MIMIC ML/DL

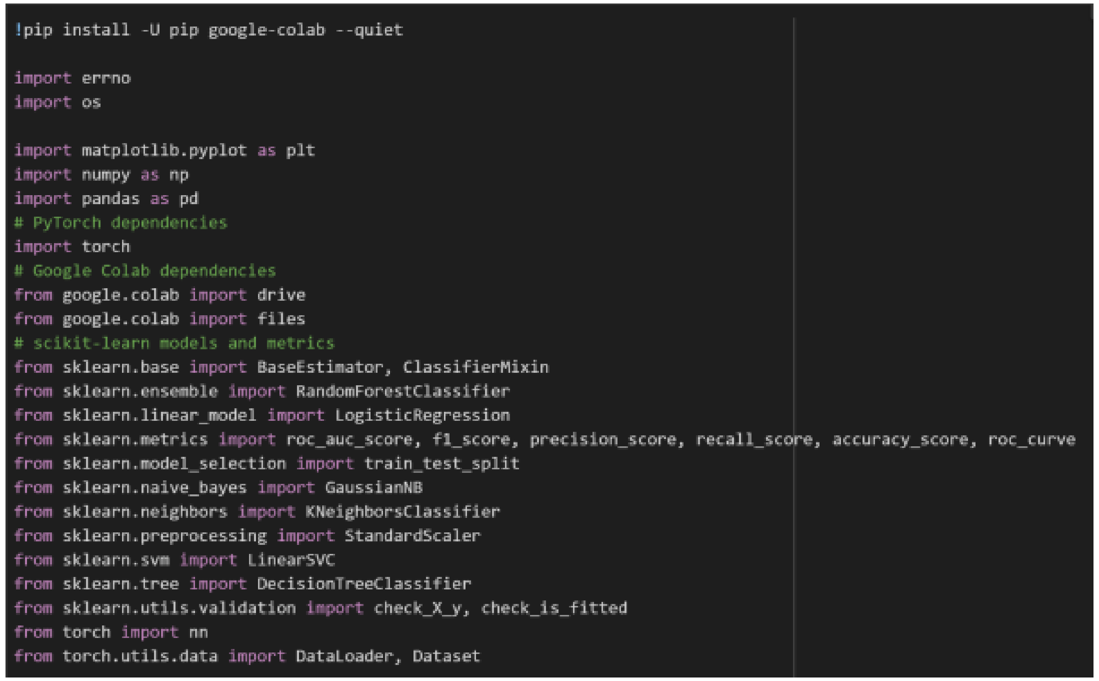
[redacted]

Learning objectives

- Make us of **MIMIC-IV** for **mortality prediction on patients diagnosed with sepsis**.
- Make use of scikit-learn built-in binary classification models for mortality prediction.
- Build, train, and evaluate a **neural network** (PyTorch) for mortality prediction.

Set up and pre-requisites

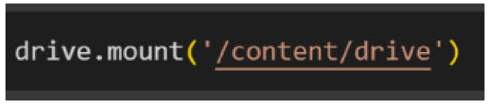
- First thing we need to do is make sure latest google-colab is installed and import required Python modules.
- Main libraries required are **PyTorch**, **scikit-learn**, **NumPy**, **pandas**, and **matplotlib**.



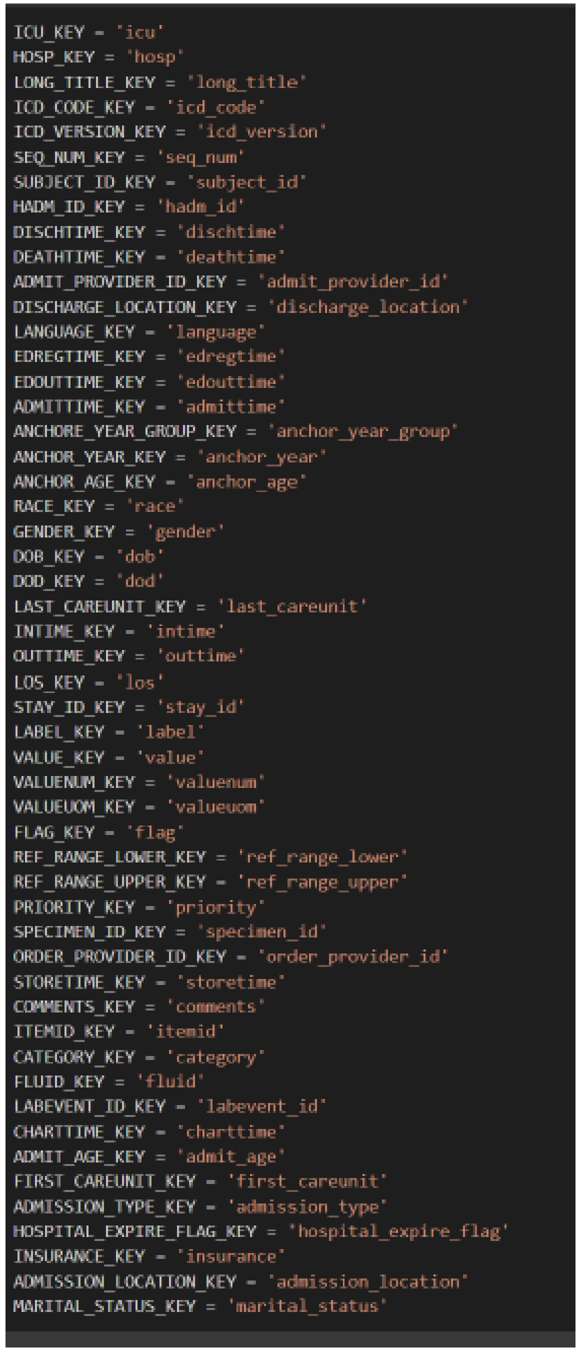
```
!pip install -U pip google-colab --quiet

import errno
import os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
# PyTorch dependencies
import torch
# Google Colab dependencies
from google.colab import drive
from google.colab import files
# scikit-learn models and metrics
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, f1_score, precision_score, recall_score, accuracy_score, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils.validation import check_X_y, check_is_fitted
from torch import nn
from torch.utils.data import DataLoader, Dataset
```

Set up and pre-requisites

- Jupyter notebook assumes MIMIC-IV dataset (https://physionet.org/content/mimiciv/2.2/) has been previously downloaded from PhysioNet, unzipped, and uploaded to Google Drive.
- Thus, we need to mount Google Colab to Google Drive.
- String constants for multiple column names found in MIMIC-IV are also defined.



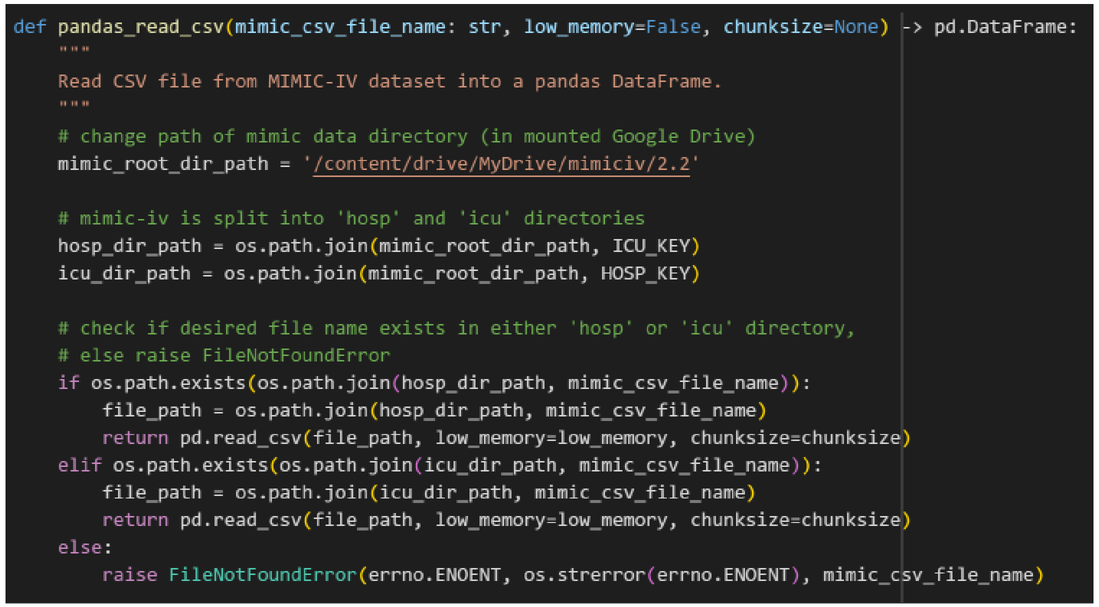
```
drive.mount('/content/drive')
```



```
ICU_KEY = 'icu'
HOSP_KEY = 'hosp'
LONG_TITLE_KEY = 'long_title'
ICD_CODE_KEY = 'icd_code'
ICD_VERSION_KEY = 'icd_version'
SEQ_NUM_KEY = 'seq_num'
SUBJECT_ID_KEY = 'subject_id'
HADM_ID_KEY = 'hadm_id'
DISCHTIME_KEY = 'dischtime'
DEATHTIME_KEY = 'deathtime'
ADMIT_PROVIDER_ID_KEY = 'admit_provider_id'
DISCHARGE_LOCATION_KEY = 'discharge_location'
LANGUAGE_KEY = 'language'
EDREGTIME_KEY = 'edregtime'
EDOUTTIME_KEY = 'edouttime'
ADMITTIME_KEY = 'admittime'
ANCHORE_YEAR_GROUP_KEY = 'anchor_year_group'
ANCHOR_YEAR_KEY = 'anchor_year'
ANCHOR_AGE_KEY = 'anchor_age'
RACE_KEY = 'race'
GENDER_KEY = 'gender'
DOB_KEY = 'dob'
DOD_KEY = 'dod'
LAST_CAREUNIT_KEY = 'last_careunit'
INTIME_KEY = 'intime'
OUTTIME_KEY = 'outtime'
LOS_KEY = 'los'
STAY_ID_KEY = 'stay_id'
LABEL_KEY = 'label'
VALUE_KEY = 'value'
VALUENUM_KEY = 'valuenum'
VALUEUOM_KEY = 'valueuom'
FLAG_KEY = 'flag'
REF_RANGE_LOWER_KEY = 'ref_range_lower'
REF_RANGE_UPPER_KEY = 'ref_range_upper'
PRIORITY_KEY = 'priority'
SPECIMEN_ID_KEY = 'specimen_id'
ORDER_PROVIDER_ID_KEY = 'order_provider_id'
STORETIME_KEY = 'storetime'
COMMENTS_KEY = 'comments'
ITEMID_KEY = 'itemid'
CATEGORY_KEY = 'category'
FLUID_KEY = 'fluid'
LABEVENT_ID_KEY = 'labevent_id'
CHARTTIME_KEY = 'charttime'
ADMIT_AGE_KEY = 'admit_age'
FIRST_CAREUNIT_KEY = 'first_careunit'
ADMISSION_TYPE_KEY = 'admission_type'
HOSPITAL_EXPIRE_FLAG_KEY = 'hospital_expire_flag'
INSURANCE_KEY = 'insurance'
ADMISSION_LOCATION_KEY = 'admission_location'
MARITAL_STATUS_KEY = 'marital_status'
```

Data processing and features

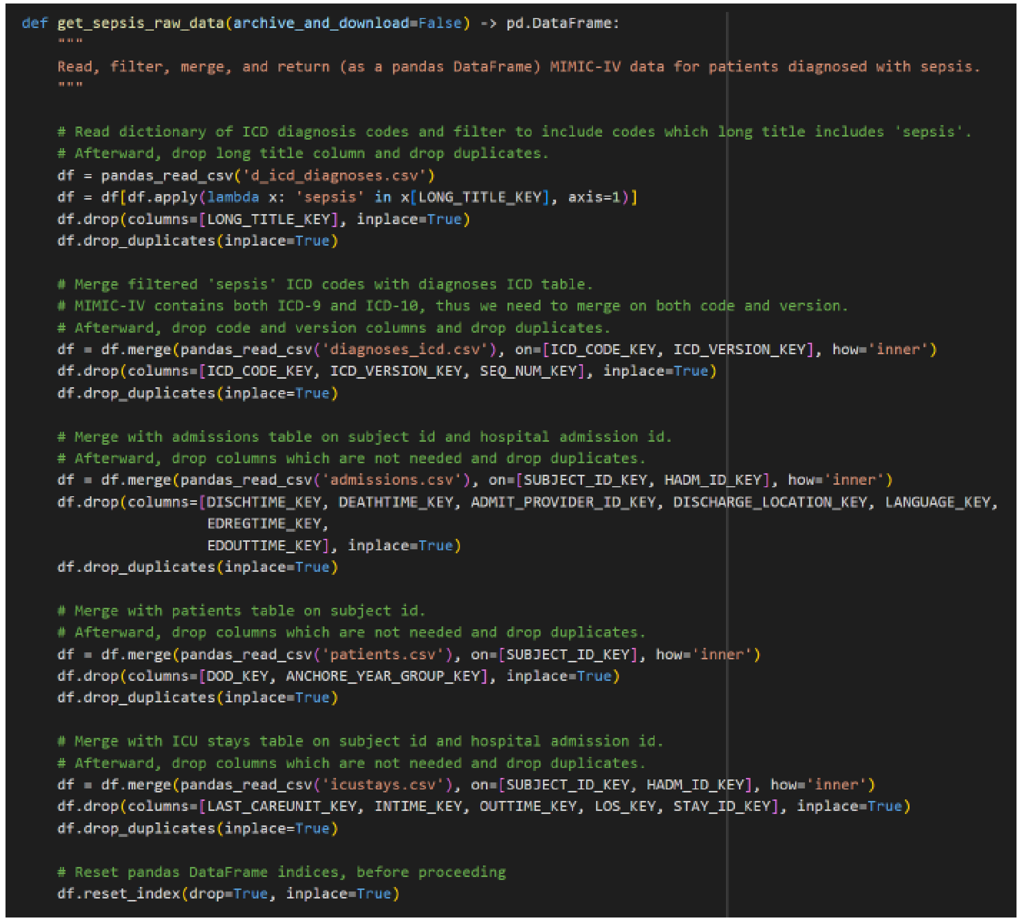
- Helper method shown on the right is used to read MIMIC-IV csv data files into a pandas data frame.
- MIMIC-IV files are split into **hosp** and **icu** directories. Thus, we need to check if target csv file exists in either directory.



```python
def pandas_read_csv(mimic_csv_file_name: str, low_memory=False, chunksize=None) -> pd.DataFrame:
    """
    Read CSV file from MIMIC-IV dataset into a pandas DataFrame.
    """
    # change path of mimic data directory (in mounted Google Drive)
    mimic_root_dir_path = '/content/drive/MyDrive/mimiciv/2.2'

    # mimic-iv is split into 'hosp' and 'icu' directories
    hosp_dir_path = os.path.join(mimic_root_dir_path, ICU_KEY)
    icu_dir_path = os.path.join(mimic_root_dir_path, HOSP_KEY)

    # check if desired file name exists in either 'hosp' or 'icu' directory,
    # else raise FileNotFoundError
    if os.path.exists(os.path.join(hosp_dir_path, mimic_csv_file_name)):
        file_path = os.path.join(hosp_dir_path, mimic_csv_file_name)
        return pd.read_csv(file_path, low_memory=low_memory, chunksize=chunksize)
    elif os.path.exists(os.path.join(icu_dir_path, mimic_csv_file_name)):
        file_path = os.path.join(icu_dir_path, mimic_csv_file_name)
        return pd.read_csv(file_path, low_memory=low_memory, chunksize=chunksize)
    else:
        raise FileNotFoundError(errno.ENOENT, os.strerror(errno.ENOENT), mimic_csv_file_name)
```

Data processing and features

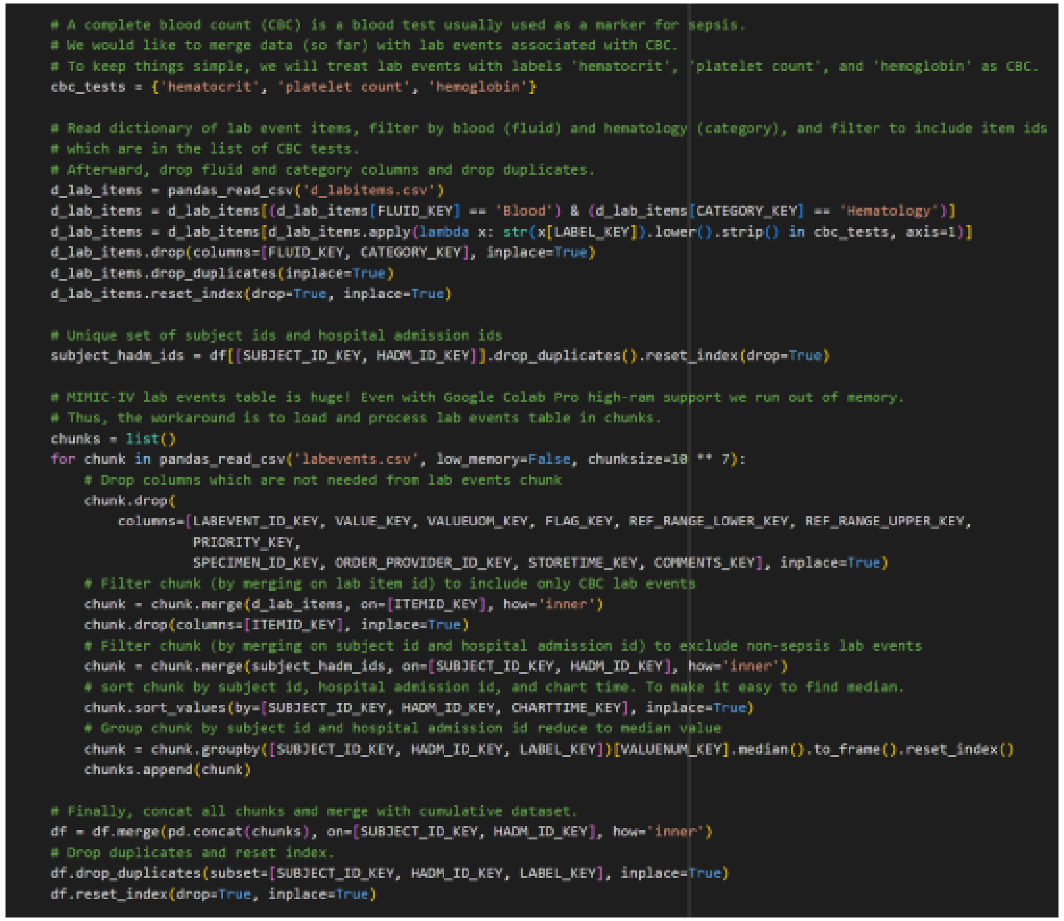
- Helper method (first half) shown on the right is used to read, filter, merge, and return raw MIMIC-IV data (as a pandas data frame) for hospital admissions diagnosed with **sepsis**.
- MIMIC-IV tables merged include dictionary of ICD-9 and ICD-10 codes, diagnoses table, admissions table, patients table, and ICU stays table.



```python
def get_sepsis_raw_data(archive_and_download=False) -> pd.DataFrame:
    """
    Read, filter, merge, and return (as a pandas DataFrame) MIMIC-IV data for patients diagnosed with sepsis.
    """

    # Read dictionary of ICD diagnosis codes and filter to include codes which long title includes 'sepsis'.
    # Afterward, drop long title column and drop duplicates.
    df = pandas_read_csv('d_icd_diagnoses.csv')
    df = df[df.apply(lambda x: 'sepsis' in x[LONG_TITLE_KEY], axis=1)]
    df.drop(columns=[LONG_TITLE_KEY], inplace=True)
    df.drop_duplicates(inplace=True)

    # Merge filtered 'sepsis' ICD codes with diagnoses ICD table.
    # MIMIC-IV contains both ICD-9 and ICD-10, thus we need to merge on both code and version.
    # Afterward, drop code and version columns and drop duplicates.
    df = df.merge(pandas_read_csv('diagnoses_icd.csv'), on=[ICD_CODE_KEY, ICD_VERSION_KEY], how='inner')
    df.drop(columns=[ICD_CODE_KEY, ICD_VERSION_KEY, SEQ_NUM_KEY], inplace=True)
    df.drop_duplicates(inplace=True)

    # Merge with admissions table on subject id and hospital admission id.
    # Afterward, drop columns which are not needed and drop duplicates.
    df = df.merge(pandas_read_csv('admissions.csv'), on=[SUBJECT_ID_KEY, HADM_ID_KEY], how='inner')
    df.drop(columns=[DISCHTIME_KEY, DEATHTIME_KEY, ADMIT_PROVIDER_ID_KEY, DISCHARGE_LOCATION_KEY, LANGUAGE_KEY,
                     EDREGTIME_KEY,
                     EDOUTTIME_KEY], inplace=True)
    df.drop_duplicates(inplace=True)

    # Merge with patients table on subject id.
    # Afterward, drop columns which are not needed and drop duplicates.
    df = df.merge(pandas_read_csv('patients.csv'), on=[SUBJECT_ID_KEY], how='inner')
    df.drop(columns=[DOD_KEY, ANCHORE_YEAR_GROUP_KEY], inplace=True)
    df.drop_duplicates(inplace=True)

    # Merge with ICU stays table on subject id and hospital admission id.
    # Afterward, drop columns which are not needed and drop duplicates.
    df = df.merge(pandas_read_csv('icustays.csv'), on=[SUBJECT_ID_KEY, HADM_ID_KEY], how='inner')
    df.drop(columns=[LAST_CAREUNIT_KEY, INTIME_KEY, OUTTIME_KEY, LOS_KEY, STAY_ID_KEY], inplace=True)
    df.drop_duplicates(inplace=True)

    # Reset pandas DataFrame indices, before proceeding
    df.reset_index(drop=True, inplace=True)
```

