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
import seaborn as sns
import statsmodels.formula.api as smf
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.inspection import PartialDependenceDisplay, permutation_importance
from alibi.explainers import ALE, plot_ale
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.metrics import accuracy_score
import statsmodels.api as sm
import lime
from lime.lime_tabular import LimeTabularExplainer
import shap
from anchor import utils
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 '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.
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 'alibi' is already installed.
[✓] Library 'lime' is already installed.
[✓] 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: 0.6621 - accuracy: 0.
5993 - val_loss: 0.5743 - val_accuracy: 0.8112
Epoch 2/50
18/18 [==============================] - 0s 3ms/step - loss: 0.5398 - accuracy: 0.7
838 - val_loss: 0.4768 - val_accuracy: 0.8182
Epoch 3/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4773 - accuracy: 0.7
891 - val_loss: 0.4319 - val_accuracy: 0.8392
Epoch 4/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4507 - accuracy: 0.8
032 - val_loss: 0.4088 - val_accuracy: 0.8392
Epoch 5/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4390 - accuracy: 0.8
067 - val_loss: 0.4015 - val_accuracy: 0.8252
Epoch 6/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4317 - accuracy: 0.8
067 - val_loss: 0.3980 - val_accuracy: 0.8182
Epoch 7/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4238 - accuracy: 0.8
120 - val_loss: 0.3918 - val_accuracy: 0.8392
Epoch 8/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4179 - accuracy: 0.8
190 - val_loss: 0.3914 - val_accuracy: 0.8462
Epoch 9/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4147 - accuracy: 0.8
172 - val_loss: 0.3904 - val_accuracy: 0.8462
Epoch 10/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4109 - accuracy: 0.8
330 - val_loss: 0.3874 - val_accuracy: 0.8462
Epoch 11/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4063 - accuracy: 0.8
313 - val_loss: 0.3922 - val_accuracy: 0.8392
Epoch 12/50
18/18 [==============================] - 0s 3ms/step - loss: 0.4030 - accuracy: 0.8
383 - val_loss: 0.3927 - val_accuracy: 0.8462
Epoch 13/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3979 - accuracy: 0.8
401 - val_loss: 0.3856 - val_accuracy: 0.8462
Epoch 14/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3963 - accuracy: 0.8
401 - val_loss: 0.3879 - val_accuracy: 0.8462
Epoch 15/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3931 - accuracy: 0.8
436 - val_loss: 0.3892 - val_accuracy: 0.8462
Epoch 16/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3924 - accuracy: 0.8
348 - val_loss: 0.3884 - val_accuracy: 0.8392
Epoch 17/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3902 - accuracy: 0.8
348 - val_loss: 0.3870 - val_accuracy: 0.8531
Epoch 18/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3874 - accuracy: 0.8
401 - val_loss: 0.3868 - val_accuracy: 0.8531
Epoch 19/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3859 - accuracy: 0.8
418 - val_loss: 0.3841 - val_accuracy: 0.8462
Epoch 20/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3830 - accuracy: 0.8
401 - val_loss: 0.3903 - val_accuracy: 0.8392
```

```
Epoch 21/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3807 - accuracy: 0.8
436 - val_loss: 0.3823 - val_accuracy: 0.8531
Epoch 22/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3816 - accuracy: 0.8
453 - val_loss: 0.3934 - val_accuracy: 0.8462
Epoch 23/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3766 - accuracy: 0.8
366 - val_loss: 0.3850 - val_accuracy: 0.8462
Epoch 24/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3769 - accuracy: 0.8
453 - val_loss: 0.3903 - val_accuracy: 0.8392
Epoch 25/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3749 - accuracy: 0.8
436 - val_loss: 0.3890 - val_accuracy: 0.8462
Epoch 26/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3722 - accuracy: 0.8
418 - val_loss: 0.3877 - val_accuracy: 0.8462
Epoch 27/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3743 - accuracy: 0.8
436 - val_loss: 0.3920 - val_accuracy: 0.8462
Epoch 28/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3705 - accuracy: 0.8
471 - val_loss: 0.3868 - val_accuracy: 0.8462
Epoch 29/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3709 - accuracy: 0.8
453 - val_loss: 0.3926 - val_accuracy: 0.8462
Epoch 30/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3708 - accuracy: 0.8
471 - val_loss: 0.3887 - val_accuracy: 0.8462
Epoch 31/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3686 - accuracy: 0.8
453 - val_loss: 0.3885 - val_accuracy: 0.8462
Epoch 32/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3663 - accuracy: 0.8
471 - val_loss: 0.3862 - val_accuracy: 0.8462
Epoch 33/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3636 - accuracy: 0.8
489 - val_loss: 0.3926 - val_accuracy: 0.8462
Epoch 34/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3622 - accuracy: 0.8
489 - val_loss: 0.3918 - val_accuracy: 0.8462
Epoch 35/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3644 - accuracy: 0.8
471 - val_loss: 0.3890 - val_accuracy: 0.8531
Epoch 36/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3607 - accuracy: 0.8
489 - val_loss: 0.3926 - val_accuracy: 0.8531
Epoch 37/50
18/18 [==============================] - 0s 5ms/step - loss: 0.3607 - accuracy: 0.8
471 - val_loss: 0.3880 - val_accuracy: 0.8531
Epoch 38/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3616 - accuracy: 0.8
453 - val_loss: 0.3956 - val_accuracy: 0.8392
Epoch 39/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3608 - accuracy: 0.8
489 - val_loss: 0.3941 - val_accuracy: 0.8462
Epoch 40/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3569 - accuracy: 0.8
453 - val_loss: 0.3949 - val_accuracy: 0.8392
```

