```
!pip3 install swifter
```

```
Collecting swifter
  Downloading swifter-1.4.0.tar.gz (1.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 8.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from swifter) (1.5.3)
Requirement already satisfied: psutil>=5.6.6 in /usr/local/lib/python3.10/dist-packages (from swifter) (5.9.5)
Requirement already satisfied: dask[dataframe]>=2.10.0 in /usr/local/lib/python3.10/dist-packages (from swifter) (2023.8
Requirement already satisfied: tqdm>=4.33.0 in /usr/local/lib/python3.10/dist-packages (from swifter) (4.66.2)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->swif
Requirement already satisfied: cloudpickle>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10
Requirement already satisfied: fsspec>=2021.09.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0-
Requirement already satisfied: partd>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->sw
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->s
Requirement already satisfied: toolz>=0.10.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->s
Requirement already satisfied: importlib-metadata>=4.13.0 in /usr/local/lib/python3.10/dist-packages (from dask[datafram
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->sw
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->swifter) (20
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->swifter) (1
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata>=4.13.0->da
Requirement already satisfied: locket in /usr/local/lib/python3.10/dist-packages (from partd>=1.2.0->dask[dataframe]>=2.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>
Building wheels for collected packages: swifter
  Building wheel for swifter (setup.py) ... done
  Created wheel for swifter: filename=swifter-1.4.0-py3-none-any.whl size=16507 sha256=46ee5ee491b22c67bbec86e12f7c08660
  Stored in directory: /root/.cache/pip/wheels/e4/cf/51/0904952972ee2c7aa3709437065278dc534ec1b8d2ad41b443
Successfully built swifter
Installing collected packages: swifter
Successfully installed swifter-1.4.0
```

```
import swifter
```

```
import pandas as pd
import keras
import numpy as np
```

```
import tensorflow as tf
print(tf.__version__)
```

```
2.15.0
```

```
df_mort = df_mort.drop('utdatotid', axis=1)
df_mort = df_mort.drop('date_column_x', axis=1)
df_mort = df_mort.drop('date_column_y', axis=1)
df_mort = df_mort.drop('last_episode_date', axis=1)
df_mort = df_mort.drop('inndatotid', axis=1)
df_mort.drop(['Unnamed: 0'], axis=1, inplace=True)

df_mort_minus = df_mort_minus.drop('date_column', axis=1)
df_mort_minus = df_mort_minus.drop('last_episode_start', axis=1)
df_mort_minus.drop(['Unnamed: 0'], axis=1, inplace=True)

df_read = df_read.drop('inndatotid', axis=1)
df_read = df_read.drop('utdatotid', axis=1)
df_read = df_read.drop('last_episode_date', axis=1)
df_read.drop(['Unnamed: 0'], axis=1, inplace=True)

#df_plos = df_plos.drop('date_column', axis=1)
df_plos = df_plos.drop('last_episode_start', axis=1)
df_plos.drop(['Unnamed: 0'], axis=1, inplace=True)
```

```
#df_plos = df_plos.drop('last_episode_start', axis=1)
df_plos.drop(['Unnamed: 0'], axis=1, inplace=True)
```

```
df_mort['Gender'] = df_mort['kjønn'].map({'Mann': 1, 'Kvinne': 0})
df_mort_minus['Gender'] = df_mort_minus['kjønn'].map({'Mann': 1, 'Kvinne': 0})
df_read['Gender'] = df_read['kjønn'].map({'Mann': 1, 'Kvinne': 0})
df_plos['Gender'] = df_plos['kjønn'].map({'Mann': 1, 'Kvinne': 0})
```

```
df_mort.fillna(0, inplace=True)
df_mort_minus.fillna(0, inplace=True)
df_read.fillna(0, inplace=True)
df_plos.fillna(0, inplace=True)
```

```
df_mort.columns.tolist()
```

```
df_mort.drop(['kjønn'], axis=1, inplace=True)
df_mort.drop(['index'], axis=1, inplace=True)


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assuming '30_day_mortality' is the target variable
X = df_mort.drop('30_day_mortality', axis=1)
y = df_mort['30_day_mortality']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features (important for neural networks and logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