Data processing and features

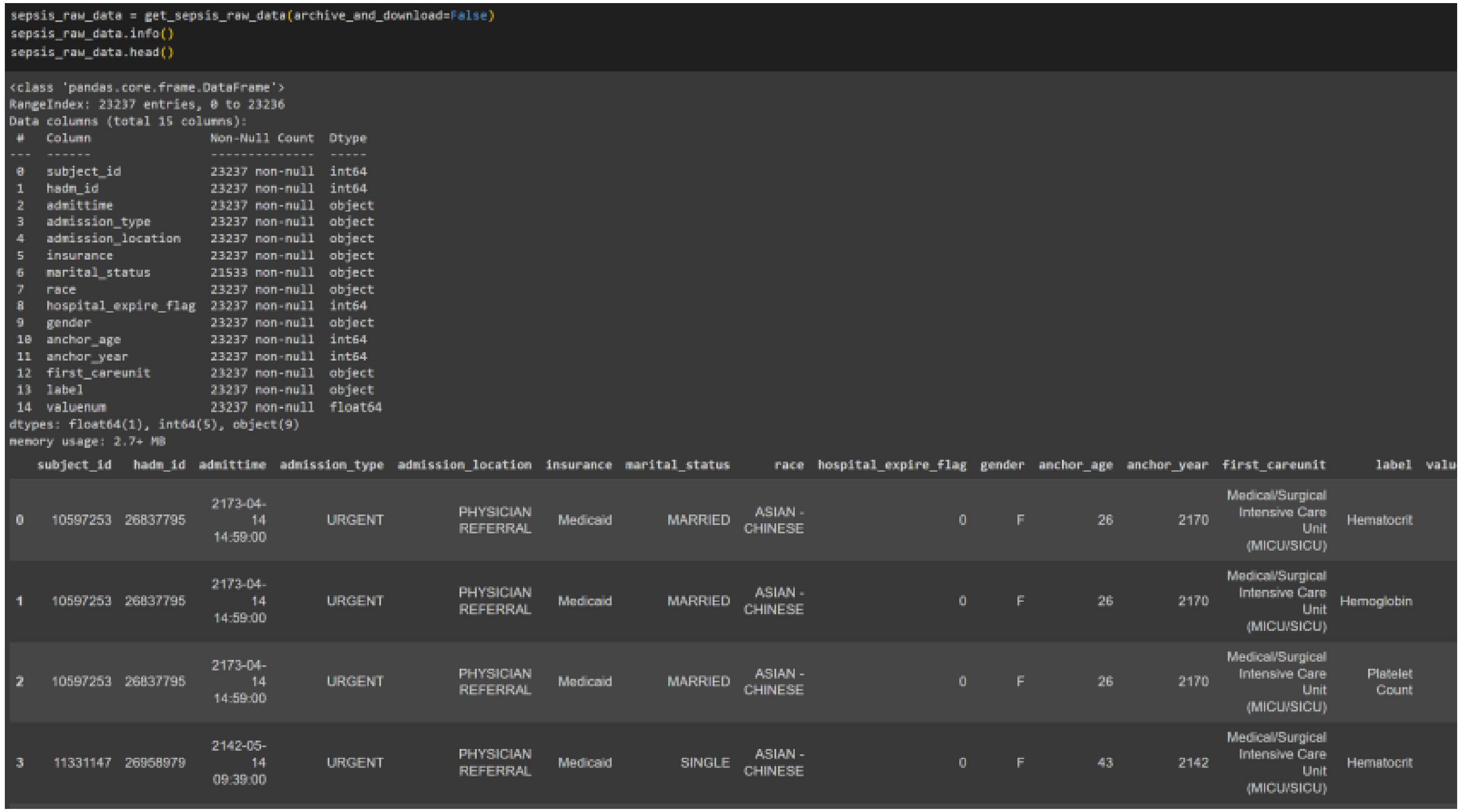
- Second half of helper method to get raw sepsis data is shown on the right.
- Usually complete blood count (CBC) tests are used to track the progression of sepsis. We would like to include these in our data as these could be helpful features in mortality prediction. To be more specific, results for **hematocrit**, **platelet count**, and **hemoglobin**.
- One issue is MIMIC-IV lab events table is large. Even with Google Colab Pro high-ram, trying to load all lab events data we run out of memory. The workaround is to load and filter lab events data in **chunks**.



```
# A complete blood count (CBC) is a blood test usually used as a marker for sepsis.
# We would like to merge data (so far) with lab events associated with CBC.
# To keep things simple, we will treat lab events with labels 'hematocrit', 'platelet count', and 'hemoglobin' as CBC.
cbc_tests = {'hematocrit', 'platelet count', 'hemoglobin'}

# Read dictionary of lab event items, filter by blood (fluid) and hematology (category), and filter to include item ids
# which are in the list of CBC tests.
# Afterward, drop fluid and category columns and drop duplicates.
d_lab_items = pandas_read_csv('d_labitems.csv')
d_lab_items = d_lab_items[(d_lab_items[FLUID_KEY] == 'Blood') & (d_lab_items[CATEGORY_KEY] == 'Hematology')]
d_lab_items = d_lab_items[d_lab_items.apply(lambda x: str(x[LABEL_KEY]).lower().strip() in cbc_tests, axis=1)]
d_lab_items.drop(columns=[FLUID_KEY, CATEGORY_KEY], inplace=True)
d_lab_items.drop_duplicates(inplace=True)
d_lab_items.reset_index(drop=True, inplace=True)

# Unique set of subject ids and hospital admission ids
subject_hadm_ids = df[[SUBJECT_ID_KEY, HADM_ID_KEY]].drop_duplicates().reset_index(drop=True)

# MIMIC-IV lab events table is huge! Even with Google Colab Pro high-ram support we run out of memory.
# Thus, the workaround is to load and process lab events table in chunks.
chunks = list()
for chunk in pandas_read_csv('labevents.csv', low_memory=False, chunksize=10 ** 7):
    # Drop columns which are not needed from lab events chunk
    chunk.drop(
        columns=[LABEVENT_ID_KEY, VALUE_KEY, VALUEUOM_KEY, FLAG_KEY, REF_RANGE_LOWER_KEY, REF_RANGE_UPPER_KEY,
                 PRIORITY_KEY,
                 SPECIMEN_ID_KEY, ORDER_PROVIDER_ID_KEY, STORETIME_KEY, COMMENTS_KEY], inplace=True)
    # Filter chunk (by merging on lab item id) to include only CBC lab events
    chunk = chunk.merge(d_lab_items, on=[ITEMID_KEY], how='inner')
    chunk.drop(columns=[ITEMID_KEY], inplace=True)
    # Filter chunk (by merging on subject id and hospital admission id) to exclude non-sepsis lab events
    chunk = chunk.merge(subject_hadm_ids, on=[SUBJECT_ID_KEY, HADM_ID_KEY], how='inner')
    # sort chunk by subject id, hospital admission id, and chart time. To make it easy to find median.
    chunk.sort_values(by=[SUBJECT_ID_KEY, HADM_ID_KEY, CHARTTIME_KEY], inplace=True)
    # Group chunk by subject id and hospital admission id reduce to median value
    chunk = chunk.groupby([SUBJECT_ID_KEY, HADM_ID_KEY, LABEL_KEY])[VALUENUM_KEY].median().to_frame().reset_index()
    chunks.append(chunk)

# Finally, concat all chunks and merge with cumulative dataset.
df = df.merge(pd.concat(chunks), on=[SUBJECT_ID_KEY, HADM_ID_KEY], how='inner')
# Drop duplicates and reset index.
df.drop_duplicates(subset=[SUBJECT_ID_KEY, HADM_ID_KEY, LABEL_KEY], inplace=True)
df.reset_index(drop=True, inplace=True)
```

Data processing and features

- Below shows execution of previous helper method to load MIMIC-IV raw sepsis data
- This will take several minutes due to the fact lab events table is very large



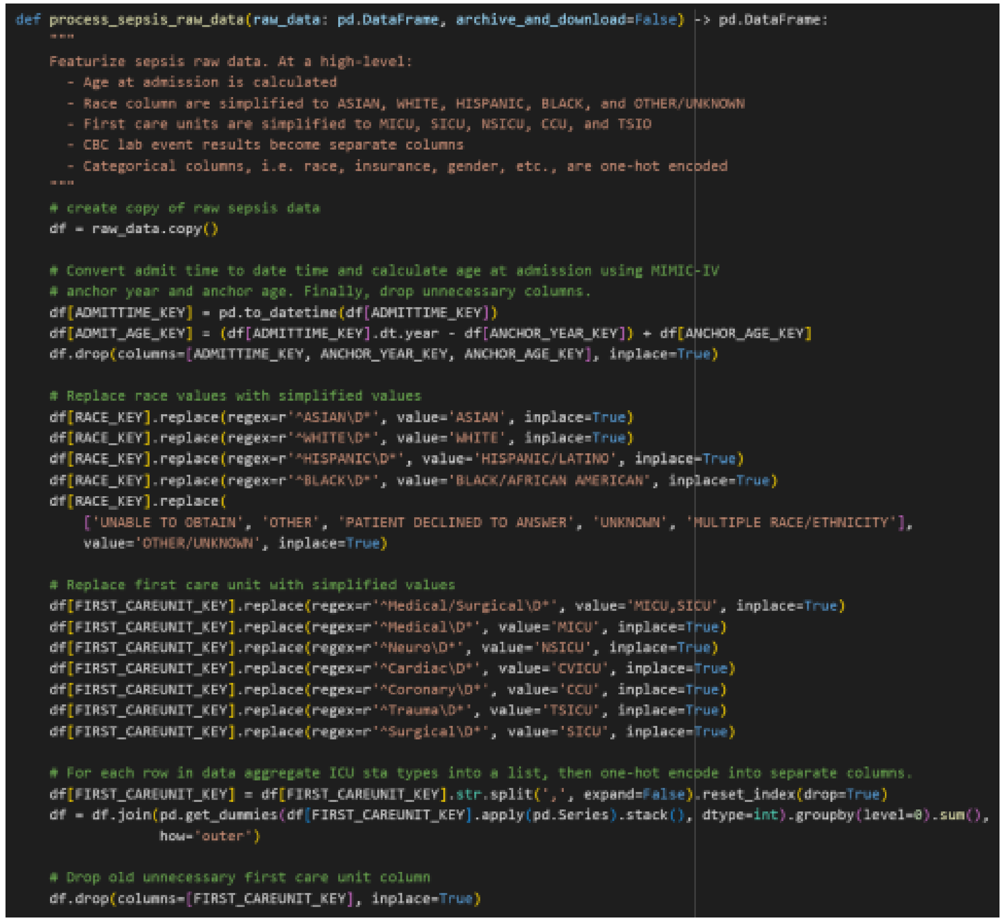
```python
sepsis_raw_data = get_sepsis_raw_data(archive_and_download=False)
sepsis_raw_data.info()
sepsis_raw_data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23237 entries, 0 to 23236
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   subject_id           23237 non-null  int64  
 1   hadm_id              23237 non-null  int64  
 2   admittime            23237 non-null  object 
 3   admission_type       23237 non-null  object 
 4   admission_location   23237 non-null  object 
 5   insurance            23237 non-null  object 
 6   marital_status       21533 non-null  object 
 7   race                 23237 non-null  object 
 8   hospital_expire_flag 23237 non-null  int64  
 9   gender               23237 non-null  object 
 10  anchor_age           23237 non-null  int64  
 11  anchor_year          23237 non-null  int64  
 12  first_careunit       23237 non-null  object 
 13  label                23237 non-null  object 
 14  valuenum             23237 non-null  float64
dtypes: float64(1), int64(5), object(9)
memory usage: 2.7+ MB
```

	subject_id	hadm_id	admittime	admission_type	admission_location	insurance	marital_status	race	hospital_expire_flag	gender	anchor_age	anchor_year	first_careunit	label	valu
0	10597253	26837795	2173-04-14 14:59:00	URGENT	PHYSICIAN REFERRAL	Medicaid	MARRIED	ASIAN - CHINESE	0	F	26	2170	Medical/Surgical Intensive Care Unit (MICU/SICU)	Hematocrit	
1	10597253	26837795	2173-04-14 14:59:00	URGENT	PHYSICIAN REFERRAL	Medicaid	MARRIED	ASIAN - CHINESE	0	F	26	2170	Medical/Surgical Intensive Care Unit (MICU/SICU)	Hemoglobin	
2	10597253	26837795	2173-04-14 14:59:00	URGENT	PHYSICIAN REFERRAL	Medicaid	MARRIED	ASIAN - CHINESE	0	F	26	2170	Medical/Surgical Intensive Care Unit (MICU/SICU)	Platelet Count	
3	11331147	26958979	2142-05-14 09:39:00	URGENT	PHYSICIAN REFERRAL	Medicaid	SINGLE	ASIAN - CHINESE	0	F	43	2142	Medical/Surgical Intensive Care Unit (MICU/SICU)	Hematocrit	

Data processing and features

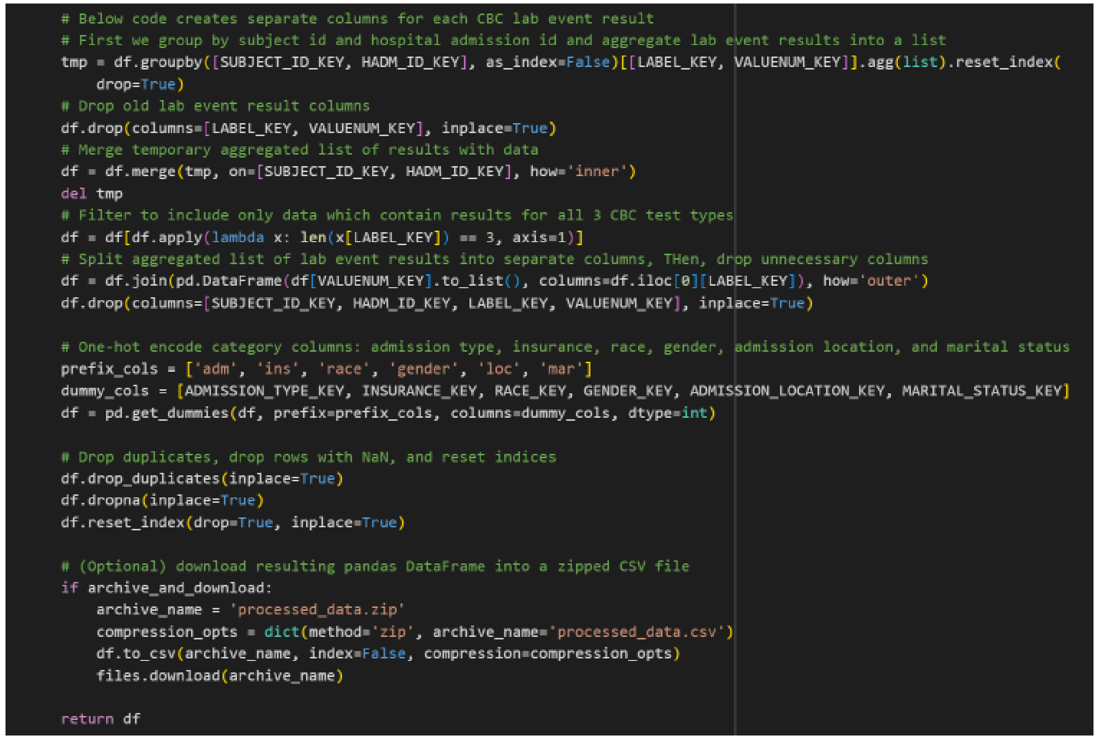
- Helper method (first half) shown on the right is used to processed previously fetched raw sepsis data and create features out of it.
- At a high level:
  - Age at admission is calculated
  - Race and first care unit column values are simplified
  - CBC lab event results become separate columns
  - Categorical columns become one-hot encoded



```python
def process_sepsis_raw_data(raw_data: pd.DataFrame, archive_and_download=False) -> pd.DataFrame:
    """
    Featurize sepsis raw data. At a high-level:
      - Age at admission is calculated
      - Race column are simplified to ASIAN, WHITE, HISPANIC, BLACK, and OTHER/UNKNOWN
      - First care units are simplified to MICU, SICU, NSICU, CCU, and TSIO
      - CBC lab event results become separate columns
      - Categorical columns, i.e. race, insurance, gender, etc., are one-hot encoded
    """
    # create copy of raw sepsis data
    df = raw_data.copy()

    # Convert admit time to date time and calculate age at admission using MIMIC-IV
    # anchor year and anchor age. Finally, drop unnecessary columns.
    df[ADMITTIME_KEY] = pd.to_datetime(df[ADMITTIME_KEY])
    df[ADMIT_AGE_KEY] = (df[ADMITTIME_KEY].dt.year - df[ANCHOR_YEAR_KEY]) + df[ANCHOR_AGE_KEY]
    df.drop(columns=[ADMITTIME_KEY, ANCHOR_YEAR_KEY, ANCHOR_AGE_KEY], inplace=True)

    # Replace race values with simplified values
    df[RACE_KEY].replace(regex=r'^ASIAN\D*', value='ASIAN', inplace=True)
    df[RACE_KEY].replace(regex=r'^WHITE\D*', value='WHITE', inplace=True)
    df[RACE_KEY].replace(regex=r'^HISPANIC\D*', value='HISPANIC/LATINO', inplace=True)
    df[RACE_KEY].replace(regex=r'^BLACK\D*', value='BLACK/AFRICAN AMERICAN', inplace=True)
    df[RACE_KEY].replace(
        ['UNABLE TO OBTAIN', 'OTHER', 'PATIENT DECLINED TO ANSWER', 'UNKNOWN', 'MULTIPLE RACE/ETHNICITY'],
        value='OTHER/UNKNOWN', inplace=True)

    # Replace first care unit with simplified values
    df[FIRST_CAREUNIT_KEY].replace(regex=r'^Medical/Surgical\D*', value='MICU,SICU', inplace=True)
    df[FIRST_CAREUNIT_KEY].replace(regex=r'^Medical\D*', value='MICU', inplace=True)
    df[FIRST_CAREUNIT_KEY].replace(regex=r'^Neuro\D*', value='NSICU', inplace=True)
    df[FIRST_CAREUNIT_KEY].replace(regex=r'^Cardiac\D*', value='CVICU', inplace=True)
    df[FIRST_CAREUNIT_KEY].replace(regex=r'^Coronary\D*', value='CCU', inplace=True)
    df[FIRST_CAREUNIT_KEY].replace(regex=r'^Trauma\D*', value='TSICU', inplace=True)
    df[FIRST_CAREUNIT_KEY].replace(regex=r'^Surgical\D*', value='SICU', inplace=True)

    # For each row in data aggregate ICU sta types into a list, then one-hot encode into separate columns.
    df[FIRST_CAREUNIT_KEY] = df[FIRST_CAREUNIT_KEY].str.split(',', expand=False).reset_index(drop=True)
    df = df.join(pd.get_dummies(df[FIRST_CAREUNIT_KEY].apply(pd.Series).stack(), dtype=int).groupby(level=0).sum(),
                 how='outer')

    # Drop old unnecessary first care unit column
    df.drop(columns=[FIRST_CAREUNIT_KEY], inplace=True)
```

