# ARTIFICIAL NEURAL NETWORK
# COURSE WORK

W O M Fernando

STUDENT NO: COMScDS241P-019

# Contents

# Question 1: Dataset Preparation and EDA

a. Dataset Preparation

**About the dataset**

This dataset is related to engineering employees related 30,000 salaries across seven different countries like Sri Lanka, Australia, Germany, India, Sweden, UK, USA and Industries including Tech, Finance, Consulting, Healthcare, Retail. Dataset included engineering positions like Software engineering, Full Stack engineering, Data scientist, Data engineering, Lead engineer, Senior software engineering, Software architect. This is a synthetically generated dataset which includes attributes related to engineering work culture, earnings, age like personal data, experience and qualifications.

**Preprocessing and preparation**

Check for missing/null values:
Dataset doesn't have missing values and null values because the dataset generated synthetically.

Prepare columns:
"AdditionalBenefits" column has multiple values commas separated. Throughout the dataset these values calculated into one array removing duplicates and all the unique values presented as a new column and add to the dataset as a categorical value. Each row, based on the values in "AdditionalBenefits" column, True or False set to the additionally added columns. In this way I can use these columns meanings into the model.

Standardization:
Standardization of a dataset is a common requirement for many machine learning models to treat the outliers. This dataset has numerical and categorical values which need to standardize before using it in training or testing the model. In the code I categorized the numerical and categorical columns separately to standardize. Categorical values standardized into 0/1 values and numerical values standardized to between zero and one.

b. EDA Findings

Data structure

- Total record count 30,000
- Features 41 variables
- Target Variable is the SalaryUSD (continues variable)
- The dataset columns have mix of categorical (e.g. education', 'Industry', 'Job Title', 'location', 'gender', 'RemoteOnsite', 'additionalBenifits') and numerical columns like ('ExperianceYears', 'Certifications', 'SalaryUSD', 'PreviousCompanies', 'Age', 'CompanySize')
- Key numerical columns are SalaryUSD ($10,500 - $745,000), ExperianceYears (0 - 20), Age (20s – 50s)
- Geographic Scope is global (Include salaries from 6 countries)

Classification of variable types

- Numerical Variable details (4)
    - ExperienceYears (0 – 20 years)
    - Certifications (0 - 5)
    - SalaryUSD (This is the target variable)
    - Age (20 – 50 years)
    - CompanySize (10 – 199,888)

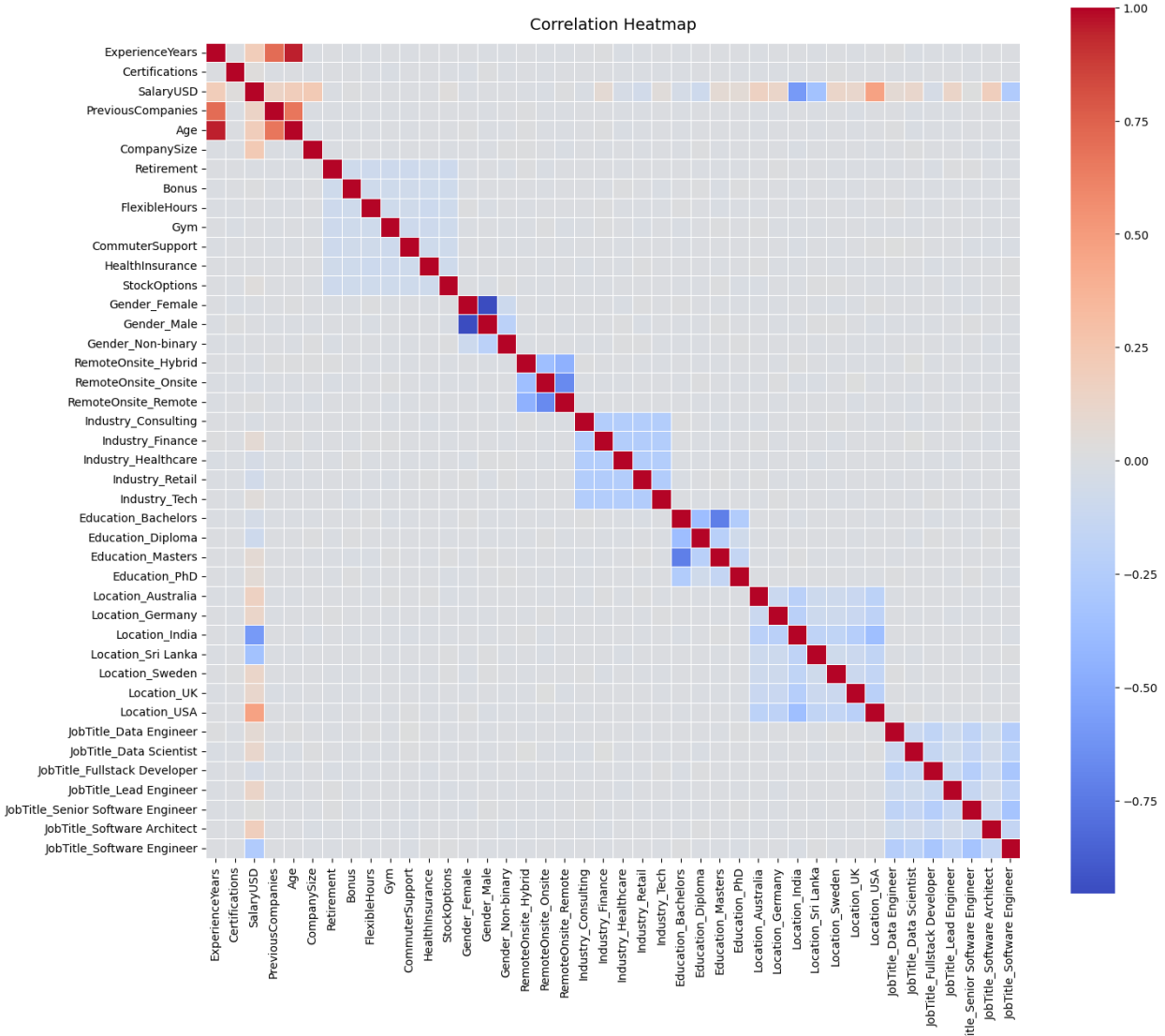
- Categorical Variable details (8)
    - Education (4 levels - Diploma, Bachelors, Masters, PhD)
    - Industry (5 categories – Tech, Healthcare, Finance, Consulting, Retail)
    - JobTitle (7 positions – Software engineering, Full Stack engineering, Data scientist, Data engineering, Lead engineer, Senior software engineering, Software architect )
    - Location (7 countries - Sri Lanka, Australia, Germany, India, Sweden, UK, USA)
    - Gender (3 categories – Male, Female, Non-binary)
    - RemoteOnsite (3 categories – Remote, Onsite, Hybrid)
    - AdditionalBenifits (can have multiple values)

