# End-to-End Machine Learning Pipeline for Symptom-Based Disease Diagnosis

## I. Introduction

The application of machine learning (ML) in healthcare holds significant promise for improving diagnostic accuracy, predicting patient outcomes, and personalizing treatment strategies.[2] One area of interest is the development of systems capable of suggesting potential disease diagnoses based on patient-reported symptoms. Such systems could potentially aid in early screening, triage, or provide preliminary insights for healthcare professionals, particularly in resource-limited settings or through telemedicine platforms.[4]

This report details the construction of a complete end-to-end machine learning pipeline designed for multi-class disease diagnosis using symptoms as input features. The process begins entirely from scratch, encompassing dataset selection and acquisition from public sources, rigorous data preprocessing tailored for medical symptom data, the development and comparative evaluation of both traditional ML (Random Forest) and Deep Learning (Feedforward Neural Network) models, and finally, the implementation of a standardized project structure for reproducibility and potential deployment. Furthermore, a simple interactive web interface is developed as a proof-of-concept for model interaction.

The objective is not to create a clinically validated diagnostic tool, but rather to demonstrate the systematic process involved in building such a system, highlighting the technical methodologies, evaluation practices, and inherent limitations and ethical considerations associated with using ML for symptom-based diagnosis.[3] The pipeline emphasizes transparency in choices, reproducibility through clear code and structure, and a critical

discussion of the model's capabilities and boundaries.

## II. Dataset Selection and Acquisition

### A. Requirements and Search Strategy

The primary requirement for this project was a dataset containing patient symptoms as input features and corresponding disease diagnoses as multi-class labels. The search focused on publicly accessible, reliable repositories commonly used in the ML community, including Kaggle, the UCI Machine Learning Repository, and resources linked by health organizations like the CDC, NIH, and WHO.[6] Key evaluation criteria for potential datasets included:

1. **Relevance:** Features must primarily consist of symptoms, and labels must represent distinct diseases.
2. **Size:** Sufficient samples and features to train meaningful ML models.
3. **Clarity:** Clear definition of symptoms and disease labels.
4. **Format:** Preferably tabular (e.g., CSV) for ease of use.
5. **Source Credibility:** Preference for datasets with clear origins, ideally linked to clinical studies or reputable institutions, although this proved challenging for readily usable symptom-disease datasets.

### B. Dataset Exploration and Challenges

Several potential datasets were identified:

- **UCI Heart Disease:** Contains clinical attributes, including some symptoms (e.g., chest pain type, exercise-induced angina), but focuses on a single disease outcome (heart disease presence/absence or severity levels).[6] It often requires handling missing values.[11] While valuable, it doesn't fit the multi-class disease diagnosis requirement based on a broad range of

symptoms.

- **MIMIC Datasets (MIMIC-III, MIMIC-IV):** Large, comprehensive, real-world clinical datasets containing de-identified EHR data, including notes, lab results, vital signs, and ICD codes.[20] Extracting structured symptom-disease pairs requires significant NLP preprocessing of clinical notes and mapping codes (e.g., ICD to SNOMED CT).[22] While representing high-fidelity clinical data, the complexity of preprocessing MIMIC for this specific task was deemed beyond the scope of building a foundational pipeline from scratch.[20] Other clinical datasets often have similar challenges or access restrictions.[8]
- **Kaggle Datasets:** Several Kaggle datasets appeared relevant:
  - *Disease-Symptom Dataset (DhivyeshRK)* [27]: Large (246k samples, 773 diseases, 377 symptoms), but explicitly stated as "artificially generated". While preserving some statistical properties, its lack of real-world clinical origin makes it unsuitable for aiming at realistic diagnosis modeling.
  - *Healthcare Dataset (Prasad22)* [1]: Synthetic data generated using Faker, mimicking EHR structure but not actual symptom-disease relationships derived from patients.
  - *Disease Symptoms and Patient Profile Dataset (UOM190346A)* [28]: Contains symptoms (Fever, Cough, Fatigue, Difficulty Breathing) and basic patient info linked to diseases. However, the symptom set is very limited, and the data source/realism is unclear.[29]
  - *Symptom2Disease (Niyar R Barman)* [30]: Contains natural language symptom descriptions mapped to 24 diseases (1200 datapoints). Interesting for NLP approaches but different from the structured symptom input requirement.
  - *Disease Prediction Using Machine Learning (Kaushil268)* [31]:

This dataset consists of two CSV files (Training, Testing) with 132 symptom columns (binary 0/1 indicating absence/presence) and a 'prognosis' column with 41 unique disease labels.[31] It has ~4920 training samples. While its exact origin and real-world validation status are not explicitly documented in the dataset description or associated papers found [35], it is frequently used in public kernels and projects for demonstrating symptom-based disease prediction.[31] Its structure directly matches the project requirements (structured symptoms, multi-class disease labels).

#### C. Selected Dataset

The **"Disease Prediction Using Machine Learning" dataset from Kaggle (by Kaushil268)** [35] was selected for this project.

- **Link:** https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning
- **Files:** Training.csv, Testing.csv
- **Structure:** 133 columns (132 binary symptom features, 1 'prognosis' target variable). 41 unique diseases. 4920 training instances, 42 testing instances (as provided).
- **Rationale:** Despite the ambiguity regarding its real-world provenance, this dataset provides a readily usable structure for demonstrating the end-to-end pipeline. It offers a multi-class classification problem with a significant number of symptom features, suitable for comparing traditional ML and DL models. The pre-structured binary symptom format simplifies the initial preprocessing steps compared to extracting features from raw text or complex EHRs.
- **Caveat:** The limitations regarding the dataset's origin and

potential lack of clinical realism will be explicitly addressed in the Discussion section. The very high accuracies often reported on this dataset [31] might indicate a simplified or overly clean representation compared to real clinical complexity.

#### D. Combining Datasets (Considered but Not Applied)

The initial search considered combining multiple datasets if a single comprehensive one wasn't found. This often involves significant challenges [2]:

1. **Schema Mapping:** Different datasets might use varying names or coding systems for the same symptoms or diseases (e.g., ICD vs. SNOMED CT vs. custom terms).[43] Mapping requires standardized terminologies or ontologies like SNOMED CT or UMLS.[45]
2. **Feature Heterogeneity:** Datasets might capture different sets of symptoms or capture them at different levels of detail (e.g., presence/absence vs. severity vs. duration). Combining requires finding common features or using imputation techniques for missing features across datasets.[42]
3. **Data Harmonization:** Differences in data collection protocols, patient populations, and diagnostic criteria can introduce biases when merging datasets.[41] Common data models (CDMs) like OMOP aim to address this but require significant effort.[43]

Given the availability of the Kaushil268 dataset which, while potentially limited, provided a single, structured source matching the requirements, the complexities of combining disparate datasets were avoided for this foundational project.

## III. Data Preprocessing

Raw data, especially in healthcare, often requires significant cleaning

and transformation before being suitable for machine learning models.[50] The goal of preprocessing is to handle inconsistencies, format data correctly, and prepare features for optimal model performance.[52]

#### A. Initial Data Loading and Inspection

The Training.csv and Testing.csv files were loaded using the pandas library. Initial inspection involved:

- Checking dimensions (rows, columns). Training: 4920 rows, 134 columns (including an unnamed last column). Testing: 42 rows, 133 columns.
- Examining data types (.info()): Most symptom columns are integers (0 or 1), 'prognosis' is object (string).
- Previewing data (.head(), .tail()).
- Checking for missing values (.isnull().sum()).