Data processing and features

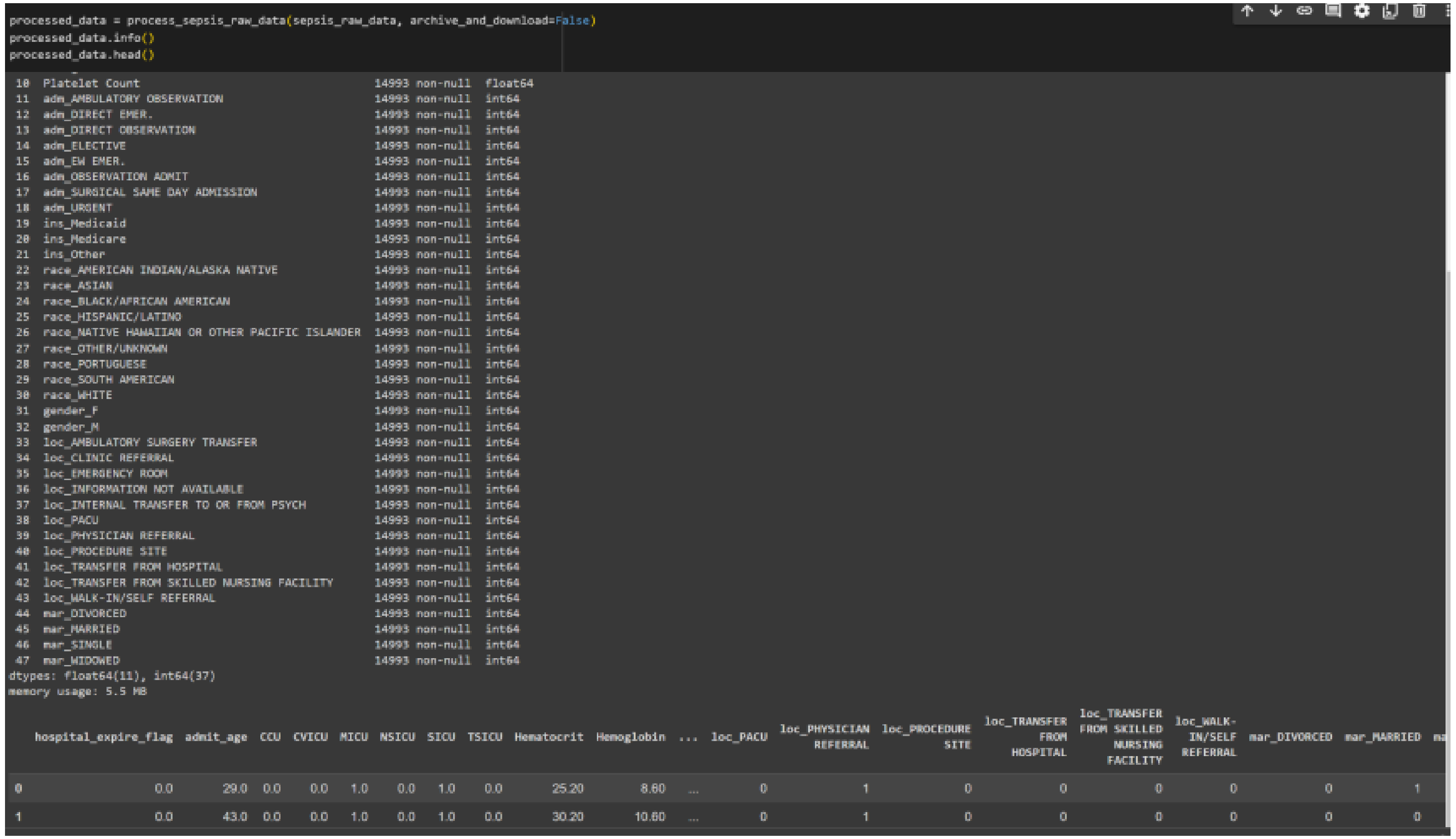
- Second half of helper method to process raw sepsis data and create features is shown on the right.
- Only hospital admission rows with lab results for all CBC tests are included.
- Furthermore, duplicate rows and rows with missing values are dropped.



```python
    # Below code creates separate columns for each CBC lab event result
    # First we group by subject id and hospital admission id and aggregate lab event results into a list
    tmp = df.groupby([SUBJECT_ID_KEY, HADM_ID_KEY], as_index=False)[[LABEL_KEY, VALUENUM_KEY]].agg(list).reset_index(
        drop=True)
    # Drop old lab event result columns
    df.drop(columns=[LABEL_KEY, VALUENUM_KEY], inplace=True)
    # Merge temporary aggregated list of results with data
    df = df.merge(tmp, on=[SUBJECT_ID_KEY, HADM_ID_KEY], how='inner')
    del tmp
    # Filter to include only data which contain results for all 3 CBC test types
    df = df[df.apply(lambda x: len(x[LABEL_KEY]) == 3, axis=1)]
    # Split aggregated list of lab event results into separate columns, THen, drop unnecessary columns
    df = df.join(pd.DataFrame(df[VALUENUM_KEY].to_list(), columns=df.iloc[0][LABEL_KEY]), how='outer')
    df.drop(columns=[SUBJECT_ID_KEY, HADM_ID_KEY, LABEL_KEY, VALUENUM_KEY], inplace=True)

    # One-hot encode category columns: admission type, insurance, race, gender, admission location, and marital status
    prefix_cols = ['adm', 'ins', 'race', 'gender', 'loc', 'mar']
    dummy_cols = [ADMISSION_TYPE_KEY, INSURANCE_KEY, RACE_KEY, GENDER_KEY, ADMISSION_LOCATION_KEY, MARITAL_STATUS_KEY]
    df = pd.get_dummies(df, prefix=prefix_cols, columns=dummy_cols, dtype=int)

    # Drop duplicates, drop rows with NaN, and reset indices
    df.drop_duplicates(inplace=True)
    df.dropna(inplace=True)
    df.reset_index(drop=True, inplace=True)

    # (Optional) download resulting pandas DataFrame into a zipped CSV file
    if archive_and_download:
        archive_name = 'processed_data.zip'
        compression_opts = dict(method='zip', archive_name='processed_data.csv')
        df.to_csv(archive_name, index=False, compression=compression_opts)
        files.download(archive_name)

    return df
```

Data processing and features

- Below shows execution of previous helper method to create features from raw sepsis data



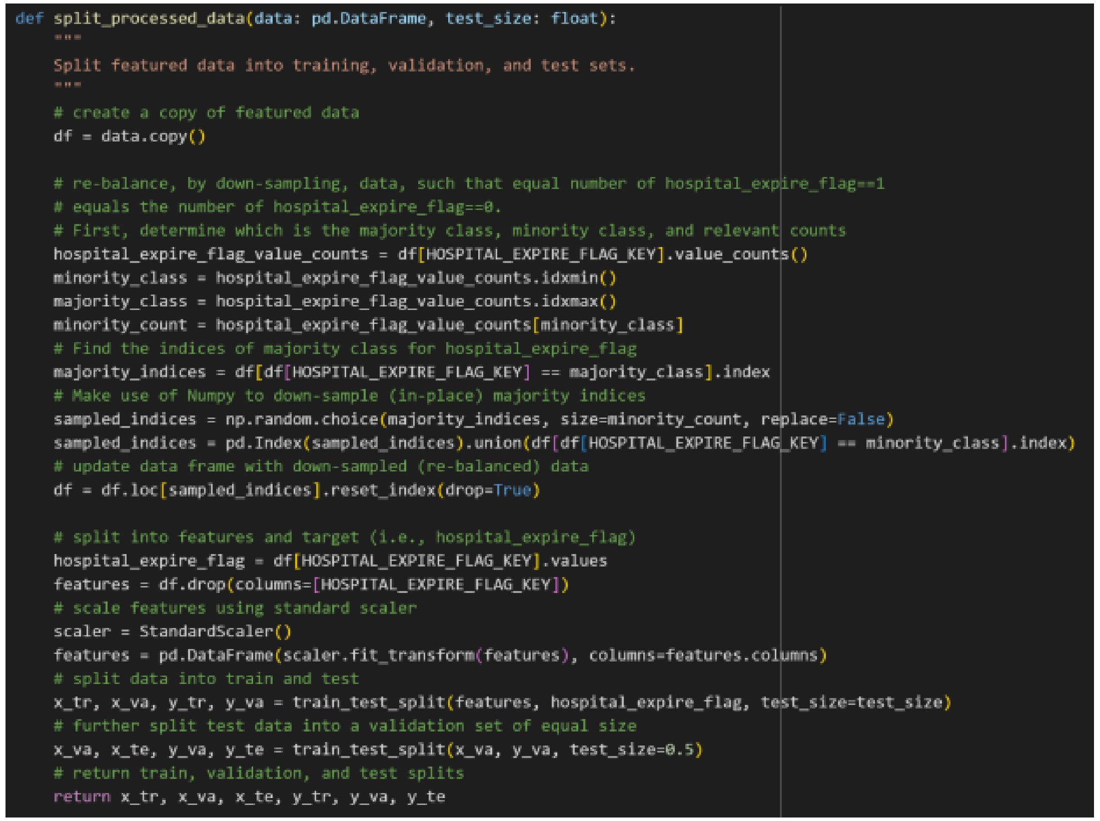
```
processed_data = process_sepsis_raw_data(sepsis_raw_data, archive_and_download=False)
processed_data.info()
processed_data.head()
```

```
 10  Platelet Count                                    14993 non-null  float64
 11  adm_AMBULATORY OBSERVATION                        14993 non-null  int64
 12  adm_DIRECT EMER.                                  14993 non-null  int64
 13  adm_DIRECT OBSERVATION                            14993 non-null  int64
 14  adm_ELECTIVE                                      14993 non-null  int64
 15  adm_EW EMER.                                      14993 non-null  int64
 16  adm_OBSERVATION ADMIT                             14993 non-null  int64
 17  adm_SURGICAL SAME DAY ADMISSION                   14993 non-null  int64
 18  adm_URGENT                                        14993 non-null  int64
 19  ins_Medicaid                                      14993 non-null  int64
 20  ins_Medicare                                      14993 non-null  int64
 21  ins_Other                                         14993 non-null  int64
 22  race_AMERICAN INDIAN/ALASKA NATIVE                14993 non-null  int64
 23  race_ASIAN                                        14993 non-null  int64
 24  race_BLACK/AFRICAN AMERICAN                       14993 non-null  int64
 25  race_HISPANIC/LATINO                              14993 non-null  int64
 26  race_NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER    14993 non-null  int64
 27  race_OTHER/UNKNOWN                                14993 non-null  int64
 28  race_PORTUGUESE                                   14993 non-null  int64
 29  race_SOUTH AMERICAN                               14993 non-null  int64
 30  race_WHITE                                        14993 non-null  int64
 31  gender_F                                          14993 non-null  int64
 32  gender_M                                          14993 non-null  int64
 33  loc_AMBULATORY SURGERY TRANSFER                   14993 non-null  int64
 34  loc_CLINIC REFERRAL                               14993 non-null  int64
 35  loc_EMERGENCY ROOM                                14993 non-null  int64
 36  loc_INFORMATION NOT AVAILABLE                     14993 non-null  int64
 37  loc_INTERNAL TRANSFER TO OR FROM PSYCH            14993 non-null  int64
 38  loc_PACU                                          14993 non-null  int64
 39  loc_PHYSICIAN REFERRAL                            14993 non-null  int64
 40  loc_PROCEDURE SITE                                14993 non-null  int64
 41  loc_TRANSFER FROM HOSPITAL                        14993 non-null  int64
 42  loc_TRANSFER FROM SKILLED NURSING FACILITY        14993 non-null  int64
 43  loc_WALK-IN/SELF REFERRAL                         14993 non-null  int64
 44  mar_DIVORCED                                      14993 non-null  int64
 45  mar_MARRIED                                       14993 non-null  int64
 46  mar_SINGLE                                        14993 non-null  int64
 47  mar_WIDOWED                                       14993 non-null  int64
dtypes: float64(11), int64(37)
memory usage: 5.5 MB
```

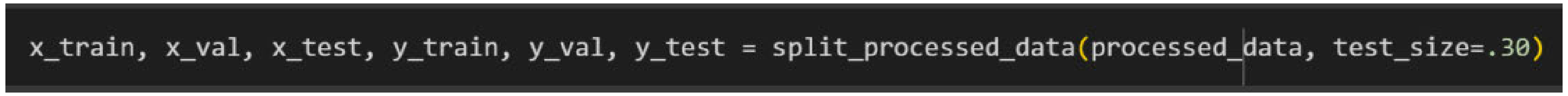
	hospital_expire_flag	admit_age	CCU	CVICU	MICU	NSICU	SICU	TSICU	Hematocrit	Hemoglobin	...	loc_PACU	loc_PHYSICIAN REFERRAL	loc_PROCEDURE SITE	loc_TRANSFER FROM HOSPITAL	loc_TRANSFER FROM SKILLED NURSING FACILITY	loc_WALK-IN/SELF REFERRAL	mar_DIVORCED	mar_MARRIED
0	0.0	29.0	0.0	0.0	1.0	0.0	1.0	0.0	25.20	8.60	...	0	1	0	0	0	0	0	1
1	0.0	43.0	0.0	0.0	1.0	0.0	1.0	0.0	30.20	10.60	...	0	1	0	0	0	0	0	0

Train, validation, and test split

- Helper method on the right is used to rebalance and split data into train, validation, and test splits.
- Rebalancing is done by randomly down-sampling data such that target values, i.e., **hospital_expire_flag**, are equally distributed for binary classification.
- The split is 70% train, 15% validation, and 15% test.



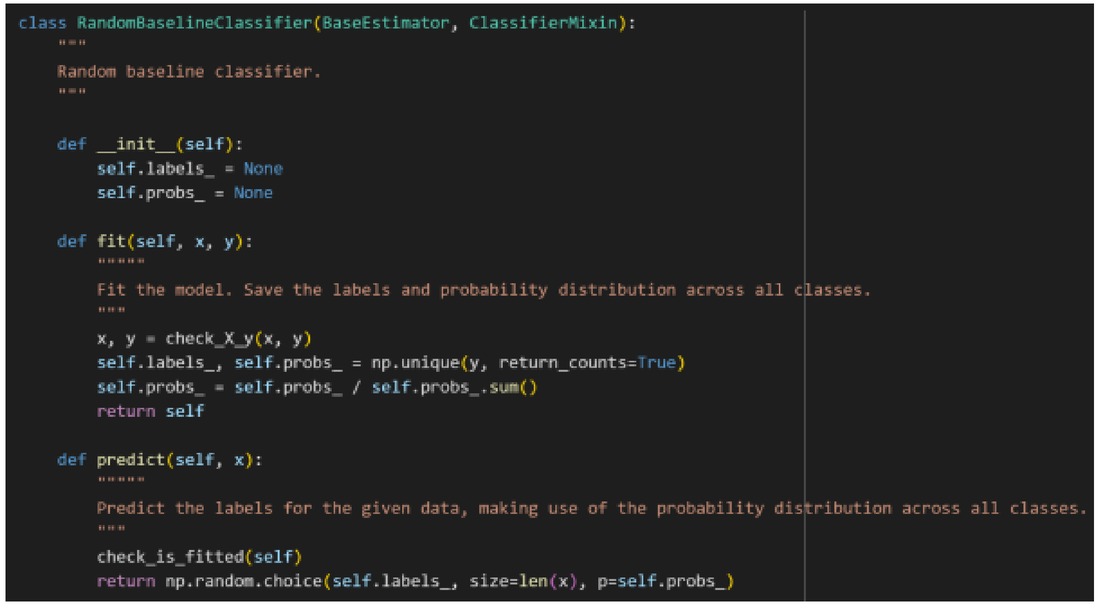
```python
def split_processed_data(data: pd.DataFrame, test_size: float):
    """
    Split featured data into training, validation, and test sets.
    """
    # create a copy of featured data
    df = data.copy()

    # re-balance, by down-sampling, data, such that equal number of hospital_expire_flag==1
    # equals the number of hospital_expire_flag==0.
    # First, determine which is the majority class, minority class, and relevant counts
    hospital_expire_flag_value_counts = df[HOSPITAL_EXPIRE_FLAG_KEY].value_counts()
    minority_class = hospital_expire_flag_value_counts.idxmin()
    majority_class = hospital_expire_flag_value_counts.idxmax()
    minority_count = hospital_expire_flag_value_counts[minority_class]
    # Find the indices of majority class for hospital_expire_flag
    majority_indices = df[df[HOSPITAL_EXPIRE_FLAG_KEY] == majority_class].index
    # Make use of Numpy to down-sample (in-place) majority indices
    sampled_indices = np.random.choice(majority_indices, size=minority_count, replace=False)
    sampled_indices = pd.Index(sampled_indices).union(df[df[HOSPITAL_EXPIRE_FLAG_KEY] == minority_class].index)
    # update data frame with down-sampled (re-balanced) data
    df = df.loc[sampled_indices].reset_index(drop=True)

    # split into features and target (i.e., hospital_expire_flag)
    hospital_expire_flag = df[HOSPITAL_EXPIRE_FLAG_KEY].values
    features = df.drop(columns=[HOSPITAL_EXPIRE_FLAG_KEY])
    # scale features using standard scaler
    scaler = StandardScaler()
    features = pd.DataFrame(scaler.fit_transform(features), columns=features.columns)
    # split data into train and test
    x_tr, x_va, y_tr, y_va = train_test_split(features, hospital_expire_flag, test_size=test_size)
    # further split test data into a validation set of equal size
    x_va, x_te, y_va, y_te = train_test_split(x_va, y_va, test_size=0.5)
    # return train, validation, and test splits
    return x_tr, x_va, x_te, y_tr, y_va, y_te
```



```python
x_train, x_val, x_test, y_train, y_val, y_test = split_processed_data(processed_data, test_size=.30)
```

Scikit-learn and random baseline binary classifiers

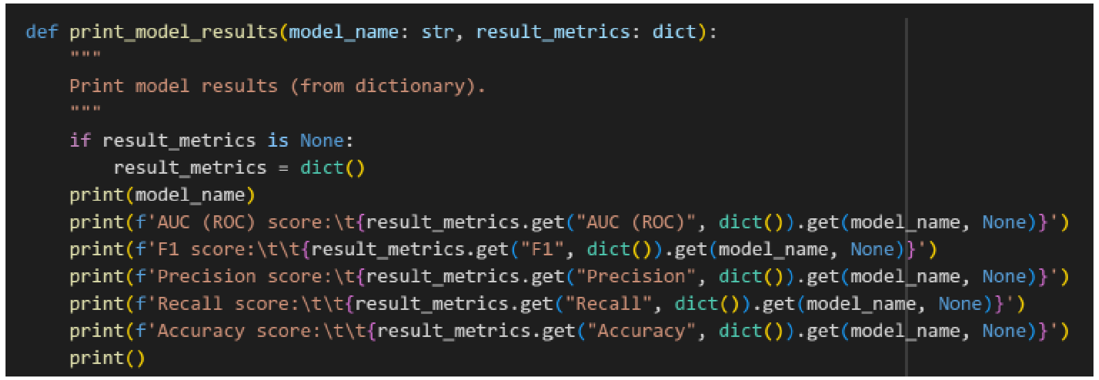
- We now define a random binary classifier (shown on right) as a baseline.
- **The end goal is to eventually train a neural network that performs better than random guessing.**
- This random binary classifier learns the probability distribution of classes from the training data. And during inference, it randomly samples from such learned distribution.



```python
class RandomBaselineClassifier(BaseEstimator, ClassifierMixin):
    """
    Random baseline classifier.
    """

    def __init__(self):
        self.labels_ = None
        self.probs_ = None

    def fit(self, x, y):
        """
        Fit the model. Save the labels and probability distribution across all classes.
        """
        x, y = check_X_y(x, y)
        self.labels_, self.probs_ = np.unique(y, return_counts=True)
        self.probs_ = self.probs_ / self.probs_.sum()
        return self

    def predict(self, x):
        """
        Predict the labels for the given data, making use of the probability distribution across all classes.
        """
        check_is_fitted(self)
        return np.random.choice(self.labels_, size=len(x), p=self.probs_)
```

