Final Project: Heart Disease Severity Detection

Author: Shambhavi Danayak
StudentID: 012654513
Presentation Date: 12/13/2024
Final Submission date: 12/19/2024
Professor: Sam Siewert

In Class Presentation Link (View only):
https://1drv.ms/p/c/a9318f007f6e648c/EV2JXko9SSRNvOA-IkGmH1gBUYcvxtEkywBqk5HecQi3_g

ML MODEL DESCRIPTION

- **Goal and Objective:**

  The goal of this project is to predict the severity of heart disease using patient medical data, minimizing false negatives to ensure that individuals at risk are identified accurately.

- **Why is ANN the best fit?**

  Artificial Neural Network (ANN) is the best fit for this project as it can handle nonlinear relationships between input features and outcomes. Heart disease risk depends on multiple factors(e.g. Age, cholesterol levels, blood pressure and other relevant medical tests results) and ANN's dense layers are designed to learn these complex patterns effectively.

  Additionally, ANN works great with multi-class classification problems, such as our project, where the output is a severity range from 0 to 4. The network's ability and flexibility in learning hierarchical patterns ensures that various correlations between input features are captured for accurate results.

- **Data Description(https://www.kaggle.com/datasets):**

  The dataset includes patient metrics (e.g. blood pressure levels, age, sex, cholesterol etc.). The dataset source is Kaggle. The target feature is called "num" in the dataset whose values range from 0 to 4. This "num " value represents the severity level of heart disease for a given patient.

Column Descriptions:

1. `id` (Unique id for each patient)
2. `age` (Age of the patient in years)
3. `origin` (place of study)
4. `sex` (Male/Female)
5. `cp` chest pain type ([typical angina, atypical angina, non-anginal, asymptomatic])
6. `trestbps` resting blood pressure (resting blood pressure (in mm Hg on admission to the hospital))
7. `chol` (serum cholesterol in mg/dl)
8. `fbs` (if fasting blood sugar > 120 mg/dl)
9. `restecg` (resting electrocardiographic results)
-- Values: [normal, stt abnormality, lv hypertrophy]
10. `thalach` : maximum heart rate achieved
11. `exang` : exercise-induced angina (True/ False)
12. `oldpeak` : ST depression induced by exercise relative to rest
13. `slope` : the slope of the peak exercise ST segment
14. `ca` : number of major vessels (0-3) colored by fluoroscopy
15. `thal` : [normal; fixed defect; reversible defect]
16. `num` : the predicted attribute

[17]:
```
df.head()
```

[17]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2.3	downsloping	0.0	fixed defect	0
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1.5	flat	3.0	normal	2
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2.6	flat	2.0	reversable defect	1
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False	3.5	downsloping	0.0	normal	0
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False	1.4	upsloping	0.0	normal	0

[18]:
```
type(df)
```

[18]: `pandas.core.frame.DataFrame`

[19]:
```
df.shape
```

[19]: `(920, 16)`

[20]:
```
df.tail()
```

[20]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
915	916	54	Female	VA Long Beach	asymptomatic	127.0	333.0	True	st-t abnormality	154.0	False	0.0	NaN	NaN	NaN	1
916	917	62	Male	VA Long Beach	typical angina	NaN	139.0	False	st-t abnormality	NaN	NaN	NaN	NaN	NaN	NaN	0
917	918	55	Male	VA Long Beach	asymptomatic	122.0	223.0	True	st-t abnormality	100.0	False	0.0	NaN	NaN	fixed defect	2
918	919	58	Male	VA Long Beach	asymptomatic	NaN	385.0	True	lv hypertrophy	NaN	NaN	NaN	NaN	NaN	NaN	0
919	920	62	Male	VA Long Beach	atypical angina	120.0	254.0	False	lv hypertrophy	93.0	True	0.0	NaN	NaN	NaN	1

[21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   id        920 non-null    int64  
 1   age       920 non-null    int64  
 2   sex       920 non-null    object 
 3   dataset   920 non-null    object 
 4   cp        920 non-null    object 
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object 
 8   restecg   918 non-null    object 
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object 
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object 
 13  ca        309 non-null    float64
 14  thal      434 non-null    object 
 15  num       920 non-null    int64  
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
```

ML MODEL PROGRAM DESCRIPTION:

MODEL	TRAINING STYLE	DESCRIPTION
Artificial Neural Network (ANN)	Supervised Learning	The ANN predicts heart disease risk by classifying individuals into one of the severity categories based on the medical metrics. It uses labeled training data for supervised classification.

- The model enables classification of patients into five levels of heart disease risk using medical data. The goal is to make these predictions as accurate as possible in order to streamline and help the healthcare system productivity.
- It minimizes the likelihood of false negatives, ensuring the individuals at high risk are not missed. The model achieves this by focusing on maximizing Recall and penalizing false negatives during training.
- The ANN uses two hidden layers (128 and 64 neurons) with ReLU activation function and an output layer with 5 classes (because severity levels are from 0 to 4) and softmax activation function.

MATHEMATICAL METHODS AND ALGORITHMS

MATHEMATICS	**ALGORITHMS**	**FORMULA**	**DESCRIPTION**
Gradient Descent	Adam(adaptive moment estimation)	Equation 11-9. Adam algorithm 1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1-\beta_1)\nabla_\theta J(\theta)$ 2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1-\beta_2)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$ 3. $\widehat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1-{\beta_1}^t}$ 4. $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-{\beta_2}^t}$ 5. $\theta \leftarrow \theta + \eta\, \widehat{\mathbf{m}} \oslash \sqrt{\hat{\mathbf{s}} + \varepsilon}$ In this equation, t represents the iteration number (starting at 1).	The optimizer minimizes the categorical cross-entropy loss function by iteratively updating weights using backpropagation and gradient descent.

Categorical Cross Entropy (CCE)	Loss Function	For a single data point i, the CCE loss is calculated as: $$L_i = -\sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$ • C: The number of classes in the dataset. • y_{ij}: The actual (true) label for class j for the i-th sample (one-hot encoded: $y_{ij} = 1$ for the true class, 0 otherwise). • $\hat{y}_{ij}$: The predicted probability for class j for the i-th sample (output of the softmax layer). For a batch of N samples, the total loss is averaged over the batch: $$L = \frac{1}{N}\sum_{i=1}^{N} L_i = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$	Calculates the difference between predicted probabilities and actual class labels. It ensures the model focuses on accurate class prediction.
ReLU	Activation Function	*The rectified linear unit function: ReLU(z) = max(0, z)* The ReLU function is continuous but unfortunately not differentiable at $z = 0$ (the slope changes abruptly, which can make gradient descent bounce around), and its derivative is 0 for $z < 0$. In practice, however, it works very well and has the advantage of being fast to compute, so it has become the default.[11] Importantly, the fact that it does not have a maximum output value helps reduce some issues during gradient descent (we will come back to this in Chapter 11).	ReLU is applied in the hidden layer (128 and 64 neurons each). This ensures dense layers learn the complex relationships.It also prevents vanishing gradient problem and facilitates backpropagation in deeper networks.
Softmax	Activation Function	**Equation 4-20. Softmax function** $$\hat{p}_k = \sigma\left(\mathbf{s}(\mathbf{x})\right)_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$ In this equation: • K is the number of classes. • $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$. • $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k, given the scores of each class for that instance.	

Standard Scaler	Pre-processing technique to standardize features	The **Standard Scaler** transforms each feature x in the dataset using the following formula: $z = \frac{x - \mu}{\sigma}$ Where: • z: The standardized value. • x: The original value of the feature. • μ: The mean of the feature in the training dataset. • σ: The standard deviation of the feature in the training dataset. This formula centers the data around 0 (mean) and ensures all features have a standard deviati of 1, which standardizes the range of the data.	A preprocessing technique used to standardize features by removing the mean and scaling them to unit variance.
L2 Regularization (weight decay)	Regularization Technique	In L2 regularization, a penalty term is added to the loss function to penalize large weights. The modified loss function becomes: $\mathcal{L}(\theta) = \mathcal{L}_{\text{original}} + \lambda \sum_{i=1}^{n} w_i^2$ Where: • $\mathcal{L}(\theta)$: Total loss with L2 regularization. • $\mathcal{L}_{\text{original}}$: Original loss (e.g., categorical cross-entropy). • λ: Regularization factor (also called weight decay), which controls the strength of the penalty. • w_i: Weights of the model. • n: Number of weights in the model.	Helps manage the issue of overfitting by penalizing large weights and thereby improves generalization.

VALUE OF THE ML MODEL:

➢ **Significance:**

The model predicts heart disease risk, addressing a critical healthcare need. Early detection allows medical professionals to intervene and reduce mortality rates.

➢ **Impact:**

Missing high-risk (false negatives) could lead to severe outcomes. The model emphasizes recall to ensure individuals at risk are not overlooked, balancing precision and accuracy effectively.

➢ **Application:**

This model can be deployed in hospitals, urgent care facilities, Emergency rooms or even clinics to assist doctors in evaluating heart disease risk, especially in case we are dealing with limited resources.

ML MODEL TRAINING AND VALIDATION:

1) **MODEL TRAINING AND VALIDATION** ***(start with preprocessing then training and finally validating)***

- **Preprocessing:**

  This project has two parts: data preprocessing to prepare the dataset for successful ANN predictions and model implementation.

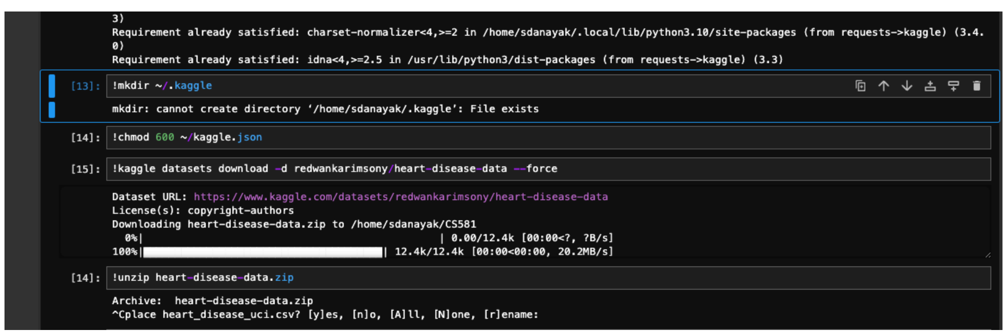
  Data preprocessing (FinalProjectCS581.ipynb)

  This is a crucial step, which will prepare our dataset in accordance with our ANN.

  This step is crucial in the context of this healthcare-focused project, as it requires a thorough understanding of the features, their correlations, and proper handling of missing data to ensure accurate and reliable outcomes.

  1. **Download the dataset:**

     Many ways to download, I connected using Kaggle API Key