Python

```
import pandas as pd
import numpy as np

# Load data
train_df = pd.read_csv("data/raw/Training.csv")
test_df = pd.read_csv("data/raw/Testing.csv")

# --- Initial Inspection ---
print("Training Data Info:")
train_df.info()
print("\nTesting Data Info:")
```

```
test_df.info()

print("\nTraining Data Missing Values:")
print(train_df.isnull().sum().sum()) # Check total missing values

print("\nTesting Data Missing Values:")
print(test_df.isnull().sum().sum())

# Check for and drop the extra unnamed column if present in training data
if 'Unnamed: 133' in train_df.columns:
    train_df = train_df.drop('Unnamed: 133', axis=1)
    print("\nDropped 'Unnamed: 133' column from training data.")

print("\nTraining Data Shape:", train_df.shape)
print("Testing Data Shape:", test_df.shape)

print("\nUnique Prognosis Labels (Diseases):")
print(train_df['prognosis'].nunique())
print(train_df['prognosis'].unique())
```

**Observations:**

- The training dataset initially had an extra empty column ('Unnamed: 133') which was dropped.
- No missing values were detected in either the training or testing datasets provided.[33] This simplifies preprocessing significantly but might be unrealistic for real-world clinical data where missing values are common.[15]
- There are 132 symptom columns (features) and 1 'prognosis' column (target).
- There are 41 unique disease labels in the 'prognosis' column.

#### B. Handling Missing Values (Strategy, though not needed here)

Although this dataset had no missing values, handling them is a critical step in most real-world scenarios.[50] Common strategies include:

1. **Deletion:**
    - *Listwise Deletion:* Remove entire rows with any missing value. Simple, but can lead to significant data loss, especially if missingness is widespread.[54]
    - *Pairwise Deletion:* Use available data for specific analyses, leading to varying sample sizes.[55]
    - *Column Deletion:* Remove entire features if they have too many missing values.
2. **Single Imputation:** Replace missing values with a single estimated value.
    - *Mean/Median Imputation:* For numerical data. Simple but reduces variance and distorts correlations.[54]
    - *Mode Imputation:* For categorical data (like symptoms if not binary). Replace with the most frequent category. Can overestimate the mode frequency.[55]
    - *Regression Imputation:* Predict missing values using a regression model based on other features.[54]
3. **Multiple Imputation:** Generate multiple plausible values for each missing entry, creating multiple complete datasets. Analyze each dataset and pool results. Accounts for imputation uncertainty.
    - *Multiple Imputation by Chained Equations (MICE):* An iterative approach where each variable with missing data is modeled conditionally based on the other variables. Flexible for mixed data types.[55] Python implementations exist in

sklearn.impute.IterativeImputer and libraries like miceforest.[60]

For symptom data (often binary or categorical), Mode Imputation or more sophisticated methods like MICE (using logistic regression or classification models internally) would be appropriate choices if missing values were present.[58]

#### C. Symptom Standardization (Discussion)

Real-world clinical text often contains variations in symptom descriptions (e.g., "shortness of breath," "dyspnea," "difficulty breathing"). Standardizing these terms is crucial for consistent feature representation.[45] Techniques include:

- **Manual Mapping:** Creating dictionaries of synonyms. Labor-intensive.
- **Stemming/Lemmatization:** Reducing words to their root form.
- **Medical Ontologies:** Using standardized terminologies like SNOMED CT or UMLS to map free-text terms to unique concept IDs.[45] This provides a structured, hierarchical representation of medical concepts. Tools and libraries exist in Python (e.g., PyMedTermino [63], integrations with NLP libraries [64], or cloud services [65]) to facilitate mapping text to these ontologies.[66]

In this project, the dataset uses pre-defined, consistently named symptom columns (e.g., abdominal_pain, loss_of_appetite). Therefore, explicit symptom standardization using ontologies was not required for this specific dataset format. However, this step would be essential if working with unstructured symptom descriptions or combining data from sources with different terminologies.[43]

#### D. Feature Encoding

Machine learning models require numerical input.[52]

1. **Symptom Features (X):** The 132 symptom columns in this dataset are already in a binary numerical format (0 or 1), representing the absence or presence of a symptom. No further encoding is needed for these features. If symptoms were categorical strings (e.g., "mild," "severe"), techniques like One-Hot Encoding or Label Encoding (if ordinal) would be necessary.[67] One-Hot Encoding is generally preferred for nominal categorical features in linear models and neural networks to avoid imposing artificial order, while tree-based models can sometimes handle label-encoded categoricals directly.[68] pandas.get_dummies or sklearn.preprocessing.OneHotEncoder are common tools.[69]
2. **Disease Labels (y):** The 'prognosis' column contains string labels for 41 diseases. This is a multi-class target variable. It needs to be converted to numerical format. *Label Encoding* is suitable here, assigning a unique integer from 0 to 40 to each disease name. This is compatible with many classifiers and required for loss functions like SparseCategoricalCrossentropy in Keras.[71]

Python

```
from sklearn.preprocessing import LabelEncoder

# Separate features (X) and target (y)
X_train = train_df.drop('prognosis', axis=1)
y_train = train_df['prognosis']
```

```
X_test = test_df.drop('prognosis', axis=1)
y_test = test_df['prognosis']

# Ensure columns are in the same order (should be by default here)
assert all(X_train.columns == X_test.columns)

# Encode the target variable
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test) # Use transform only on test data

# Display mapping
label_mapping = dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_)))
print("\nLabel Encoding Mapping (Disease to Integer):")
# print(label_mapping) # Can be very long, maybe print first few
print(list(label_mapping.items())[:5])
print(f"... and {len(label_mapping)-5} more.")

# Store the list of symptoms (feature names)
symptom_list = X_train.columns.tolist()
```

#### E. Data Splitting (Train/Validation/Test)

To evaluate model generalization and tune hyperparameters, the data is split into training, validation, and testing sets.[73]

- **Training Set:** Used to train the models (fit parameters/weights).
- **Validation Set:** Used during training to tune hyperparameters (e.g., number of trees in RF, layers/neurons in FNN) and for early stopping. Helps prevent overfitting to the training set.[74]
- **Test Set:** Held out completely until the final model is chosen.

Used for the final, unbiased evaluation of the model's performance on unseen data.[73]

The original dataset provided a Training.csv and a Testing.csv. However, the Testing.csv is very small (42 samples). For robust evaluation and hyperparameter tuning, it's better practice to split the larger Training.csv into new training and validation sets, and use the original Testing.csv as the final hold-out test set (or combine them and re-split if the original test set is deemed too small or unrepresentative).

Given the potential for class imbalance in medical datasets (some diseases being much rarer than others) [31], **stratified splitting** is crucial. Stratification ensures that the proportion of each class (disease) in the original dataset is preserved in the training, validation, and test splits.[73] This prevents scenarios where a rare disease might be entirely absent from the validation or test set.

We will split the Training.csv data (X_train, y_train_encoded) into an 80% training set and a 20% validation set, using stratification. The original Testing.csv (X_test, y_test_encoded) will serve as our final test set.

Python

```
from sklearn.model_selection import train_test_split

# Split the original training data into new training and validation sets (80/20 split)
# Stratify ensures proportional representation of diseases in both sets
```

```
X_train_split, X_val, y_train_split, y_val = train_test_split(
    X_train, y_train_encoded,
    test_size=0.2,
    random_state=42, # for reproducibility
    stratify=y_train_encoded # Use encoded labels for stratification
)

print("\nData Split Shapes:")
print("X_train_split:", X_train_split.shape)
print("y_train_split:", y_train_split.shape)
print("X_val:", X_val.shape)
print("y_val:", y_val.shape)
print("X_test (original):", X_test.shape)
print("y_test_encoded (original):", y_test_encoded.shape)

# Final datasets for modeling:
# Training: X_train_split, y_train_split
# Validation: X_val, y_val
# Testing: X_test, y_test_encoded
```

#### F. Data Normalization/Scaling

While tree-based models like Random Forest are generally insensitive to feature scaling, Neural Networks (FNNs) are sensitive to it. Features with larger value ranges can dominate the learning process.[52] Standardizing features (e.g., to zero mean and unit variance) is common practice for NNs.[51]

We will use StandardScaler from scikit-learn. It's crucial to fit the scaler *only* on the training data (X_train_split) and then use it to transform the training, validation, and test sets.[75] This prevents data leakage from the validation/test sets into the training process.

