ್ಲಿ ಬರಹ ಉಂಟುಮಾಡಿತ್ತೆನೆ."
                    if any(k in font_file.lower() for k in ["kannada", "lohit", "nudi", "baraha", "tunga", "baloo", "akshar"])
                    else "The quick brown fox jumps over the lazy dog.")
            font = ImageFont.truetype(font_file, 40)
            w, h = font.getbbox(text)[2:4]
            img = Image.new("RGB", (w+80, max(120, h+80)), "white")
            ImageDraw.Draw(img).text((40, 40), text, font=font, fill=(0,0,0))
            preview_filename = f"preview_{font_file[:-4]}.png"
            img.save(preview_filename)
            print(f"{font_file} → {preview_filename}")
            os.startfile(preview_filename)
        except Exception as e:
            print(f"Could not preview {font_file}: {e}")

# Font selection prompt
def get_font_choice():
    fonts = list_fonts()
    if not fonts:
        print("No .ttf fonts found in this folder.")
        exit()
    generate_font_previews(fonts)
    print("\nAvailable Fonts:")
    for i, font in enumerate(fonts, 1):
        print(f"  {i}. {font} (See preview_{font.replace('.ttf', '')}.png)")
    try:
        choice = int(input("\nChoose a font by number: "))
        if 1 <= choice <= len(fonts):
            return fonts[choice - 1]
    except:
        pass
    print("Invalid font choice. Exiting.")
    exit()

# Display ink color presets
def show_color_presets():
    print("\nCommon Ink Colors:")
```

```
    for name, rgb in COLOR_PRESETS.items():
        print(f"  {name.capitalize():<10} - RGB: {rgb}")

# Color selection prompt
def get_rgb_input():
    show_color_presets()
    print("\nType a color name from above or enter custom RGB values")
    while True:
        choice = input("Enter color name or 'custom': ").strip().lower()
        if choice in COLOR_PRESETS:
            return COLOR_PRESETS[choice]
        elif choice == "custom":
            try:
                r = int(input("R (0-255): "))
                g = int(input("G (0-255): "))
                b = int(input("B (0-255): "))
                if all(0 <= val <= 255 for val in (r, g, b)):
                    return (r, g, b)
            except ValueError:
                pass
        print("Invalid input. Try again.")

# File path and extension validation
def get_valid_file(prompt, extensions):
    while True:
        path = input(prompt).strip()
        file = pathlib.Path(path)
        if file.is_file() and file.suffix.lower() in extensions:
            return str(file)
        print("Invalid file or unsupported extension.")

# Image preprocessing for OCR
def preprocess_image(img_path):
    img = Image.open(img_path).convert("L")
    img = ImageOps.invert(img)
    img = img.filter(ImageFilter.MedianFilter())
    img = ImageOps.autocontrast(img)
    return img

# User text input source selection
def get_text():
    print("\nChoose input method:")
    print("1. Type manually")
    print("2. Read from TXT file")
    print("3. Read from PDF file")
    print("4. Extract from handwritten image (any format)")
    choice = input("Enter 1/2/3/4: ").strip()
    if choice == "1":
        return input("\nEnter your handwritten text:\n"), "typed"
    elif choice == "2":
        filename = get_valid_file("Enter TXT filename: ", [".txt"])
        with open(filename, "r", encoding="utf-8") as f:
            return f.read(), "txt"
    elif choice == "3":
        pdf_path = get_valid_file("Enter PDF filename: ", [".pdf"])
        doc = fitz.open(pdf_path)
        text = "\n".join([page.get_text() for page in doc])
        return text, "pdf"
    elif choice == "4":
```

```
        image_path = get_valid_file("Enter the image filename: ", [".jpg", ".jpeg", ".png", ".bmp"])
        print(f"Using OCR on {image_path}...")
        preprocessed_img = preprocess_image(image_path)
        text = pytesseract.image_to_string(preprocessed_img, lang='eng', config='--oem 3 --psm 6')
        return text, "image"
    return "", "invalid"

# English to Kannada translation
def translate_english_to_kannada(text):
    try:
        return Translator().translate(text, "Kannada").result
    except:
        return text

# Pixel-based text wrapping for book layout
def book_style_wrap(text, font, max_width):
    paragraphs = text.split('\n\n')
    all_lines = []
    for paragraph in paragraphs:
        if not paragraph.strip():
            all_lines.append("")
            continue
        words = paragraph.split()
        if not words:
            continue
        lines = []
        current_line = ""
        for word in words:
            test_line = current_line + (" " if current_line else "") + word
            test_width = font.getbbox(test_line)[2] - font.getbbox(test_line)[0]
            if test_width <= max_width:
                current_line = test_line
            else:
                if current_line:
                    lines.append(current_line)
                    current_line = word
                else:
                    if font.getbbox(word)[2] - font.getbbox(word)[0] > max_width:
                        char_line = ""
                        for char in word:
                            test_char_line = char_line + char
                            if font.getbbox(test_char_line)[2] - font.getbbox(test_char_line)[0] <= max_width:
                                char_line = test_char_line
                            else:
                                if char_line:
                                    lines.append(char_line)
                                char_line = char
                        if char_line:
                            current_line = char_line
                    else:
                        current_line = word
        if current_line:
            lines.append(current_line)
        all_lines.extend(lines)
        if paragraph != paragraphs[-1]:
            all_lines.append("")
    return all_lines
```

```python

# Adjust ink color intensity for pressure effect
def apply_pressure_effect(color, pressure_factor):
    return tuple(int(c * pressure_factor) for c in color)