Scikit-learn and random baseline binary classifiers

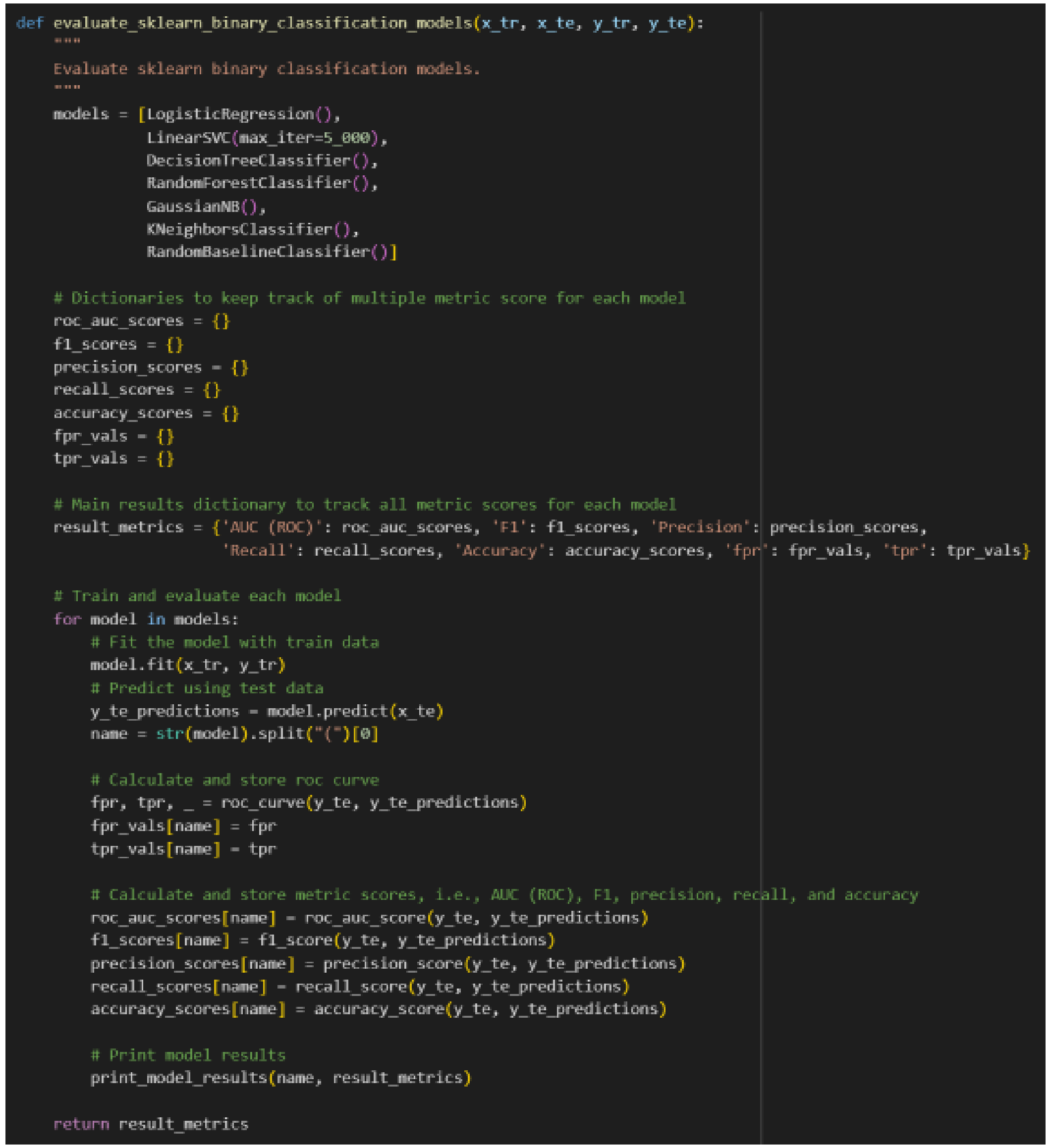
- But first, shown on the right is a simple helper method to print results.
- We'll store result metrics for multiple models inside a dictionary.
- Some of the metrics we'll explore further (and print) include **AUC (ROC)** score, **F1** score, **precision** score, **recall** score, and **accuracy**.



```python
def print_model_results(model_name: str, result_metrics: dict):
    """
    Print model results (from dictionary).
    """
    if result_metrics is None:
        result_metrics = dict()
    print(model_name)
    print(f'AUC (ROC) score:\t{result_metrics.get("AUC (ROC)", dict()).get(model_name, None)}')
    print(f'F1 score:\t\t{result_metrics.get("F1", dict()).get(model_name, None)}')
    print(f'Precision score:\t{result_metrics.get("Precision", dict()).get(model_name, None)}')
    print(f'Recall score:\t\t{result_metrics.get("Recall", dict()).get(model_name, None)}')
    print(f'Accuracy score:\t\t{result_metrics.get("Accuracy", dict()).get(model_name, None)}')
    print()
```

Scikit-learn and random baseline binary classifiers

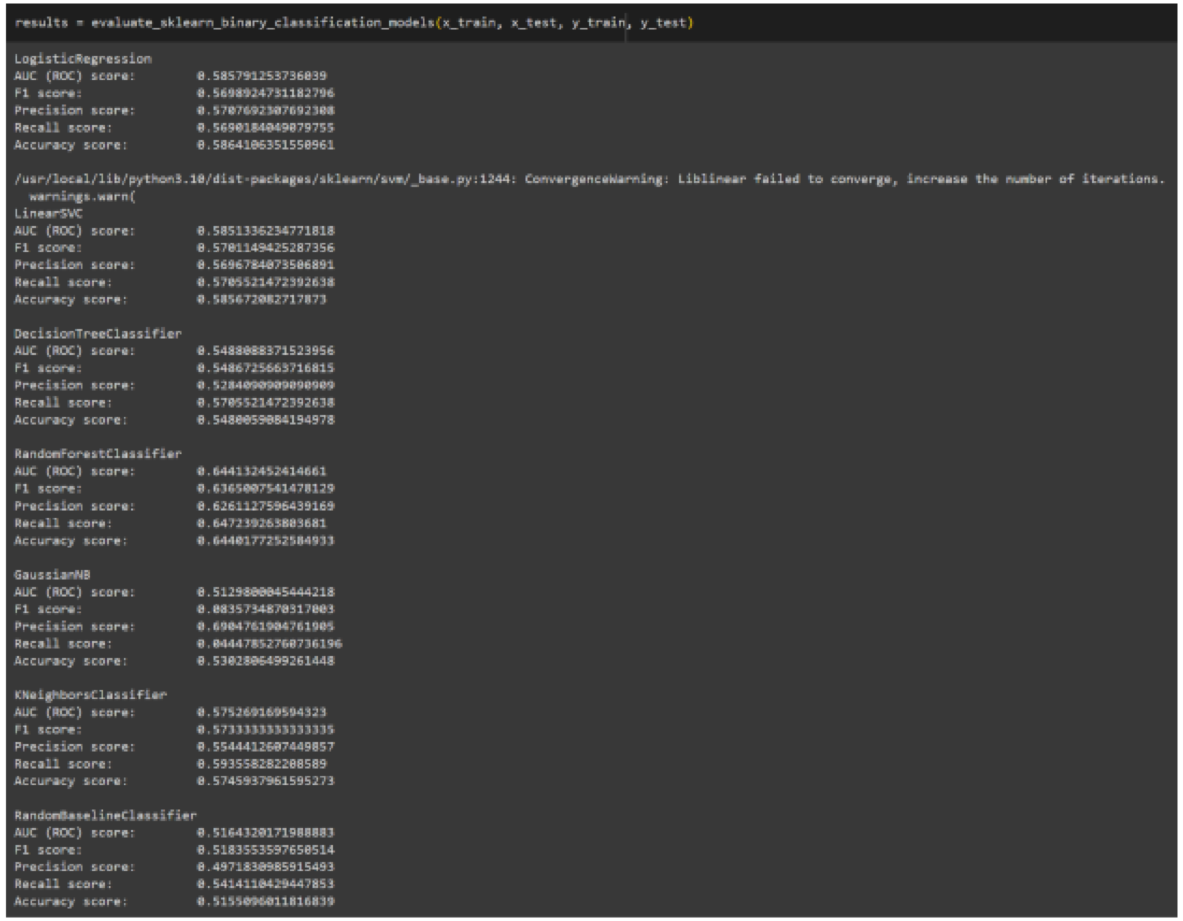
- Method on the right evaluates following binary classifiers (most from scikit-learn):
  - Logistic regression
  - Linear support vector machines
  - Decision tree classifier
  - Random forest classifier
  - Gaussian naïve Bayes
  - K-neighbors classifier
  - Random (baseline) classifier
- Each model is trained on training data and evaluated on test dataset. Result metrics are stored in dictionary and printed to console.



```python
def evaluate_sklearn_binary_classification_models(x_tr, x_te, y_tr, y_te):
    """
    Evaluate sklearn binary classification models.
    """
    models = [LogisticRegression(),
              LinearSVC(max_iter=5_000),
              DecisionTreeClassifier(),
              RandomForestClassifier(),
              GaussianNB(),
              KNeighborsClassifier(),
              RandomBaselineClassifier()]

    # Dictionaries to keep track of multiple metric score for each model
    roc_auc_scores = {}
    f1_scores = {}
    precision_scores = {}
    recall_scores = {}
    accuracy_scores = {}
    fpr_vals = {}
    tpr_vals = {}

    # Main results dictionary to track all metric scores for each model
    result_metrics = {'AUC (ROC)': roc_auc_scores, 'F1': f1_scores, 'Precision': precision_scores,
                      'Recall': recall_scores, 'Accuracy': accuracy_scores, 'fpr': fpr_vals, 'tpr': tpr_vals}

    # Train and evaluate each model
    for model in models:
        # Fit the model with train data
        model.fit(x_tr, y_tr)
        # Predict using test data
        y_te_predictions = model.predict(x_te)
        name = str(model).split("(")[0]

        # Calculate and store roc curve
        fpr, tpr, _ = roc_curve(y_te, y_te_predictions)
        fpr_vals[name] = fpr
        tpr_vals[name] = tpr

        # Calculate and store metric scores, i.e., AUC (ROC), F1, precision, recall, and accuracy
        roc_auc_scores[name] = roc_auc_score(y_te, y_te_predictions)
        f1_scores[name] = f1_score(y_te, y_te_predictions)
        precision_scores[name] = precision_score(y_te, y_te_predictions)
        recall_scores[name] = recall_score(y_te, y_te_predictions)
        accuracy_scores[name] = accuracy_score(y_te, y_te_predictions)

        # Print model results
        print_model_results(name, result_metrics)

    return result_metrics
```

Scikit-learn and random baseline binary classifiers

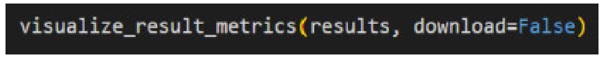
- Below shows execution of evaluating scikit-learn binary classifiers and random (baseline) classifier
- Result metrics are printed to console, but these are visualized and discussed on in slides that follow



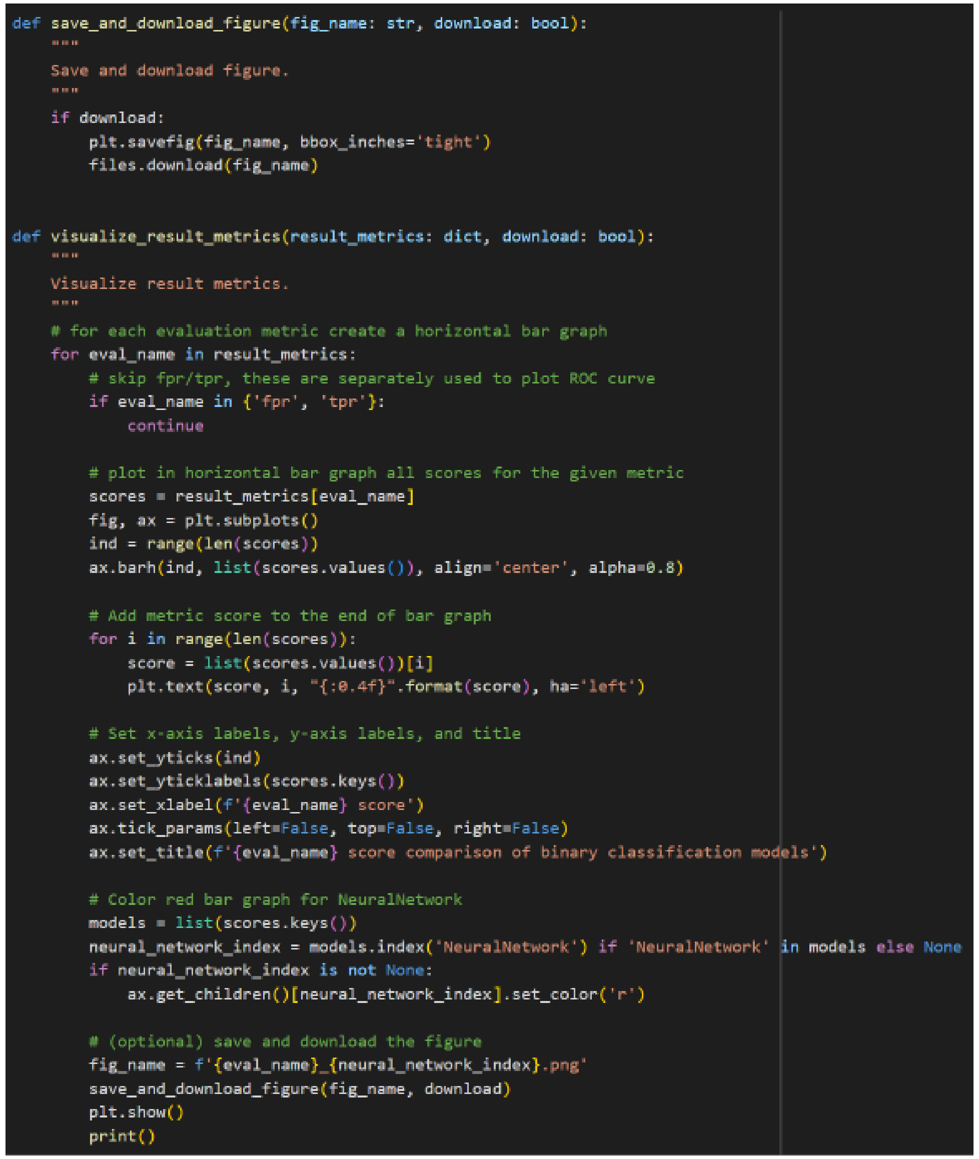
```
results = evaluate_sklearn_binary_classification_models(x_train, x_test, y_train, y_test)

LogisticRegression
AUC (ROC) score:        0.585791253736839
F1 score:               0.5698924731182796
Precision score:        0.5707692307692308
Recall score:           0.5690184049079755
Accuracy score:         0.5864106351550961

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
LinearSVC
AUC (ROC) score:        0.5851336234771818
F1 score:               0.5701149425287356
Precision score:        0.5696784073506891
Recall score:           0.5705521472392638
Accuracy score:         0.585672082717873

DecisionTreeClassifier
AUC (ROC) score:        0.5488088371523956
F1 score:               0.5486725663716815
Precision score:        0.5284090909090909
Recall score:           0.5705521472392638
Accuracy score:         0.5480059084194978

RandomForestClassifier
AUC (ROC) score:        0.644132452414661
F1 score:               0.6365007541478129
Precision score:        0.6261127596439169
Recall score:           0.647239263803681
Accuracy score:         0.6440177252584933

GaussianNB
AUC (ROC) score:        0.5129800045444218
F1 score:               0.0835734870317003
Precision score:        0.6904761904761905
Recall score:           0.04447852760736196
Accuracy score:         0.5302806499261448

KNeighborsClassifier
AUC (ROC) score:        0.575260169594323
F1 score:               0.5733333333333335
Precision score:        0.5544412607449857
Recall score:           0.593558282208589
Accuracy score:         0.5745937961595273

RandomBaselineClassifier
AUC (ROC) score:        0.5164320171988883
F1 score:               0.5183553597650514
Precision score:        0.4971830985915493
Recall score:           0.5414110429447853
Accuracy score:         0.5155096011816839
```

Scikit-learn and random baseline binary classifiers

- Method on the right visualizes result metrics (stored in dictionary).
- For each evaluation metric a horizontal bar plot is created.
- Horizontal bar for neural network (in future slides) is colored red for easy comparison.
- In the slides that follow are these plots.

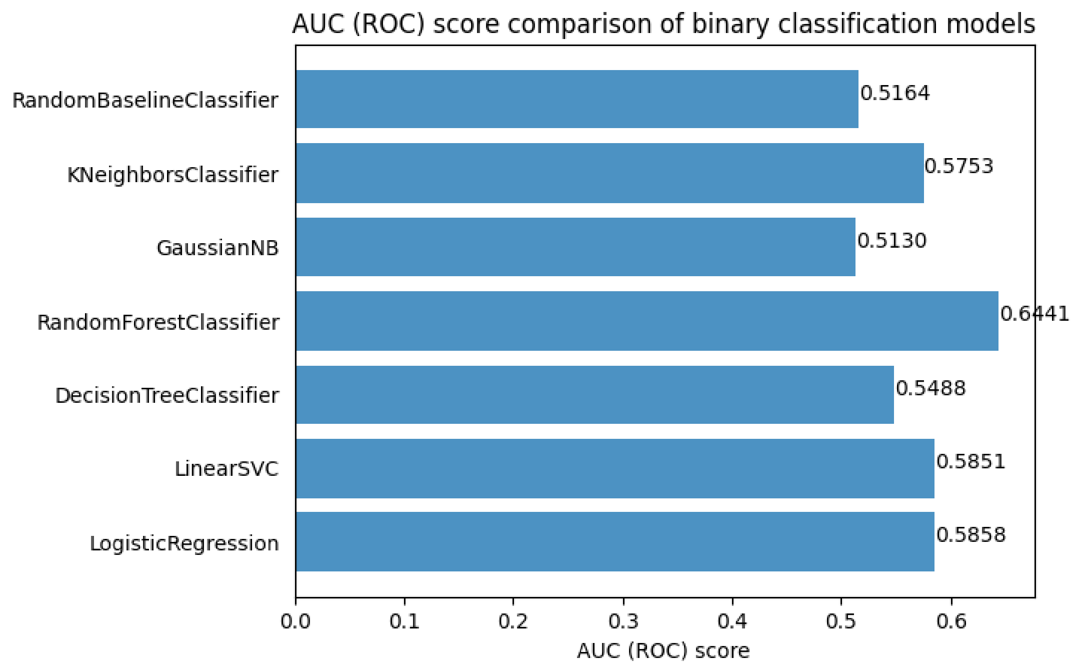


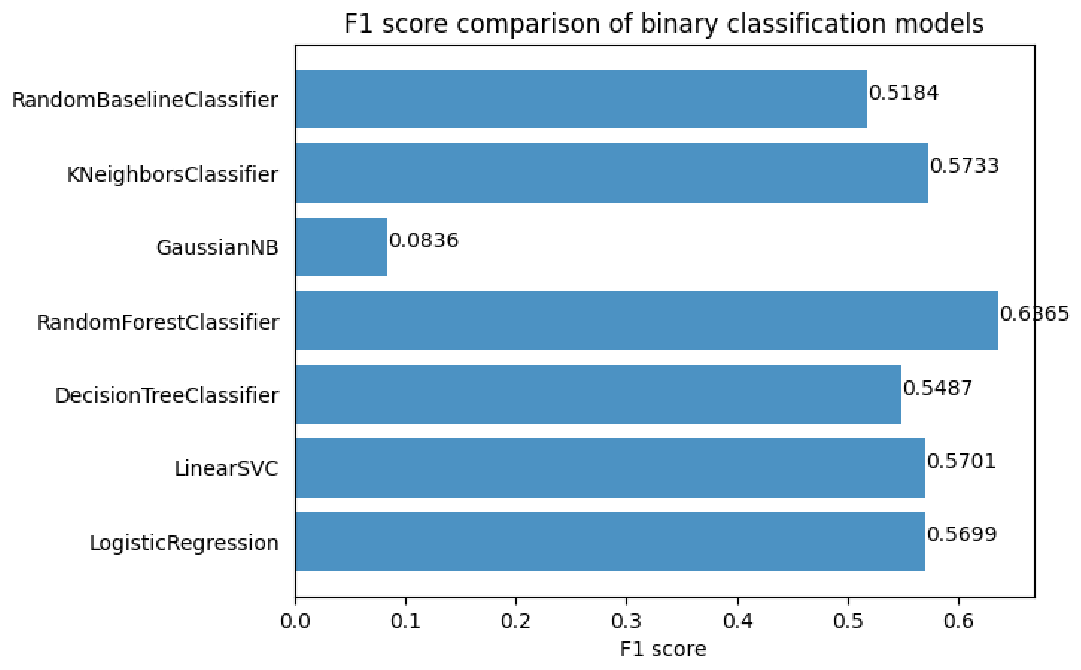
```python
visualize_result_metrics(results, download=False)
```