Python

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Fit scaler ONLY on the training data
X_train_scaled = scaler.fit_transform(X_train_split)

# Transform validation and test data using the SAME fitted scaler
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Convert back to DataFrame for consistency (optional, depends on model input needs)
# X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
# X_val_scaled = pd.DataFrame(X_val_scaled, columns=X_val.columns)
# X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

print("\nScaling applied. Example scaled training data mean (should be close to 0):")
print(np.mean(X_train_scaled[:, 0]))
print("Example scaled training data std dev (should be close to 1):")
print(np.std(X_train_scaled[:, 0]))
```

The preprocessed data (split, encoded, and scaled) is now ready for model training. Processed data files should be saved to the data/processed/ directory.

## IV. Model Architecture and Training

Two distinct models will be developed and evaluated: a traditional ensemble ML model (Random Forest) and a Deep Learning model (Feedforward Neural Network).

#### A. Traditional ML Model: Random Forest

Random Forest (RF) is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the mode of the classes output by individual trees.[31] It is generally robust to overfitting (compared to single decision trees), handles high-dimensional data well, and requires less hyperparameter tuning than some other models. It's often a strong baseline for tabular classification tasks.[40]

- **Implementation:** Scikit-learn's RandomForestClassifier.
- **Input:** Non-scaled data (X_train_split, y_train_split) can be used, as RF is not sensitive to feature scaling.
- **Hyperparameters:** Key parameters include n_estimators (number of trees), max_depth (maximum depth of each tree), min_samples_split, min_samples_leaf, and criterion ('gini' or 'entropy').[52] These can be tuned using techniques like GridSearchCV or RandomizedSearchCV on the validation set (X_val, y_val). For this baseline, default or reasonable starting values will be used (e.g., n_estimators=100, random_state=42).
- **Training:** The model is trained using the .fit() method on the training data.

Python

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Initialize Random Forest model
rf_model = RandomForestClassifier(n_estimators=100,
random_state=42, n_jobs=-1) # Using n_jobs=-1 uses all available CPU
cores

# Train the model on the non-scaled training data
print("\nTraining Random Forest model...")
rf_model.fit(X_train_split, y_train_split)
print("Random Forest training complete.")

# Evaluate on validation set (initial check)
y_val_pred_rf = rf_model.predict(X_val)
val_accuracy_rf = accuracy_score(y_val, y_val_pred_rf)
print(f"Random Forest Validation Accuracy: {val_accuracy_rf:.4f}")
```

#### B. Deep Learning Model: Feedforward Neural Network (FNN)

FNNs, also known as Multi-Layer Perceptrons (MLPs), consist of an input layer, one or more hidden layers, and an output layer. Information flows in one direction, from input to output.[80] They can model complex non-linear relationships in data.[82]

- **Implementation:** Keras API within TensorFlow.
- **Input:** Scaled data (X_train_scaled, y_train_split) is required.[51]
- **Architecture Design:**
  - **Input Layer:** Size equal to the number of features (132 symptoms). Implicitly defined by the input_shape in the first Dense layer.
  - **Hidden Layers:** Start with 2-3 hidden layers. The number of neurons per layer is a key hyperparameter. Common rules of

thumb (though highly problem-dependent) suggest starting with a number between the input and output layer sizes, often decreasing in subsequent layers.[83] We'll use a simple architecture like 128 -> 64 neurons.
    - **Activation Functions:** ReLU (relu) is a standard choice for hidden layers due to its efficiency and ability to mitigate vanishing gradients.[83]
    - **Output Layer:** Size equal to the number of classes (41 diseases). Use softmax activation for multi-class classification to output probabilities that sum to 1.[72]
    - **Regularization:** Dropout layers can be added between hidden layers to reduce overfitting by randomly setting a fraction of neuron outputs to zero during training.[83] A rate of 0.2-0.5 is common.
- **Compilation:**
    - **Optimizer:** Adam is a popular and generally effective optimizer.[72]
    - **Loss Function:** Since the labels (y_train_split) are integers (due to LabelEncoder), SparseCategoricalCrossentropy is the appropriate loss function.[71] If labels were one-hot encoded, CategoricalCrossentropy would be used.[71] For multi-label problems (where one instance can have multiple diseases), BinaryCrossentropy with a sigmoid output activation would be used.[87]
    - **Metrics:** Monitor accuracy during training.[90]
- **Training:** Use the .fit() method. Specify training data, validation data (validation_data=(X_val_scaled, y_val)), number of epochs (iterations over the dataset), and batch size. Early stopping can be used to prevent overfitting by monitoring validation loss and stopping training when it no longer improves.

Python

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping

# Define FNN architecture
input_shape = X_train_scaled.shape[1]
num_classes = len(label_encoder.classes_)

fnn_model = keras.Sequential(
    [
        keras.Input(shape=(input_shape,)),
        layers.Dense(128, activation="relu", name="hidden_layer_1"),
        layers.Dropout(0.3), # Add dropout for regularization
        layers.Dense(64, activation="relu", name="hidden_layer_2"),
        layers.Dropout(0.2),
        layers.Dense(num_classes, activation="softmax",
name="output_layer"),
    ]
)

fnn_model.summary()

# Compile the model
fnn_model.compile(
    loss="sparse_categorical_crossentropy", # Use sparse version for integer
```

```
labels
    optimizer="adam",
    metrics=["accuracy"],
)

# Define Early Stopping
early_stopping = EarlyStopping(
    monitor='val_loss', # Monitor validation loss
    patience=10,        # Number of epochs with no improvement after which training will be stopped
    restore_best_weights=True # Restore model weights from the epoch with the best value of the monitored quantity.
)

# Train the model
print("\nTraining Feedforward Neural Network (FNN) model...")
batch_size = 32
epochs = 100 # Set a higher number, early stopping will prevent overfitting

history = fnn_model.fit(
    X_train_scaled,
    y_train_split,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(X_val_scaled, y_val),
    callbacks=[early_stopping], # Add early stopping callback
    verbose=1 # Set to 1 or 2 to see progress, 0 for silent
)
print("FNN training complete.")

# Evaluate on validation set (initial check)
```

```
loss_val, accuracy_val_fnn = fnn_model.evaluate(X_val_scaled, y_val, verbose=0)
print(f"FNN Validation Accuracy: {accuracy_val_fnn:.4f}")
```

## V. Model Evaluation and Comparison

After training, models must be rigorously evaluated on the unseen test set (X_test, y_test_encoded or X_test_scaled, y_test_encoded for FNN) to estimate their real-world performance.

### A. Evaluation Metrics

For multi-class classification, especially in medical domains where class imbalance is common and different types of errors have different consequences, relying solely on accuracy is insufficient.[76] A suite of metrics provides a more comprehensive picture:

1. **Accuracy:** Overall percentage of correct predictions. (TP + TN) / Total. Can be misleading in imbalanced datasets.[76]
2. **Precision:** Of the instances predicted as positive for a class, what fraction actually belong to that class? TP / (TP + FP). High precision means fewer false positives. Crucial when the cost of a false positive is high (e.g., wrongly diagnosing a serious disease).[76]
3. **Recall (Sensitivity):** Of all the actual positive instances of a class, what fraction did the model correctly identify? TP / (TP + FN). High recall means fewer false negatives. Crucial when the cost of a false negative is high (e.g., missing an actual disease).[76]
4. **F1-Score:** The harmonic mean of Precision and Recall (2 * (Precision * Recall) / (Precision + Recall)). Provides a single score balancing both metrics, useful for imbalanced classes.[76]
5. **Confusion Matrix:** A table showing the counts of True Positives (TP), True Negatives (TN), False Positives (FP), and False

Negatives (FN) for each class. Allows detailed analysis of where the model is making errors.[76]

6. **Classification Report (Scikit-learn):** Provides precision, recall, and F1-score for each class, along with macro and weighted averages. Essential for understanding per-class performance.

**B. Evaluation Procedure**

1. Make predictions on the test set using both the trained RF model (using X_test) and the FNN model (using X_test_scaled). For FNN, predictions will be probabilities; convert these to class labels using np.argmax.
2. Calculate the metrics mentioned above using sklearn.metrics functions (accuracy_score, classification_report, confusion_matrix).
3. Compare the performance of the two models based on these metrics.

Python