Insights into the dataset

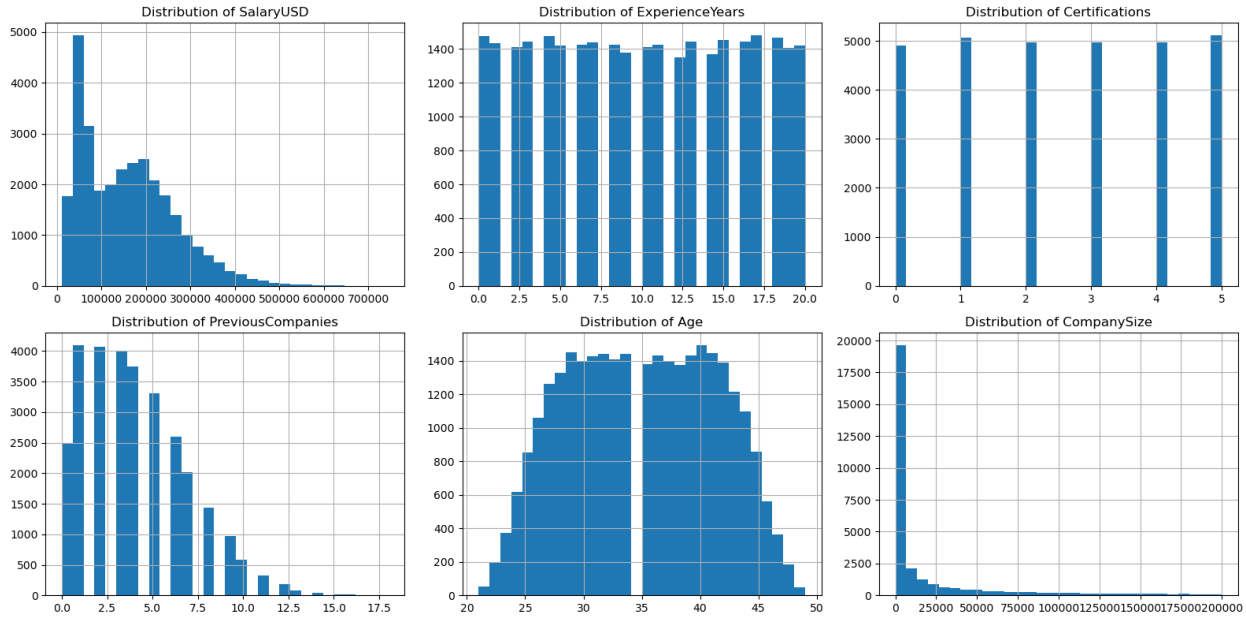
- Wide salary range suggest diverse experience levels and roles. Min salary $10,669, max salary is $744,333 and median as $ 466,85
- Global dataset represents a major tech hub.
- Mix of modern and traditional work arrangements.
- Some employees have 0 years of experience but multiple previous companies.
- In some cases, high salaries are assigned to junior positions
- All salary values are positive
- No duplicate employee records are detected

| Feature | Count | Mean | Std Dev | Min | 50% | Max |
|---|---|---|---|---|---|---|
| ExperienceYears | 30000 | | | 0 | 10 | 20 |
| Certifications | 30000 | | | 0 | 3 | 5 |
| SalaryUSD | 30000 | 158,354.172 | 104,106.6083 | 10,669 | 146,685 | 744,333 |
| PreviousCompanies | 30000 | | | 0 | 4 | 18 |
| Age | 30000 | 34.99 | | 21 | 35 | 49 |
| CompanySize | 30000 | | | 10 | 1,435.5 | 199,942 |

# Correlation Heat Map



## Distribution of key numerical features



# Question 2: Solution Design

a. ANN Architecture

**High-Level Architecture Overview**

Network type and purpose

The implemented model, a fully connected feed - forward neural network design for a regression type task, especially for salary prediction. Architecture represents a deep learning approach for a non-linear relationship between input features and the continues target variable (predicted salary).

