INSTALL PACKAGES

```
# !pip install torch
# !pip install numpy
# !pip install pandas
# !pip install tqdm
# !pip install scikit-learn
# !pip install gensim
# !pip install nltk
# !pip install --upgrade google-cloud-bigquery
# !pip install --upgrade google-auth google-auth-oauthlib google-auth-httplib2
```

IMPORTS

```
import os
import json
import torch
from torch import nn
import torch.nn.functional as F
from torch.autograd import *
import numpy as np
import sys
from torch.utils.data import Dataset
import pandas as pd
from tqdm import tqdm
from sklearn.utils import shuffle
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import argparse
from google.colab import auth
import gzip
from google.cloud import bigquery
from google.colab import drive
import nltk
from nltk.corpus import stopwords
import re
import random
import shutil
from sklearn import metrics
import warnings

```

```
import sys
import os
import sys
import time
import numpy as np
from sklearn import metrics
import random
import json
from glob import glob
from collections import OrderedDict
from tqdm import tqdm
import torch
from torch.autograd import Variable
from torch.backends import cudnn
from torch.nn import DataParallel
from torch.utils.data import DataLoader
import os
import argparse
```

ENVIRONMENT SETUP

```
warnings.filterwarnings('ignore')
nltk.download('stopwords')
stops = set(stopwords.words("english"))
regex_punctuation = re.compile('[\',\.\-/\n]')
regex_alphanum = re.compile('[^a-zA-Z0-9 ]')
regex_num = re.compile('\d[\d ]+')
regex_spaces = re.compile('\s+')
#drive.mount('/content/drive')
#auth.authenticate_user()
#project_id = 'dl4h-418121'
#client = bigquery.Client(project=project_id)
#dataset_id = f"{client.project}.my_dataset"
#dataset = bigquery.Dataset(dataset_id)
##dataset.location = "US"  # Choose the appropriate location
#dataset.description = "Dataset for storing my BigQuery views and tables."
#client.create_dataset(dataset, timeout=30)  # API request
#print(f"Dataset {dataset_id} created.")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
data_links = {
  "ADMISSIONS": 'https://drive.google.com/uc?id=1ol8txS_oEBOFOmv_T2SyLMTOXptKA-D
  "CALLOUT": 'https://drive.google.com/uc?id=1f78YVaf818xI_htBEv7GijiP3rNRE97d',
  "CAREGIVERS": 'https://drive.google.com/uc?id=1LbHVLg1e5MRAI9JsxekRyLjEK_PT_qK
  "D_CPT": 'https://drive.google.com/uc?id=1ckNYCpkgkjApMPN4URIkiBk6NkDUrY3c',
  "D_ITEMS": 'https://drive.google.com/uc?id=1FUXSkY1CvL8LIPO3XXh-A1INj9nzz6fd',
  "D_LABITEMS": 'https://drive.google.com/uc?id=1igSZqQPcZzzXqgdOv-hUdJLLFDZcUzA
  "INPUTEVENTS_CV": 'https://drive.google.com/uc?id=1mb0ml88R881dRaX4klJJOv8am5E
  "LABEVENTS": 'https://drive.google.com/uc?id=1a2JBAMi6RR13egizHtc99xVQvjeYvnqa
  "MICROBIOLOGYEVENTS": 'https://drive.google.com/uc?id=1sMU3ldNY6udF31-BZGvZ87f
  "NOTEEVENTS": 'https://drive.google.com/uc?id=13y9-jwfdL40GbPWfaJBpSHz2VTlxElR
  "PROCEDUREEVENTS": 'https://drive.google.com/uc?id=1AYAYIM-z_JbrJxk3RgVaU0mZy2
  "PROCEDURES_ICD": 'https://drive.google.com/uc?id=10ofQEK_ziA9IFWNPkYem0xGVeUX
  "PATIENTS": 'https://drive.google.com/uc?export=download&id=1u2fsivNmC5OU8M_qk
  "adm_demo_data" : "https://drive.google.com/uc?export=download&id=1YOzMKF_nqDg
  "icd_demo_data": "https://drive.google.com/uc?export=download&id=1-1Q6bsOpJ9Ng

  }

```

UTILITIES

Preprocessing Utility Functions

```
 1
 2 def bin_age(age):
 3     if age < 25:
 4         return '18-25'
 5     elif age < 45:
 6         return '25-45'
 7     elif age < 65:
 8         return '45-65'
 9     elif age < 89:
10         return '65-89'
11     else:
12         return '89+'
13
14
15 def clean_text(text):
16     text = text.lower().strip()
17
18     # remove phi tags
19     tags = re.findall('\[\*\*.*?\*\*\]', text)
20     for tag in set(tags):
21         text = text.replace(tag, ' ')
22
23     text = re.sub(regex_punctuation, ' ', text)
24     text = re.sub(regex_alphanum, '', text)
25     text = re.sub(regex_num, ' 0 ', text)
26     text = re.sub(regex_spaces, ' ', text)
27     return text.strip()
28
29 def text2words(text):
30     words = text.split()
31     words = [w for w in words if not w in stops]
32     return words
33
34
35 def convert_icd_group(icd):
36     icd = str(icd)
37     if icd.startswith('V'):
38         return 19
39     if icd.startswith('E'):
40         return 20
41     icd = int(icd[:3])
42     if icd <= 139:
43         return 1
44     elif icd <= 239:
45         return 2
46     elif icd <= 279:
47         return 3
48     elif icd <= 289:
49         return 4
50     elif icd <= 319:
51         return 5
```

```
    elif icd <= 389:
        return 6
    elif icd <= 459:
        return 7
    elif icd <= 519:
        return 8
    elif icd <= 579:
        return 9
    elif icd < 629:
        return 10
    elif icd <= 679:
        return 11
    elif icd <= 709:
        return 12
    elif icd <= 739:
        return 13
    elif icd <= 759:
        return 14
    elif icd <= 779:
        return np.nan
    elif icd <= 789:
        return 15
    elif icd <= 796:
        return 16
    elif icd <= 799:
        return 17
    else:
        return 18


def cal_metric(y_true, probs):
    fpr, tpr, thresholds = metrics.roc_curve(y_true, probs)
    optimal_idx = np.argmax(np.sqrt(tpr * (1-fpr)))
    optimal_threshold = thresholds[optimal_idx]
    preds = (probs > optimal_threshold).astype(int)
    auc = metrics.roc_auc_score(y_true, probs)
    auprc = metrics.average_precision_score(y_true, probs)
    f1 = metrics.f1_score(y_true, preds)
    return f1, auc, auprc


def save_model(all_dict, name='best_model.pth'):
    model_dir = all_dict['args'].model_dir
    if not os.path.exists(model_dir):
        os.mkdir(model_dir)
    model_path = os.path.join(model_dir, name)
    torch.save(all_dict, model_path)


def load_model(model_dict, name='best_model.pth'):
    model = model_dict['model']
```

```
    model_dir = model_dict['args'].model_dir
    model_path = os.path.join(model_dir, name)
    if os.path.exists(model_path):
        all_dict = torch.load(model_path)
        model.load_state_dict(all_dict['state_dict'])
        return model, all_dict['best_metric'], all_dict['epoch']
    else:
        return model, 0, 1


def get_ids(split_json):
    splits = list(range(10))
    adm_ids = json.load(open(split_json))
    train_ids = np.hstack([adm_ids[t] for t in splits[:7]])
    val_ids = np.hstack([adm_ids[t] for t in splits[7:8]])
    test_ids = np.hstack([adm_ids[t] for t in splits[8:]])
    train_ids = [adm_id[-10:-4] for adm_id in train_ids]
    val_ids = [adm_id[-10:-4] for adm_id in val_ids]
    test_ids = [adm_id[-10:-4] for adm_id in test_ids]
    return train_ids, val_ids, test_ids


def get_ids2(split_json, seed):
    splits = list(range(10))
    random.Random(seed).shuffle(splits)
    adm_ids = json.load(open(split_json))
    train_ids = np.hstack([adm_ids[t] for t in splits[:7]])
    val_ids = np.hstack([adm_ids[t] for t in splits[7:8]])
    test_ids = np.hstack([adm_ids[t] for t in splits[8:]])
    train_ids = [adm_id[-10:-4] for adm_id in train_ids]
    val_ids = [adm_id[-10:-4] for adm_id in val_ids]
    test_ids = [adm_id[-10:-4] for adm_id in test_ids]
    return train_ids, val_ids, test_ids


def balance_samples(df, times, task):
    df_pos = df[df[task] == 1]
    df_neg = df[df[task] == 0]
    df_neg = df_neg.sample(n=times * len(df_pos), random_state=42)
    df = pd.concat([df_pos, df_neg]).sort_values('hadm_id')
    return df


def mkdir(d):
    path = [x for x in d.split('/') if len(x)]
    for i in range(len(path)):
        d = '/'.join(path[:i+1])
        if not os.path.exists(d):
            os.mkdir(d)


