Faculty of Media Engineering and Technology
German University in Cairo
Dr. Nourhan Ehab

CSEN 903: Advanced Computer Lab, Winter 2025
Lab 4 Manual: Explainable AI

Part 1: Read and Watch Before the Lab

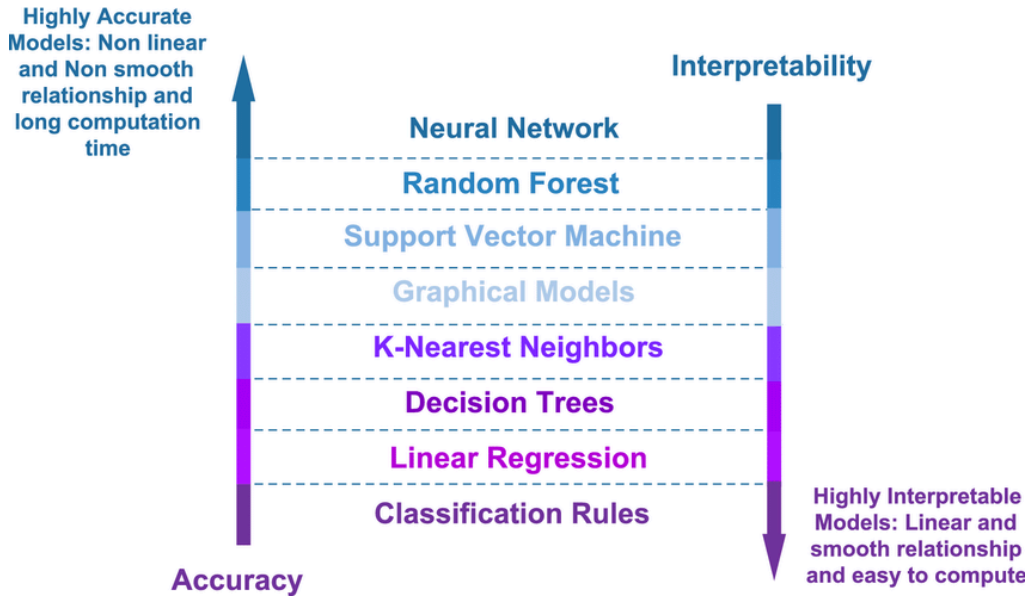
Machine learning models, especially complex ones like neural networks, can be difficult to interpret. This raises questions like:

- **Why did the model make this decision?**
- **Which features contributed most?**
- **Was the model fair or biased?**

Explainable Artificial Intelligence (XAI) refers to **techniques and methods that make the output and workings of AI systems understandable to humans**.
Its goal is to:

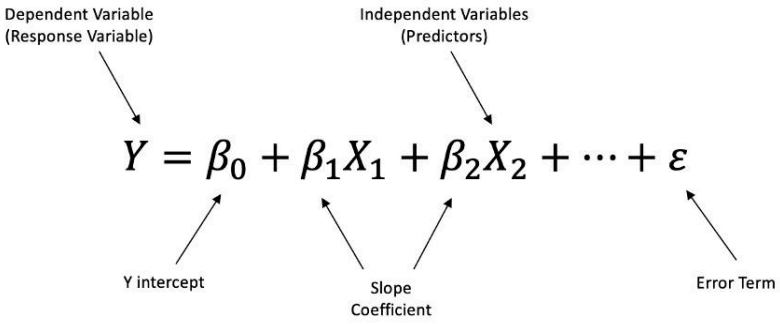
- Increase **transparency** in decision-making.
- Build **trust** between humans and AI systems.
- Allow **debugging** and **bias detection** in models.

Classes of XAI:

