Static Analyzer Features:

Since we are going to test on executables, we'd want some signature features that is 1) easy to extract and 2) Informative and differentiating between benign and ransomware executables.

- ### Imports:
  - Certain DLLs:

Crypto DLLs	Rundll32.exe, advapi32.dll, bcrypt.dll, ncrypt.dll, crypt32.dll, wincrypt.h, etc.
File system enumeration	kernel32.dll, shell32.dll, shlwapi.dll, ntdll.dll, ole32.dll, etc.
Internet requests	Wininet32.dll, winhttp.dll, ws2_32.dll, urlmon.dll, httpapi.dll, dnsapi.dll, etc

- ### Famous suspicious Functions:
  - LoadLibrary / GetProcAddress (to dynamically load DLL files)
  - FindFirstFile / FindNextFile / FindClos (Enumerating files/folders for encryption)
  - CreateFile (Opening target files for reading or writing)
  - ReadFile / WriteFile (Reading original file content, writing encrypted content)
  - DeleteFile / RemoveDirectory (Removing original files (some ransomware deletes the plaintext after encryption))
  - SetFileAttributes (Hiding files or removing protections)
  - GetLogicalDrives / GetDriveType (Discovering available drives (including network or removable))
  - CryptAcquireContext (crypto function in advapi32)
  - CryptGenKey
  - CryptEncrypt / CryptDecrypt (Classic CryptoAPI for encryption)
  - BCryptEncrypt / BCryptGenRandom (in bcrypt.dll, CNG-based modern crypto)
  - Custom AES, RSA, or ECC routines (Often identified by S-boxes, key schedules, or known constants)
  - InternetOpen / InternetConnect / HttpOpenRequest / HttpSendRequest (wininet.dll)
  - WinHttpOpen / WinHttpConnect / WinHttpSendRequest (winhttp.dll)
  - socket / connect / send / recv (ws2_32.dll)
  - URLDownloadToFile (in urlmon.dll, Communicating with command-and-control servers to fetch keys, send victim info)
  - OpenProcess
  - VirtualAlloc / VirtualAllocEx
  - WriteProcessMemory / ReadProcessMemory
  - CreateRemoteThread (For process injection or evasion)
  - AdjustTokenPrivileges / OpenProcessToken (Escalating privileges)
  - IsDebuggerPresent / CheckRemoteDebuggerPresent (Anti-debugging)

- NtQueryInformationProcess (in ntdll.dll for Detecting debuggers or sandbox environments)
- RegOpenKey / RegSetValue / RegCreateKey (for Modifying registry run keys to achieve persistence)
- CreateService / OpenService / StartService (Installing malicious services for autorun or scheduling)
- CopyFile to %AppData% / %Startup% (Achieving autorunning)
- system / ShellExecute / CreateProcess (Executing commands like vssadmin to delete shadow copies)
- DeleteVolumeMountPoint (for hard disk ransomwares)
- DeviceIoControl (Direct disk wiping or MBR manipulation ,rare, but dangerous)
- Sleep (Delays to evade analysis and sandboxes)
- GetTickCount / QueryPerformanceCounter / RDTSC (Timing checks for debuggers or emulators)
- LoadLibrary / GetProcAddress (Dynamically loading APIs or DLLs to hide imports)
- NtDelayExecution (More precise anti-debugging sleep)

- **Byte entropy for the entire file (to detect usage of packers)**
- **Weird section names, anything other than the known section names which are:**

.text	.rdata	**.**edata	.tls
.data	.rodata	.rsrc	.pdata
.bss	.idata	.reloc	.CRT

- **Static strings:**
  - Hardcoded IP addresses / URLs
  - Ransom Note words (“encrypted”, “pay”, “disk”, “if”, “before” , “delete”, etc.)
  - Base64-encoded or XOR’d strings (especially filenames, extensions, C2 addresses)
  - A lot of File extensions, more than 5 (.txt, .jpg, .pdf, .png, .html, etc.)
  - RSA public key blobs or AES S-box constants
  - Weird File extension patterns like (.encrypted, .locked, .payforunlock, .fun, etc).
  - Hardcoded commands like:
    - vssadmin delete shadows
    - bcdedit /set {default} recoveryenabled no
    - cipher /w
  - Hardcoded absolute paths or relative paths
- **Yara Rules for ransomware executables** (Just search for them, duh!)

Python Libraries to achieve this abomination:

1- pefile, lief, capstone (to extract imported DLLs and used imported functions, also to extract section names)

2- using subprocess, invoking “strings” shell command (to extract static strings)

```
import subprocess

output = subprocess.check_output(["strings", "sample.exe"])
print(output.decode())
```

3- Entropy calculation (google it, afaik there are no libraries for this, you probably have to implement this yourself). There are open source ranges to check for normal entropy and suspicious entropy, google it.

You can install “pestudio”, it extracts a lot of info, but this is a GUI program, not a library.