FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published" (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

~~I /~~ **We***, the undersigned, declare that the [**assignment** ~~/ Assigned Practical Task report / Final Year Project report~~] submitted is ~~my /~~ **our*** work, except where acknowledged and referenced.

~~I /~~ **We*** understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Andrea Filiberto Lucas
Student Name

Signature

Sean David Muscat
Student Name

Signature

Student Name

Signature

ARI3205
Course Code

ARI3205 – Course Project
Title of work submitted

17-01-25
Date

ARI3205 - Course Project 2024/25

Interpretable AI for Deep Learning Models

Name: **Andrea Filiberto Lucas & Sean David Muscat**
Course: ARI3205 - Artificial Intelligence (AI)
ID No: 0279704L & 0172004L

Project's Distribution of Work

Section	Responsible Person
Task 1: Part1_BostonDataset.ipynb/.pdf	Andrea F. Lucas
Task 1: Part1_TitanicDataset.ipynb/.pdf	Andrea F. Lucas
Task 2: Part2_TitanicDataset.ipynb/.pdf	Sean D. Muscat
Task 3: Part3.1_TitanicDataset.ipynb/.pdf	Sean D. Muscat
Task 3: Part3.2_TitanicDataset.ipynb/.pdf	Sean D. Muscat
Task 4: Part4_CIFAR-10Dataset.ipynb/.pdf	Andrea F. Lucas

Project ARI3205 Interpretable AI for Deep Learning Models *(Part 1.1)*

Name: Andrea Filiberto Lucas
ID No: 0279704L

Importing Necessary Libraries

```python
import json

# Read the libraries from the text file
with open('../Libraries/Part1_Lib.json', 'r') as file:
    libraries = json.load(file)

# ANSI escape codes for colored output
GREEN = "\033[92m"  # Green text
RED = "\033[91m"    # Red text
RESET = "\033[0m"   # Reset to default color

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        try:
            # Attempt to import the library
            __import__(import_name)
            print(f"[{GREEN}✔{RESET}] Library '{lib}' is already 
installed.")
        except ImportError:
            # If import fails, try to install the library
            print(f"[{RED}✖{RESET}] Library '{lib}' is not installed. 
Installing...")
            %pip install {lib}

# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Import necessary libraries for data analysis and modeling
import pandas as pd
# Data manipulation and analysis                    #type: ignore
import numpy as np
# Numerical computations                            #type: ignore
import matplotlib.pyplot as plt
# Data visualization                                #type: ignore
import seaborn as sns
# Statistical data visualization                    #type: ignore
import statsmodels.formula.api as smf
# Statistical models                                #type: ignore
```

```python
from sklearn.model_selection import train_test_split
# Train-test split                                          #type: ignore
from tensorflow.keras.models import Sequential
# Neural network model                                      #type: ignore
from tensorflow.keras.layers import Dense
# Neural network layers                                     #type: ignore

from tensorflow.keras.optimizers import Adam
# Neural network optimizer                                  #type: ignore
from sklearn.preprocessing import StandardScaler
# Data scaling                                              #type: ignore
from sklearn.impute import SimpleImputer
# Missing value imputation                                  #type: ignore
from sklearn.inspection import PartialDependenceDisplay,
permutation_importance     # Feature importance
#type: ignore
from sklearn.neural_network import MLPRegressor
# Neural network model                                      #type: ignore
from sklearn.metrics import mean_squared_error
# Model evaluation                                          #type: ignore
from alibi.explainers import ALE, plot_ale
# ALE plots                                                 #type: ignore

# Suppress specific warnings
import warnings
warnings.filterwarnings("ignore", message="X does not have valid
feature names")
```

```
[✔] Library 'tensorflow' is already installed.
[✔] Library 'scikit-learn' is already installed.
[✔] Library 'matplotlib' is already installed.
[✔] Library 'seaborn' is already installed.
[✔] Library 'pandas' is already installed.
[✔] Library 'numpy' is already installed.
[✔] Library 'scipy' is already installed.
[✔] Library 'alibi' is already installed.
```

General Information on Boston Housing Dataset

https://www.kaggle.com/datasets/altavish/boston-housing-dataset/data

```python
# Define the filename
filename = '../Datasets/Boston/Boston.csv'

# Load the dataset
try:
    boston_data = pd.read_csv(filename)
    print(f"'{filename}' dataset loaded successfully.")
except FileNotFoundError:
```

```
    print(f"Error: The file '{filename}' was not found. Please ensure
it is in the correct directory.")
    exit()
except pd.errors.EmptyDataError:
    print(f"Error: The file '{filename}' is empty.")
    exit()
except pd.errors.ParserError:
    print(f"Error: There was a problem parsing '{filename}'. Please
check the file format.")
    exit()

# Dataset insights
print("\nDataset Overview:")
print(boston_data.info())
print("\nStatistical Summary:")
print(boston_data.describe())
```

```
'../Datasets/Boston/Boston.csv' dataset loaded successfully.

Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
None

Statistical Summary:
             CRIM          ZN       INDUS        CHAS         NOX
RM  \
count  486.000000  486.000000  486.000000  486.000000  506.000000
506.000000
mean     3.611874   11.211934   11.083992    0.069959    0.554695
6.284634
```

```
std      8.720192   23.388876    6.835896    0.255340    0.115878
0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000
3.561000
25%      0.081900    0.000000    5.190000    0.000000    0.449000
5.885500
50%      0.253715    0.000000    9.690000    0.000000    0.538000
6.208500
75%      3.560263   12.500000   18.100000    0.000000    0.624000
6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000
8.780000

              AGE         DIS         RAD         TAX     PTRATIO
B  \
count  486.000000  506.000000  506.000000  506.000000  506.000000
506.000000
mean    68.518519    3.795043    9.549407  408.237154   18.455534
356.674032
std     27.999513    2.105710    8.707259  168.537116    2.164946
91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000
0.320000
25%     45.175000    2.100175    4.000000  279.000000   17.400000
375.377500
50%     76.800000    3.207450    5.000000  330.000000   19.050000
391.440000
75%     93.975000    5.188425   24.000000  666.000000   20.200000
396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000
396.900000

            LSTAT        MEDV
count  486.000000  506.000000
mean    12.715432   22.532806
std      7.155871    9.197104
min      1.730000    5.000000
25%      7.125000   17.025000
50%     11.430000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

Feed-Forward Neural Network

```python
# Separate features and target
X = boston_data.drop(columns=['MEDV'])  # Features
y = boston_data['MEDV']  # Target

# Handle missing values with mean imputation
```

```
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Standardize the features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_imputed),
columns=X.columns)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Training data shape: (404, 13)
Test data shape: (102, 13)
```

```
# Build the feed-forward neural network
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for regression
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse',
metrics=['mae'])

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50,
batch_size=32, verbose=1)

# Evaluate the model
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss: {test_loss:.4f}, Test MAE: {test_mae:.4f}")
```

```
/opt/anaconda3/lib/python3.11/site-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Epoch 1/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 610.9976 - mae:
22.8815 - val_loss: 539.9886 - val_mae: 21.7096
Epoch 2/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 612.5311 - mae: 22.7226
- val_loss: 507.0454 - val_mae: 20.9489
Epoch 3/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 536.2062 - mae: 21.3987
```

```
- val_loss: 470.3633 - val_mae: 20.0871
Epoch 4/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 487.4485 - mae: 20.1953
- val_loss: 425.6573 - val_mae: 19.0079
Epoch 5/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 439.9500 - mae: 19.0457
- val_loss: 372.1023 - val_mae: 17.6258
Epoch 6/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 383.8455 - mae: 17.3025
- val_loss: 309.6705 - val_mae: 15.8827
Epoch 7/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 310.1775 - mae: 15.6345
- val_loss: 241.4313 - val_mae: 13.8186
Epoch 8/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 243.0562 - mae: 13.4235
- val_loss: 176.3131 - val_mae: 11.5594
Epoch 9/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 173.2832 - mae: 11.1322
- val_loss: 118.8051 - val_mae: 9.0884
Epoch 10/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 127.6825 - mae: 8.9653
- val_loss: 77.8989 - val_mae: 6.9037
Epoch 11/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 83.9577 - mae: 7.2203 -
val_loss: 56.1831 - val_mae: 5.5618
Epoch 12/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 67.8990 - mae: 6.5558 -
val_loss: 45.2629 - val_mae: 4.8866
Epoch 13/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 61.0328 - mae: 5.9132 -
val_loss: 38.2517 - val_mae: 4.4154
Epoch 14/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 49.0374 - mae: 5.5509 -
val_loss: 33.7060 - val_mae: 4.1265
Epoch 15/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 42.0957 - mae: 4.9648 -
val_loss: 31.0094 - val_mae: 3.9563
Epoch 16/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 37.2767 - mae: 4.4003 -
val_loss: 29.3667 - val_mae: 3.8668
Epoch 17/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 31.7819 - mae: 4.2120 -
val_loss: 28.4223 - val_mae: 3.8305
Epoch 18/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 27.9506 - mae: 4.0479 -
val_loss: 28.1628 - val_mae: 3.8181
Epoch 19/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 26.2952 - mae: 3.7631
- val_loss: 27.6732 - val_mae: 3.7870
```

```
Epoch 20/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 23.9321 - mae: 3.6995 -
val_loss: 27.5123 - val_mae: 3.7783
Epoch 21/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.4681 - mae: 3.7127 -
val_loss: 27.4019 - val_mae: 3.7606
Epoch 22/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 24.0510 - mae: 3.6659
- val_loss: 26.6372 - val_mae: 3.7184
Epoch 23/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 18.1317 - mae: 3.2892 -
val_loss: 26.2528 - val_mae: 3.6857
Epoch 24/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 20.0552 - mae: 3.3898 -
val_loss: 26.0505 - val_mae: 3.6825
Epoch 25/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 23.3051 - mae: 3.3978 -
val_loss: 25.6083 - val_mae: 3.6522
Epoch 26/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 20.8880 - mae: 3.3739 -
val_loss: 24.9881 - val_mae: 3.5586
Epoch 27/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 19.5687 - mae: 3.3424 -
val_loss: 24.4712 - val_mae: 3.5129
Epoch 28/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 20.0316 - mae: 3.3424 -
val_loss: 24.1334 - val_mae: 3.4745
Epoch 29/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 16.8976 - mae: 3.0855
- val_loss: 23.7792 - val_mae: 3.4335
Epoch 30/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 16.8489 - mae: 3.0665 -
val_loss: 23.6043 - val_mae: 3.4182
Epoch 31/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 19.3166 - mae: 3.1728
- val_loss: 23.2198 - val_mae: 3.4048
Epoch 32/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 21.4817 - mae: 3.1872 -
val_loss: 22.9691 - val_mae: 3.3908
Epoch 33/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.6022 - mae: 3.1756 -
val_loss: 23.1211 - val_mae: 3.3862
Epoch 34/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 18.8767 - mae: 3.0806 -
val_loss: 23.0233 - val_mae: 3.3929
Epoch 35/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 16.3944 - mae: 2.9686
- val_loss: 22.6667 - val_mae: 3.3721
Epoch 36/50
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 18.7178 - mae: 3.2670 -
val_loss: 22.1120 - val_mae: 3.3367
Epoch 37/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 16.5506 - mae: 3.0255 -
val_loss: 21.8547 - val_mae: 3.3120
Epoch 38/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.5920 - mae: 3.2243 -
val_loss: 21.2016 - val_mae: 3.2525
Epoch 39/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 17.1421 - mae: 3.0439
- val_loss: 20.7961 - val_mae: 3.2071
Epoch 40/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 14.4766 - mae: 2.8270 -
val_loss: 20.5739 - val_mae: 3.1850
Epoch 41/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.1502 - mae: 2.8702 -
val_loss: 20.6122 - val_mae: 3.2112
Epoch 42/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 14.2076 - mae: 2.7147
- val_loss: 20.5578 - val_mae: 3.2263
Epoch 43/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 16.2328 - mae: 2.9353 -
val_loss: 20.0096 - val_mae: 3.1807
Epoch 44/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 15.4274 - mae: 2.8205 -
val_loss: 19.6446 - val_mae: 3.1490
Epoch 45/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.5106 - mae: 2.8110 -
val_loss: 19.5938 - val_mae: 3.1421
Epoch 46/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 14.8909 - mae: 2.7844
- val_loss: 19.7601 - val_mae: 3.1517
Epoch 47/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 14.3406 - mae: 2.7273 -
val_loss: 20.1172 - val_mae: 3.1564
Epoch 48/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 12.7872 - mae: 2.6688 -
val_loss: 19.4098 - val_mae: 3.1072
Epoch 49/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 12.6852 - mae: 2.6272 -
val_loss: 18.8763 - val_mae: 3.0776
Epoch 50/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 14.3075 - mae: 2.7343
- val_loss: 18.8568 - val_mae: 3.0921
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 11.2812 - mae: 2.2573
Test Loss: 15.3235, Test MAE: 2.3687
```

Surrogate Model - MLPRegressor

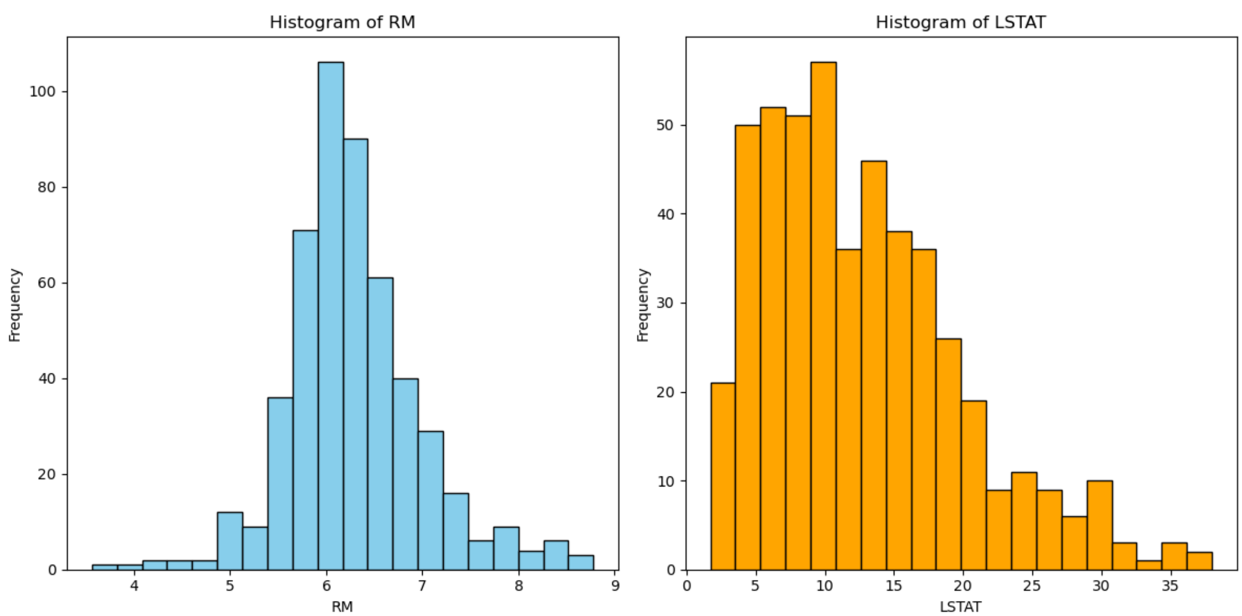
```
# Train an MLPRegressor as a surrogate model
surrogate_model = MLPRegressor(hidden_layer_sizes=(64, 32),
max_iter=1000, random_state=42)
surrogate_model.fit(X_train, y_train)

# Evaluate the surrogate model
y_pred = surrogate_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Surrogate Model Mean Squared Error: {mse:.4f}")
```

```
Surrogate Model Mean Squared Error: 12.7475
```

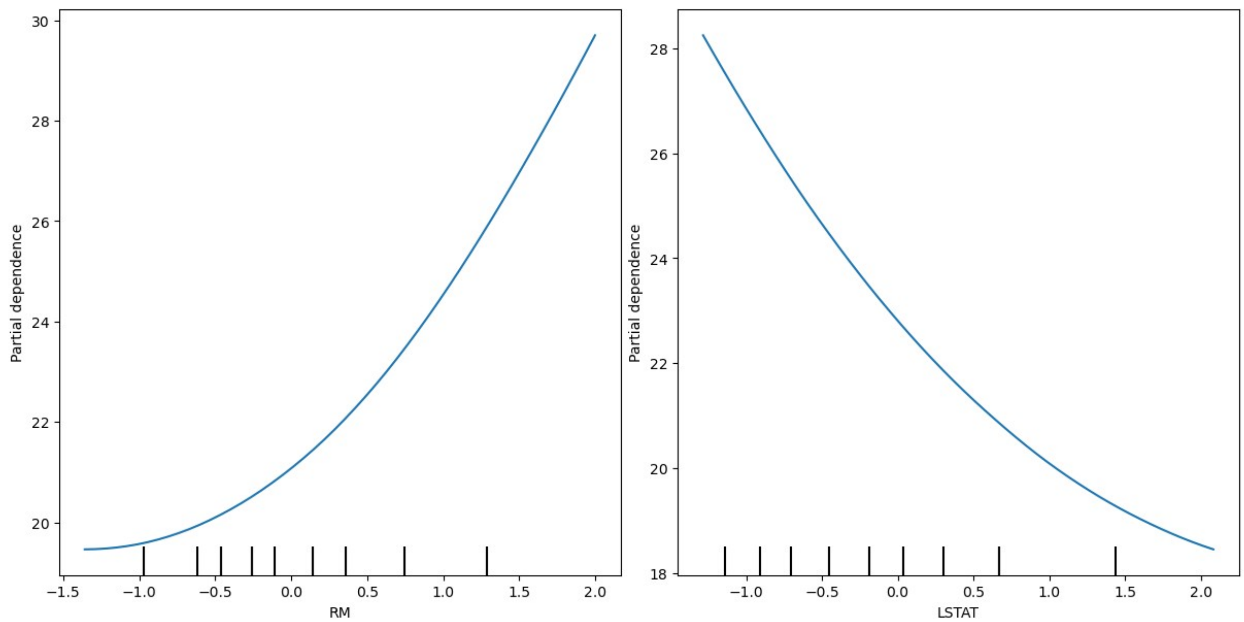
Partial Dependence Plots (PDP) and Individual Conditional Expectation (ICE) plots

```
# Partial Dependence Plots (PDP)
def plot_pdp(features):
    print("\nGenerating PDP for features:", features)
    fig, ax = plt.subplots(1, len(features), figsize=(12, 6),
constrained_layout=True)
    for i, feature in enumerate(features):
        PartialDependenceDisplay.from_estimator(
            surrogate_model,  # The trained surrogate model
(MLPRegressor)
            X_train,          # Training data
            features=[feature],  # Single feature for PDP
            kind="average",   # PDP only
            ax=ax[i] if len(features) > 1 else ax,
            grid_resolution=50,
        )
        ax[i].set_title(f"PDP for {feature}")
    plt.show()

# Individual Conditional Expectation (ICE) Plots
def plot_ice(features):
    print("\nGenerating ICE for features:", features)
    fig, ax = plt.subplots(1, len(features), figsize=(12, 6),
constrained_layout=True)
    for i, feature in enumerate(features):
        PartialDependenceDisplay.from_estimator(
            surrogate_model,  # The trained surrogate model
(MLPRegressor)
            X_train,          # Training data
            features=[feature],  # Single feature for ICE
            kind="both",      # PDP and ICE
            ax=ax[i] if len(features) > 1 else ax,
            grid_resolution=50,
```

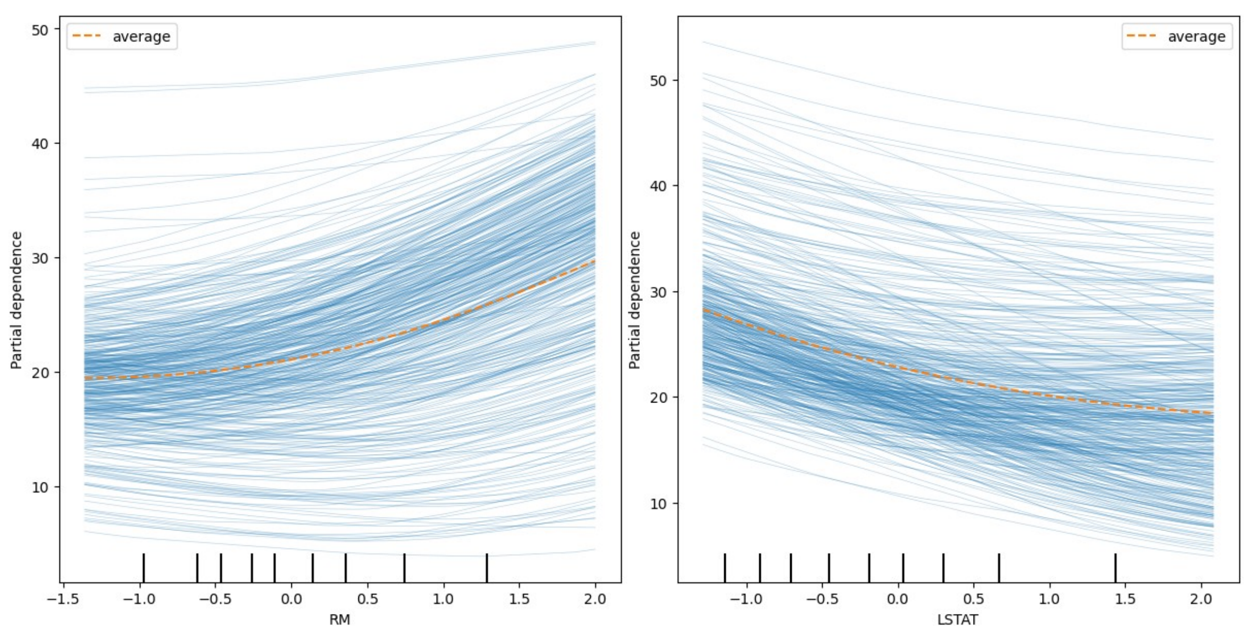
```
        )
        ax[i].set_title(f"ICE and PDP for {feature}")
    plt.show()

# Call PDP and ICE plot functions
features_to_analyze = ['RM', 'LSTAT']

# Plot histograms for features_to_analyze
plt.figure(figsize=(12, 6))
for i, feature in enumerate(features_to_analyze):
    plt.subplot(1, len(features_to_analyze), i + 1)
    plt.hist(boston_data[feature], bins=20, edgecolor='black',
color='skyblue' if i % 2 == 0 else 'orange')
    plt.title(f'Histogram of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

plot_pdp(features_to_analyze)
plot_ice(features_to_analyze)
```




```
Generating PDP for features: ['RM', 'LSTAT']
```




```
Generating ICE for features: ['RM', 'LSTAT']
```



Explain what insights PDP and ICE give about the model's behaviour.

The **Partial Dependence Plots (PDPs)** and **Individual Conditional Expectation (ICE) plots** for the features `RM` (average number of rooms per dwelling) and `LSTAT` (percentage of lower-status population) provide critical insights into the predictive behavior of the trained model. These features were selected due to their strong correlation with the target variable, `MEDV` (median house prices), and their contrasting trends, which are both relevant and interpretable in the context of housing prices.

Partial Dependence Plots (PDP)

PDPs offer a global perspective on the relationship between features and the predicted target variable. The PDP for `RM` shows a **positive monotonic relationship**, indicating that an increase in the number of rooms correlates with higher predicted house prices. This trend aligns with real-world expectations, as larger homes are typically associated with higher market values. On the other hand, the PDP for `LSTAT` reveals a **negative monotonic relationship**, suggesting that areas with a higher percentage of lower-status populations are associated with lower predicted house prices. This reflects the model's ability to capture the socioeconomic factors influencing housing prices effectively.

While PDPs are excellent for understanding the overall trends, they average the effects across all data points and fail to capture individual variations or interactions between features. Thus, they provide a general but limited view of feature importance.

Individual Conditional Expectation (ICE) Plots

ICE plots complement PDPs by revealing how predictions vary for individual instances when the feature values change. For `RM`, the ICE plots demonstrate that most individual instances follow the positive trend shown in the PDP. However, there are subtle variations in the slopes of individual lines, indicating that the sensitivity of predictions to `RM` differs across instances. For example, homes in different neighborhoods or price ranges might respond differently to changes in the number of rooms. Similarly, for `LSTAT`, the ICE plots reveal a consistent negative slope across most instances, consistent with the PDP. However, the variability in slopes highlights heterogeneous relationships, where certain neighborhoods or homes might be less affected by changes in the percentage of lower-status populations.

ICE plots are especially valuable for identifying outliers or subgroups that deviate from the average behavior. They provide a granular view, offering insights into how specific instances behave, which is critical for understanding model predictions at an individual level.

Insights from Combining PDP and ICE

The combined analysis of PDPs and ICE plots offers a comprehensive understanding of the model's behavior. PDPs provide a **global average perspective**, while ICE plots uncover **instance-level variability** and highlight heterogeneous effects. For example, while the global trend for `RM` suggests a steady increase in house prices with more rooms, the ICE plots reveal that the degree of sensitivity varies across instances. Similarly, for `LSTAT`, while the global trend shows a decrease in prices with higher percentages of lower-status populations, the ICE plots expose differences in how sensitive various neighborhoods are to this feature.

Key Takeaways

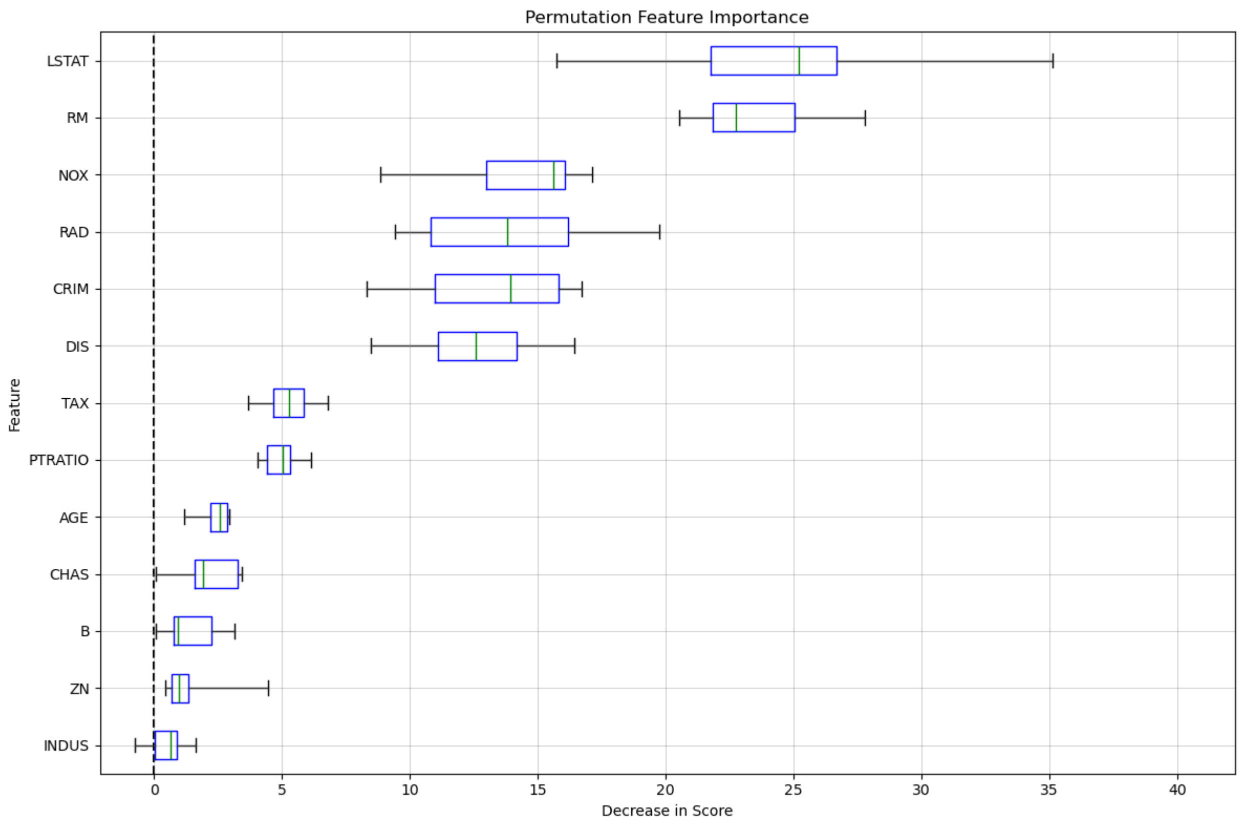
The analysis of `RM` and `LSTAT` underscores their relevance to housing prices. The positive relationship of `RM` with house prices reflects the impact of housing size and quality, while the negative relationship of `LSTAT` highlights the influence of socioeconomic factors. By leveraging both PDPs and ICE plots, we gain a robust understanding of the model's predictions, ensuring that they align with real-world expectations while identifying potential areas for improvement. Together, these tools enhance interpretability, providing both broad insights and detailed individual-level analysis.

Permutation Feature Importance (PFI)

