Implementation of Resume Screening with Python

```
import requests
```

```
import requests

# Raw GitHub URL of your notebook
url =
"https://raw.githubusercontent.com/samvincy7/pyin/main/AI_and_Automati
on_Lab.ipynb"

# Download and save the notebook file
response = requests.get(url)
with open("pyin_downloaded.ipynb", "wb") as f:
    f.write(response.content)

print("Notebook downloaded as pyin_downloaded.ipynb")
```

```
Notebook downloaded as pyin_downloaded.ipynb
```

```
import PyPDF2
import spacy
import pandas as pd
nlp = spacy.load("en_core_web_sm")
def extract_text(pdf_path):
    text=' '
    with open(pdf_path,'rb')as f:
        reader =  PyPDF2.PdfReader(f)
        for page in reader.pages:
            text += page.extract_text()
            return text.lower()
def screen_resume(text,keywords):
    doc = nlp(text)
    return{kw:text.count(kw)for kw in keywords}
def accept_resume(matches,threshold=1):
    return all(count>=threshold for count in matches.values())
keywords=["amazon","cluny","networking"]
pdf_path =r"C:\Users\HP\Downloads\Stockholm-Resume-Template-
Simple.pdf"
text = extract_text(pdf_path)
matches = screen_resume(text,keywords)
accepted = accept_resume(matches)
df = pd.DataFrame(matches.items(),columns=["keywords","scores"])
print(df)
print("resume accepted" if accepted else"resume rejected")
```

```
     keywords  scores
0      amazon       4
1       cluny       0
2  networking       0
resume rejected
```

```
import PyPDF2
import spacy
import pandas as pd

nlp = spacy.load("en_core_web_sm")

def extract_text(pdf_path):
    text = ''
    with open(pdf_path, 'rb') as f:
        reader = PyPDF2.PdfReader(f)
        for page in reader.pages:
            text += page.extract_text()
    return text.lower()  # <- moved outside loop

def screen_resume(text, keywords):
    return {kw: text.count(kw) for kw in keywords}

def accept_resume(matches, threshold=1):
    return all(count >= threshold for count in matches.values())

# Define keywords and path
keywords = ["amazon", "cluny", "networking"]
pdf_path = r"C:\Users\HP\Downloads\Stockholm-Resume-Template-
Simple.pdf"

# Process resume
text = extract_text(pdf_path)
matches = screen_resume(text, keywords)
accepted = accept_resume(matches)

# Display results
df = pd.DataFrame(matches.items(), columns=["Keyword", "Score"])
print(df)
print("Resume Accepted" if accepted else "Resume Rejected")
```

```
      Keyword  Score
0      amazon      4
1       cluny      0
2  networking      0
Resume Rejected
```

NER

```
import spacy
def perform_ner(text):
    nlp = spacy.load("en_core_web_sm")
    doc = nlp(text)
    return[(ent.text,ent.label_) for ent in doc.ents]
text ="Elon Musk is the CEO of Tesla."
te="Sam is the president of India,TN"
for ent,label in perform_ner(te):
    print(f"{ent}-->{label}")
```

```
Sam-->PERSON
India-->GPE
TN-->ORG
```

```
import spacy

nlp = spacy.load("en_core_web_sm")
text = "Steve Jobs founded Apple in California in 1976 with $1,300."

doc = nlp(text)
for ent in doc.ents:
    print(f"{ent.text} -> {ent.label_}")
```

```
Steve Jobs -> PERSON
Apple -> ORG
California -> GPE
1976 -> DATE
1,300 -> MONEY
```

Sentiment Analysis

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
def sentiment_analyzer(text):
    sia = SentimentIntensityAnalyzer()
    score = sia.polarity_scores(text)['compound']
    return "positive" if score >=0.5 else"negative"if score<=-0.5
else"neutral"
text ="i loved the movie."
print("sentiment:",sentiment_analyzer(text))
```

```
sentiment: positive
```

Keyword Extraction

```
import nltk
import yake
text ="Python is a versatile language used in datascience and artifial
intellingence"
kw_extractor = yake.KeywordExtractor(lan="en",n=1,top=5)
keywords = kw_extractor.extract_keywords(text)
for kw,score in keywords:
    print(kw)
```

```
Python
intellingence
versatile
language
datascience
```

Spelling Correction Model

```
from textblob import TextBlob
def correct_spelling(text):
    return TextBlob(text).correct()
text ="i lke to ployy in parrk"
print("correct text:",correct_spelling(text))
```

```
correct text: i like to play in park
```

```
from autocorrect import Speller
def correct_spelling(text):
        spell = Speller(lang="en")
        return spell(text)
text = "i lke to ployy in parrk"
print("original text:",text)
print("corrected text:",correct_spelling(text))
```