!pip3 install xgboost lightgbm catboost scikit-learn keras tensorflow


from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix

# Define a simple neural network model for binary classification
def build_nn(input_shape):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_shape,)),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    "LightGBM": LGBMClassifier(random_state=42),
    "CatBoost": CatBoostClassifier(verbose=0, random_state=42),
    "Neural Network": build_nn(X_train_scaled.shape[1])
}

# Train each model and evaluate on the test set
results = {}
for name, model in models.items():
    if name == "Neural Network":  # NN requires scaled data
        model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, verbose=0)
        y_pred = (model.predict(X_test_scaled) > 0.5).astype(int).reshape(-1)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    specificity = tn / (tn+fp)
    aucpr = roc_auc_score(y_test, y_pred)  # Use AUC-PR as a proxy for AUPRC; for exact AUPRC, consider using sklearn.metric

    results[name] = {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1,
        "Specificity": specificity,
        "AUPRC": aucpr
    }
print(results)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 5923, number of negative: 22549
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.075563 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 9253
[LightGBM] [Info] Number of data points in the train set: 28472, number of used features: 172
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.208029 -> initscore=-1.336848
[LightGBM] [Info] Start training from score -1.336848
223/223 [==============================] - 0s 1ms/step
{'Random Forest': {'Accuracy': 0.9056047197640118, 'Precision': 0.8532188841201717, 'Recall': 0.6648829431438127, 'F1 Sc
```

```
results
```

```
{'Random Forest': {'Accuracy': 0.9056047197640118,
  'Precision': 0.8532188841201717,
  'Recall': 0.6648829431438127,
  'F1 Score': 0.7473684210526316,
  'Specificity': 0.9695945945945946,
  'AUPRC': 0.8172387688692037},
 'Logistic Regression': {'Accuracy': 0.8319988762466639,
  'Precision': 0.6458536585365854,
  'Recall': 0.442809364548495,
  'F1 Score': 0.5253968253968254,
  'Specificity': 0.9354551920341394,
  'AUPRC': 0.6891322782913172},
 'XGBoost': {'Accuracy': 0.9130495856159573,
  'Precision': 0.8220588235294117,
  'Recall': 0.7478260869565218,
  'F1 Score': 0.7831873905429072,
  'Specificity': 0.9569701280227596,
  'AUPRC': 0.8523981074896407},
 'LightGBM': {'Accuracy': 0.916842253125439,
  'Precision': 0.8332103321033211,
  'Recall': 0.7551839464882943,
  'F1 Score': 0.792280701754386,
  'Specificity': 0.9598150782361309,
  'AUPRC': 0.8574995123622127},
 'CatBoost': {'Accuracy': 0.917685068127546,
  'Precision': 0.8399401645474944,
  'Recall': 0.7511705685618729,
  'F1 Score': 0.7930790960451978,
  'Specificity': 0.9619487908961594,
  'AUPRC': 0.8565596797290161},
 'Neural Network': {'Accuracy': 0.8863604438825677,
  'Precision': 0.7418899858956276,
  'Recall': 0.7036789297658863,
  'F1 Score': 0.7222794370065224,
  'Specificity': 0.9349217638691323,
  'AUPRC': 0.8193003468175093}}
```

```
df_mort_minus.columns.tolist()
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assuming '30_day_mortality' is the target variable
X = df_mort_minus.drop('30_day_mortality', axis=1)
y = df_mort_minus['30_day_mortality']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features (important for neural networks and logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Define a simple neural network model for binary classification
def build_nn(input_shape):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_shape,)),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    "LightGBM": LGBMClassifier(random_state=42),
    "CatBoost": CatBoostClassifier(verbose=0, random_state=42),
    "Neural Network": build_nn(X_train_scaled.shape[1])
}

# Train each model and evaluate on the test set
results = {}
for name, model in models.items():
    if name == "Neural Network":  # NN requires scaled data