```
# Compute Permutation Feature Importance
def compute_pfi(model, X_test, y_test, feature_names):
    pfi_result = permutation_importance(model, X_test, y_test,
n_repeats=10, random_state=42, scoring='neg_mean_squared_error')

    # Convert PFI results into a DataFrame for better visualization
    importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': pfi_result.importances_mean,
        'Std': pfi_result.importances_std
    })

    # Sort features by importance
    importance_df = importance_df.sort_values(by='Importance',
ascending=False)

    print("\nPermutation Feature Importance:\n", importance_df)
    return importance_df

# Plot Permutation Feature Importance as a Boxplot
def plot_pfi(model, X, y, feature_names):
    result = permutation_importance(model, X, y,
scoring='neg_mean_squared_error', n_repeats=10, random_state=42,
n_jobs=2)

    sorted_importances_idx = result.importances_mean.argsort()
    importances =
pd.DataFrame(result.importances[sorted_importances_idx].T,
                               columns=[feature_names[i] for i in
sorted_importances_idx])

    ax = importances.plot.box(vert=False, whis=10, figsize=(12, 8),
color=dict(boxes="blue", whiskers="black", medians="green",
caps="black"))
    ax.axvline(x=0, color="k", linestyle="--")

    # Add faint grey lines across the graph for each feature
    for i in range(len(importances.columns)):
        plt.axhline(y=i + 1, color="grey", linestyle="-",
linewidth=0.5, alpha=0.5)

    # Add faint grey lines upwards from the x-axis ticks
    xticks = ax.get_xticks()
    for tick in xticks:
        plt.axvline(x=tick, color="grey", linestyle="-",
linewidth=0.5, alpha=0.5)

    # Set the x-axis limits
```

```
    ax.set_xlim(left=0 - 0.05 * (ax.get_xlim()[1] - 0))

    ax.set_xlabel("Decrease in Score")
    ax.set_ylabel("Feature")
    ax.set_title("Permutation Feature Importance")
    plt.tight_layout()
    plt.show()

feature_names = X_test.columns
importance_df = compute_pfi(surrogate_model, X_test, y_test,
feature_names)
plot_pfi(surrogate_model, X_test, y_test, feature_names)
```

```
Permutation Feature Importance:
     Feature  Importance       Std
12     LSTAT   24.594452  4.819524
5         RM   23.492247  2.306411
4        NOX   14.488889  2.533761
8        RAD   13.979112  3.490214
0       CRIM   13.408561  2.758039
7        DIS   12.747733  2.383452
9        TAX    5.263374  0.893353
10   PTRATIO    4.990779  0.644088
6        AGE    2.380155  0.586128
3       CHAS    2.149071  1.070397
11         B    1.384748  1.012892
1         ZN    1.331404  1.110311
2      INDUS    0.470038  0.717250
```



Permutation Feature Importance Results

The **Permutation Feature Importance (PFI)** results, visualized in the boxplot above, rank features based on their impact on the model's predictions for `MEDV` (median house prices). `LSTAT` (percentage of lower-status population) emerges as the most critical feature with a mean importance score of 24.59 and a standard deviation of 4.82. This underscores its significant negative influence on housing prices, aligning with socioeconomic realities. `RM` (average number of rooms per dwelling) follows closely, with an importance score of 23.49 and a lower standard deviation of 2.31, reflecting its strong positive correlation with property values.

Other influential features include `NOX` (nitric oxide concentration), `RAD` (accessibility to radial highways), and `CRIM` (per capita crime rate), which reflect environmental and neighborhood-related factors affecting housing prices, with importance scores of 14.49, 13.98, and 13.41, respectively. Conversely, features like `PTRATIO`, `AGE`, and `CHAS` show relatively lower importance, suggesting limited predictive value, while `INDUS` and `ZN` have the least influence, indicating minimal relevance to the model.

The boxplot highlights variability in feature importance scores across permutations. `LSTAT` and `RM` show compact whiskers, indicating consistent importance, while features like `CHAS` display greater variability, suggesting more context-dependent contributions. This ranking and variability provide a clear understanding of which features are robustly influential and which may have situational relevance.

Explain what the term “important” means when using the PFI method.

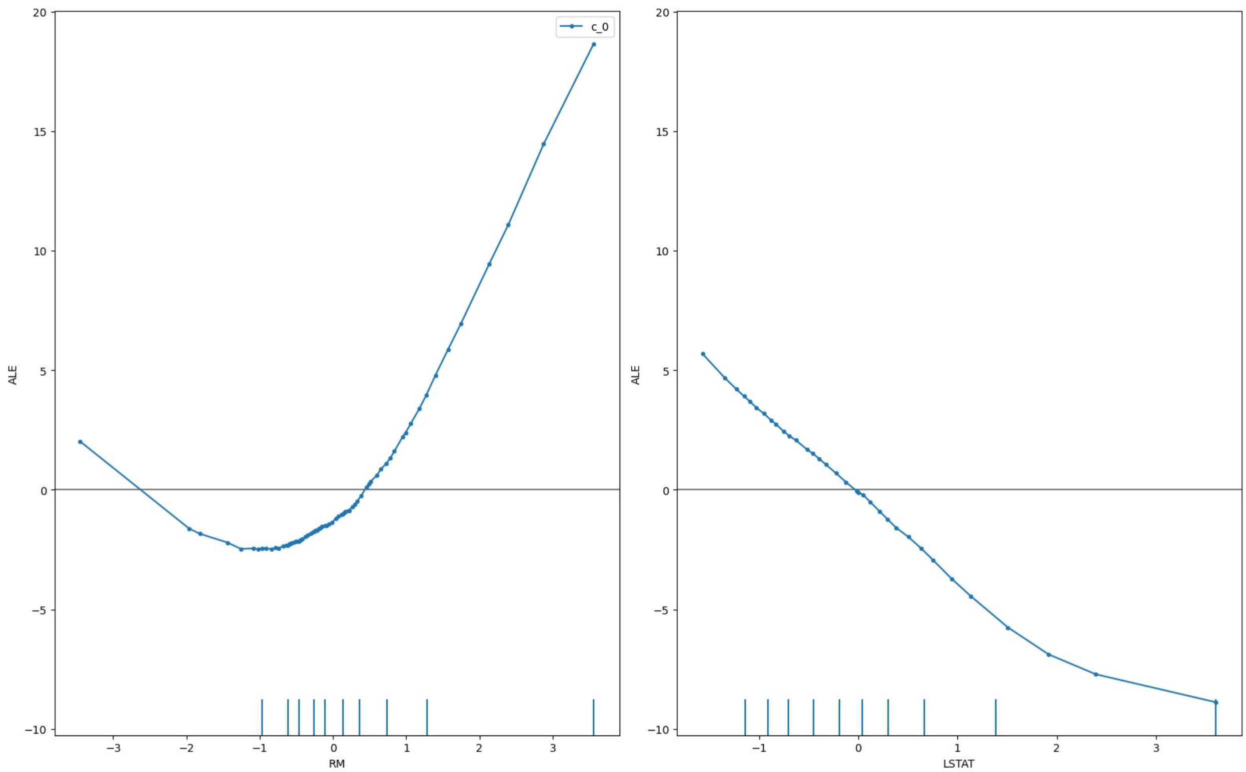
In the PFI method, **importance** quantifies the contribution of a feature to the model's predictive performance. This is measured by observing the increase in prediction error when a feature's values are permuted randomly while keeping other features unchanged. A higher importance score indicates that randomization significantly degrades the model's accuracy, implying that the feature provides critical information. Conversely, a lower score suggests that the feature's randomization has minimal effect, reflecting limited predictive value.

For the given results, the high importance scores of `LSTAT` and `RM` illustrate their dominant role in capturing critical factors like socioeconomic status and home size, directly influencing housing prices. The consistent importance (low standard deviations) of these features indicates their robust and global relevance across the dataset. In contrast, the low scores for `INDUS` and `ZN` show that these features contribute minimally to the model's predictions. The variability in features like `CHAS` highlights that their importance may vary depending on specific data subsets or interactions with other features. This demonstrates how PFI captures both direct and indirect feature effects, offering a comprehensive view of feature relevance.

Accumulated Local Effects (ALE)

```
# Combine features and target for context if needed
data = pd.concat([X_train, y_train], axis=1)
# Define feature names
feature_names = X_train.columns

# Ensure valid input for ALE explainer
X_train_array = X_train.to_numpy()  # Convert to NumPy array to avoid
warnings

# Create and compute ALE explainer
ale_explainer = ALE(surrogate_model.predict,
feature_names=feature_names)
ale_explanation = ale_explainer.explain(X_train_array)

# Plot ALE for all features
plot_ale(
    ale_explanation,
    features=['RM', 'LSTAT'], # Select specific features
    n_cols=4,  # Arrange plots in 4 columns for better visualization
    fig_kw={'figwidth': 16, 'figheight': 10}  # Adjust figure size for
clarity
)
```

```
array([[<Axes: xlabel='RM', ylabel='ALE'>,
        <Axes: xlabel='LSTAT', ylabel='ALE'>]], dtype=object)
```



Comparing ALE and PDP for `RM` and `LSTAT`

The **Accumulated Local Effects (ALE)** plots and **Partial Dependence Plots (PDPs)** offer valuable insights into the relationships between features and the target variable (`MEDV`, median house prices). Both techniques aim to interpret model behavior but differ in their methodologies, which is evident when examining the features `RM` (average number of rooms per dwelling) and `LSTAT` (percentage of lower-status population).

The ALE plots for `RM` and `LSTAT` provide a localized view of feature effects by calculating the average change in predictions within intervals of the feature values. For `RM`, the ALE plot reveals a strong positive and nonlinear relationship, with house prices increasing steeply as the number of rooms rises, particularly for higher values of `RM`. Interestingly, the plot also highlights a slight dip for lower values of `RM`, indicating a small negative impact on house prices in cases of very small homes before the overall upward trend dominates. For `LSTAT`, the ALE plot shows a consistent negative relationship across the feature range. Housing prices decline as the percentage of lower-status populations increases, with the effect intensifying at higher values of `LSTAT`. These localized trends highlight how the model captures nuanced relationships that vary across different feature ranges.

In comparison, the PDPs for `RM` and `LSTAT` provide a global perspective by showing the average effect of each feature across all instances. For `RM`, the PDP depicts a smooth monotonic increase, reinforcing the positive association between the number of rooms and housing prices. Similarly, the PDP for `LSTAT` demonstrates a monotonic decline, confirming that higher percentages of lower-status populations are correlated with lower house prices. However, unlike ALE, the PDPs do not capture the subtle dip observed for lower values of `RM`. This is because PDPs average predictions across the dataset, which can smooth out localized variations and obscure interactions between features.

The primary distinction between ALE and PDP lies in their interpretability. PDPs provide a straightforward global view, offering an easy-to-understand summary of feature effects. However, they may be influenced by feature correlations, as the marginalization process does not account for dependencies between features. In contrast, ALE plots focus on localized effects and are more robust to feature correlations. They offer a clearer picture of nonlinear relationships and feature interactions, as evidenced by the additional details visible in the ALE plot for `RM`.

Both techniques agree on the general trends for `RM` and `LSTAT`: `RM` has a positive impact on housing prices, while `LSTAT` has a negative influence. However, the ALE plots add granularity by capturing localized behaviors and variations that the PDPs overlook. Together, these methods complement each other, with PDPs providing a broad overview and ALE offering detailed insights into localized effects, enabling a deeper understanding of the model's behavior.

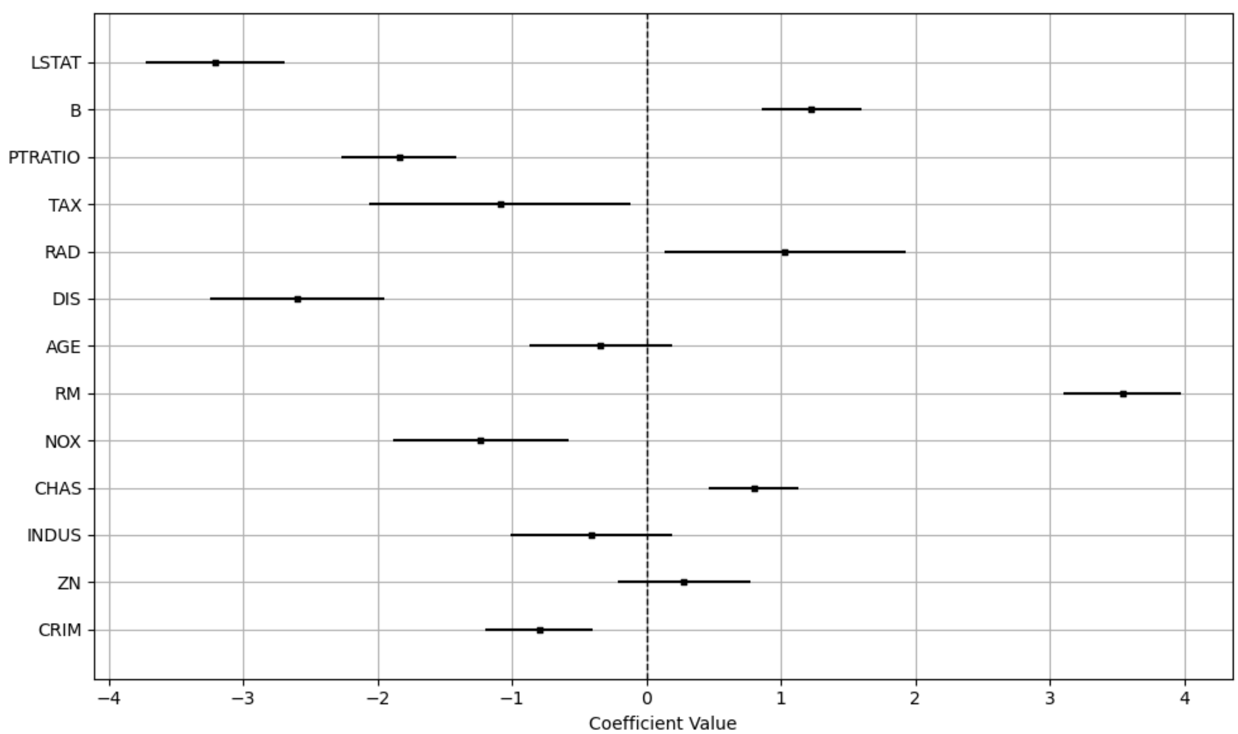
Global Surrogates

```
# Generate predictions from the neural network
NN_labels = model.predict(X_train).flatten()
X_train['NN_labels'] = NN_labels

# Train an interpretable linear regression model
formula = 'NN_labels ~ ' + ' + '.join(X_train.columns[:-1])  # Include
all features in the formula
lin_reg = smf.ols(formula=formula, data=X_train).fit()
print(lin_reg.summary())
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
                            OLS Regression Results

======================================================================
========
Dep. Variable:              NN_labels   R-squared:
0.865
Model:                            OLS   Adj. R-squared:
0.861
Method:                 Least Squares   F-statistic:
192.2
Date:                Fri, 17 Jan 2025   Prob (F-statistic):
1.75e-160
Time:                        16:47:05   Log-Likelihood:
-1047.3
No. Observations:                 404   AIC:
2123.
Df Residuals:                     390   BIC:
2179.
Df Model:                          13
```

```
Covariance Type:            nonrobust

==============================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
Intercept     22.4288      0.164    136.478      0.000      22.106
22.752
CRIM          -0.7999      0.204     -3.931      0.000      -1.200
-0.400
ZN             0.2769      0.251      1.105      0.270      -0.216
0.770
INDUS         -0.4080      0.305     -1.339      0.181      -1.007
0.191
CHAS           0.7963      0.168      4.739      0.000       0.466
1.127
NOX           -1.2324      0.332     -3.709      0.000      -1.886
-0.579
RM             3.5376      0.221     15.982      0.000       3.102
3.973
AGE           -0.3417      0.271     -1.262      0.208      -0.874
0.191
DIS           -2.5984      0.328     -7.919      0.000      -3.243
-1.953
RAD            1.0275      0.456      2.251      0.025       0.130
1.925
TAX           -1.0905      0.493     -2.212      0.028      -2.060
-0.121
PTRATIO       -1.8416      0.216     -8.539      0.000      -2.266
-1.418
B              1.2243      0.189      6.494      0.000       0.854
1.595
LSTAT         -3.2076      0.263    -12.197      0.000      -3.725
-2.691
==============================================================================
========
Omnibus:                       63.781   Durbin-Watson:
2.068
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
121.955
Skew:                           0.880   Prob(JB):
3.30e-27
Kurtosis:                       5.036   Cond. No.
9.69
==============================================================================
========
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.

# Extract coefficients and confidence intervals
err_series = lin_reg.params - lin_reg.conf_int()[0]
coef_df = pd.DataFrame({
    'coef': pd.to_numeric(lin_reg.params.values[1:], errors='coerce'),
    'err': pd.to_numeric(err_series.values[1:], errors='coerce'),
    'varname': err_series.index.values[1:]
})

# Visualize the coefficients and confidence intervals
fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(coef_df['varname'], coef_df['coef'], xerr=coef_df['err'],
color='none', edgecolor=None)
ax.scatter(y=coef_df['varname'], x=coef_df['coef'], marker='s', s=10,
color='black')
ax.axvline(x=0, linestyle='--', color='black', linewidth=1)

ax.set_xlabel('Coefficient Value')
ax.set_ylabel('')
ax.grid(True)

plt.tight_layout()
plt.show()
```



Analyse the surrogate model's effectiveness and discuss when such approximations are helpful.

The surrogate model, represented by a linear regression trained on the predictions of the neural network, demonstrates strong performance, with an R-squared value of 0.865. This suggests that 86.5% of the variance in the neural network's predictions (`NN_labels`) is captured by the linear regression model. The feature coefficients provide interpretable insights into the relationships between predictors and predictions, which are visualized in the coefficient plot. For example, `LSTAT` (percentage of lower-status population) shows the strongest negative relationship with a coefficient of -3.21, while `RM` (average number of rooms per dwelling) exhibits the strongest positive relationship with a coefficient of 3.54. These findings are consistent with prior domain knowledge, further validating the surrogate model's ability to approximate the behavior of the original neural network.

However, it is essential to recognize the limitations of such approximations, particularly in the context of these results. While the linear regression surrogate successfully captures the general trends of the neural network, it may oversimplify complex nonlinear interactions or dependencies between features. For instance, the neural network may model intricate relationships between features like `NOX` (nitric oxide concentration) and `DIS` (distance to employment centers) that are not reflected in the linear coefficients. This potential oversimplification becomes evident when considering features with weaker coefficients, such as `INDUS` (proportion of non-retail business acres) or `ZN` (proportion of residential land zoned for large lots). The neural network might account for interactions or nonlinearities involving these features, but the surrogate model reduces them to linear, independent contributions.

The visualized coefficients further support this observation. Features like `PTRATIO` (pupil-teacher ratio) and `TAX` (property tax rate), which have significant but modest coefficients, might have interactions with other variables that the linear model cannot capture. This limitation underscores the risk of misinterpretation if the surrogate model is used as the sole explanation of the neural network's behavior. For example, while `CHAS` (proximity to the Charles River) has a positive coefficient in the surrogate model, its influence in the neural network may be conditional on other features, such as `RM` or `DIS`.

In conclusion, surrogate models like the linear regression used here are valuable for enhancing interpretability while preserving much of the predictive power of the original model. They are particularly effective in distilling insights from complex models into an accessible form, as seen with the clear contributions of `LSTAT` and `RM` to predictions. However, these approximations come with trade-offs. Care must be taken to communicate that the linear regression model provides a simplified view of the neural network's behavior, and its insights should be complemented with other interpretability techniques to ensure a more comprehensive understanding of the model's decision-making process.

Project ARI3205 Interpretable AI for Deep Learning Models *(Part 1.2)*

Name: Andrea Filiberto Lucas
ID No: 0279704L

Importing Necessary Libraries

```
# Check and install required libraries from the libraries.json file
import json

# Read the libraries from the text file
with open('../Libraries/Part1_Lib.json', 'r') as file:
    libraries = json.load(file)

# ANSI escape codes for colored output
GREEN = "\033[92m"  # Green text
RED = "\033[91m"    # Red text
RESET = "\033[0m"   # Reset to default color

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        try:
            # Attempt to import the library
            __import__(import_name)
            print(f"[{GREEN}✔{RESET}] Library '{lib}' is already 
installed.")
        except ImportError:
            # If import fails, try to install the library
            print(f"[{RED}✖{RESET}] Library '{lib}' is not installed. 
Installing...")
            %pip install {lib}

# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Import necessary libraries for data analysis and modeling
import warnings
# Disable warnings
import pandas as pd
# Data manipulation and analysis                        #type: ignore
import numpy as np
# Numerical computations                                #type: ignore
import matplotlib.pyplot as plt
# Data visualization                                    #type: ignore
import seaborn as sns
```

```
# Statistical data visualization                          #type: ignore
import statsmodels.formula.api as smf
# Statistical models                                      #type: ignore
from sklearn.model_selection import train_test_split
# Train-test split                                        #type: ignore
from tensorflow.keras.models import Sequential
# Neural network model                                    #type: ignore
from tensorflow.keras.layers import Dense, Input
# Neural network layers                                   #type: ignore

from tensorflow.keras.optimizers import Adam
# Neural network optimizer                                #type: ignore
from sklearn.preprocessing import StandardScaler,  OneHotEncoder
# Data scaling                                            #type: ignore
from sklearn.impute import SimpleImputer
# Missing value imputation                                #type: ignore
from sklearn.inspection import PartialDependenceDisplay,
permutation_importance     # Feature importance
#type: ignore
from alibi.explainers import ALE, plot_ale
# ALE plots                                               #type: ignore
from sklearn.neural_network import MLPClassifier
# Neural network classifier                               #type: ignore
from sklearn.metrics import accuracy_score
# Model evaluation                                        #type: ignore
import statsmodels.api as sm
# Statistical models                                      #type: ignore

# Suppress specific warnings
warnings.filterwarnings("ignore", message="X does not have valid
feature names")
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

```
[✔] Library 'tensorflow' is already installed.
[✔] Library 'scikit-learn' is already installed.
[✔] Library 'matplotlib' is already installed.
[✔] Library 'seaborn' is already installed.
[✔] Library 'pandas' is already installed.
[✔] Library 'numpy' is already installed.
[✔] Library 'scipy' is already installed.
[✔] Library 'alibi' is already installed.
```

