Task 1 - Sanjana Ramkumar BE24B038

Data Preprocessing

- I analysed the dataset using perplexity
- The dataset contains 48,842 instances with 14 features, including both categorical and numerical attributes
- Numerical features: age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week
- Categorical features: workclass, education, marital-status, occupation, relationship, race, sex, native-country
- The dataset was imbalanced and had more instances of income less than 50K
- **Handling missing values -**
    - The dataset had missing values in the categorical columns, written as ‘?’
    - workclass: 2,799 missing values
    - occupation: 2,809 missing values
    - native-country: 857 missing values
    - I chose to do a combination of both: I looked at the columns where there were many missing values and dropped those columns, and for the rest, I applied imputation
    - I looked at the pattern of missing values and figured out that 2 is the ideal number. Setting the threshold at 2 is very safe and drops only a negligible number of rows
- I then split the data BEFORE preprocessing to prevent data leakage
- **Ablation Study -**
    - To figure out the optimal combo, I integrated this within my code itself
    - I defined a function that evaluated this and printed the results
    - To quickly evaluate the model, I used a random forest
    - Evaluated it on both training and validation sets
    - Loops through the combinations
    - I converted the results into a pandas DataFrame for easy inspection and to identify the best combination

```
============================================================
SUMMARY OF RESULTS
============================================================
encoding  scaling  train_accuracy  val_accuracy  overfitting  status
  onehot   robust        0.999912      0.846879     0.153033 Success
  onehot standard        0.999912      0.846879     0.153033 Success
  onehot   minmax        0.999912      0.845923     0.153989 Success
 ordinal   robust        0.999912      0.851796     0.148116 Success
 ordinal standard        0.999912      0.852069     0.147843 Success
 ordinal   minmax        0.999912      0.851796     0.148116 Success
```

-
- But then I realised that we are implementing a neural network and that this combo might not be optimal for it (based on my research beforehand, I kinda knew that the optimal combo would be standard + one hot intuitively)
- I then changed my code for a sample neural net to find the optimal features
- One-hot preserves categorical feature relationships without having artificial ordinal relationships
- I decided to go with one hot for everything except education which I made ordinal because the order matter and the model should know that someone with less education should get a lower encoding value
- I previously used accuracy as my metric, but then realised that the F1 score might be a better indicator since this dataset is imbalanced
- It balances false positives and false negatives and makes sure the false positives are lesser
- Accuracy is just that true positives are max
- **Based on F1, this is the modified result -**

```
Running ablation: Encoding=onehot, Scaling=standard
Validation F1: 0.6947, Best Threshold: 0.62

Running ablation: Encoding=onehot, Scaling=minmax
Validation F1: 0.6905, Best Threshold: 0.64

Running ablation: Encoding=onehot, Scaling=robust
Validation F1: 0.6831, Best Threshold: 0.52

Running ablation: Encoding=label, Scaling=standard
Validation F1: 0.6746, Best Threshold: 0.58

Running ablation: Encoding=label, Scaling=minmax
Validation F1: 0.6681, Best Threshold: 0.68

Running ablation: Encoding=label, Scaling=robust
Validation F1: 0.6500, Best Threshold: 0.46

Running ablation: Encoding=ordinal, Scaling=standard
Validation F1: 0.6960, Best Threshold: 0.58

Running ablation: Encoding=ordinal, Scaling=minmax
Validation F1: 0.6866, Best Threshold: 0.66

Running ablation: Encoding=ordinal, Scaling=robust
Validation F1: 0.6843, Best Threshold: 0.52

Ablation Study Results:
  encoding   scaling    val_f1  threshold
6  ordinal  standard  0.696029       0.58
0   onehot  standard  0.694732       0.62
1   onehot    minmax  0.690493       0.64
7  ordinal    minmax  0.686645       0.66
8  ordinal    robust  0.684264       0.52
2   onehot    robust  0.683091       0.52
3    label  standard  0.674637       0.58
4    label    minmax  0.668083       0.68
5    label    robust  0.650013       0.46

Best Combination: Encoding=ordinal, Scaling=standard
Validation F1: 0.6960
```

- **Analysis -**
  - Standard worked best because it deals with input distribution that most ML algorithms are designed to work. In Minmax outliers get squeezed, reducing the discriminative power between normal and extreme values. Robust is too conservative and it removes important variance information
  - This suggests that the dataset has Meaningful outliers that are important
  - Normal-ish distributions
  - Features with different scales that benefit from standardization without losing variance