```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# --- Random Forest Evaluation ---
print("\n--- Evaluating Random Forest on Test Set ---")
y_test_pred_rf = rf_model.predict(X_test)
test_accuracy_rf = accuracy_score(y_test_encoded, y_test_pred_rf)
print(f"Random Forest Test Accuracy: {test_accuracy_rf:.4f}")
print("\nRandom Forest Classification Report:")
```

```
print(classification_report(y_test_encoded, y_test_pred_rf,
target_names=label_encoder.classes_))

# Confusion Matrix for RF
cm_rf = confusion_matrix(y_test_encoded, y_test_pred_rf)
# Plotting requires more space, consider saving or selective display
# disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf,
display_labels=label_encoder.classes_)
# fig, ax = plt.subplots(figsize=(15, 15)) # Adjust size as needed
# disp_rf.plot(ax=ax, xticks_rotation='vertical')
# plt.tight_layout()
# plt.show()
print("\nRandom Forest Confusion Matrix (snippet):")
print(cm_rf[:5, :5]) # Show top-left 5x5 part

# --- FNN Evaluation ---
print("\n--- Evaluating FNN on Test Set ---")
y_test_pred_proba_fnn = fnn_model.predict(X_test_scaled)
y_test_pred_fnn = np.argmax(y_test_pred_proba_fnn, axis=1) # Get
class labels from probabilities

test_accuracy_fnn = accuracy_score(y_test_encoded,
y_test_pred_fnn)
print(f"FNN Test Accuracy: {test_accuracy_fnn:.4f}")
print("\nFNN Classification Report:")
print(classification_report(y_test_encoded, y_test_pred_fnn,
target_names=label_encoder.classes_))

# Confusion Matrix for FNN
cm_fnn = confusion_matrix(y_test_encoded, y_test_pred_fnn)
# Plotting requires more space, consider saving or selective display
# disp_fnn = ConfusionMatrixDisplay(confusion_matrix=cm_fnn,
```

```
display_labels=label_encoder.classes_)
# fig, ax = plt.subplots(figsize=(15, 15)) # Adjust size as needed
# disp_fnn.plot(ax=ax, xticks_rotation='vertical')
# plt.tight_layout()
# plt.show()
print("\nFNN Confusion Matrix (snippet):")
print(cm_fnn[:5, :5]) # Show top-left 5x5 part
```

#### C. Model Comparison and Recommendation

Based on the evaluation metrics (especially weighted F1-score, which accounts for imbalance, and examination of per-class performance in the classification report), a recommendation is made.

- **Observation:** Both models might achieve high accuracy on this specific dataset, as often reported in public kernels.[31] However, minor differences in precision/recall for specific diseases, computational cost during training/inference, and interpretability should be considered. Random Forests are generally faster to train and more interpretable (feature importances can be extracted). FNNs might capture more complex patterns but require careful tuning and scaling.
- **Recommendation:** If both models show similar high performance (e.g., >95% accuracy/F1 on this dataset), the **Random Forest** model might be recommended due to its simplicity, faster training, inherent robustness, and better interpretability. If the FNN demonstrates a significant performance advantage, especially on rarer classes (check classification report), it might be preferred despite its complexity. *For this specific dataset, very high scores (near 100%) are common, potentially favoring simpler models like RF*

*unless specific nuances are better captured by the FNN.*

The chosen best model and the scaler (if used) should be saved for later use (e.g., in the web application).

Python

```
import joblib

# --- Save Models and Preprocessors ---
output_model_dir = "models/"
os.makedirs(output_model_dir, exist_ok=True) # Ensure directory exists

# Save Random Forest model
joblib.dump(rf_model, os.path.join(output_model_dir,
"random_forest_model.pkl"))
print(f"\nSaved Random Forest model to {output_model_dir}")

# Save FNN model (Keras format)
fnn_model.save(os.path.join(output_model_dir, "fnn_model.keras"))
print(f"Saved FNN model to {output_model_dir}")

# Save the Label Encoder
joblib.dump(label_encoder, os.path.join(output_model_dir,
"label_encoder.pkl"))
print(f"Saved Label Encoder to {output_model_dir}")

# Save the Scaler (fitted on training data)
joblib.dump(scaler, os.path.join(output_model_dir, "scaler.pkl"))
```

```
print(f"Saved Scaler to {output_model_dir}")

# Save the list of symptoms (feature order is important)
joblib.dump(symptom_list, os.path.join(output_model_dir,
"symptom_list.pkl"))
print(f"Saved Symptom List to {output_model_dir}")
```

### VI. Project Structure and Reproducibility

A well-organized project structure is essential for maintainability, collaboration, and reproducibility.[94] The following structure is adopted:

```
disease-diagnosis-symptom/
├── data/
│   ├── raw/              # Original Training.csv, Testing.csv
│   └── processed/        # Processed train/val/test splits (e.g.,.csv
or.pkl)
├── notebooks/
│   ├── 1_EDA.ipynb       # Exploratory Data Analysis (optional, can
be integrated)
│   └── 2_Modeling.ipynb  # Contains preprocessing, training,
evaluation steps
├── src/                  # Python source code modules
│   ├── __init__.py
│   ├── preprocess.py     # Functions for loading, cleaning, encoding,
scaling
│   ├── train.py          # Script to train models (loads processed data,
```

trains, saves models)
│   ├── evaluate.py      # Script to evaluate models on test set (loads models, test data)
│   └── utils.py         # Helper functions (e.g., saving/loading objects)
├── models/              # Saved model files and preprocessors
│   ├── random_forest_model.pkl
│   ├── fnn_model.keras
│   ├── label_encoder.pkl
│   ├── scaler.pkl
│   └── symptom_list.pkl
├── app/                 # Folder for Streamlit app (Bonus)
│   ├── app.py           # Streamlit application script
│   └── requirements_app.txt # Dependencies specific to the Streamlit app
├── requirements.txt     # Main project dependencies (pandas, numpy, scikit-learn, tensorflow, etc.)
└── README.md            # Project overview, setup instructions, usage guide

**Directory Explanations [94]:**

- **data/**: Stores all datasets.
  - raw/: Original, immutable data files.
  - processed/: Cleaned, transformed data ready for modeling (train/val/test splits).
- **notebooks/**: Jupyter notebooks for experimentation, visualization, and potentially executing the pipeline steps interactively.
- **src/**: Contains modular Python scripts for core logic.
  - preprocess.py: Functions for data loading, cleaning, encoding, splitting, scaling.

    - train.py: Script to orchestrate model training, loading processed data and saving trained models.
    - evaluate.py: Script to load trained models and evaluate them on the test set, generating metrics.
    - utils.py: Common helper functions used across scripts.
- **models/**: Stores serialized/saved trained models and necessary preprocessing objects (scaler, encoder, feature list).
- **app/**: Contains files for the interactive web application (Bonus).
- **requirements.txt**: Lists all Python package dependencies required to run the project.
- **README.md**: Essential documentation covering project goals, data sources, setup instructions, how to run the code, and results summary.

**Setup and Execution Instructions (Example for README.md):**

1. **Clone Repository:**
   Bash
   ```
   git clone <your-github-repo-url>
   cd disease-diagnosis-symptom
   ```

2. **Create Virtual Environment (Recommended):**
    - Using venv:
      Bash
      ```
      python -m venv venv
      source venv/bin/activate  # On Windows use `venv\Scripts\activate`
      ```

    - Using conda:
      Bash
      ```
      conda create -n disease_diag python=3.9 # Or desired version
      conda activate disease_diag
      ```

3. **Install Dependencies:**
   Bash
   pip install -r requirements.txt

4. **Run Pipeline:**
   - **Option 1 (Using Scripts):**
     - *Preprocess Data:* python src/preprocess.py (This script would load raw data, perform steps, save processed data to data/processed/)
     - *Train Models:* python src/train.py (This script would load processed data, train RF and FNN, save models to models/)
     - *Evaluate Models:* python src/evaluate.py (This script would load models and test data, print evaluation metrics)
   - **Option 2 (Using Notebook):**
     - Open and run the cells sequentially in notebooks/2_Modeling.ipynb. This notebook should perform preprocessing, training, and evaluation.
5. **Run Interactive App (Bonus):**
   Bash
   pip install -r app/requirements_app.txt # Install Streamlit if needed
   streamlit run app/app.py

**Code Availability:**

The complete project, including code, notebooks, and saved model placeholders, should be made available via a downloadable archive or a public GitHub repository. (A placeholder link would be provided here in a real scenario).

## VII. Bonus: Interactive Diagnosis Tool (Streamlit)

To provide a practical interface for interacting with the trained model, a simple web application is developed using Streamlit.[97] Streamlit allows for rapid development of data-centric web apps directly from Python scripts.[99]

### Functionality:

The application allows a user to select symptoms from the list the model was trained on and receive a predicted disease based on the best-performing model (assumed to be Random Forest for this example, but easily adaptable).

1. **Load Artifacts:** The app loads the saved Random Forest model (.pkl), the Label Encoder (.pkl) to decode predictions, the list of all symptoms (.pkl) to create the input interface, and the Scaler (.pkl) if the chosen model requires scaled input (FNN would, RF doesn't strictly need it, but consistency might be desired if switching models).
2. **User Interface:** A multi-select dropdown (st.multiselect) presents all 132 possible symptoms to the user.
3. **Input Processing:**
    - When the user selects symptoms, the app creates a binary feature vector of length 132 (matching X_train_split.shape[1]).
    - This vector is initialized with zeros.
    - For each symptom selected by the user, the corresponding index in the vector is set to 1. This precisely replicates the input format used during training.[97]
    - *(If using the FNN model, this binary vector would then be scaled using the loaded scaler.transform())*.
4. **Prediction:** The processed feature vector is fed into the loaded model's .predict() method.[97]

5. **Output:** The model outputs an integer prediction. The loaded label_encoder.inverse_transform() converts this integer back into the predicted disease name, which is displayed to the user.[101]
6. **Disclaimer:** A clear disclaimer is included, stating that the prediction is based on a limited model and is **not** a substitute for professional medical diagnosis.

**Code (app/app.py):**

Python