```
3)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/sdanayak/.local/lib/python3.10/site-packages (from requests->kaggle) (3.4.
0)
Requirement already satisfied: idna<4,>=2.5 in /usr/lib/python3/dist-packages (from requests->kaggle) (3.3)
[13]: !mkdir ~/.kaggle
mkdir: cannot create directory '/home/sdanayak/.kaggle': File exists
[14]: !chmod 600 ~/kaggle.json
[15]: !kaggle datasets download -d redwankarimsony/heart-disease-data --force
Dataset URL: https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data
License(s): copyright-authors
Downloading heart-disease-data.zip to /home/sdanayak/CS581
  0%|                                              | 0.00/12.4k [00:00<?, ?B/s]
100%|██████████████████████████████████████| 12.4k/12.4k [00:00<00:00, 20.2MB/s]
[14]: !unzip heart-disease-data.zip
Archive:  heart-disease-data.zip
^Cplace heart_disease_uci.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

  2. **Dataset at a glance:**

     Meaning we will see how it looks, what features are there, how many features, any apparent correlations, is the feature relevant to the project, statistics etc.

Data exploration

[17]:
```
df.head()
```

[17]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2.3	downsloping	0.0	fixed defect	0
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1.5	flat	3.0	normal	2
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2.6	flat	2.0	reversable defect	1
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False	3.5	downsloping	0.0	normal	0
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False	1.4	upsloping	0.0	normal	0

[18]:
```
type(df)
```

[18]: pandas.core.frame.DataFrame

[19]:
```
df.shape
```

[19]: (920, 16)

[20]:
```
df.tail()
```

[20]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
915	916	54	Female	VA Long Beach	asymptomatic	127.0	333.0	True	st-t abnormality	154.0	False	0.0	NaN	NaN	NaN	1
916	917	62	Male	VA Long Beach	typical angina	NaN	139.0	False	st-t abnormality	NaN	NaN	NaN	NaN	NaN	NaN	0
917	918	55	Male	VA Long Beach	asymptomatic	122.0	223.0	True	st-t abnormality	100.0	False	0.0	NaN	NaN	fixed defect	2
918	919	58	Male	VA Long Beach	asymptomatic	NaN	385.0	True	lv hypertrophy	NaN	NaN	NaN	NaN	NaN	NaN	0
919	920	62	Male	VA Long Beach	atypical angina	120.0	254.0	False	lv hypertrophy	93.0	True	0.0	NaN	NaN	NaN	1

[21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   dataset   920 non-null    object
 4   cp        920 non-null    object
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object
 8   restecg   918 non-null    object
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object
 13  ca        309 non-null    float64
 14  thal      434 non-null    object
 15  num       920 non-null    int64
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
```

Inference made:

- **Rows(observations):** 920.
- **columns(features):** 16.
- **Target Variable:** num (severity of heart disease).
- **About target variable (num):** has all 920 non-null values that means no missing values, data type=integer.

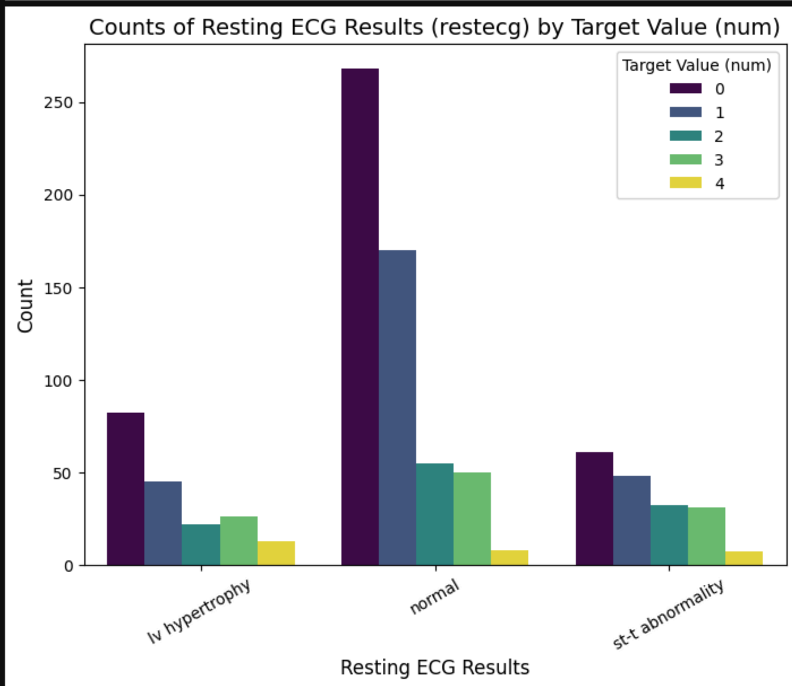
Potential Challenges inferred:

- Class Imbalance num ranges (0-4): Maybe have to use class weights.
- Handling Missing Data
- Conversion of object-type to numerical?
- how to handle features like chol it is in mg/dl like some Scaler, sex from male/female to 0/1?

3) Feature by feature data exploration: Feature by feature exploration is important as at a glance some features might seem irrelevant but they may have correlation with other features making them important for target variable “num” predictions. Eg. Resting blood pressure, sex etc.
Visualizations are also used to understand the relationships.

- no apparent relationship between categories can be seen.

55]:
```
# Count plot
plt.figure(figsize=(8, 6))
sns.countplot(x='restecg', hue='num', data=df, palette='viridis')
plt.title('Counts of Resting ECG Results (restecg) by Target Value (num)', fontsize=14)
plt.xlabel('Resting ECG Results', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.legend(title='Target Value (num)')
plt.xticks(rotation=30)
plt.show()
```



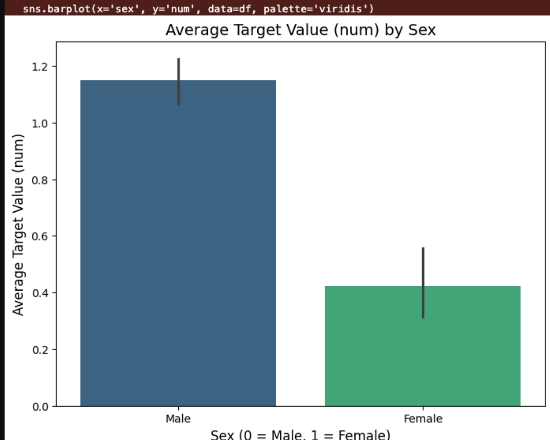
INFERENCE

- for num=0 majority fall into category of normal. As expected from real world clinical observations
- higher num values have a stronger representtaion in the remaining 2 categories

56]:
```
#we can fill the missing values only 2 as normal category
```

[27]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.barplot(x='sex', y='num', data=df, palette='viridis')
plt.title('Average Target Value (num) by Sex', fontsize=14)
plt.xlabel('Sex (0 = Male, 1 = Female)', fontsize=12)
plt.ylabel('Average Target Value (num)', fontsize=12)
plt.xticks([0, 1], ['Male', 'Female'])
plt.show()
```

```
/tmp/ipykernel_2877025/3654145115.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=Fa
lse` for the same effect.

  sns.barplot(x='sex', y='num', data=df, palette='viridis')
```