---

Model Building (From Scratch)

- The base architecture I used followed standard feedforward neural network principles
- Each hidden layer consisted of a linear transformation followed by an activation function
- The output layer used a single neuron with sigmoid activation to produce binary classification probabilities
- I stored the Linear layers (dense layers) in a list
- For each layer, added the linear layer, optionally added BatchNorm, optionally added dropout, chose the activation function
- **Dropout -**
  - Randomly setting a fraction of neurons to zero during training
  - This forces the network to learn redundant representations and reduces overfitting
  - used a dropout rate of 0.5
- **Batch normalisation -**
  - Normalises layer inputs (mean of 0, standard deviation of 1) for small batches of data
  - Reduces wild swings in activations
  - Allows for higher learning rates, accelerating convergence
  - Improves overall learning and generalisation
  - I placed it after a linear layer but before the activation function
  - Ensures clean, stable input for the activation function, optimising its non-linear operation
- **Ablation study -**
  - For this, I had 36 different combos and reduced it to 10 epochs to save time

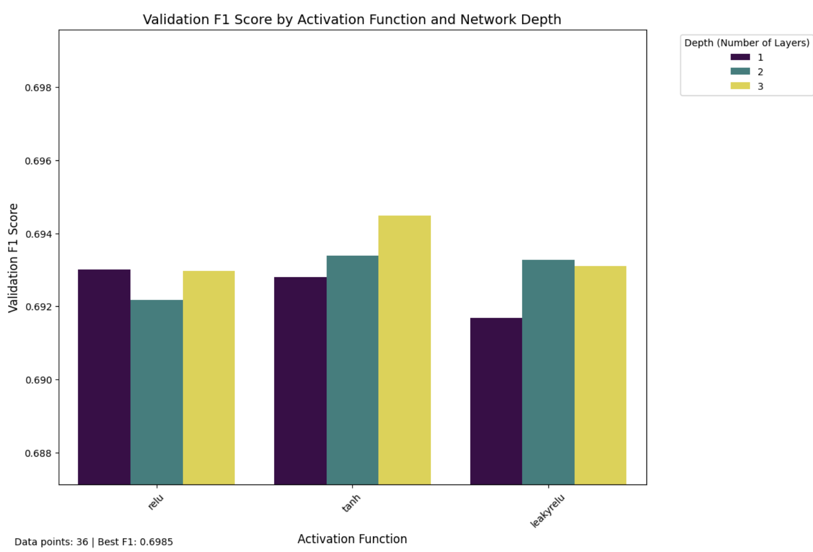
- Hidden layer architectures: [64], [128, 64], [256, 128, 64]
- Activation functions: ReLU, Tanh, LeakyReLU
- Dropout: True, False
- BatchNorm: True, False
- There are 3×3×2×2=36 total combinations
- Ran all the combos through a loop to figure out how the loss and accuracy were changing with different parameters
- **The one below is with F1 as the metric -**

```
Top 5 Configurations:
              hidden activation  dropout  batchnorm    val_f1
27  [256, 128, 64]       relu    False      False  0.698543
35  [256, 128, 64]  leakyrelu    False      False  0.698524
28  [256, 128, 64]       tanh     True       True  0.698468
29  [256, 128, 64]       tanh     True      False  0.698055
15       [128, 64]       relu    False      False  0.697201

Saved best configuration to best_36_config.json
```

- Creates a DataFrame with all configurations and their accuracies
- Sorts them to show the best performing ones first

- **Graph to visualise the results -**



- **Analysis -**
  - A deep architecture with 3 layers and many neurons can capture complex patterns in income prediction (especially non-linear relationships between features like education, hours/week, and occupation)
  - No regularization worked better here
  - Dropout and BatchNorm can sometimes hurt if the model is already generalizing well or when training data is large and clean
  - Avoiding dropout preserved full representational power

- ReLU activation tends to perform well in deep networks due to sparse activation (fewer neurons firing at once)
- Avoidance of vanishing gradients (which affects tanh more)