Architecture Summary

Input Layer → Hidden Layer 1 → Hidden Layer 2 → Hidden Layer 3 → Output Layer

(n) → (128) → (64) → (32) → (1)

Key Characteristics of the network

- Depth of the Network: 4 layers (3 hidden + 1 output)
- Width of the network: Narrowing from 128 to 1 neuron
- Architecture pattern: Pyramid/Funnel structure
- Parameter count: (41*128 + 128*64 + 64*32 + 32*1) + biases

**Layer-by-Layer Architecture Analysis**

Input Layer

- Input shape 41
- Data Type is normalized/scaled continues numerical features
- No activation

Hidden Layer 1: (Feature extraction layer)

- Neurons 128 for complex pattern recognition
- Weights: 41 * 128 + 128
- Used ReLU activation function
- Used batch normalization function and dropouts 0.2
- The largest neuron count layer to learn complex problems to capture maximum information

Hidden Layer 2: (Refine features)

- Neuron count 64. Reduced 50% from the initial count
- Weight: 128 * 64 + 64
- ReLU activation function
- Used batch normalization function and dropouts 0.2
- Progressively reduce the dimensionality creating a bottleneck

Hidden Layer 3:

- Neurons 32 further reduce to 50%
- Weights: 64 * 32 + 32
- ReLU activation function
- Used batch normalization function and dropouts 0.2
- High level abstraction layer before the final output

Output layer: (Prediction layer)

- Neurons 1 (output as a single continues value)
- Weights: 32 * 1 + 1
- No activation function used in the output layer (linear output for regression)

# Question 3: Model Development & Evaluation

a. Model development and optimization process

Problem context
Developing a neural network system to predict employee salaries based on multiple factors, including experience, education, skills, location, sector, and performance metrics. The goal is to ensure fair compensation practices and support human resources decision-making.

Initial Baseline Model

Architecture

Base line model implements with a simple feed forward network. Following is the architecture of the network.

- Input layers: 41 features (numerical and encoded categorical variables)
- Hidden Layer 1: 64 neurons with activation ReLU
- Hidden Layer 2: 32 neurons with activation ReLU
- Output Layer: 1 neuron (Output the predicted salary)

Initial Architecture

Model

├— Dense(64, activation='relu', input_shape=(41))

├— Dense(32, activation='relu')

└— Dense(1, 'linear')

Optimizer : Adam optimizer (learning-rate=0.01)
Loss function : Mean squared error (MSE)
Batch size : 32
Epochs: 400

This base line shows predictive capabilities. But it seems always overfitting and struggles to properly predict for extreme values. So decided to add early stopping to capture the models optimum level of the optimized version and changed the model structure too. And did some changes to train, validate and test dataset portions. Previously the test size was 0.2 but it increased to 0.4 later.

Following is the optimized deeper network architecture of the model

FullyConnectedNeuralNetwork(

Sequentially execution (

Linear ( input features = 41, output features = 128, bias = true)

BatchNorm1d (128, eps = 1e-05, momentum = 0.1, affine = true, track-running-stats = true)

ReLU ()

Dropout (p = 0.2, in-place = false)

Linear (in-features = 128, out-features = 64, bias = true)

BatchNorm1d (64, eps = 1e-05, momentum = 0.1, affine = true, track-running-stats=true)

ReLU ()

Dropout (p = 0.2, in-place = false)

Linear (in-features = 64, out-features = 32, bias = true)

BatchNorm1d (32, eps = 1e-05, momentum = 0.1, affine = true, track-running-stats=true)

ReLU ()

Dropout (p = 0.2, in-place = false)

Linear (in-features = 32, out-features = 1, bias = true)

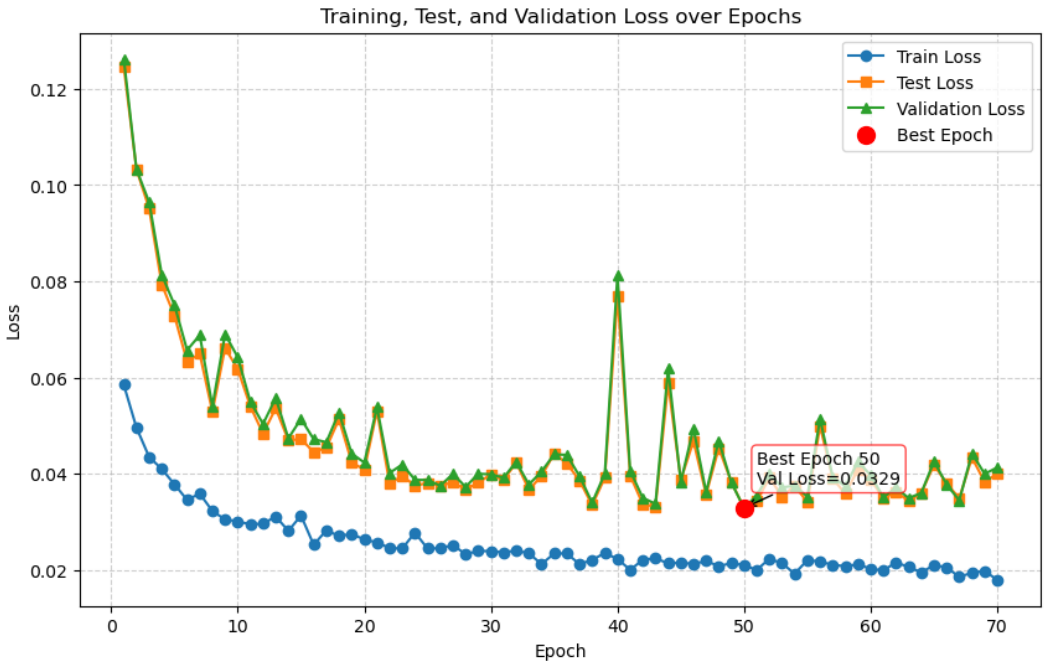
))

