# ATS Format Checker - Code Explanation

## 1. Importing Required Libraries

This section imports necessary libraries for file handling, GUI interaction, text extraction, hashing, logging, and JSON storage.

import magic  
import tkinter as tk  
from tkinter import filedialog, messagebox  
import fitz # PyMuPDF  
from pathlib import Path  
import docx # python-docx for Word documents  
import os  
import hashlib  
from datetime import datetime  
import json  
import logging

## 2. ATSFormatChecker Class

The core class that performs ATS compliance checks for resumes.

class ATSFormatChecker:  
 def \_\_init\_\_(self):  
 self.format\_scores = {  
 'application/pdf': 100,  
 'application/vnd.openxmlformats-officedocument.wordprocessingml.document': 100,  
 'application/msword': 80,  
 'text/plain': 70,  
 'application/rtf': 60,  
 }  
 self.root = tk.Tk()  
 self.root.withdraw()  
 self.setup\_logging()  
 self.config = self.load\_config()  
 self.history\_file = 'file\_history.json'  
 self.load\_history()

## 3. Logging Configuration

Configures logging to track processing steps and errors.

def setup\_logging(self):  
 logging.basicConfig(  
 filename='ats\_checker.log',  
 level=logging.INFO,  
 format='%(asctime)s - %(levelname)s - %(message)s'  
 )

## 4. Loading Configuration

Loads or sets default values for file size, word count, and formatting rules.

def load\_config(self):  
 default\_config = {  
 'min\_file\_size\_kb': 50,  
 'max\_file\_size\_mb': 10,  
 'min\_word\_count': 200,  
 'max\_word\_count': 5000,  
 'required\_sections': ['experience', 'education', 'skills'],  
 'forbidden\_characters': ['□', '■', '�', '°'],  
 'max\_image\_percentage': 30  
 }  
 if os.path.exists('config.json'):  
 with open('config.json', 'r') as f:  
 return {\*\*default\_config, \*\*json.load(f)}  
 return default\_config

## 5. Loading File History

Loads previously analyzed files to prevent redundant processing.

def load\_history(self):  
 if os.path.exists(self.history\_file):  
 with open(self.history\_file, 'r') as f:  
 self.history = json.load(f)  
 else:  
 self.history = {}

## 6. Saving Processing Results

Saves the ATS compliance results for each file based on its hash.

def save\_to\_history(self, file\_path, result):  
 file\_hash = self.get\_file\_hash(file\_path)  
 self.history[file\_hash] = {  
 'filename': os.path.basename(file\_path),  
 'last\_checked': datetime.now().isoformat(),  
 'result': result  
 }

## 7. Generating File Hash

Creates a unique MD5 hash to identify files.

def get\_file\_hash(self, file\_path):  
 hasher = hashlib.md5()  
 with open(file\_path, 'rb') as f:  
 buf = f.read(65536)  
 while len(buf) > 0:  
 hasher.update(buf)  
 buf = f.read(65536)  
 return hasher.hexdigest()

## 8. File Size Check

Ensures the document falls within acceptable file size limits.

def check\_file\_size(self, file\_path):  
 file\_size = os.path.getsize(file\_path)  
 min\_size = self.config['min\_file\_size\_kb'] \* 1024  
 max\_size = self.config['max\_file\_size\_mb'] \* 1024 \* 1024  
 if file\_size < min\_size:  
 return False, f"File too small (minimum {self.config['min\_file\_size\_kb']}KB)"  
 if file\_size > max\_size:  
 return False, f"File too large (maximum {self.config['max\_file\_size\_mb']}MB)"  
 return True, "File size acceptable

## 9. Text Extraction & Analysis

Extracts text from DOCX and PDF files for analysis.

def extract\_text\_from\_docx(self, file\_path):  
 doc = docx.Document(file\_path)  
 return '\n'.join([paragraph.text for paragraph in doc.paragraphs])

## 10. Checking Word Count & Required Sections

Checks if the document has enough words and includes mandatory sections like 'experience' and 'education'.

def count\_words(self, text):  
 return len(text.split())  
  
def check\_word\_count(self, text):  
 word\_count = self.count\_words(text)  
 if word\_count < self.config['min\_word\_count']:  
 return False, f"Too few words (minimum {self.config['min\_word\_count']})"  
 if word\_count > self.config['max\_word\_count']:  
 return False, f"Too many words (maximum {self.config['max\_word\_count']})"  
 return True, f"Word count acceptable ({word\_count} words)