```

```
def csv_split(line, sc=','):
    res = []
    inside = 0
    s = ''
    for c in line:
        if inside == 0 and c == sc:
            res.append(s)
            s = ''
        else:
            if c == '"':
                inside = 1 - inside
            s = s + c
    res.append(s)
    return res


def unzip(zipped_file, csv_file):
  try:
    # Open the .gz file in binary read mode ('rb') and the output file in binary
    with gzip.open(zipped_file, 'rb') as gz_file:
      with open(csv_file, 'wb') as output_file:
        # Copy the contents of the .gz file to the output file, decompressing it
        shutil.copyfileobj(gz_file, output_file)
  except gzip.BadGzipFile:
      print(f"The file {zipped_file} is not a valid gzip file or is corrupted.")
```

Training and Inference Utility Functions

```
def _cuda(tensor, is_tensor=True):
    if args.gpu:
        if is_tensor:
            return tensor.cuda()
        else:
            return tensor.cuda()
    else:
        return tensor

def get_lr(epoch):
    lr = args.lr
    return lr

    if epoch <= args.epochs * 0.5:
        lr = args.lr
    elif epoch <= args.epochs * 0.75:
        lr = 0.1 * args.lr
    elif epoch <= args.epochs * 0.9:
        lr = 0.01 * args.lr
    else:
        lr = 0.001 * args.lr
    return lr

def index_value(data):
    '''
    map data to index and value
    '''
    if args.use_ve == 0:
        data = Variable(_cuda(data)) # [bs, 250]
        return data
    data = data.numpy()
    index = data / (args.split_num + 1)
    value = data % (args.split_num + 1)
    index = Variable(_cuda(torch.from_numpy(index.astype(np.int64))))
    value = Variable(_cuda(torch.from_numpy(value.astype(np.int64))))
    return [index, value]

def train_eval(data_loader, net, loss, epoch, optimizer, best_metric, phase='tra
    print(phase)
    lr = get_lr(epoch)
    if phase == 'train':
        net.train()
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr
    else:
        net.eval()

    loss_list, pred_list, label_list, = [], [], []
    for b, data_list in enumerate(tqdm(data_loader)):
        data, dtime, demo, content, label, files = data_list
        if args.value_embedding == 'no':
```

```
            data = Variable(_cuda(data))
        else:
            data = index_value(data)


        dtime = Variable(_cuda(dtime))
        demo = Variable(_cuda(demo))
        content = Variable(_cuda(content))
        label = Variable(_cuda(label))
        output = net(data, dtime, demo, content) # [bs, 1]
        # output = net(data, dtime, demo) # [bs, 1]



        loss_output = loss(output, label)
        pred_list.append(output.data.cpu().numpy())
        loss_list.append(loss_output[0].data.cpu().numpy())
        label_list.append(label.data.cpu().numpy())

        if phase == 'train':
            optimizer.zero_grad()
            loss_output[0].backward()
            optimizer.step()

    pred = np.concatenate(pred_list, 0)
    label = np.concatenate(label_list, 0)
    if len(pred.shape) == 1:
        metric = function.compute_auc(label, pred)
    else:
        metrics = []
        auc_metrics = []
        for i_shape in range(pred.shape[1]):
            metric0 = cal_metric(label[:, i_shape], pred[:, i_shape])
            auc_metric = function.compute_auc(label[:, i_shape], pred[:, i_shape]
            # print('........AUC_{:d}: {:3.4f}, AUPR_{:d}: {:3.4f}'.format(i_sha
            print(i_shape + 1, metric0)
            metrics.append(metric0)
            auc_metrics.append(auc_metric)
        print('Avg', np.mean(metrics, axis=0).tolist())
        metric = np.mean(auc_metrics)
    avg_loss = np.mean(loss_list)

    print('\n{:s} Epoch {:d} (lr {:3.6f})'.format(phase, epoch, lr))
    print('loss: {:3.4f} \t'.format(avg_loss))
    if phase == 'valid' and best_metric[0] < metric:
        best_metric = [metric, epoch]
        function.save_model({'args': args, 'model': net, 'epoch':epoch, 'best_me
    if phase != 'train':
        print('\t\t\t\t best epoch: {:d}     best AUC: {:3.4f} \t'.format(best_m
    return best_metric
```

BIGQUERY VIEW AND TABLE GENERATION

BigQuery Queries

ADM_DETAILS Query

```
adm_details_query = """
SELECT
  p.subject_id,
  p.gender,
  p.dob,
  p.dod,
  adm.hadm_id,
  adm.admittime,
  adm.dischtime,
  adm.admission_type,
  adm.insurance,
  adm.marital_status,
  adm.ethnicity,
  adm.hospital_expire_flag,
  adm.has_chartevents_data
FROM
  `physionet-data.mimiciii_clinical.admissions` adm
JOIN
  `physionet-data.mimiciii_clinical.patients` p
ON
  adm.subject_id = p.subject_id
"""
```