Custom Training Loop

- To evaluate the impact of different loss functions and optimisers, I built a custom training and evaluation loop using PyTorch
- I used a feedforward neural network with the best architecture identified earlier
- **Loss - implemented two variants of this architecture**
  - IncomeNetLogits (for BCEWithLogitsLoss) — returns raw logits
  - IncomeNetSigmoid (for BCELoss) — returns sigmoid probabilities
  - This way, I tested both loss functions cleanly
  - They both support customizable hidden layers (e.g., [128, 64]), activation functions (ReLU, LeakyReLU, Tanh) and optional Dropout and BatchNorm
- **I then made the custom reusable training loop - CustomTrainer**
  - This is a class with many components
  - Tracks training loss, validation accuracy, and learning rate
  - Implements checkpointing (saves the best model)
  - Handles both sigmoid-based and logits-based outputs
  - Saved the best model using torch.save()
- **Ablation study -**
  - I tested 2 loss functions × 3 optimizers × 3 learning rates = 18 combinations
  - Loss Functions: BCEWithLogitsLoss, BCELoss
  - Optimisers: SGD, Adam, RMSprop
  - Learning Rates: 0.1, 0.01, 0.001
  - Each configuration was trained for 50 epochs
  - For each combo, the code trains the model, tracks metrics like best_val_accuracy, final_train_loss, and convergence_epoch
  - I then stored all the results in a dataframe
  - This identified the best overall combo
  - Best generalising model (least variance in val accuracy)
  - Final model is trained using the best configuration and saved to disk (final_best_model_weights.pth and final_best_model_config.json)

- **Observations -**

```
Running: BCE, Adam, LR=0.001
Running: BCE, Adam, LR=0.01
Running: BCE, Adam, LR=0.1
Running: BCE, SGD, LR=0.001
Running: BCE, SGD, LR=0.01
Running: BCE, SGD, LR=0.1
Running: BCE, RMSprop, LR=0.001
Running: BCE, RMSprop, LR=0.01
Running: BCE, RMSprop, LR=0.1
Running: BCEWithLogits, Adam, LR=0.001
Running: BCEWithLogits, Adam, LR=0.01
Running: BCEWithLogits, Adam, LR=0.1
Running: BCEWithLogits, SGD, LR=0.001
Running: BCEWithLogits, SGD, LR=0.01
Running: BCEWithLogits, SGD, LR=0.1
Running: BCEWithLogits, RMSprop, LR=0.001
Running: BCEWithLogits, RMSprop, LR=0.01
Running: BCEWithLogits, RMSprop, LR=0.1

Top Performing Config: {'loss': 'BCEWithLogits', 'optimizer': 'Adam', 'lr': 0.1, 'val_f1': 0.706359945872801}
```

- **Analysis -**
  - BCEWithLogitsLoss combines the sigmoid activation and binary cross-entropy into one function
  - This makes it more numerically stable and avoids issues like vanishing gradients that can occur when using BCE with a manual Sigmoid
  - So, it’s better suited for raw logits output

  - Adam is adaptive
  - It adjusts the learning rate per parameter and includes momentum, which makes training faster and more stable
  - SGD requires careful tuning of the learning rate and is slower to converge
  - RMSprop didn’t work well maybe because it doesn’t combine as well with BCEWithLogits as Adam does

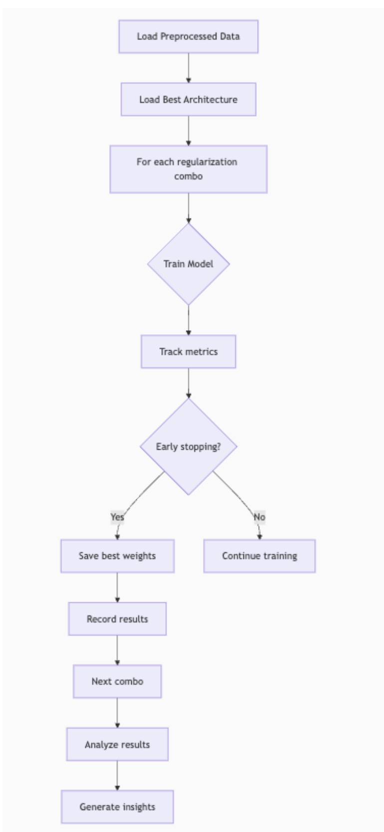
  - A learning rate of 0.001 is very stable but often too slow to converge in just 10 epochs, which might lead to underfitting
  - 0.01 is a balanced choice, but it still didn’t let the model reach optimal performance quickly enough
  - 0.1 worked best only when combined with Adam and BCEWithLogitsLoss