## 11. Checking Forbidden Characters

Detects unwanted characters that might interfere with ATS parsing.

def check\_forbidden\_characters(self, text):  
 found\_chars = [char for char in self.config['forbidden\_characters'] if char in text]  
 return not found\_chars, found\_chars

## 12. Calculating ATS Score

Computes a final ATS score based on file format, word count, and formatting issues.

def calculate\_format\_score(self, file\_path):  
 result = {'score': 0, 'messages': [], 'file\_type': None, 'word\_count': 0, 'sections\_missing': [], 'formatting\_issues': [], 'recommendations': []}  
 file\_type = self.get\_file\_type(file\_path)  
 if file\_type not in self.format\_scores:  
 result['messages'].append(f"Unsupported file format: {file\_type}")  
 return result  
 return result

## 13. File Selection & Checking

Opens a file selection dialog and processes the chosen file.

def check\_file(self):  
 file\_path = filedialog.askopenfilename(title="Select Resume File")  
 if not file\_path:  
 return {'score': 0, 'messages': ["No file selected."], 'file\_type': None}  
 return self.calculate\_format\_score(file\_path)

## 14. Generating a Report

Creates a detailed report of the analysis results, including recommendations.

def generate\_report(self, result):  
 report = ["=== ATS Format Check Report ===", f"Score: {result['score']}/100", f"File Type: {result['file\_type']}", "\nMessages:"]  
 report.extend([f"- {msg}" for msg in result['messages']])  
 return '\n'.join(report)

## 15. Running the Checker

Executes the program, allowing users to select a file and view ATS compliance results.

def main():  
 checker = ATSFormatChecker()  
 result = checker.check\_file()  
 report = checker.generate\_report(result)  
 print(report)  
 messagebox.showinfo("ATS Check Results", f"Score: {result['score']}/100\nSee console for full report.")  
  
  
  
  
  
Features   
  
This code covers a comprehensive **ATS (Applicant Tracking System) Resume Format Checker**, analyzing various aspects of a resume's formatting, content, and compliance. Here’s a breakdown of all the features:

**1. File Handling & Format Recognition**

* Uses the magic library to detect the **file type** (PDF, DOCX, DOC, TXT, RTF).
* Supports and assigns **scores** for different file formats:
  + **PDF (100)**
  + **DOCX (100)**
  + **DOC (80)**
  + **TXT (70)**
  + **RTF (60)**
* Ensures the **file exists** before processing.
* Implements **file selection UI** via tkinter.filedialog.

**2. File Size Validation**

* Checks if file size is within acceptable limits:
  + **Minimum size**: 50 KB
  + **Maximum size**: 10 MB
* Provides **error messages** for files that are too small or too large.

**3. Word Count Analysis**

* Extracts text and **counts words** in the document.
* Ensures the word count is between **200 to 5000 words**.
* Provides feedback if the resume is too short or too long.

**4. Section Presence Validation**

* Ensures critical sections exist in the resume:
  + **Experience**
  + **Education**
  + **Skills**
* Identifies missing sections and provides recommendations.

**5. Forbidden Character Detection**

* Scans for special **non-ATS-friendly characters**:
  + □, ■, �, °
* Reports formatting issues if found.

**6. ATS Compliance Check for PDFs**

* Extracts text from PDFs using PyMuPDF (fitz).
* Checks if the PDF contains **selectable text**.
* Calculates **image percentage** in the PDF.
  + Flags PDFs with **>30% images** as non-ATS-friendly.

**7. Resume Formatting Analysis (ResumeFormatChecker)**

* **Font consistency check**: Identifies **inconsistent fonts**.
* **Most used font detection**: Reports the primary font in use.
* **Bullet point count**: Flags resumes with **>15 bullet points**.
* **Header analysis**: Ensures at least **3 section headers** are present.

**8. ATS Compatibility Scoring**

* Assigns **a base score** based on file format.
* Deducts points for:
  + **Word count issues** (-30)
  + **Missing sections** (-20 per section)
  + **Formatting issues** (-10 per special character)
  + **Font inconsistencies** (-10)
  + **Excessive bullet points** (-10)
  + **Few section headers** (-15)
* Ensures the final score is **capped at 0** if deductions exceed the base score.