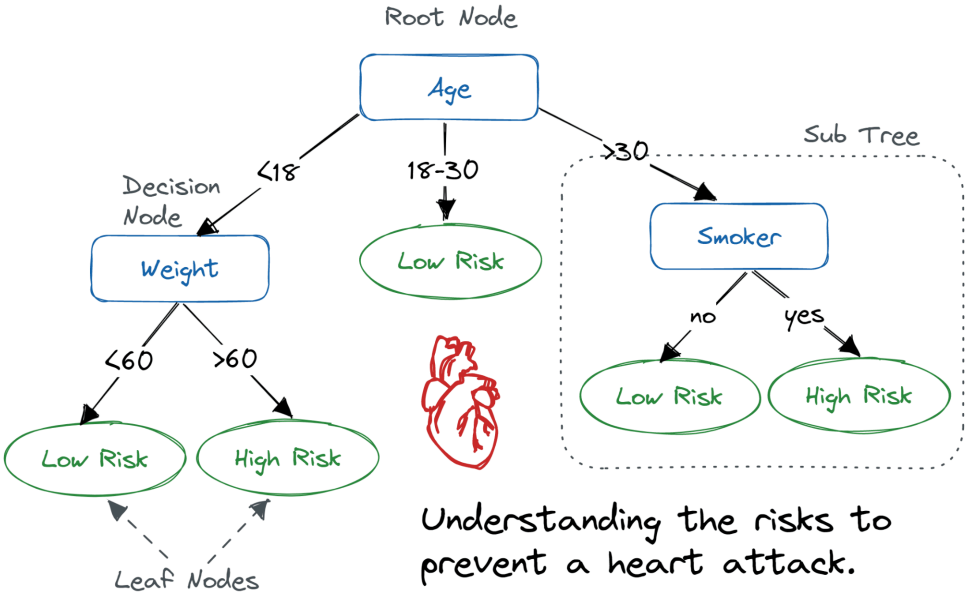


1. **Intrinsically Interpretable —** These models are **inherently transparent,** meaning their structure is understandable *without the need for additional tools*. Examples of such models are:

a. **Linear Regression —** Here, we can "explain" the model's output in terms of the coefficients of each variable. This means that we can explain which variables affect the prediction and to what extent, which is a great insight to have. Using this insight, we can analyse whether the output is valid or not, and if not, we can have an intuition where the issue is.



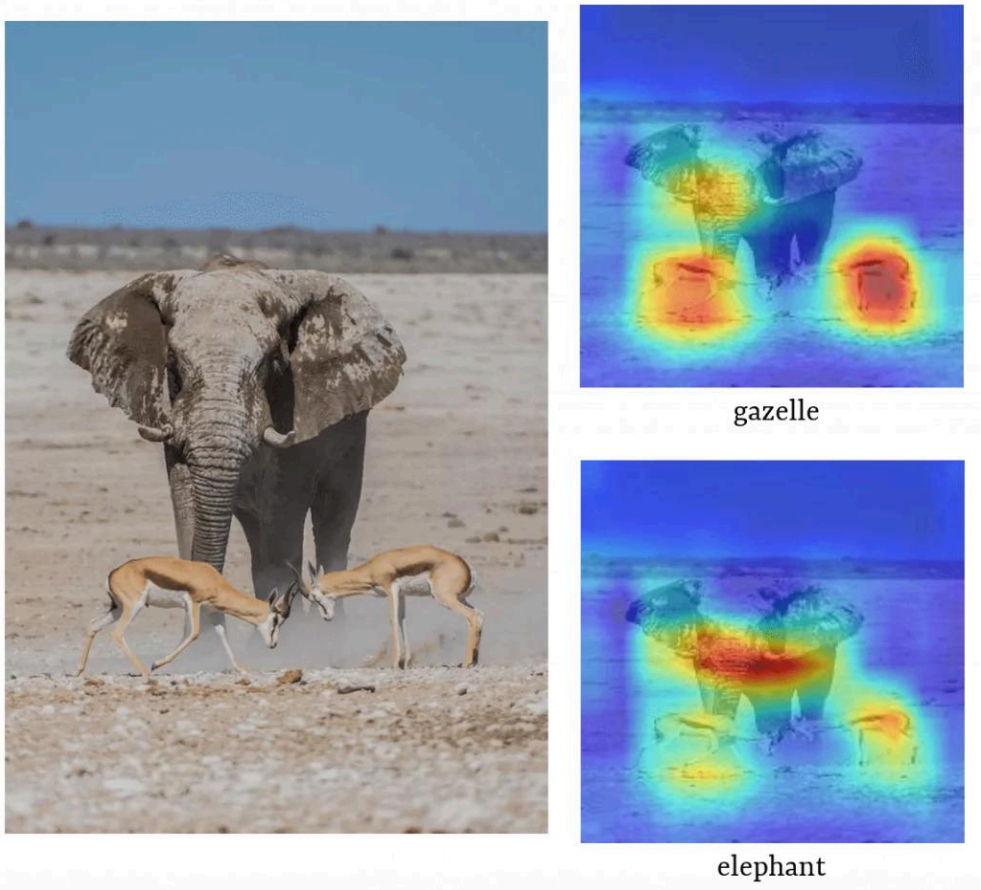
b. **Logistic Regression —** similar to linear regression (and uses the same formula, just with the added sigmoid function). We can understand its behavior inherently from the formula.
c. **Decision Trees —** These trees train on the input data and find out the best way to split it in order to make "rules" according to the values of each feature! The benefit here is that after the rules are built, we now know how each prediction is made, as we can follow the tree structure to explain the model's "thought process."