General Information on Titanic Dataset

https://www.kaggle.com/competitions/titanic/data

```
# Define the filenames
train_filename = '../Datasets/Titanic/train.csv'
```

```python
test_filename = '../Datasets/Titanic/test.csv'
gender_submission_filename =
'../Datasets/Titanic/gender_submission.csv'

# Load the datasets
try:
    train_data = pd.read_csv(train_filename)
    test_data = pd.read_csv(test_filename)
    gender_submission_data = pd.read_csv(gender_submission_filename)
    print(f"'{train_filename}' dataset loaded successfully.")
    print(f"'{test_filename}' dataset loaded successfully.")
    print(f"'{gender_submission_filename}' dataset loaded
successfully.")
except FileNotFoundError as e:
    print(f"Error: {e.filename} was not found. Please ensure it is in
the correct directory.")
    exit()
except pd.errors.EmptyDataError as e:
    print(f"Error: {e.filename} is empty.")
    exit()
except pd.errors.ParserError as e:
    print(f"Error: There was a problem parsing {e.filename}. Please
check the file format.")
    exit()

# Dataset insights
print("\nTrain Dataset Overview:")
print(train_data.info())
print("\nTrain Dataset Statistical Summary:")
print(train_data.describe())

print("\nTest Dataset Overview:")
print(test_data.info())
print("\nTest Dataset Statistical Summary:")
print(test_data.describe())

print("\nGender Submission Dataset Overview:")
print(gender_submission_data.info())
```

```
'../Datasets/Titanic/train.csv' dataset loaded successfully.
'../Datasets/Titanic/test.csv' dataset loaded successfully.
'../Datasets/Titanic/gender_submission.csv' dataset loaded
successfully.

Train Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

Train Dataset Statistical Summary:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000   
mean    446.000000    0.383838    2.308642   29.699118    0.523008   
std     257.353842    0.486592    0.836071   14.526497    1.102743   
min       1.000000    0.000000    1.000000    0.420000    0.000000   
25%     223.500000    0.000000    2.000000   20.125000    0.000000   
50%     446.000000    0.000000    3.000000   28.000000    0.000000   
75%     668.500000    1.000000    3.000000   38.000000    1.000000   
max     891.000000    1.000000    3.000000   80.000000    8.000000   

            Parch        Fare  
count  891.000000  891.000000  
mean     0.381594   32.204208  
std      0.806057   49.693429  
min      0.000000    0.000000  
25%      0.000000    7.910400  
50%      0.000000   14.454200  
75%      0.000000   31.000000  
max      6.000000  512.329200  

Test Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
```

```
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
None

Test Dataset Statistical Summary:
       PassengerId      Pclass         Age       SibSp       Parch
Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000
417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344
35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429
55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000
0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000
7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000
14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000
31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000
512.329200

Gender Submission Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Survived     418 non-null    int64
dtypes: int64(2)
memory usage: 6.7 KB
None
```

Feed-Forward Neural Network

```
# Load the Titanic dataset
train_data = pd.read_csv('../Datasets/Titanic/train.csv')

# Preprocessing
# Separate features and target
y = train_data['Survived']  # Target
X = train_data.drop(columns=['Survived', 'PassengerId', 'Name',
```

```
'Ticket', 'Cabin'])  # Features

# Handle categorical variables with one-hot encoding
categorical_features = ['Sex', 'Embarked']
one_hot_encoder = OneHotEncoder(sparse_output=False,
handle_unknown='ignore')
categorical_encoded =
one_hot_encoder.fit_transform(X[categorical_features])
categorical_encoded_df = pd.DataFrame(categorical_encoded,
columns=one_hot_encoder.get_feature_names_out(categorical_features))

# Drop original categorical columns and append the encoded columns
X = X.drop(columns=categorical_features)
X = pd.concat([X.reset_index(drop=True),
categorical_encoded_df.reset_index(drop=True)], axis=1)

# Handle missing values with mean imputation
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Standardize the features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_imputed),
columns=X.columns)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Training data shape: (712, 11)
Test data shape: (179, 11)
```

```
# Build the feed-forward neural network
model = Sequential([
    Input(shape=(X_train.shape[1],)),  # Define input shape explicitly
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer for binary
classification
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50,
batch_size=32, verbose=1)
```

```python
# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy:
{test_accuracy:.4f}")
```

```
Epoch 1/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.4462 - loss:
0.7103 - val_accuracy: 0.7902 - val_loss: 0.6079
Epoch 2/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7787 - loss:
0.5854 - val_accuracy: 0.8042 - val_loss: 0.5221
Epoch 3/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7958 - loss:
0.5179 - val_accuracy: 0.8252 - val_loss: 0.4649
Epoch 4/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7779 - loss:
0.4995 - val_accuracy: 0.8322 - val_loss: 0.4272
Epoch 5/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8028 - loss:
0.4671 - val_accuracy: 0.8322 - val_loss: 0.4102
Epoch 6/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7966 - loss:
0.4537 - val_accuracy: 0.8322 - val_loss: 0.3986
Epoch 7/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7936 - loss:
0.4661 - val_accuracy: 0.8322 - val_loss: 0.3977
Epoch 8/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8057 - loss:
0.4464 - val_accuracy: 0.8322 - val_loss: 0.3929
Epoch 9/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8232 - loss:
0.4316 - val_accuracy: 0.8462 - val_loss: 0.3881
Epoch 10/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8328 - loss:
0.4220 - val_accuracy: 0.8462 - val_loss: 0.3878
Epoch 11/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8295 - loss:
0.4202 - val_accuracy: 0.8531 - val_loss: 0.3837
Epoch 12/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8391 - loss:
0.4120 - val_accuracy: 0.8462 - val_loss: 0.3836
Epoch 13/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8360 - loss:
0.3959 - val_accuracy: 0.8531 - val_loss: 0.3821
Epoch 14/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8176 - loss:
0.4084 - val_accuracy: 0.8462 - val_loss: 0.3770
Epoch 15/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8343 - loss:
```

```
0.4136 - val_accuracy: 0.8462 - val_loss: 0.3815
Epoch 16/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8288 - loss:
0.4137 - val_accuracy: 0.8462 - val_loss: 0.3814
Epoch 17/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8090 - loss:
0.4318 - val_accuracy: 0.8462 - val_loss: 0.3856
Epoch 18/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8454 - loss:
0.3822 - val_accuracy: 0.8462 - val_loss: 0.3863
Epoch 19/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8377 - loss:
0.4013 - val_accuracy: 0.8531 - val_loss: 0.3789
Epoch 20/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8456 - loss:
0.3778 - val_accuracy: 0.8531 - val_loss: 0.3773
Epoch 21/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8156 - loss:
0.4283 - val_accuracy: 0.8531 - val_loss: 0.3857
Epoch 22/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8446 - loss:
0.3684 - val_accuracy: 0.8531 - val_loss: 0.3872
Epoch 23/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8625 - loss:
0.3392 - val_accuracy: 0.8392 - val_loss: 0.3830
Epoch 24/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8524 - loss:
0.3888 - val_accuracy: 0.8531 - val_loss: 0.3787
Epoch 25/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8371 - loss:
0.3774 - val_accuracy: 0.8392 - val_loss: 0.3881
Epoch 26/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8528 - loss:
0.3761 - val_accuracy: 0.8392 - val_loss: 0.3788
Epoch 27/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8532 - loss:
0.3660 - val_accuracy: 0.8531 - val_loss: 0.3846
Epoch 28/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8494 - loss:
0.3844 - val_accuracy: 0.8531 - val_loss: 0.3827
Epoch 29/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8406 - loss:
0.3679 - val_accuracy: 0.8462 - val_loss: 0.3893
Epoch 30/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8451 - loss:
0.3659 - val_accuracy: 0.8531 - val_loss: 0.3827
Epoch 31/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8520 - loss:
0.3701 - val_accuracy: 0.8531 - val_loss: 0.3814
```

```
Epoch 32/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8392 - loss:
0.3686 - val_accuracy: 0.8531 - val_loss: 0.3833
Epoch 33/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8522 - loss:
0.3582 - val_accuracy: 0.8531 - val_loss: 0.3846
Epoch 34/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8504 - loss:
0.3743 - val_accuracy: 0.8531 - val_loss: 0.3870
Epoch 35/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8622 - loss:
0.3349 - val_accuracy: 0.8531 - val_loss: 0.3847
Epoch 36/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8375 - loss:
0.3691 - val_accuracy: 0.8531 - val_loss: 0.3856
Epoch 37/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8617 - loss:
0.3335 - val_accuracy: 0.8462 - val_loss: 0.3852
Epoch 38/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8523 - loss:
0.3645 - val_accuracy: 0.8531 - val_loss: 0.3827
Epoch 39/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8452 - loss:
0.3633 - val_accuracy: 0.8462 - val_loss: 0.3856
Epoch 40/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8633 - loss:
0.3322 - val_accuracy: 0.8531 - val_loss: 0.3858
Epoch 41/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8604 - loss:
0.3598 - val_accuracy: 0.8531 - val_loss: 0.3879
Epoch 42/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8531 - loss:
0.3479 - val_accuracy: 0.8462 - val_loss: 0.3895
Epoch 43/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8407 - loss:
0.3577 - val_accuracy: 0.8531 - val_loss: 0.3862
Epoch 44/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8777 - loss:
0.3241 - val_accuracy: 0.8462 - val_loss: 0.3863
Epoch 45/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8193 - loss:
0.3994 - val_accuracy: 0.8531 - val_loss: 0.3924
Epoch 46/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8334 - loss:
0.3799 - val_accuracy: 0.8462 - val_loss: 0.3863
Epoch 47/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8413 - loss:
0.3703 - val_accuracy: 0.8531 - val_loss: 0.3872
Epoch 48/50
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8534 - loss: 0.3525 - val_accuracy: 0.8392 - val_loss: 0.3896
Epoch 49/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8310 - loss: 0.3862 - val_accuracy: 0.8462 - val_loss: 0.3909
Epoch 50/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8724 - loss: 0.3351 - val_accuracy: 0.8462 - val_loss: 0.3850
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8117 - loss: 0.4185

Test Loss: 0.4368, Test Accuracy: 0.8045
```

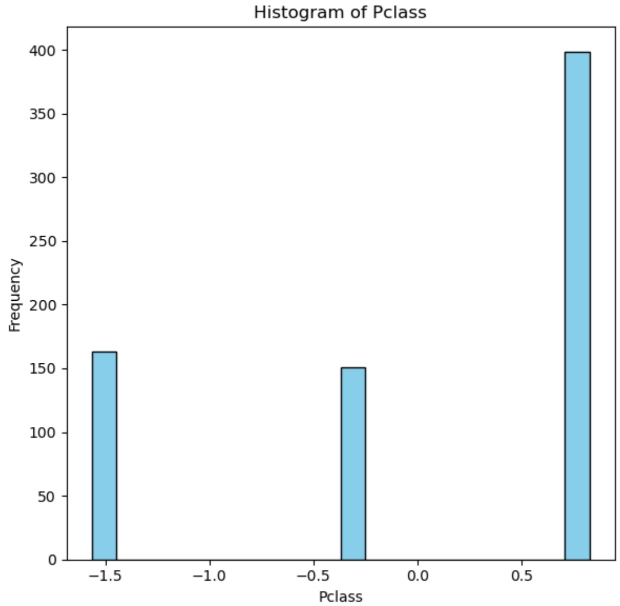
Surrogate Model - MLPClassifier

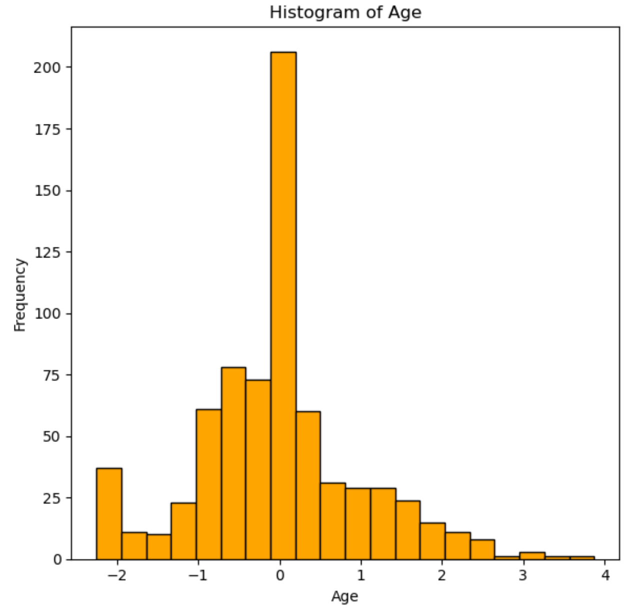
```
# Train a surrogate model (MLPClassifier)
surrogate_model = MLPClassifier(hidden_layer_sizes=(32,), activation='logistic', random_state=1, max_iter=1000).fit(X_train, y_train)
print('Accuracy (MLPClassifier): ' + str(surrogate_model.score(X_train, y_train)))

Accuracy (MLPClassifier): 0.800561797752809
```

Partial Dependence Plots (PDP) and Individual Conditional Expectation (ICE) plots

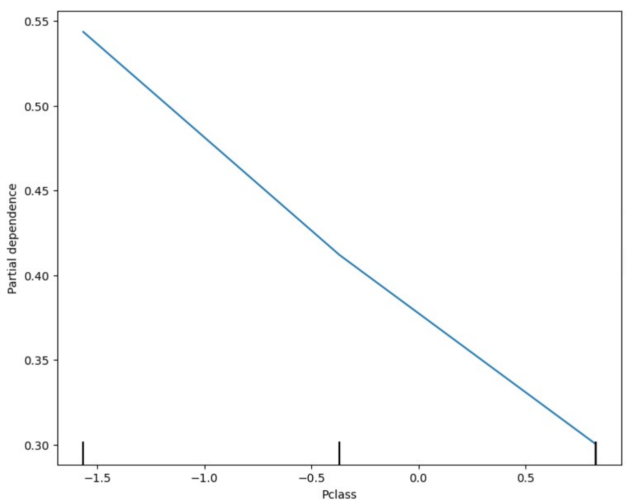
```
# Partial Dependence Plots (PDP) Function
def plot_pdp(surrogate_model, X_train, features_to_analyze):
    print("\nGenerating Partial Dependence Plots (PDP) for features:", features_to_analyze)
    fig, ax = plt.subplots(1, len(features_to_analyze), figsize=(15, 6), constrained_layout=True)
    for i, feature in enumerate(features_to_analyze):
        PartialDependenceDisplay.from_estimator(
            surrogate_model,  # The trained surrogate model (RandomForestClassifier)
            X_train,  # Training data
            features=[X_train.columns.get_loc(feature)],  # Single feature for PDP
            kind="average",  # PDP only
            ax=ax[i] if len(features_to_analyze) > 1 else ax,
            grid_resolution=50,
        )
        ax[i].set_title(f"PDP for {feature}")
    plt.show()

# Individual Conditional Expectation (ICE) Plots Function
def plot_ice(surrogate_model, X_train, features_to_analyze):
    print("\nGenerating Individual Conditional Expectation (ICE) Plots
```

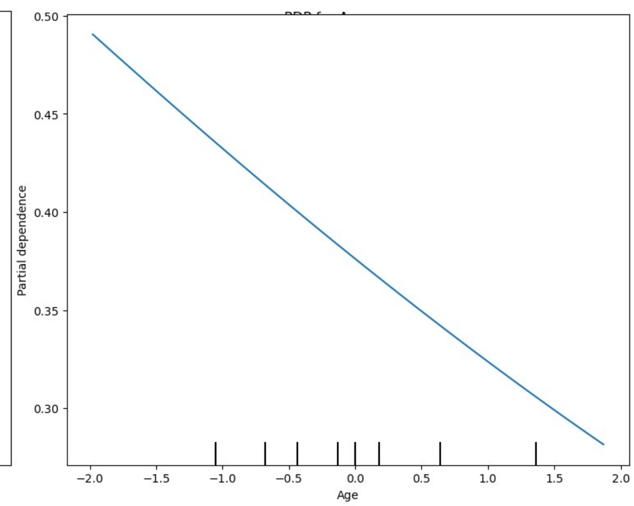
```
for features:", features_to_analyze)
    fig, ax = plt.subplots(1, len(features_to_analyze), figsize=(15,
6), constrained_layout=True)
    for i, feature in enumerate(features_to_analyze):
        PartialDependenceDisplay.from_estimator(
            surrogate_model,  # The trained surrogate model
(RandomForestClassifier)
            X_train,  # Training data
            features=[X_train.columns.get_loc(feature)],  # Single
feature for ICE
            kind="both",  # PDP and ICE
            ax=ax[i] if len(features_to_analyze) > 1 else ax,
            grid_resolution=50,
        )
        ax[i].set_title(f"ICE and PDP for {feature}")
    plt.show()

# Call PDP and ICE plot functions
features_to_analyze = ["Pclass", "Age"]

# Plot histograms for features_to_analyze
plt.figure(figsize=(12, 6))
for i, feature in enumerate(features_to_analyze):
    plt.subplot(1, len(features_to_analyze), i + 1)
    plt.hist(X_train[feature], bins=20, edgecolor='black',
color='skyblue' if i % 2 == 0 else 'orange')
    plt.title(f'Histogram of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

plot_pdp(surrogate_model, X_train, features_to_analyze)
plot_ice(surrogate_model, X_train, features_to_analyze)
```



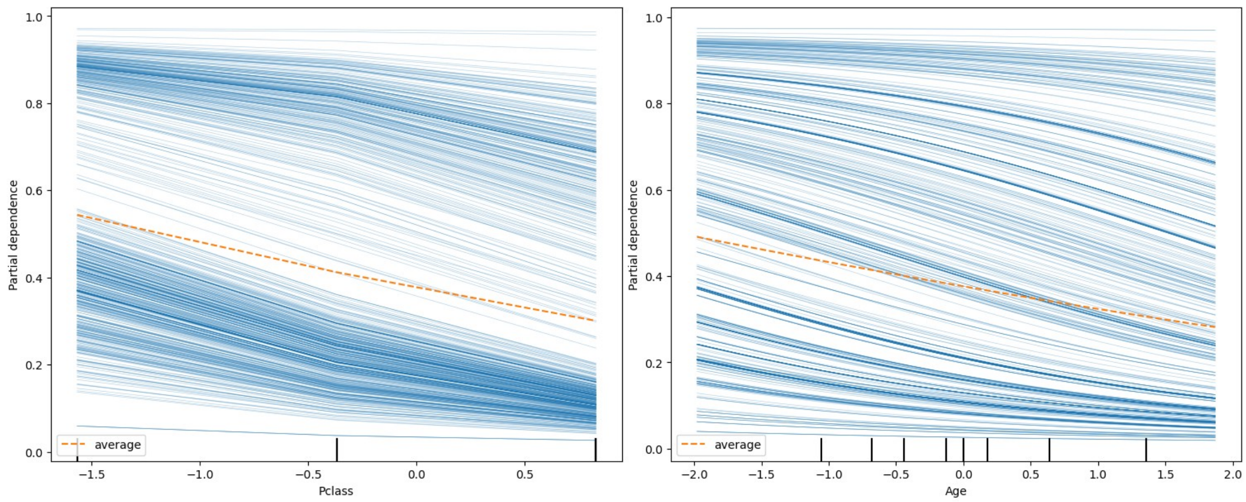



```
Generating Partial Dependence Plots (PDP) for features: ['Pclass', 'Age']
```






```
Generating Individual Conditional Expectation (ICE) Plots for features: ['Pclass', 'Age']
```



The variables `Pclass` (passenger class) and `Age` were chosen for their relevance in modeling survival outcomes on the Titanic dataset. `Pclass` is a categorical variable that captures the socioeconomic status of passengers, an essential factor influencing survival probability during a disaster. Historically, passengers in higher classes often had better access to lifeboats and safety provisions, making this variable critical for understanding survival disparities. On the other hand, `Age` represents an individual's stage of life, directly tied to survival priorities during emergencies, where children and younger individuals might be given precedence in rescue efforts.

Insights from Partial Dependence Plots (PDPs)

The **PDPs** for `Pclass` and `Age` reveal distinct global trends in their relationship with the predicted survival probability. The PDP for `Pclass` shows a **negative monotonic trend**, indicating that higher classes (lower numeric values in `Pclass`) are associated with increased survival probabilities. This aligns with the historical context of the Titanic disaster, where first-class passengers had better access to lifeboats compared to those in third class. Similarly, the PDP for `Age` displays a **negative relationship** with survival probability, where younger passengers, especially children, had a higher likelihood of survival. This trend reflects the "women and children first" policy that was partly followed during the evacuation.

While PDPs provide a useful global perspective, they average the effects across all passengers, which can obscure individual variations or interactions between features.

Insights from Individual Conditional Expectation (ICE) Plots

The **ICE plots** add granularity to the PDP analysis by showing how `Pclass` and `Age` affect the survival probability for individual passengers. For `Pclass`, the ICE plots reveal consistent negative slopes for most passengers, confirming that higher classes are universally beneficial for survival. However, there are slight deviations in the slopes, indicating that the effect of `Pclass` might vary slightly for certain individuals, potentially due to interactions with other features like `Sex` or `SibSp`.

The ICE plots for `Age` similarly show a predominantly negative trend, where survival probability decreases with increasing age. However, individual instances exhibit variability, with some passengers showing less sensitivity to age changes. For instance, middle-aged passengers in specific contexts may experience a smaller decline in survival probability compared to those in lower classes.

Combined Insights and Key Takeaways

The analysis highlights the critical role of both `Pclass` and `Age` in determining survival outcomes. PDPs provide a **broad average perspective**, confirming the overall trends: higher socioeconomic status and younger age improve survival probabilities. Meanwhile, ICE plots uncover **individual-level nuances**, showcasing heterogeneity in how these variables impact survival across different passengers.

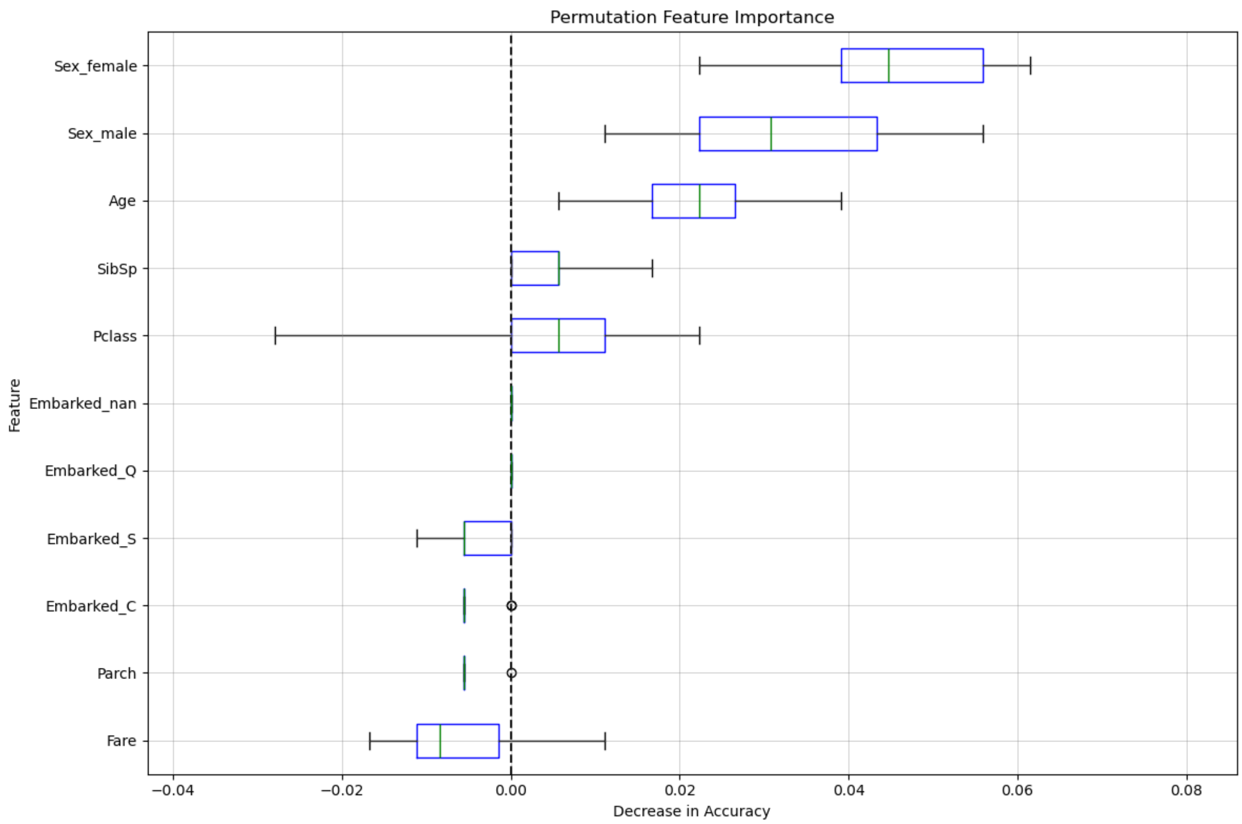
By combining PDPs and ICE plots, we gain a comprehensive understanding of the model's behavior. `Pclass` strongly reflects the structural inequalities during the Titanic disaster, while `Age` emphasizes the prioritization of specific demographic groups. These insights ensure the interpretability of the model and help align its predictions with historical and contextual expectations.