Optimizer: Adam (learning_rate=0.001)
Loss Function: Mean Squared Error (MSE)
Batch Size: 64
Epochs: 100

Even if I set up the Epoch to 100 the model stops early at Epoch 50. Following are the stats of the model at epoch 50. As improvements,

- I have added early stopping technique with the patience 20
- Hyperparameter tuning (change the batch size and the learning rate)
- Model level added another layer for the model (Started from 128, 64, 32, 1)

Epoch 50:
Train Loss: 0.1022
Test Loss: 0.0328
Validation Loss: 0.0329



b. Optimization techniques for ANN

## I. Optimization Process

Optimization is a main vital process in ANN. It is an iterative process to adjust the model parameters weights and biases to minimize the loss function of the model. Selecting the optimization technique and it's hyperparameters is a very critical to achieve a fast convergence.

1. Variants of gradient descent

   The base algorithm of the gradient descent is to minimize the cost function, adjusting its model parameters iteratively. The primary difference between it's variants is the data used for each parameter update.

   a. Batch Gradient Descent(Batch GD):
   This computes the whole input data set to update model parameters.
   **Pros**: Convergence is stable
   **Cons**: This might get slow for lager datasets, computationally and memory intensive for large datasets.

   b. Stochastic Gradient Descent (SGD)
   Instead of using entire dataset to compute the gradient decent SGD use a small batch to compute the gradient decent (Use a single randomly selected dataset at a time).
   **Pros**: Much faster compared to Batch Gradient Descent, less computationally intensive, Inherent noise helps to avoid local minima and saddle points.
   **Cons**: Noisy updates can lead to less or unstable convergence.

   c. Mini Batch Gradient Decent:
   Update parameters processing a small batch of training data.
   **Pros**: This method identified as the best of the balance between the SGC speed and the stability of the Batch GD.
   **Cons**: Since this is introducing a new hyperparameter called batch size which required a tuning.

   d. Momentum

This can accelerate the GD by adding fraction helping the optimizer to continue in a consistent direction.
**Pros**: It helps the optimizer to push through the flat regions and escape the shallow local minima which faster the converges.
**Cons**: Required tuning due to adding another hyperparameter.

2. Adaptive Learning Rate Methods (RMSprop, Adam)

   These algorithms adjust learning rate for every parameter independently based on past gradients.

   a. RMSprop:
      An adaptive learning rate algorithm that adjusts the learning rates for each parameter independently based on a moving average of the squared gradient.
      **Pros**: RMSprop normalizes the gradient to prevent oscillations and achieve smoother and faster convergence, especially for non-stationary objective functions.
      **Cons**: Hyperparameters such as decay rate need to be adjusted

   b. Adam:
      It is a very popular and effective optimization tool that combines the benefits of both Momentum and RMSprop. It calculates an adaptive learning rate for each parameter based on estimates of both the first moment (mean, like Momentum) and second moment (uncentered variance, like RMSprop) of the gradient.
      **Pros**: It is computationally efficient, has low memory requirements, and is suitable for problems with large data sets and parameters. It also features bias correction, which improves the accuracy of moment estimates, especially in the early stages of training.
      **Cons**: It is more complex than other methods and requires several hyperparameters, but the default values often work well.

3. Initialization and stability:
   Strategies to ensure a good starting point for training and maintain numerical stability.

   a. Batch Normalization:

Normalize the input of each layer so that the mean of each minibatch is 0 and the variance is 1.
**Pros**: It stabilizes and accelerates training/ improves flow of gradient/ decreases the model's sensitivity to the weight initialization which helps to mitigate "internal covariate shift" where the distribution of layer inputs changes during training.
**Cons**: Introduces extra computational overhead during training.

b. Early Stopping:
This is a practical and effective way to prevent overfitting by monitoring the model's performance on the validation set during training. Also, it helps to mitigate overfitting.
**Pros**: An effective and easy way to prevent overfitting and reduce training time.
**Cons**: Validation performance should be closely monitored, and training may be stopped prematurely if the validation loss fluctuates temporarily.

4. Overfitting Mitigation
Techniques to improve model generalization and avoid fitting noise in the training data.

a. L1 and L2 Regularization:
Depending on the size of the model's weight, a penalty term is added to the cost function. L1 (Lasso) adds the sum of absolute values, while L2 (Ridge) adds the sum of squared values.
**Pros**: Effectively prevent models with excessive weights and complexity, and improve generalization capabilities.
**Cons**: New regularization parameters that need to be adjusted

b. Dropout:
In each training iteration, a random portion of neurons are temporarily "dropped" or ignored.
**Pros**: A powerful yet simple technique that prevents neurons from co-adapting excessively and forces the network to learn more robust and redundant representations. It can be viewed as training an ensemble of many smaller subnetworks, which is thought to improve generalization.
**Cons**: Introducing dropout rate as hyperparameter that needs to be adjusted.

## II. Optimization Techniques

Hyperparameters are external configurations of a model that are set before the training process begins and are not learned from the data. Tuning hyperparameters is crucial for optimizing model performance because their values directly control the behavior and efficiency of the learning process.