**9. Resume Processing History**

* **Generates a file hash** using hashlib to track resumes.
* Saves **file history** (file\_history.json) to prevent duplicate evaluations.
* Stores:
  + **Filename**
  + **Last checked timestamp**
  + **Analysis results**

**10. Logging & Configuration Management**

* Logs errors and processing steps using the logging module.
* Loads and updates a **config file (config.json)** for:
  + Min/max **file size**
  + Min/max **word count**
  + Required **sections**
  + Forbidden **characters**
  + Max **image percentage** in PDFs

**11. Report Generation**

* Generates a **detailed ATS compliance report**, including:
  + **Date & time**
  + **Final ATS score**
  + **File type**
  + **Analysis messages**
  + **Recommendations**
* Prints the report to the console.
* Displays a **summary in a message box** (tkinter.messagebox).

Your **ATS System** consists of multiple modules that collectively offer a **comprehensive resume analysis tool**. Below is a **precise feature list** that your entire codebase covers:

**📌 1. File Handling & Format Validation**

✅ **Supports multiple resume file types:** PDF, DOCX, DOC, TXT, RTF  
✅ **Detects file format** using magic library  
✅ **Checks file size constraints** (min/max size validation)  
✅ **Extracts text** from DOCX (python-docx) and PDF (PyMuPDF)  
✅ **Verifies ATS compliance of PDFs** (ensuring selectable text & analyzing structure)

**📌 2. ATS Resume Analysis**

✅ **Checks resume format for ATS compliance**  
✅ **Analyzes word count** (ensuring it falls within the optimal range)  
✅ **Detects required resume sections** (Experience, Education, Skills, Summary, Certifications, etc.)  
✅ **Identifies missing sections** and suggests improvements  
✅ **Detects forbidden/unreadable characters** (□, ■, �, °) that ATS may not parse properly  
✅ **Calculates ATS compliance score** based on multiple factors  
✅ **Saves previous resume checks in a JSON-based history log**

**📌 3. Resume Format & Readability Analysis**

✅ **Analyzes font consistency** across the document  
✅ **Detects bullet points & excessive usage**  
✅ **Identifies missing or inconsistent section headers**  
✅ **Checks for passive voice usage** and recommends improvements  
✅ **Evaluates readability using multiple metrics:**

* **Flesch Reading Ease**
* **Gunning Fog Index**
* **SMOG Index**
* **Automated Readability Index**
* **Coleman Liau Index**
* **Dale Chall Readability**  
  ✅ **Provides recommendations for improved readability & structure**

**📌 4. Grammar & Style Checking**

✅ **Grammar checking using language\_tool\_python**  
✅ **Provides spelling & grammatical corrections**  
✅ **Detects passive voice usage and suggests corrections**  
✅ **Analyzes sentence structure and suggests readability improvements**  
✅ **Identifies overuse of complex words and recommends simpler alternatives**

**📌 5. Keyword & Industry-Specific Analysis**

✅ **Extracts industry-specific keywords** from resume  
✅ **Matches resume content with relevant job description keywords**  
✅ **Analyzes keyword density to ensure sufficient use of relevant terms**  
✅ **Provides insights on missing key industry skills & terminology**

**📌 6. LinkedIn Profile & Job Matching Analysis**

✅ **Extracts structured sections from LinkedIn profiles**  
✅ **Checks LinkedIn best-practice sections (Summary, Experience, Skills, etc.)**  
✅ **Identifies missing sections that could improve LinkedIn visibility**  
✅ **Performs job match scoring** using **TF-IDF & Cosine Similarity**  
✅ **Provides recommendations to align resume with job description**

**📌 7. Duplicate Content & Plagiarism Detection**

✅ **Identifies duplicated content in resumes**  
✅ **Flags overused generic resume templates**  
✅ **Suggests rewording for unique and impactful resume content**

**📌 8. User Interface & Logging**

✅ **Tkinter GUI file selection for user convenience**  
✅ **Displays results using pop-up message boxes**  
✅ **Logs all errors & analysis results in dedicated log files**  
✅ **Stores configuration settings in JSON for easy customization**