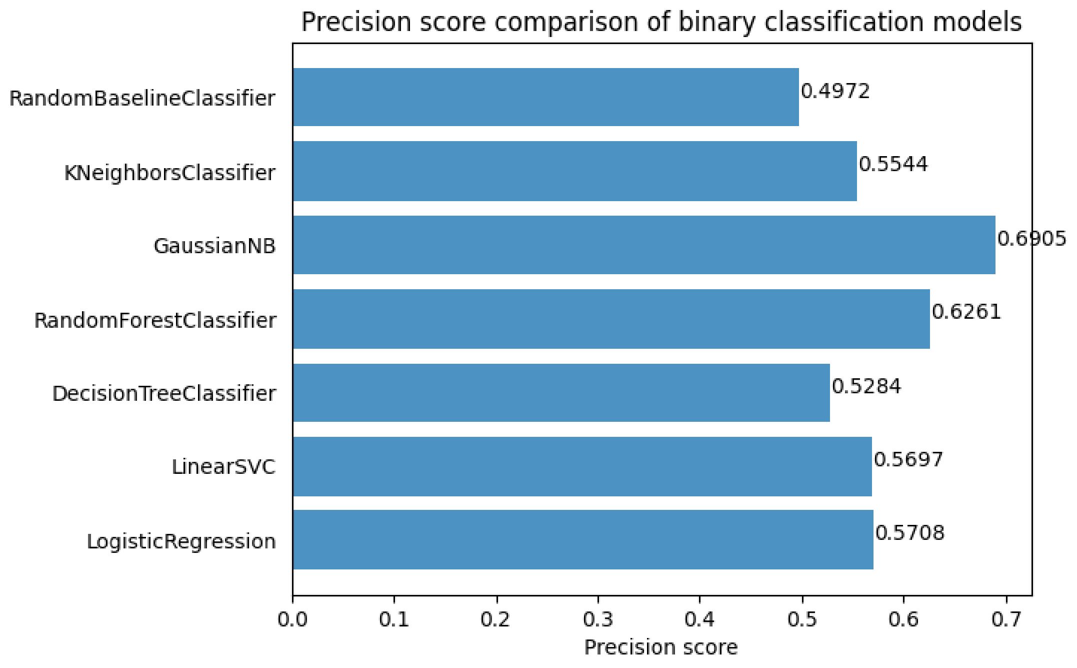
```python
def save_and_download_figure(fig_name: str, download: bool):
    """
    Save and download figure.
    """
    if download:
        plt.savefig(fig_name, bbox_inches='tight')
        files.download(fig_name)


def visualize_result_metrics(result_metrics: dict, download: bool):
    """
    Visualize result metrics.
    """
    # for each evaluation metric create a horizontal bar graph
    for eval_name in result_metrics:
        # skip fpr/tpr, these are separately used to plot ROC curve
        if eval_name in {'fpr', 'tpr'}:
            continue

        # plot in horizontal bar graph all scores for the given metric
        scores = result_metrics[eval_name]
        fig, ax = plt.subplots()
        ind = range(len(scores))
        ax.barh(ind, list(scores.values()), align='center', alpha=0.8)

        # Add metric score to the end of bar graph
        for i in range(len(scores)):
            score = list(scores.values())[i]
            plt.text(score, i, "{:0.4f}".format(score), ha='left')

        # Set x-axis labels, y-axis labels, and title
        ax.set_yticks(ind)
        ax.set_yticklabels(scores.keys())
        ax.set_xlabel(f'{eval_name} score')
        ax.tick_params(left=False, top=False, right=False)
        ax.set_title(f'{eval_name} score comparison of binary classification models')

        # Color red bar graph for NeuralNetwork
        models = list(scores.keys())
        neural_network_index = models.index('NeuralNetwork') if 'NeuralNetwork' in models else None
        if neural_network_index is not None:
            ax.get_children()[neural_network_index].set_color('r')

        # (optional) save and download the figure
        fig_name = f'{eval_name}_{neural_network_index}.png'
        save_and_download_figure(fig_name, download)
        plt.show()
        print()
```

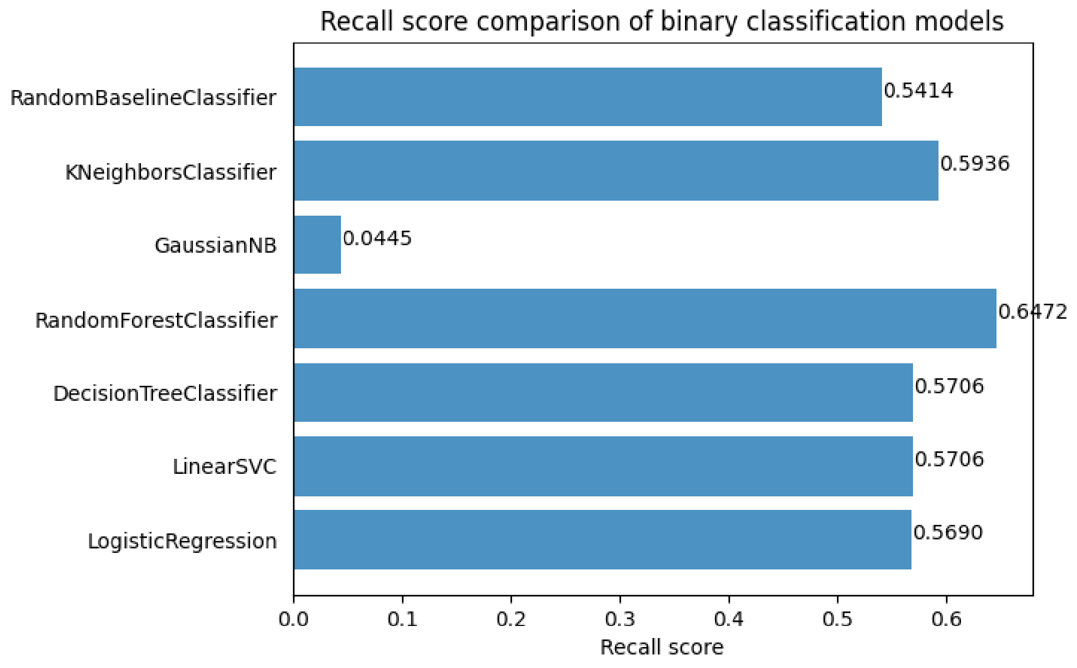
Scikit-learn and random baseline binary classifiers



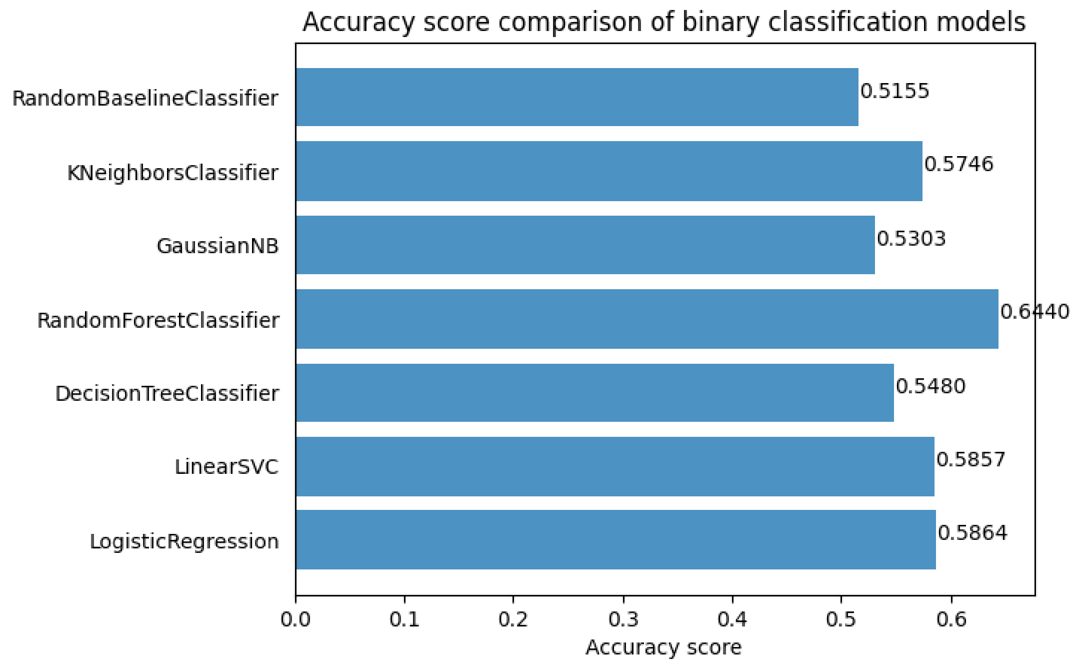


Scikit-learn and random baseline binary classifiers





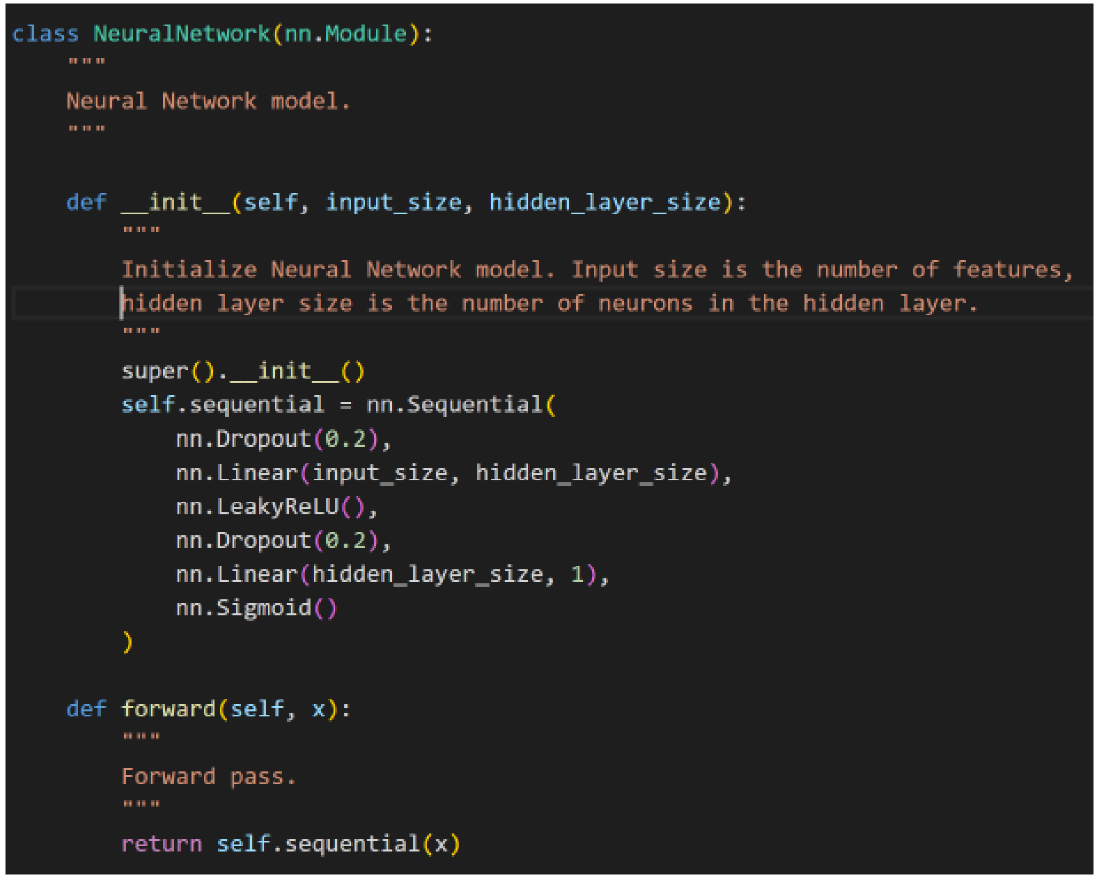
Scikit-learn and random baseline binary classifiers



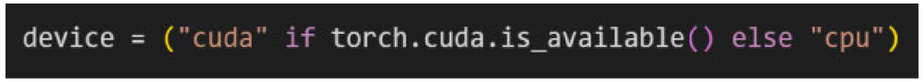
- Key insights:
  - Random forest classifier performs best at mortality prediction overall. With an accuracy of **~0.644**.
  - Random (baseline) classifier performs the worst overall (as expected), with **~0.5155 accuracy** and **~0.5164 AUC (ROC)**.

Train and evaluate PyTorch neural network

- Now, we will build, train, and evaluate a very simple PyTorch neural network.
- First, we need to define a device. This is helpful if running on a GPU. **However, GPU is not required.**
- On the right is implementation of a very simple neural network. It takes as input:
  - **input_size** – The number of features in the training data.
  - **hidden_layer_size** – The number of neurons in the hidden layer.
- The architecture of this neural network is very simple: a single hidden layer followed by a single neuron and a sigmoid layer. Between these we make use of dropout (20%) for regularization and a leaky ReLu function (for non-linearity).



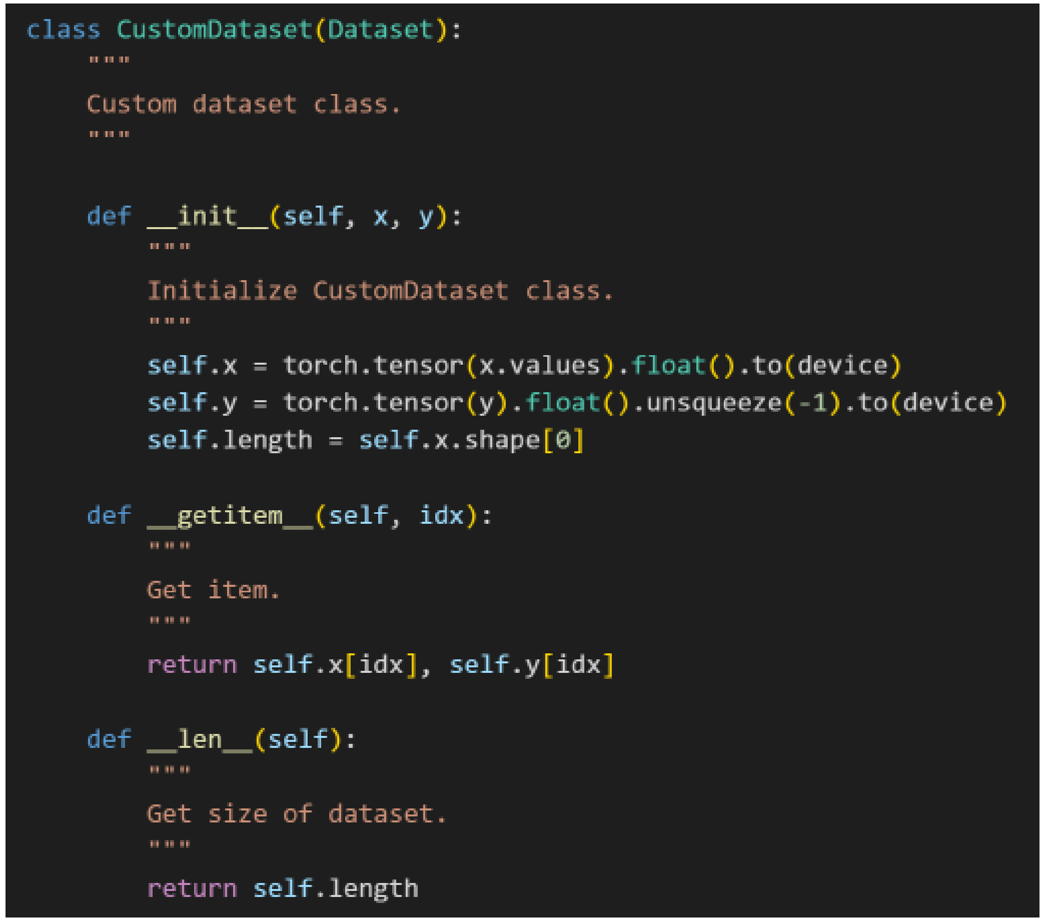
```
class NeuralNetwork(nn.Module):
    """
    Neural Network model.
    """

    def __init__(self, input_size, hidden_layer_size):
        """
        Initialize Neural Network model. Input size is the number of features,
        hidden layer size is the number of neurons in the hidden layer.
        """
        super().__init__()
        self.sequential = nn.Sequential(
            nn.Dropout(0.2),
            nn.Linear(input_size, hidden_layer_size),
            nn.LeakyReLU(),
            nn.Dropout(0.2),
            nn.Linear(hidden_layer_size, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        """
        Forward pass.
        """
        return self.sequential(x)
```



```
device = ("cuda" if torch.cuda.is_available() else "cpu")
```

Train and evaluate PyTorch neural network

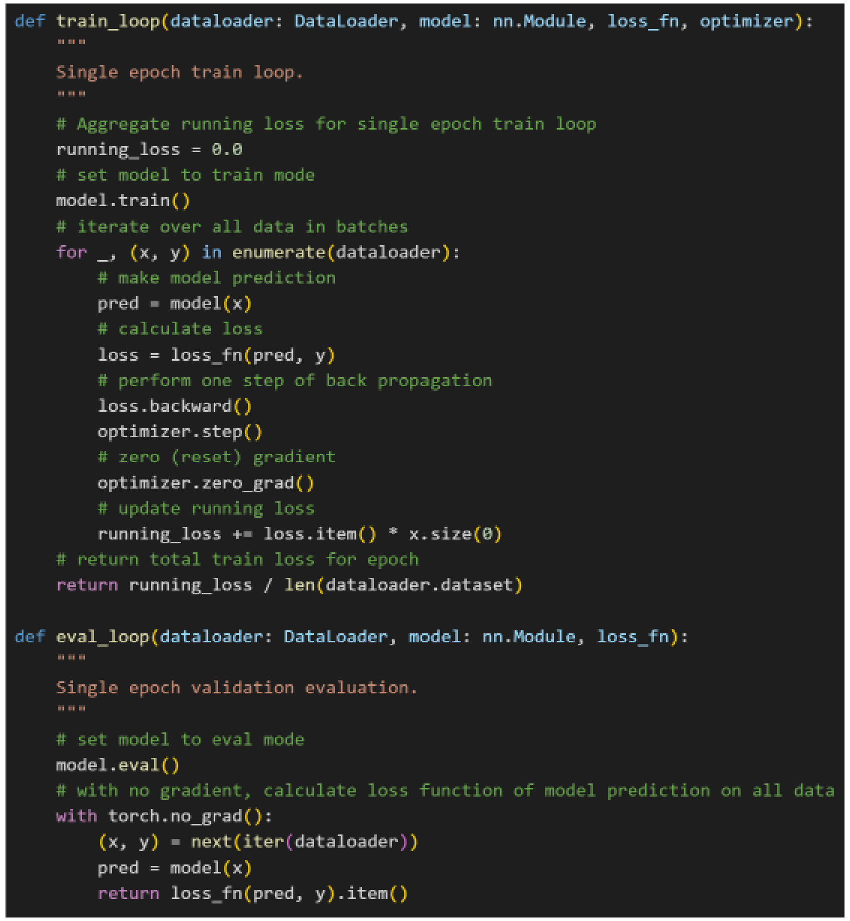
- Before proceeding, we need to define a custom dataset wrapper object, shown on the right.
- Data, features and labels, are stored as PyTorch tensors (float) to device.



```python
class CustomDataset(Dataset):
    """
    Custom dataset class.
    """

    def __init__(self, x, y):
        """
        Initialize CustomDataset class.
        """
        self.x = torch.tensor(x.values).float().to(device)
        self.y = torch.tensor(y).float().unsqueeze(-1).to(device)
        self.length = self.x.shape[0]

    def __getitem__(self, idx):
        """
        Get item.
        """
        return self.x[idx], self.y[idx]

    def __len__(self):
        """
        Get size of dataset.
        """
        return self.length
```