1. Learning Rate
   This is arguably the most important hyperparameter. It controls the step size taken during each parameter update.

   **Role**: A larger learning rate and a larger step size may lead to faster convergence, but may exceed the minimum and cause divergence. A smaller learning rate may ensure stable convergence, but the speed may be very slow and may cause the model to fall into a local minimum.

   **Impact**: The ideal learning rate should be high enough to quickly escape from local minima, but low enough to allow for stable convergence. Techniques such as learning rate scheduling (e.g., decreasing the learning rate over time) or cyclic learning rates can help achieve both goals.

   **Tuning Strategy**: Start with a logarithmic search (e.g., 0.1, 0.01, 0.001). Use the Learning Rate Finder: Train for a few epochs while exponentially increasing the learning rate and plot the loss. The optimal learning rate is usually where the loss decreases the most.

2. Batch Size
   This determines the number of training examples used to process a single estimate of the gradient.

   **Role**:
   For small batch size: Provides a noisy stochastic estimate of the true gradient. This noise can act as a regularizer, helping the model generalize better. However, it is computationally less efficient per iteration (requiring more updates) and can lead to convergence instabilities.
   For larger batch size: They provide a more accurate estimate of the true gradient, leading to more stable and direct convergence in each period. They are also

more amenable to parallelization. However, they often converge to sharp minima that generalize poorly and require more memory.

**Impact**: There is a trade-off between computational efficiency and generalizability. The "generalizability gap" observed in large classes is a key research topic.

**Tuning Strategy**: It is often limited by the available GPU memory. A common starting point is 32 or 64. As a general rule, when you increase the batch size by a factor of *k*, you can often increase the learning rate by a factor of $\sqrt{k}$ to maintain stability.

3. Network Depth
   This denotes to the number of layers in the neural network.

   **Role**: Depth is a key factor in the representational power of deep learning. Deeper networks can learn hierarchical features - simple patterns (edges) in early layers, complex patterns (faces, objects) in later layers.

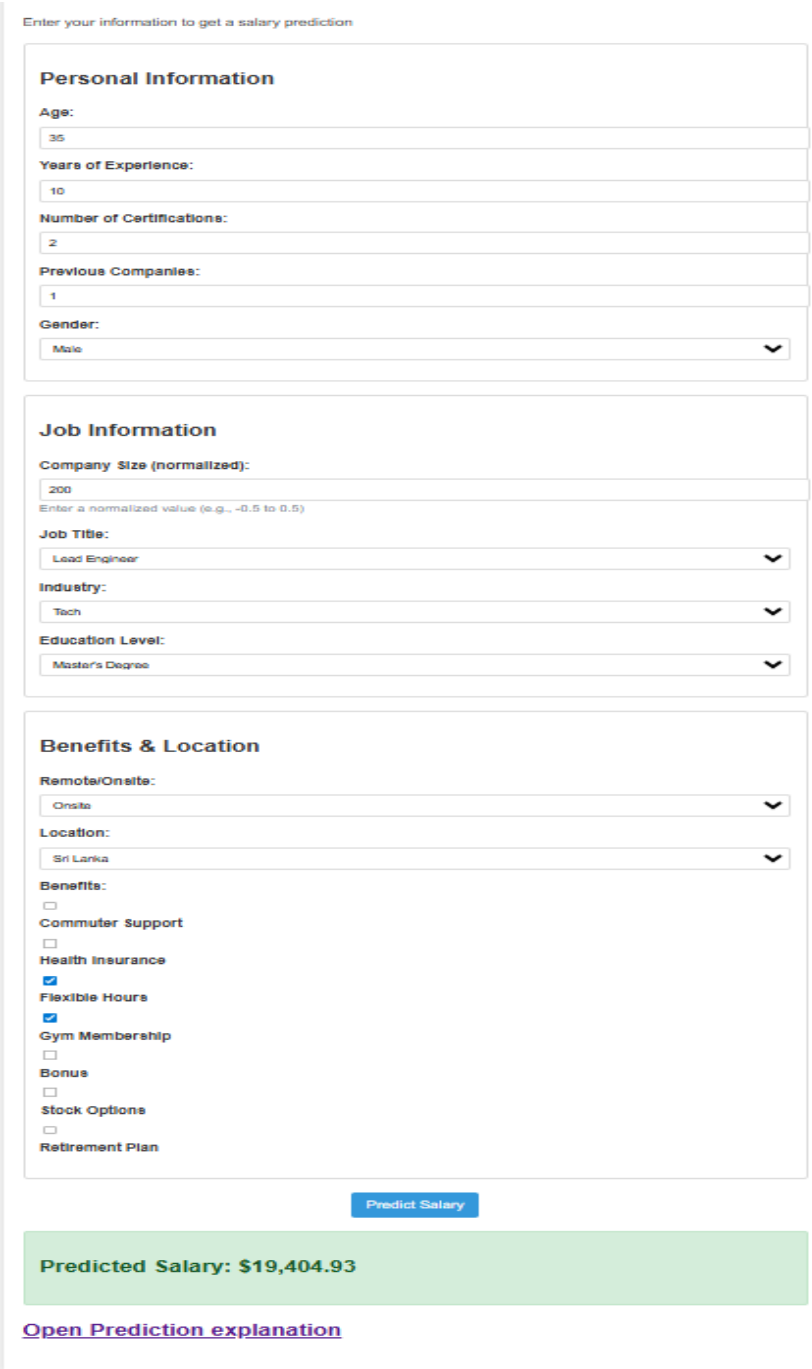
   **Impact**:
   *Too Shallow*: The model may lack the ability to learn the underlying complexities of the data, leading to underfitting.
   Too Deep:
   - *Vanishing Gradient*: Gradients become too small or too large with backpropagation in many layers and stop learning. (Relured with ReLU, BatchNorm, residual connections).
   - *Degradation Problem*: Accuracy quickly saturates and then decreases as layers increase, indicating that deeper networks are more difficult to optimize, not less expressive. (Solved by Residual Networks/ResNet).