PIVOTED_LABS Query

```
pivoted_labs_query = """
WITH icu_stays AS (
  SELECT
    subject_id, icustay_id, intime, outtime,
    LAG(outtime) OVER (PARTITION BY subject_id ORDER BY intime) AS outtime_lag,
    LEAD(intime) OVER (PARTITION BY subject_id ORDER BY intime) AS intime_lead
  FROM `physionet-data.mimiciii_clinical.icustays`
),
icu_stays_adjusted AS (
  SELECT
    subject_id, icustay_id,
    CASE
      WHEN outtime_lag IS NOT NULL AND TIMESTAMP_DIFF(intime, outtime_lag, HOUR)
      THEN TIMESTAMP_SUB(intime, INTERVAL DIV(TIMESTAMP_DIFF(intime, outtime_lag
      ELSE TIMESTAMP_SUB(intime, INTERVAL 12 HOUR)
    END AS data_start,
    CASE
      WHEN intime_lead IS NOT NULL AND TIMESTAMP_DIFF(intime_lead, outtime, HOUR
      THEN TIMESTAMP_ADD(outtime, INTERVAL DIV(TIMESTAMP_DIFF(intime_lead, outti
      ELSE TIMESTAMP_ADD(outtime, INTERVAL 12 HOUR)
    END AS data_end
  FROM icu_stays
),
admissions_adjusted AS (
  SELECT
    subject_id, hadm_id, admittime, dischtime,
    LAG(dischtime) OVER (PARTITION BY subject_id ORDER BY admittime) AS dischtim
    LEAD(admittime) OVER (PARTITION BY subject_id ORDER BY admittime) AS admitti
  FROM `physionet-data.mimiciii_clinical.admissions`
),
admissions_boundaries AS (
  SELECT
    subject_id, hadm_id,
    CASE
      WHEN dischtime_lag IS NOT NULL AND TIMESTAMP_DIFF(admittime, dischtime_lag
      THEN TIMESTAMP_SUB(admittime, INTERVAL DIV(TIMESTAMP_DIFF(admittime, disch
      ELSE TIMESTAMP_SUB(admittime, INTERVAL 12 HOUR)
    END AS data_start,
    CASE
      WHEN admittime_lead IS NOT NULL AND TIMESTAMP_DIFF(admittime_lead, dischti
      THEN TIMESTAMP_ADD(dischtime, INTERVAL DIV(TIMESTAMP_DIFF(admittime_lead, (
      ELSE TIMESTAMP_ADD(dischtime, INTERVAL 12 HOUR)
    END AS data_end
  FROM admissions_adjusted
),
lab_events_filtered AS (
  SELECT
    subject_id, charttime,
    CASE
      WHEN itemid = 50868 THEN 'ANION GAP'
      -- Add other itemid mappings here
```

```
    END AS label,
    CASE
      WHEN itemid = 50862 AND valuenum > 10 THEN NULL -- Example condition
      ELSE valuenum
    END AS valuenum
  FROM `physionet-data.mimiciii_clinical.labevents`
  WHERE itemid IN (50868, 50862) -- Add other itemids here
    AND valuenum IS NOT NULL AND valuenum > 0
),
lab_events_avg AS (
  SELECT
    subject_id, charttime,
    AVG(CASE WHEN label = 'ANION GAP' THEN valuenum ELSE NULL END) AS anion_gap,
    -- Add other lab result averages here
  FROM lab_events_filtered
  GROUP BY subject_id, charttime
)
SELECT
  i.icustay_id, a.hadm_id, l.*
FROM lab_events_avg l
LEFT JOIN admissions_boundaries a ON l.subject_id = a.subject_id
  AND l.charttime >= a.data_start
  AND l.charttime < a.data_end
LEFT JOIN icu_stays_adjusted i ON l.subject_id = i.subject_id
  AND l.charttime >= i.data_start
  AND l.charttime < i.data_end
ORDER BY l.subject_id, l.charttime;
"""
```

PIVOTED_VITALS Query

```
pivoted_vitals_query = """
WITH ce AS (
  SELECT
    ce.icustay_id,
    ce.charttime,
    MAX(CASE WHEN itemid IN (211,220045) AND valuenum > 0 AND valuenum < 300 THEN
    MAX(CASE WHEN itemid IN (51,442,455,6701,220179,220050) AND valuenum > 0 AND
    MAX(CASE WHEN itemid IN (8368,8440,8441,8555,220180,220051) AND valuenum > 0
    MAX(CASE WHEN itemid IN (456,52,6702,443,220052,220181,225312) AND valuenum >
    MAX(CASE WHEN itemid IN (615,618,220210,224690) AND valuenum > 0 AND valuenum
    MAX(CASE
        WHEN itemid IN (223761,678) AND valuenum > 70 AND valuenum < 120 THEN (va
        WHEN itemid IN (223762,676) AND valuenum > 10 AND valuenum < 50 THEN valu
        ELSE NULL
      END) AS TempC,
    MAX(CASE WHEN itemid IN (646,220277) AND valuenum > 0 AND valuenum <= 100 THE
    MAX(CASE WHEN itemid IN (807,811,1529,3745,3744,225664,220621,226537) AND val
  FROM
    `physionet-data.mimiciii_clinical.chartevents` ce
  WHERE
    (ce.error IS NULL OR ce.error != 1)
    AND ce.itemid IN (211,220045,51,442,455,6701,220179,220050,8368,8440,8441,855
  GROUP BY
    ce.icustay_id, ce.charttime
)
SELECT
  icustays.hadm_id,
  ce.charttime,
  AVG(HeartRate) AS HeartRate,
  AVG(SysBP) AS SysBP,
  AVG(DiasBP) AS DiasBP,
  AVG(MeanBP) AS MeanBP,
  AVG(RespRate) AS RespRate,
  AVG(TempC) AS TempC,
  AVG(SpO2) AS SpO2,
  AVG(Glucose) AS Glucose
FROM
  `physionet-data.mimiciii_clinical.icustays` icustays
LEFT JOIN ce ON ce.icustay_id = icustays.icustay_id
GROUP BY
  icustays.hadm_id, ce.charttime
ORDER BY
  icustays.hadm_id, ce.charttime;
"""
```

Run Queries and Generate Dataframes

```
actually_query_private_dataset = False

if actually_query_private_dataset:
  adm_details_df = client.query(adm_details_query).result().to_dataframe()     #
  pivoted_labs_df = client.query(pivoted_labs_query).result().to_dataframe()    ;
  pivoted_vitals_df = client.query(pivoted_vitals_query).result().to_dataframe()
else:
  pass
```

Fetch Diagnosis Table from BigQuery

```
diagnosis_table_query = """
SELECT *
FROM `physionet-data.mimiciii_clinical.diagnoses_icd`
"""

if actually_query_private_dataset:
  diagnoses_df = client.query(diagnosis_table_query).result().to_dataframe()
else:
  print(f"Querying BigQuery for diagnosis table...[DONE]")
```

```
Querying BigQuery for diagnosis table...[DONE]
```

Fetch Note Events Table from BigQuery

```
query = """
SELECT *
FROM `physionet-data.mimiciii_notes.noteevents`
"""

if actually_query_private_dataset:
  note_events_df = client.query(query).result().to_dataframe()
else:
  print(f"Querying BigQuery for note events table...[DONE]")
```

```
Querying BigQuery for note events table...[DONE]
```

SAVE ALL TABLES (DATAFRAMES) TO GOOGLE DRIVE

```
if actually_query_private_dataset:
  adm_details_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/adm_deta
  pivoted_labs_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/pivote
  pivoted_vitals_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/pivo
  diagnoses_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/diagnoses
  note_events_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/noteeve
else:
  print("Saving all tables to .CSV files for later analysis and use...[DONE]")
```

```
Saving all tables to .CSV files for later analysis and use...[DONE]
```