Regularisation & Overfitting Control

- This code implements regularisation and tries preventing overfitting
- Optimises model generalisation
- Systematically evaluates dropout, weight decay, and early stopping
- Identifies optimal regularisation configurations
- **I chose a design that has -**
  - Dropout layers at each hidden layer
  - Optional batch normalisation
  - Selectable activation functions (ReLU, LeakyReLU, Tanh)
  - Forward pass - Input → [Linear → BatchNorm → Activation → Dropout]×n → Sigmoid
  - Flexible architecture to test regularisation
- **Regularisation Techniques -**
  - Dropout - Tested 0.2, 0.5, 0.7
  - Weight decay (L2 Regularization) - Tested 0.0, 1e-5, 1e-4, 1e-3
  - Early stopping - Monitors validation loss, 15 epochs, restores best weights
- I then trained the regularised model and checked the best performing one using F1 again, overfitting gap, epochs trained and convergence stability
- **Ablation study -**
  - 3 dropout rates × 4 weight decays × 2 early stopping options = 24 experiments
  - Uses best architecture from previous task (best_income_model_config.json)

```
============================================================
ABLATION STUDY RESULTS (Sorted by F1 Score)
============================================================
 Dropout  Weight Decay Early Stopping   Val F1
     0.2       0.00000             No 0.702140
     0.5       0.00001             No 0.699911
     0.5       0.00010             No 0.699565
     0.7       0.00001             No 0.698194
     0.2       0.00001             No 0.697605
     0.5       0.00000             No 0.697005
     0.7       0.00000             No 0.696900
     0.7       0.00010             No 0.696119
     0.5       0.00100             No 0.693158
     0.2       0.00100             No 0.691452
     0.2       0.00010             No 0.691250
     0.7       0.00100             No 0.691185
     0.2       0.00000            Yes 0.690564
     0.7       0.00100            Yes 0.689908
     0.7       0.00001            Yes 0.688093
     0.5       0.00010            Yes 0.686781
     0.2       0.00010            Yes 0.682435
     0.5       0.00000            Yes 0.681875
     0.7       0.00000            Yes 0.680556
     0.2       0.00001            Yes 0.679295
     0.2       0.00100            Yes 0.679165
     0.7       0.00010            Yes 0.679118
     0.5       0.00100            Yes 0.677388
     0.5       0.00001            Yes 0.667327

============================================================
BEST CONFIGURATION:
============================================================
Dropout: 0.2
Weight Decay: 0.0
Early Stopping: No
Best F1 Score: 0.7021
```

- 

-

Analysis -

- Moderate dropout (0.2) provides strong regularisation without sacrificing too much model capacity
- Zero weight decay (0.0) allows the network to learn freely and surprisingly leads to highest F1
- Full 100 epochs training (no early stopping) allows optimal convergence and performance
- Best configuration relies on minimal regularisation and full training — model generalises well
- Dropout 0.5–0.7 is too aggressive for this dataset
- Excessive neuron dropping with higher dropout leads to underfitting
- Dropout 0.2 is optimal: enough regularisation to avoid overfitting while preserving expressiveness
- Weight decay = 1e-5 or 1e-4 gives moderate control but didn't beat the no-penalty setup
- Weight decay = 1e-3 consistently hurt performance, especially when combined with early stopping
- Early Stopping prevented the model from reaching peak accuracy (often stopped early)
- **Max performance configuration:**
    - Dropout: 0.2 + Weight Decay: 0.0 + No Early Stop
    - F1 Score: 0.7021
    - Best when maximum accuracy is the goal
    - Suggests regularisation via dropout alone is sufficient
- **Best overfitting control:**
    - Dropout: 0.5 + Weight Decay: 1e-5 + No Early Stop
    - F1 Score: 0.6999
    - Slightly lower accuracy, but strong regularisation
    - Best for robustness and stability in production
- **Balanced approach**
    - Dropout: 0.5 + Weight Decay: 1e-4 + No Early Stop
    - F1 Score: 0.6996
    - Great mix of accuracy and overfitting control
    - Good for general usage where both matter

