Code Analysis Report

Analyzed Code:

```
import pyttsx3
import pywin32_system32
import datetime
import speech_recognition as sr
import wikipedia
import webbrowser as wb
import os
import random
import pyautogui

engine = pyttsx3.init()

def speak(audio) -> None:
engine.say(audio)
engine.runAndWait()

def time() -> None:
Time = datetime.datetime.now().strftime("%I:%M:%S")
speak("the current time is")
speak(Time)
print("The current time is ", Time)

def date() -> None:
day: int = datetime.datetime.now().day
month: int = datetime.datetime.now().month
year: int = datetime.datetime.now().year
speak("the current date is")
speak(day)
speak(month)
speak(year)
print(f"The current date is {day}/{month}/{year}")

def wishme() -> None:
print("Welcome back sir!!")
speak("Welcome back sir!!")
```

```
hour: int = datetime.datetime.now().hour
if 4 <= hour < 12:
speak("Good Morning Sir!!")
print("Good Morning Sir!!")
elif 12 <= hour < 16:
speak("Good Afternoon Sir!!")
print("Good Afternoon Sir!!")
elif 16 <= hour < 24:
speak("Good Evening Sir!!")
print("Good Evening Sir!!")
else:
speak("Good Night Sir, See You Tommorrow")

speak("Jarvis at your service sir, please tell me how may I help you.")
print("Jarvis at your service sir, please tell me how may I help you.")

def screenshot() -> None:
img = pyautogui.screenshot()
img_path = os.path.expanduser("~\\Pictures\\ss.png")
img.save(img_path)

def takecommand():
r = sr.Recognizer()
with sr.Microphone() as source:
print("Listening...")
r.pause_threshold = 1
audio = r.listen(source)

try:
print("Recognizing...")
query = r.recognize_google(audio, language="en-in")
print(query)

except sr.UnknownValueError:
speak("Sorry, I did not understand that.")
return "Try Again"

except sr.RequestError:
speak("Sorry, my speech service is down.")
return "Try Again"
```

```
except Exception as e:
print(e)
speak("Please say that again")
return "Try Again"

return query

if __name__ == "__main__":
wishme()
while True:
query = takecommand().lower()
if "time" in query:
time()

elif "date" in query:
date()

elif "who are you" in query:
speak("I'm JARVIS created by Mr. Kishan and I'm a desktop voice assistant.")
print("I'm JARVIS created by Mr. Kishan and I'm a desktop voice assistant.")

elif "how are you" in query:
speak("I'm fine sir, What about you?")
print("I'm fine sir, What about you?")

elif "fine" in query:
speak("Glad to hear that sir!!")
print("Glad to hear that sir!!")

elif "good" in query:
speak("Glad to hear that sir!!")
print("Glad to hear that sir!!")

elif "wikipedia" in query:
try:
speak("Ok wait sir, I'm searching...")
query = query.replace("wikipedia", "")
result = wikipedia.summary(query, sentences=2)
print(result)
speak(result)
except:
speak("Can't find this page sir, please ask something else")
```

```
elif "open youtube" in query:
wb.open("youtube.com")

elif "open google" in query:
wb.open("google.com")

elif "open stack overflow" in query:
wb.open("stackoverflow.com")

elif "play music" in query:
song_dir = os.path.expanduser("~\\Music")
songs = os.listdir(song_dir)
print(songs)
x = len(songs)
y = random.randint(0, x)
os.startfile(os.path.join(song_dir, songs[y]))

elif "open chrome" in query:
chromePath = "C:\\Program Files
(x86)\\Google\\Chrome\\Application\\chrome.exe"
os.startfile(chromePath)

elif "search on chrome" in query:
try:
speak("What should I search?")
print("What should I search?")
chromePath = "C:\\Program Files
(x86)\\Google\\Chrome\\Application\\chrome.exe"
search = takecommand()
wb.get(chromePath).open_new_tab(search)
print(search)

except Exception as e:
speak("Can't open now, please try again later.")
print("Can't open now, please try again later.")

elif "remember that" in query:
speak("What should I remember")
data = takecommand()
speak("You said me to remember that" + data)
print("You said me to remember that " + str(data))
remember = open("data.txt", "w")
```

```
remember.write(data)
remember.close()

elif "do you remember anything" in query:
remember = open("data.txt", "r")
speak("You told me to remember that" + remember.read())
print("You told me to remember that " + str(remember))

elif "screenshot" in query:
screenshot()
speak("I've taken screenshot, please check it")

elif "offline" in query:
quit()
```