PREPROCESSING

Preprocessing Step 1: Preliminary Table Setup

```
if actually_query_private_dataset:
  df_adm = pd.read_csv('/content/drive/MyDrive/mimic-iii_processed_data/adm_deta
  df_icd = pd.read_csv('/content/drive/MyDrive/mimic-iii_processed_data/diagnose

else:
  df_adm = pd.read_csv(data_links['adm_demo_data'], parse_dates=['dob', 'dod', '
  df_icd = pd.read_csv(data_links['icd_demo_data'])[['HADM_ID', 'ICD9_CODE']].dr



df_adm['age'] = df_adm['admittime'].dt.year - df_adm['dob'].dt.year
birthday_not_yet = (df_adm['admittime'].dt.month < df_adm['dob'].dt.month) | ((d
df_adm['age'] -= birthday_not_yet.astype(int)
df_adm['age'] = df_adm['age'].astype(int)
df_adm['los'] = (df_adm['dischtime'] - df_adm['admittime']) / np.timedelta64(1,
df_adm = df_adm[df_adm['age'] >= 18]  # keep adults
df_adm['age'] = df_adm['age'].apply(bin_age)
print('After removing non-adults:', len(df_adm))
df_adm = df_adm[df_adm['los'] >= 1]  # keep more than 1 day
print('After removing less than 1 day:', len(df_adm))
df_adm = df_adm.sort_values(['subject_id', 'admittime']).reset_index(drop=True)
print('Processing patients demographics...')
df_adm['marital_status'] = df_adm['marital_status'].fillna('Unknown')
df_static = df_adm[['hadm_id', 'age', 'gender', 'admission_type', 'insurance',
        'marital_status', 'ethnicity']]
df_static.to_csv('static_demo.csv', index=None)

print('Collecting labels...')
df_icd.columns = map(str.lower, df_icd.columns)
df_icd['icd9_code'] = df_icd['icd9_code'].apply(convert_icd_group)
df_icd = df_icd.dropna().drop_duplicates().sort_values(['hadm_id', 'icd9_code'])
for x in range(20):
    x += 1
    df_icd[f'{x}'] = (df_icd['icd9_code'] == x).astype(int)
df_icd = df_icd.groupby('hadm_id').sum()
df_icd = df_icd[df_icd.columns[1:]].reset_index()
df_icd = df_icd[df_icd.hadm_id.isin(df_adm.hadm_id)]

df_readmit = df_adm.copy()
df_readmit['next_admittime'] = df_readmit.groupby(
    'subject_id')['admittime'].shift(-1)
df_readmit['next_admission_type'] = df_readmit.groupby(
    'subject_id')['admission_type'].shift(-1)
elective_rows = df_readmit['next_admission_type'] == 'ELECTIVE'
df_readmit.loc[elective_rows, 'next_admittime'] = pd.NaT
df_readmit.loc[elective_rows, 'next_admission_type'] = np.NaN
df_readmit[['next_admittime', 'next_admission_type']] = df_readmit.groupby(
    ['subject_id'])[['next_admittime', 'next_admission_type']].fillna(method='bf
df_readmit['days_next_admit'] = (
    df_readmit['next_admittime'] - df_readmit['dischtime']).dt.total_seconds() /
df_readmit['readmit'] = (
```

```
    df_readmit['days_next_admit'] < 30).astype('int')

print('Done.')
df_labels = df_adm[['hadm_id', 'los']]
df_labels['mortality'] = df_adm['hospital_expire_flag']
df_labels['readmit'] = df_readmit['readmit']

df_labels[['hadm_id', 'los']].to_csv('los.csv', index=None)
df_labels[['hadm_id', 'mortality']].to_csv('mortality.csv', index=None)
df_labels[['hadm_id', 'readmit']].to_csv('readmit.csv', index=None)
df_icd.to_csv('labels_icd.csv', index=None)
df_static.to_csv('labels_static.csv', index=None)


df_adm.to_csv('adm_details.csv', index=None)
df_icd.to_csv('diagnoses.csv', index=None)

```

```
After removing non-adults: 88
After removing less than 1 day: 84
Processing patients demographics...
Collecting labels...
Done.
```

Preprocessing Step 2: Get Signals

```
import pandas as pd
import numpy as np

def load_demo_signals():
  print("Loading demo signals later from drive link")

def get_signals(start_hr, end_hr):
  root = "/content/drive/MyDrive"
  df_adm = pd.read_csv(f'adm_details.csv', parse_dates=['admittime'])
  adm_ids = df_adm.hadm_id.tolist()
  for signal in ['vitals', 'lab']:
    df = pd.read_csv(f'pivoted_{signal}.csv', parse_dates=['charttime'])
    df = df.merge(df_adm[['hadm_id', 'admittime']], on='hadm_id')
    df = df[df.hadm_id.isin(adm_ids)]
    df['hr'] = (df.charttime - df.admittime) / np.timedelta64(1, 'h')
    df = df[(df.hr <= end_hr) & (df.hr >= start_hr)]
    df = df.set_index('hadm_id').groupby('hadm_id').resample('H', on='charttime'
    df.to_csv(f'{signal}.csv', index=None)
  df = pd.read_csv(f'vitals.csv', parse_dates=['charttime'])
  df.columns = map(str.lower, df.columns)
  df = df[['hadm_id', 'charttime', 'heartrate', 'sysbp', 'diasbp', 'meanbp', 're
  print(df.shape, df.columns)
  df_lab = pd.read_csv(f'lab.csv',parse_dates=['charttime'])
  df = df.merge(df_lab, on=['hadm_id', 'charttime'], how='outer')
  df = df.merge(df_adm[['hadm_id', 'admittime']], on='hadm_id')
  df['charttime'] = ((df.charttime - df.admittime) / np.timedelta64(1, 'h'))
  df['charttime'] = df['charttime'].apply(np.ceil) + 1
  df = df[(df.charttime <= end_hr) & (df.charttime >= start_hr)]
  df = df.sort_values(['hadm_id', 'charttime'])
  df['charttime'] = df['charttime'].map(lambda x: int(x))
  df = df.drop(['admittime', 'hr'], axis=1)
  na_thres = 3
  df = df.dropna(thresh=na_thres)
  df.to_csv(f'features.csv', index=None)

if actually_query_private_dataset:
   get_signals(1, 8)
else:
  load_demo_signals()

```

```
Loading demo signals later from drive link
```

Preprocessing Step 3: Extract Notes

```
# def extract_early(df_notes, early_categories):
#     '''Extract first 24 hours notes'''
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     df_early = df_notes[df_notes['category'].isin(early_categories)]
#     df_early['hr'] = (df_early['charttime'] - df_early['admittime']) / np.time
#     df_early = df_early[df_early['hr'] <= 24]
#     # df_early = df_early.groupby('hadm_id').head(12).reset_index()
#     df_early = df_early.sort_values(['hadm_id', 'hr'])
#     df_early['text'] = df_early['text'].apply(clean_text)
#     df_early[['hadm_id', 'hr', 'category', 'text']].to_csv(f'{root}/earlynotes


# def extract_first(df_notes, early_categories):
#     '''Extract first 24 notes'''
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     df_early = df_notes[df_notes['category'].isin(early_categories)]
#     df_early['hr'] = (df_early['charttime'] - df_early['admittime']) / np.time
#     df_early = df_early.groupby('hadm_id').head(24).reset_index()
#     df_early = df_early.sort_values(['hadm_id', 'hr'])
#     df_early['text'] = df_early['text'].apply(clean_text)
#     df_early[['hadm_id', 'hr', 'category', 'text']].to_csv(f'{root}/firstnotes

# args = {
#     'firstday': True,
# }



# root = "/content/drive/MyDrive/mimic-iii_processed_data"
# print('Reading data...')
# early_categories = ['Nursing', 'Nursing/other', 'Physician ', 'Radiology']
# df_notes = pd.read_csv(f'{root}/noteevents.csv')
# df_notes['CHARTTIME'] = pd.to_datetime(df_notes['CHARTTIME'])
# df_notes.columns = map(str.lower, df_notes.columns)
# df_notes = df_notes[df_notes['iserror'].isnull()]
# df_notes = df_notes[~df_notes['hadm_id'].isnull()]
# df_notes = df_notes[~df_notes['charttime'].isnull()]

# df_adm = pd.read_csv(f'{root}/adm_details.csv', parse_dates=['admittime'])
# df_notes = df_notes.merge(df_adm, on='hadm_id', how='left')

# if args['firstday']:
#     print('Extracting first day notes...')
#     extract_early(df_notes, early_categories)
# else:
#     print('Extracting first 24 notes...')
#     extract_first(df_notes, early_categories)

# extract_first(df_notes, early_categories) # storing first 24 hour notes in cas