Model Evaluation - Results from first test (I tuned the model after this to get better results) - Final results at the end

```
Accuracy: 0.3130
Precision: 0.2584
Recall: 1.0000
F1-score: 0.4106
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.10      0.18      5568
           1       0.26      1.00      0.41      1752

    accuracy                           0.31      7320
   macro avg       0.63      0.55      0.29      7320
weighted avg       0.82      0.31      0.23      7320

Confusion Matrix:
[[ 539 5029]
 [   0 1752]]
Normalized Confusion Matrix (by true class):
[[0.09680316 0.90319684]
 [0.         1.        ]]
Asian skipped – no samples in test set.
Hispanic skipped – no samples in test set.
Male misclassification rate: 0.6387
Female misclassification rate: 0.7862
White misclassification rate: 0.6745675848814863
Black misclassification rate: 0.7864357864357865
Asian misclassification rate: nan
Hispanic misclassification rate: nan
Other misclassification rate: 0.684931506849315
Performance Summary:
   Group  Accuracy  Precision  Recall  F1-score
 Overall  0.312978   0.258369     1.0  0.410641
    Male  0.361309   0.320285     1.0  0.485176
  Female  0.213839   0.125637     1.0  0.223229
   White  0.325432   0.275292     1.0  0.431732
   Black  0.213564   0.115260     1.0  0.206696
   Asian       NaN        NaN     NaN       NaN
Hispanic       NaN        NaN     NaN       NaN
   Other  0.315068   0.166667     1.0  0.285714
```

- This part of the code evaluates the model on test data
- First, I assessed the model based on the basic metrics - accuracy, recall, precision, and F1
  - Accuracy: Measures overall correctness
  - Precision: Fraction of predicted positives that are positive
  - Recall: Fraction of actual positives that were identified correctly
  - F1-score: Harmonic mean of precision and recall (balances false positives and false negatives)
- Extremely high recall (1.0) but very low precision (0.258) suggests the model predicts almost everything as positive (class 1)
- Accuracy is low (0.31) because the majority class (0: ‘≤50K’) is misclassified most of the time
- F1-score is low, confirming poor balance between precision and recall
- Class 0 (≤50K) has very high precision (1.0) but very low recall (0.10) - the model rarely predicts it, but when it does, it's correct
- Class 1 (>50K) is overpredicted, explaining its perfect recall but low precision
- **Confusion matrix -**
  - 539 true negatives (correctly predicted ≤50K)
  - 5029 false positives (predicted >50K instead of ≤50K)
  - 1752 true positives (correctly predicted >50K)

    - 0 false negatives (no >50K misclassified as ≤50K)
- **Normalised -**
- 90% of class 0 examples were wrongly classified as class 1.
- The model learned to favour class 1 extremely
- **Sex based metric -**

Metric	Male	Female
Accuracy	0.361	0.214
Precision	0.32	0.126
Recall	1	1
F1-score	0.485	0.223

- The model performs significantly worse for females across all metrics.
  The male subgroup receives higher precision and F1-score, suggesting potential gender bias
- **Race-based metric -**

Race	Accuracy	Precision	Recall	F1-score
White	0.325	0.275	1	0.432
Black	0.214	0.115	1	0.207
Other	0.315	0.167	1	0.286
Asian, Hispanic	N/A	N/A	N/A	N/A

- Black and Other groups have much lower precision and F1 than White
- Consistently high recall again suggests all subgroups are being over-classified as positive
- Misclassification Analysis - To quantify how frequently each group was incorrectly predicted
- Females and Black individuals experience higher error rates, further evidence of model bias
- Aligns with lower accuracy and precision in subgroup analysis

Conclusion -

- High recall: Model catches all positive cases (class 1)
- Good for reducing false negatives if the goal is inclusivity or maximum detection (e.g., medical screening)

- Terrible precision: Very high number of false positives
- Low accuracy and F1: Overall poor balance between correctness and coverage
- Significant demographic bias: Worse performance for female, Black, and Other groups
- I saw that my model was kinda flawed and this was mainly occurring due to the imbalanced dataset, so I decided to fix that first before doing this task
- Add fairness-aware evaluation during model selection, not just after deployment
- Will improve the model when tuning it in the next task