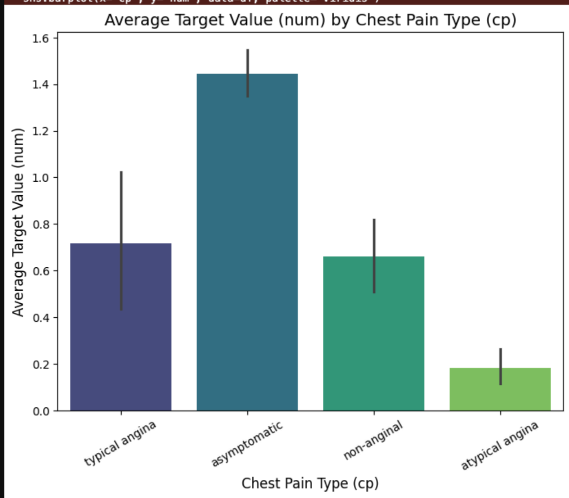


INFERENCE

- Higher average for males, suggesting on average males have higher severity of heart disease.
- Female have lower average num suggesting on average women are less likely to have severe heart disease compared to males.
- Error bar point here to a problem: men have shorter error bar than women indicating that variability in the mean of "num" value is lower than women wth a larger error bar. This could be because men may have more samples in the data than female.

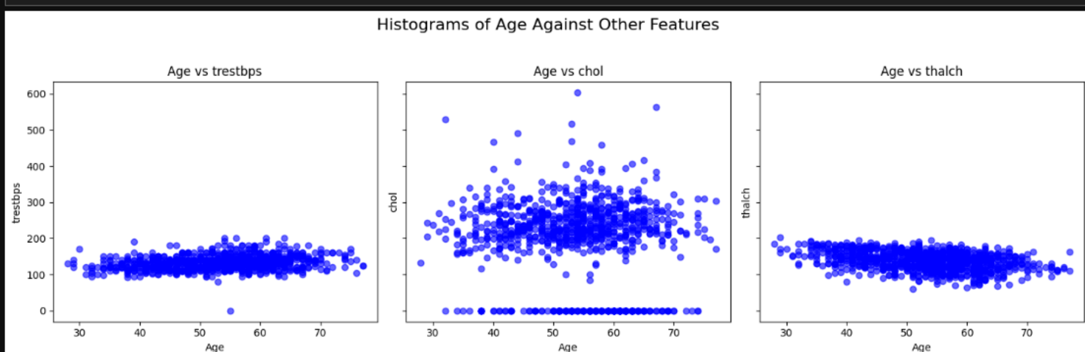
[33]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.barplot(x='cp', y='num', data=df, palette='viridis')
plt.title('Average Target Value (num) by Chest Pain Type (cp)', fontsize=14)
plt.xlabel('Chest Pain Type (cp)', fontsize=12)
plt.ylabel('Average Target Value (num)', fontsize=12)
plt.xticks(rotation=30)
plt.show()
```

```
/tmp/ipykernel_2877025/1449141637.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=Fa
lse` for the same effect.

  sns.barplot(x='cp', y='num', data=df, palette='viridis')
```



"Asymptotic" has the highest average num value suggesting that these patients are more likely to have a sever heart disease.

[23]:
```python
import pandas as pd
import matplotlib.pyplot as plt

features = ['trestbps', 'chol', 'thalch']

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5), sharey=True)
fig.suptitle('Histograms of Age Against Other Features', fontsize=16)

for i, feature in enumerate(features):
    axes[i].scatter(df['age'], df[feature], alpha=0.6, color='b')
    axes[i].set_title(f'Age vs {feature}')
    axes[i].set_xlabel('Age')
    axes[i].set_ylabel(feature)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



INFERENCE:

trestbps relationship with cp

[90]:
```python
# Reshape one-hot encoded columns for visualization
cp_cols = ['cp_asymptomatic', 'cp_atypical angina', 'cp_non-anginal', 'cp_typical angina']
df_melted = df.melt(id_vars=['trestbps'], value_vars=cp_cols,
                    var_name='Chest Pain Type', value_name='Presence')

# Filter for rows where the chest pain type is present
df_present = df_melted[df_melted['Presence'] == 1]

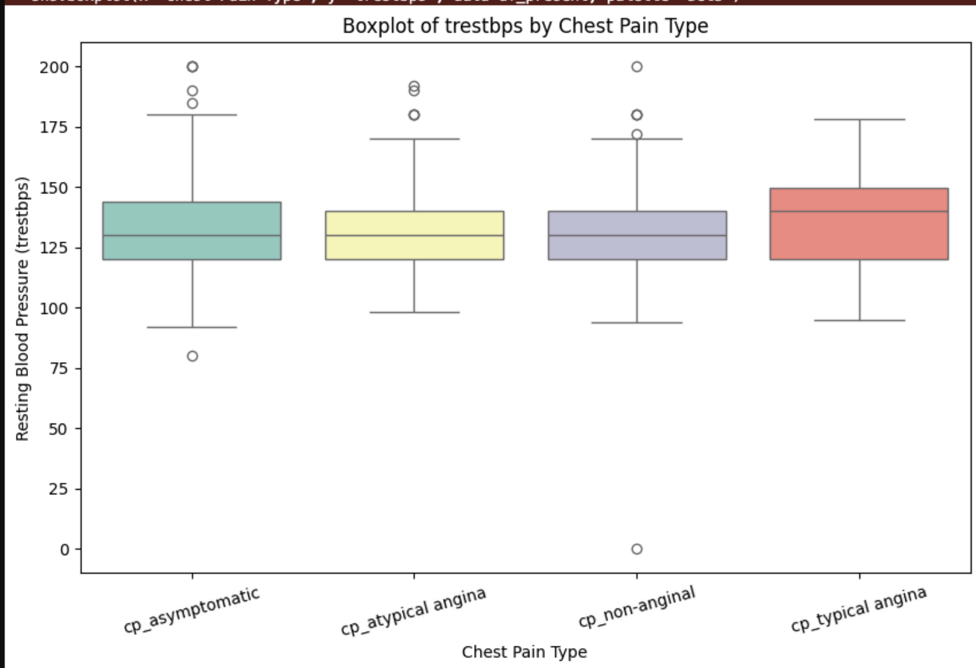
# Boxplot of trestbps for different chest pain types
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.boxplot(x='Chest Pain Type', y='trestbps', data=df_present, palette='Set3')
plt.title('Boxplot of trestbps by Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.ylabel('Resting Blood Pressure (trestbps)')
plt.xticks(rotation=15)
plt.show()
```

```
/tmp/ipykernel_2877025/484248324.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable t
lse` for the same effect.

  sns.boxplot(x='Chest Pain Type', y='trestbps', data=df_present, palette='Set3')
```



After understanding the relationship of features in hand, how it is relevant to our project etc. I handle the issue of missing data. In this dataset from kaggle there is a huge chunk of data missing hence for accurate results that needs to be managed. There are 2 ways i tried impute the missing values:

1. **Create a sub ML model** (linear regression, K-nearest neighbor etc.) based on the features relationship with other features in the dataset and impute the missing values by making predictions.
   Advantage:
   - avoids information loss,
   - enhances model accuracy by reducing data bias,
   - consistent feature scaling and correlation analysis.

   Disadvantages:
   - difficult to implement,
   - Imputed values might inadvertently align with the target variable, causing overfitting
   - can reduce the natural variability in the data, affecting the model's ability to generalize.

2. **Use of statistics:** Impute values using simple statistics such as mode, median etc.
   Advantages:
   - Easy to implement and computationally efficient
   - Median is robust to outliers, ensuring unbiased central value representation
   - Mode and median help maintain the dataset's original distribution.