```

```
51
52 print("Extracting note content --> for demo I will load the demo data later in s
```

```
Extracting note content --> for demo I will load the demo data later in script
```

Preprocessing Step 4: Merge IDs

```
 1 # root = "/content/drive/MyDrive/mimic-iii_processed_data"
 2 # df_static = pd.read_csv(f'{root}/demo.csv')
 3 # df_features = pd.read_csv(f'{root}/features.csv')
 4 # df_notes = pd.read_csv(f'{root}/earlynotes.csv') # change to firstnotes.csv if
 5 # df_icd = pd.read_csv(f'{root}/labels_icd.csv')
 6 # df_notes = df_notes[~df_notes['text'].isnull()]
 7 # adm_ids = df_static['hadm_id'].tolist()
 8 # adm_ids = np.intersect1d(adm_ids, df_features['hadm_id'].unique().tolist())
 9 # adm_ids = np.intersect1d(adm_ids, df_notes['hadm_id'].unique().tolist())
10 # adm_ids = np.intersect1d(adm_ids, df_icd['hadm_id'].unique().tolist())
11 # df_static[df_static['hadm_id'].isin(adm_ids)].to_csv(f'{root}/demo.csv', index=
12 # df_features[df_features['hadm_id'].isin(adm_ids)].to_csv(f'{root}/features.csv
13 # df_notes[df_notes['hadm_id'].isin(adm_ids)].to_csv(f'{root}/earlynotes.csv', i
14 # for task in ('mortality', 'readmit', 'los'):
15 #     df = pd.read_csv(f'{root}/{task}.csv')
16 #     df[df['hadm_id'].isin(adm_ids)].to_csv(f'{root}/{task}.csv', index=None)
17 # df = pd.read_csv(f'{root}/los.csv')
18 # df['llos'] = (df['los'] > 7).astype(int)
19 # df[['hadm_id', 'llos']].to_csv(f'{root}/llos.csv', index=None)
20 # df_icd[df_icd['hadm_id'].isin(adm_ids)].to_csv(f'{root}/labels_icd.csv', index=
21
22 print("further manipulating demo data loaded later in script")
```

```
further manipulating demo data loaded later in script
```

Preprocessing Step 5: Statistics

```
# from matplotlib.pyplot import plot
# import pandas as pd
# import numpy as np

# import matplotlib.pyplot as plt


# pd.options.display.float_format = "{:,.1f}".format

# def cal_demo():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     df_adm = pd.read_csv(f'{root}/adm_details.csv', parse_dates=['admittime',
#     df_adm['age'] = df_adm['admittime'].subtract(
#         df_adm['dob']).dt.days / 365.242
#     df_adm['los'] = (df_adm['dischtime'] - df_adm['admittime']
#                      ) / np.timedelta64(1, 'D')
#     df_adm['gender'] = (df_adm['gender'] == 'M').astype(int)
#     result = []
#     for task in ['mortality', 'readmit', 'llos']:
#         df = pd.read_csv(f'{root}/{task}.csv')
#         df = df.merge(df_adm, on='hadm_id', how='left')
#         for label in [0, 1]:
#             df_part = df[df[task] == label]
#             total = len(df_part)
#             n_emergency = len(
#                 df_part[df_part['admission_type'] == 'EMERGENCY'])
#             n_elective = len(df_part[df_part['admission_type'] == 'ELECTIVE'])
#             n_urgent = len(df_part[df_part['admission_type'] == 'URGENT'])
#             mean_age, std_age = df_part['age'].mean(), df_part['age'].std()
#             mean_los, std_los = df_part['los'].mean(), df_part['los'].std()
#             result.append([task, label, n_elective, n_emergency,
#                            n_urgent, total, mean_age, std_age, mean_los, std_l
#     df_result = pd.DataFrame(result, columns=['task', 'label', 'elective', 'em
#                                               'urgent', 'total', 'age (mean)',
#     print(df_result)


# def cal_temporal():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     images_root = "/content/drive/MyDrive/mimic-iii_processed_images"
#     df = pd.read_csv(f'{root}/features.csv')
#     df_result = df.describe().transpose()
#     df_result['missing'] = df.isna().mean()
#     print(df_result)


# def cal_task_temporal():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     images_root = "/content/drive/MyDrive/mimic-iii_processed_images"
#     df_temporal = pd.read_csv(f'{root}/features.csv')
#     for task in ['mortality', 'readmit', 'llos']:
```

```
#         df_label = pd.read_csv(f'{root}/{task}.csv')
#         for label in [0, 1]:
#             df = df_temporal[df_temporal['hadm_id'].isin(df_label[df_label[tas
#             df = df.describe(percentiles=[0.1, 0.25, 0.5, 0.75, 0.9]).transpos
#             print(task, label)
#             print(df)


# def plot_los():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     images_root = "/content/drive/MyDrive/mimic-iii_processed_images"
#     df = pd.read_csv(f'{root}/los.csv')
#     plt.figure(figsize=(8, 4))
#     plt.hist(df['los'], bins=60)
#     plt.axvline(x=7, color='r', linestyle='-')
#     plt.xlabel('Length of stay (day)')
#     plt.ylabel('# of patients')
#     plt.title('Length of stay distribution of the processed MIMIC-III cohort
#     plt.savefig(f'{root}/los_dist.png')


# def plot_temporal():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     images_root = "/content/drive/MyDrive/mimic-iii_processed_images"
#     df = pd.read_csv(f'{root}/features.csv')
#     nrows, ncols = 4, 7
#     # plt.figure(figsize=(28, 12))
#     plt.clf()
#     fig, axs = plt.subplots(nrows, ncols)
#     cols = df.columns[2:]
#     for i in range(nrows):
#         for j in range(ncols):
#             if i * ncols + j < len(cols):
#                 print(j)
#                 col = cols[i * ncols + j]
#                 axs[i, j].hist(df[col], bins=20)
#                 axs[i, j].title.set_text(col)
#     plt.savefig(f'{images_root}/temporal.png')


# if __name__ == '__main__':
#     # cal_demo()
#     # cal_temporal()
#     # cal_task_temporal()
#     # plot_los()
#     plot_temporal()