Train and evaluate PyTorch neural network

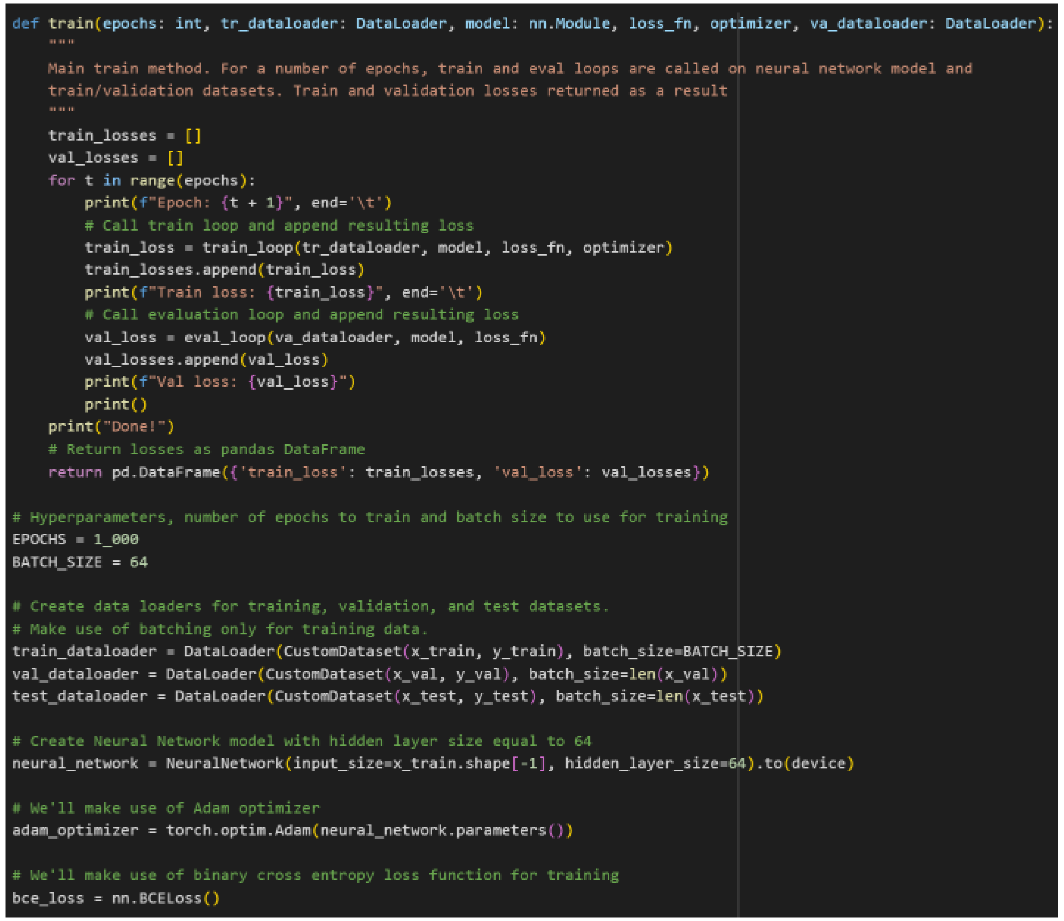
- On the right we define helper methods to train and evaluate neural network a single epoch.
- At a high-level, train loop makes a model prediction, calculates the loss, and performs one step of back propagation. A running loss over the training data is computed and returned.
- The evaluation loop returns the loss of model prediction over all validation dataset.



```python
def train_loop(dataloader: DataLoader, model: nn.Module, loss_fn, optimizer):
    """
    Single epoch train loop.
    """
    # Aggregate running loss for single epoch train loop
    running_loss = 0.0
    # set model to train mode
    model.train()
    # iterate over all data in batches
    for _, (x, y) in enumerate(dataloader):
        # make model prediction
        pred = model(x)
        # calculate loss
        loss = loss_fn(pred, y)
        # perform one step of back propagation
        loss.backward()
        optimizer.step()
        # zero (reset) gradient
        optimizer.zero_grad()
        # update running loss
        running_loss += loss.item() * x.size(0)
    # return total train loss for epoch
    return running_loss / len(dataloader.dataset)

def eval_loop(dataloader: DataLoader, model: nn.Module, loss_fn):
    """
    Single epoch validation evaluation.
    """
    # set model to eval mode
    model.eval()
    # with no gradient, calculate loss function of model prediction on all data
    with torch.no_grad():
        (x, y) = next(iter(dataloader))
        pred = model(x)
        return loss_fn(pred, y).item()
```

Train and evaluate PyTorch neural network

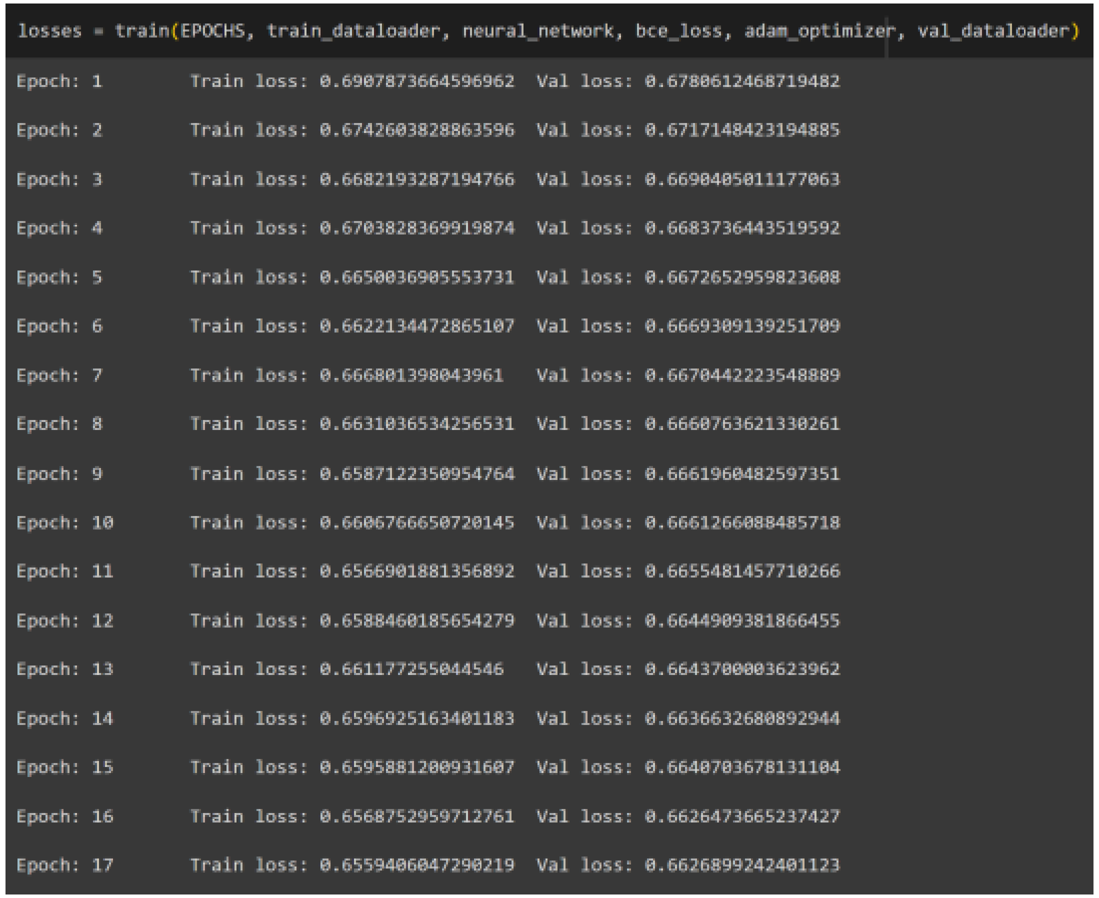
- Putting all previous helper methods together, the train method on the right performs the end-2-end training of the neural network.
- Train method takes as input the model to train, a training data loader, a validation data loader, and multiple hyperparameters such as number of epochs, the loss function, and an optimizer.
- We'll train for **1000 epochs** using a **batch size of 64**.
- **Adam** optimizer and **binary cross entropy loss** are used for training.



```python
def train(epochs: int, tr_dataloader: DataLoader, model: nn.Module, loss_fn, optimizer, va_dataloader: DataLoader):
    """
    Main train method. For a number of epochs, train and eval loops are called on neural network model and
    train/validation datasets. Train and validation losses returned as a result
    """
    train_losses = []
    val_losses = []
    for t in range(epochs):
        print(f"Epoch: {t + 1}", end='\t')
        # Call train loop and append resulting loss
        train_loss = train_loop(tr_dataloader, model, loss_fn, optimizer)
        train_losses.append(train_loss)
        print(f"Train loss: {train_loss}", end='\t')
        # Call evaluation loop and append resulting loss
        val_loss = eval_loop(va_dataloader, model, loss_fn)
        val_losses.append(val_loss)
        print(f"Val loss: {val_loss}")
        print()
    print("Done!")
    # Return losses as pandas DataFrame
    return pd.DataFrame({'train_loss': train_losses, 'val_loss': val_losses})

# Hyperparameters, number of epochs to train and batch size to use for training
EPOCHS = 1_000
BATCH_SIZE = 64

# Create data loaders for training, validation, and test datasets.
# Make use of batching only for training data.
train_dataloader = DataLoader(CustomDataset(x_train, y_train), batch_size=BATCH_SIZE)
val_dataloader = DataLoader(CustomDataset(x_val, y_val), batch_size=len(x_val))
test_dataloader = DataLoader(CustomDataset(x_test, y_test), batch_size=len(x_test))

# Create Neural Network model with hidden layer size equal to 64
neural_network = NeuralNetwork(input_size=x_train.shape[-1], hidden_layer_size=64).to(device)

# We'll make use of Adam optimizer
adam_optimizer = torch.optim.Adam(neural_network.parameters())

# We'll make use of binary cross entropy loss function for training
bce_loss = nn.BCELoss()
```

Train and evaluate PyTorch neural network

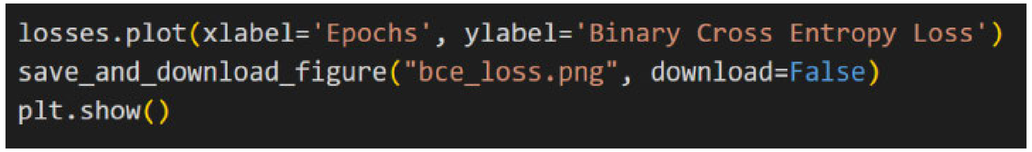
- Running end-2-end training of neural network takes a minute or two. Train and validation losses are printed for each epoch (screenshot on right) and returned at the end.



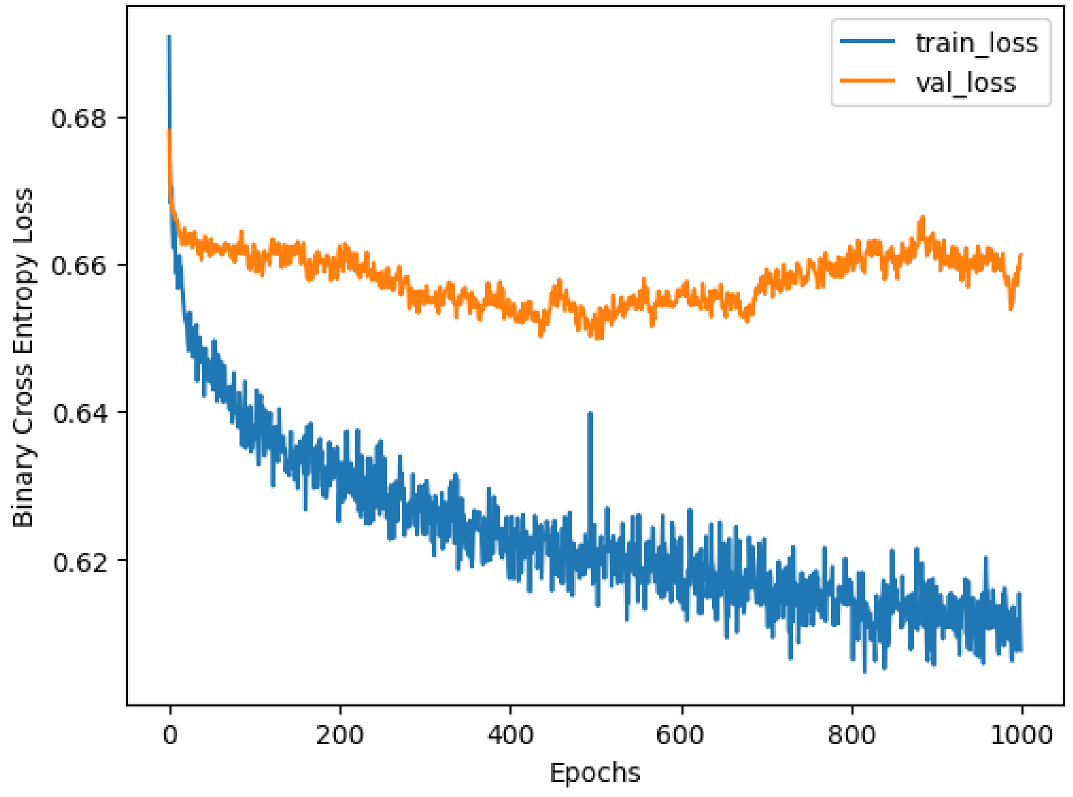
```
losses = train(EPOCHS, train_dataloader, neural_network, bce_loss, adam_optimizer, val_dataloader)

Epoch: 1        Train loss: 0.6907873664596962  Val loss: 0.6780612468719482
Epoch: 2        Train loss: 0.6742603828863596  Val loss: 0.6717148423194885
Epoch: 3        Train loss: 0.6682193287194766  Val loss: 0.6690405011177063
Epoch: 4        Train loss: 0.6703828369919874  Val loss: 0.6683736443519592
Epoch: 5        Train loss: 0.6650036905553731  Val loss: 0.6672652959823608
Epoch: 6        Train loss: 0.6622134472865107  Val loss: 0.6669309139251709
Epoch: 7        Train loss: 0.666801398043961   Val loss: 0.6670442223548889
Epoch: 8        Train loss: 0.6631036534256531  Val loss: 0.6660763621330261
Epoch: 9        Train loss: 0.6587122350954764  Val loss: 0.6661960482597351
Epoch: 10       Train loss: 0.6606766650720145  Val loss: 0.6661266088485718
Epoch: 11       Train loss: 0.6566901881356892  Val loss: 0.6655481457710266
Epoch: 12       Train loss: 0.6588460185654279  Val loss: 0.6644909381866455
Epoch: 13       Train loss: 0.661177255044546   Val loss: 0.6643700003623962
Epoch: 14       Train loss: 0.6596925163401183  Val loss: 0.6636632680892944
Epoch: 15       Train loss: 0.6595881200931607  Val loss: 0.6640703678131104
Epoch: 16       Train loss: 0.6568752959712761  Val loss: 0.6626473665237427
Epoch: 17       Train loss: 0.6559406047290219  Val loss: 0.6626899242401123
```

Train and evaluate PyTorch neural network

- Plotting train and validation losses after end-2-end training of neural network some observations:
  - Train loss decreases throughout epochs.
  - However, validation loss plateaus around ~0.66.
- We need to be very careful with underfitting on the training data and overfitting on the validation data.

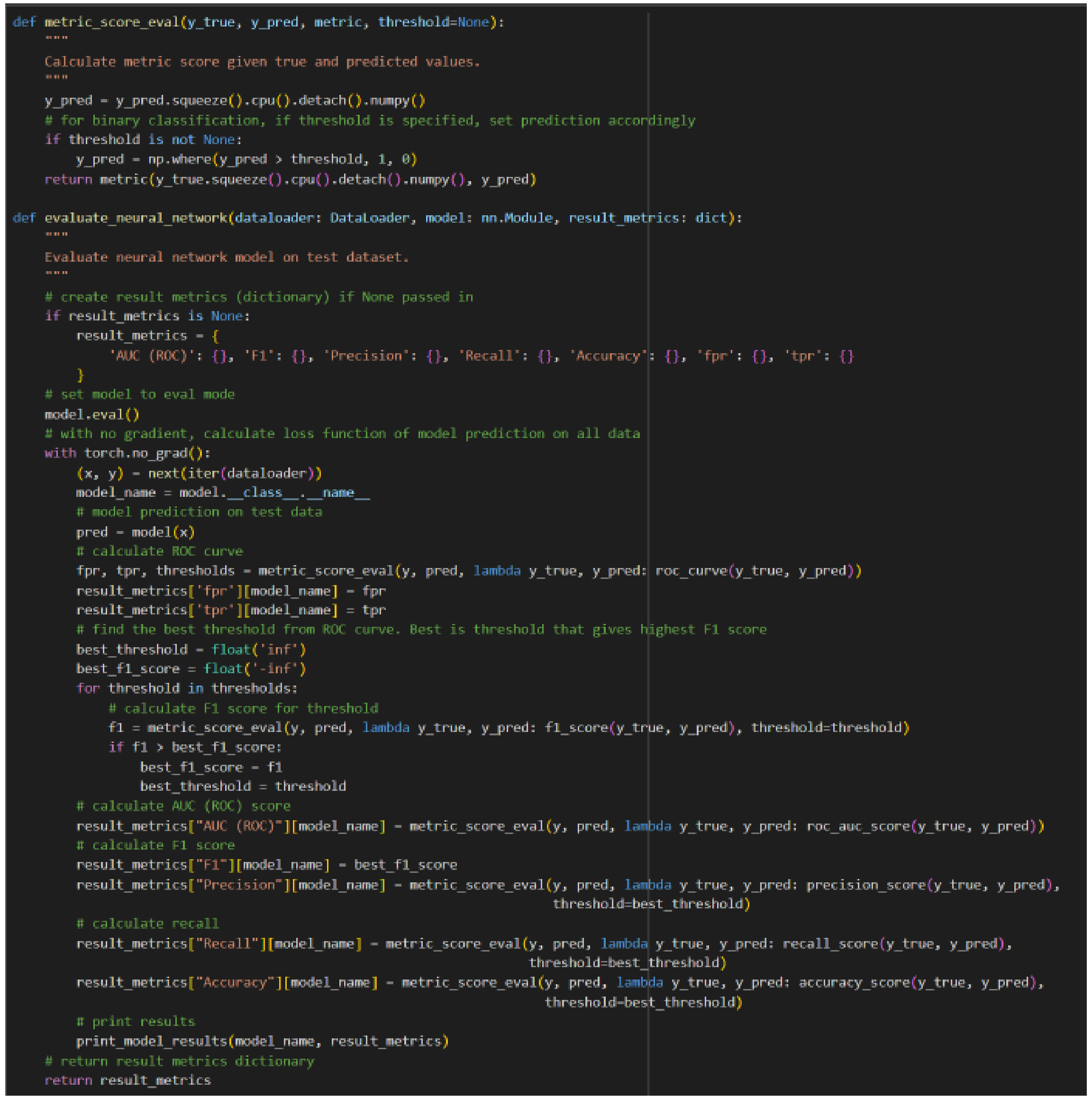


```
losses.plot(xlabel='Epochs', ylabel='Binary Cross Entropy Loss')
save_and_download_figure("bce_loss.png", download=False)
plt.show()
```