   Disadvantages:
   - replacing missing values with a central tendency may introduce bias, especially in skewed distributions.
   - It overlooks the relationships between variables, leading to less accurate imputations.
   - Data leak may happen but can be prevented if properly divided the dataset into test, train and validate sets.
3. **Make them a new category as unknown or drop the columns based on relevancy.**

While all 3 ways were tried before making a decision, this project is completed using statistics to impute the values. Only one feature "oldpeak" for understanding purposes is done using a sub ML model (LinearRegression).

```
[133]: from sklearn.model_selection import train_test_split
# Split the data into training and temporary sets
train_df_Predictions_to_fill, temp_df_Predictions_to_fill = train_test_split(df, test_size=0.3, random_state=42)

# Split the temporary set into validation and test sets
val_df_Predictions_to_fill, test_df_Predictions_to_fill = train_test_split(temp_df_Predictions_to_fill, test_size=0.5, random_state=42)

# Verify the sizes
print(f"Training Set: {train_df_Predictions_to_fill.shape}")
print(f"Validation Set: {val_df_Predictions_to_fill.shape}")
print(f"Test Set: {test_df_Predictions_to_fill.shape}")
```

```
Training Set: (644, 19)
Validation Set: (138, 19)
Test Set: (138, 19)
```

Oldpeak

```
[134]: # separate rows with null values in oldpeak
train_not_null_oldpeak = train_df_Predictions_to_fill[train_df_Predictions_to_fill['oldpeak'].notnull()]
train_missing_oldpeak = train_df_Predictions_to_fill[train_df_Predictions_to_fill['oldpeak'].isnull()]
```

```
[135]: from sklearn.linear_model import LinearRegression
predictors = ['age', 'thalch', 'trestbps', 'num', 'chol']
target = 'oldpeak'
```

```
[136]: #train and fit
lr_oldpeak = LinearRegression()
lr_oldpeak.fit(train_not_null_oldpeak[predictors], train_not_null_oldpeak[target])
```

```
[136]: LinearRegression
LinearRegression()
```

```
[137]: #train predict
# Predict and fill missing values for 'oldpeak' in training set
train_df_Predictions_to_fill.loc[train_df_Predictions_to_fill['oldpeak'].isnull(), 'oldpeak'] = lr_oldpeak.predict(train_missing_oldpeak[pre
```

```
[138]: print(train_df_Predictions_to_fill['oldpeak'].isnull().sum())
```

```
0
```

```
[139]: #compare stats to confirm correctness of predicted val
print("Oldpeak Summary (Before Imputation):")
print(train_not_null_oldpeak['oldpeak'].describe())
```

```
Oldpeak Summary (Before Imputation):
count    606.000000
mean       0.890759
std        1.116361
min       -2.600000
25%        0.000000
50%        0.550000
75%        1.500000
max        6.200000
Name: oldpeak, dtype: float64
```

- Predicted values are in alinment with the original dataset.

```
[141]: #fill val set and test set
if val_df_Predictions_to_fill['oldpeak'].isnull().sum() > 0:
    val_missing = val_df_Predictions_to_fill[val_df_Predictions_to_fill['oldpeak'].isnull()]
    val_df_Predictions_to_fill.loc[val_df_Predictions_to_fill['oldpeak'].isnull(), 'oldpeak'] = lr_oldpeak.predict(val_missing[predictors])

# Predict and fill missing values in the test set
if test_df_Predictions_to_fill['oldpeak'].isnull().sum() > 0:
    test_missing = test_df_Predictions_to_fill[test_df_Predictions_to_fill['oldpeak'].isnull()]
    test_df_Predictions_to_fill.loc[test_df_Predictions_to_fill['oldpeak'].isnull(), 'oldpeak'] = lr_oldpeak.predict(test_missing[predictors
```

```
[142]: #combine the filled data with main dataframe
combined_df = pd.concat([train_df_Predictions_to_fill, val_df_Predictions_to_fill, test_df_Predictions_to_fill])
```

```
[143]: combined_df['oldpeak'].isnull().sum()
```

```
[143]: np.int64(0)
```

```
[144]: combined_df.shape
```

```
[144]: (920, 19)
```

ca ca: number of major vessels (0-3) colored by fluoroscopy; missing values

```
[185]: print(modified_df['ca'].dtype) #already float type no need to encode

float64

[187]: #strong relationship with num again
#splitting
train_df, temp_df = train_test_split(modified_df, test_size=0.3, random_state=42)
val_df, test_df = train_test_split(temp_df, test_size=0.5, random_state=42)

[188]: train_df.shape

[188]: (644, 20)

[189]: val_df.shape

[189]: (138, 20)

[190]: #calculate group mode for ca in training data
group_modes = train_df.groupby('num')['ca'].agg(lambda x: x.mode()[0] if not x.mode().empty else np.nan)

[191]: print(group_modes)

num
0    0.0
1    0.0
2    1.0
3    2.0
4    3.0
Name: ca, dtype: float64

[192]: ##apply to sets
def impute_ca(row, group_modes):
    if pd.isnull(row['ca']):
        return group_modes[row['num']]
    return row['ca']

# Step 4: Apply imputation to all sets using the calculated group modes
train_df['ca'] = train_df.apply(lambda row: impute_ca(row, group_modes), axis=1)
val_df['ca'] = val_df.apply(lambda row: impute_ca(row, group_modes), axis=1)
test_df['ca'] = test_df.apply(lambda row: impute_ca(row, group_modes), axis=1)

[200]: train_df['ca'].isnull().sum()

[200]: np.int64(0)

[201]: val_df['ca'].isnull().sum()

[201]: np.int64(0)

[202]: #finally combine
combined_df = pd.concat([train_df, val_df, test_df])

[203]:  combined_df['ca'].isnull().sum()

[203]: np.int64(0)

[204]: combined_df.to_csv('heart_disease_final_modified_filled_with_oldpeak_slope_ca.csv', index=False)
```

Also techniques like label- encoder and one-hot encoding are used to convert object data type to numerical values depending upon its relationship with target variable as well as with other features.

```
[268]: #one hot encoding
thal_one_hot = pd.get_dummies(modified_df['thal'], prefix='thal', drop_first=False)
thal_one_hot = thal_one_hot.astype(int)

[269]: # Combine one-hot encoded columns with the original dataset
modified_df = pd.concat([modified_df, thal_one_hot], axis=1)
# Drop the original 'thal' column (optional)
modified_df.drop(columns=['thal'], inplace=True)
modified_df.head()
modified_df.info()
```

```
Name: slope, dtype: object

[173]: from sklearn.model_selection import train_test_split
       from sklearn.preprocessing import LabelEncoder

       # Encode using label because data has a ordinal relationship
       label_encoder = LabelEncoder()
       modified_df['slope'] = label_encoder.fit_transform(modified_df['slope'].astype(str))
       print("Encoded slope values:", modified_df['slope'].unique())

       Encoded slope values: [2 1 3 0]
```

Final modified dataset:

```
[180]: combined_df.info()

       <class 'pandas.core.frame.DataFrame'>
       Index: 920 entries, 363 to 428
       Data columns (total 20 columns):
        #   Column                    Non-Null Count  Dtype
       ---  ------                    --------------  -----
        0   age                       920 non-null    int64
        1   sex                       920 non-null    int64
        2   trestbps                  920 non-null    float64
        3   chol                      920 non-null    float64
        4   fbs                       920 non-null    int64
        5   thalch                    920 non-null    float64
        6   exang                     920 non-null    int64
        7   oldpeak                   920 non-null    float64
        8   slope                     920 non-null    int64
        9   ca                        309 non-null    float64
        10  thal                      434 non-null    object
        11  num                       920 non-null    int64
        12  cp_asymptomatic           920 non-null    int64
        13  cp_atypical angina        920 non-null    int64
        14  cp_non-anginal            920 non-null    int64
        15  cp_typical angina         920 non-null    int64
        16  restecg_lv hypertrophy    920 non-null    int64
        17  restecg_normal            920 non-null    int64
        18  restecg_st-t abnormality  920 non-null    int64
        19  slope_encoded             920 non-null    int64
       dtypes: float64(5), int64(14), object(1)
       memory usage: 150.9+ KB
```

- **Model Definition:**

  Model is implemented using the final modified dataset generated after the preprocessing step “heart_disease_For_ANN.csv'” **.**

  Below are the steps explaining code:

  1. **One-hot encoding to transform target column "num"** into a format that accurately represents independent classes for Multi Class-Classification num(0-4).

```
[11]: # separating target val i.e. num
      X = df.drop(columns=['num'])  #feature columns
      y = df['num'] #contains target column i.e. ANN will predict
```

We need to use one-hot encoding to tranform our target colulmun "num" into a format that accurately represents independent classes for Multi Class-Classification num(0-4).

Why do we need to do this?