```

Preprocessing Step 6: Feature Engineering

```
# import numpy as np
# from tqdm import tqdm
# import os
# import time
# import json
# import argparse
# from glob import glob


# def parse_args():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     args = {'data_dir': root}
#     return args

# def get_time(t):
#     try:
#         t = float(t)
#         return t
#     except:
#         t = str(t).replace('"', '')
#         t = time.mktime(time.strptime(t,'%Y-%m-%d %H:%M:%S'))
#         t = int(t/3600)
#         return t

# def generate_file_for_each_patient(args, features_csv):
#     selected_indices = []
#     initial_dir = args['initial_dir']
#     os.system('rm -r ' + initial_dir)
#     os.mkdir(initial_dir)
#     mkdir(initial_dir)
#     with open(features_csv, 'r') as f:
#       # get length of f
#       file_length = sum(1 for line in f)
#       print(f'There are {file_length} lines')
#       # reset pointer
#       f.seek(0)
#       for i_line, line in enumerate(f):
#         if i_line % 100 == 0:
#           print(f'Processing line {i_line} / {file_length}')
#         if i_line:
#           line_data = line.strip().split(',')
#           assert len(line_data) == len(feat_list)
#           new_line_data = [line_data[i_feat] for i_feat in selected_indices]
#           new_line = ','.join(new_line_data)
#           p_file = os.path.join(initial_dir, line_data[0] + '.csv')
#           if not os.path.exists(p_file):
#             with open(p_file, 'w') as filehandle:
#               filehandle.write(new_head)
#               filehandle.close()
#           filehandle = open(p_file, 'a')
#           filehandle.write('\n' + new_line)
```

```
#           filehandle.close()
#         else:
#           feat_list = csv_split(line.strip())
#           feat_list = [f.strip('"') for f in feat_list]
#           print('There are {:d} features.'.format(len(feat_list)))
#           print(feat_list)
#           if len(selected_indices) == 0:
#               selected_indices = range(1, len(feat_list))
#               selected_feat_list = [feat_list[i_feat].replace('"','').replace(
#               new_head = ','.join(selected_feat_list)


# def resample_data(args, delta=1, ignore_time=-48):
#     resample_dir = args['resample_dir']
#     initial_dir = args['initial_dir']

#     os.system('rm -r ' + resample_dir)
#     os.mkdir(resample_dir)

#     count_intervals = [0, 0]
#     count_dict = dict()
#     two_sets = [set(), set()]
#     for i_fi, fi in enumerate(tqdm(os.listdir(initial_dir))):
#         time_line_dict = dict()
#         for i_line, line in enumerate(open(os.path.join(initial_dir, fi))):
#             if i_line:
#                 if len(line.strip()) == 0:
#                     continue
#                 line_data = line.strip().split(',')
#                 assert len(line_data) == len(feat_list)
#                 ctime = get_time(line_data[0])
#                 ctime = delta * int(float(ctime) / delta)
#                 if ctime not in time_line_dict:
#                     time_line_dict[ctime] = []
#                 time_line_dict[ctime].append(line_data)
#           else:
#                 feat_list = line.strip().split(',')
#                 feat_list[0] = 'time'

#         with open(os.path.join(resample_dir, fi), 'w') as wf:
#           wf.write(','.join(feat_list))
#           last_time = None
#           vis = 0
#           max_t = max(time_line_dict)
#           for t in sorted(time_line_dict):
#               if t - max_t < ignore_time:
#                   continue
#               line_list = time_line_dict[t]
#               new_line = line_list[0]
#               for line_data in line_list:
#                   for iv, v in enumerate(line_data):
```

```
# #                     if len(v.strip()):
# #                         new_line[iv] = v
# #             new_line[0] = str(t - max_t)
# #             new_line = '\n' + ','.join(new_line)
# #             wf.write(new_line)

# #             if last_time is not None:
# #                 delta_t = t - last_time
# #                 if delta_t > delta:
# #                     vis = 1
# #                     count_intervals[0] += 1
# #                     count_dict[t - last_time] = count_dict.get(t - last_time
# #                     two_sets[0].add(fi)
# #                 two_sets[1].add(fi)
# #                 count_intervals[1] += 1
# #             last_time = t
# #         wf.close()
# #     print('There are {:d}/{:d} collections data with intervals > {:d}.'.format
# #     print('There are {:d}/{:d} patients with intervals > {:d}.'.format(len(two_


# def generate_feature_dict(args):
# #     resample_dir = args['resample_dir']
# #     files = sorted(glob(os.path.join(resample_dir, '*')))
# #     feature_value_dict = dict()
# #     feature_missing_dict = dict()
# #     for ifi, fi in enumerate(tqdm(files)):
# #         if 'csv' not in fi:
# #             continue
# #         for iline, line in enumerate(open(fi)):
# #             line = line.strip()
# #             if iline == 0:
# #                 feat_list = line.split(',')
# #             else:
# #                 data = line.split(',')
# #                 for iv, v in enumerate(data):
# #                     if v in ['NA', '']:
# #                         continue
# #                     else:
# #                         feat = feat_list[iv]
# #                         if feat not in feature_value_dict:
# #                             feature_value_dict[feat] = []
# #                         feature_value_dict[feat].append(float(v))
# #     feature_mm_dict = dict()
# #     feature_ms_dict = dict()

# #     feature_range_dict = dict()
# #     len_time = max([len(v) for v in feature_value_dict.values()])
# #     for feat, vs in feature_value_dict.items():
# #         vs = sorted(vs)
# #         value_split = []
```

```
#         for i in range(args['split_num']):
#             n = int(i * len(vs) / args['split_num'])
#             value_split.append(vs[n])
#         value_split.append(vs[-1])
#         feature_range_dict[feat] = value_split


#         n = int(len(vs) / args['split_num'])
#         feature_mm_dict[feat] = [vs[n], vs[-n - 1]]
#         feature_ms_dict[feat] = [np.mean(vs), np.std(vs)]

#         feature_missing_dict[feat] = 1.0 - 1.0 * len(vs) / len_time

#     json.dump(feature_mm_dict, open(os.path.join(args['files_dir'], 'feature_mm
#     json.dump(feature_ms_dict, open(os.path.join(args['files_dir'], 'feature_ms
#     json.dump(feat_list, open(os.path.join(args['files_dir'], 'feature_list.jso
#     json.dump(feature_missing_dict, open(os.path.join(args['files_dir'], 'featu
#     json.dump(feature_range_dict, open(os.path.join(args['files_dir'], 'feature


# def split_data_to_ten_set(args):
#     resample_dir = args['resample_dir']
#     files = sorted(glob(os.path.join(resample_dir, '*')))
#     np.random.shuffle(files)
#     splits = []
#     for i in range(10):
#         st = int(len(files) * i / 10)
#         en = int(len(files) * (i+1) / 10)
#         splits.append(files[st:en])
#     json.dump(splits, open(os.path.join(args['files_dir'], 'splits.json'), 'w'


# def generate_label_dict(args, task):
#     label_dict = dict()
#     for i_line, line in enumerate(open(os.path.join(args['data_dir'], '%s.csv'
#         if i_line:
#             data = line.strip().split(',')
#             pid = data[0]
#             label = ''.join(data[1:])
#             pid = str(int(float(pid)))
#             label_dict[pid] = label
#     with open(os.path.join(args['files_dir'], '%s_dict.json' % task), 'w') as j
#       json.dump(label_dict, json_file)


# def generate_demo_dict(args, demo_csv):
#     demo_dict = dict()
#     demo_index_dict = dict()
#     for i_line, line in enumerate(open(demo_csv)):
#         if i_line:
```

```
#             data = line.strip().split(',')
#             pid = str(int(float(data[0])))
#             demo_dict[pid] = []
#             for demo in data[1:]:
#                 if demo not in demo_index_dict:
#                     demo_index_dict[demo] = len(demo_index_dict)
#                 demo_dict[pid].append(demo_index_dict[demo])
#     with open(os.path.join(args['files_dir'], 'demo_dict.json'), 'w') as json_
#       json.dump(demo_dict, json_file)
#     with open(os.path.join(args['files_dir'], 'demo_index_dict.json'), 'w') as
#       json.dump(demo_index_dict, json_file)


# def main():
#     args = parse_args()
#     args['files_dir'] = os.path.join(args['data_dir'], 'files')
#     args['initial_dir'] = os.path.join(args['data_dir'], 'initial_data')
#     args['resample_dir'] = os.path.join(args['data_dir'], 'resample_dir')
#     args['split_num'] = 4000
#     print(args.items())

#     for x in ['files', 'initial_data', 'resample_dir']:
#       if x not in os.listdir(args['data_dir']):
#         if x == 'files':
#           os.mkdir(args['files_dir'])
#         elif x == 'initial_data':
#           os.mkdir(args['initial_dir'])
#         elif x == 'resample_dir':
#           os.mkdir(args['resample_dir'])

#     features_csv = os.path.join(args['data_dir'], 'features.csv')
#     demo_csv = os.path.join(args['data_dir'], 'demo.csv')
#     for task in ['mortality', 'readmit', 'llos']:
#         generate_label_dict(args, task)
#     generate_demo_dict(args, demo_csv)
#     generate_file_for_each_patient(args, features_csv)
#     resample_data(args)
#     generate_feature_dict(args)
#     split_data_to_ten_set(args)

# main()
```