Train and evaluate PyTorch neural network

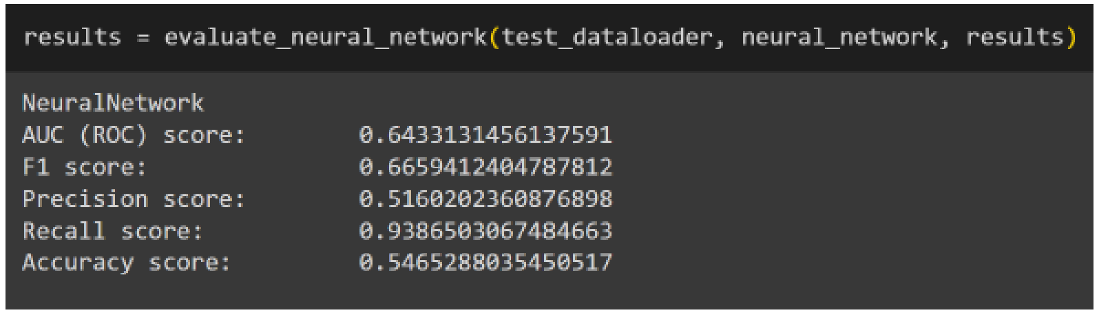
- Helper methods on the right are used to evaluate neural network on test dataset. This includes computing multiple metrics such as **AUC (ROC)**, **F1**, **precision**, **recall**, and **accuracy**.
- The best threshold is also computed based on ROC curve.



```python
def metric_score_eval(y_true, y_pred, metric, threshold=None):
    """
    Calculate metric score given true and predicted values.
    """
    y_pred = y_pred.squeeze().cpu().detach().numpy()
    # for binary classification, if threshold is specified, set prediction accordingly
    if threshold is not None:
        y_pred = np.where(y_pred > threshold, 1, 0)
    return metric(y_true.squeeze().cpu().detach().numpy(), y_pred)

def evaluate_neural_network(dataloader: DataLoader, model: nn.Module, result_metrics: dict):
    """
    Evaluate neural network model on test dataset.
    """
    # create result metrics (dictionary) if None passed in
    if result_metrics is None:
        result_metrics = {
            'AUC (ROC)': {}, 'F1': {}, 'Precision': {}, 'Recall': {}, 'Accuracy': {}, 'fpr': {}, 'tpr': {}
        }
    # set model to eval mode
    model.eval()
    # with no gradient, calculate loss function of model prediction on all data
    with torch.no_grad():
        (x, y) = next(iter(dataloader))
        model_name = model.__class__.__name__
        # model prediction on test data
        pred = model(x)
        # calculate ROC curve
        fpr, tpr, thresholds = metric_score_eval(y, pred, lambda y_true, y_pred: roc_curve(y_true, y_pred))
        result_metrics['fpr'][model_name] = fpr
        result_metrics['tpr'][model_name] = tpr
        # find the best threshold from ROC curve. Best is threshold that gives highest F1 score
        best_threshold = float('inf')
        best_f1_score = float('-inf')
        for threshold in thresholds:
            # calculate F1 score for threshold
            f1 = metric_score_eval(y, pred, lambda y_true, y_pred: f1_score(y_true, y_pred), threshold=threshold)
            if f1 > best_f1_score:
                best_f1_score = f1
                best_threshold = threshold
        # calculate AUC (ROC) score
        result_metrics["AUC (ROC)"][model_name] = metric_score_eval(y, pred, lambda y_true, y_pred: roc_auc_score(y_true, y_pred))
        # calculate F1 score
        result_metrics["F1"][model_name] = best_f1_score
        result_metrics["Precision"][model_name] = metric_score_eval(y, pred, lambda y_true, y_pred: precision_score(y_true, y_pred),
                                                                    threshold=best_threshold)
        # calculate recall
        result_metrics["Recall"][model_name] = metric_score_eval(y, pred, lambda y_true, y_pred: recall_score(y_true, y_pred),
                                                                 threshold=best_threshold)
        result_metrics["Accuracy"][model_name] = metric_score_eval(y, pred, lambda y_true, y_pred: accuracy_score(y_true, y_pred),
                                                                   threshold=best_threshold)
        # print results
        print_model_results(model_name, result_metrics)
    # return result metrics dictionary
    return result_metrics
```

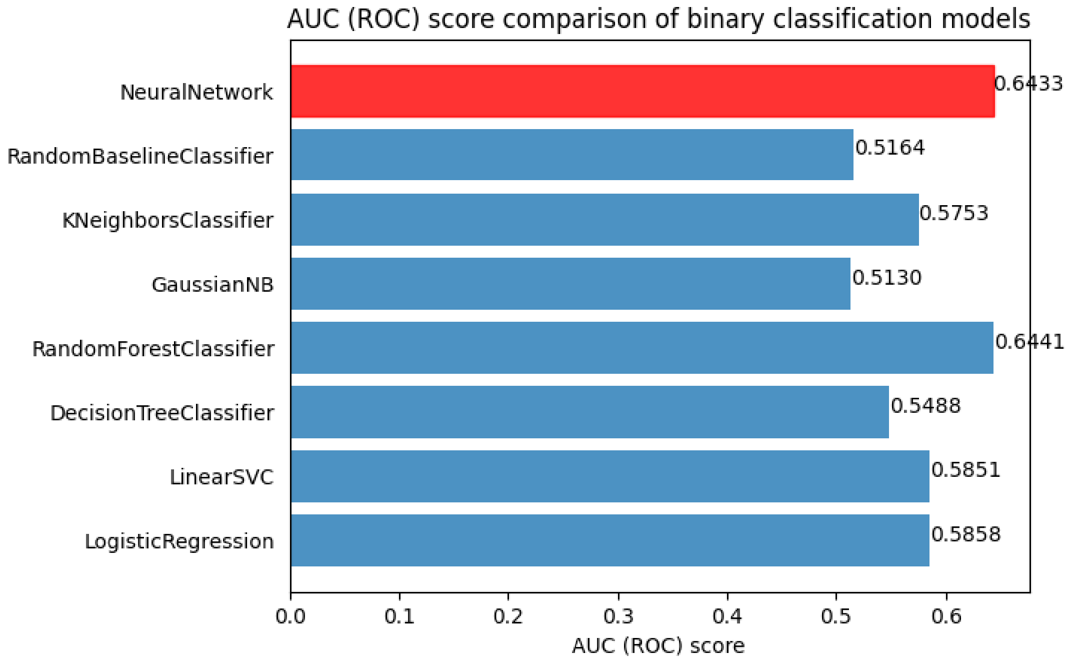
Train and evaluate PyTorch neural network

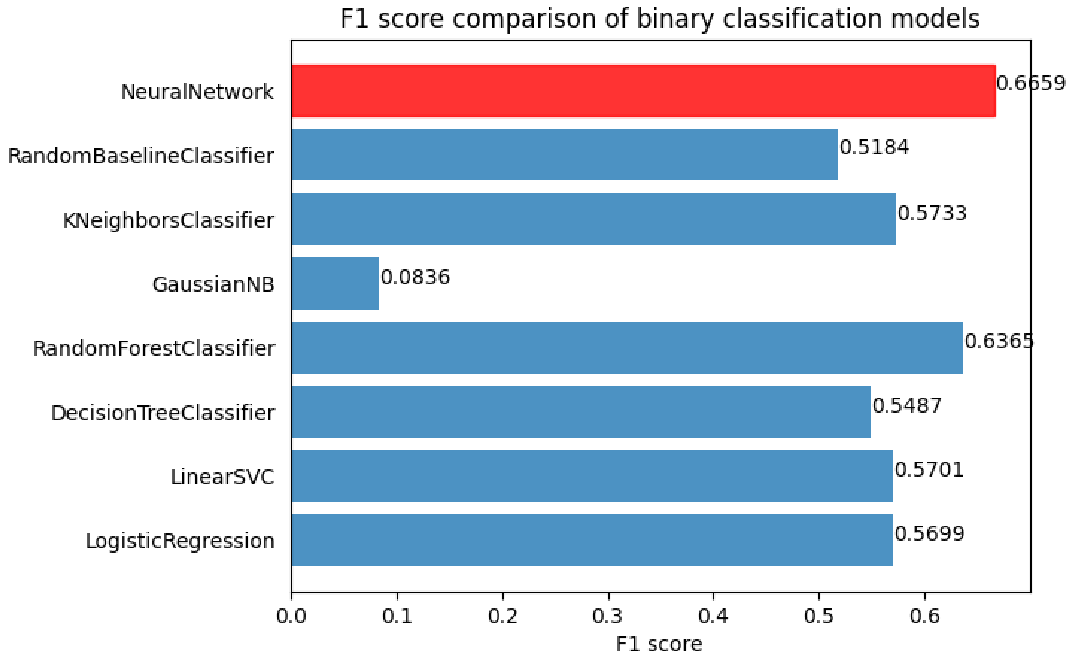
- Evaluating neural network metrics are printed to console and stored in existing results dictionary.
- In next slides we plot and compare all models, including this trained neural network.



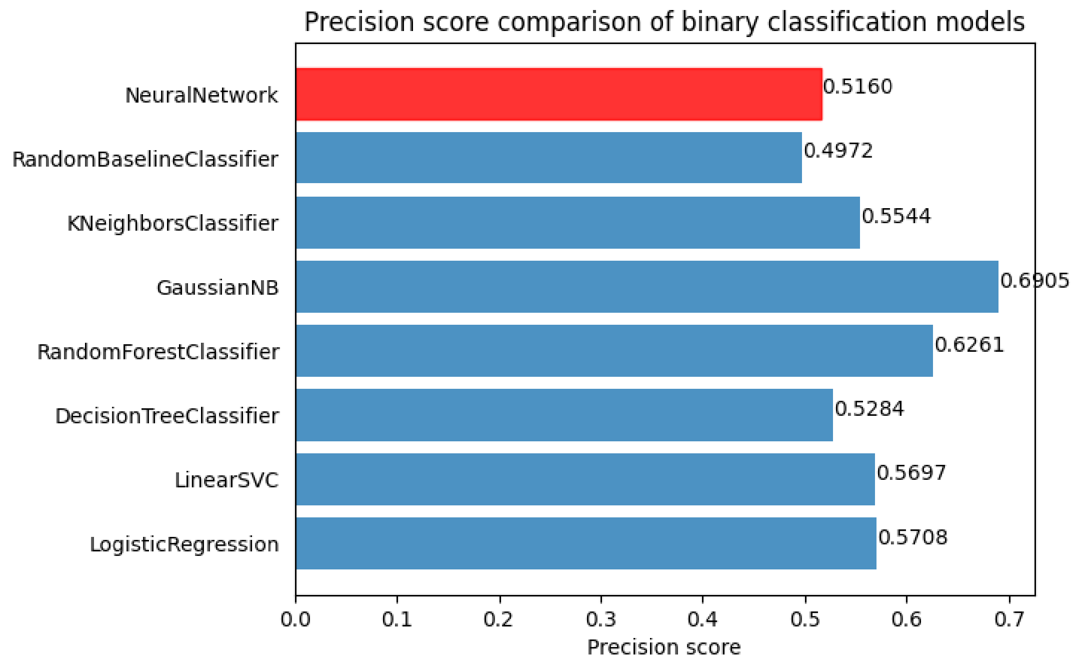
```
results = evaluate_neural_network(test_dataloader, neural_network, results)

NeuralNetwork
AUC (ROC) score:        0.6433131456137591
F1 score:               0.6659412404787812
Precision score:        0.5160202360876898
Recall score:           0.9386503067484663
Accuracy score:         0.5465288035450517
```

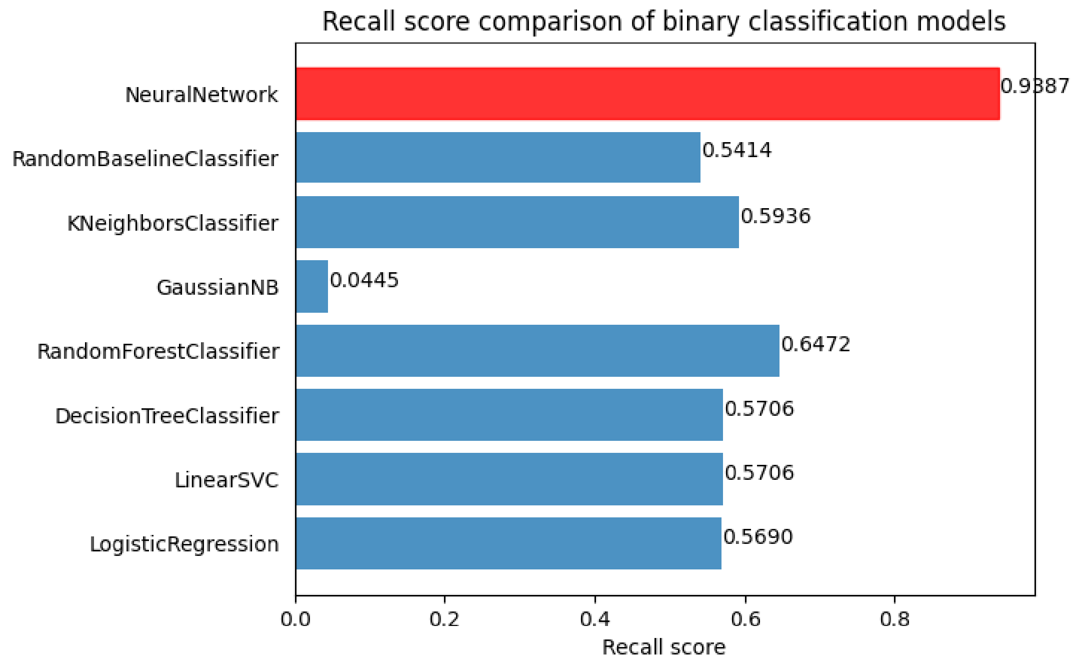
Compare all models



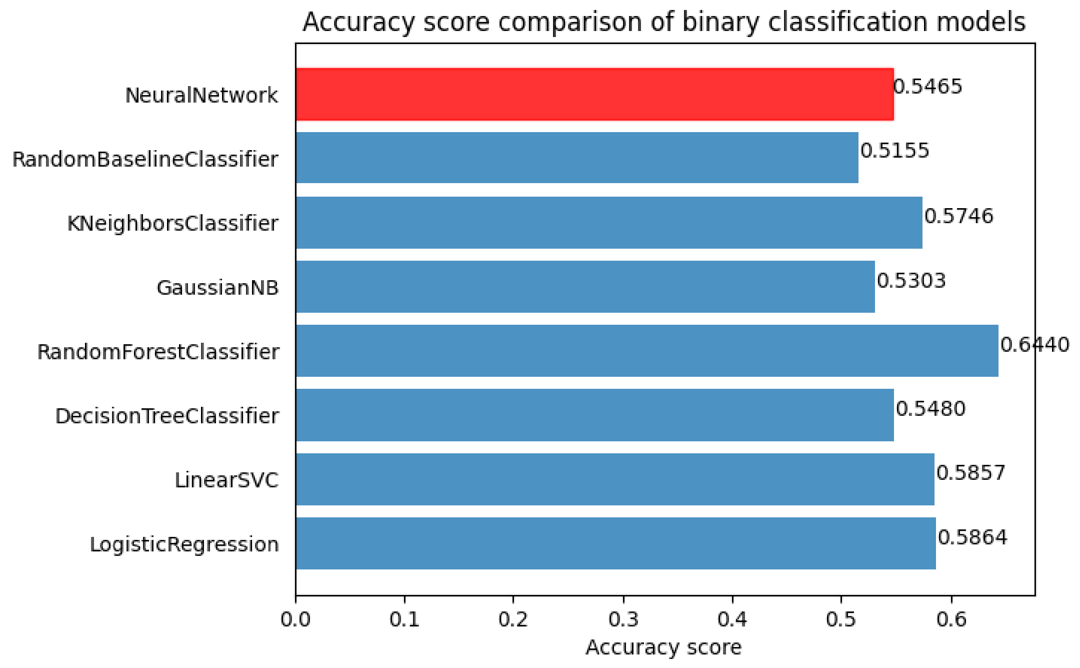


Compare all models





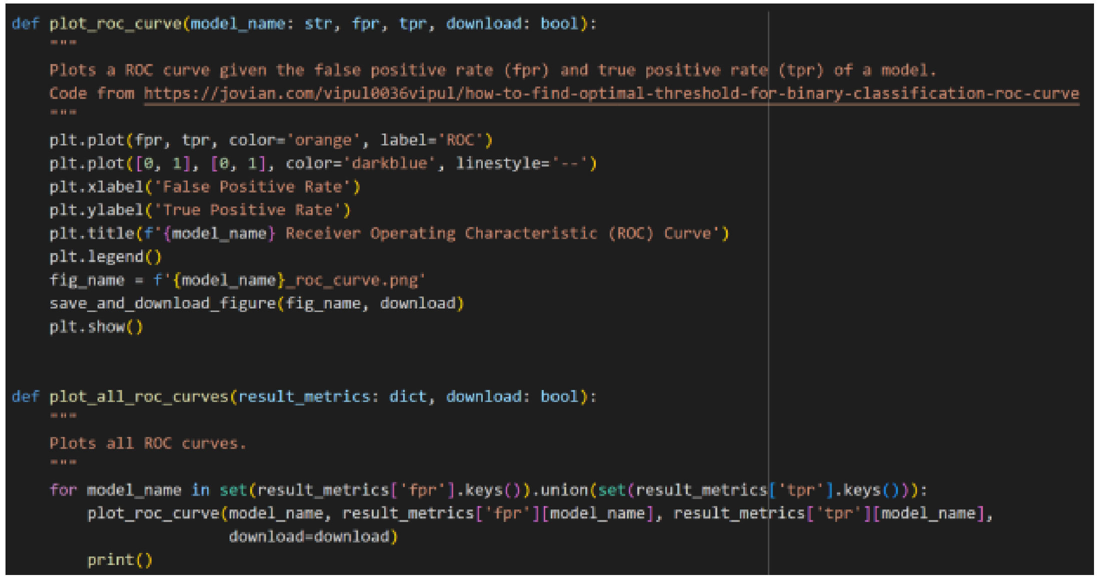
Compare all models



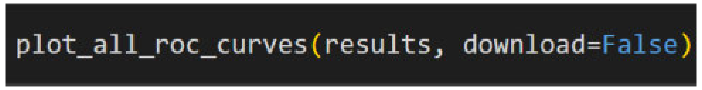
- Key insights:
  - Trained neural network performs better than random guessing.
  - However, random forest classifier still performs better at mortality prediction compared to neural network with regards to accuracy.
  - Neural network outperforms all other models on recall but underperforms on precision.

Compare all models

- In last few slides we will plot ROC curves for all models.
- Code (shown on the right) comes from https://jovian.com/vipul0036vipul/how-to-find-optimal-threshold-for-binary-classification-roc-curve



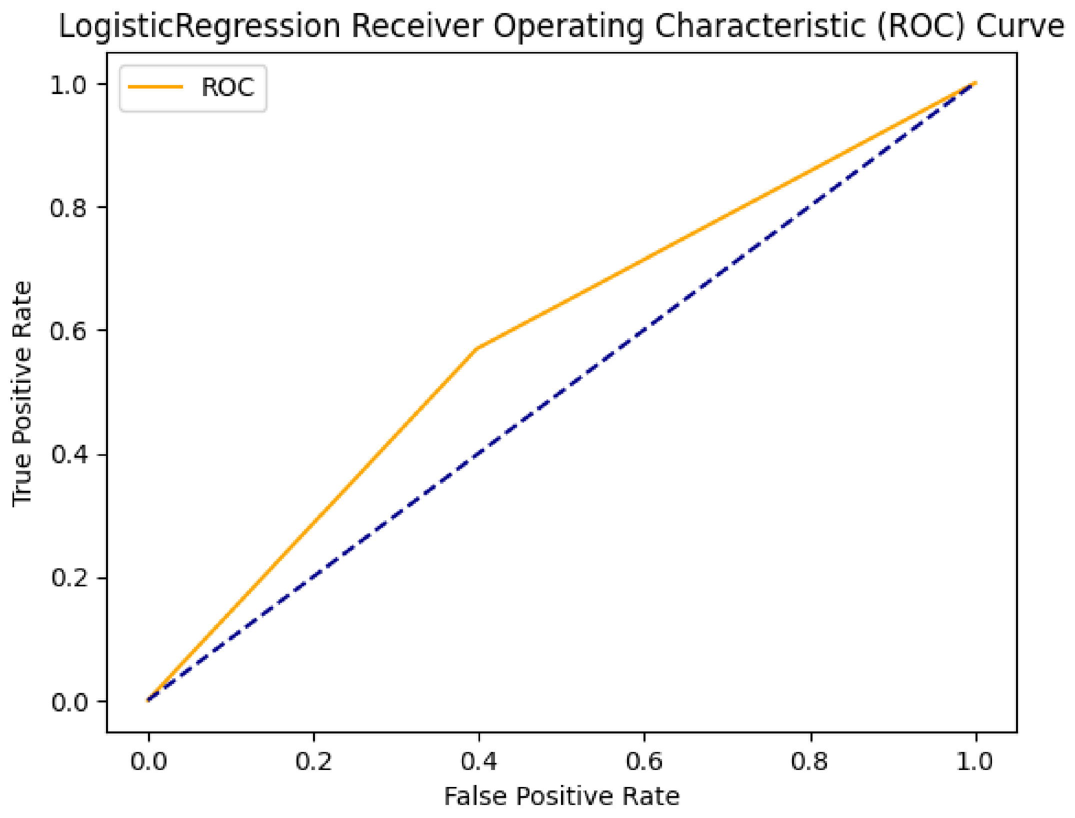
```python
def plot_roc_curve(model_name: str, fpr, tpr, download: bool):
    """
    Plots a ROC curve given the false positive rate (fpr) and true positive rate (tpr) of a model.
    Code from https://jovian.com/vipul0036vipul/how-to-find-optimal-threshold-for-binary-classification-roc-curve
    """
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'{model_name} Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    fig_name = f'{model_name}_roc_curve.png'
    save_and_download_figure(fig_name, download)
    plt.show()


def plot_all_roc_curves(result_metrics: dict, download: bool):
    """
    Plots all ROC curves.
    """
    for model_name in set(result_metrics['fpr'].keys()).union(set(result_metrics['tpr'].keys())):
        plot_roc_curve(model_name, result_metrics['fpr'][model_name], result_metrics['tpr'][model_name],
                       download=download)
        print()
```