- each severity category in num (0,1,2,3,4) has no ordinal relationship. If these levels are passed as is ANN will treat the integer labels (0,1,2,3,4) as continuos classes and therefore model accuracy will be impacted.
- One-hot encoding ensures the neural network understands that each class is distinct and avoids making false assumptions about relationships between classes.
    - Example: One-hot encoding represents each class as a separate vector where only one index is 1, rest 0
        - num=0:[1,0,0,0,0]
        - num=1:[0,1,0,0,0]
        - num=4:[0,0,0,0,1]

```
[12]: print(y.unique())
      [0 3 1 2 4]

[13]: y_encoded = tf.keras.utils.to_categorical(y, num_classes=5)

[14]: print(np.unique(y_encoded, axis=0))
      [[0. 0. 0. 0. 1.]
       [0. 0. 0. 1. 0.]
       [0. 0. 1. 0. 0.]
       [0. 1. 0. 0. 0.]
       [1. 0. 0. 0. 0.]]
```

2. Split the dataset into test, train and validate set

Now splitting the data into test, train and validate sets

```
[15]: X_train, X_temp, y_train, y_temp = train_test_split(X, y_encoded, test_size=0.3, random_state=42, stratify=y)
      X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=np.argmax(y_temp, axis=1))

[16]: X_train.shape
[16]: (644, 18)

[17]: y_train.shape
[17]: (644, 5)

[18]: X_val.shape
[18]: (138, 18)

[19]: X_test.shape
[19]: (138, 18)
```

3. “StandardScaler”

A preprocessing technique used to standardize features by removing the mean and scaling them to unit variance. It is part of the sklearn.preprocessing module in Python's Scikit-learn library.

- **Result of scaling:** The mean of each feature becomes 0, The standard deviation of each feature becomes 1.
- **Benefits of using Scaler for ANN:** optimization algorithms (e.g., Gradient Descent) to minimize the error during training. It performs better with similar scales.
- Features with larger ranges or magnitudes can dominate the training process, leading the ANN to give them more importance. Scaling removes this bias by making all features comparable.

- **Benefits of using Scaler for ANN:**
- optimization algorithms (e.g., Gradient Descent) to minimize the error during training. It performs better with similar scales.
- Features with larger ranges or magnitudes can dominate the training process, leading the ANN to give them more importance. Scaling removes this bias by making all features comparable.

[20]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

[21]:
```
#Clear any previous session
tf.keras.backend.clear_session()
tf.random.set_seed(42)
```

4. Clear and set Random seed:

[21]:
```
#Clear any previous session
tf.keras.backend.clear_session()
tf.random.set_seed(42)
```


Why clear and set random seed?

.clear_session() resets TensorFlow backend state and clears any existing computation graphs, layers or models. Tensorflow maintains global state of computation graphs, layers etc. if ran multiple times could lead to memory leaks, error "layer already exists" etc.

5. Model Definition:

[22]:
```
from tensorflow.keras.regularizers import l2
```

[28]:
```
# Step 2: Create the model using the Sequential API
model = tf.keras.Sequential([
    # Input layer
    tf.keras.layers.Input(shape=(X_train_scaled.shape[1],)),

    # Hidden layers
    tf.keras.layers.Dense(128, activation="relu",kernel_regularizer=l2(0.01)),
    tf.keras.layers.Dense(64, activation="relu" ,kernel_regularizer=l2(0.01)),

    # Output layer
    tf.keras.layers.Dense(5, activation="softmax")  # 5 classes for multi-class classification
])
```

EXPLANATION:

- **model = tf.keras.Sequential():** used to define a neural network as a linear stack of layers.
- ```
  # Input layer
  tf.keras.layers.Input(shape=(X_train_scaled.shape[1],)),
  ```

    - ➔ It initializes the input pipeline and ensures the model is compatible with the input data dimensions.
    - ➔ Specifies the shape of the input data to the model.
    - ➔ X_train_scaled.shape[1] represents the number of features in the input data, 18 in our case.
  ```

➔ X_train_scaled.shape[1] is set to 1 indicating the model to dynamically extract the number of features from the training dataset.

- First Hidden Layer:

```
 tf.keras.layers.Dense(128,
activation="relu",kernel_regularizer=l2(0.01)),
```

➔ Fully connected (dense) layer where every neuron in this layer is connected to every neuron in the previous layer.

➔ 128 neurons in this layer. The higher the number of neurons, more complex data can be captured.

➔ activation= "relu" Rectified Linear Unit. Prevents vanishing gradients and allows the network to learn complex relationships efficiently.

The rectified linear unit function: ReLU(z) = max(0, z)

The ReLU function is continuous but unfortunately not differentiable at $z = 0$ (the slope changes abruptly, which can make gradient descent bounce around), and its derivative is 0 for $z < 0$. In practice, however, it works very well and has the advantage of being fast to compute, so it has become the default.[11] Importantly, the fact that it does not have a maximum output value helps reduce some issues during gradient descent (we will come back to this in Chapter 11).

➔ L2 Regularization: Adds a penalty proportional to the square of the weights during training.

In L2 regularization, a penalty term is added to the loss function to penalize large weights. The modified loss function becomes:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{original}} + \lambda \sum_{i=1}^{n} w_i^2$$

Where:

- $\mathcal{L}(\theta)$: Total loss with L2 regularization.
- $\mathcal{L}_{\text{original}}$: Original loss (e.g., categorical cross-entropy).
- λ: Regularization factor (also called weight decay), which controls the strength of the penalty.
- w_i: Weights of the model.
- n: Number of weights in the model.

- Second Hidden Layer:

```
tf.keras.layers.Dense(64,
activation="relu"
,kernel_regularizer=l2(0.01)),
```

  - ➔ 64 number of neurons for this layer. It is reduced compared to the first layer to gradually decrease the model's complexity.
  - ➔ Activation function used is still ReLU
  - ➔ L2 Regularization is also the same as layer one.
  - ➔ This layer further refines the learned features by learning intermediate representations between input features and the final output.

- Output Layer:

```
# Output layer
tf.keras.layers.Dense(5, activation="softmax")
```

➔ 5 number of output neurons. This is for the number of classes in this multi class classification problem (severity levels 0,1,2,3,4).

➔ Activation function used:
Softmax activation converts scores into probabilities. The sum of these probabilities is 1 across the 5 classes.

Equation 4-20. Softmax function

$$\hat{p}_k = \sigma\left(\mathbf{s}(\mathbf{x})\right)_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

In this equation:

- K is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k, given the scores of each class for that instance.

6. MODEL TRAINING

```
[29]: # configuring model for training
model.compile(
    # OPTIMIZER:  How the model updates weights during training.
    #Adam aka Adaptive Moment Estimation.
    # Why adam? helps adapts the learning rate for each parameter dynamically, Works well with a wide range of architectures and tasks
    optimizer=tf.keras.optimizers.Adam(),

    #LOSS Function: Categorical Crossentropy Loss since we want multiclass classification
    # since our target labels are one-hot encoded
    # formula: Loss= - summation of (true label for class i * log(predicted probabilities for class i)from i=1 to c
    # The loss is averaged over all training samples in a batch.
    loss=tf.keras.losses.CategoricalCrossentropy(),

    # metric is set as accuracy to measure the fraction of correctly predicted samples.
    # In multi-class classification, accuracy is calculated by comparing the predicted class (highest probability) with the true class.
    #Formula: Accuracy= 1/N * summation 1(argmax(predicted values)= argmax(index of true label)) fromi-1 to N where N is num samples
    metrics=['accuracy']
)
```

```
[30]: model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               2432

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 5)                 325

=================================================================
Total params: 11,013
Trainable params: 11,013
Non-trainable params: 0
```

Now we fit the model with first 50 epochs

```
[32]: #model.fit to start the training
#Returns a History object that stores information about training and validation loss, accuracy, and other metrics for each epoch.
history = model.fit(
    X_train_scaled, y_train,  # input features and corresponding target labels

    # a subset of data used to evaluate the model's performanceduring training but not used to adjust the weights
    validation_data=(X_val_scaled, y_val),
    epochs=50,  # number of full passes through the training dataset during training.