        model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, verbose=0)
        y_pred = (model.predict(X_test_scaled) > 0.5).astype(int).reshape(-1)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    specificity = tn / (tn+fp)
    aucpr = roc_auc_score(y_test, y_pred)  # Use AUC-PR as a proxy for AUPRC; for exact AUPRC, consider using sklearn.metric

    results[name] = {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1,
        "Specificity": specificity,
        "AUPRC": aucpr
    }
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 5923, number of negative: 22549
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.075142 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 8765
[LightGBM] [Info] Number of data points in the train set: 28472, number of used features: 173
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.208029 -> initscore=-1.336848
[LightGBM] [Info] Start training from score -1.336848
223/223 [==============================] - 0s 1ms/step
```

```
results
```

```
{'Random Forest': {'Accuracy': 0.8893102963899424,
  'Precision': 0.812555260831123,
  'Recall': 0.6147157190635452,
  'F1 Score': 0.6999238385377,
  'Specificity': 0.9623044096728307,
  'AUPRC': 0.7885100643681879},
 'Logistic Regression': {'Accuracy': 0.8137378845343447,
  'Precision': 0.586489252814739,
  'Recall': 0.3832775919732441,
  'F1 Score': 0.46359223300970875,
  'Specificity': 0.9281650071123755,
  'AUPRC': 0.6557212995428098},
 'XGBoost': {'Accuracy': 0.9004073605843518,
  'Precision': 0.788546255506608,
  'Recall': 0.7183946488294315,
  'F1 Score': 0.751837591879594,
```

```
      'Specificity': 0.9487908961593172,
      'AUPRC': 0.8335927724943744},
     'LightGBM': {'Accuracy': 0.905464250596994,
      'Precision': 0.8004385964912281,
      'Recall': 0.7324414715719063,
      'F1 Score': 0.7649318896262662,
      'Specificity': 0.9514580369843528,
      'AUPRC': 0.8419497542781297},
     'CatBoost': {'Accuracy': 0.9058856580980474,
      'Precision': 0.8030859662013226,
      'Recall': 0.7311036789297659,
      'F1 Score': 0.7654061624649858,
      'Specificity': 0.9523470839260313,
      'AUPRC': 0.8417253814278985},
     'Neural Network': {'Accuracy': 0.8741396263520157,
      'Precision': 0.7197358767424799,
      'Recall': 0.6561872909698997,
      'F1 Score': 0.6864940517844647,
      'Specificity': 0.932076813655761,
      'AUPRC': 0.7941320523128303}}
```

```
df_read.columns.tolist()
```

```
df_read.drop(['kjønn'], axis=1, inplace=True)
df_read.drop(['index'], axis=1, inplace=True)
df_read.drop(['second_last_episode_end'], axis=1, inplace=True)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assuming '30_day_mortality' is the target variable
X = df_read.drop('Labels', axis=1)
y = df_read['Labels']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features (important for neural networks and logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
# Define a simple neural network model for binary classification
def build_nn(input_shape):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_shape,)),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    "LightGBM": LGBMClassifier(random_state=42),
    "CatBoost": CatBoostClassifier(verbose=0, random_state=42),
    "Neural Network": build_nn(X_train_scaled.shape[1])
}

# Train each model and evaluate on the test set
results = {}
for name, model in models.items():
    if name == "Neural Network":  # NN requires scaled data
        model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, verbose=0)
        y_pred = (model.predict(X_test_scaled) > 0.5).astype(int).reshape(-1)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    specificity = tn / (tn+fp)
    aucpr = roc_auc_score(y_test, y_pred)  # Use AUC-PR as a proxy for AUPRC; for exact AUPRC, consider using sklearn.metric

    results[name] = {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1,
        "Specificity": specificity,
        "AUPRC": aucpr
    }
results
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 4741, number of negative: 23731