```
Epoch 41/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3555 - accuracy: 0.8
471 - val_loss: 0.3940 - val_accuracy: 0.8531
Epoch 42/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3554 - accuracy: 0.8
453 - val_loss: 0.3919 - val_accuracy: 0.8462
Epoch 43/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3528 - accuracy: 0.8
489 - val_loss: 0.3947 - val_accuracy: 0.8462
Epoch 44/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3533 - accuracy: 0.8
489 - val_loss: 0.3955 - val_accuracy: 0.8392
Epoch 45/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3513 - accuracy: 0.8
489 - val_loss: 0.3956 - val_accuracy: 0.8531
Epoch 46/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3488 - accuracy: 0.8
506 - val_loss: 0.3935 - val_accuracy: 0.8531
Epoch 47/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3507 - accuracy: 0.8
541 - val_loss: 0.3962 - val_accuracy: 0.8531
Epoch 48/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3508 - accuracy: 0.8
576 - val_loss: 0.3959 - val_accuracy: 0.8462
Epoch 49/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3478 - accuracy: 0.8
559 - val_loss: 0.3982 - val_accuracy: 0.8392
Epoch 50/50
18/18 [==============================] - 0s 3ms/step - loss: 0.3459 - accuracy: 0.8
559 - val_loss: 0.3975 - val_accuracy: 0.8462
6/6 [==============================] - 0s 1ms/step - loss: 0.4419 - accuracy: 0.810
1
Test Loss: 0.4419, Test Accuracy: 0.8101
```

Surrogate Model - MLPClassifier

In [5]:

```
# 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 a

Set up the LIME explainer

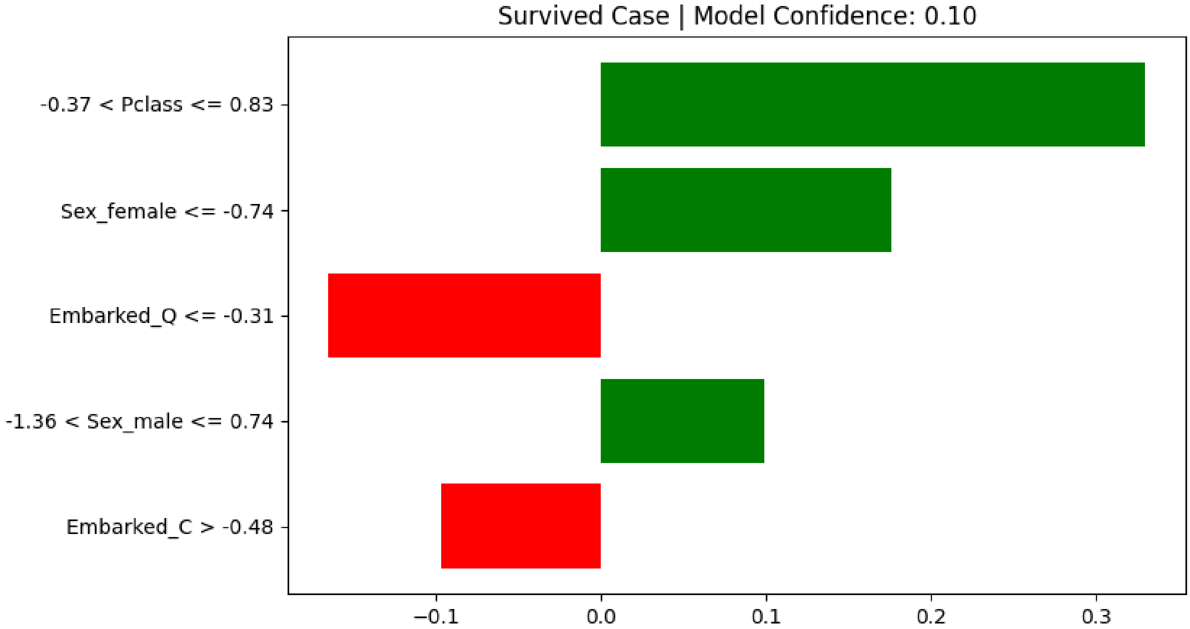
In [6]:

```
# 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()
```