    #num samples processed, 32 is commonly used because it provides a good balance b/w convergence speed and computional efficiancy
    batch_size=32,

    #Controls the verbosity of the training output, helps monitor training.
    #1= output progress bar, 0silent mode, 2= outpts metrics for each progress
    verbose=1
)
```

```
Epoch 1/50
21/21 [==============================] - 1s 18ms/step - loss: 2.3669 - accuracy: 0.4860 - val_loss: 2.1291 - val_accuracy: 0.5652
Epoch 2/50
21/21 [==============================] - 0s 4ms/step - loss: 1.9832 - accuracy: 0.5745 - val_loss: 1.8926 - val_accuracy: 0.6304
Epoch 3/50
21/21 [==============================] - 0s 4ms/step - loss: 1.7629 - accuracy: 0.6522 - val_loss: 1.7433 - val_accuracy: 0.6304
Epoch 4/50
21/21 [==============================] - 0s 4ms/step - loss: 1.6044 - accuracy: 0.6724 - val_loss: 1.6392 - val_accuracy: 0.6087
Epoch 5/50
21/21 [==============================] - 0s 4ms/step - loss: 1.4765 - accuracy: 0.7065 - val_loss: 1.5540 - val_accuracy: 0.6449
Epoch 6/50
21/21 [==============================] - 0s 6ms/step - loss: 1.3788 - accuracy: 0.7283 - val_loss: 1.4857 - val_accuracy: 0.6449
Epoch 7/50
21/21 [==============================] - 0s 5ms/step - loss: 1.2926 - accuracy: 0.7578 - val_loss: 1.4242 - val_accuracy: 0.6377
Epoch 8/50
21/21 [==============================] - 0s 7ms/step - loss: 1.2251 - accuracy: 0.7593 - val_loss: 1.3652 - val_accuracy: 0.6449
Epoch 9/50
21/21 [==============================] - 0s 7ms/step - loss: 1.1650 - accuracy: 0.7578 - val_loss: 1.3159 - val_accuracy: 0.6522
Epoch 10/50
21/21 [==============================] - 0s 7ms/step - loss: 1.1230 - accuracy: 0.7671 - val_loss: 1.2706 - val_accuracy: 0.6667
Epoch 11/50
21/21 [==============================] - 0s 7ms/step - loss: 1.0766 - accuracy: 0.7811 - val_loss: 1.2391 - val_accuracy: 0.6884
Epoch 12/50
21/21 [==============================] - 0s 11ms/step - loss: 1.0372 - accuracy: 0.7640 - val_loss: 1.2069 - val_accuracy: 0.6812
Epoch 13/50
21/21 [==============================] - 0s 11ms/step - loss: 1.0016 - accuracy: 0.7873 - val_loss: 1.1811 - val_accuracy: 0.6667
Epoch 14/50
21/21 [==============================] - 0s 11ms/step - loss: 0.9727 - accuracy: 0.7702 - val_loss: 1.1638 - val_accuracy: 0.6884
Epoch 15/50
21/21 [==============================] - 0s 12ms/step - loss: 0.9405 - accuracy: 0.7904 - val_loss: 1.1392 - val_accuracy: 0.6739
Epoch 16/50
21/21 [==============================] - 0s 11ms/step - loss: 0.9214 - accuracy: 0.7873 - val_loss: 1.1215 - val_accuracy: 0.6957
Epoch 17/50
21/21 [==============================] - 0s 10ms/step - loss: 0.8995 - accuracy: 0.7842 - val_loss: 1.1056 - val_accuracy: 0.6884
Epoch 18/50
21/21 [==============================] - 0s 13ms/step - loss: 0.8802 - accuracy: 0.7842 - val_loss: 1.0961 - val_accuracy: 0.6957
Epoch 19/50
21/21 [==============================] - 0s 12ms/step - loss: 0.8583 - accuracy: 0.7888 - val_loss: 1.0919 - val_accuracy: 0.6884
```

```
21/21 [==============================] - 0s 9ms/step - loss: 0.6735 - accuracy: 0.8137 - val_loss: 0.9977 - val_accuracy: 0.7246
Epoch 42/50
21/21 [==============================] - 0s 11ms/step - loss: 0.6765 - accuracy: 0.8106 - val_loss: 0.9994 - val_accuracy: 0.7174
Epoch 43/50
21/21 [==============================] - 0s 5ms/step - loss: 0.6624 - accuracy: 0.8261 - val_loss: 1.0040 - val_accuracy: 0.6957
Epoch 44/50
21/21 [==============================] - 0s 8ms/step - loss: 0.6738 - accuracy: 0.8292 - val_loss: 1.0007 - val_accuracy: 0.6957
Epoch 45/50
21/21 [==============================] - 0s 7ms/step - loss: 0.6618 - accuracy: 0.8245 - val_loss: 1.0168 - val_accuracy: 0.6884
Epoch 46/50
21/21 [==============================] - 0s 9ms/step - loss: 0.6535 - accuracy: 0.8261 - val_loss: 1.0020 - val_accuracy: 0.6957
Epoch 47/50
21/21 [==============================] - 0s 8ms/step - loss: 0.6501 - accuracy: 0.8230 - val_loss: 1.0016 - val_accuracy: 0.7029
Epoch 48/50
21/21 [==============================] - 0s 7ms/step - loss: 0.6440 - accuracy: 0.8276 - val_loss: 1.0058 - val_accuracy: 0.6957
Epoch 49/50
21/21 [==============================] - 0s 8ms/step - loss: 0.6417 - accuracy: 0.8385 - val_loss: 1.0131 - val_accuracy: 0.7029
Epoch 50/50
21/21 [==============================] - 0s 7ms/step - loss: 0.6393 - accuracy: 0.8416 - val_loss: 1.0064 - val_accuracy: 0.7101
```

```
[33]: test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test)
      print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
5/5 [==============================] - 0s 3ms/step - loss: 0.9498 - accuracy: 0.6594
Test Accuracy: 0.66
```

➔ Adam optimizer

Stands for adaptive moment estimation. It combines momentum and adaptive learning rates for faster training.

Adam

Adam,[20] which stands for *adaptive moment estimation*, combines the ideas of momentum optimization and RMSProp: just like momentum optimization, it keeps track of an exponentially decaying average of past gradients; and just like RMSProp, it keeps track of an exponentially decaying average of past squared gradients (see Equation 11-9). These are estimations of the mean and (uncentered) variance of the gradients. The mean is often called the *first moment* while the variance is often called the *second moment*, hence the name of the algorithm.

Equation 11-9. Adam algorithm

$$
\begin{aligned}
&1. \quad \mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_\theta J(\theta) \\
&2. \quad \mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta) \\
&3. \quad \widehat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1 - {\beta_1}^t} \\
&4. \quad \hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - {\beta_2}^t} \\
&5. \quad \theta \leftarrow \theta + \eta \, \widehat{\mathbf{m}} \oslash \sqrt{\hat{\mathbf{s}} + \varepsilon}
\end{aligned}
$$

In this equation, t represents the iteration number (starting at 1).

➔ Categorical Crossentropy:

Loss function that measures the difference between predicted probabilities and the true labels. This ensures the model predicts probabilities closer to 1 for the correct class.

For a single data point i, the CCE loss is calculated as:

$$L_i = -\sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$

- C: The number of classes in the dataset.
- y_{ij}: The actual (true) label for class j for the i-th sample (one-hot encoded: $y_{ij} = 1$ for the true class, 0 otherwise).
- $\hat{y}_{ij}$: The predicted probability for class j for the i-th sample (output of the softmax layer).

For a batch of N samples, the total loss is averaged over the batch:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$

➔ **metrics= “accuracy”**

Measures how many predictions the model got correct.

Accuracy= Number of Correct Predictions ⁄ Total Predictions

➔ **model.summary()**

Provides overview of the model architecture, number of layers, type of layers etc.

MODEL TESTING FOR GENERALIZATION

1. Test Set Definition and Performance Metrics

```
[15]: X_train, X_temp, y_train, y_temp = train_test_split(X, y_encoded, test_size=0.3, random_state=42, stratify=y)
      X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=np.argmax(y_temp, axis=1))

[16]: X_train.shape
[16]: (644, 18)

[17]: y_train.shape
[17]: (644, 5)

[18]: X_val.shape
[18]: (138, 18)

[19]: X_test.shape
[19]: (138, 18)
```

➔ **Training Set:** 70% of the original dataset
Validation Set: 15% of the original dataset
Test Set: 15% of the original

2. INFERENCE FROM RAN EPOCHS:

- ➔ The model performs well on the training data, as indicated by high training accuracy and low loss but gap between training accuracy and validation accuracy suggests overfitting issue.
- ➔ The validation metrics (accuracy and loss) indicate reasonable generalization, but further tuning may be required to improve validation accuracy and reduce overfitting.

Some steps taken to overcome the issue of overfitting:

- ➔ **Regularization:** L2 regularization to the model to combat overfitting.
- ➔ **Hyperparameter Tuning:** different learning rates, batch sizes, and architectures to improve validation accuracy.
- ➔ Increase the size or diversity of the training dataset to improve generalization.

Trying Early stopping

[34]:
```python
#import lib
from tensorflow.keras.callbacks import EarlyStopping
```

[35]:
```python
#Define Early stopping
# metric to monitor: val_loss, val_accuracy etc.
#set patience: parameter to define how many epochs to wait for improvement before stopping.
#restore_best_weights: parameter to revert to the model state with the best performance.