Permutation Feature Importance (PFI)

```
# Compute Permutation Feature Importance
def compute_pfi(model, X_test, y_test, feature_names):
    pfi_result = permutation_importance(
        model, X_test, y_test, n_repeats=10, random_state=42,
scoring='accuracy'
    )

    # Convert PFI results into a DataFrame for better visualization
    importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': pfi_result.importances_mean,
        'Std': pfi_result.importances_std
    })

    # Sort features by importance
    importance_df = importance_df.sort_values(by='Importance',
ascending=False)

    print("\nPermutation Feature Importance:\n", importance_df)
    return importance_df

# Plot Permutation Feature Importance as a Boxplot
def plot_pfi(model, X_test, y_test, feature_names):
    result = permutation_importance(
        model, X_test, y_test, scoring='accuracy', n_repeats=10,
random_state=42, n_jobs=2
    )

    sorted_importances_idx = result.importances_mean.argsort()
    importances = pd.DataFrame(
        result.importances[sorted_importances_idx].T,
        columns=[feature_names[i] for i in sorted_importances_idx]
```

```
    )

    ax = importances.plot.box(
        vert=False, whis=10, figsize=(12, 8),
        color=dict(boxes="blue", whiskers="black", medians="green",
caps="black")
    )
    ax.axvline(x=0, color="k", linestyle="--")

    # Add faint grey lines across the graph for each feature
    for i in range(len(importances.columns)):
        plt.axhline(y=i + 1, color="grey", linestyle="-",
linewidth=0.5, alpha=0.5)

    # Add faint grey lines upwards from the x-axis ticks
    xticks = ax.get_xticks()
    for tick in xticks:
        plt.axvline(x=tick, color="grey", linestyle="-",
linewidth=0.5, alpha=0.5)

    # Set the x-axis limits
    ax.set_xlim(left=0 - 0.5 * (ax.get_xlim()[1] - 0))

    # Set the x-axis limits
    ax.set_xlabel("Decrease in Accuracy")
    ax.set_ylabel("Feature")
    ax.set_title("Permutation Feature Importance")
    plt.tight_layout()
    plt.show()

feature_names = X_test.columns
importance_df = compute_pfi(surrogate_model, X_test, y_test,
feature_names)
plot_pfi(surrogate_model, X_test, y_test, feature_names)
```

```
Permutation Feature Importance:
          Feature  Importance       Std
5      Sex_female    0.045251  0.011571
6        Sex_male    0.031285  0.014177
1             Age    0.021788  0.009497
2           SibSp    0.005028  0.005270
0          Pclass    0.003911  0.013232
8      Embarked_Q    0.000000  0.000000
10   Embarked_nan    0.000000  0.000000
9      Embarked_S   -0.003911  0.003577
7      Embarked_C   -0.004469  0.002235
3           Parch   -0.005028  0.001676
4            Fare   -0.006145  0.008815
```



Permutation Feature Importance Results

The **Permutation Feature Importance (PFI)** results, visualized in the boxplot above, highlight the relative contributions of each feature to the model's predictive performance for Titanic survival outcomes. `Sex_female` is identified as the most critical feature, with a mean importance score of 0.045 and a standard deviation of 0.012. This underscores its significant influence on survival predictions, reflecting historical biases in rescue operations that prioritized women and children. The second most influential feature is `Sex_male`, with an importance score of 0.031 and a higher standard deviation of 0.014, reinforcing the strong yet slightly less consistent impact of gender on survival probabilities.

The variable `Age` follows as the third most important feature, with an importance score of 0.022 and a lower standard deviation of 0.009, reflecting its consistent influence on survival. Younger individuals, especially children, were often prioritized, aligning with historical records. The variables `SibSp` (number of siblings/spouses aboard) and `Pclass` (passenger class) show lower importance scores of 0.005 and 0.004, respectively, suggesting their more limited, yet still meaningful, impact on survival outcomes. These results align with expectations that family connections and class influenced access to lifeboats, albeit less strongly than gender or age.

Interestingly, features such as `Embarked_C`, `Embarked_S`, and `Fare` exhibit negative importance scores, implying that their randomization slightly improved the model's performance. This suggests that these features may introduce noise or have weak or non-linear relationships with the target variable. The variables `Embarked_Q` and `Embarked_nan` have zero importance, indicating no measurable effect on the model’s predictions.

The boxplot further highlights variability in feature importance scores across permutations. Features like `Sex_female` and `Age` exhibit tight whiskers, indicating consistent importance across permutations, while features such as `Pclass` display more variability, reflecting context-dependent effects.

Explain what the term "important" means when using the PFI method.

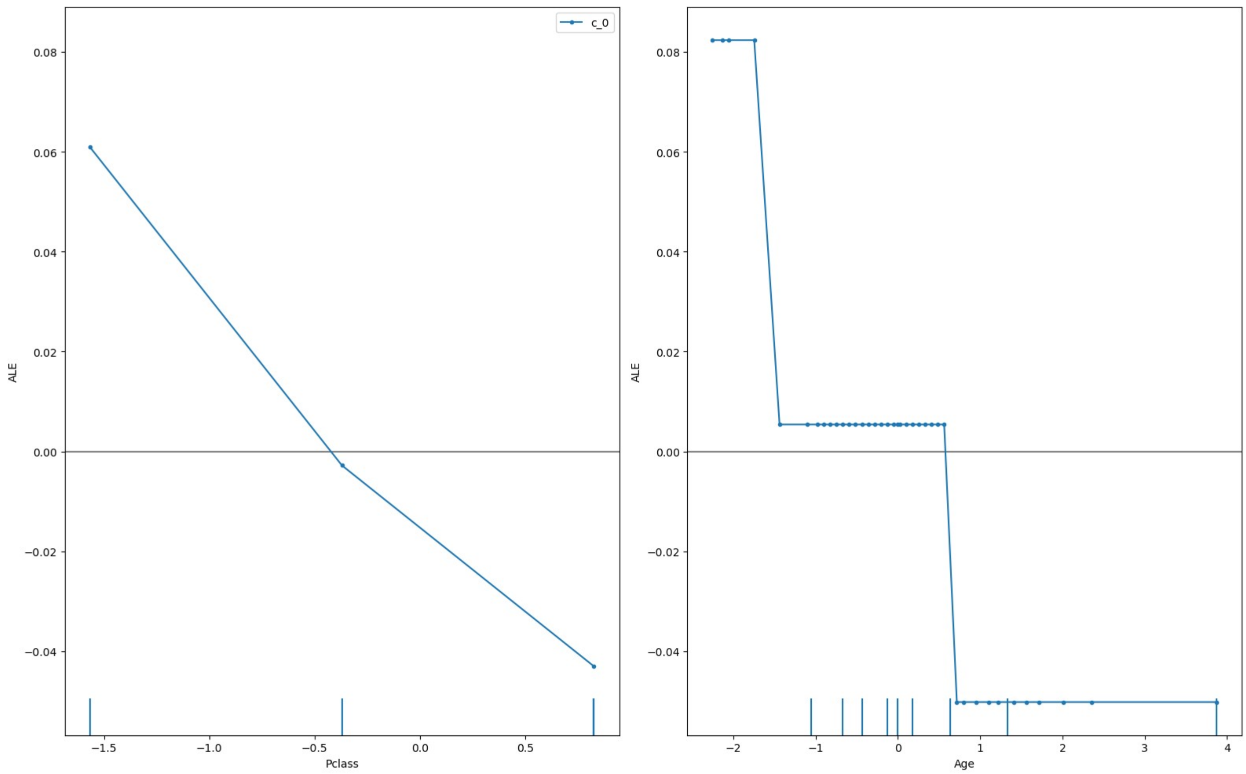
In the context of PFI, **importance** measures the extent to which a feature contributes to the model's predictive accuracy. This is quantified by observing the increase (or decrease) in error when a feature's values are randomly permuted while keeping all other features constant. A high importance score suggests that the feature provides critical information for predictions, as its randomization significantly degrades the model's performance. Conversely, a low or negative score implies that the feature's contribution is minimal or may even act as noise.

For the current results, the high importance scores of `Sex_female` and `Sex_male` highlight their dominant roles in capturing survival disparities based on gender. The consistent importance of `Age` reflects the prioritization of younger individuals in survival efforts. On the other hand, the negligible or negative importance scores for features like `Embarked_C` and `Fare` suggest that these variables either contribute weakly to the model or may have indirect or non-linear relationships with survival. The PFI results thus offer a nuanced understanding of feature relevance, capturing both their direct and indirect effects on model performance.

Accumulated Local Effects (ALE)

```
# Combine features and target for context if needed
data = pd.concat([X_train, y_train], axis=1)

# Define feature names
feature_names = X_train.columns

# Ensure valid input for ALE explainer
X_train_array = X_train.to_numpy()  # Convert to NumPy array to avoid
warnings

# Create and compute ALE explainer
ale_explainer = ALE(surrogate_model.predict,
feature_names=feature_names)
ale_explanation = ale_explainer.explain(X_train_array)

# Plot ALE for selected features
plot_ale(
    ale_explanation,
    features=["Pclass", "Age"],  # Select specific features
    n_cols=2,  # Arrange plots in 2 columns for better visualization
    fig_kw={'figwidth': 16, 'figheight': 10}  # Adjust figure size for
clarity
)
```

```
array([[<Axes: xlabel='Pclass', ylabel='ALE'>,
        <Axes: xlabel='Age', ylabel='ALE'>]], dtype=object)
```



Comparing ALE and PDP for `Pclass` and `Age`

The **Accumulated Local Effects (ALE)** plots and **Partial Dependence Plots (PDPs)** provide complementary perspectives on the model's behavior in predicting Titanic survival outcomes. Both tools aim to interpret the influence of features on the model, but their methodologies highlight distinct aspects of the relationships between the features and the predictions.

Accumulated Local Effects (ALE)

The ALE plots for `Pclass` and `Age` reveal localized trends in how these features impact survival probabilities. For `Pclass`, the ALE plot shows a strong negative relationship, where survival likelihood decreases as passenger class increases (lower numeric values indicate higher classes, with 1 being the first class). The steep decline emphasizes the significant advantage of being in a higher class, reflecting historical rescue priorities where first-class passengers were more likely to survive.

Similarly, for `Age`, the ALE plot demonstrates a sharp and non-linear relationship. Younger passengers (negative standardized values) have markedly higher survival probabilities, with the effect dropping abruptly for older passengers. This aligns with real-world events where younger individuals, particularly children, were given priority during rescue operations. The localized approach of ALE highlights the sharp transitions in survival probabilities for specific age ranges, offering a nuanced understanding of the model's behavior.

Partial Dependence Plots (PDPs)

The PDPs for `Pclass` and `Age` provide a global perspective on feature effects. For `Pclass`, the PDP similarly shows a negative trend, reinforcing the inverse relationship between class and survival likelihood. However, the smoothness of the PDP masks localized variations and interactions that are evident in the ALE plot. For instance, the PDP averages the impact of passenger class across all data points, potentially smoothing out critical distinctions between specific class ranges.

The PDP for `Age` highlights a consistent decline in survival probabilities as age increases, with younger passengers exhibiting higher survival rates. While this trend aligns with the ALE results, the PDP's global averaging approach fails to capture the sharp transitions and localized effects observed in the ALE plot, such as the steep drop-off for specific age groups.

Key Differences and Insights

The primary distinction between ALE and PDP lies in their interpretability. The ALE plots provide a localized view of feature effects, offering insights into the variations within specific ranges of the features. This is particularly valuable for understanding sharp transitions or non-linear relationships, such as the significant drop in survival probabilities for older passengers or the steep decline in survival likelihood with increasing `Pclass`.

In contrast, PDPs offer a broader, global perspective by averaging the effects across all instances. While they provide a straightforward summary of feature relationships, they may obscure localized nuances and interactions, as seen in the smoother trends for both `Pclass` and `Age`.

Key Takeaways

Both ALE and PDP agree on the general trends for `Pclass` and `Age`: higher-class passengers and younger individuals have better survival outcomes. However, the ALE plots add depth by revealing localized behaviors and sharp transitions that the PDPs overlook. Together, these tools provide a comprehensive understanding of the model's behavior, with PDPs offering an accessible overview and ALE enhancing interpretability by uncovering detailed, localized effects.

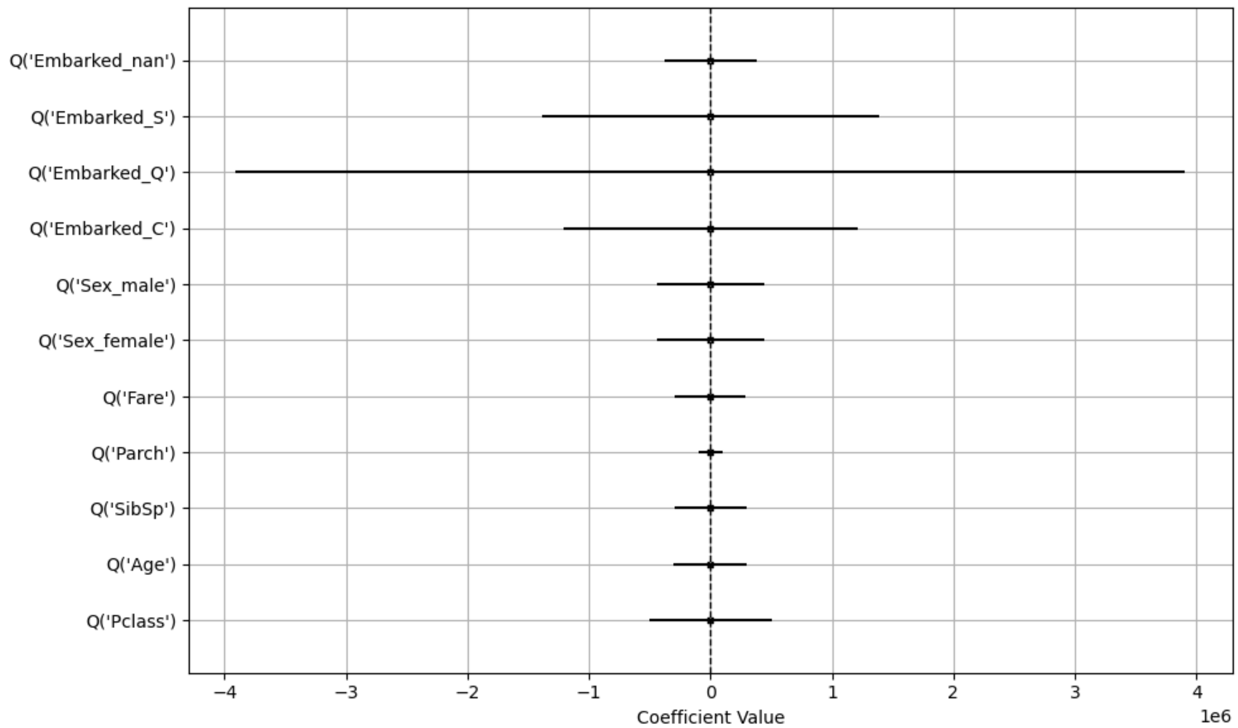
Global Surrogates

```python
# Get predictions from the neural network surrogate model
NN_labels = surrogate_model.predict(X_train)
X_train['NN_labels'] = NN_labels

# Prepare formula for logistic regression analysis
all_columns = " + ".join([f"Q('{col}')" for col in X_train.columns[:-
1]])  # Exclude NN_labels
my_formula = f"NN_labels ~ {all_columns}"

# Train logistic regression surrogate model
logistic = smf.glm(formula=my_formula, family=sm.families.Binomial(),
data=X_train).fit()
print(logistic.summary())

# Predict using the logistic regression model
y_pred = logistic.predict(X_train)
y_class = [0 if x < 0.5 else 1 for x in y_pred]
```

```python
# Calculate accuracy of the surrogate model
score = accuracy_score(y_class, X_train['NN_labels'])
print(f"Logistic Regression Surrogate Accuracy: {score}")
```

```
                 Generalized Linear Model Regression Results

==============================================================================
========
Dep. Variable:              NN_labels   No. Observations:
712
Model:                            GLM   Df Residuals:
702
Model Family:                Binomial   Df Model:
9
Link Function:                  Logit   Scale:
1.0000
Method:                          IRLS   Log-Likelihood:
nan
Date:                Fri, 17 Jan 2025   Deviance:
3.9547e-09
Time:                        17:08:46   Pearson chi2:
1.98e-09
No. Iterations:                    31   Pseudo R-squ. (CS):
nan
Covariance Type:            nonrobust

==============================================================================
===============
                        coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
---------------
Intercept          -283.3624    7.7e+05     -0.000      1.000   -
1.51e+06    1.51e+06
Q('Pclass')        -387.2768   2.59e+05     -0.001      0.999   -
5.08e+05    5.07e+05
Q('Age')           -238.3064   1.54e+05     -0.002      0.999   -
3.01e+05    3.01e+05
Q('SibSp')         -229.5399    1.5e+05     -0.002      0.999   -
2.95e+05    2.95e+05
Q('Parch')          -49.3698   4.88e+04     -0.001      0.999   -
9.57e+04    9.56e+04
Q('Fare')           158.7766   1.48e+05      0.001      0.999    -
2.9e+05    2.91e+05
Q('Sex_female')     348.4167   2.24e+05      0.002      0.999   -
4.39e+05     4.4e+05
Q('Sex_male')      -348.4167   2.24e+05     -0.002      0.999    -
4.4e+05    4.39e+05
Q('Embarked_C')      33.6186   6.17e+05   5.45e-05      1.000   -
1.21e+06    1.21e+06
```

```
Q('Embarked_Q')       23.1477   1.99e+06   1.16e-05      1.000   -
3.91e+06    3.91e+06
Q('Embarked_S')      -39.0481   7.07e+05  -5.52e-05      1.000   -
1.39e+06    1.39e+06
Q('Embarked_nan')    -45.9638   1.92e+05     -0.000      1.000   -
3.76e+05    3.76e+05
======================================================================
===============
Logistic Regression Surrogate Accuracy: 1.0

# Analyze coefficients of the logistic regression model
err_series = logistic.params - logistic.conf_int()[0]
coef_df = pd.DataFrame({
    'coef': pd.to_numeric(logistic.params.values[1:],
errors='coerce'),
    'err': pd.to_numeric(err_series.values[1:], errors='coerce'),
    'varname': err_series.index.values[1:]
})

# Plot coefficient values with error bars
fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(coef_df['varname'], coef_df['coef'], xerr=coef_df['err'],
color='none', edgecolor=None)
ax.scatter(y=coef_df['varname'], x=coef_df['coef'], marker='s', s=10,
color='black')
ax.axvline(x=0, linestyle='--', color='black', linewidth=1)

ax.set_xlabel('Coefficient Value')
ax.set_ylabel('')
ax.grid(True)

plt.tight_layout()
plt.show()
```



Analyze the Surrogate Model's Effectiveness and Discuss When Such Approximations Are Helpful

The surrogate model, represented by a generalized linear model (GLM) trained on the predictions of the neural network, demonstrates perfect accuracy in approximating the neural network's predictions, as indicated by an accuracy score of 1.0. This highlights its ability to capture the behavior of the original neural network for the Titanic dataset. However, a deeper examination of the coefficient values provides nuanced insights into its interpretability and limitations.

Coefficients and Interpretability

The coefficient plot shows the relative contributions of each feature to the neural network's predictions as approximated by the GLM. For instance, the coefficients for `Sex_female` (348.42) and `Sex_male` (-348.42) suggest that gender plays a critical role in survival predictions, reflecting real-world rescue priorities where women were prioritized. Similarly, `Fare` has a positive coefficient (158.78), indicating that passengers who paid higher fares (likely in higher classes) were more likely to survive, aligning with historical accounts of class-based rescue advantages.

Conversely, features such as `Pclass` (-387.28), `Age` (-238.31), and `SibSp` (-229.54) exhibit significant negative coefficients, suggesting that older passengers, individuals in lower classes, and those with more siblings or spouses onboard had reduced survival probabilities. These findings align with historical trends and the model's learned patterns. Features such as `Embarked` and `Parch` have smaller coefficients, indicating weaker or more context-dependent relationships with survival outcomes.

Limitations of the Surrogate Model

While the surrogate model achieves perfect accuracy in approximating the neural network, its reliance on linear relationships may oversimplify complex interactions present in the original model. For example, the neural network might capture non-linear dependencies or interactions between features like `Pclass` and `Fare` that are not represented in the GLM. This limitation is particularly evident in features with near-zero or statistically insignificant coefficients, such as `Embarked` and `Parch`, which might have more nuanced effects in the neural network.

The extreme magnitude of some coefficients, coupled with their wide confidence intervals, highlights another limitation. These values suggest potential instability or overfitting in the surrogate model, where the coefficients may be sensitive to small changes in the data or the training process.

Usefulness of Surrogate Models

Despite its limitations, the GLM surrogate provides valuable interpretability for understanding the neural network's behavior. It allows for clear visualization of feature importance and directionality, making it easier to communicate insights to stakeholders. This is particularly useful in contexts where explainability is critical, such as compliance or ethical decision-making.

However, care must be taken to acknowledge the trade-offs involved. The surrogate model provides a simplified view of the neural network's behavior and may not capture all interactions or non-linearities. As such, it should be complemented with other interpretability techniques, such as PDPs, ICE plots, or ALE plots, to gain a more comprehensive understanding of the model.

Conclusion

The surrogate model effectively captures the general trends in the neural network's predictions while providing interpretable insights into feature contributions. It is a powerful tool for distilling complex model behavior into accessible and actionable information. However, its simplifications and potential limitations must be carefully communicated to ensure that stakeholders do not misinterpret its findings. By combining surrogate modeling with other interpretability techniques, a more holistic understanding of the model can be achieved.

Project ARI3205 Interpretable AI for Deep Learning Models *(Part 2)*

Name: Sean David Muscat

ID No: 0172004L

Importing Necessary Libraries

In [1]:
```
# Check and install required libraries from the libraries.json file
import json

# Read the libraries from the text file
with open('../Libraries/Part2_Lib.json', 'r') as file:
    libraries = json.load(file)

# ANSI escape codes for colored output
GREEN = "\033[92m"  # Green text
RED = "\033[91m"    # Red text
RESET = "\033[0m"   # Reset to default color

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        try:
            # Attempt to import the library
            __import__(import_name)
            print(f"[{GREEN}✓{RESET}] Library '{lib}' is already installed.")
        except ImportError:
            # If import fails, try to install the library
            print(f"[{RED}✗{RESET}] Library '{lib}' is not installed. Installing...
            %pip install {lib}

# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Import necessary libraries for data analysis and modeling
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler,  OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.neural_network import MLPClassifier
import lime
from lime.lime_tabular import LimeTabularExplainer
import shap
from anchor import anchor_tabular
```

```
# Suppress specific warnings
warnings.filterwarnings("ignore", message="X does not have valid feature names")
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

```
[✓] Library 'tensorflow' is already installed.
[✓] Library 'matplotlib' is already installed.
[✓] Library 'pandas' is already installed.
[✓] Library 'numpy' is already installed.
[✓] Library 'lime' is already installed.
C:\Users\Sean Muscat\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qb
z5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\tqdm\auto.py:21: Tqdm
Warning: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
[✓] Library 'shap' is already installed.
[✓] Library 'anchor' is already installed.
```

In [2]:

```
# Define the filenames
train_filename = '../Datasets/Titanic/train.csv'
test_filename = '../Datasets/Titanic/test.csv'
gender_submission_filename = '../Datasets/Titanic/gender_submission.csv'

# Load the datasets
try:
    train_data = pd.read_csv(train_filename)
    test_data = pd.read_csv(test_filename)
    gender_submission_data = pd.read_csv(gender_submission_filename)
    print(f"'{train_filename}' dataset loaded successfully.")
    print(f"'{test_filename}' dataset loaded successfully.")
    print(f"'{gender_submission_filename}' dataset loaded successfully.")
except FileNotFoundError as e:
    print(f"Error: {e.filename} was not found. Please ensure it is in the correct di
    exit()
except pd.errors.EmptyDataError as e:
    print(f"Error: {e.filename} is empty.")
    exit()
except pd.errors.ParserError as e:
    print(f"Error: There was a problem parsing {e.filename}. Please check the file f
    exit()

# Dataset insights
print("\nTrain Dataset Overview:")
print(train_data.info())
print("\nTrain Dataset Statistical Summary:")
print(train_data.describe())

print("\nTest Dataset Overview:")
print(test_data.info())
print("\nTest Dataset Statistical Summary:")
print(test_data.describe())

print("\nGender Submission Dataset Overview:")
print(gender_submission_data.info())
```

```
'../Datasets/Titanic/train.csv' dataset loaded successfully.
'../Datasets/Titanic/test.csv' dataset loaded successfully.
'../Datasets/Titanic/gender_submission.csv' dataset loaded successfully.