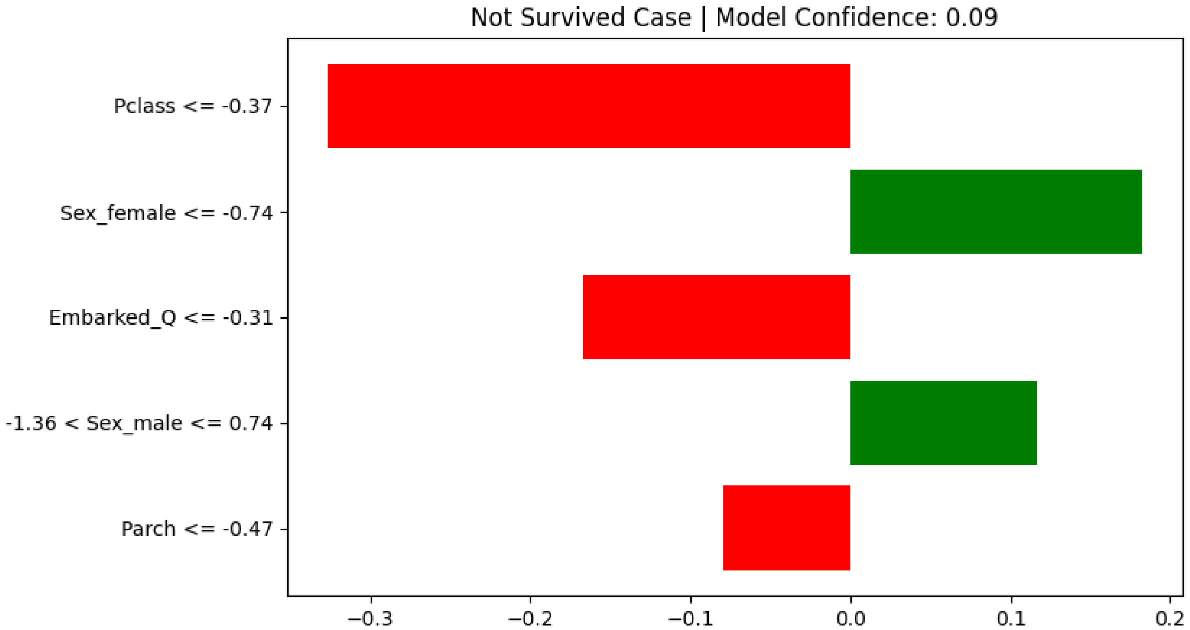
```
    names.reverse()

    # 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 794us/step
Available label for Survived instance: 0
1/1 [==============================] - 0s 19ms/step
```




```
157/157 [==============================] - 0s 987us/step
Available label for Not Survived instance: 0
1/1 [==============================] - 0s 19ms/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 a

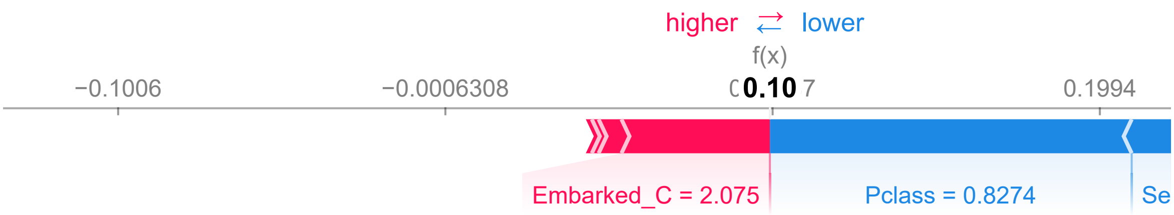
Adding SHAP to Explain Model Predictions

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 1ms/step
  0%|          | 0/10 [00:00<?, ?it/s]
1/1 [==============================] - 0s 20ms/step
6394/6394 [==============================] - 5s 759us/step
 10%|█         | 1/10 [00:08<01:16,  8.54s/it]
1/1 [==============================] - 0s 20ms/step
6394/6394 [==============================] - 5s 811us/step
 20%|██        | 2/10 [00:16<01:07,  8.44s/it]
1/1 [==============================] - 0s 20ms/step
6394/6394 [==============================] - 5s 789us/step
 30%|███       | 3/10 [00:24<00:57,  8.28s/it]
1/1 [==============================] - 0s 20ms/step
6394/6394 [==============================] - 5s 789us/step
 40%|████      | 4/10 [00:33<00:49,  8.22s/it]
1/1 [==============================] - 0s 19ms/step
6394/6394 [==============================] - 5s 772us/step
 50%|█████     | 5/10 [00:41<00:40,  8.17s/it]
1/1 [==============================] - 0s 18ms/step
6394/6394 [==============================] - 5s 808us/step
 60%|██████    | 6/10 [00:49<00:32,  8.25s/it]
1/1 [==============================] - 0s 20ms/step
6394/6394 [==============================] - 5s 806us/step
 70%|███████   | 7/10 [00:57<00:24,  8.20s/it]
1/1 [==============================] - 0s 19ms/step
6394/6394 [==============================] - 5s 749us/step
 80%|████████  | 8/10 [01:05<00:16,  8.06s/it]
1/1 [==============================] - 0s 19ms/step
6394/6394 [==============================] - 5s 763us/step
 90%|█████████ | 9/10 [01:13<00:08,  8.02s/it]
1/1 [==============================] - 0s 22ms/step
6394/6394 [==============================] - 5s 770us/step
100%|██████████| 10/10 [01:21<00:00,  8.14s/it]
```