   **Tuning Strategy:** Start with a well-known and proven architecture (e.g. ResNet-50 for image tasks). Increasing depth should be done to increase the capacity of the model, but it should be accompanied by techniques such as batch normalization and connection rejection to ensure that the network remains trainable.

# Question 4: Web Application

Following is the sample images of the implemented web application.

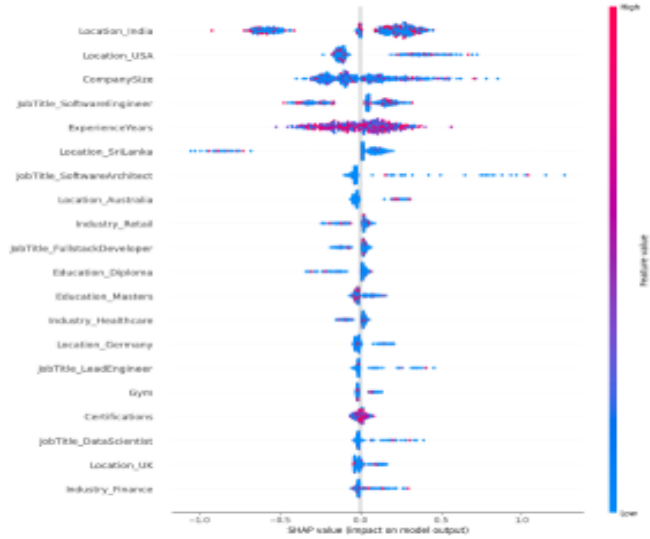
User interaction form to enter the values to get the predicted salary.



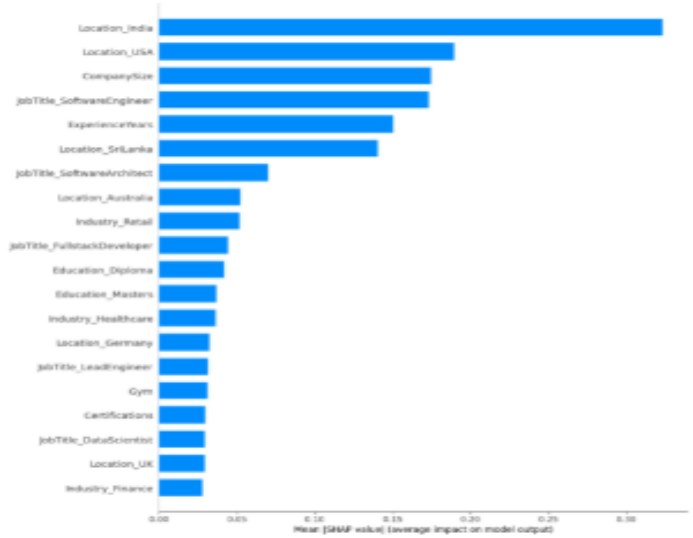
Following is the predicted result explainable AI page.

**AI Explainable Dashboard**

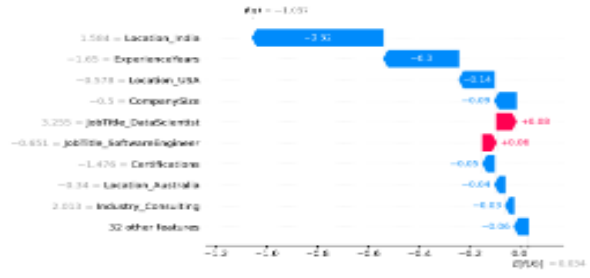
**SHAP Summary Plot**




**SHAP Bar Plot**




**SHAP Waterfall Plot**



# Question 5: Explainable AI

1. Method Comparison

Explainable AI is a set of techniques and methods which is used to interpret the decisions taken from an AI model understandable to humans. Without the explainable AI, the model will be a “Blackbox” and the outputs or the results generating from the model don’t have a proper transparency how the decisions are made. XAI will give insights into why model made the prediction, it enhances transparency, trustworthiness and improves debugging capabilities.

XAI methodologies can broadly divide into two categories.

1. Intrinsic (Model-Specific)
2. Post-hoc (Model-Agnostic)

Intrinsic (Model-Specific):

Such models are interpretable by design. Their processes of decision making are simple and can be comprehended easily without the help of a separate explanation model.

Ex: Decision Trees, Rule-Based Systems

Post-Hoc (Model-Agnostic):

These methods are used after training a model. They help explain complex models, often called "black boxes," like deep neural networks and gradient boosting machines. They do this by looking at the connection between input features and model outputs.

- LIME (Local Interpretable Model-agnostic Explanation):
  It explains individual predictions by closely modeling the situation with an interpretable substitute, which is often a linear model. It changes input data points and learns a local interpretable model to estimate the prediction.
- SHAP (Shapley Additive Explanation):
  Uses ideas from cooperative game theory, specifically Shapley values, to fairly assign the contribution of each feature to the model's output.