Static Code Metrics:

Metric	Value
Lines of Code (LOC)	142
Comment Density	0.0
Cyclomatic Complexity	12
Maintainability Index	30.61620133307405
Halstead Metrics	
Unique Operators	5
Unique Operands	33
Total Operators	28
Total Operands	53
Volume	425.0821285889304
Effort	1706.7691526676754
Depth of Inheritance Tree (DIT)	0
Coupling Between Object classes (CBO)	0
Lack of Cohesion of Methods (LCOM)	0
Fan-in	26
Fan-out	26
Number of Methods (NOM)	0

McCall's Intermediate Quality Metrics:

Metric	Value
Modifiability	0.4
Testability	0.53
Reliability	0.6
Understandability	0.37
Self-Descriptiveness	0.07
Reusability	1.0
Portability	0.83
Efficiency	0.19

GPT Analysis Report:

Code Quality Analysis Report

Observations:

1. **Comment Density**: The absence of comments or docstrings significantly reduces understandability for other developers or even for future reference. Comments should describe the purpose of functions and complex code blocks.

2. **Maintainability Index**: The maintainability index is relatively low (30.62), indicating that the code could be challenging to maintain and modify.

3. **Cyclomatic Complexity**: With a complexity score of 12, this indicates that the code contains multiple paths and nests, making it harder to understand and test.

4. **Lack of Cohesion**: The main functions are somewhat disparate in their functionality, indicating a lack of separation of concerns, which may hinder code reusability and maintenance.

5. **Redundant Code**: There are repeated patterns, e.g., the way responses are handled for various queries. This presents an opportunity for refactoring.

Specific Suggestions for Improvement

1. **Add Comments and Docstrings**:

- Include comments and docstrings for functions to explain their purpose, input parameters, and return values. For example, for the `speak` function:

```
python
def speak(audio: str) -> None:
"""
Converts the given text to speech.
:param audio: Text to be spoken.
```

```
:return: None
"""
engine.say(audio)
engine.runAndWait()
```

2. **Refactor Redundant Code**:

- Many recognition responses involve similar `speak` and `print` calls. Creating a helper function to handle these responses can reduce redundancy:

```
python
def respond(message: str):
speak(message)
print(message)
```

3. **Use of Constants**:

- Store frequently used strings and paths in constants to avoid magic strings throughout the code. This enhances maintainability:

```
python
YOUTUBE_URL = "youtube.com"
GOOGLE_URL = "google.com"
STACKOVERFLOW_URL = "stackoverflow.com"
```

4. **Improve Command Handling**:

- Use a dictionary to map queries to function calls or responses, simplifying the command processing loop. This aids in scalability and readability:

```
python
command_map = {
"time": time,
"date": date,
"who are you": lambda: respond("I'm JARVIS created by Mr. Kishan and I'm a desktop voice assistant."),
# Additional mappings here...
}
query = takecommand().lower()
if query in command_map:
command_map[query]() # Call the associated function
```

5. **Handle Error Cases More Robustly**:

- In the `takecommand` function, instead of using a broad `except Exception as e`, specify what kind of exceptions are expected. This avoids masking potential bugs in the code.

- Instead, you might want to explicitly handle known issues and log unexpected errors.

6. **Encapsulate Speech Recognition Logic**:

- Create a dedicated class for speech recognition operations. This separates concerns and makes the code more modular:

```
python
class SpeechAssistant:
def __init__(self):
self.recognizer = sr.Recognizer()
def listen(self):
# Implement listening logic
```

7. **Optimize Imports**:

- Remove any unnecessary imports. The `import pywin32_system32` line is suspicious as it is not used in the provided code. Unused libraries can lead to confusion and unnecessarily increase dependencies.

8. **Data Handling Improvements**:

- When writing and reading from files (like remembering notes), consider using context managers (`with open(...) as f:`) for better handling and to avoid file leaks.

9. **Enhance Testability**:

- Design the functions to be more test-friendly. You may want to avoid direct interactions (like speaking and screen management) within the methods to increase unit test coverage.

10. **Use Type Annotations**:

- The code lacks type annotations in several places. Adding these will increase clarity for users and allow tools like mypy to check type correctness at compile time.

Conclusion

Implementing these recommendations will lead to a cleaner, more maintainable, and more structured codebase. This not only benefits current development but also makes future enhancements or collaborations with others easier and more efficient.