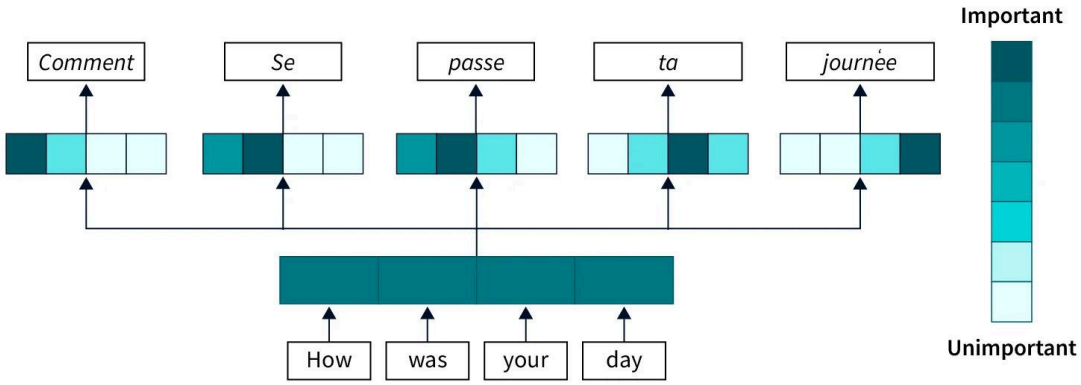
These models are inherently interpretable due to how simple & direct their inner workings are. However, if we rely on them, we are confining ourselves to simple

models, which means we are trading higher performance for interpretability. This isn't always the correct choice, so we need to think about our use case when deciding whether or not to use these models. What should we do if we don't want to compromise our performance?

2. **Post-hoc —** Since we don't want to compromise our performance with simple models, we can use post-hoc methods, which are applied **after** a model is trained (usually a black box like neural networks, ensembles, etc.). They aim to **explain a model's decisions without changing its structure**. There are two types of post-hoc methods:
    a. **Model Specific —** a method that is particular for a specific model, which uses details **inside the model** (like weights or layers) to explain its prediction. Some model-specific:
        i. **Activation Maps in Neural Networks (like for an image) —**



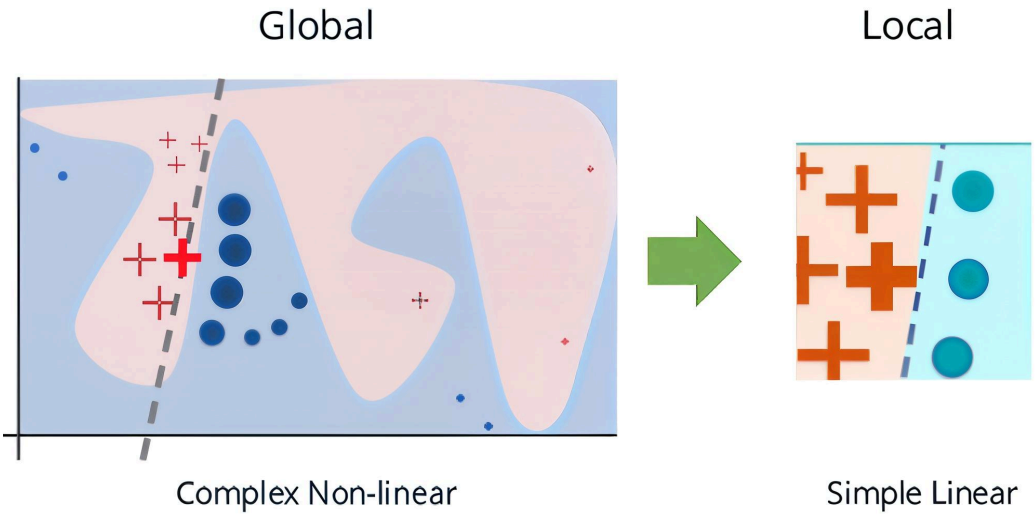
ii. **Attention Mechanisms in Transformers (which are a type of NN) —** in the below example, we can see how "important" each word is in predicting the French translation.



b. **Model Agnostic —** a method that does not depend on the model itself, uses only **inputs and outputs**, not the internal structure, to create the explanations. Some model-agnostic methods are:
   i. **SHAP** (SHapley Additive exPlanations) **–** based on game theory
   ii. **LIME** (Local Interpretable Model-Agnostic Explanations) **–** focuses on local "surrogate" models

Types of Explanations:

1. Global Explanations — Shows **how the whole model works** (like what features matter most overall)
2. Local Explanations — Shows **why one specific prediction** was made.