Train Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

Train Dataset Statistical Summary:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200

Test Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
```

```
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
None

Test Dataset Statistical Summary:
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344   35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429   55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000    0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000    7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000   14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000   31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000  512.329200

Gender Submission Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Survived     418 non-null    int64
dtypes: int64(2)
memory usage: 6.7 KB
None
```

Feed-Forward Neural Network

In [3]:

```
# Load the Titanic dataset
train_data = pd.read_csv('../Datasets/Titanic/train.csv')

# Preprocessing
# Separate features and target
y = train_data['Survived']  # Target
X = train_data.drop(columns=['Survived', 'PassengerId', 'Name', 'Ticket', 'Cabin'])

# Handle categorical variables with one-hot encoding
categorical_features = ['Sex', 'Embarked']
one_hot_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
categorical_encoded = one_hot_encoder.fit_transform(X[categorical_features])
categorical_encoded_df = pd.DataFrame(categorical_encoded, columns=one_hot_encoder.g

# Drop original categorical columns and append the encoded columns
X = X.drop(columns=categorical_features)
X = pd.concat([X.reset_index(drop=True), categorical_encoded_df.reset_index(drop=Tru

# Handle missing values with mean imputation
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Standardize the features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_imputed), columns=X.columns)
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ranc
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Training data shape: (712, 11)
Test data shape: (179, 11)
```

In [4]:

```
# Build the feed-forward neural network
model = Sequential([
    Input(shape=(X_train.shape[1],)),  # Define input shape explicitly
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metri

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_size=32

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
```

```
Epoch 1/50
18/18 [==============================] - 1s 13ms/step - loss: 0.6211 - accuracy: 0.
7346 - val_loss: 0.5474 - val_accuracy: 0.7762
Epoch 2/50
18/18 [==============================] - 0s 3ms/step - loss: 0.5318 - accuracy: 0.7
803 - val_loss: 0.4718 - val_accuracy: 0.7972
Epoch 3/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4837 - accuracy: 0.7
873 - val_loss: 0.4397 - val_accuracy: 0.7972
Epoch 4/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4585 - accuracy: 0.7
961 - val_loss: 0.4183 - val_accuracy: 0.8182
Epoch 5/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4414 - accuracy: 0.8
049 - val_loss: 0.4095 - val_accuracy: 0.8182
Epoch 6/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4312 - accuracy: 0.8
137 - val_loss: 0.4031 - val_accuracy: 0.8252
Epoch 7/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4217 - accuracy: 0.8
207 - val_loss: 0.4020 - val_accuracy: 0.8252
Epoch 8/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4155 - accuracy: 0.8
190 - val_loss: 0.3981 - val_accuracy: 0.8252
Epoch 9/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4128 - accuracy: 0.8
278 - val_loss: 0.3926 - val_accuracy: 0.8112
Epoch 10/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4064 - accuracy: 0.8
295 - val_loss: 0.3952 - val_accuracy: 0.8252
Epoch 11/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4056 - accuracy: 0.8
278 - val_loss: 0.4027 - val_accuracy: 0.8182
Epoch 12/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4012 - accuracy: 0.8
278 - val_loss: 0.3951 - val_accuracy: 0.8322
Epoch 13/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3985 - accuracy: 0.8
348 - val_loss: 0.3984 - val_accuracy: 0.8182
Epoch 14/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3951 - accuracy: 0.8
383 - val_loss: 0.3963 - val_accuracy: 0.8252
Epoch 15/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3942 - accuracy: 0.8
383 - val_loss: 0.3897 - val_accuracy: 0.8252
Epoch 16/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3926 - accuracy: 0.8
348 - val_loss: 0.4031 - val_accuracy: 0.8252
Epoch 17/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3880 - accuracy: 0.8
401 - val_loss: 0.3934 - val_accuracy: 0.8112
Epoch 18/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3860 - accuracy: 0.8
313 - val_loss: 0.3901 - val_accuracy: 0.8252
Epoch 19/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3859 - accuracy: 0.8
383 - val_loss: 0.3992 - val_accuracy: 0.8392
Epoch 20/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3843 - accuracy: 0.8
366 - val_loss: 0.3961 - val_accuracy: 0.8182
```

```
Epoch 21/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3826 - accuracy: 0.8
418 - val_loss: 0.3932 - val_accuracy: 0.8112
Epoch 22/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3795 - accuracy: 0.8
436 - val_loss: 0.3968 - val_accuracy: 0.8182
Epoch 23/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3785 - accuracy: 0.8
401 - val_loss: 0.3990 - val_accuracy: 0.8182
Epoch 24/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3789 - accuracy: 0.8
436 - val_loss: 0.3930 - val_accuracy: 0.8112
Epoch 25/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3776 - accuracy: 0.8
418 - val_loss: 0.3973 - val_accuracy: 0.8462
Epoch 26/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3744 - accuracy: 0.8
471 - val_loss: 0.3942 - val_accuracy: 0.8322
Epoch 27/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3733 - accuracy: 0.8
436 - val_loss: 0.3979 - val_accuracy: 0.8182
Epoch 28/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3722 - accuracy: 0.8
453 - val_loss: 0.3979 - val_accuracy: 0.8322
Epoch 29/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3700 - accuracy: 0.8
436 - val_loss: 0.3948 - val_accuracy: 0.8322
Epoch 30/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3697 - accuracy: 0.8
453 - val_loss: 0.4008 - val_accuracy: 0.8252
Epoch 31/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3674 - accuracy: 0.8
401 - val_loss: 0.4008 - val_accuracy: 0.8252
Epoch 32/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3675 - accuracy: 0.8
436 - val_loss: 0.3976 - val_accuracy: 0.8322
Epoch 33/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3665 - accuracy: 0.8
366 - val_loss: 0.3964 - val_accuracy: 0.8392
Epoch 34/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3628 - accuracy: 0.8
471 - val_loss: 0.3966 - val_accuracy: 0.8392
Epoch 35/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3632 - accuracy: 0.8
489 - val_loss: 0.3994 - val_accuracy: 0.8392
Epoch 36/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3635 - accuracy: 0.8
453 - val_loss: 0.3971 - val_accuracy: 0.8322
Epoch 37/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3627 - accuracy: 0.8
418 - val_loss: 0.3993 - val_accuracy: 0.8252
Epoch 38/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3601 - accuracy: 0.8
436 - val_loss: 0.4007 - val_accuracy: 0.8392
Epoch 39/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3582 - accuracy: 0.8
471 - val_loss: 0.4058 - val_accuracy: 0.8392
Epoch 40/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3567 - accuracy: 0.8
471 - val_loss: 0.3996 - val_accuracy: 0.8392
```

```
Epoch 41/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3560 - accuracy: 0.8
418 - val_loss: 0.4045 - val_accuracy: 0.8322
Epoch 42/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3578 - accuracy: 0.8
471 - val_loss: 0.4043 - val_accuracy: 0.8392
Epoch 43/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3544 - accuracy: 0.8
471 - val_loss: 0.4044 - val_accuracy: 0.8392
Epoch 44/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3556 - accuracy: 0.8
471 - val_loss: 0.4084 - val_accuracy: 0.8322
Epoch 45/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3536 - accuracy: 0.8
436 - val_loss: 0.4014 - val_accuracy: 0.8392
Epoch 46/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3519 - accuracy: 0.8
489 - val_loss: 0.4081 - val_accuracy: 0.8392
Epoch 47/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3520 - accuracy: 0.8
453 - val_loss: 0.4007 - val_accuracy: 0.8392
Epoch 48/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3502 - accuracy: 0.8
453 - val_loss: 0.4066 - val_accuracy: 0.8322
Epoch 49/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3507 - accuracy: 0.8
418 - val_loss: 0.4093 - val_accuracy: 0.8392
Epoch 50/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3476 - accuracy: 0.8
489 - val_loss: 0.4084 - val_accuracy: 0.8392
6/6 [==============================] - 0s 1ms/step - loss: 0.4422 - accuracy: 0.843
6
Test Loss: 0.4422, Test Accuracy: 0.8436
```

Surrogate Model - MLPClassifier

In [5]:
```python
# Train a surrogate model (MLPClassifier)
surrogate_model = MLPClassifier(hidden_layer_sizes=(32,), activation='logistic', rar
print('Accuracy (MLPClassifier): ' + str(surrogate_model.score(X_train, y_train)))
```

```
Accuracy (MLPClassifier): 0.800561797752809
```

PART 2.1

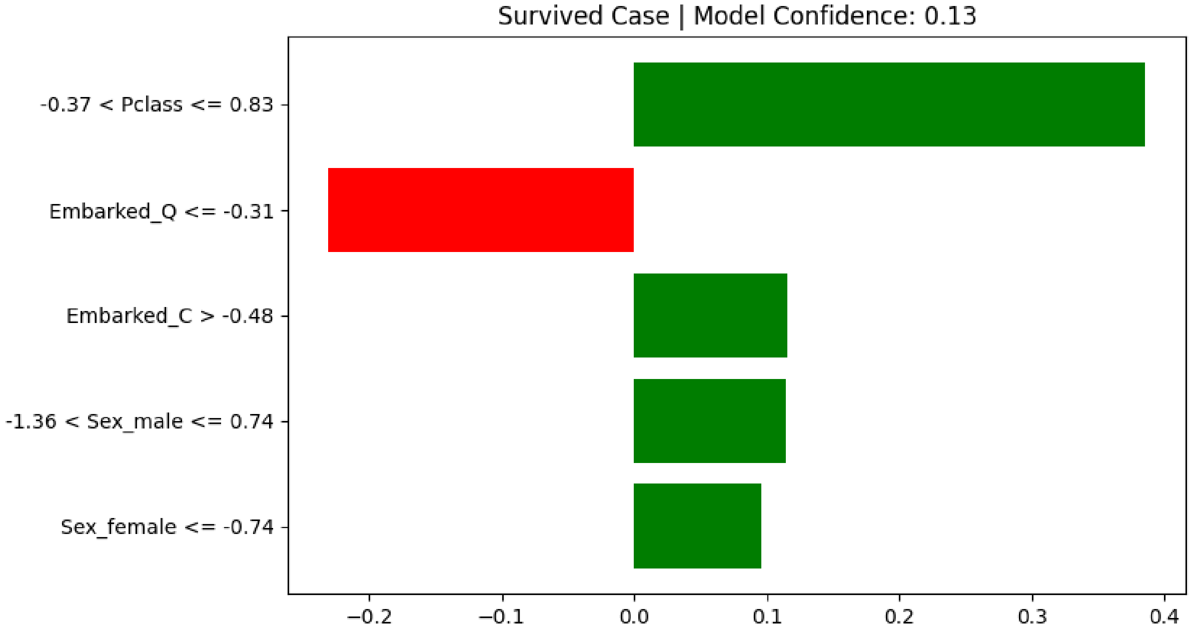
Set up the LIME explainer

In [6]:
```python
# Function to visualize LIME explanations as a bar plot
def lime_exp_as_pyplot(exp, label=1, figsize=(8, 5)):
    exp_list = exp.as_list(label=label)
    fig, ax = plt.subplots(figsize=figsize)

    # Extract feature names and importance values
    vals = [x[1] for x in exp_list]
    names = [x[0] for x in exp_list]

    # Reverse for descending order of feature importance
    vals.reverse()
```

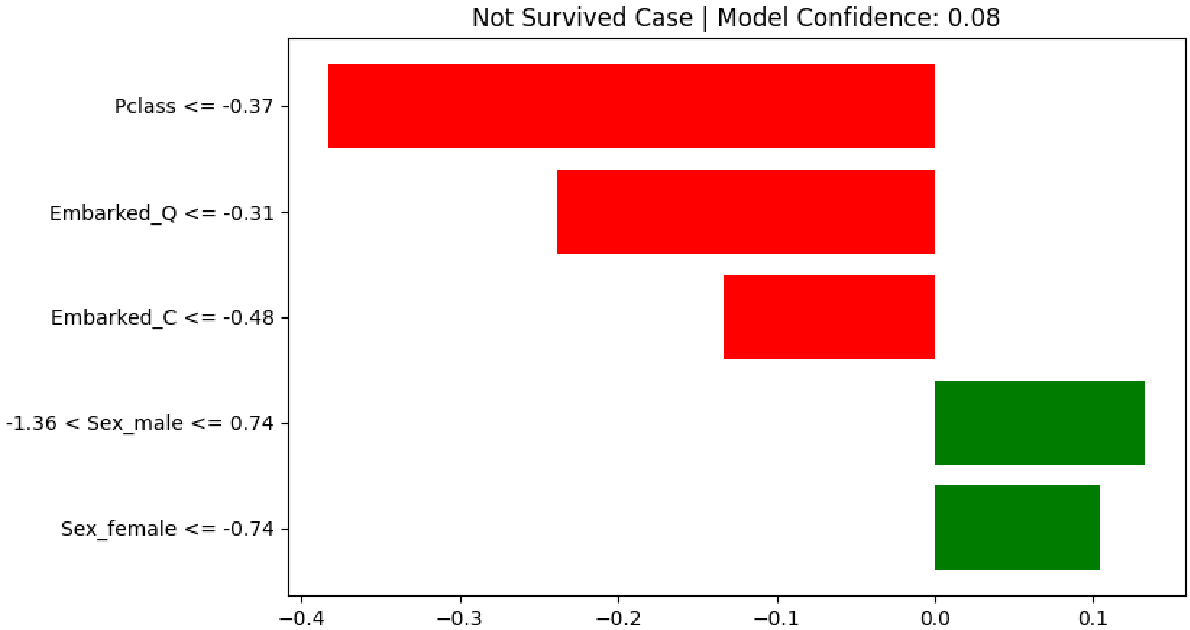
```
    names.reverse()

    # Color the bars: green for positive, red for negative
    colors = ['green' if x > 0 else 'red' for x in vals]

    # Positions for the bars
    pos = np.arange(len(exp_list)) + .5

    # Plot the bars
    ax.barh(pos, vals, align='center', color=colors)
    plt.yticks(pos, names)

    return fig, ax

# Wrap the Keras model's prediction function for LIME
def predict_proba(X):
    """Custom function for LIME to get model predictions."""
    prob_class_1 = model.predict(X)  # Predicted probability for class 1
    prob_class_0 = 1 - prob_class_1  # Predicted probability for class 0
    return np.hstack((prob_class_0, prob_class_1))  # Combine probabilities

# Initialize the LIME Tabular Explainer
explainer = lime.lime_tabular.LimeTabularExplainer(
    X_train.to_numpy(),
    feature_names=X_train.columns.to_list(),
    class_names=['Not Survived', 'Survived'],
    discretize_continuous=True,
    random_state=42
)

# Example instance index for "Survived" and "Not Survived"
survived_idx = np.where(y_test.to_numpy() == 1)[0][0]
not_survived_idx = np.where(y_test.to_numpy() == 0)[0][0]

# Explanation for "Survived" instance
survived_exp = explainer.explain_instance(
    X_test.iloc[survived_idx].to_numpy(),
    predict_proba,
    num_features=5,
    top_labels=1
)

# Dynamically find the label for "Survived" instance
available_label = list(survived_exp.local_exp.keys())[0]  # Pick the first available
print(f"Available label for Survived instance: {available_label}")

# Visualize explanation for the "Survived" instance
f, ax = lime_exp_as_pyplot(survived_exp, label=available_label)
survived_confidence = model.predict(X_test.iloc[survived_idx:survived_idx + 1].to_nu
ax.set_title(f'Survived Case | Model Confidence: {survived_confidence:.2f}')
plt.show()

# Explanation for "Not Survived" instance
not_survived_exp = explainer.explain_instance(
    X_test.iloc[not_survived_idx].to_numpy(),
    predict_proba,
    num_features=5,
    top_labels=1
)
```

```
# Dynamically find the label for "Not Survived" instance
available_label = list(not_survived_exp.local_exp.keys())[0]  # Pick the first avail
print(f"Available label for Not Survived instance: {available_label}")

# Visualize explanation for the "Not Survived" instance
f, ax = lime_exp_as_pyplot(not_survived_exp, label=available_label)
not_survived_confidence = model.predict(X_test.iloc[not_survived_idx:not_survived_ic
ax.set_title(f'Not Survived Case | Model Confidence: {not_survived_confidence:.2f}')
plt.show()
```

```
157/157 [==============================] - 0s 780us/step
Available label for Survived instance: 0
1/1 [==============================] - 0s 21ms/step
```




```
157/157 [==============================] - 0s 754us/step
Available label for Not Survived instance: 0
1/1 [==============================] - 0s 22ms/step
```



Part 2.1 b

LIME (Local Interpretable Model-agnostic Explanations) is an algorithm designed to provide interpretability for complex, black-box models by approximating their local decision boundaries. It operates by perturbing the input data around a specific instance and observing the model’s outputs for these slightly modified samples. The results of these perturbations are used to fit an interpretable surrogate model, typically a linear model, that captures the behaviour of the black-box model within the vicinity of the specific instance.

For our Titanic dataset, LIME highlights the contributions of individual features to the model's decision-making process for specific instances. For example, in the visualisations above:

- In the "Survived" case, features such as "Pclass" (passenger class) and "Sex_female" have significant positive contributions to the prediction, as indicated by the green bars. On the other hand, features like "Embarked_Q" negatively influence the outcome, as indicated by the red bars. This suggests that higher socio-economic status and being female are strongly associated with survival, whereas embarking from certain locations may decrease survival probability.
- In the "Not Survived" case, features such as "Pclass" negatively influence the prediction, suggesting that lower socio-economic status correlates with non-survival. Similarly, factors like "Parch" (number of parents/children aboard) might also contribute negatively. Positive influences like "Sex_female" show a mitigating factor, indicating that the model considers gender but not sufficiently to alter the outcome.

By presenting feature contributions as weights (positive or negative) for each instance, LIME provides a clear interpretative framework. The approximations, while not perfectly reflecting the global decision boundary, give useful insights into how the model uses features locally. This interpretability is crucial for datasets like Titanic, where fairness and historical biases (e.g., gender and class disparity) can be critically analysed.

Part 2.2

Adding SHAP to Explain Model Predictions

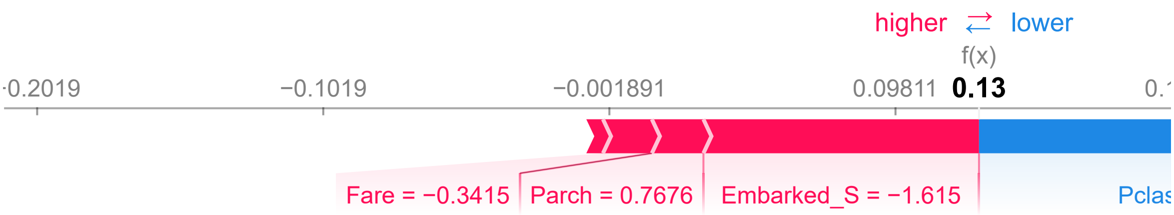
In [7]:

```
# Use SHAP's DeepExplainer for neural networks
explainer = shap.KernelExplainer(model.predict, X_train[:100])  # Use a small sample

# Calculate SHAP values for a set of instances
shap_values = explainer.shap_values(X_test[:10])  # Explaining the first 10 samples

# Visualize the SHAP values for the first test sample (e.g., index 0)
shap.initjs()

# Reshape SHAP values if necessary
shap_values_reshaped = shap_values[0].reshape(1, -1)

# Now plot with reshaped values
shap.force_plot(
    explainer.expected_value[0],
    shap_values_reshaped[0],  # SHAP values for the first sample (class 0)
    X_test.iloc[0],           # Actual features for the first sample
```

```
    feature_names=X.columns  # Feature names
)
```

```
4/4 [==============================] - 0s 997us/step
  0%|          | 0/10 [00:00<?, ?it/s]
1/1 [==============================] - 0s 21ms/step
6394/6394 [==============================] - 6s 884us/step
 10%|█         | 1/10 [00:09<01:28,  9.79s/it]
1/1 [==============================] - 0s 21ms/step
6394/6394 [==============================] - 5s 848us/step
 20%|██        | 2/10 [00:18<01:13,  9.16s/it]
1/1 [==============================] - 0s 21ms/step
6394/6394 [==============================] - 5s 823us/step
 30%|███       | 3/10 [00:27<01:02,  8.88s/it]
1/1 [==============================] - 0s 20ms/step
6394/6394 [==============================] - 5s 814us/step
 40%|████      | 4/10 [00:35<00:52,  8.76s/it]
1/1 [==============================] - 0s 20ms/step
6394/6394 [==============================] - 5s 824us/step
 50%|█████     | 5/10 [00:44<00:43,  8.66s/it]
1/1 [==============================] - 0s 19ms/step
6394/6394 [==============================] - 5s 829us/step
 60%|██████    | 6/10 [00:52<00:34,  8.63s/it]
1/1 [==============================] - 0s 23ms/step
6394/6394 [==============================] - 6s 859us/step
 70%|███████   | 7/10 [01:01<00:26,  8.71s/it]
1/1 [==============================] - 0s 23ms/step
6394/6394 [==============================] - 5s 836us/step
 80%|████████  | 8/10 [01:10<00:17,  8.77s/it]
1/1 [==============================] - 0s 24ms/step
6394/6394 [==============================] - 6s 963us/step
 90%|█████████ | 9/10 [01:20<00:09,  9.05s/it]
1/1 [==============================] - 0s 23ms/step
6394/6394 [==============================] - 5s 797us/step
100%|██████████| 10/10 [01:28<00:00,  8.87s/it]
```

Out[7]:



Part 2.2 b

LIME works by approximating the decision boundary of a model in the vicinity of a particular instance. It perturbs the input data around the instance to generate nearby samples, evaluates the model's predictions on these samples, and fits an interpretable surrogate model, such as a linear regression, to mimic the model's behaviour locally. In the visualisations provided for LIME, we see feature contributions represented as positive or negative weights, indicating whether each feature supports or opposes a specific outcome. For example, "Pclass" and "Sex_female" contribute significantly to predicting survival, as denoted by the green bars, while features like "Embarked_Q" negatively influence the same prediction. LIME's strength lies in its simplicity and ability to provide intuitive explanations for specific instances. However, its reliance on local approximations can sometimes result in less accurate explanations for complex, non-linear models.

SHAP, on the other hand, is based on Shapley values from cooperative game theory, ensuring that feature attributions are both consistent and theoretically sound. SHAP calculates the contribution of each feature to the prediction by considering all possible combinations of feature presence and absence, making it more computationally intensive than LIME. In the SHAP visualisation, feature contributions are displayed on a scale from negative (red, reducing the prediction) to positive (blue, increasing the prediction), with the sum of contributions equalling the model's prediction. For example, "Embarked_C" strongly increases the prediction score for survival, while "Embarked_S" has a significant negative impact. SHAP's additive nature and consistency make it particularly suitable for gaining a more holistic and globally consistent understanding of a model's behaviour.

The distinctions between SHAP and LIME are evident when compared. LIME approximates the decision boundary for a particular instance and its surroundings, concentrating only on local explanations. This method might not have the capability to fully capture the intricacies of the global model, although being quicker and frequently intuitive. SHAP, on the other hand, guarantees that feature contributions are uniform throughout the dataset, offering both local and global interpretability. SHAP becomes more resource-intensive but more resilient as a result. In conclusion, SHAP provides a deeper and more trustworthy understanding of feature attributions at the expense of greater computational effort, whereas LIME is useful for rapid and local insights.

Part 2.3

Implementing Anchors to interpret model predictions in specific cases

In [8]:
```
# Define the explainer
anchor_explainer = anchor_tabular.AnchorTabularExplainer(
    class_names=['Not Survived', 'Survived'],  # Adjust based on the binary target
    feature_names=X.columns.tolist(),
    train_data=X_train.values,  # Use training data for the explainer
    categorical_names={i: one_hot_encoder.categories_[i] for i in range(len(categori
)

# Define the prediction function for the neural network
pred_fn = lambda x: surrogate_model.predict(x)

# Select an instance to explain (example: first test instance)
instance_to_explain = X_test.iloc[0].values

# Explain the instance using Anchors
exp = anchor_explainer.explain_instance(
```

```
    instance_to_explain,
    pred_fn,
    threshold=0.95
)

# Display the results
print('Anchor: %s' % (' AND '.join(exp.names())))
print('Precision: %.2f' % exp.precision())
print('Coverage: %.2f' % exp.coverage())
exp.show_in_notebook()
```

```
Anchor: Sex_female <= -0.74 AND Pclass = female
Precision: 0.98
Coverage: 0.40
```

Example
Pclass = female
Age = C
-0.47 < SibSp <= 0.43
Parch > -0.47
-0.36 < Fare <= -0.03
Sex_female <= -0.74
-1.36 < Sex_male <= 0.74
Embarked_C > -0.48
Embarked_Q <= -0.31
Embarked_S <= -1.61
Embarked_nan <= -0.05

A.I. pre...

Not Survived

Explanation of A.I. prediction

If ALL of these are true:

✓ Sex_female <= -0.74

✓ Pclass = female

The A.I. will predict Not Survived **97.7%** of the time

› Examples where the A.I. agent predicts Not Survived

› Examples where the A.I. agent DOES NOT predict Not Survived

Part 2.3 b

Each of SHAP, LIME, and Anchors offers a different way to interpret machine learning predictions. By precisely determining the minimal circumstances that "anchor" a prediction, anchors concentrate on rule-based explanations. Because Anchors generate straightforward, understandable principles, they are therefore very interpretable. However, by disregarding interactions that do not fall under the designated parameters, these rules may oversimplify the decision-making process.

An option is provided by LIME, which approximates the local decision boundary by fitting a linear model and perturbing input data. Individual feature contributions, such "Pclass" or "Sex_female," are highlighted as either in favour of or against a prediction. Although LIME works well for producing concise and understandable explanations, its unpredictability in perturbations may cause variability and make it difficult to adequately reflect non-linear interactions in the model.

In contrast, SHAP ensures consistency and equity in attribution by employing Shapley values to assign additive contributions to features. Features like "Embarked_C" greatly support survival predictions, whereas "Embarked_S" lowers the likelihood of survival, as shown in SHAP visualisations. Although SHAP is more computationally demanding and may be too detailed for consumers, it excels at delivering both local and global explanations.

To sum up, SHAP is best suited for thorough and trustworthy feature attributions, LIME is helpful for quick and easy local explanations, and Anchors are perfect for producing straightforward and actionable rules. The model's complexity and the particular requirements for interpretability will determine which of these approaches is best.

Project ARI3205 Interpretable AI for Deep Learning Models *(Part 3.1)*

Name: Sean David Muscat

ID No: 0172004L

Importing Necessary Libraries

In [1]:
```
# Check and install required libraries from the libraries.json file
import json

# Read the libraries from the text file
with open('../Libraries/Part3.1_Lib.json', 'r') as file:
    libraries = json.load(file)

# ANSI escape codes for colored output
GREEN = "\033[92m"  # Green text
RED = "\033[91m"    # Red text
RESET = "\033[0m"   # Reset to default color

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        try:
            # Attempt to import the library
            __import__(import_name)
            print(f"[{GREEN}✓{RESET}] Library '{lib}' is already installed.")
        except ImportError:
            # If import fails, try to install the library
            print(f"[{RED}✗{RESET}] Library '{lib}' is not installed. Installin
            %pip install {lib}

# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Import necessary libraries for data analysis and modeling
import warnings
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
# Alibi imports for the MNIST example
import tensorflow as tf
tf.get_logger().setLevel(40)  # suppress deprecation messages
tf.compat.v1.disable_v2_behavior()  # disable TF2 behaviour as Alibi code still
tf.compat.v1.reset_default_graph()
tf.keras.backend.clear_session()
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
```

```
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler,  OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.neural_network import MLPClassifier
from alibi.explainers import Counterfactual
import numpy as np