SHAP offers both local and global interpretability and has solid theoretical support.

- Partial Dependance Plots (PDPs):

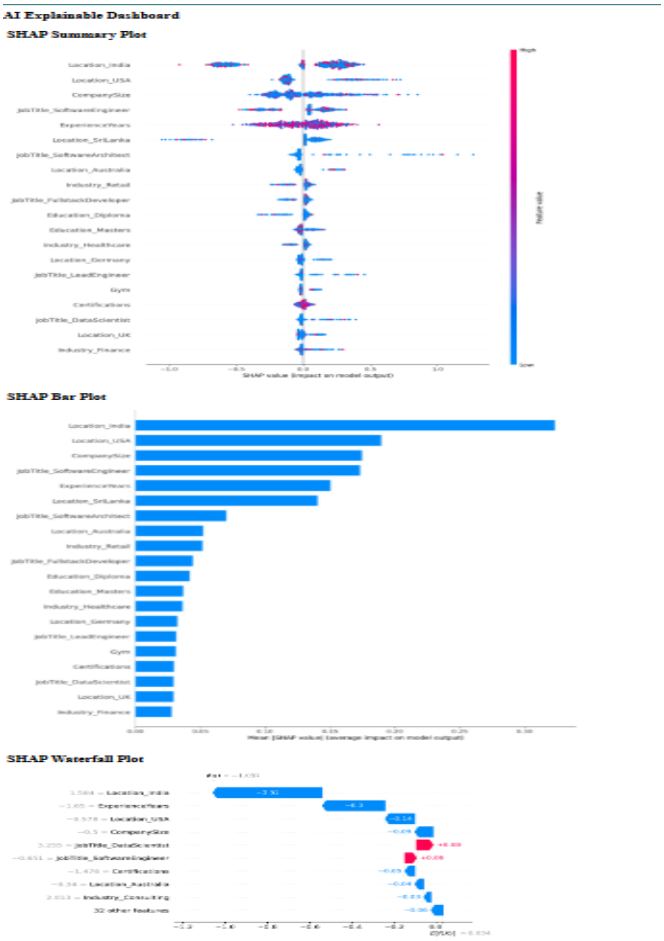
  Visualize how a feature affects the predicted outcome by averaging over other features.

- Individual Conditional Expectation (ICE):

  Like PDP but shows one curve per case, revealing the heterogeneity of feature effects.

2. Implementation an explainable AI

Following is the screen shot of the implemented explainable AI integration with the developed model.



# Question 6: GenAI Implementation

b. GenAI Implementation

gemini-2.5-flash output of the chosen problem

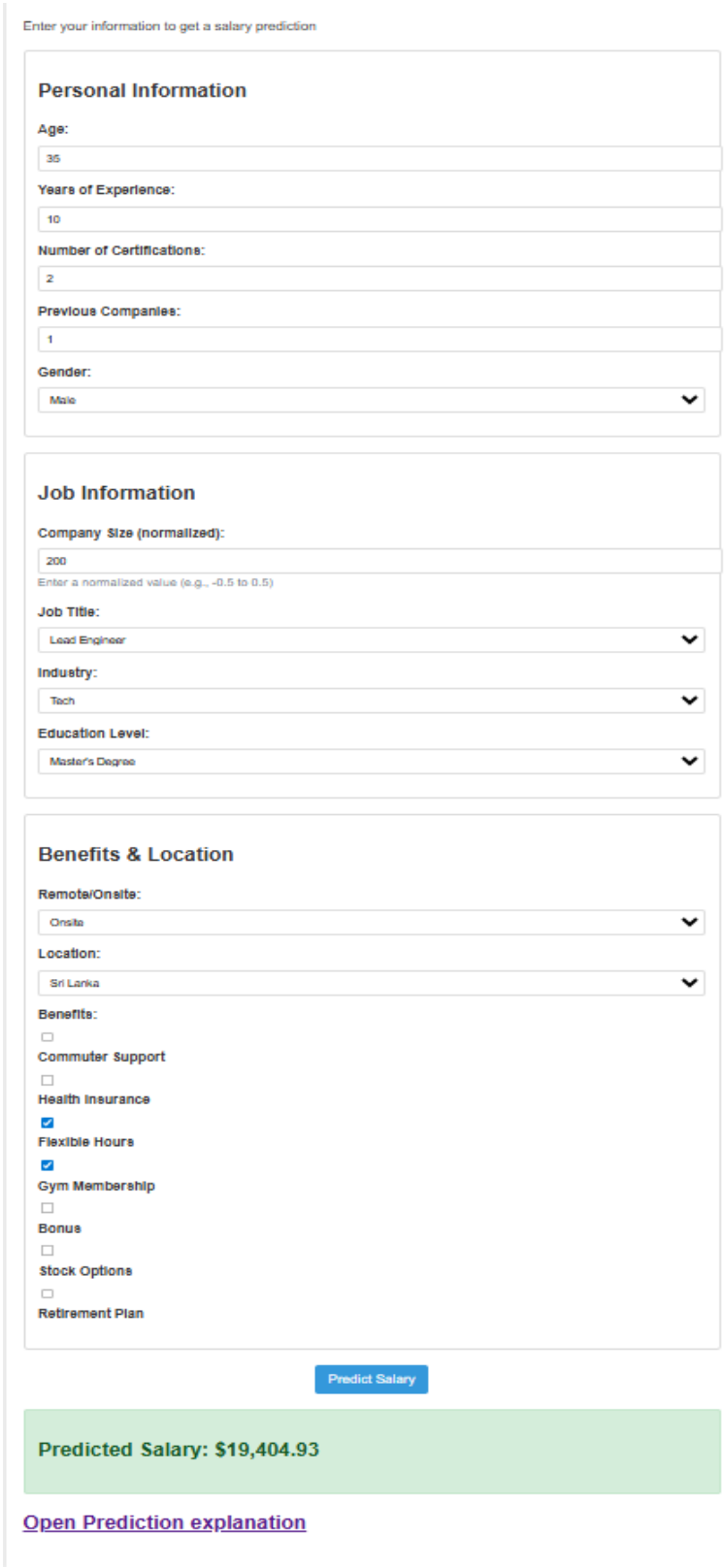
```
Let's break down the attributes and their likely impact on salary:

**Positive Factors:**

*   **ExperienceYears: 10** (Very strong positive)
*   **Age: 35** (Consistent with high experience)
*   **Certifications: 2** (Adds value)
*   **HealthInsurance: 1, FlexibleHours: 1** (Indicates a more established company, though these are benefits, they often correlate with better base pay)
*   **Industry: Tech** (Generally higher salaries than other industries)
*   **Education: Masters** (Highest degree, strong positive)
*   **JobTitle: Lead Engineer** (Senior, leadership role, commands higher pay)
*   **CompanySize: 200** (Medium-sized, often able to pay competitive salaries)
*   **Gender_Male: True** (Historically, may sometimes correlate with higher pay, though ideally a neutral factor)
*   **RemoteOnsite_Onsite: True** (Can sometimes command slightly higher pay or be neutral)

**Negative/Neutral/Missing Factors:**

*   **PreviousCompanies: 1** (Could be loyal or less varied experience; largely neutral for prediction but some argue less job hopping can mean less salary growth)
*   **CommuterSupport: 0, Gym: 0** (Missing benefits, not a direct salary impact but suggests less comprehensive package)
*   **Bonus: 0, StockOptions: 0, Retirement: 0** (These are significant parts of total compensation, especially in tech. Their absence means the *base salary* needs to carry more weight

**Dominant Negative Factor:**

*   **Location: Sri Lanka** (This is the *most critical* factor that will drastically pull down the salary compared to someone with similar qualifications in Western countries or even o
...
**Predicted Salary:**

I predict the base salary to be around **$35,000 USD per year**.
(This equates to approximately LKR 10,500,000 per year or LKR 875,000 per month).
```