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.044832 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 9189
[LightGBM] [Info] Number of data points in the train set: 28472, number of used features: 174
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.166514 -> initscore=-1.610534
[LightGBM] [Info] Start training from score -1.610534
223/223 [==============================] - 0s 1ms/step
{'Random Forest': {'Accuracy': 0.844922039612305,
  'Precision': 0.7333333333333333,
  'Recall': 0.1196652719665272,
  'F1 Score': 0.20575539568345322,
  'Specificity': 0.9912221471978393,
  'AUPRC': 0.5554437095821833},
 'Logistic Regression': {'Accuracy': 0.8277848012361286,
  'Precision': 0.4124293785310734,
  'Recall': 0.06108786610878661,
  'F1 Score': 0.10641399416909621,
  'Specificity': 0.9824442943956786,
  'AUPRC': 0.5217660802522326},
 'XGBoost': {'Accuracy': 0.8460457929484478,
  'Precision': 0.5866900175131349,
  'Recall': 0.2803347280334728,
  'F1 Score': 0.3793884484711212,
  'Specificity': 0.9601620526671168,
  'AUPRC': 0.6202483903502949},
 'LightGBM': {'Accuracy': 0.8525073746312685,
  'Precision': 0.6593406593406593,
```

```
  'Recall': 0.2510460251046025,
  'F1 Score': 0.3636363636363636,
  'Specificity': 0.9738352464550979,
  'AUPRC': 0.6124406357798502},
 'CatBoost': {'Accuracy': 0.8563000421407501,
  'Precision': 0.6885964912280702,
  'Recall': 0.26276150627615064,
  'F1 Score': 0.3803755299818293,
  'Specificity': 0.9760297096556381,
  'AUPRC': 0.6193956079658943},
 'Neural Network': {'Accuracy': 0.7937912628178115,
  'Precision': 0.3828326180257511,
  'Recall': 0.3732217573221757,
  'F1 Score': 0.3779661016949153,
  'Specificity': 0.8786293045239703,
  'AUPRC': 0.625925530923073}}
```

Start coding or generate with AI.

```
df_plos.columns.tolist()
```

```
df_plos.drop(['kjønn'], axis=1, inplace=True)
#df_plos.drop(['index'], axis=1, inplace=True)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assuming '30_day_mortality' is the target variable
X = df_plos.drop('PLOS', axis=1)
y = df_plos['PLOS']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features (important for neural networks and logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Define a simple neural network model for binary classification
def build_nn(input_shape):
    model = Sequential([
        Dense(128, activation='relu', input_shape=(input_shape,)),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    "LightGBM": LGBMClassifier(random_state=42),
    "CatBoost": CatBoostClassifier(verbose=0, random_state=42),
    "Neural Network": build_nn(X_train_scaled.shape[1])
}

# Train each model and evaluate on the test set
results = {}
for name, model in models.items():
    if name == "Neural Network":  # NN requires scaled data
        model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, verbose=0)
        y_pred = (model.predict(X_test_scaled) > 0.5).astype(int).reshape(-1)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    specificity = tn / (tn+fp)
    aucpr = roc_auc_score(y_test, y_pred)  # Use AUC-PR as a proxy for AUPRC; for exact AUPRC, consider using sklearn.metrics

    results[name] = {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
```

```
    "Recall": recall,
    "F1 Score": f1,
    "Specificity": specificity,
    "AUPRC": aucpr
  }
results
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 7063, number of negative: 21409
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.044835 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 9020
[LightGBM] [Info] Number of data points in the train set: 28472, number of used features: 174
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.248068 -> initscore=-1.108942
[LightGBM] [Info] Start training from score -1.108942
223/223 [==============================] - 0s 1ms/step
{'Random Forest': {'Accuracy': 0.9237252423093131,
  'Precision': 0.8049853372434017,
  'Recall': 0.9195979899497487,
  'F1 Score': 0.8584831899921814,
  'Specificity': 0.9251126126126126,
  'AUPRC': 0.9223553012811807},
```