# Suppress specific warnings
warnings.filterwarnings("ignore", message="X does not have valid feature names")
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

```
[✓] Library 'tensorflow' is already installed.
[✓] Library 'scikit-learn' is already installed.
[✓] Library 'matplotlib' is already installed.
[✓] Library 'seaborn' is already installed.
[✓] Library 'pandas' is already installed.
[✓] Library 'numpy' is already installed.
[✓] Library 'statsmodels' is already installed.
[✗] Library 'alibi[tensorflow]' is not installed. Installing...
Requirement already satisfied: alibi[tensorflow] in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (0.9.6)
Requirement already satisfied: numpy<2.0.0,>=1.16.2 in c:\users\sean muscat\appda
ta\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\l
ocal-packages\python311\site-packages (from alibi[tensorflow]) (1.23.5)
Requirement already satisfied: pandas<3.0.0,>=1.0.0 in c:\users\sean muscat\appda
ta\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\l
ocal-packages\python311\site-packages (from alibi[tensorflow]) (2.2.2)
Requirement already satisfied: scikit-learn<2.0.0,>=1.0.0 in c:\users\sean muscat
\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localc
ache\local-packages\python311\site-packages (from alibi[tensorflow]) (1.5.0)
Requirement already satisfied: spacy<4.0.0,>=2.0.0 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from spacy[lookups]<4.0.0,>=2.0.0->alibi[te
nsorflow]) (3.7.5)
Requirement already satisfied: blis<0.8.0 in c:\users\sean muscat\appdata\local\p
ackages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packa
ges\python311\site-packages (from alibi[tensorflow]) (0.7.11)
Requirement already satisfied: scikit-image<0.23,>=0.17.2 in c:\users\sean muscat
\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localc
ache\local-packages\python311\site-packages (from alibi[tensorflow]) (0.22.0)
Requirement already satisfied: requests<3.0.0,>=2.21.0 in c:\users\sean muscat\ap
pdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcach
e\local-packages\python311\site-packages (from alibi[tensorflow]) (2.32.2)
Requirement already satisfied: Pillow<11.0,>=5.4.1 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from alibi[tensorflow]) (10.3.0)
Requirement already satisfied: attrs<24.0.0,>=19.2.0 in c:\users\sean muscat\appd
ata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from alibi[tensorflow]) (23.2.0)
Requirement already satisfied: scipy<2.0.0,>=1.1.0 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from alibi[tensorflow]) (1.13.1)
Requirement already satisfied: matplotlib<4.0.0,>=3.0.0 in c:\users\sean muscat\a
ppdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcac
he\local-packages\python311\site-packages (from alibi[tensorflow]) (3.9.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in c:\users\sean muscat
\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localc
ache\local-packages\python311\site-packages (from alibi[tensorflow]) (4.11.0)
Requirement already satisfied: dill<0.4.0,>=0.3.0 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from alibi[tensorflow]) (0.3.9)
Requirement already satisfied: transformers<5.0.0,>=4.7.0 in c:\users\sean muscat
\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localc
ache\local-packages\python311\site-packages (from alibi[tensorflow]) (4.45.1)
Requirement already satisfied: tqdm<5.0.0,>=4.28.1 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from alibi[tensorflow]) (4.66.5)
Requirement already satisfied: tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0 in c:\us
ers\sean muscat\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n
2kfra8p0\localcache\local-packages\python311\site-packages (from alibi[tensorflo
```

```
w]) (2.12.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\sean muscat\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from matplotlib<4.0.0,>=3.0.0->alibi[tensorflo
w]) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\users\sean muscat\appdata\local
\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pac
kages\python311\site-packages (from matplotlib<4.0.0,>=3.0.0->alibi[tensorflow])
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from matplotlib<4.0.0,>=3.0.0->alibi[tensorf
low]) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from matplotlib<4.0.0,>=3.0.0->alibi[tensorf
low]) (1.4.5)
Requirement already satisfied: packaging>=20.0 in c:\users\sean muscat\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib<4.0.0,>=3.0.0->alibi[tensorflo
w]) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\sean muscat\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from matplotlib<4.0.0,>=3.0.0->alibi[tensorflo
w]) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\sean muscat\appda
ta\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\l
ocal-packages\python311\site-packages (from matplotlib<4.0.0,>=3.0.0->alibi[tenso
rflow]) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\sean muscat\appdata\local
\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pac
kages\python311\site-packages (from pandas<3.0.0,>=1.0.0->alibi[tensorflow]) (202
4.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\sean muscat\appdata\loc
al\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-p
ackages\python311\site-packages (from pandas<3.0.0,>=1.0.0->alibi[tensorflow]) (2
024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sean muscat\a
ppdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcac
he\local-packages\python311\site-packages (from requests<3.0.0,>=2.21.0->alibi[te
nsorflow]) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sean muscat\appdata\local
\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pac
kages\python311\site-packages (from requests<3.0.0,>=2.21.0->alibi[tensorflow])
(3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from requests<3.0.0,>=2.21.0->alibi[tensorfl
ow]) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from requests<3.0.0,>=2.21.0->alibi[tensorfl
ow]) (2024.2.2)
Requirement already satisfied: networkx>=2.8 in c:\users\sean muscat\appdata\loca
l\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pa
ckages\python311\site-packages (from scikit-image<0.23,>=0.17.2->alibi[tensorflo
w]) (3.3)
Requirement already satisfied: imageio>=2.27 in c:\users\sean muscat\appdata\loca
l\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pa
ckages\python311\site-packages (from scikit-image<0.23,>=0.17.2->alibi[tensorflo
```

```
w]) (2.36.1)
Requirement already satisfied: tifffile>=2022.8.12 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from scikit-image<0.23,>=0.17.2->alibi[tens
orflow]) (2024.12.12)
Requirement already satisfied: lazy_loader>=0.3 in c:\users\sean muscat\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from scikit-image<0.23,>=0.17.2->alibi[tensorf
low]) (0.4)
Requirement already satisfied: joblib>=1.2.0 in c:\users\sean muscat\appdata\loca
l\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pa
ckages\python311\site-packages (from scikit-learn<2.0.0,>=1.0.0->alibi[tensorflo
w]) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\sean muscat\appda
ta\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\l
ocal-packages\python311\site-packages (from scikit-learn<2.0.0,>=1.0.0->alibi[ten
sorflow]) (3.5.0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in c:\users\sean musca
t\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\local
cache\local-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[loo
kups]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in c:\users\sean musca
t\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\local
cache\local-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[loo
kups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in c:\users\sean muscat
\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localc
ache\local-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[look
ups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]<4.
0.0,>=2.0.0->alibi[tensorflow]) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\sean muscat\appd
ata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]
<4.0.0,>=2.0.0->alibi[tensorflow]) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]<4.
0.0,>=2.0.0->alibi[tensorflow]) (8.2.5)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in c:\users\sean muscat\appda
ta\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\l
ocal-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]<
4.0.0,>=2.0.0->alibi[tensorflow]) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]<4.
0.0,>=2.0.0->alibi[tensorflow]) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in c:\users\sean muscat\ap
pdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcach
e\local-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookup
s]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in c:\users\sean muscat\appda
ta\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\l
ocal-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]<
4.0.0,>=2.0.0->alibi[tensorflow]) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]<4.
```

```
0.0,>=2.0.0->alibi[tensorflow]) (0.13.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in c:\users\s
ean muscat\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra
8p0\localcache\local-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->
spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.9.2)
Requirement already satisfied: jinja2 in c:\users\sean muscat\appdata\local\packa
ges\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages
\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0-
>alibi[tensorflow]) (3.1.4)
Requirement already satisfied: setuptools in c:\users\sean muscat\appdata\local\p
ackages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packa
ges\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.
0.0->alibi[tensorflow]) (75.8.0)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in c:\users\sean muscat\ap
pdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcach
e\local-packages\python311\site-packages (from spacy<4.0.0,>=2.0.0->spacy[lookup
s]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.4.1)
Requirement already satisfied: spacy-lookups-data<1.1.0,>=1.0.3 in c:\users\sean
muscat\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0
\localcache\local-packages\python311\site-packages (from spacy[lookups]<4.0.0,>=
2.0.0->alibi[tensorflow]) (1.0.5)
Requirement already satisfied: tensorflow-intel==2.12.0 in c:\users\sean muscat\a
ppdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcac
he\local-packages\python311\site-packages (from tensorflow!=2.6.0,!=2.6.1,<2.15.
0,>=2.0.0->alibi[tensorflow]) (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\sean muscat\appdata\loc
al\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-p
ackages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=2.6.
0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=
2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\sean muscat\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=2.
6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (24.12.23)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=
2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=
2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\sean muscat\appdata\local
\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pac
kages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.12.1)
Requirement already satisfied: jax>=0.3.15 in c:\users\sean muscat\appdata\local
\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pac
kages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.4.30)
Requirement already satisfied: libclang>=13.0.0 in c:\users\sean muscat\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=2.
6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=
```

```
2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.4.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.2
1.4,!=4.21.5,<5.0.0dev,>=3.20.3 in c:\users\sean muscat\appdata\local\packages\py
thonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python
311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.1
5.0,>=2.0.0->alibi[tensorflow]) (4.25.5)
Requirement already satisfied: six>=1.12.0 in c:\users\sean muscat\appdata\local
\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pac
kages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=2.6.0,!
=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\sean muscat\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=2.
6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=
2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=
2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (1.69.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in c:\users\sean muscat\ap
pdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcach
e\local-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorfl
ow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.12.3)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in c:\users\sea
n muscat\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p
0\localcache\local-packages\python311\site-packages (from tensorflow-intel==2.12.
0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.12.0)
Requirement already satisfied: keras<2.13,>=2.12.0 in c:\users\sean muscat\appdat
a\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\lo
cal-packages\python311\site-packages (from tensorflow-intel==2.12.0->tensorflow!=
2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.12.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\s
ean muscat\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra
8p0\localcache\local-packages\python311\site-packages (from tensorflow-intel==2.1
2.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.31.0)
Requirement already satisfied: colorama in c:\users\sean muscat\appdata\local\pac
kages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-package
s\python311\site-packages (from tqdm<5.0.0,>=4.28.1->alibi[tensorflow]) (0.4.6)
Requirement already satisfied: filelock in c:\users\sean muscat\appdata\local\pac
kages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-package
s\python311\site-packages (from transformers<5.0.0,>=4.7.0->alibi[tensorflow])
(3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in c:\users\sean musc
at\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\loca
lcache\local-packages\python311\site-packages (from transformers<5.0.0,>=4.7.0->a
libi[tensorflow]) (0.25.1)
Requirement already satisfied: pyyaml>=5.1 in c:\users\sean muscat\appdata\local
\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pac
kages\python311\site-packages (from transformers<5.0.0,>=4.7.0->alibi[tensorflo
w]) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from transformers<5.0.0,>=4.7.0->alibi[tenso
rflow]) (2024.9.11)
Requirement already satisfied: safetensors>=0.4.1 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from transformers<5.0.0,>=4.7.0->alibi[tenso
```

```
rflow]) (0.4.5)
Requirement already satisfied: tokenizers<0.21,>=0.20 in c:\users\sean muscat\app
data\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from transformers<5.0.0,>=4.7.0->alibi[t
ensorflow]) (0.20.0)
Requirement already satisfied: fsspec>=2023.5.0 in c:\users\sean muscat\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from huggingface-hub<1.0,>=0.23.2->transformer
s<5.0.0,>=4.7.0->alibi[tensorflow]) (2024.9.0)
Requirement already satisfied: language-data>=1.2 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from langcodes<4.0.0,>=3.2.0->spacy<4.0.0,>=
2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.2.0)
Requirement already satisfied: annotated-types>=0.6.0 in c:\users\sean muscat\app
data\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.
4->spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (0.7.0)
Requirement already satisfied: pydantic-core==2.23.4 in c:\users\sean muscat\appd
ata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.
4->spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.23.4)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in c:\users\sean muscat\a
ppdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcac
he\local-packages\python311\site-packages (from thinc<8.3.0,>=8.2.2->spacy<4.0.0,
>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (0.1.5)
Requirement already satisfied: click>=8.0.0 in c:\users\sean muscat\appdata\local
\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pac
kages\python311\site-packages (from typer<1.0.0,>=0.3.0->spacy<4.0.0,>=2.0.0->spa
cy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (8.1.7)
Requirement already satisfied: shellingham>=1.3.0 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from typer<1.0.0,>=0.3.0->spacy<4.0.0,>=2.0.
0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in c:\users\sean muscat\appdata\loca
l\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pa
ckages\python311\site-packages (from typer<1.0.0,>=0.3.0->spacy<4.0.0,>=2.0.0->sp
acy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (13.9.4)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in c:\users\sean muscat
\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localc
ache\local-packages\python311\site-packages (from weasel<0.5.0,>=0.1.0->spacy<4.
0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (0.20.0)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in c:\users\sean muscat\a
ppdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcac
he\local-packages\python311\site-packages (from weasel<0.5.0,>=0.1.0->spacy<4.0.
0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (7.0.5)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\sean muscat\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from jinja2->spacy<4.0.0,>=2.0.0->spacy[lookup
s]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.1.5)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from astunparse>=1.6.0->tensorflow-intel==2.
12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.45.1)
Requirement already satisfied: jaxlib<=0.4.30,>=0.4.27 in c:\users\sean muscat\ap
pdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcach
e\local-packages\python311\site-packages (from jax>=0.3.15->tensorflow-intel==2.1
2.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.4.30)
Requirement already satisfied: ml-dtypes>=0.2.0 in c:\users\sean muscat\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from jax>=0.3.15->tensorflow-intel==2.12.0->te
```

```
nsorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.2.0)
Requirement already satisfied: marisa-trie>=0.7.7 in c:\users\sean muscat\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\loc
al-packages\python311\site-packages (from language-data>=1.2->langcodes<4.0.0,>=
3.2.0->spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (1.
2.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\sean muscat\appd
ata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0-
>spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\sean muscat\ap
pdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcach
e\local-packages\python311\site-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0
->spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflow]) (2.18.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\sean muscat\appd
ata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from tensorboard<2.13,>=2.12->tensorflow
-intel==2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (2.
37.0)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in c:\users\sean mu
scat\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\lo
calcache\local-packages\python311\site-packages (from tensorboard<2.13,>=2.12->te
nsorflow-intel==2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorfl
ow]) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in c:\users\sean muscat\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from tensorboard<2.13,>=2.12->tensorflow-intel=
=2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\users
\sean muscat\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kf
ra8p0\localcache\local-packages\python311\site-packages (from tensorboard<2.13,>=
2.12->tensorflow-intel==2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi
[tensorflow]) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\sean muscat\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from tensorboard<2.13,>=2.12->tensorflow-intel=
=2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.0.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\sean muscat\app
data\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from google-auth<3,>=1.6.3->tensorboard<
2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0-
>alibi[tensorflow]) (5.5.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\sean muscat\appd
ata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from google-auth<3,>=1.6.3->tensorboard<
2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0-
>alibi[tensorflow]) (0.4.1)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\sean muscat\appdata\loca
l\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pa
ckages\python311\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.
12->tensorflow-intel==2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[te
nsorflow]) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\sean muscat\a
ppdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcac
he\local-packages\python311\site-packages (from google-auth-oauthlib<1.1,>=0.5->t
ensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow!=2.6.0,!=2.6.1,<2.1
5.0,>=2.0.0->alibi[tensorflow]) (2.0.0)
Requirement already satisfied: mdurl~=0.1 in c:\users\sean muscat\appdata\local\p
ackages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packa
ges\python311\site-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.
```

```
0.0,>=0.3.0->spacy<4.0.0,>=2.0.0->spacy[lookups]<4.0.0,>=2.0.0->alibi[tensorflo
w]) (0.1.2)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in c:\users\sean muscat\appda
ta\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\l
ocal-packages\python311\site-packages (from pyasn1-modules>=0.2.1->google-auth<3,
>=1.6.3->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow!=2.6.0,!=
2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (0.6.1)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\sean muscat\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from requests-oauthlib>=0.7.0->google-auth-oaut
hlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow!=2.
6.0,!=2.6.1,<2.15.0,>=2.0.0->alibi[tensorflow]) (3.2.2)
Note: you may need to restart the kernel to use updated packages.
```

```
C:\Users\Sean Muscat\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_
qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\tqdm\auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See http
s://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:

```
# Define the filenames
train_filename = '../Datasets/Titanic/train.csv'
test_filename = '../Datasets/Titanic/test.csv'
gender_submission_filename = '../Datasets/Titanic/gender_submission.csv'

# Load the datasets
try:
    train_data = pd.read_csv(train_filename)
    test_data = pd.read_csv(test_filename)
    gender_submission_data = pd.read_csv(gender_submission_filename)
    print(f"'{train_filename}' dataset loaded successfully.")
    print(f"'{test_filename}' dataset loaded successfully.")
    print(f"'{gender_submission_filename}' dataset loaded successfully.")
except FileNotFoundError as e:
    print(f"Error: {e.filename} was not found. Please ensure it is in the correc
    exit()
except pd.errors.EmptyDataError as e:
    print(f"Error: {e.filename} is empty.")
    exit()
except pd.errors.ParserError as e:
    print(f"Error: There was a problem parsing {e.filename}. Please check the fi
    exit()

# Dataset insights
print("\nTrain Dataset Overview:")
print(train_data.info())
print("\nTrain Dataset Statistical Summary:")
print(train_data.describe())

print("\nTest Dataset Overview:")
print(test_data.info())
print("\nTest Dataset Statistical Summary:")
print(test_data.describe())

print("\nGender Submission Dataset Overview:")
print(gender_submission_data.info())
```

```
'../Datasets/Titanic/train.csv' dataset loaded successfully.
'../Datasets/Titanic/test.csv' dataset loaded successfully.
'../Datasets/Titanic/gender_submission.csv' dataset loaded successfully.

Train Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

Train Dataset Statistical Summary:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000   
mean    446.000000    0.383838    2.308642   29.699118    0.523008   
std     257.353842    0.486592    0.836071   14.526497    1.102743   
min       1.000000    0.000000    1.000000    0.420000    0.000000   
25%     223.500000    0.000000    2.000000   20.125000    0.000000   
50%     446.000000    0.000000    3.000000   28.000000    0.000000   
75%     668.500000    1.000000    3.000000   38.000000    1.000000   
max     891.000000    1.000000    3.000000   80.000000    8.000000   

            Parch        Fare  
count  891.000000  891.000000  
mean     0.381594   32.204208  
std      0.806057   49.693429  
min      0.000000    0.000000  
25%      0.000000    7.910400  
50%      0.000000   14.454200  
75%      0.000000   31.000000  
max      6.000000  512.329200  

Test Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
```

```
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
None

Test Dataset Statistical Summary:
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344   35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429   55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000    0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000    7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000   14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000   31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000  512.329200

Gender Submission Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Survived     418 non-null    int64
dtypes: int64(2)
memory usage: 6.7 KB
None
```

Feed-Forward Neural Network

In [3]:
```python
# Load the Titanic dataset
train_data = pd.read_csv('../Datasets/Titanic/train.csv')

# Preprocessing
# Separate features and target
y = train_data['Survived']  # Target
X = train_data.drop(columns=['Survived', 'PassengerId', 'Name', 'Ticket', 'Cabin

# Handle categorical variables with one-hot encoding
categorical_features = ['Sex', 'Embarked']
one_hot_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
categorical_encoded = one_hot_encoder.fit_transform(X[categorical_features])
categorical_encoded_df = pd.DataFrame(categorical_encoded, columns=one_hot_encod

# Drop original categorical columns and append the encoded columns
X = X.drop(columns=categorical_features)
X = pd.concat([X.reset_index(drop=True), categorical_encoded_df.reset_index(drop

# Handle missing values with mean imputation
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Standardize the features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_imputed), columns=X.columns)
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Training data shape: (712, 11)
Test data shape: (179, 11)
```

In [ ]:
```
# Build the feed-forward neural network
model = Sequential([
    Input(shape=(X_train.shape[1],)),  # Define input shape explicitly
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', m

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_siz

# # Evaluate the model
# test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
# print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
```

Surrogate Model - MLPClassifier

In [5]:
```
# Train a surrogate model (MLPClassifier)
surrogate_model = MLPClassifier(hidden_layer_sizes=(32,), activation='logistic',
print('Accuracy (MLPClassifier): ' + str(surrogate_model.score(X_train, y_train)
```

```
Accuracy (MLPClassifier): 0.800561797752809
```

Part 3.1

Set up Counterfactuals

We begin by predicting labels on the test set and identifying which samples the model misclassifies. For each misclassified passenger, we record their scaled features and define a prediction function that converts our model's single sigmoid output into a two-column probability array: [p(died), p(survived)]. Alibi's counterfactual explainer then searches within specified min/max bounds for a new set of feature values that shifts the model's predicted outcome (for example, from "died" to "survived"). Finally, we compare these counterfactual features with the originals to see how small changes in attributes like Age or Fare can flip the prediction.

In [11]:
```
# 1. Make predictions on the test set
y_pred_probs = model.predict(X_test)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()

# 2. Identify misclassified samples
incorrect_indices = np.where(y_pred != y_test.values)[0]
print(f"Number of incorrectly predicted samples: {len(incorrect_indices)}")
```

```
# Make sure we have at least 2 misclassified samples
if len(incorrect_indices) < 2:
    print("Fewer than 2 misclassified samples found. Cannot generate two counter
else:
    #####################################
    # LOOP OVER THE FIRST 2 MISCLASSIFIED
    #####################################
    for i in range(2):  # generate counterfactual for the first two misclassifie
        print(f"\n*** COUNTERFACTUAL #{i+1} ***")

        # 3. Select one misclassified example
        sample_idx = incorrect_indices[i]  # pick the i-th misclassified sample
        x_test_sample = X_test.iloc[[sample_idx]].values
        actual_label = y_test.values[sample_idx]
        print(f"Sample index: {sample_idx}, Actual label: {actual_label}, Predic
        print("\nSample features (scaled):")
        display(X_test.iloc[[sample_idx]])

        # 4. Define a new predict_fn that outputs [p(died), p(survived)] for eac
        def predict_fn(x: np.ndarray) -> np.ndarray:
            if x.ndim == 1:
                x = x.reshape(1, -1)
            p_survived = model.predict(x).flatten()
            p_died = 1.0 - p_survived
            return np.vstack([p_died, p_survived]).T

        # 5. Determine feature_range from training data
        lower_bounds = X_train.min(axis=0).values
        upper_bounds = X_train.max(axis=0).values
        feature_range = (lower_bounds, upper_bounds)

        # 6. Decide on target_proba to 'flip' the original label
        desired_proba = 0.8 if actual_label == 0 else 0.2

        # 7. Instantiate the Counterfactual explainer
        cf_explainer = Counterfactual(
            predict_fn=predict_fn,
            shape=(1, X_train.shape[1]),
            target_proba=desired_proba,
            max_iter=1000,
            feature_range=feature_range,
            lam_init=1e-1,
            max_lam_steps=10,
            learning_rate_init=1e-2
        )

        # 8. Generate a counterfactual explanation
        explanation = cf_explainer.explain(x_test_sample)

        # 9. Print results
        print("\n--- Counterfactual Explanation ---")
        print("Original 2-column probability:", predict_fn(x_test_sample))
        if explanation.cf is not None:
            cf_sample = explanation.cf['X']  # shape => (1, n_features)
            print("\nCounterfactual feature values (scaled):")
            display(cf_sample)

            print("Counterfactual 2-column probability:", predict_fn(cf_sample))
```

```
            # Show the numerical difference
            changes = cf_sample[0] - x_test_sample[0]
            print("\nDifference between CF and original sample:")
            for col, diff in zip(X_test.columns, changes):
                print(f"{col}: {diff:.3f}")
        else:
            print("No counterfactual found within the specified parameters.")
```

```
Number of incorrectly predicted samples: 83

*** COUNTERFACTUAL #1 ***
Sample index: 1, Actual label: 0, Predicted: 1

Sample features (scaled):
```

	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarke
439	-0.369365	0.100109	-0.474545	-0.473674	-0.437007	-0.737695	0.737695	-0.482

```
--- Counterfactual Explanation ---
Original 2-column probability: [[0.43319923 0.5668008 ]]
No counterfactual found within the specified parameters.