# Local smudge effect at (x, y)
def apply_smudge_effect(image, x, y, radius=1):
    if radius <= 0:
        return image
    try:
        crop_box = (max(0, x-radius), max(0, y-radius),
                    min(image.width, x+radius*2), min(image.height, y+radius*2))
        if crop_box[2] > crop_box[0] and crop_box[3] > crop_box[1]:
            cropped = image.crop(crop_box)
            blurred = cropped.filter(ImageFilter.GaussianBlur(radius=0.5))
            image.paste(blurred, crop_box)
    except:
        pass
    return image

# Render one page image with handwriting effects
def generate_page(lines, page_num, font_path, ink_color):
    try:
        bg = Image.open("background.png").convert("RGB")
        if bg.size != (PAGE_WIDTH, PAGE_HEIGHT):
            bg = bg.resize((PAGE_WIDTH, PAGE_HEIGHT), Image.LANCZOS)
    except:
        bg = Image.new("RGB", (PAGE_WIDTH, PAGE_HEIGHT), color="white")
    draw = ImageDraw.Draw(bg)
    font = ImageFont.truetype(font_path, FONT_SIZE)
    x, y = MARGIN_LEFT, MARGIN_TOP
    for line in lines:
        if not line:
            y += FONT_SIZE + LINE_SPACING
            continue
        if JITTER_ENABLED and CHARACTER_JITTER:
            curr_x = x
            for char in line:
                jitter_x = random.randint(-MAX_X_JITTER, MAX_X_JITTER)
                jitter_y = random.randint(-MAX_Y_JITTER, MAX_Y_JITTER)
                fade = random.randint(-MAX_FADE_VARIATION, 0)
                color = tuple(max(0, min(255, c + fade)) for c in ink_color)
                if PRESSURE_SIMULATION:
                    pressure = random.uniform(0.7, 1.0)
                    color = apply_pressure_effect(color, pressure)
                char_x = curr_x + jitter_x
                char_y = y + jitter_y
                draw.text((char_x, char_y), char, fill=color, font=font)
                if SMUDGE_ENABLED:
                    bg = apply_smudge_effect(bg, char_x, char_y, SMUDGE_RADIUS)
                    draw = ImageDraw.Draw(bg)
                char_width = font.getbbox(char)[2] - font.getbbox(char)[0]
                curr_x += char_width
        else:
            color = ink_color
            if PRESSURE_SIMULATION:
                pressure = random.uniform(0.8, 1.0)
                color = apply_pressure_effect(color, pressure)
            draw.text((x, y), line, fill=color, font=font)
            if SMUDGE_ENABLED:
```

```
                bg = apply_smudge_effect(bg, x, y, SMUDGE_RADIUS)
                draw = ImageDraw.Draw(bg)
        y += FONT_SIZE + LINE_SPACING
    bg.save(f"handwriting_page{page_num}.png")
    print(f"Saved page {page_num}")

# Combine page images into PDF
def combine_images_to_pdf():
    print("\nCombining pages into PDF...")
    images = [Image.open(f).convert("RGB") for f in sorted(glob("handwriting_page*.png"))]
    if images:
        images[0].save("handwriting_output.pdf", save_all=True, append_images=images[1:])
        print("PDF saved as handwriting_output.pdf")
    else:
        print("No pages found to combine")

# OCR extraction from generated images
def extract_text_from_images():
    print("\nExtracting text from generated pages...")
    result = ""
    for img_file in sorted(glob("handwriting_page*.png")):
        try:
            text = pytesseract.image_to_string(Image.open(img_file), lang='eng', config='--psm 6')
            result += f"\n\n--- {img_file} ---\n{text}"
            print(f"Extracted from {img_file}")
        except Exception as e:
            print(f"OCR failed for {img_file}: {e}")
    with open("ocr_output.txt", "w", encoding="utf-8") as f:
        f.write(result)
    print("OCR results saved to ocr_output.txt")

# Delete temporary page images
def cleanup_temp_images():
    for f in glob("handwriting_page*.png"):
        try:
            os.remove(f)
        except:
            pass

if __name__ == "__main__":
    print("=== Enhanced Handwriting Generator with Realistic Effects ===")

# Input text
    raw_text, source_type = get_text()
    if not raw_text.strip():
        print("No text found. Exiting.")
        exit()

# Optional translation
    if source_type != "image":
        if input("\nTranslate to Kannada? (Y/N): ").strip().lower() == 'y':
            print("Translating...")
            text = translate_english_to_kannada(raw_text)
            with open("translated_kannada.txt", "w", encoding="utf-8") as f:
                f.write(text)
            print("Translation saved to translated_kannada.txt")
        else:
```

```
            text = raw_text
    else:
        text = raw_text

# Font and color
    font_path = get_font_choice()
    ink_color = get_rgb_input()
    font = ImageFont.truetype(font_path, FONT_SIZE)
    usable_width = PAGE_WIDTH - MARGIN_LEFT - MARGIN_RIGHT

# Effects
    get_effect_toggles()

# Wrap text
    lines = book_style_wrap(text, font, usable_width)
    print(f"Text wrapped into {len(lines)} lines")
    print("\nGenerating pages...")

# Page generation
    page_num = 1
    for i in range(0, len(lines), LINES_PER_PAGE):
        page_lines = lines[i:i + LINES_PER_PAGE]
        generate_page(page_lines, page_num, font_path, ink_color)
        page_num += 1

# Output
    if source_type == "image":
        extract_text_from_images()
    elif source_type == "pdf":
        if input("\nCombine to PDF? (Y/N): ").strip().lower() == 'y':
            combine_images_to_pdf()
    else:
        combine_images_to_pdf()

# Cleanup
    cleanup_temp_images()
    print("\nProcess complete.")

```