SHAP:

Let's imagine that I am playing a game with two of my friends as part of one team. In this game, depending on how we perform, we are awarded a specific amount of money as a cash prize.

If I were to play this game alone, I'd earn 2500 pounds, but when friend A and I play together, we get 3500 pounds. Friend A's prize if they were to play alone would be about 500 pounds. However, when our lazy friend B joins the two of us, he distracts us a little, and we get a total prize of 3000. Now imagine all the other possible scenarios we have, like friend A and B without me, friend B alone, friend B with me but without A, and so on.

Now, the truth of the matter is that we played all together as one team and earned 3000 pounds. Would it be fair to divide the 3000 pounds equally? We know that I did most of the work, but my friends also contributed to our score (be it positively or negatively). Deciding how to divide the cash prize and each player's "contribution" to the prize is basically what SHAP is all about.

If we take the above example and change it so that:

- The cash prize is the final prediction
- The players on the team represent the features (or attributes) of a single data point

We can then find the contribution of each feature (a.k.a. *feature attribution*) for a single prediction! SHAP does this by finding the baseline prediction (if we had no information, what would we predict), then for a specific feature X, finding all possible feature combinations without it (e.g., {A}, {B}, {A, B}, etc.) and seeing how the prediction differs before and after the inclusion of X.

We can also average the feature attributions for all features across the entire dataset to find the global feature importance score (which tells us how much each feature matters on average across all predictions).

LIME:

LIME utilizes a concept called "surrogate" models. The intuition behind this concept is: Since our black box model is too complex and captures complex global non-linear patterns, why not train another simpler (or intrinsically interpretable) model to mimic the behaviour of our black box? In this case, we will not train our surrogate model on the

inputs & the ground truth, but actually we will train our surrogate on the inputs and the black-box's predictions! This is because we aren't using our simple model to predict; we are using it to *learn* the black-box's decision-making strategy! However, LIME is limited to making local explanations, due to the fact that the nature of our surrogate model usually forces it to only capture simple linear relationships, which is why we didn't just use it for prediction in the first place. Take a look at the above figure comparing the global & local explanations. The local explanation is linear, which means that in order to train our surrogate model to provide an explanation for a specific data point, we need to:

- Create "perturbations" of this data point, which are **slightly modified versions of the original instance** obtained by **randomly changing some of its feature values**.
- These perturbed samples are then **fed into the black-box model** to see how the predictions change.

By observing how small changes in each feature affect the output, LIME can **learn a simple, interpretable model** (like a linear regression or decision tree) that **locally approximates** the complex model's behavior around that specific data point.

Transparency, Trust & Bias Detection:

When dealing with human beings, some ethical concerns come into play. To the model, it doesn't care whether or not you understand its inner workings; it just gives you a prediction! However, to the person which the model decides their fate, they really do care about the reasoning! Transparency is important as a person has the right to an explanation, whether it be that their loan was approved/denied or that our model predicted they are at a high risk of a disease.

Not only does the end user deserve an explanation, but there are other stakeholders at play. The doctor using the aid of the model to predict the risk of a disease will not **trust** the model blindly, but if the model gives them the reasoning, then they might feel more trusting of the results. Even for us programmers, who build the models, we are always happy with high accuracy, but when making life-changing decisions, we would want to debug the model's predictions and make sure they make sense. How would you do this if you couldn't comprehend what is causing these predictions? We understand the math behind the statistics and the weight updates in a neural network, but in the end, do the patterns the model catches make sense or not?
We also want to make sure that the model isn't biased in any way, especially when dealing with sensitive information such as age, gender, or race. Here, an issue arises; however, if the data I have is already biased, is it truly an issue for my model to learn

this bias and continue propagating it? One can argue that we are just ensuring we are accurate with the data; however, here is where we have to make a stop.
Let's take, for example, the Recidivism Dataset; in this dataset, there is an attribute called perpetrator_race. There was a higher tendency back then for officers to arrest more people of the African American race, which meant that the database of perpetrators was imbalanced. This would lead the model to learn this bias and predict that a person will reoffend due to their race. If we are using this model to predict future reoffenders, then it isn't fair for us to decide a person's fate because of an officer's prejudice. However, for the sensitive attributes, we need to be smart about them. If we are predicting whether a person has a disease that is highly prevalent in women, then gender is a fair and important attribute to include. However, if we are predicting whether a person has earned a promotion or not, gender wouldn't be a fair feature to include in the decision-making.