*** COUNTERFACTUAL #2 ***
Sample index: 2, Actual label: 0, Predicted: 1

Sample features (scaled):
```

	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarke
840	0.827377	-0.746389	-0.474545	-0.473674	-0.488854	-0.737695	0.737695	-0.482

```
ERROR:alibi.explainers.counterfactual:No appropriate lambda range found, try decr
easing lam_init
--- Counterfactual Explanation ---
Original 2-column probability: [[0.48958385 0.51041615]]
No counterfactual found within the specified parameters.
```

This code was run multiple times, this was another of the counterfactuals given:

*** COUNTERFACTUAL #1 ***

Sample index: 0, Actual label: 1, Predicted: 0

--- Counterfactual Explanation ---

Original 2-column probability: [[0.64625597 0.353744]]

Counterfactual feature values (scaled):

array([[0.82737726, -1.394982 , 1.6170563 , 2.1170812 , -0.64842165, -0.73769516, 0.73769516, 2.074505 , 0.58823454, -1.6147097 , 1.3070657]], dtype=float32)

Counterfactual 2-column probability: [[0.75010574 0.24989426]]

Difference between CF and original sample:

Pclass: 0.000

Age: -1.395

SibSp: 1.184

Parch: 1.349

Fare: -0.307

Sex_female: -0.000

Sex_male: 0.000

Embarked_C: -0.000

Embarked_Q: 0.896

Embarked_S: -0.000

Embarked_nan: 1.354

*** COUNTERFACTUAL #2 ***

Sample index: 4, Actual label: 1, Predicted: 0

--- Counterfactual Explanation ---

Original 2-column probability: [[0.54528 0.45472002]]

Counterfactual feature values (scaled):

array([[0.82737726, -1.2068506 , 0.43296716, -0.4736736 , -0.422202 , 1.3555735 , -1.3555735 , 2.074505 , 0.4825647 , 0.6193064 , 2.7697735]], dtype=float32)

Counterfactual 2-column probability: [[0.79656625 0.20343377]]

Difference between CF and original sample:

Pclass: 0.000

Age: 0.001

SibSp: 0.000

Parch: -0.000

Fare: -0.000

Sex_female: -0.000

Sex_male: 0.000

Embarked_C: -0.000

Embarked_Q: 0.790

Embarked_S: 2.234

Embarked_nan: 2.817

Output Explanation: Counterfactual #1 (Sample index: 0, Actual label: 1, Predicted: 0) In this example, the model originally assigns a probability of approximately 64.63% to class 0 (and 35.37% to class 1). Several features are then adjusted—most notably, Age decreases substantially (by -1.395 in scaled units), while SibSp (number of siblings/spouses) and Parch (number of parents/children) both increase. Additionally, there are changes in Embarked_Q and Embarked_nan. After these modifications, the model's probability for class 0 rises to about 75.01%, moving further away from predicting the correct label of 1. This indicates that these specific adjustments to the features cause the model to become even more confident in the incorrect prediction. It suggests that age and the number of family members travelling (as encoded in SibSp and Parch) may be influential in pushing the prediction toward non-survival under this particular counterfactual setting.

Counterfactual #2 (Sample index: 4, Actual label: 1, Predicted: 0) Here, the original probabilities are roughly 54.53% for class 0 versus 45.47% for class 1—still an incorrect prediction, though the model is slightly less certain compared with Counterfactual #1. The counterfactual modifies the feature representation of passenger embarkation, with notable jumps in Embarked_Q, Embarked_S, and Embarked_nan. Despite these changes, the probability for class 0 increases further to approximately 79.66%. This outcome indicates that shifts in certain embarkation features, under the current model, do not bring the prediction closer to the correct label for this sample but instead reinforce the model's belief that the passenger did not survive.

3.1 b

Counterfactual explanations are vital because they tell us how to alter specific features in a model's input so that its prediction changes to a desired outcome. When we see how even small changes in passenger attributes (for instance, lowering their age or increasing their fare) flip the prediction from "died" to "survived," we gain insights into what the model deems crucial for its decision.

In debugging models, counterfactuals help us pinpoint problematic behaviours and potential biases. If the counterfactual requires unrealistic feature shifts—such as setting the fare far above any real-world range—then our model may be over-reliant on that feature, or it might not generalise well. We can use this knowledge to refine data preprocessing or adjust hyperparameters, ensuring our model bases decisions on more sensible factors.

Counterfactuals can direct real-world interventions from a decision-making perspective. For instance, if a passenger's survival probability increases significantly with a slight increase in fare, this indicates that socioeconomic position (as measured by fare) has a significant impact on the model. Managers, legislators, or end users can then evaluate the fairness or realism of these elements. Counterfactuals essentially assist stakeholders in understanding how to modify inputs in a meaningful way, increasing the transparency of model outputs and enabling more informed choices in practical situations.

Project ARI3205 Interpretable AI for Deep Learning Models *(Part 3.2)*

Name: Sean David Muscat

ID No: 0172004L

Importing Necessary Libraries

In [7]:
```
# Check and install required libraries from the libraries.json file
import json

# Read the libraries from the text file
with open('../Libraries/Part3.2_Lib.json', 'r') as file:
    libraries = json.load(file)

# ANSI escape codes for colored output
GREEN = "\033[92m"  # Green text
RED = "\033[91m"    # Red text
RESET = "\033[0m"   # Reset to default color

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        try:
            # Attempt to import the library
            __import__(import_name)
            print(f"[{GREEN}✓{RESET}] Library '{lib}' is already installed.")
        except ImportError:
            # If import fails, try to install the library
            print(f"[{RED}✗{RESET}] Library '{lib}' is not installed. Installin
            %pip install {lib}

# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Import necessary libraries for data analysis and modeling
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler,  OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.neural_network import MLPClassifier
# For MMD-Critic
from mmd_critic import MMDCritic
from mmd_critic.kernels import RBFKernel
```

```
from sklearn.decomposition import PCA


# Suppress specific warnings
warnings.filterwarnings("ignore", message="X does not have valid feature names")
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

```
[✓] Library 'tensorflow' is already installed.
[✓] Library 'scikit-learn' is already installed.
[✓] Library 'matplotlib' is already installed.
[✓] Library 'seaborn' is already installed.
[✓] Library 'pandas' is already installed.
[✓] Library 'numpy' is already installed.
[✓] Library 'alibi' is already installed.
[✓] Library 'statsmodels' is already installed.
[✓] Library 'mmd-critic' is already installed.
```

In [8]:

```
# Define the filenames
train_filename = '../Datasets/Titanic/train.csv'
test_filename = '../Datasets/Titanic/test.csv'
gender_submission_filename = '../Datasets/Titanic/gender_submission.csv'

# Load the datasets
try:
    train_data = pd.read_csv(train_filename)
    test_data = pd.read_csv(test_filename)
    gender_submission_data = pd.read_csv(gender_submission_filename)
    print(f"'{train_filename}' dataset loaded successfully.")
    print(f"'{test_filename}' dataset loaded successfully.")
    print(f"'{gender_submission_filename}' dataset loaded successfully.")
except FileNotFoundError as e:
    print(f"Error: {e.filename} was not found. Please ensure it is in the correc
    exit()
except pd.errors.EmptyDataError as e:
    print(f"Error: {e.filename} is empty.")
    exit()
except pd.errors.ParserError as e:
    print(f"Error: There was a problem parsing {e.filename}. Please check the fi
    exit()

# Dataset insights
print("\nTrain Dataset Overview:")
print(train_data.info())
print("\nTrain Dataset Statistical Summary:")
print(train_data.describe())

print("\nTest Dataset Overview:")
print(test_data.info())
print("\nTest Dataset Statistical Summary:")
print(test_data.describe())

print("\nGender Submission Dataset Overview:")
print(gender_submission_data.info())
```

```
'../Datasets/Titanic/train.csv' dataset loaded successfully.
'../Datasets/Titanic/test.csv' dataset loaded successfully.
'../Datasets/Titanic/gender_submission.csv' dataset loaded successfully.

Train Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

Train Dataset Statistical Summary:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000   
mean    446.000000    0.383838    2.308642   29.699118    0.523008   
std     257.353842    0.486592    0.836071   14.526497    1.102743   
min       1.000000    0.000000    1.000000    0.420000    0.000000   
25%     223.500000    0.000000    2.000000   20.125000    0.000000   
50%     446.000000    0.000000    3.000000   28.000000    0.000000   
75%     668.500000    1.000000    3.000000   38.000000    1.000000   
max     891.000000    1.000000    3.000000   80.000000    8.000000   

            Parch        Fare  
count  891.000000  891.000000  
mean     0.381594   32.204208  
std      0.806057   49.693429  
min      0.000000    0.000000  
25%      0.000000    7.910400  
50%      0.000000   14.454200  
75%      0.000000   31.000000  
max      6.000000  512.329200  

Test Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
```

```
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
None

Test Dataset Statistical Summary:
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344   35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429   55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000    0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000    7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000   14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000   31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000  512.329200

Gender Submission Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Survived     418 non-null    int64
dtypes: int64(2)
memory usage: 6.7 KB
None
```

Feed-Forward Neural Network

In [9]:

```
# Load the Titanic dataset
train_data = pd.read_csv('../Datasets/Titanic/train.csv')

# Preprocessing
# Separate features and target
y = train_data['Survived']  # Target
X = train_data.drop(columns=['Survived', 'PassengerId', 'Name', 'Ticket', 'Cabin

# Handle categorical variables with one-hot encoding
categorical_features = ['Sex', 'Embarked']
one_hot_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
categorical_encoded = one_hot_encoder.fit_transform(X[categorical_features])
categorical_encoded_df = pd.DataFrame(categorical_encoded, columns=one_hot_encod

# Drop original categorical columns and append the encoded columns
X = X.drop(columns=categorical_features)
X = pd.concat([X.reset_index(drop=True), categorical_encoded_df.reset_index(drop

# Handle missing values with mean imputation
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Standardize the features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_imputed), columns=X.columns)
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Training data shape: (712, 11)
Test data shape: (179, 11)
```

In [10]:
```
# Build the feed-forward neural network
model = Sequential([
    Input(shape=(X_train.shape[1],)),  # Define input shape explicitly
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', m

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_siz

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
```

```
Epoch 1/50
18/18 [==============================] - 1s 10ms/step - loss: 0.6789 - accuracy: 0.5923 - val_loss: 0.5875 - val_accuracy: 0.7552
Epoch 2/50
18/18 [==============================] - 0s 3ms/step - loss: 0.5487 - accuracy: 0.7750 - val_loss: 0.4977 - val_accuracy: 0.7832
Epoch 3/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4948 - accuracy: 0.7926 - val_loss: 0.4517 - val_accuracy: 0.7832
Epoch 4/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4670 - accuracy: 0.8014 - val_loss: 0.4337 - val_accuracy: 0.8182
Epoch 5/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4519 - accuracy: 0.8102 - val_loss: 0.4241 - val_accuracy: 0.8252
Epoch 6/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4409 - accuracy: 0.8155 - val_loss: 0.4153 - val_accuracy: 0.8112
Epoch 7/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4289 - accuracy: 0.8278 - val_loss: 0.4136 - val_accuracy: 0.8252
Epoch 8/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4233 - accuracy: 0.8278 - val_loss: 0.4124 - val_accuracy: 0.8182
Epoch 9/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4178 - accuracy: 0.8295 - val_loss: 0.4046 - val_accuracy: 0.8252
Epoch 10/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4112 - accuracy: 0.8330 - val_loss: 0.4047 - val_accuracy: 0.8252
Epoch 11/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4061 - accuracy: 0.8348 - val_loss: 0.4043 - val_accuracy: 0.8252
Epoch 12/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4037 - accuracy: 0.8366 - val_loss: 0.4034 - val_accuracy: 0.8252
Epoch 13/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3981 - accuracy: 0.8366 - val_loss: 0.3985 - val_accuracy: 0.8182
Epoch 14/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3970 - accuracy: 0.8401 - val_loss: 0.3997 - val_accuracy: 0.8322
Epoch 15/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3938 - accuracy: 0.8453 - val_loss: 0.4018 - val_accuracy: 0.8252
Epoch 16/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3903 - accuracy: 0.8436 - val_loss: 0.3958 - val_accuracy: 0.8322
Epoch 17/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3875 - accuracy: 0.8436 - val_loss: 0.3982 - val_accuracy: 0.8252
Epoch 18/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3849 - accuracy: 0.8436 - val_loss: 0.3984 - val_accuracy: 0.8252
Epoch 19/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3822 - accuracy: 0.8436 - val_loss: 0.4014 - val_accuracy: 0.8252
Epoch 20/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3821 - accuracy: 0.8401 - val_loss: 0.3984 - val_accuracy: 0.8182
```

```
Epoch 21/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3781 - accuracy:
0.8471 - val_loss: 0.3986 - val_accuracy: 0.8182
Epoch 22/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3767 - accuracy:
0.8383 - val_loss: 0.3983 - val_accuracy: 0.8112
Epoch 23/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3753 - accuracy:
0.8471 - val_loss: 0.4048 - val_accuracy: 0.8182
Epoch 24/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3736 - accuracy:
0.8453 - val_loss: 0.3941 - val_accuracy: 0.8252
Epoch 25/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3710 - accuracy:
0.8489 - val_loss: 0.3997 - val_accuracy: 0.8182
Epoch 26/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3708 - accuracy:
0.8418 - val_loss: 0.3985 - val_accuracy: 0.8182
Epoch 27/50
18/18 [==============================] - 0s 5ms/step - loss: 0.3693 - accuracy:
0.8401 - val_loss: 0.3957 - val_accuracy: 0.8252
Epoch 28/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3688 - accuracy:
0.8453 - val_loss: 0.4022 - val_accuracy: 0.8252
Epoch 29/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3667 - accuracy:
0.8471 - val_loss: 0.3971 - val_accuracy: 0.8252
Epoch 30/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3645 - accuracy:
0.8436 - val_loss: 0.4006 - val_accuracy: 0.8182
Epoch 31/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3631 - accuracy:
0.8471 - val_loss: 0.3967 - val_accuracy: 0.8322
Epoch 32/50
18/18 [==============================] - 0s 4ms/step - loss: 0.3623 - accuracy:
0.8489 - val_loss: 0.3934 - val_accuracy: 0.8322
Epoch 33/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3627 - accuracy:
0.8383 - val_loss: 0.4075 - val_accuracy: 0.8322
Epoch 34/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3604 - accuracy:
0.8436 - val_loss: 0.3973 - val_accuracy: 0.8252
Epoch 35/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3586 - accuracy:
0.8471 - val_loss: 0.3953 - val_accuracy: 0.8322
Epoch 36/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3567 - accuracy:
0.8489 - val_loss: 0.4053 - val_accuracy: 0.8322
Epoch 37/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3539 - accuracy:
0.8489 - val_loss: 0.4014 - val_accuracy: 0.8322
Epoch 38/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3540 - accuracy:
0.8506 - val_loss: 0.3966 - val_accuracy: 0.8392
Epoch 39/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3519 - accuracy:
0.8489 - val_loss: 0.4024 - val_accuracy: 0.8252
Epoch 40/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3517 - accuracy:
0.8436 - val_loss: 0.4021 - val_accuracy: 0.8252
```

```
Epoch 41/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3506 - accuracy:
0.8453 - val_loss: 0.3942 - val_accuracy: 0.8392
Epoch 42/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3503 - accuracy:
0.8506 - val_loss: 0.3976 - val_accuracy: 0.8392
Epoch 43/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3480 - accuracy:
0.8541 - val_loss: 0.4034 - val_accuracy: 0.8252
Epoch 44/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3492 - accuracy:
0.8489 - val_loss: 0.4037 - val_accuracy: 0.8392
Epoch 45/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3511 - accuracy:
0.8418 - val_loss: 0.4072 - val_accuracy: 0.8252
Epoch 46/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3467 - accuracy:
0.8471 - val_loss: 0.4000 - val_accuracy: 0.8252
Epoch 47/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3447 - accuracy:
0.8436 - val_loss: 0.4007 - val_accuracy: 0.8392
Epoch 48/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3444 - accuracy:
0.8453 - val_loss: 0.4037 - val_accuracy: 0.8392
Epoch 49/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3453 - accuracy:
0.8524 - val_loss: 0.3952 - val_accuracy: 0.8392
Epoch 50/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3450 - accuracy:
0.8541 - val_loss: 0.4129 - val_accuracy: 0.8112
6/6 [==============================] - 0s 2ms/step - loss: 0.4448 - accuracy: 0.8
212
Test Loss: 0.4448, Test Accuracy: 0.8212
```

Surrogate Model - MLPClassifier

In [11]:

```
# Train a surrogate model (MLPClassifier)
surrogate_model = MLPClassifier(hidden_layer_sizes=(32,), activation='logistic',
print('Accuracy (MLPClassifier): ' + str(surrogate_model.score(X_train, y_train)
```

```
Accuracy (MLPClassifier): 0.800561797752809
```

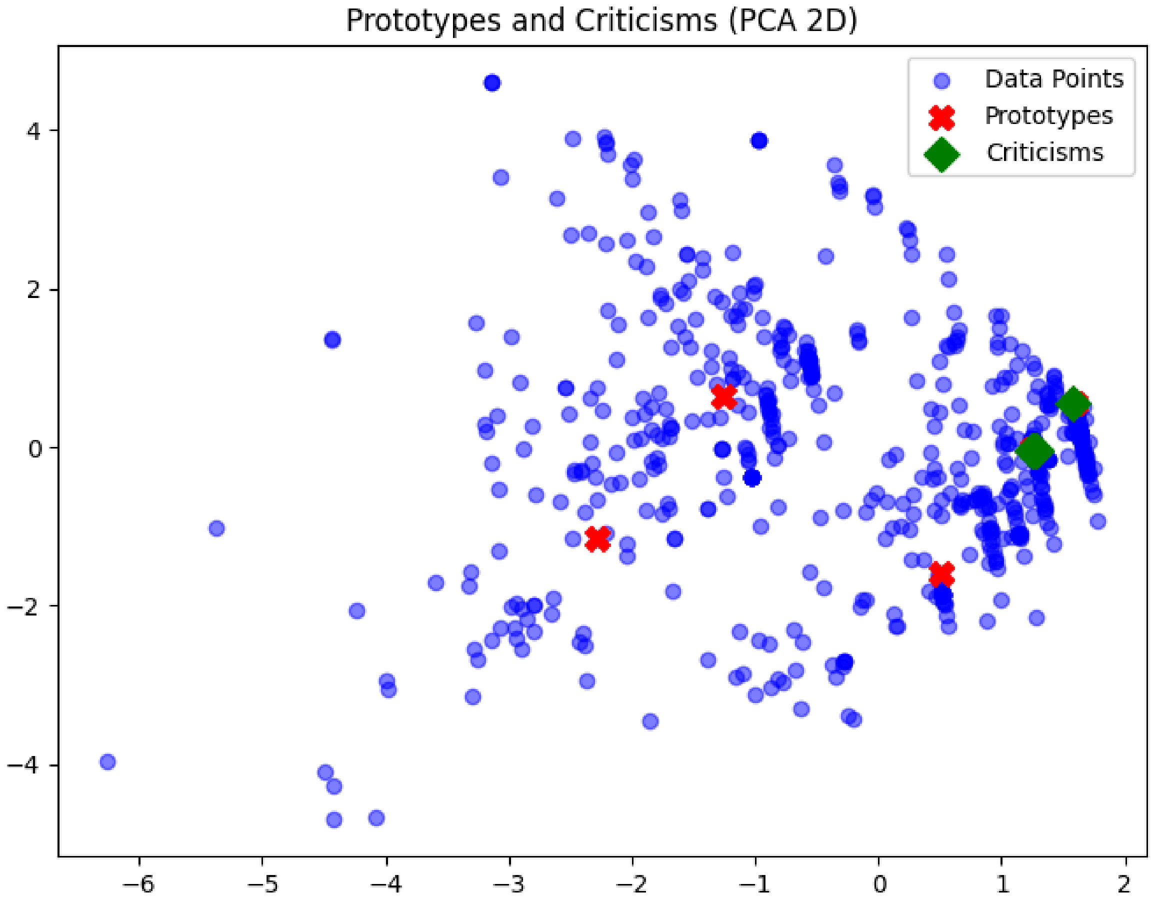
Part 3.2

Set up Prototypes and Criticisms

In [12]:

```
# Cell 5: Integrate MMD-Critic to obtain prototypes and criticisms
# We will use PCA to reduce the dimensionality of X_train to 2D for visualisatio

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_list = X_train_pca.tolist()

# Set the number of prototypes and criticisms you want to extract
n_prototypes = 5
n_criticisms = 5
```

```python
# Initialise MMD-Critic with two RBF kernels using different bandwidths
critic = MMDCritic(X_list, RBFKernel(1), RBFKernel(0.025))

# Select prototypes
prototypes, _ = critic.select_prototypes(n_prototypes)

# Select criticisms
criticisms, _ = critic.select_criticisms(n_criticisms, prototypes)

# Convert everything back to NumPy arrays for plotting
prototypes = np.array(prototypes)
criticisms = np.array(criticisms)
X_train_pca = np.array(X_list)

# Plot the data points, prototypes, and criticisms
plt.figure(figsize=(8, 6))
plt.scatter(
    X_train_pca[:, 0],
    X_train_pca[:, 1],
    c='blue',
    alpha=0.5,
    label='Data Points'
)
plt.scatter(
    prototypes[:, 0],
    prototypes[:, 1],
    c='red',
    label='Prototypes',
    marker='X',
    s=100
)
plt.scatter(
    criticisms[:, 0],
    criticisms[:, 1],
    c='green',
    label='Criticisms',
    marker='D',
    s=100
)
plt.title("Prototypes and Criticisms (PCA 2D)")
plt.legend()
plt.show()
```



This graph illustrates the result of applying MMD-Critic to identify prototypes and criticisms within the dataset. The blue circles represent the data points once they have been projected into two dimensions using PCA. The points marked with red crosses are the selected prototypes, which serve as representative samples of the dataset's main patterns or clusters. In contrast, the green diamonds denote the identified criticisms, which highlight data points that are not well explained or captured by the prototypes.

By examining the locations of these prototypes, one can observe the typical examples of the dataset that best characterise the underlying distribution. Conversely, the criticisms provide insight into observations that may be outliers or less typical, suggesting areas where the model's performance or the dataset's coverage might warrant further investigation.

3.2 b

Prototypes and criticisms are important tools in interpretable AI because they offer a way to understand a dataset and model performance beyond standard metrics. Prototypes highlight examples that capture the most prominent patterns in the dataset, effectively showing what "typical" instances look like. This helps one see how the model generalises by referencing samples deemed highly representative of the underlying data distribution.

Criticisms, on the other hand, draw attention to points that originate from these representative samples, potentially indicating outliers or subsets of data that do not conform to main trends. Identifying such points can prompt further inspection of whether the model handles these "unusual" cases effectively or whether the dataset requires augmentation or refinement. Together, prototypes and criticisms facilitate a more nuanced understanding of model decisions and data coverage, supporting informed decisions about model reliability and fairness.

Project ARI3205 Interpretable AI for Deep Learning Models *(Part 4.0)*

Name: Andrea Filiberto Lucas
ID No: 0279704L

Importing Necessary Libraries

```
import json
import os
import subprocess
import warnings
import logging
import absl.logging
#type: ignore

# Constants for colored output
COLORS = {
    "green": "\033[92m",  # Green text
    "red": "\033[91m",    # Red text
    "reset": "\033[0m"    # Reset to default color
}

# Path to the JSON file
lib_file_path = os.path.join("..", "Libraries", "Part4_Lib.json")

# Read the libraries from the JSON file
try:
    with open(lib_file_path, 'r') as file:
        libraries = json.load(file)
except FileNotFoundError:
    print(f"{COLORS['red']}Error: Library file not found at
{lib_file_path}{COLORS['reset']}")
    exit(1)
except json.JSONDecodeError:
    print(f"{COLORS['red']}Error: Failed to decode JSON from the
library file.{COLORS['reset']}")
    exit(1)

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        try:
            # Attempt to import the library
            __import__(import_name)
            print(f"[{COLORS['green']}✔{COLORS['reset']}] Library
```

```
'{lib}' is already installed.")
        except ImportError:
            # If import fails, try to install the library
            print(f"[{COLORS['red']}✖{COLORS['reset']}] Library
'{lib}' is not installed. Installing...")
            try:
                subprocess.check_call(["pip", "install", lib])
                print(f"[{COLORS['green']}✔{COLORS['reset']}]
Successfully installed '{lib}'.")
            except subprocess.CalledProcessError:
                print(f"[{COLORS['red']}✖{COLORS['reset']}] Failed to
install '{lib}'. Please install it manually.")

# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Suppress specific warnings
warnings.filterwarnings("ignore")

# Import necessary libraries for data analysis and modeling
import tensorflow as tf
#type: ignore
import numpy as np
#type: ignore
import random
#type: ignore
import matplotlib.pyplot as plt
#type: ignore
from tensorflow.keras.layers import (
#type: ignore
    Input, Conv2D, Dense, Flatten, Dropout, GlobalMaxPooling2D,
MaxPooling2D, BatchNormalization
)
from tensorflow.keras.preprocessing.image import ImageDataGenerator
#type: ignore
from tensorflow.keras.models import Model,  load_model
#type: ignore
from tf_explain.core.grad_cam import GradCAM  # Grad-CAM explainer
#type: ignore
from alibi.explainers import IntegratedGradients  # Integrated
Gradients explainer
#type: ignore
from alibi.utils.visualization import visualize_image_attr  #
Visualization function
#type: ignore

# Display TensorFlow version
print(f"TensorFlow Version: {tf.__version__}")
```

```
# Suppress specific warnings
warnings.filterwarnings("ignore")
absl.logging.set_verbosity(absl.logging.ERROR)
```

```
[✔] Library 'tensorflow' is already installed.
[✔] Library 'tensorflow_datasets' is already installed.
[✔] Library 'tf_explain' is already installed.
[✔] Library 'numpy' is already installed.
[✔] Library 'matplotlib' is already installed.
[✔] Library 'alibi' is already installed.
TensorFlow Version: 2.18.0
```

Loading and Preprocessing the CIFAR-10 Dataset

This script demonstrates how to load the CIFAR-10 dataset, split it into training and testing sets, normalize pixel values to the range [0, 1], and flatten label arrays for further use. It also prints dataset summaries at each step for clarity.