```
import streamlit as st
import joblib
import pandas as pd
import numpy as np
import os

# --- Configuration ---
MODEL_DIR = "../models/" # Adjust path relative to app.py location
MODEL_PATH = os.path.join(MODEL_DIR, "random_forest_model.pkl") # Or "fnn_model.keras"
LABEL_ENCODER_PATH = os.path.join(MODEL_DIR, "label_encoder.pkl")
SYMPTOM_LIST_PATH = os.path.join(MODEL_DIR, "symptom_list.pkl")
# SCALER_PATH = os.path.join(MODEL_DIR, "scaler.pkl") # Needed if using FNN

# --- Load Artifacts ---
try:
    model = joblib.load(MODEL_PATH)
```

```
    label_encoder = joblib.load(LABEL_ENCODER_PATH)
    symptom_list = joblib.load(SYMPTOM_LIST_PATH)
    # scaler = joblib.load(SCALER_PATH) # Uncomment if using FNN
except FileNotFoundError:
    st.error("Error: Model or preprocessor files not found. Please ensure models are trained and saved.")
    st.stop()
except Exception as e:
    st.error(f"An error occurred loading files: {e}")
    st.stop()

# --- Streamlit App Interface ---
st.set_page_config(page_title="Symptom-Based Disease Predictor", layout="wide")
st.title("🩺 Symptom-Based Disease Prediction (Demonstration)")
st.markdown("""
**Disclaimer:** This tool provides a potential disease prediction based on a machine learning model trained on symptom data.
It is **not** a substitute for professional medical advice, diagnosis, or treatment. Always seek the advice of your physician or other qualified health provider with any questions you may have regarding a medical condition.
""")

st.sidebar.header("Enter Your Symptoms")

# Use multiselect for symptom input
selected_symptoms = st.sidebar.multiselect(
    "Select symptoms you are experiencing:",
    options=symptom_list,
    # default= # Optional: set default selected symptoms
)
```

```
st.write("### Selected Symptoms:")
if selected_symptoms:
    st.write(selected_symptoms)
else:
    st.write("Please select symptoms from the sidebar.")

# --- Prediction Logic ---
if st.sidebar.button("Predict Disease"):
    if not selected_symptoms:
        st.sidebar.warning("Please select at least one symptom.")
    else:
        # Create the input vector (binary encoding)
        input_vector = np.zeros(len(symptom_list))
        for symptom in selected_symptoms:
            if symptom in symptom_list:
                index = symptom_list.index(symptom)
                input_vector[index] = 1

        # Reshape for the model (expects 2D array)
        input_df = pd.DataFrame([input_vector], columns=symptom_list)

        # --- Scaling (Uncomment if using FNN) ---
        # try:
        #     input_scaled = scaler.transform(input_df)
        #     input_data_for_model = input_scaled
        # except NameError: # If scaler wasn't loaded (e.g., for RF)
        #     st.error("Scaler not loaded. Required for FNN.")
        #     st.stop()
        # except Exception as e:
        #     st.error(f"Error during scaling: {e}")
```

```
        #     st.stop()
        # --- End Scaling ---

        # Use non-scaled data for Random Forest by default
        input_data_for_model = input_df # For RF

        # --- Make Prediction ---
        try:
            # If using Keras FNN model:
            # prediction_proba = model.predict(input_data_for_model)
            # prediction_index = np.argmax(prediction_proba, axis=1)

            # If using Scikit-learn model (like RF):
            prediction_index = model.predict(input_data_for_model)

            # Decode prediction
            predicted_disease =
label_encoder.inverse_transform([prediction_index])

            st.write("---")
            st.subheader("Prediction Result:")
            st.success(f"The model predicts the potential disease as:
**{predicted_disease}**")
            st.markdown("_Please consult a healthcare professional for an
accurate diagnosis._")

        except Exception as e:
            st.error(f"An error occurred during prediction: {e}")

# Add some padding at the bottom
st.markdown("<br><br>", unsafe_allow_html=True)
```

**Running the App:**

Ensure Streamlit is installed (pip install streamlit) and navigate to the project's root directory in the terminal. Run:

Bash