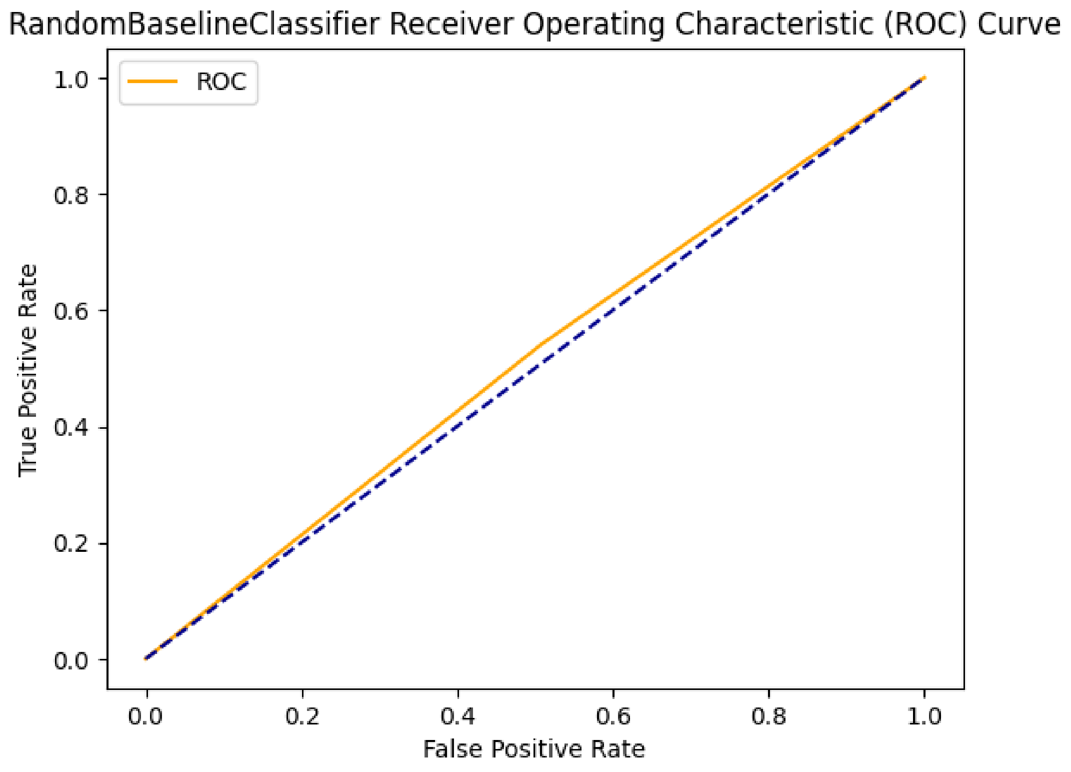


```python
plot_all_roc_curves(results, download=False)
```

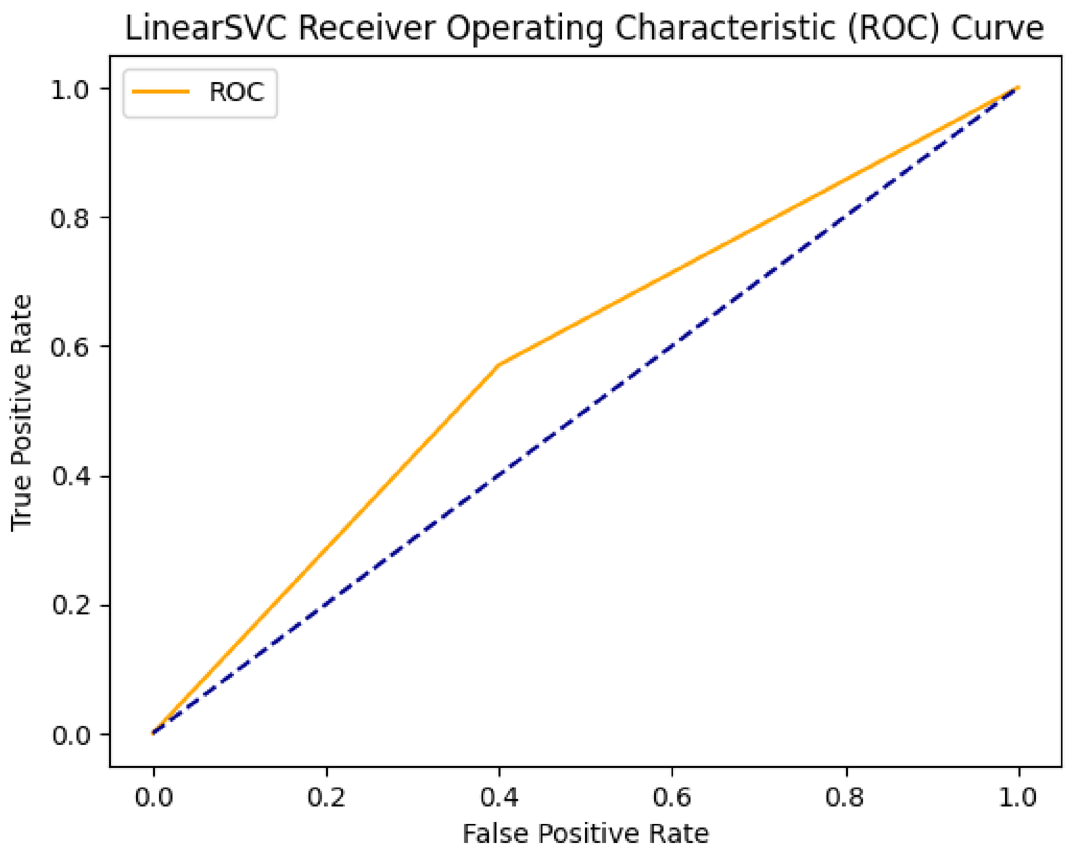
Compare all models

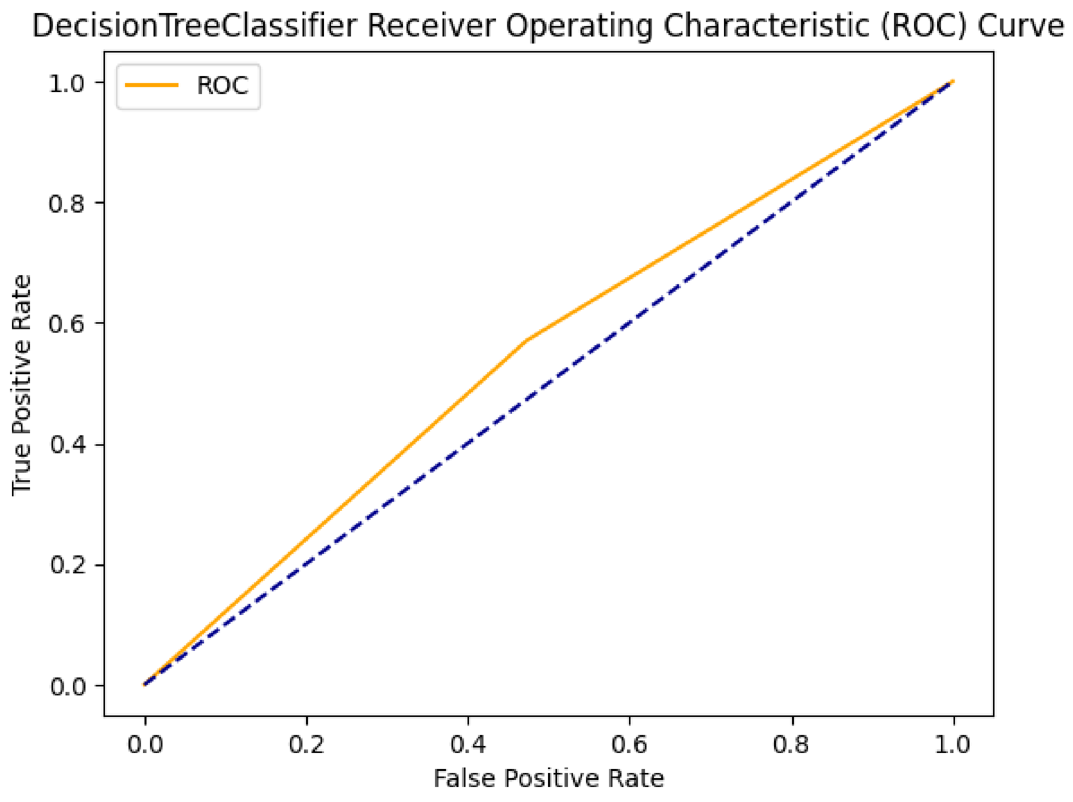
Key insights:
- ROC for random classifier is as expected





Compare all models

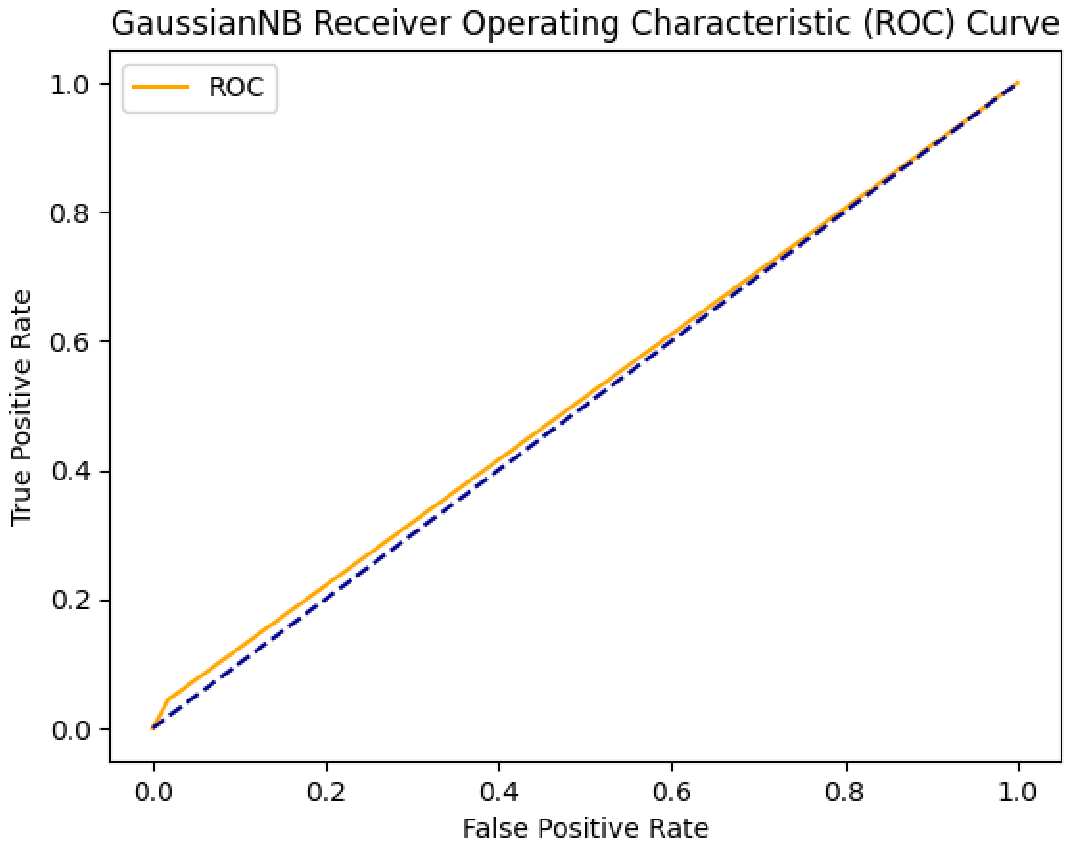


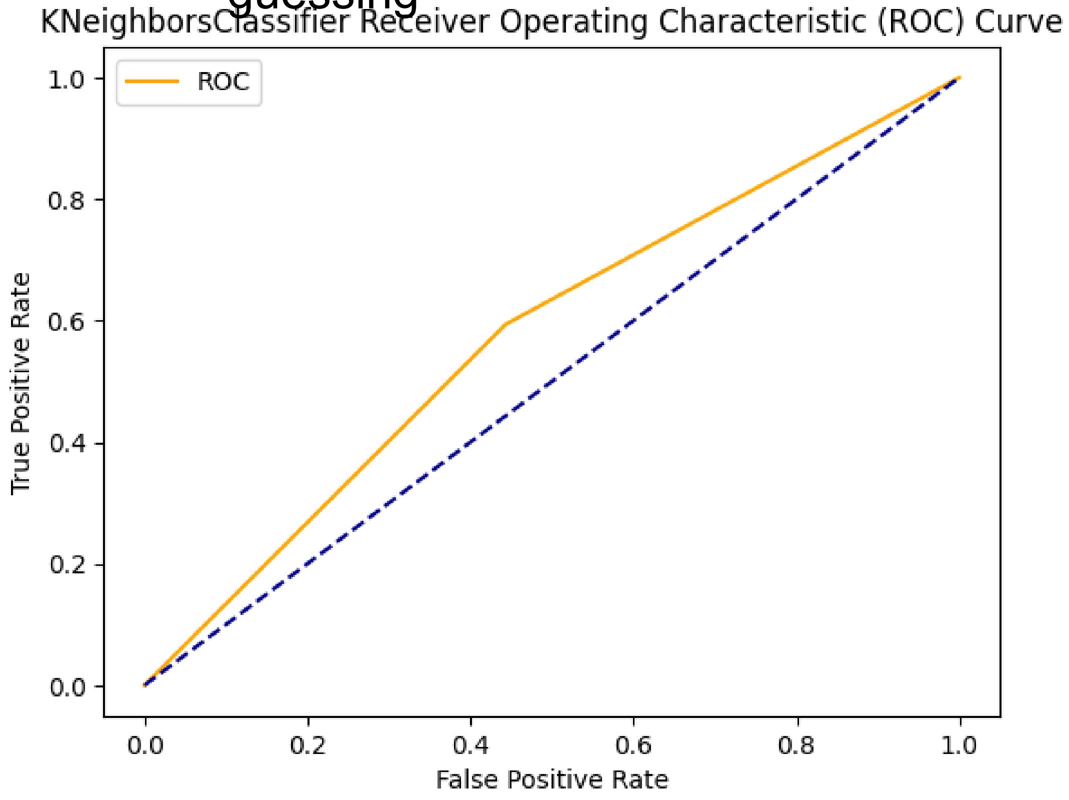


Compare all models

Key insights:

- Gaussian naïve Bayes, which performed the worst, has similar a ROC as random guessing

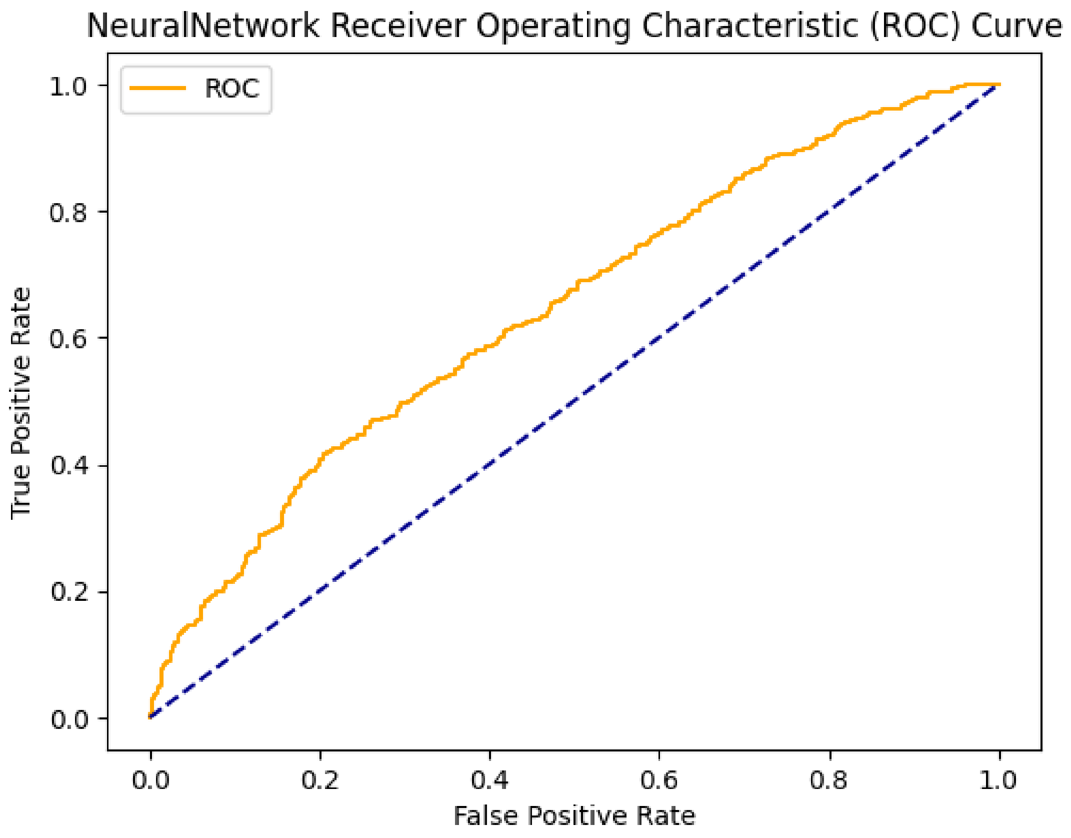


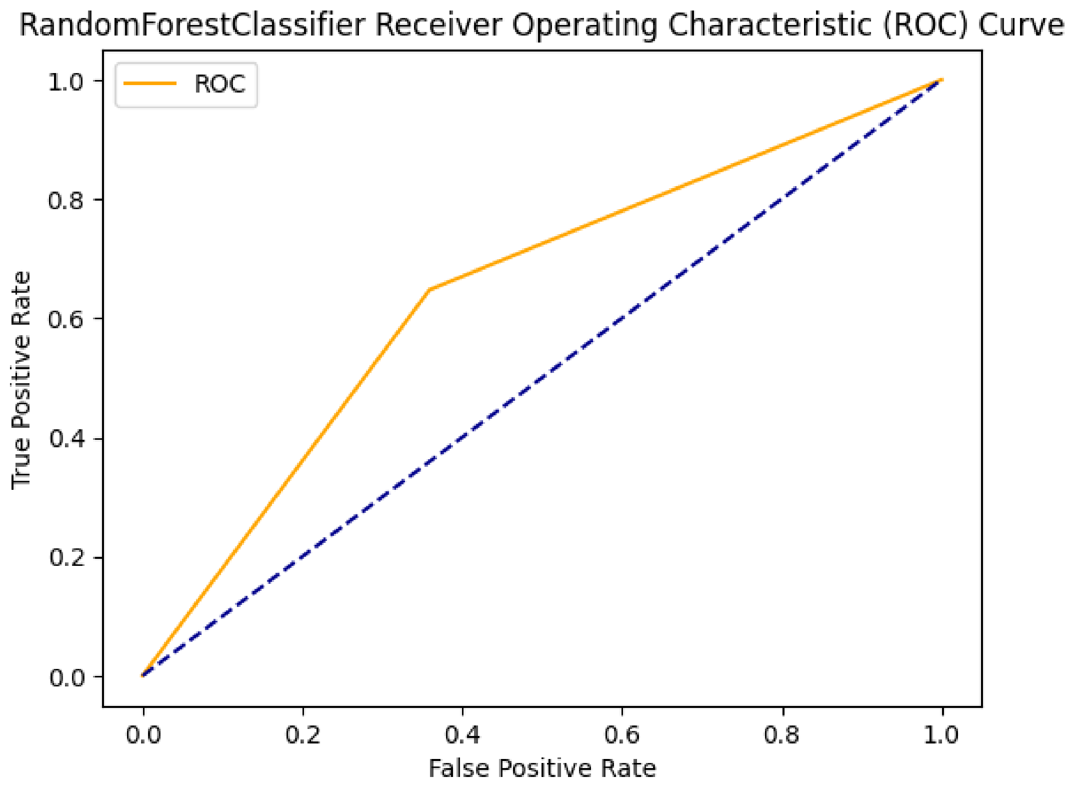


Compare all models

Key insights:

- ROC for neural network and random forest classifier are similar. But there is still a lot of room for improvement.





Learnings and future work

- Some things learned:
  - Mortality prediction on MIMIC-IV (framed as binary classification) is difficult.
  - We see a very simple neural network does perform better than a random classifier. However, there is still a lot of room for improvement.
- Future work / enhancements include:
  - Making use of data augmentation to train better performing model.
  - Play with different model architectures and training hyperparameters. Though we need to be careful not to overfit on the training data.
  - Make use of additional features for the training data. For example, more lab event results, prescribed medications, etc.