Appendix

Data Source

- Medical Transcripts: https://mtsamples.com
- Kaggle Dataset: https://www.kaggle.com/datasets/tboyle10/medicaltranscriptions
- Emergency Department Dataset: https://mimic.mit.edu/docs/iv/modules/ed/

Clinical Note Extraction

Install necessary packages

```
!pip install scispacy
!pip install https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.5.1/en_core_sci_sm-0.5.1.tar.gz
!pip install https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.5.1/en_core_sci_md-0.5.1.tar.gz
!pip install https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.5.1/en_ner_bc5cdr_md-0.5.1.tar.gz
```

Loading packages

```
import pandas as pd
import spacy
import scispacy
import en_core_sci_sm
import en_core_sci_md
#NER specific models
import en_ner_bc5cdr_md
#Tools for extracting & displaying data
from spacy import displacy
```

Upload csv files

```
from google.colab import files
uploaded = files.upload()

df = pd.read_csv(r'mimic_clinical_note.csv', encoding='ISO-8859-1')
df

text = df.loc[0, 'transcription']
text
```

```
# Load specific model: en_core_sci_sm and pass text through
nlp_sm = en_core_sci_sm.load()
doc = nlp_sm(text)

# display results by entity extraction
displacy_image = displacy.render(doc, jupyter = True, style = 'ent')

# Load specific model: en_core_sci_md and pass text through
nlp_md = en_core_sci_md.load()
doc = nlp_md(text)
displacy_image = displacy.render(doc, jupyter = True, style = 'ent')

# Now Load specific model: import en_ner_bc5cdr_md and pass text through
nlp_bc = en_ner_bc5cdr_md.load()
doc = nlp_bc(text)
#Display resulting entity extraction
displacy_image = displacy.render(doc, jupyter=True,style='ent')

print("TEXT", "START", "END", "ENTITY TYPE")
for ent in doc.ents:
   print(ent.text, ent.start_char, ent.end_char, ent.label_)

df.dropna(subset=['transcription'], inplace=True)
df_subset = df.sample(n=2, replace=False, random_state=42)
df_subset.info()
df_subset.head()

from spacy.matcher import Matcher
pattern = [{'ENT_TYPE':'CHEMICAL'}, {'LIKE_NUM': True}, {'IS_ASCII': True}]
matcher = Matcher(nlp_bc.vocab)
matcher.add("DRUG_DOSE", [pattern])
for transcription in df_subset['transcription']:
   doc = nlp_bc(transcription)
   matches = matcher(doc)
   for match_id, start, end in matches:
      string_id = nlp_bc.vocab.strings[match_id]  # get string representation
      span = doc[start:end]  # the matched span adding drugs doses
      print(span.text, start, end, string_id,)
#Add disease and drugs
      for ent in doc.ents:
         print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

Emergency Department data pre-processing

```
import pandas as pd
from google.colab import files
uploaded = files.upload()
```

```
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
from sklearn.cluster import KMeans

# patient stay
edstay = pd.read_csv(r'edstays.csv', encoding = "ISO-8859-1")
edstay = edstay.dropna()
df_ed = edstays.drop(columns = ['intime', 'outtime', 'gender', 'race', 'arrival_transport',
'disposition'])
df_ed = df_ed.astype({'hadm_id': 'int'})

# triage
triage = pd.read_csv(r'triage.csv', encoding = "ISO-8859-1")
triage = triage.dropna()
df_tri = triage.drop(columns = [ 'temperature', 'heartrate', 'resprate', 'o2sat', 'sbp', 'dbp', 'pain'])
df_tri = df_tri.astype({'acuity':'int'})
df_tri['chiefcomplaint'] = df_tri['chiefcomplaint'].str.lower()
df_tri

# removing duplicates
df_tri.drop_duplicates(subset=['stay_id'])

combined = df_ed.merge(df_tri.drop_duplicates(subset=['stay_id']), how='left')
df = combined.dropna()
df = df.astype({'acuity': int})

# medication reconillation
medrecon = pd.read_csv(r'medrecon.csv', encoding = "ISO-8859-1")
df_medrecon = medrecon.dropna()
df_medrecon.head(10)
df_medrecon = df_medrecon.drop(columns = ['charttime', 'gsn', 'ndc', 'etc_rn', 'etccode'])

df_medrecon.drop_duplicates(subset=['stay_id'])

combined_med = df.merge(df_medrecon.drop_duplicates(subset=['stay_id']), how='left')
df_med = combined_med
df_med

# diagnosis
diagnosis = pd.read_csv(r'diagnosis.csv', encoding = "ISO-8859-1")
df_diagnosis = diagnosis.drop(columns = ['seq_num','icd_version', 'Unnamed: 6', 'Unnamed: 7',
'Unnamed: 8'])
df_diagnosis['icd_title'] = df_diagnosis['icd_title'].str.lower()
df_diagnosis

df_dia = df_diagnosis.drop_duplicates(subset = ['stay_id'])
```

```
combined_dia = pd.merge(df_med, df_dia, how = 'left')
# joined all necessary tables, cleaned dataset
df_final = combined_dia

df = pd.read_csv(r'df_final (2).csv', encoding = "ISO-8859-1")
df

# acuity score distribution
import seaborn as sns
ax = sns.countplot(x="acuity", data=df_final)

# visualizing each variable
fig, ax = plt.subplots(figsize=(20, 20))
df_final.hist(bins=50, ax=ax)
```

K – means clustering

```
df = pd.read_csv(r'df_final_chiefcomp.csv', encoding="ISO-8859-1")
df = df.rename(columns={'ï»¿subject_id': 'subject_id'})

# clustering with acuity and chief complaint
df = df.loc[:125]
data = list(zip(df.chief_comp_numeric, df.acuity))
print(data)

inertias = []
for i in range(1,126):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(10,10))
plt.plot(range(1,126), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
k_means_optimum = KMeans(n_clusters = 2, init = 'k-means++',  random_state=42)
y = k_means_optimum.fit_predict(data)
print(y)

plt.figure(figsize=(5,5))
plt.scatter(df.chief_comp_numeric, df.acuity,   c=kmeans.labels_)
plt.show()

from sklearn.metrics import silhouette_score
```

```
# model prediction score
score = silhouette_score(data, y)
print(score)


# elbow method - clustering icd_code and acuity

df = df.loc[:125]
data = list(zip(df.icd_code, df.acuity))
print(data)

inertias = []
for i in range(1,126):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

# plt.figure(figsize=(10,10))
plt.plot(range(1,126), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

kmeans = KMeans(n_clusters=3)
kmeans.fit(data)
plt.figure(figsize=(10,5))
# plt.scatter(df.stay_id, df.chief_comp_numeric,  c=kmeans.labels_)
plt.scatter(df.icd_code, df.acuity,  c=kmeans.labels_)
plt.show()

k_means_optimum = KMeans(n_clusters = 3, init = 'k-means++',  random_state=42)
y = k_means_optimum.fit_predict(data)
print(y)

# model prediction score
score = silhouette_score(data, y)
print(score)
```