Ans to question 1

Task1:

Q1_Task_1.py > ...

```python
import pandas as pd

# Define the file paths
csv_files = [
    r'CSV1.csv',
    r'CSV2.csv',
    r'CSV3.csv',
    r'CSV4.csv'
]

columns_to_check = ['SHORT-TEXT', 'TEXT']
output_txt_file = 'Q1_extracted_texts.txt'

def extract_and_save_text(df, column_name, output_file, mode='a'):
    """Extracts text from the specified column and saves it to a text file."""
    if column_name in df.columns:
        texts = df[column_name].dropna().tolist()  # Drop NaN values
        with open(output_file, mode, encoding='utf-8') as outfile:
            for text in texts:
                outfile.write(text + '\n')
        return True
    return False

# Process each CSV file and extract text from the appropriate columns
with open(output_txt_file, 'w', encoding='utf-8') as outfile:
    outfile.write("")  # Clear the file at the beginning

for i, csv_file in enumerate(csv_files):
    try:
        # Read the CSV file into a DataFrame
        df = pd.read_csv(csv_file)

        # Check for both possible column names and save text if the column exists
        saved = False
        for column in columns_to_check:
            if extract_and_save_text(df, column, output_txt_file, 'a'):
                print(f"Extracted text from '{column}' in {csv_file}.")
                saved = True
                break  # Stop checking other columns if one is found

        if not saved:
            print(f"None of the specified columns found in {csv_file}.")

    except FileNotFoundError:
        print(f"Error: The file '{csv_file}' does not exist.")
    except pd.errors.EmptyDataError:
        print(f"Error: The file '{csv_file}' is empty.")
    except Exception as e:
        print(f"An unexpected error occurred while processing '{csv_file}': {str(e)}")
```

Task2: Research

Q1_Task_2.py > ...

```python
#imports
import spacy
from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline

#Installation Command:
    ##Install the libraries(SpaCy - 'en_core_sci_sm').
        ###pip install spacy==2.3.5
        ###python -m spacy download en_core_sci_sm
    ##Transformer Install Command
        ###pip install transformers

#Sample Usage in my code

## spacy
def load_spacy_model(model_path):
    try:
        return spacy.load(model_path)
    except OSError as e:
        print(f"Error: Failed to load the SpaCy model '{model_path}'. {str(e)}")
        return None

    nlp_sci_sm = load_spacy_model("en_core_sci_sm")


## transformers
def load_biobert_pipeline(model_name):
    try:
        tokenizer = AutoTokenizer.from_pretrained(model_name)
        model = AutoModelForTokenClassification.from_pretrained(model_name)
        return pipeline("ner", model=model, tokenizer=tokenizer)
    except Exception as e:
        print(f"Error: Failed to load BioBERT model '{model_name}'. {str(e)}")
        return None

biobert_pipeline = load_biobert_pipeline("dmis-lab/biobert-base-cased-v1.1")
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased", use_fast=True)

```

Task3: Programming and Research

Q1_Task_3.py > ...

```python
import pandas as pd
from collections import Counter
from transformers import AutoTokenizer
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')

output_txt_file = 'Q1_extracted_texts.txt'
top_words_csv = 'Q1_top_30_words.csv'

# Task 3.1: Count the Top 30 Most Common Words and Store in a CSV File
def count_top_words(text_file, output_csv):
    try:
        with open(text_file, 'r', encoding='utf-8') as file:
            text = file.read()

        if not text.strip():  # Check if the file is empty
            print(f"Error: The file '{text_file}' is empty.")
            return

        # Split the text into words and count occurrences
        words = text.split()
        word_counts = Counter(words)

        # Get the 30 most common words
        top_30_words = word_counts.most_common(30)

        # Store the result in a CSV file
        df_top_words = pd.DataFrame(top_30_words, columns=['Word', 'Count'])
        df_top_words.to_csv(output_csv, index=False)

        print(f"\nTask 3.1: Top 30 words saved to {output_csv}\n")
    except FileNotFoundError:
        print(f"Error: The file '{text_file}' was not found.")
    except Exception as e:
        print(f"An error occurred while counting words: {str(e)}")

# Task 3.2: Count Unique Tokens Using Transformers AutoTokenizer (with Chunking and Truncation)
def count_unique_tokens(text_file, chunk_size=5000, max_length=500):
    try:
        tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased", use_fast=True)

        token_counts = Counter()

        with open(text_file, 'r', encoding='utf-8') as file:
            while True:
```

```
                # Read the file in chunks
                chunk = file.read(chunk_size)
                if not chunk:
                    break  # End of file

                # Tokenize the chunk with truncation to handle long sequences
                tokens = tokenizer.tokenize(chunk, truncation=True, max_length=max_length)

                # Count token occurrences
                token_counts.update(tokens)

        # Get the 30 most common tokens
        top_30_tokens = token_counts.most_common(30)

        print(f"\nTask 3.2: Top 30 tokens:\n {top_30_tokens}")
        return top_30_tokens

    except FileNotFoundError:
        print(f"Error: The file '{text_file}' was not found.")
    except Exception as e:
        print(f"An error occurred while counting tokens: {str(e)}")

# Run the function for Task 3.1
count_top_words(output_txt_file, top_words_csv)

# Run the function for Task 3.2
count_unique_tokens(output_txt_file)
```