In this lab, you'll learn to:

1. Interpret feature importance in trained ML models.
2. Explain **individual predictions** and **overall model behaviour.**
3. Discuss the relevance of **fairness**, **trust**, and **debugging** in AI

Required Readings:

1. SHAP GitHub Documentation
2. LIME GitHub Documentation

Optional Readings:

3. SHAP Inner Workings
4. LIME Inner Workings

Required Watch List:

Note: This is just to get an overview of how these conceptually work! If they start getting into math, you can skip that part!

1. SHAP Overview
2. LIME Overview

Part 2: In-Lab Task

1. Set Up Your Environment.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Install explainability libraries
!pip install shap lime
import shap
import lime
import lime.lime_tabular
```

2. Load and prepare the data.

```
df = pd.read_csv('/kaggle/input/titanic/train.csv')

# Clean & prepare
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)
df = df.drop(columns=['Cabin', 'Name', 'Ticket'])

X = df[['Pclass', 'Sex', 'Age', 'Fare', 'SibSp', 'Parch']]
y = df['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

3. Train a decision tree and visualize it.

```
tree = DecisionTreeClassifier(
    max_depth=2,       # smaller depth → simpler tree
    min_samples_split=20,  # don't split unless enough samples
    random_state=42
)
```

```
tree.fit(X_train, y_train)

# Evaluate
print(f"Accuracy on test set: {tree.score(X_test, y_test):.2f}")

# Visualize
plt.figure(figsize=(10, 5))
plot_tree(
    tree,
    feature_names=features,
    class_names=['Did not survive', 'Survived'],
    filled=True,
    rounded=True,
    fontsize=11
)
plt.show()
```

4. Train a random forest classifier.

   A **Random Forest Classifier (RFC)** is a machine learning model that makes predictions by combining the results of many **decision trees**. Each tree looks at a slightly different subset of the data and features, then votes on the final answer.

   While a single decision tree is easy to understand (you can follow its "if—then" rules step by step), a Random Forest becomes much harder to interpret because it's made up of so many trees. The process of training multiple trees on data samples (called **bagging**) helps the model make more accurate predictions, but it also turns it into a **black box**. That means it's difficult for humans to see *why* the forest made a particular decision, even though each tree on its own is interpretable.

```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
```

4. Global feature importance with SHAP. (SHAP is model agnostic however there are more optimized versions of SHAP depending on the use case)

```
explainer = shap.TreeExplainer(clf)
shap_values = explainer.shap_values(X_test)
```

```
shap.summary_plot(shap_values[1], X_test, feature_names=X.columns)
```

5. Local feature importance with SHAP.

```
i=0 #Explain row 0
shap.initjs()
shap.force_plot(
    Explainer.expected_value[1], #1 is the class we want to explain(survived)
    shap_values[1][i],
    X_test.iloc[i],
    feature_names=X.columns
)
```

6. Local feature importance with LIME.

```
explainer_lime = lime.lime_tabular.LimeTabularExplainer(
    X_train.values, #### .values transforms our dataframe into a numpy array
    feature_names=X.columns,
    class_names=['Not Survived', 'Survived'],
    discretize_continuous=True
)
exp = explainer_lime.explain_instance(
    X_test.values[i], #### .values transforms our dataframe into a numpy array
    clf.predict_proba,
    num_features=6
)

exp.show_in_notebook()
```

Part 3: Graded Task

1. **Train your neural network** again (or reuse it from Lab 3) on the cleaned Titanic dataset.
2. Use **SHAP** to:
   - Generate a **global feature importance** summary plot.
   - Create a **local force plot** for one prediction.

3. Use **LIME** to explain a **different prediction** from the model.

   Extra help (SHAP setup for neural networks):

```
# Wrap your model prediction function
def predict_fn(X):
    return model.predict(X)
# Use Explainer -> finds the best XAI model to use for your model
# Use a small background (reference) set to speed up explanations
# If X_train_scaled is a DataFrame, shap.sample will preserve columns
background = shap.sample(X_train_scaled, 100, random_state=42)

# Let SHAP pick the best algorithm for your Keras model
explainer = shap.Explainer(model, background)
shap_values = explainer(X_test_scaled)


# Global summary
shap.summary_plot(shap_values.values, X_test_scaled,
feature_names=X_test_scaled.columns, plot_type='bar')

# Local explanation
i = 0

# Easiest way with the new API
shap.plots.force(shap_values[i])
```

Deliverables:

Submit an .ipynb notebook with all the completed tasks above to the form. The deadline for submission is **@11:59 PM the day before your lab during the week starting 18/10**. The evaluation will be held in the labs running during the aforementioned week.