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

[37]:
```python
#define model for training
history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=50,
    batch_size=32,
    verbose=1,
    callbacks=[early_stopping]  # Added the EarlyStopping callback
)
```

```
Epoch 1/50
21/21 [==============================] - 0s 8ms/step - loss: 0.6332 - accuracy: 0.8370 - val_loss: 1.0100 - val_accuracy: 0.7246
Epoch 2/50
21/21 [==============================] - 0s 5ms/step - loss: 0.6272 - accuracy: 0.8354 - val_loss: 1.0090 - val_accuracy: 0.6884
Epoch 3/50
21/21 [==============================] - 0s 4ms/step - loss: 0.6188 - accuracy: 0.8525 - val_loss: 1.0235 - val_accuracy: 0.7029
Epoch 4/50
21/21 [==============================] - 0s 4ms/step - loss: 0.6168 - accuracy: 0.8432 - val_loss: 1.0049 - val_accuracy: 0.6957
Epoch 5/50
21/21 [==============================] - 0s 4ms/step - loss: 0.6158 - accuracy: 0.8401 - val_loss: 1.0221 - val_accuracy: 0.6957
Epoch 6/50
21/21 [==============================] - 0s 4ms/step - loss: 0.6140 - accuracy: 0.8478 - val_loss: 1.0286 - val_accuracy: 0.6812
Epoch 7/50
21/21 [==============================] - 0s 5ms/step - loss: 0.6062 - accuracy: 0.8478 - val_loss: 1.0207 - val_accuracy: 0.6957
Epoch 8/50
21/21 [==============================] - 0s 5ms/step - loss: 0.6033 - accuracy: 0.8540 - val_loss: 1.0204 - val_accuracy: 0.6957
Epoch 9/50
21/21 [==============================] - 0s 6ms/step - loss: 0.6016 - accuracy: 0.8509 - val_loss: 1.0179 - val_accuracy: 0.6957
```

[38]:
```python
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
5/5 [==============================] - 0s 5ms/step - loss: 0.9395 - accuracy: 0.6739
Test Accuracy: 0.67
```

even worse results- let's try increasing patience parameter, and monitoring metric

[56]:
```
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=15,
    restore_best_weights=True
)
```

[57]:
```
history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=50,
    batch_size=32,
    verbose=1,
    callbacks=[early_stopping]  # Added the EarlyStopping callback
)
```

```
Epoch 1/50
21/21 [==============================] - 0s 6ms/step - loss: 0.1621 - accuracy: 0.9612 - val_loss: 1.2554 - val_accuracy: 0.6739
Epoch 2/50
21/21 [==============================] - 0s 3ms/step - loss: 0.1648 - accuracy: 0.9643 - val_loss: 1.2757 - val_accuracy: 0.6667
Epoch 3/50
21/21 [==============================] - 0s 3ms/step - loss: 0.1486 - accuracy: 0.9705 - val_loss: 1.2952 - val_accuracy: 0.6667
Epoch 4/50
21/21 [==============================] - 0s 3ms/step - loss: 0.1437 - accuracy: 0.9658 - val_loss: 1.2888 - val_accuracy: 0.6594
Epoch 5/50
21/21 [==============================] - 0s 4ms/step - loss: 0.1403 - accuracy: 0.9767 - val_loss: 1.3162 - val_accuracy: 0.6667
Epoch 6/50
```

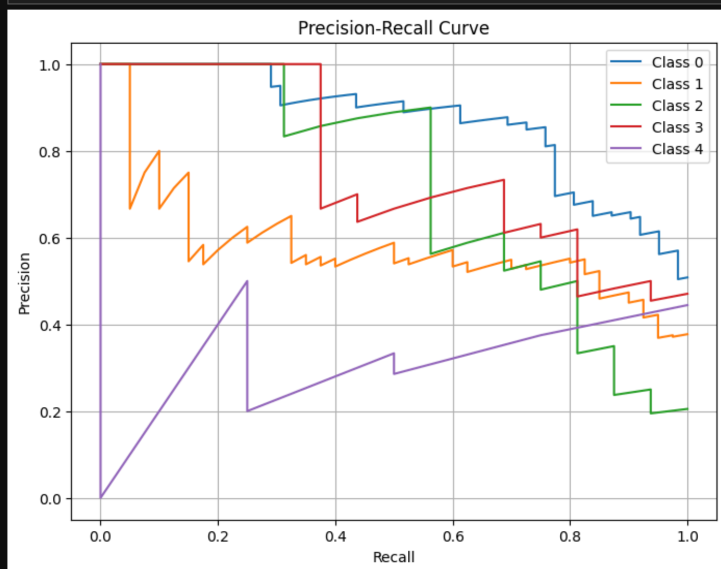
3. PERFORMANCE METRICS

Precision-Recall Curve:

Even though it is pretty clear from the detailed epoch results that the model is overfitting, checking Recall will benefit as in our project prioritizing recall minimizes the likelihood of undiagnosed cases, which aligns with ethical needs as well. In the healthcare system, diagnosing potential risks is often more important than avoiding false positives. Therefore Recall is a critical metric.

Test Accuracy: 0.66
Test Precision: 0.68
Test Recall: 0.64

[51]:
```
y_pred_probs = model.predict(X_test_scaled)
precision = {}
recall = {}

for i in range(y_test.shape[1]):
    precision[i], recall[i], _ = precision_recall_curve(y_test[:, i], y_pred_probs[:, i])

plt.figure(figsize=(8, 6))
for i in range(y_test.shape[1]):
    plt.plot(recall[i], precision[i], label=f'Class {i}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid()
plt.show()
```



Inference:

- ➔ The visualization above shows the trade-off between precision and recall for a specific class.
- ➔ Irregular curves point to the issue of class imbalance or difficulty predicting these classes. Eg. class 4 (highest severity)

Confusion Matrix:

This will show how well the model performs for each class:

> Diagonal elements: Correctly classified instances

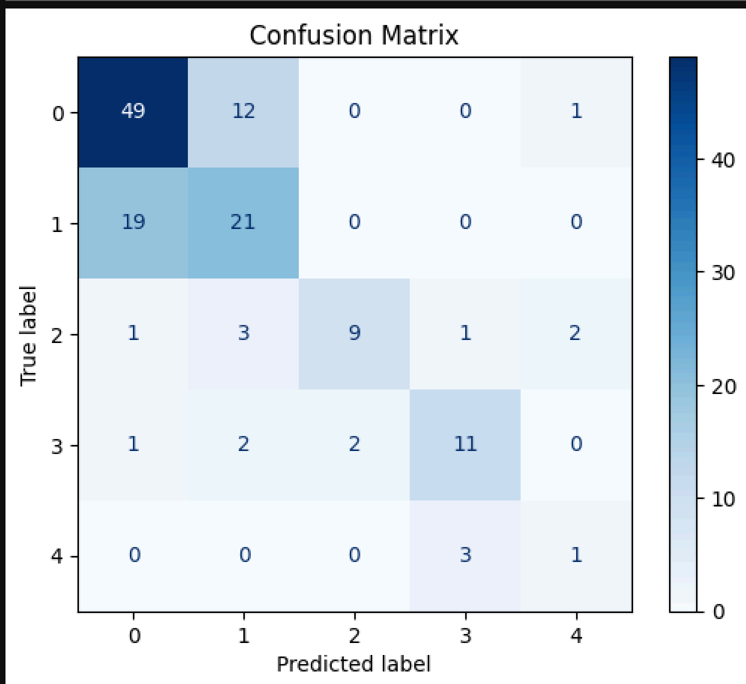
> Off-Diagonal elements: Misclassifications (type and count of errors)

Inference:

1. Most samples in class 0 are correctly predicted (49 instances). This can be due to the fact that number of instances for class 0 is high
2. Class 1 struggles as it has a high number of misclassifications; only 21 instances were correctly identified.
3. Class 2 and 3 had 9 and 11 instances correctly classified.
4. Class 4 had the worst performance. The predictions are sparse,showing poor performance for this class.

5. Overall: the model is performing worst on class 4 and that is our high priority target level. This points to the clear issue that arises with class imbalance.

[55]:
```
y_pred = np.argmax(model.predict(X_test_scaled), axis=1)
y_true = np.argmax(y_test, axis=1)

cm = confusion_matrix(y_true, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1, 2, 3, 4])
disp.plot(cmap=plt.cm.Blues)

plt.title("Confusion Matrix")
plt.show()
```



4. MODEL INTEGRATION INTO APPLICATION OR SYSTEM

Due to the potential issue prevalent from the above implementation this step will be taken after the accurate performance metrics are achieved which is still steps away.

After the desired metrics are achieved the following are some ways to deploy the model:

1. **Cloud based Deployment:** platforms like AWS, Google Cloud Platform etc. can be used .
   Upload the saved model to the cloud platform
   Configure the serving endpoint
   Cloud's API endpoint can be used to make predictions

2. **TensorFlow Extended (TFX https://www.tensorflow.org/tfx):**
   This includes tools for model serving, monitoring and updating models directly on production.

3. **ONNX (https://onnx.ai/):**
   Convert the model to ONNX format using appropriate environments such as pytorch, Microsoft ML.NET etc.

DETAILED MODEL IMPLEMENTATION SUMMARY

note programming language used, lines of code, key functions, what you wrote and what you adapted and refactored, tools used, and system you used to host your model.