Task 4: Named-Entity Recognition (NER)

Q1_Task_4.py > ...

```python
import pandas as pd
import spacy
from collections import Counter
from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline
import warnings

# Suppress specific warnings related to truncation
warnings.filterwarnings("ignore")

output_txt_file = 'Q1_extracted_texts.txt'

# Define maximum token length for BioBERT (500 tokens, plus 2 for [CLS] and [SEP])
MAX_TOKEN_LENGTH = 500

# Try to load the SpaCy models with error handling
def load_spacy_model(model_path):
    try:
        return spacy.load(model_path)
    except OSError as e:
        print(f"Error: Failed to load the SpaCy model '{model_path}'. {str(e)}")
        return None

# Try to load BioBERT model and tokenizer with error handling
def load_biobert_pipeline(model_name):···

# Function to extract entities using SpaCy models
def extract_entities_spacy(text_file, model, label_filter):
    try:
        with open(text_file, 'r', encoding='utf-8') as file:
            text = file.read()

        if model is None:
            print("Error: No model available for entity extraction.")
            return []

        doc = model(text)
        entities = [ent.text for ent in doc.ents if ent.label_ in label_filter]
        return entities
    except FileNotFoundError:
        print(f"Error: The file '{text_file}' was not found.")
        return []
    except Exception as e:
        print(f"An error occurred during entity extraction: {str(e)}")
        return []

```

```python
# Function to extract entities using BioBERT with correct truncation and chunking
def extract_entities_biobert(text_file, biobert_pipeline, max_length=MAX_TOKEN_LENGTH):
    try:
        with open(text_file, 'r', encoding='utf-8') as file:
            text = file.read()

        if biobert_pipeline is None:
            print("Error: No BioBERT pipeline available for entity extraction.")
            return [], []

        # Tokenize the text and apply truncation and padding
        tokenizer = biobert_pipeline.tokenizer
        tokens = tokenizer(text, truncation=True, padding='max_length', max_length=max_length, returr

        # Ensure the total number of tokens is within the 512 limit, including [CLS] and [SEP]
        input_texts = tokenizer.batch_decode(tokens['input_ids'], skip_special_tokens=True)

        diseases, drugs = [], []
        for chunk in input_texts:
            entities = biobert_pipeline(chunk)  # Pass text chunks to the BioBERT pipeline
            diseases += [entity['word'] for entity in entities if entity['entity'] == 'B-Disease']
            drugs += [entity['word'] for entity in entities if entity['entity'] == 'B-Drug']

        return diseases, drugs

    except FileNotFoundError:
        print(f"Error: The file '{text_file}' was not found.")
        return [], []
    except Exception as e:
        print(f"An error occurred during BioBERT entity extraction: {str(e)}")
        return [], []

# Load models with error handling
nlp_sci_sm = load_spacy_model("en_core_sci_sm")
biobert_pipeline = load_biobert_pipeline("dmis-lab/biobert-base-cased-v1.1")

# Extract diseases and drugs using SpaCy models with error handling
diseases_sci_sm = extract_entities_spacy(output_txt_file, nlp_sci_sm, ['DISEASE'])

# Extract diseases and drugs using BioBERT with correct chunking
diseases_biobert, drugs_biobert = extract_entities_biobert(output_txt_file, biobert_pipeline)

# Compare the results of SpaCy models and BioBERT
print(f"SpaCy (en_core_sci_sm) detected {len(diseases_sci_sm)} diseases")
print(f"BioBERT detected {len(diseases_biobert)} diseases and {len(drugs_biobert)} drugs")

# Output most common entities if found
if diseases_biobert:
    print("Most common diseases detected by BioBERT:", Counter(diseases_biobert).most_common(5))
else:
    print("No diseases detected by BioBERT.")

if drugs_biobert:
    print("Most common drugs detected by BioBERT:", Counter(drugs_biobert).most_common(5))
else:
    print("No drugs detected by BioBERT.")
```