Out[7]:



Part 2.2 b

LIME operates by approximating the decision boundary of a model around a specific instance. It perturbs the input data near the instance, evaluates the model's predictions on these samples, and fits an interpretable surrogate model, such as a linear regression, to approximate the model's local behaviour. The visualisations for LIME highlight feature contributions as positive or negative weights, showing whether each feature supports or opposes a given outcome. For example, in the Titanic dataset, "Pclass" and "Sex_female" significantly support predicting survival, represented by green bars, while "Embarked_Q" negatively impacts the same prediction. LIME's strength lies in its simplicity and its ability to provide clear, instance-specific explanations. However, its reliance on local approximations can sometimes fail to capture the full complexity of highly non-linear models, which may limit its accuracy.

SHAP, in contrast, calculates feature contributions using Shapley values from cooperative game theory. This ensures both consistency and fairness in feature attribution. Unlike LIME, SHAP considers all possible combinations of feature presence and absence when determining a feature's impact, making it computationally heavier. In the SHAP visualisation for the Titanic dataset, feature contributions are shown on a scale, with features like "Embarked_C" positively influencing survival and "Embarked_S" having a strong negative effect. SHAP's additive nature ensures that the contributions of all features sum up to the model's prediction, making it a reliable method for both local and global explanations.

The key difference lies in their focus. LIME concentrates on approximating the local decision boundary and providing quick, intuitive insights for a single instance, while SHAP ensures consistent and theoretically sound attributions for both local and global interpretations. While LIME is faster and more straightforward, SHAP provides a more comprehensive understanding of feature impacts, albeit with higher computational costs.

Part 2.3 a

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 <= 0.4
3

Parch > -0.47

-0.36 < Fare <= -0.03

Sex_female <= -0.74

-1.36 < Sex_male <=
0.74

Embarked_C > -0.48

Embarked_Q <= -0.3
1

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

Anchors, LIME, and SHAP each provide unique methods to interpret machine learning predictions. Anchors focus on rule-based explanations by identifying minimal conditions that "anchor" a prediction with high precision. For example, in the Titanic dataset, the rule Sex_female <= -0.74 AND Pclass = female predicts "Not Survived" with 97.7% accuracy. This makes Anchors highly interpretable, as they produce simple, easy-to-follow rules. However, these rules may oversimplify the decision-making process by ignoring interactions outside the specified conditions.

LIME offers an alternative by perturbing input data and fitting a linear model to approximate the local decision boundary. It highlights individual feature contributions, such as "Pclass" or "Sex_female," as either supporting or opposing a prediction. While LIME is effective for generating quick and intuitive explanations, it may struggle to fully capture non-linear interactions in the model and can introduce variability due to the randomness in its perturbations.

SHAP, meanwhile, assigns additive contributions to features using Shapley values, ensuring consistency and fairness in attribution. In the Titanic dataset, SHAP visualisations detail how features like "Embarked_C" strongly support survival predictions, while "Embarked_S" reduces the likelihood of survival. SHAP excels in providing both local and global explanations but is more computationally intensive and may overwhelm users with its level of detail.

In conclusion, Anchors are ideal for generating simple and actionable rules, LIME is useful for intuitive and fast local explanations, and SHAP is best suited for comprehensive and reliable feature attributions. The choice among these methods depends on the complexity of the model and the specific interpretability needs.