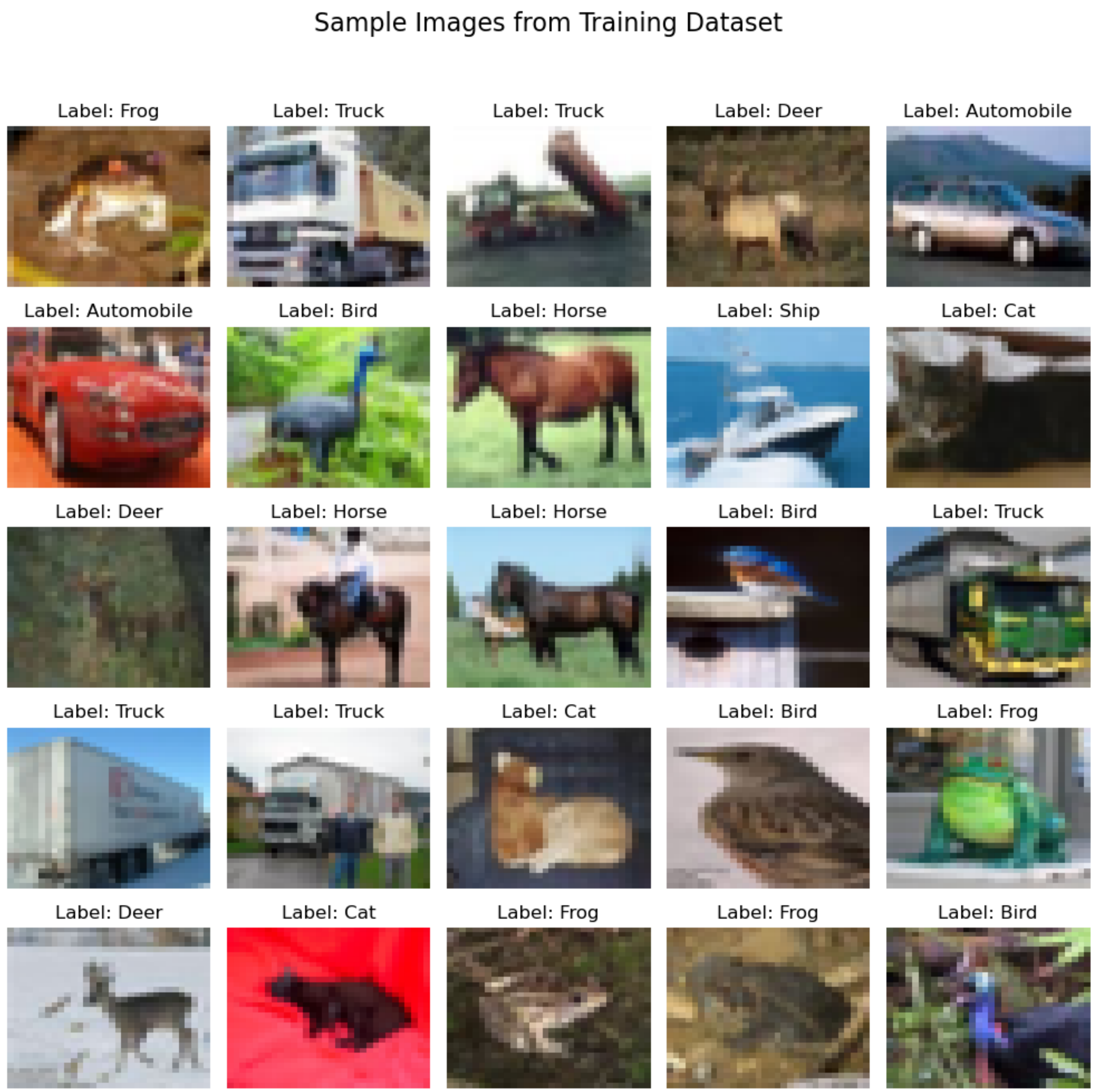
```
# Load the CIFAR-10 dataset
cifar10 = tf.keras.datasets.cifar10

# Split the dataset into training and testing sets
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print(f"Training Data Shape: {x_train.shape}, Labels Shape:
{y_train.shape}")
print(f"Testing Data Shape: {x_test.shape}, Labels Shape:
{y_test.shape}")

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# Flatten the label arrays to 1D
y_train = y_train.flatten()
y_test = y_test.flatten()

# Print dataset summary after preprocessing
print(f"Normalized Training Data: {x_train.shape}, Labels:
{y_train.shape}")
print(f"Normalized Testing Data: {x_test.shape}, Labels:
{y_test.shape}")
```

```
Training Data Shape: (50000, 32, 32, 3), Labels Shape: (50000, 1)
Testing Data Shape: (10000, 32, 32, 3), Labels Shape: (10000, 1)
Normalized Training Data: (50000, 32, 32, 3), Labels: (50000,)
Normalized Testing Data: (10000, 32, 32, 3), Labels: (10000,)
```

Visualizing CIFAR-10 Dataset with Class Names

This script defines the CIFAR-10 class names and provides a visualization function to display sample images from the training dataset along with their corresponding class labels. It uses a grid layout for better clarity.

```
# Define CIFAR-10 class names
class_names = [
    "Airplane", "Automobile", "Bird", "Cat", "Deer",
    "Dog", "Frog", "Horse", "Ship", "Truck"
]

# Visualize sample images from the training dataset
def visualize_images(images, labels, num_rows=5, num_cols=5, class_names=None):
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, 10))
    fig.suptitle("Sample Images from Training Dataset", fontsize=16)
    k = 0

    for i in range(num_rows):
        for j in range(num_cols):
            axes[i, j].imshow(images[k], aspect="auto")
            # Display the class name instead of numeric label
            label_name = class_names[labels[k]] if class_names else labels[k]
            axes[i, j].set_title(f"Label: {label_name}")
            axes[i, j].axis("off")  # Turn off axes for clarity
            k += 1

    plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout to fit the title
    plt.show()

# Call the function to visualize images with class names
visualize_images(x_train, y_train, class_names=class_names)
```



Building a CNN Model for CIFAR-10 Classification

This script constructs a Convolutional Neural Network (CNN) using TensorFlow's Functional API to classify images in the CIFAR-10 dataset. The model includes three convolutional blocks, batch normalization, max-pooling layers, dropout regularization, and a fully connected output layer for class prediction. The number of classes is dynamically determined from the dataset.

```python
# Number of classes based on the unique values in y_train
K = len(set(y_train))
print(f"Number of classes: {K}")

# Build the CNN model using the Functional API
```

```python
# Input layer
input_layer = Input(shape=x_train[0].shape, name="Input_Layer")

# First Convolutional Block
x = Conv2D(32, (3, 3), activation='relu', padding='same',
name="Conv2D_Block1_Layer1")(input_layer)
x = BatchNormalization(name="BatchNorm_Block1_Layer1")(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same',
name="Conv2D_Block1_Layer2")(x)
x = BatchNormalization(name="BatchNorm_Block1_Layer2")(x)
x = MaxPooling2D((2, 2), name="MaxPool_Block1")(x)

# Second Convolutional Block
x = Conv2D(64, (3, 3), activation='relu', padding='same',
name="Conv2D_Block2_Layer1")(x)
x = BatchNormalization(name="BatchNorm_Block2_Layer1")(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same',
name="Conv2D_Block2_Layer2")(x)
x = BatchNormalization(name="BatchNorm_Block2_Layer2")(x)
x = MaxPooling2D((2, 2), name="MaxPool_Block2")(x)

# Third Convolutional Block
x = Conv2D(128, (3, 3), activation='relu', padding='same',
name="Conv2D_Block3_Layer1")(x)
x = BatchNormalization(name="BatchNorm_Block3_Layer1")(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same',
name="Conv2D_Block3_Layer2")(x)
x = BatchNormalization(name="BatchNorm_Block3_Layer2")(x)
x = MaxPooling2D((2, 2), name="MaxPool_Block3")(x)

# Flatten the output and add Dropout
x = Flatten(name="Flatten")(x)
x = Dropout(0.2, name="Dropout_Flatten")(x)

# Fully Connected Hidden Layer
x = Dense(1024, activation='relu', name="Dense_Hidden")(x)
x = Dropout(0.2, name="Dropout_Hidden")(x)

# Output Layer
output_layer = Dense(K, activation='softmax', name="Output_Layer")(x)

# Create the model
model = Model(inputs=input_layer, outputs=output_layer,
name="CIFAR10_CNN_Model")

# Print the model summary
model.summary()
```

```
Number of classes: 10
```

```
Model: "CIFAR10_CNN_Model"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ Input_Layer (InputLayer)         │ (None, 32, 32, 3)      │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ Conv2D_Block1_Layer1 (Conv2D)    │ (None, 32, 32, 32)     │
896 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ BatchNorm_Block1_Layer1          │ (None, 32, 32, 32)     │
128 │
│ (BatchNormalization)             │                        │
│
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ Conv2D_Block1_Layer2 (Conv2D)    │ (None, 32, 32, 32)     │
9,248 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ BatchNorm_Block1_Layer2          │ (None, 32, 32, 32)     │
128 │
│ (BatchNormalization)             │                        │
│
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ MaxPool_Block1 (MaxPooling2D)    │ (None, 16, 16, 32)     │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ Conv2D_Block2_Layer1 (Conv2D)    │ (None, 16, 16, 64)     │
18,496 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ BatchNorm_Block2_Layer1          │ (None, 16, 16, 64)     │
256 │
│ (BatchNormalization)             │                        │
│
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ Conv2D_Block2_Layer2 (Conv2D)    │ (None, 16, 16, 64)     │
36,928 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
```

```
│ BatchNorm_Block2_Layer2          │ (None, 16, 16, 64)       │
256 │
│ (BatchNormalization)             │                          │
│
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ MaxPool_Block2 (MaxPooling2D)    │ (None, 8, 8, 64)         │
0 │
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ Conv2D_Block3_Layer1 (Conv2D)    │ (None, 8, 8, 128)        │
73,856 │
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ BatchNorm_Block3_Layer1          │ (None, 8, 8, 128)        │
512 │
│ (BatchNormalization)             │                          │
│
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ Conv2D_Block3_Layer2 (Conv2D)    │ (None, 8, 8, 128)        │
147,584 │
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ BatchNorm_Block3_Layer2          │ (None, 8, 8, 128)        │
512 │
│ (BatchNormalization)             │                          │
│
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ MaxPool_Block3 (MaxPooling2D)    │ (None, 4, 4, 128)        │
0 │
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ Flatten (Flatten)                │ (None, 2048)             │
0 │
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ Dropout_Flatten (Dropout)        │ (None, 2048)             │
0 │
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ Dense_Hidden (Dense)             │ (None, 1024)             │
2,098,176 │
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ Dropout_Hidden (Dropout)         │ (None, 1024)             │
0 │
├──────────────────────────────────┼──────────────────────────┼──────────
───────┤
```

```
│ Output_Layer (Dense)              │ (None, 10)                │
10,250 │
└───────────────────────────────────┴───────────────────────────┴───────────
───────┘

 Total params: 2,397,226 (9.14 MB)

 Trainable params: 2,396,330 (9.14 MB)

 Non-trainable params: 896 (3.50 KB)
```

Model Selection and Training with Data Augmentation

This subsection outlines the process for selecting and training a model:

1. **Checking for Pre-Trained Models:**
   - The script searches the specified directory (`../Models`) for available `.h5` files.
   - If models are found, they are listed, and the user can choose to load one or train a new model.
2. **Training a New Model:**
   - If no models are available or the user opts to train a new model, they can specify the number of epochs.
   - The model is compiled with the Adam optimizer and sparse categorical cross-entropy loss.
   - Data augmentation is applied to the training dataset using `ImageDataGenerator` with random shifts and flips.
3. **Visualization and Model Saving:**
   - Training and validation accuracy and loss are plotted over epochs.
   - The trained model is saved in the `../Models` directory with the number of epochs in the filename.

This approach ensures flexibility in leveraging pre-trained models while enabling efficient training of new models with augmented data.

```python
# Define the directory to check for models
model_dir = os.path.join("..", "Models")

# List available models
available_models = [f for f in os.listdir(model_dir) if
f.endswith('.h5')]

if available_models:
    print("Available models:")
    print("0: Train a new model")
    for idx, model_name in enumerate(available_models, start=1):
        print(f"{idx}: {model_name}")

    # Prompt user to select a model
    selected_idx = int(input("Enter the number of the model to load"))
```

```
    if selected_idx > 0 and selected_idx <= len(available_models):
        # Load the selected model
        selected_model_path = os.path.join(model_dir,
available_models[selected_idx - 1])
        model = load_model(selected_model_path)
        print(f"'{selected_model_path}' loaded successfully.")
    else:
        print("Training a new model...")
        # Ask user for number of epochs
        epochs = int(input("Enter the number of epochs for training:
"))

        # Compile the model
        model.compile(
            optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']
        )

        # Define batch size
        batch_size = 32

        # Data Augmentation for Training Data
        data_generator = ImageDataGenerator(
            width_shift_range=0.1,  # Random horizontal shift
            height_shift_range=0.1,  # Random vertical shift
            horizontal_flip=True     # Random horizontal flip
        )

        # Create a data generator for the training dataset
        train_generator = data_generator.flow(x_train, y_train,
batch_size=batch_size)

        # Steps per epoch
        steps_per_epoch = x_train.shape[0] // batch_size

        # Train the model using augmented data
        history = model.fit(
            train_generator,
            validation_data=(x_test, y_test),
            steps_per_epoch=steps_per_epoch,
            epochs=epochs,
            verbose=1
        )

        # Plot training and validation accuracy
        plt.figure(figsize=(10, 5))
        plt.plot(history.history['accuracy'], label='Training
Accuracy')
```

```
        plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
        plt.title('Accuracy Over Epochs')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()
        plt.show()

        # Plot training and validation loss
        plt.figure(figsize=(10, 5))
        plt.plot(history.history['loss'], label='Training Loss')
        plt.plot(history.history['val_loss'], label='Validation Loss')
        plt.title('Loss Over Epochs')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()

        # Save the model with the number of epochs in the filename
        save_path = os.path.join(model_dir, f"AFL_{epochs}.h5")
        model.save(save_path)
        print(f"Model saved successfully at: {save_path}")
else:
    print("No models found. Training a new model...")

    # Ask user for number of epochs
    epochs = int(input("Enter the number of epochs for training: "))

    # Compile the model
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    # Define batch size
    batch_size = 32

    # Data Augmentation for Training Data
    data_generator = ImageDataGenerator(
        width_shift_range=0.1,  # Random horizontal shift
        height_shift_range=0.1,  # Random vertical shift
        horizontal_flip=True     # Random horizontal flip
    )

    # Create a data generator for the training dataset
    train_generator = data_generator.flow(x_train, y_train,
batch_size=batch_size)

    # Steps per epoch
```

```
    steps_per_epoch = x_train.shape[0] // batch_size

    # Train the model using augmented data
    history = model.fit(
        train_generator,
        validation_data=(x_test, y_test),
        steps_per_epoch=steps_per_epoch,
        epochs=epochs,
        verbose=1
    )

    # Save the model with the number of epochs in the filename
    save_path = os.path.join(model_dir, f"AFL_{epochs}.h5")
    model.save(save_path)
    print(f"Model saved successfully at: {save_path}")
```

```
Available models:
0: Train a new model
1: AFL_15T.h5
2: AFL_1T.h5
'../Models/AFL_15T.h5' loaded successfully.
```

Predicting a Random Test Image

A random image from the test dataset is displayed with its original label. The image is reshaped, and the model predicts its label, which is then compared to the original label.

```
# Select a random image from the test dataset
image_number = random.randint(0, x_test.shape[0] - 1)

# Display the selected image
plt.imshow(x_test[image_number])
plt.title(f"Original Label: {class_names[y_test[image_number]]}")
plt.axis("off")
plt.show()

# Load the selected image into an array
image_array = np.array(x_test[image_number])

# Reshape the image to match the input shape expected by the model
reshaped_image = image_array.reshape(1, 32, 32, 3)

# Predict the label using the model
predicted_label = class_names[model.predict(reshaped_image).argmax()]

# Load the original label
original_label = class_names[y_test[image_number]]

# Display the result
```

```
print(f"Original label: {original_label}")
print(f"Predicted label: {predicted_label}")
```


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
Original label: Bird
Predicted label: Dog
```

Visualizing Model Explanations with Integrated Gradients

This section demonstrates how to use Integrated Gradients (IG) to explain model predictions:

1. **Integrated Gradients Setup:**
   - IG is initialized with parameters such as the number of steps, method, and internal batch size.
2. **Select a Test Instance:**
   - A random test image is selected from the dataset. Specific indices can be preferred for reproducibility.
3. **Generate Attributions:**
   - A baseline image (black) is used to calculate attributions for the selected instance, focusing on the true class label.

4. **Visualization:**
    - The original image and the attribution heatmap are visualized side by side. The heatmap highlights the regions of the image that contributed to the model's prediction.

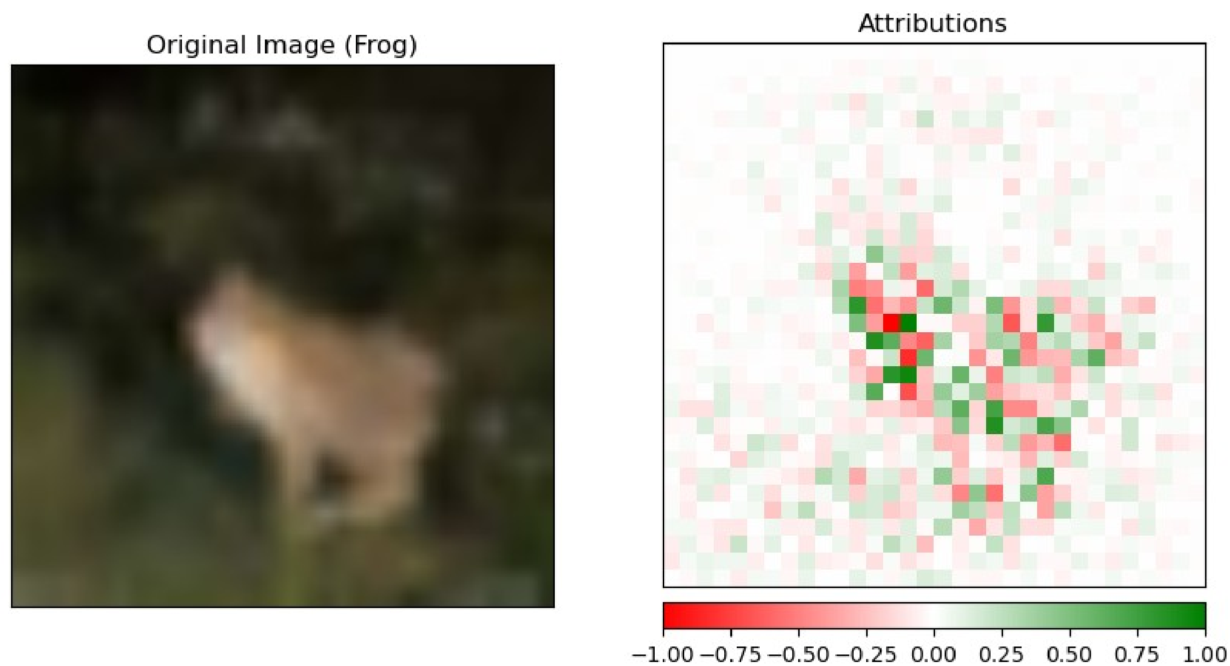
This approach helps to understand which parts of the image the model relies on for its decisions.

```
# Set up Integrated Gradients
n_steps = 50
method = "gausslegendre"
internal_batch_size = 50
ig = IntegratedGradients(
    model,
    n_steps=n_steps,
    method=method,
    internal_batch_size=internal_batch_size
)

# Select a random instance to explain
i = random.randint(0, len(x_test) - 1)  # Random index from the test
dataset
print(f"Selected instance index: {i}") # Preferred Indexes: 2864,
2003, 4028, 86644, 91, 2741, 5238, 5441, 98, 5113
instance = np.expand_dims(x_test[i], axis=0)

# Use the true class label as the target
true_label = int(y_test[i])  # Ensure the target is an integer

# Generate attributions using Integrated Gradients
baseline = np.zeros(instance.shape)  # Black image as baseline
explanation = ig.explain(instance, baselines=baseline,
target=true_label)
attrs = explanation.attributions[0]  # Get the attributions for the
selected instance

# Upscale the original image for better visualization (optional)
original_image = tf.image.resize(x_test[i], size=(128, 128)).numpy()

# Visualize original image and attributions
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))

# Visualize the original image
visualize_image_attr(
    attr=None,
    original_image=original_image,
    method='original_image',
    title=f'Original Image ({class_names[true_label]})',
    plt_fig_axis=(fig, ax[0]),
    use_pyplot=False
)
```

```python
# Visualize the attributions
visualize_image_attr(
    attr=attrs.squeeze(),
    original_image=original_image,
    method='heat_map',
    sign='all',  # Show both positive and negative contributions
    show_colorbar=True,
    title='Attributions',
    plt_fig_axis=(fig, ax[1]),
    use_pyplot=True
)

plt.tight_layout()
plt.show()
```

```
Selected instance index: 5113
```

```
2025-01-07 17:03:15.919054: I
tensorflow/core/framework/local_rendezvous.cc:405] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence
```




```
<Figure size 640x480 with 0 Axes>
```

Discussion of the results obtained

The **Integrated Gradients (IG)** visualization for the selected instance (index 5113) provides a detailed explanation of the model's decision-making process. **IG** is a technique used to interpret neural network predictions by attributing importance to input features (e.g., pixels in an image). It computes these attributions by accumulating gradients along a path from a baseline input, such as a black image, to the actual input. This method adheres to mathematical principles like **sensitivity** and **implementation invariance**, ensuring that the attributions are reliable and consistent. **IG** is particularly valuable for enhancing model interpretability by highlighting the features most influential to its output.

In the visualization, **green** and **red regions** signify the importance of specific pixels to the model's prediction. **Green areas** represent *positive contributions*, indicating that these regions increase the model's confidence in its classification. Conversely, **red areas** represent *negative contributions*, meaning they detract from the model's confidence. The attribution map's boxy nature arises from the pixelated structure of **CIFAR-10 images**, which are low resolution (32x32 pixels). Each pixel represents a significant portion of the image, and the blocky attributions align with this granularity, providing a meaningful coarse-grained analysis.

The results show that the model likely focused on specific **green-highlighted regions** that correspond to distinctive features of the frog, such as its body outline or texture. **Red regions**, often sparse and located near the edges or background, signify areas that are less relevant or even distracting to the model's prediction. The **high concentration of green pixels in the central area** of the image demonstrates that the model effectively prioritizes the object's most relevant features, aligning with human intuition.

This visualization exemplifies how **Integrated Gradients** enhances model transparency by offering a clear explanation of its decision-making process. By distinguishing between foreground and background elements and **de-emphasizing irrelevant features**, **IG** enables debugging and builds trust in deep learning models. Such insights are invaluable for improving model reliability, particularly in applications requiring high levels of interpretability.

Visualizing Model Explanations with Grad-CAM

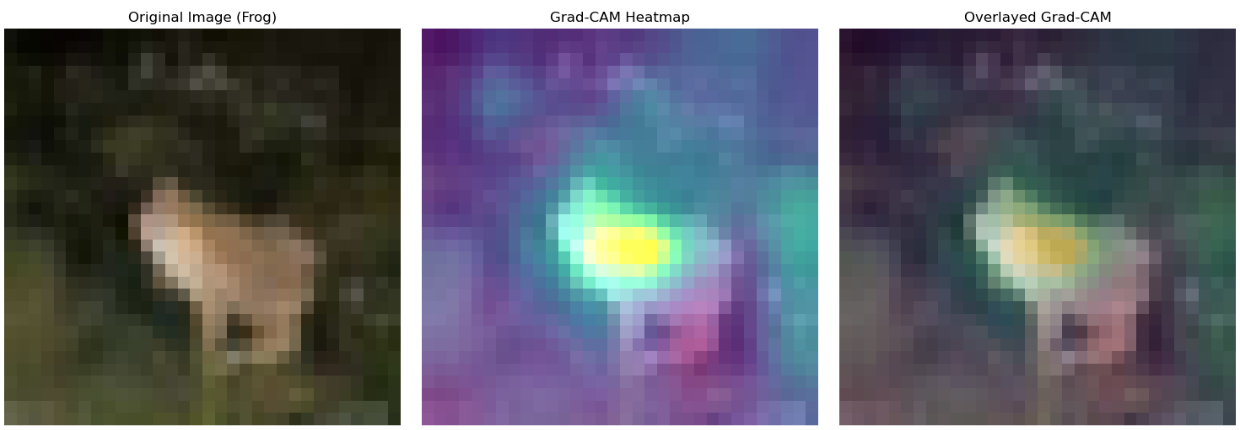
This section illustrates how to use Grad-CAM to generate visual explanations for model predictions:

1. **Select a Test Instance:**
   - A random image from the test dataset is chosen, with the true class label identified for generating explanations.
2. **Target Layer Specification:**
   - The target convolutional layer (`Conv2D_Block3_Layer1`) is specified to focus on feature maps relevant to the prediction.
3. **Grad-CAM Initialization and Explanation:**
   - Grad-CAM is used to compute a heatmap that highlights areas of the image contributing to the prediction for the true class.
4. **Visualization:**
   - Three visualizations are presented:
     - The original image.
     - The Grad-CAM heatmap, which indicates important regions.
     - An overlay of the heatmap on the original image for better interpretability.

This technique helps to localize regions in the image that influence the model's decision-making process.

```
print(f"Selected instance index for Grad-CAM: {i}")

# Prepare the input instance and metadata
instance = np.expand_dims(x_test[i], axis=0)  # Expand dimensions for
model input
true_label = int(y_test[i])  # Convert true class to integer
original_image = x_test[i]  # Keep the original image for
visualization

# Specify the target convolutional layer
target_layer_name = "Conv2D_Block3_Layer1"  # Update to match your
model's layer name
target_layer = model.get_layer(target_layer_name)

# Initialize Grad-CAM
grad_cam = GradCAM()

# Generate the Grad-CAM explanation
explanation = grad_cam.explain(
    validation_data=(instance, None),  # Model input, no labels
required
    model=model,
    class_index=true_label,  # Target class for explanation
    layer_name=target_layer.name  # Specify target layer
)

# Process the heatmap
heatmap = np.maximum(explanation, 0)  # Clip negative values
heatmap = heatmap / np.max(heatmap)  # Normalize to [0, 1]

# Resize the heatmap to match the original image size
heatmap_resized = tf.image.resize(heatmap, (original_image.shape[0],
original_image.shape[1])).numpy()

# Create the overlayed image
overlayed_image = 0.6 * original_image + 0.4 * heatmap_resized  #
Blend original and heatmap

# Visualize the results
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

# Display the original image
ax[0].imshow(original_image)
ax[0].set_title(f"Original Image ({class_names[true_label]})")
ax[0].axis("off")

# Display the Grad-CAM heatmap
```

```
ax[1].imshow(heatmap_resized, cmap="jet")
ax[1].set_title("Grad-CAM Heatmap")
ax[1].axis("off")

# Display the overlayed image
ax[2].imshow(overlayed_image)
ax[2].set_title("Overlayed Grad-CAM")
ax[2].axis("off")

# Adjust layout and display
plt.tight_layout()
plt.show()

Selected instance index for Grad-CAM: 5113
```



Discussion of the results obtained

The **Grad-CAM** visualization for the selected instance (index 5113) provides a clear understanding of the model's decision-making process. **Grad-CAM (Gradient-weighted Class Activation Mapping)** generates a heatmap highlighting the spatial regions of the input image that are most relevant to the model's prediction. This method leverages gradients flowing into the final convolutional layers to identify and visualize the areas of the image contributing the most to the predicted class. Unlike pixel-level attribution techniques like Integrated Gradients, Grad-CAM provides a broader, spatially coherent explanation by focusing on feature maps.

In the results, the **original image** on the left shows the input labeled as "Frog," while the **Grad-CAM heatmap** in the center highlights the areas of highest importance in yellow and green. The yellow regions represent the strongest positive contributions to the "Frog" prediction, primarily focusing on the central part of the image, aligning with the frog's body or distinctive features. The **overlayed Grad-CAM** image on the right combines the heatmap with the original image, offering a seamless representation of the model's focus areas. Background regions, such as those in the periphery, are de-emphasized (appearing darker or in blue), demonstrating that the model correctly disregards irrelevant parts of the image.

The Grad-CAM heatmap provides a coarse but spatially accurate attribution, with smoother transitions due to its operation on convolutional feature maps. This makes it less granular than methods like Integrated Gradients but sufficient to highlight the critical regions influencing the model's decision. By focusing on the central areas of the image, the visualization confirms that the model is leveraging the frog's features, such as shape and texture, to make an informed classification.

The overlayed visualization enhances interpretability by merging the heatmap with the input image, making it evident how the model aligns its prediction with the visual features of the frog. This demonstration of Grad-CAM highlights its utility in debugging and validating convolutional models, as it provides valuable insights into the spatial reasoning behind the model's predictions.