```
streamlit run app/app.py
```

This command starts a local web server and opens the application in a web browser, allowing users to interact with the model.

## VIII. Discussion: Performance, Limitations, and Ethics

While the pipeline successfully builds and evaluates models, it's crucial to contextualize the results and acknowledge the significant limitations and ethical responsibilities involved in developing ML systems for healthcare.

### A. Model Performance Summary

The evaluation on the test set provides an estimate of how the models might perform on unseen data *from the same distribution as the training data*. For the chosen Kaushil268 dataset, both Random Forest and FNN models often achieve very high accuracy and F1-scores (potentially >95%, sometimes near 100% as seen in public examples [33]).

- **Random Forest:** Typically performs strongly, is relatively fast, and robust.
- **FNN:** Can potentially model more complex interactions but

requires careful tuning, data scaling, and more computational resources. Its performance advantage over RF might be marginal on this specific, potentially simplified, dataset.

The final recommendation (e.g., Random Forest) is based on a balance of performance, simplicity, and interpretability for this demonstration project. However, the high scores themselves warrant caution, as they might not reflect performance on more complex, noisy, real-world clinical data.

#### B. Limitations

This project, while demonstrating an end-to-end process, has several significant limitations:

1. **Dataset Limitations:**
    - **Provenance and Realism:** The primary limitation stems from the chosen dataset.[35] Its origin is unclear, and it may be synthetic or heavily simplified.[35] It likely does not capture the true complexity, noise, and variability of real clinical encounters.[44] Using truly representative clinical datasets (like MIMIC [20] or registry data [8]) would require significantly more complex preprocessing and ethical approvals.
    - **Symptom Representation:** The binary presence/absence of symptoms is a major simplification. Real diagnosis considers symptom severity, duration, onset, timing, combinations, and patient narrative.[103]
    - **Scope:** The dataset covers 41 diseases. Real-world diagnosis involves considering thousands of possibilities.
    - **Balance:** The dataset appears relatively balanced across diseases [31], which might not reflect real-world prevalence where some diseases are far rarer than others.[76]
2. **Model Limitations:**

- **Input Scope:** The models rely *solely* on the provided symptom list. They do not incorporate crucial diagnostic information like patient demographics (beyond what might be implicit in symptoms), medical history, family history, lifestyle factors, physical examination findings, laboratory results, or imaging data.[25] This fundamentally limits their diagnostic power compared to a clinician.
- **Correlation vs. Causation:** The models learn correlations between symptom patterns and disease labels in the dataset. They do not understand the underlying biological causal mechanisms.
- **Generalizability:** Performance is validated only on a test set likely drawn from the same limited distribution. Generalization to different patient populations, geographical regions, or healthcare systems is not guaranteed and would require external validation.[41]

3. **Project Scope Limitations:**
   - **No Clinical Validation:** This is a technical demonstration, not a clinically validated tool. Real-world deployment requires rigorous testing in clinical settings, comparison against clinicians, and regulatory approval.[5]
   - **Static Knowledge:** The model is trained on a static dataset and does not incorporate evolving medical knowledge unless retrained.

#### C. Ethical Considerations and Responsible Use

Deploying ML models in healthcare, especially for diagnosis, carries significant ethical responsibilities.[3]

1. **Bias and Fairness:**
   - **Data Bias:** If the training data underrepresents certain

demographic groups (age, gender, ethnicity, socioeconomic status) or specific disease presentations within those groups, the model's predictions may be systematically less accurate for those groups.[5] This can perpetuate or even exacerbate existing health inequities.[105] Historical biases present in healthcare data collection can be learned and amplified by ML models.[105]
    - **Algorithmic Bias:** Model choices and optimization criteria can also introduce bias. Fairness audits and mitigation strategies are necessary but complex.[105]
2. **Accuracy, Safety, and Harm:**
    - **Misdiagnosis Risk:** False positives (predicting a disease when absent) can cause anxiety and lead to unnecessary, costly, or even harmful tests and treatments.[92] False negatives (missing a disease when present) can delay necessary treatment, leading to worse outcomes.[92] The acceptable balance between precision and recall depends heavily on the specific disease and clinical context.[76]
    - **Over-reliance:** Users (patients or clinicians) might place undue trust in the model's output, potentially overriding clinical judgment or delaying proper consultation.[3] The "black box" nature of some complex models can hinder understanding of *why* a prediction was made.
3. **Data Privacy and Security:**
    - While this project used a public dataset, any application involving real patient data must adhere to strict privacy regulations (like HIPAA). Data must be securely stored, processed, and rigorously de-identified.[3] Combining data streams increases re-identification risks.[41]
4. **Transparency and Interpretability:**

    - Understanding how a model arrives at a prediction is crucial for trust and clinical utility. While RF offers feature importances, interpreting complex FNNs remains challenging. Methods for explainable AI (XAI) are an active area of research.
5. **The "Symptom Checker" Dilemma:** Tools like the one demonstrated, if deployed without extreme caution, can be easily misinterpreted as definitive diagnostic aids. They operate on vastly incomplete information compared to a clinical assessment.[25] Clear communication of limitations and intended use (e.g., educational, preliminary information gathering only) is paramount. The potential for harm from inaccurate or misinterpreted predictions generated by simplistic symptom-only models is significant.[104]

**Responsible Development requires:** careful problem selection avoiding bias [5], meticulous data validation and bias assessment [105], transparency about model capabilities and limitations, prioritizing safety and fairness, ensuring privacy, and involving domain experts and diverse stakeholders throughout the development lifecycle.

## IX. Conclusion

This report detailed the end-to-end development of a machine learning pipeline for symptom-based disease prediction. Starting with dataset selection (choosing the Kaushil268 Kaggle dataset for its structure, while acknowledging provenance limitations), the process covered data preprocessing (encoding labels), model training (Random Forest and Feedforward Neural Network), and comprehensive evaluation using metrics appropriate for multi-class classification (accuracy, precision, recall, F1-score, confusion matrix). A standardized project structure was implemented to ensure

reproducibility, and a basic Streamlit application was developed to demonstrate model interaction.

The evaluation indicated that both Random Forest and FNN models could achieve high performance metrics on this particular dataset, with Random Forest potentially being preferable due to its simplicity and interpretability given similar performance levels. Key artifacts, including the trained models, label encoder, scaler, and symptom list, were saved to facilitate deployment and future use.

However, the project's primary value lies in illustrating the *process* rather than the specific diagnostic capability of the resulting models. The discussion highlighted critical limitations, particularly concerning the dataset's potential lack of real-world fidelity and the inherent constraints of relying solely on symptom presence/absence for diagnosis. Furthermore, the ethical considerations surrounding bias, fairness, patient safety, data privacy, and responsible deployment in healthcare AI were emphasized. Symptom-based ML models, while technically feasible, must be approached with extreme caution and should never substitute professional medical evaluation.

Future work could involve:

- Utilizing more complex and clinically validated datasets (e.g., MIMIC, requiring advanced NLP and data integration techniques [20]).
- Incorporating richer features like symptom severity, duration, patient demographics, and medical history.
- Exploring more sophisticated model architectures (e.g., attention mechanisms, graph neural networks for symptom relationships).
- Implementing robust bias detection and mitigation techniques.[105]
- Focusing on model interpretability and explainability (XAI).

- Conducting rigorous external validation and, eventually, clinical trials if intended for real-world use.[5]