Console output:

**I predict the base salary to be around **$35,000 USD per year**.**

**(This equates to approximately LKR 10,500,000 per year or LKR 875,000 per month).**

Custom model implementation trained model output is **$19,404.93**



c. Performance Analysis

Performance Comparison:

**Computational Requirement**: Locally implemented model have much lower computational overhead cause the traditional networks tackles a narrow well-defined problem. But large language models are built to handle open-ended generation. Because of this the implemented model runs faster than the Gen-AI model in local machines.

**Training Efficiency**: Locally implemented model trained faster cause it required less amount of data. But Gen-AI model typically need massive amount of data and time to train.

**Inference Speed**: Locally implemented model usually faster in inferencing mostly in millisecond. However the Gen-AI models will take more time for the inferencing.

Pros and Cons Analysis:

Traditional neural networks:

Advantages:

- Highly optimized for a specific task
- Computational resource requirement is very low
- Faster inference
- Test and validate the model is easy

Disadvantages:

- Problem domain in narrow
- Training and implementation is a task specific
- Scoped to the trained problem cannot generalized

Gen-AI models:

Advantages:

- Not limit to a one domain
- Ability to handle creative and open-ended problems
- Natural language interfaces reduce need for specialized programing
- Strong  learning capabilities

Disadvantages:

- High memory and computational requirements
- Output cannot be predicted, can hallucinate.
- Consistency and reliability is hard to ensure
- Latency might high for real-time applications

Novel opportunities and Enhancement applications:

**Adaptive learning systems**: GenAI models can provide meta-learning capabilities, helping traditional networks adapt to new domains or tasks more efficiently by providing strategic guidance on learning methods.

**Natural language programming**: GenAI enables traditional AI systems to be programmed using natural language descriptions rather than code, making AI development easier and enabling rapid prototyping of complex systems.

**Hybrid Architecture**: GenAI models can serve as intelligent coordinators for traditional neural networks. For example, an LLM can analyze a problem, select appropriate specialized models, and coordinate their outputs. This combines the efficiency of traditional networks with the reasoning capabilities of genAI.

**Scientific Discovery**: Using GenAI to generate hypotheses alongside traditional networks for simulation and verification.

**Personalized education**: GenAI for adaptive content generation combined with traditional networks for learning analytics.

**Creative industry**: GenAI for generating ideas and content, and traditional networks for quality improvement and quality control.

# References

- Salary prediction web application
  - Link: Salary-predictor-web-app
- Presentation
  - Link: AI_course_work_presentation.pptx
- All Notebooks
  - Link: Notebooks
- Dataset
  - Link: software_engineer_salaries.csv
- Video Recording
  - Link: COMScDS241P-019.mp4
- GitHub
  - Link: https://github.com/osandafdo/salary-prediction-system/tree/master
- SHAP implementation
  - Link: https://www.youtube.com/watch?v=MQ6fFDwjuco&list=PLqDyyww9y-1SJgMw92x90qPYpHgahDLIK&index=1