```
original text: i lke to ployy in parrk
corrected text: i like to ploy in park
```

Keyboard Autocorrection Model

```
from difflib import get_close_matches
def get_matches(word,word_list):
    matches = get_close_matches(word,word_list,n=1,cutoff=0.8)
    return matches[0]if matches else word
word_list=["apple","banana","elephant"]
input_text="she likes appless and bananana."
corrected="  ".join([autocorrect(word,word_list)for word in
```

```
input_text.split()])
print("correct text:",corrected)
```

```
correct text: she  likes  apple  and  banana
```

Implementation of Election Results prediction by analysing Tweets

```
from textblob import TextBlob

tweets_candidate1 = [
    "I really like Candidate A, they are honest and capable!",
    "Candidate A is doing great work.",
    "I'm voting for Candidate A!"
]

tweets_candidate2 = [
    "Candidate B is not trustworthy.",
    "I don't like Candidate B's ideas.",
    "Candidate B is a bad choice."
]
def analyse_sentiment(tweets):
    score=0
    for tweet in tweets:
        analysis = TextBlob(tweet)
        score += analysis.sentiment.polarity
        return score
score_a=analyse_sentiment(tweets_candidate1)
score_b=analyse_sentiment(tweets_candidate2)
print(score_a)
print(score_b)
if score_a>score_b:
    print("candidate A win")
elif score_a<score_b:
     print("candidate A win")
else:
    print("it's a tie")


0.35000000000000003
0.0
candidate A win
```

Development of NLP for other Language

```
import spacy
from textblob import TextBlob

nlp = spacy.load("es_core_news_sm")
text="hola como estas"
doc = nlp(text)

print("token and pos")
for token in doc:
    print(token.text,token.pos_)
print("Named entity")
for ent in doc.ents:
    print(ent.text,ent.label_)
sentiment = sum(TextBlob(sent.text).sentiment.polarity for sent in
doc.sents)
print("sentiment :",sentiment)
```

```
token and pos
hola VERB
como SCONJ
estas DET
Named entity
hola ORG
sentiment : 0.0
```

```
import spacy
from textblob import TextBlob
nlp = spacy.load("es_core_news_sm")
text = "Me encanta la comida mexicana y el tequila."
doc = nlp(text)
print("Tokens and POS:")
for token in doc:
    print(token.text, token.pos_)
print("\nNamed Entities:")
for ent in doc.ents:
    print(ent.text, ent.label_)
sentiment = sum(TextBlob(sent.text).sentiment.polarity for sent in
doc.sents)
print("\nSentiment Score:", sentiment)
```

```
Tokens and POS:
Me PRON
encanta VERB
la DET
comida NOUN
mexicana ADJ
y CCONJ
el DET
```

```
tequila PROPN
. PUNCT

Named Entities:

Sentiment Score: 0.0
```

Creation of Text Classification using Deep Learning

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
from sklearn.model_selection import train_test_split

# Data
texts = [
    'I love programming',
    'Python is great',
    'I hate bugs',
    'Coding is fun',
    'Debugging is hard',
    'I feel okay today'
]
labels = ['positive', 'positive', 'negative', 'positive', 'negative',
'neutral']

# Label encoding
label_map = {'positive': 0, 'negative': 1, 'neutral': 2}
reverse_label_map = {v: k for k, v in label_map.items()}
labels_encoded = [label_map[label] for label in labels]

# Tokenization and padding
tokenizer = Tokenizer(num_words=100)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
X = pad_sequences(sequences, padding='post')
y = np.array(labels_encoded)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=100, output_dim=16,
```

```
input_length=X.shape[1]),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train
model.fit(X_train, y_train, epochs=10, batch_size=2)

# Evaluate
loss, accuracy = model.evaluate(X_test, y_test)
print(f"\nTest loss: {loss}")
print(f"Test accuracy: {accuracy}")

# Prediction examples
test_texts = [
    "This movie was amazing!",
    "The food was terrible at this restaurant."
]

print()
for text in test_texts:
    seq = tokenizer.texts_to_sequences([text])
    pad = pad_sequences(seq, maxlen=X.shape[1], padding='post')
    pred = model.predict(pad)
    pred_label = reverse_label_map[np.argmax(pred)]
    print(f"Text: {text}")
    print(f"Predicted label: {pred_label.capitalize()}\n")
```

```
Epoch 1/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.5000 - loss:
1.0939
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3333 - loss:
1.0917.10
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6667 - loss: 1.0817
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6667 - loss:
1.0813
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8333 - loss: 1.0788

Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6667 - loss: 1.0689
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6667 - loss: 1.0714
```

```
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 1.0701
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 1.0677
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 1.0519
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 228ms/step - accuracy: 0.5000 - loss:
1.0885

Test loss: 1.0884790420532227
Test accuracy: 0.5

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
Text: This movie was amazing!
Predicted label: Positive

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
Text: The food was terrible at this restaurant.
Predicted label: Positive


def chatbot():
    print("Hi! I'm a simple chatbot. Ask me anything (type 'exit' to
quit).")

    while True:
        user_input = input("You: ").lower()

        if user_input == "exit":
            print("Chatbot: Goodbye!")
            break

        # Health domain
        elif "fever" in user_input or "sick" in user_input:
            print("Chatbot: I'm sorry to hear that. You should rest
and maybe see a doctor.")

        elif "headache" in user_input:
            print("Chatbot: Try drinking water or taking a short
break. If it continues, see a doctor.")

        # Education domain
        elif "study tips" in user_input:
            print("Chatbot: Study in short sessions, take breaks, and
test yourself often.")

        elif "math" in user_input:
            print("Chatbot: Practice regularly and try solving sample
problems.")

        # Travel ma
```

```
        elif "book ticket" in user_input or "flight" in user_input:
            print("Chatbot: You can book tickets online using travel
sites like MakeMyTrip or IRCTC.")

        elif "places to visit" in user_input:
            print("Chatbot: You should try visiting historical places,
beaches, or mountains depending on your interest!")

        # Tech support domain
        elif "internet not working" in user_input:
            print("Chatbot: Try restarting your router. If it still
doesn't work, call your internet provider.")

        elif "password reset" in user_input:
            print("Chatbot: You can reset your password using the
'Forgot Password' link on the login page.")

        # Fallback response
        else:
            print("Chatbot: Sorry, I don't understand that. Can you
rephrase or ask something else?")

# Run the chatbot
chatbot()
```

```
Hi! I'm a simple chatbot. Ask me anything (type 'exit' to quit).

You:  hi

Chatbot: Sorry, I don't understand that. Can you rephrase or ask
something else?

You:  fever

Chatbot: I'm sorry to hear that. You should rest and maybe see a
doctor.

You:  sick

Chatbot: I'm sorry to hear that. You should rest and maybe see a
doctor.

You:  hi

Chatbot: Sorry, I don't understand that. Can you rephrase or ask
something else?

You:  math

Chatbot: Practice regularly and try solving sample problems.

You:  q
```

```
Chatbot: Sorry, I don't understand that. Can you rephrase or ask
something else?

You:  exit

Chatbot: Goodbye!

pip install scikit-learn nltk

Requirement already satisfied: scikit-learn in c:\users\hp\anaconda3\
lib\site-packages (1.6.1)
Requirement already satisfied: nltk in c:\users\hp\anaconda3\lib\site-
packages (3.9.1)
Requirement already satisfied: numpy>=1.19.5 in c:\users\hp\anaconda3\
lib\site-packages (from scikit-learn) (2.2.5)
Requirement already satisfied: scipy>=1.6.0 in c:\users\hp\anaconda3\
lib\site-packages (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in c:\users\hp\anaconda3\
lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\hp\
anaconda3\lib\site-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: click in c:\users\hp\anaconda3\lib\
site-packages (from nltk) (8.1.7)
Requirement already satisfied: regex>=2021.8.3 in c:\users\hp\
anaconda3\lib\site-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in c:\users\hp\anaconda3\lib\site-
packages (from nltk) (4.66.5)
Requirement already satisfied: colorama in c:\users\hp\anaconda3\lib\
site-packages (from click->nltk) (0.4.6)
Note: you may need to restart the kernel to use updated packages.

from collections import Counter

def summarize_text(text, num_sentences=2):
    sentences = text.split('.')
    words = text.lower().split()
    word_freq = Counter(words)

    sentence_scores = []
    for sentence in sentences:
        score = sum(word_freq[word] for word in
sentence.lower().split())
        sentence_scores.append((sentence, score))

    return '. '.join([sentence for sentence, _ in
sorted(sentence_scores, key=lambda x: x[1], reverse=True)
[:num_sentences]])

# Input text
text = "Natural Language Processing (NLP) is a field of AI. It allows
machines to understand human language. NLP is used in chatbots, speech
```

```
recognition, and more."

# Summarize
summary = summarize_text(text, 2)
print("Summary:", summary)
```

```
Summary: Natural Language Processing (NLP) is a field of AI.  NLP is
used in chatbots, speech recognition, and more
```