In conclusion, while machine learning offers powerful tools for analyzing health data, its application to sensitive tasks like disease diagnosis demands a rigorous, systematic, and ethically conscious approach. This project serves as a foundational demonstration of the technical steps involved, while underscoring the critical importance of data quality, model validation, and responsible innovation in this domain.[3]

**Works cited**


1. Healthcare Dataset - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/datasets/prasad22/healthcare-dataset
2. Machine learning for precision medicine - Canadian Science Publishing, accessed on April 10, 2025, https://cdnsciencepub.com/doi/10.1139/gen-2020-0131
3. Delving into Machine Learning's Influence on Disease Diagnosis and Prediction, accessed on April 10, 2025, https://openpublichealthjournal.com/VOLUME/17/ELOCATOR/e18749445297804/FULLTEXT/
4. Optimized Machine Learning Classifiers for Symptom-Based Disease Screening - MDPI, accessed on April 10, 2025, https://www.mdpi.com/2073-431X/13/9/233
5. Ethical Machine Learning in Healthcare - PMC - PubMed Central, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC8362902/
6. UCI Machine Learning Repository: Home, accessed on April 10, 2025, https://archive.ics.uci.edu/
7. Data Sets | CDC Open Technology, accessed on April 10, 2025, https://open.cdc.gov/data.html
8. Data Resources in the Health Sciences: Clinical Data - Library Guides, accessed on April 10, 2025, https://guides.lib.uw.edu/hsl/data/findclin
9. MedlinePlus - Health Information from the National Library of Medicine, accessed on April 10, 2025, https://medlineplus.gov/
10. UCI Heart Disease Data - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data
11. Heart Disease - UCI Machine Learning Repository, accessed on April 10, 2025, https://archive.ics.uci.edu/dataset/45/heart+disease
12. A novel approach for the effective prediction of cardiovascular disease using applied artificial intelligence techniques - PubMed Central, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11631242/
13. Python Data Cleaning for Heart Disease UCI Dataset - Kaggle, accessed on April

10, 2025, https://www.kaggle.com/code/nhiyen/python-data-cleaning-for-heart-disease-uci-dataset
14. Increasing efficiency of SVMp+ for handling missing values in healthcare prediction - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10309617/
15. Ischemic Heart Disease Multiple Imputation Technique using Machine Learning Algorithm - Engineered Science Publisher, accessed on April 10, 2025, https://www.espublisher.com/uploads/article_pdf/es8d681.pdf
16. Increasing efficiency of SVMp+ for handling missing values in healthcare prediction - NSF Public Access Repository, accessed on April 10, 2025, https://par.nsf.gov/servlets/purl/10433021
17. (PDF) Feature-Limited Prediction on the UCI Heart Disease Dataset - ResearchGate, accessed on April 10, 2025, https://www.researchgate.net/publication/366774648_Feature-Limited_Prediction_on_the_UCI_Heart_Disease_Dataset
18. abroniewski/Heart-Disease-Machine-Learning-Exploration - GitHub, accessed on April 10, 2025, https://github.com/abroniewski/Heart-Disease-Machine-Learning-Exploration
19. Heart Disease Prediction Binary Classification - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/code/abdmental01/heart-disease-prediction-binary-classification/code
20. An Extensive Data Processing Pipeline for MIMIC-IV - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9854277/
21. MIMIC-III Dataset - Papers With Code, accessed on April 10, 2025, https://paperswithcode.com/dataset/mimic-iii
22. International Classification of Diseases Prediction from MIMIIC-III Clinical Text Using Pre-Trained ClinicalBERT and NLP Deep Learning Models Achieving State of the Art - City Research Online, accessed on April 10, 2025, https://openaccess.city.ac.uk/id/eprint/33037/
23. International Classification of Diseases Prediction from MIMIIC-III Clinical Text Using Pre-Trained ClinicalBERT and NLP Deep Learning Models Achieving State of the Art - MDPI, accessed on April 10, 2025, https://www.mdpi.com/2504-2289/8/5/47
24. NIH Clinical Center releases dataset of 32000 CT images, accessed on April 10, 2025, https://www.nih.gov/news-events/news-releases/nih-clinical-center-releases-dataset-32000-ct-images
25. Multi-label classification of biomedical data - Spandidos Publications, accessed on April 10, 2025, https://www.spandidos-publications.com/10.3892/mi.2024.192
26. Deep learning model for multi-classification of infectious diseases from unstructured electronic medical records - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC8848865/
27. Disease-Symptom Dataset - Kaggle, accessed on April 10, 2025,

https://www.kaggle.com/datasets/dhivyeshrk/diseases-and-symptoms-dataset
28. Disease Symptoms and Patient Profile Dataset - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/datasets/uom190346a/disease-symptoms-and-patient-profile-dataset
29. Disease Symptoms and Patient Profile Dataset - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/discussions/general/412790
30. Symptom2Disease - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/datasets/niyarrbarman/symptom2disease
31. Disease Type Prediction Using Machine Learning | by Oğuzhan Aydın - Medium, accessed on April 10, 2025, https://medium.com/@oguzhanaydinDS/disease-type-prediction-using-machine-learning-f510ece046a6
32. Disease Symptom Prediction - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/datasets/itachi9604/disease-symptom-description-dataset
33. Disease Prediction - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/code/annelieseneo/disease-prediction
34. Disease Prediction based on user input using LR - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/code/ahmed321abozeid/disease-prediction-based-on-user-input-using-lr
35. Disease Prediction Using Machine Learning | Kaggle, accessed on April 10, 2025, https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning/data
36. Disease-Prediction-Using-Machine-Learning - GitHub, accessed on April 10, 2025, https://github.com/chonlapatsri/Disease-Prediction-Using-Machine-Learning
37. Anushree-kumar/Disease-Prediction - GitHub, accessed on April 10, 2025, https://github.com/Anushree-kumar/Disease-Prediction
38. anujdutt9/Disease-Prediction-from-Symptoms - GitHub, accessed on April 10, 2025, https://github.com/anujdutt9/Disease-Prediction-from-Symptoms
39. Disease Prediction using Real-Life Data and Machine Learning Algorithms - ResearchGate, accessed on April 10, 2025, https://www.researchgate.net/publication/384493251_Disease_Prediction_using_Real-Life_Data_and_Machine_Learning_Algorithms
40. (PDF) Disease Prediction Using Machine Learning Methods - ResearchGate, accessed on April 10, 2025, https://www.researchgate.net/publication/382964926_Disease_Prediction_Using_Machine_Learning_Methods
41. Merging heterogeneous clinical data to enable knowledge discovery - PubMed Central, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC6447393/
42. Combining datasets to improve model fitting - arXiv, accessed on April 10, 2025, https://arxiv.org/pdf/2210.05165

43. Learning across diverse biomedical data modalities and cohorts: Challenges and opportunities for innovation - PubMed Central, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10873158/
44. Opportunities and challenges in developing deep learning models using electronic health records data: a systematic review - Oxford Academic, accessed on April 10, 2025, https://academic.oup.com/jamia/article/25/10/1419/5035024
45. A review of medical terminology standards and structured reporting - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC6504145/
46. Identifying Symptom Groups from Emergency Department Presenting Complaint Free Text using SNOMED CT - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC3243271/
47. Use of SNOMED CT in Large Language Models: Scoping Review, accessed on April 10, 2025, https://medinform.jmir.org/2024/1/e62924
48. Ontology-Based Clinical Information Extraction Using SNOMED CT - DigitalCommons@TMC, accessed on April 10, 2025, https://digitalcommons.library.tmc.edu/cgi/viewcontent.cgi?article=1042&context=uthshis_dissertations
49. Unified Medical Language System (UMLS) - National Library of Medicine, accessed on April 10, 2025, https://www.nlm.nih.gov/research/umls/index.html
50. Data Preprocessing Techniques for Improving Fever Diagnosis Accuracy - ResearchGate, accessed on April 10, 2025, https://www.researchgate.net/publication/388769097_Data_Preprocessing_Techniques_for_Improving_Fever_Diagnosis_Accuracy
51. Data Preprocessing Techniques for AI and Machine Learning Readiness: Scoping Review of Wearable Sensor Data in Cancer Care, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11470224/
52. Symptom Based Disease Prediction Using Machine Learning Techniques - ijrpr, accessed on April 10, 2025, https://ijrpr.com/uploads/V6ISSUE4/IJRPR41439.pdf
53. SYMPTOM BASED DISEASE PREDICTION USING MACHINE LEARNING AND TKINTER - IRJMETS, accessed on April 10, 2025, https://www.irjmets.com/uploadedfiles/paper//issue_3_march_2024/51408/final/fin_irjmets1711715327.pdf
54. Missing Data in Clinical Research: A Tutorial on Multiple Imputation - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC8499698/
55. Handling Missing Values in Categorical Variables in Medical Implant Data, accessed on April 10, 2025, https://massedcompute.com/faq-answers/?question=What%20are%20the%20common%20techniques%20for%20handling%20missing%20values%20in%20categorical%20variables%20in%20medical%20implant%20data?