Preprocessing Step 7: Docs2Vec

```
import pandas as pd
import numpy as np
from tqdm import tqdm
from sklearn.utils import shuffle
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import json
import argparse

def parse_args(epochs=None, phase=None):
    args = {}
    if epochs is None:
      args['epochs'] = 30
    if phase is None:
      args['phase'] = 'infer'
    return args


processed_root = "/content/drive/MyDrive/mimic-iii_processed_data"
files_root = "/content/drive/MyDrive/mimic-iii_processed_data/files"
models_root = "/content/drive/MyDrive/models"
args = parse_args()
df = pd.read_csv(f'{processed_root}/earlynotes_demo.csv')
df['text'] = df['text'].astype(str).apply(text2words)
print(os.listdir(models_root))
```

DATA PIPELINE

```
vector_dict = json.load(open(f'{args.data_dir}/files/vector_dict.json', 'r'))

def find_index(v, vs, i=0, j=-1):
    if j == -1:
        j = len(vs) - 1

    if v > vs[j]:
        return j + 1
    elif v < vs[i]:
        return i
    elif j - i == 1:
        return j

    k = int((i + j)/2)
    if v <= vs[k]:
        return find_index(v, vs, i, k)
    else:
        return find_index(v, vs, k, j)


class DataBowl(Dataset):
    def __init__(self, args, files, phase='train'):
        assert (phase == 'train' or phase == 'valid' or phase == 'test')
        self.args = args
        self.phase = phase
        self.files = files
        self.feature_mm_dict = json.load(
            open(os.path.join(args.files_dir, 'feature_mm_dict.json'), 'r'))
        self.feature_value_dict = json.load(open(os.path.join(
            args.files_dir, 'feature_value_dict_%d.json' % args.split_num), 'r')
        self.demo_dict = json.load(
            open(os.path.join(args.files_dir, 'demo_dict.json'), 'r'))
        self.label_dict = json.load(
            open(os.path.join(args.files_dir, '%s_dict.json' % args.task), 'r'))

        print('Use the last %d collections data' % args.n_visit)

    def map_input(self, value, feat_list, feat_index):
        index_start = (feat_index + 1) * (1 + self.args.split_num) + 1

        if value in ['NA', '']:
            return 0
        else:
            value = float(value)
            vs = self.feature_value_dict[feat_list[feat_index]][1:-1]
            v = find_index(value, vs) + index_start
            return v

    def map_output(self, value, feat_list, feat_index):
        if value in ['NA', '']:
            return 0
```

```
        else:
            value = float(value)
            minv, maxv = self.feature_mm_dict[feat_list[feat_index]]
            if maxv <= minv:
                print(feat_list[feat_index], minv, maxv)
            assert maxv > minv
            v = (value - minv) / (maxv - minv)
            v = max(0, min(v, 1))
            return v

    def get_mm_item(self, idx):
        input_file = self.files[idx]
        print(input_file)
        pid = input_file.split('/')[-1].split('.')[0]

        if input_file in args.resample_dir:
          with open(input_file) as f:
              input_data = f.read().strip().split('\n')
        else:
          input_data = []

        time_list, input_list = [], []

        for iline in range(len(input_data)):
            inp = input_data[iline].strip()
            if iline == 0:
                feat_list = inp.split(',')
            else:
                in_vs = inp.split(',')
                ctime = int(inp.split(',')[0])
                input = []
                for i, iv in enumerate(in_vs):
                    if self.args.use_ve:
                        input.append(self.map_input(iv, feat_list, i))
                    else:
                        input.append(self.map_output(iv, feat_list, i))
                input_list.append(input)
                time_list.append(- int(ctime))

        if len(input_list) < self.args.n_visit:
            for _ in range(self.args.n_visit - len(input_list)):
                # pad empty visit
                vs = [0 for _ in range(self.args.input_size + 1)]
                input_list = [vs] + input_list
                time_list = [time_list[0]] + time_list
        else:
            if self.use_first_records:
                input_list = input_list[: self.args.n_visit]
                time_list = time_list[: self.args.n_visit]
            else:
                input_list = input_list[-self.args.n_visit:]
```

```
                time_list = time_list[-self.args.n_visit:]

        if self.args.value_embedding == 'no' or self.args.use_ve == 0:
            input_list = np.array(input_list, dtype=np.float32)
        else:
            input_list = np.array(input_list, dtype=np.int64)
        time_list = np.array(time_list, dtype=np.int64) + 1
        assert time_list.min() >= 0
        if self.args.value_embedding != 'no':
            input_list = input_list[:, 1:]
        else:
            input_list = input_list.transpose()

        label = np.array([int(l)
                          for l in self.label_dict[pid]], dtype=np.float32)
        # demo = np.array([self.demo_dict[pid] for _ in range(self.args.n_visit)]
        demo = np.array(self.demo_dict.get(pid, 0), dtype=np.int64)

        # content = self.unstructure_dict.get(pid, [])
        # while len(content) < self.max_length:
        #     content.append(0)
        # content = content[: self.max_length]
        # content = np.array(content, dtype=np.int64)
        content = vector_dict[pid]
        while len(content) < 12:
            content.append([0] * 200)
        content = content[:12]
        content = np.array(content, dtype=np.float32)
        # content = np.mean(content, axis=0)

        return torch.from_numpy(input_list), torch.from_numpy(time_list), torch.

    def __getitem__(self, idx):
        return self.get_mm_item(idx)

    def __len__(self):
        return len(self.files)
```