Programming Language used:
Python is used for its vast library support for machine learning. Some important libraries used in this project are TensorFlow, Pandas, Scikit-learn, matplotlib etc.

Code: Divided into 2 sections first “FinalProjectCS581.ipynb” and second “ANNImplementation.ipynb”.

Environment Setup Requirements:

NOTE:
You will need to install python 3 for this project, tensorflow version 2.8.0, tensorboard=2.8.0 and tensorflow gpu=2.8.0.

- ➔ Connect to the Global Protect VPN and ssh to CSCI gpu server. We will be using an A100 virtual machine.

```
Last login: Thu Dec 19 19:08:50 on ttys000
(base) shambhavidanayak@gp-od-vpn-160-204 ~ % ssh sdanayak@cscigpu.csuchico.edu

******************************************************************************
This system is the property of California State University, Chico. Use of this
system is subject to the policies and standards set forth by the CSU system &
Chico State at https://www.csuchico.edu/it/about/policies. Unauthorized access
or misuse of resources or disclosure of protected information may result in
disciplinary or legal action. By continuing, you indicate your willingness
to comply with applicable policies and standards.
******************************************************************************

sdanayak@cscigpu.csuchico.edu's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-126-generic x86_64)

 System information as of Thu Dec 19 07:09:09 PM PST 2024

  System load:    3.71                Processes:              474
  Usage of /home: 65.0% of 1006.85GB  Users logged in:        3
  Memory usage:   5%                  IPv4 address for ens192: 132.241.1.14
  Swap usage:     0%

*** Livepatch has fixed vulnerabilities in the running kernel. If there is a new kernel available, upgrade and reboot ***

=================================
-- Welcome to CSCI GPU SERVER  --
=================================

Accounts and data for students will
be purged from this server before
the start of the next semester so
make sure to backup your files at
the end of the semester.
```

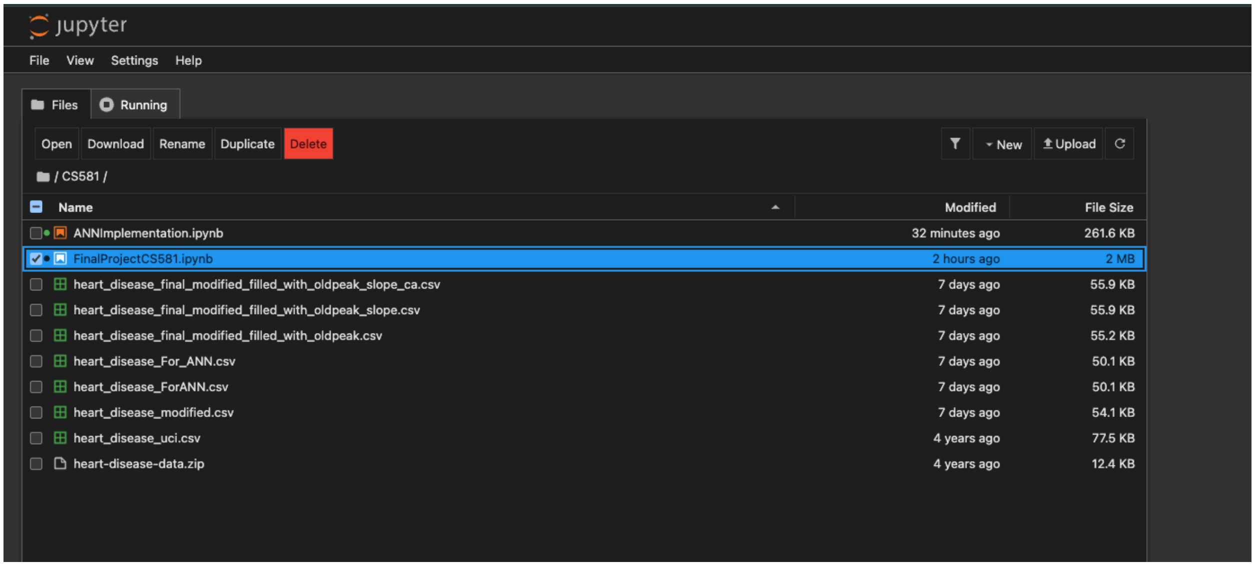
- ➔ Create a virtual python environment and activate it. Use the following commands to create the venv,

```
conda create --name myenv python=3.x
conda activate myenv
conda deactivate myenv
```

- ➔ Then install jupyter notebook and start it
- ➔ You will have to port forward to localhost in order to be able to use and edit jupyter notebook locally on your browser,

```
shambhavidanayak — sdanayak@cscigpu: ~ — ssh -L 8888:localhost:8888 sdanayak@cscigpu.csuchico.edu — 139×35
Last login: Thu Dec 19 19:09:01 on ttys000
(base) shambhavidanayak@gp-od-vpn-160-204 ~ % ssh -L 8888:localhost:8888 sdanayak@cscigpu.csuchico.edu


******************************************************************************
This system is the property of California State University, Chico. Use of this
system is subject to the policies and standards set forth by the CSU system &
Chico State at https://www.csuchico.edu/it/about/policies. Unauthorized access
or misuse of resources or disclosure of protected information may result in
disciplinary or legal action. By continuing, you indicate your willingness
to comply with applicable policies and standards.
******************************************************************************

sdanayak@cscigpu.csuchico.edu's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-126-generic x86_64)

 System information as of Thu Dec 19 07:09:09 PM PST 2024

  System load:    3.71               Processes:               474
  Usage of /home: 65.0% of 1006.85GB   Users logged in:         3
  Memory usage:   5%                 IPv4 address for ens192: 132.241.1.14
  Swap usage:     0%

*** Livepatch has fixed vulnerabilities in the running kernel. If there is a new kernel available, upgrade and reboot ***

=================================
-- Welcome to CSCI GPU SERVER  --
=================================

Accounts and data for students will
be purged from this server before
the start of the next semester so
make sure to backup your files at
the end of the semester.
```

The picture below shows the file configuration of project for reference:



MODEL IMPLEMENTATION SUMMARY

Programming Language	**System Used to Model & Train**	**% Code Reuse / % New**	**Description**
Python 3	CSCIGPU access, NVIDIA A100, Local machine, Conda environment	• No code is re- used but the coursebook "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition" and relevant jupyter notebook code is used as a starter code. • Reused standard ML libraries such as Tensorflow, Scikit-learn, pandas etc. for preprocessing, model building and evaluation.	Information, study, formulae etc. to implement the model came from various sources such as course book HML3, documentation for libraries such as Scikit-learn, tensorflow etc.

FUTURE STEPS:

There is always a scope of improvement:

- ➔ **Addressing overfitting:**
    - This needs to be further investigated. Remember during Imputation of missing values I used simple statistics which may have skewed the results.
    - Simple sub model to make predictions for important missing values such as fasting blood sugar (fbs), slope, thal etc. based on number of missing values and relevance in the project.
- ➔ **More balanced data:**
    - In the current dataset there is a huge class imbalance (less entries for severity level “num” = 4 compared to other classes 1,2 and 3) which is affecting the model accuracy to some extent. Since the goal is to prioritize recall we need to lower the false negatives. It is fine if the model results in some false positives but false negatives can be deadly.
    - Focus on techniques such as oversampling, undersampling SMOTE etc. for minority classes.
- ➔ **Model Benchmarking:**
  Performance comparison with other model types such as LogisticRegression, Decision tree, SVM etc. This will help evaluate if ANN is indeed the best choice for this task or if simpler models can achieve comparable results.

REFERENCES:

1. Computational Biology and Chemistry https://www.sciencedirect.com/science/article/abs/pii/S1476927122000524
2. Example platform that calculates heart risk factor, “Cardiac Risk Calculator”. https://my.clevelandclinic.org/health/articles/17085-heart-risk-factor-calculators
3. Steps to perform Exploratory Data Analysis, https://www.geeksforgeeks.org/steps-for-mastering-exploratory-data-analysis-eda-steps/
4. https://learning.oreilly.com/library/view/hands-on-machine-learning Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition
5. Tensorflow GitHub, https://github.com/tensorflow/tensorflow
6. TensorFlow Documentation https://www.tensorflow.org/tutorials/
7. Scikit-learn https://scikit-learn.org/stable/
8. Kaggle https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data
9. Neural Network study, https://en.wikipedia.org/wiki/Neural_network_(machine_learning)
10. Artificial Neural Network, https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/
11. Cross-Entropy, https://www.geeksforgeeks.org/categorical-cross-entropy-in-multi-class-classificat

ion/
12. Deep neural network, https://www.geeksforgeeks.org/deep-neural-net-with-forward-and-back-propagation-from-scratch-python/
13. Activation Function, https://www.geeksforgeeks.org/activation-functions-neural-networks/#2-nonlinear-activation-functions