56. The Handling of Missing Values in Medical Domains with Respect to Pattern Mining Algorithms - CEUR-WS.org, accessed on April 10, 2025, https://ceur-ws.org/Vol-1492/Paper_38.pdf
57. Imputing Missing Values in Python - Kaggle, accessed on April 10, 2025, https://www.kaggle.com/code/muhammadaammartufail/imputing-missing-values-in-python

58. Two-stage imputation method to handle missing data for categorical response variable, accessed on April 10, 2025, http://www.csam.or.kr/journal/view.html?uid=2082&pn=lastest&vmd=Full
59. amices/mice: Multivariate Imputation by Chained Equations - GitHub, accessed on April 10, 2025, https://github.com/amices/mice
60. MICE imputation - How to predict missing values using machine learning in Python, accessed on April 10, 2025, https://www.machinelearningplus.com/machine-learning/mice-imputation/
61. Multiple imputation by chained equations: what is it and how does it work? - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC3074241/
62. Multiple Imputation with Chained Equations (MICE) - what is it? - Train in Data's Blog, accessed on April 10, 2025, https://www.blog.trainindata.com/multiple-imputation-with-chained-equations-mice-what-is-it/
63. PyMedTermino tutorial - Pythonhosted.org, accessed on April 10, 2025, https://pythonhosted.org/PyMedTermino/tuto_en.html
64. Snomed to UMLS Code Mapping | snomed_umls_mapping | Healthcare NLP 3.1.0, accessed on April 10, 2025, https://nlp.johnsnowlabs.com/2021/07/01/snomed_umls_mapping_en.html
65. Clinical text mining using the Amazon Comprehend Medical new SNOMED CT API - AWS, accessed on April 10, 2025, https://aws.amazon.com/blogs/machine-learning/clinical-text-mining-using-the-amazon-comprehend-medical-new-snomed-ct-api/
66. A Simple Terminology-Based Approach to Clinical Entity Recognition - CEUR-WS, accessed on April 10, 2025, https://ceur-ws.org/Vol-3180/paper-16.pdf
67. Working with categorical data | Machine Learning - Google for Developers, accessed on April 10, 2025, https://developers.google.com/machine-learning/crash-course/categorical-data
68. Categorical Data Encoding Techniques | by Krishnakanth Naik Jarapala | AI Skunks, accessed on April 10, 2025, https://medium.com/aiskunks/categorical-data-encoding-techniques-d6296697a40f
69. Encoding Categorical Features. Introduction | by Yang Liu - Towards Data Science, accessed on April 10, 2025, https://towardsdatascience.com/encoding-categorical-features-21a2651a065c?source=---------4----------
70. Encoding Categorical data in Machine Learning | by Akhil Reddy Mallidi | #ByCodeGarage, accessed on April 10, 2025, https://medium.com/bycodegarage/encoding-categorical-data-in-machine-learning-def03ccfbf40
71. Keras Loss Functions: Everything You Need to Know - Neptune.ai, accessed on April 10, 2025, https://neptune.ai/blog/keras-loss-functions
72. Multi-Class Classification Tutorial with the Keras Deep Learning Library - MachineLearningMastery.com, accessed on April 10, 2025, https://machinelearningmastery.com/multi-class-classification-tutorial-keras-dee

p-learning-library/
73. Trade-off between training and testing ratio in machine learning for medical image processing - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11419616/
74. Training, Validation, Test Split for Machine Learning Datasets - Encord, accessed on April 10, 2025, https://encord.com/blog/train-val-test-split/
75. Practical Implementation of Stratified Cross-Validation in Machine Learning, accessed on April 10, 2025, https://www.numberanalytics.com/blog/practical-implementation-stratified-cross-validation
76. Using a Confusion Matrix to Calculate Precision and Recall - Keylabs, accessed on April 10, 2025, https://keylabs.ai/blog/using-a-confusion-matrix-to-calculate-precision-and-recall/
77. Train-validation-test split for small and unbalanced dataset? - Cross Validated, accessed on April 10, 2025, https://stats.stackexchange.com/questions/647996/train-validation-test-split-for-small-and-unbalanced-dataset
78. Large-scale multi-label text classification - Keras, accessed on April 10, 2025, https://keras.io/examples/nlp/multi_label_classification/
79. Comparative Analysis of Machine Learning Algorithms for Heart Attack Prediction - IJFMR, accessed on April 10, 2025, https://www.ijfmr.com/papers/2024/6/29946.pdf
80. Feedforward neural networks: everything you need to know - CUDO Compute, accessed on April 10, 2025, https://www.cudocompute.com/topics/neural-networks/feedforward-neural-networks-everything-you-need-to-know
81. Medical Dataset Classification: A Machine Learning Paradigm Integrating Particle Swarm Optimization with Extreme Learning Machine Classifier - PMC, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC4605351/
82. Weight Predictor Network with Feature Selection for Small Sample Tabular Biomedical Data, accessed on April 10, 2025, https://ojs.aaai.org/index.php/AAAI/article/view/26090/25862
83. Designing Your Neural Networks. A Step by Step Walkthrough - Medium, accessed on April 10, 2025, https://medium.com/towards-data-science/designing-your-neural-networks-a5e4617027ed
84. How to choose the number of hidden layers and nodes in a feedforward neural network?, accessed on April 10, 2025, https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
85. Loss and Loss Functions for Training Deep Learning Neural Networks - MachineLearningMastery.com, accessed on April 10, 2025, https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

86. A Review on Dropout Regularization Approaches for Deep Neural Networks within the Scholarly Domain - MDPI, accessed on April 10, 2025, https://www.mdpi.com/2079-9292/12/14/3106
87. A Guide to Multi-Label Classification on Keras | dl-question-bank - Wandb, accessed on April 10, 2025, https://wandb.ai/ayush-thakur/dl-question-bank/reports/A-Guide-to-Multi-Label-Classification-on-Keras--VmlldzoyMDgyMDU
88. What loss function and metric should I use for multi-label classification in keras?, accessed on April 10, 2025, https://stackoverflow.com/questions/64951910/what-loss-function-and-metric-should-i-use-for-multi-label-classification-in-ker
89. Which loss function and metrics to use for multi-label classification with very high ratio of negatives to positives? - Stack Overflow, accessed on April 10, 2025, https://stackoverflow.com/questions/59336899/which-loss-function-and-metrics-to-use-for-multi-label-classification-with-very
90. Basic classification: Classify images of clothing | TensorFlow Core, accessed on April 10, 2025, https://www.tensorflow.org/tutorials/keras/classification
91. F1 Score in Machine Learning Explained | Encord, accessed on April 10, 2025, https://encord.com/blog/f1-score-in-machine-learning/
92. Confusion Matrix, Precision, and Recall - Train in Data's Blog, accessed on April 10, 2025, https://www.blog.trainindata.com/confusion-matrix-precision-and-recall/
93. Precision, Recall and F1 Explained [With 10 ML Use case] - Ruman - Medium, accessed on April 10, 2025, https://rumn.medium.com/precision-recall-and-f1-explained-with-10-ml-use-case-6ef2fbe458e5
94. Generic Folder Structure for your Machine Learning Projects. - DEV Community, accessed on April 10, 2025, https://dev.to/luxdevhq/generic-folder-structure-for-your-machine-learning-projects-4coe
95. kylebradbury/ml-project-structure-demo - GitHub, accessed on April 10, 2025, https://github.com/kylebradbury/ml-project-structure-demo
96. ghimiresunil/Machine-Learning-Project-Structure - GitHub, accessed on April 10, 2025, https://github.com/ghimiresunil/Machine-Learning-Project-Structure
97. Deploying Machine Learning Models with Streamlit and Docker: A Comprehensive Guide | by Afouda Josue | Medium, accessed on April 10, 2025, https://medium.com/@afouda.josue/deploying-machine-learning-models-with-streamlit-and-docker-a-comprehensive-guide-30109e76c147
98. Streamlit Tutorial with Machine Learning Project - DEV Community, accessed on April 10, 2025, https://dev.to/jagroop2001/streamlit-the-magic-wand-for-ml-app-creation-43i8
99. How to Quickly Deploy Machine Learning Models with Streamlit - MachineLearningMastery.com, accessed on April 10, 2025, https://machinelearningmastery.com/how-to-quickly-deploy-machine-learning-models-streamlit/
100. Creating & Testing Machine Learning Models using Streamlit | by cem akpolat |

Medium, accessed on April 10, 2025, https://akpolatcem.medium.com/creating-testing-machine-learning-models-using-streamlit-a2d4a628fbf6
101. Accepting User Input for Predictions - Flask Tutorial - Meritshot, accessed on April 10, 2025, https://www.meritshot.com/accepting-user-input-for-predictions/
102. A Guide to Connect Machine Learning Model Backend to Frontend Using Flask, accessed on April 10, 2025, https://dev.to/alfaechonineraitt/a-guide-to-connect-machine-learning-model-backend-to-frontend-using-flask-5geh
103. DISNET: a framework for extracting phenotypic disease information from public sources, accessed on April 10, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC7032061/
104. Detecting Bias and Enhancing Diagnostic Accuracy in Large Language Models for Healthcare - arXiv, accessed on April 10, 2025, https://arxiv.org/html/2410.06566v1
105. Exploring Bias and Prediction Metrics to Characterise the Fairness of Machine Learning for Equity-Centered Public Health Decision-Making: A Narrative Review - arXiv, accessed on April 10, 2025, https://arxiv.org/html/2408.13295v2