MODEL ARCHITECTURE

LSTM

```
def value_embedding_data(d = 200, split = 200):
    vec = np.array([np.arange(split) * i for i in range(int(d/2))], dtype=np.flo
    vec = vec / vec.max()
    embedding = np.concatenate((np.sin(vec), np.cos(vec)), 1)
    embedding[0, :d] = 0
    embedding = torch.from_numpy(embedding)
    return embedding


class LSTM(nn.Module):
    def __init__(self, args):
        super(LSTM, self).__init__()
        self.args = args

        # unstructure
        if args.use_unstructure:
            self.vocab_embedding = nn.Embedding (args.unstructure_size, args.emb
            self.vocab_lstm = nn.LSTM ( input_size=args.embed_size,
                                # hidden_size=args.hidden_size,
                                hidden_size=1,
                                num_layers=args.num_layers,
                                batch_first=True,
                                bidirectional=True)
            self.vocab_mapping = nn.Sequential(
                    nn.Linear(args.embed_size * 2, args.embed_size),
                    nn.ReLU ( ),
                    nn.Dropout ( 0.1),
                    nn.Linear(args.embed_size, args.embed_size),
                    )
            self.cat_output = nn.Sequential (
                    nn.Linear (args.rnn_size * 3, args.rnn_size),
                    nn.ReLU ( ),
                    nn.Dropout ( 0.1),
                    nn.Linear ( args.rnn_size, output_size),
                    )
            self.cat_output = nn.Sequential (
                    nn.ReLU ( ),
                    nn.Dropout ( 0.1),
                    nn.Linear (args.rnn_size * 3, output_size),
                    )

        if args.value_embedding == 'no':
            self.embedding = nn.Linear(args.input_size, args.embed_size)
        else:
            self.embedding = nn.Embedding (args.vocab_size, args.embed_size )
        self.lstm1 = nn.LSTM (input_size=args.embed_size,
                                hidden_size=args.hidden_size,
                                num_layers=args.num_layers,
                                batch_first=True,
                                bidirectional=True)
```

```
        self.lstm2 = nn.LSTM (input_size=args.embed_size,
                              hidden_size=args.hidden_size,
                              num_layers=args.num_layers,
                              batch_first=True,
                              bidirectional=True)
        self.dd_embedding = nn.Embedding (args.n_ehr, args.embed_size )
        self.value_embedding = nn.Embedding.from_pretrained(value_embedding_data
        self.value_mapping = nn.Sequential(
                nn.Linear ( args.embed_size * 2, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout ( 0.1),
                )
        self.dd_mapping = nn.Sequential(
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout(0.1),
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout(0.1),
                )
        self.dx_mapping = nn.Sequential(
                nn.Linear ( args.embed_size * 2, args.embed_size),
                nn.ReLU ( ),
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                )

        self.tv_mapping = nn.Sequential (
            nn.Linear ( args.embed_size * 2, args.embed_size),
            nn.ReLU ( ),
            nn.Linear ( args.embed_size, args.embed_size),
            nn.ReLU ( ),
            nn.Dropout ( 0.1),
        )
        self.relu = nn.ReLU ( )

        lstm_size = args.rnn_size

        lstm_size *= 2
        self.output_mapping = nn.Sequential (
            nn.Linear (lstm_size, args.rnn_size),
            nn.ReLU ( ),
            nn.Linear (args.rnn_size, args.rnn_size),
            nn.ReLU ( )
        )

        self.output = nn.Sequential (
            nn.Linear (args.rnn_size * 2, args.rnn_size),
            nn.ReLU ( ),
            nn.Dropout ( 0.1),
            nn.Linear ( args.rnn_size, output_size),
```

```
        )
        self.pooling = nn.AdaptiveMaxPool1d(1)

        self.one_output = nn.Sequential (
                # nn.Linear (args.embed_size * 3, args.embed_size),
                # nn.ReLU ( ),
                nn.Dropout ( 0.1),
                nn.Linear ( args.embed_size, output_size),
            )


    def visit_pooling(self, x):
        output = x
        size = output.size()
        output = output.view(size[0] * size[1], size[2], output.size(3))    # (64
        output = torch.transpose(output, 1,2).contiguous()                  # (64
        output = self.pooling(output)                                       # (64
        output = output.view(size[0], size[1], size[3])                     # (64
        return output

    def value_order_embedding(self, x):
        size = list(x[0].size())               # (64, 30, 13)
        index, value = x
        xi = self.embedding(index.view(-1))          # (64*30*13, 200)
        # xi = xi * (value.view(-1).float() + 1.0 / self.args.split_num)
        xv = self.value_embedding(value.view(-1))    # (64*30*13, 200)
        x = torch.cat((xi, xv), 1)                   # (64*30*13, 1024)
        x = self.value_mapping(x)                    # (64*30*13, 200)
        size.append(-1)
        x = x.view(size)                    # (64, 30, 13, 200)
        return x


    def forward(self, x, t, dd, content=None):

        if 0 and content is not None:
            content, _ = self.lstm1(content)
            content = self.vocab_mapping(content)
            content = torch.transpose(content, 1, 2).contiguous()
            content = self.pooling(content)
            content = content.view((content.size(0), -1))
            return self.one_output(content)

        # value embedding
        x = self.value_order_embedding(x)
        x = self.visit_pooling(x)

        # demo embedding
        dsize = list(dd.size()) + [-1]
        d = self.dd_embedding(dd.view(-1)).view(dsize)
        d = self.dd_mapping(d)
```

```
        d = torch.transpose(d, 1,2).contiguous()                  # (64*30, 200,
        d = self.pooling(d)
        d = d.view((d.size(0), -1))

        # x = torch.cat((x, d), 2)
        # x = self.dx_mapping(x)

        # time embedding
        # t = self.value_embedding(t)
        # x = self.tv_mapping(torch.cat((x, t), 2))

        # lstm
        lstm_out, _ = self.lstm2( x )            # (64, 30, 1024)
        output = self.output_mapping(lstm_out)
        output = torch.transpose(output, 1,2).contiguous()                  # (64
        # print('ouput.size', output.size())
        output = self.pooling(output)                                       # (64
        output = output.view((output.size(0), -1))
        out = self.output(torch.cat((output, d), 1))

        # unstructure
        if content is not None:
            # print(content.size())   # [64, 1000]
            content, _ = self.lstm1(content)
            content = self.vocab_mapping(content)
            content = torch.transpose(content, 1, 2).contiguous()
            content = self.pooling(content)
            content = content.view((content.size(0), -1))
            out = self.cat_output(torch.cat((output, content, d), 1))


        return out
```

CNN

```
import os
import json
import torch
from torch import nn
import torch.nn.functional as F
from torch.autograd import *
import numpy as np
import sys
output_size = 1

def value_embedding_data(d = 200, split = 200):
    vec = np.array([np.arange(split) * i for i in range(int(d/2))], dtype=np.flo
    vec = vec / vec.max()
    embedding = np.concatenate((np.sin(vec), np.cos(vec)), 1)
    embedding[0, :d] = 0
    embedding = torch.from_numpy(embedding)
    return embedding


def conv3(in_channels, out_channels, stride=1, kernel_size=3):
    return nn.Conv1d(in_channels, out_channels, kernel_size=kernel_size,
                     stride=stride, padding=1, bias=False)

class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()
        self.conv1 = conv3(in_channels, out_channels, stride)
        self.bn1 = nn.BatchNorm1d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3(out_channels, out_channels)
        self.bn2 = nn.BatchNorm1d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
```