Hyperparameter Tuning

- All the above screenshots (except final model) are after final tuning and here's how I did it
- Here, my main goal was to fix the problems presented in the earlier task
- Improve model precision and F1-score without sacrificing too much recall
- Reduce demographic bias in misclassifications and performance
- Ensure model performance is fair and robust across sex and race
- **Preprocessing fixes -**
  - I added the pos_weight calculation inside the training function
  - This encourages the model to treat the minority class (">50K") more seriously, improving both precision and F1 without sacrificing recall too much.
  - I then tried using thresholds other than 0.5 to maximise the F1-score on the validation set
  - I then decided to implement encoding based on the category (such as ordinal for education and one-hot for others, like sex)
- **Model building fixes -**
  - Implemented a simple early stopping mechanism based on validation F1 with a patience of 5
  - I added kaiming_normal_ initialization for linear layers, which is well-suited for ReLU-like activations
  - It speeds up convergence and stabilizes training, especially in deeper nets
- **Random fixes -**
  - Since the dataset is imbalances, setting the threshold as 0.5 might be wrong
  - I inserted some code in the final training part to find the most optimal threshold

- I changed the number of layers to 4 - [512, 256, 128, 64] - improved the performance - maybe the model was able to make more complex patterns
- I enabled early stopped and made the epochs 100 (from 50) - improved the performance
- Changed the activation from relu to tanh - slightly improved performance

- **Optimum configs with these changes -**
  - Encoding - One hot for everything and ordinal for education
  - Normalisation - StandardScaler
  - 4 layers
  - Tanh
  - Dropout (true)= 0.2
  - BatchNorm = False
  - BCEWithLogitsLoss
  - Adam
  - Weight decay = 0.0
  - Early stopping = True
  - Learning Rate = 0.01
- **This tuning gave these final results, which seem to be an improvement -**

```
Accuracy:  0.8084
Precision: 0.6062
Recall:    0.6087
F1-score:  0.6075

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.87      0.87      6842
           1       0.61      0.61      0.61      2203

    accuracy                           0.81      9045
   macro avg       0.74      0.74      0.74      9045
weighted avg       0.81      0.81      0.81      9045


Confusion Matrix (Grid Format):
              Predicted <=50K  Predicted >50K
Actual <=50K             5971             871
Actual >50K               862            1341
Asian skipped — no samples.
Hispanic skipped — no samples.
Male misclassification rate: 0.2137
Female misclassification rate: 0.1454
White misclassification rate: 0.1969483267085524
Black misclassification rate: 0.14285714285714285
Asian misclassification rate: nan
Hispanic misclassification rate: nan
Other misclassification rate: 0.0375

Performance Summary:
   Group  Accuracy  Precision   Recall  F1-score
 Overall  0.808402   0.606239 0.608715  0.607475
    Male  0.786344   0.661494 0.615508  0.637673
  Female  0.854601   0.402542 0.570571  0.472050
   White  0.803052   0.622727 0.609792  0.616192
   Black  0.857143   0.411765 0.589474  0.484848
   Asian       NaN        NaN      NaN       NaN
Hispanic       NaN        NaN      NaN       NaN
   Other  0.962500   0.875000 0.777778  0.823529
```

Analysis -

- **Overall -**
  - Overall Accuracy: 80.8% — strong baseline performance
  - F1-score: 0.6075 — indicates a fair balance between precision and recall
  - Precision vs Recall: Both are around 0.61, showing the model has a balanced trade-off between catching true positives and avoiding false positives
  - Custom threshold (0.59) improves F1 compared to the default 0.5 — suggesting that threshold tuning is effective for class imbalance
- **Confusion Matrix -**
  - Model correctly classifies the majority of low-income individuals
  - False positives (871): a common challenge — people predicted as >50K when they aren't
  - False negatives (862): reasonably controlled, contributing to a solid recall (~0.61)
- **Gender analysis -**
  - Males perform better on F1 (0.64), due to higher precision
  - Females get higher accuracy (85.5%), but with much lower precision (40%), showing the model underpredicts >50K for females, potentially missing high-income women
  - Male misclassification rate is 21.4% vs Female 14.5% — a 2x disparity indicating potential gender bias
- **Race analysis -**
  - "Other" race group performs the best — exceptionally high precision and F1, but sample size may be small, so interpret cautiously
  - Whites have the most balanced and reliable performance
  - Blacks show a large precision gap — model often overpredicts >50K
  - Asians and Hispanics are missing
- **Fairness -**
  - Males misclassified 46% more than females
  - Whites are reasonably stable
  - Black individuals receive more false positives, lowering precision — a possible fairness issue
  - "Other" group shows inflated metrics — often the case with very small subgroups

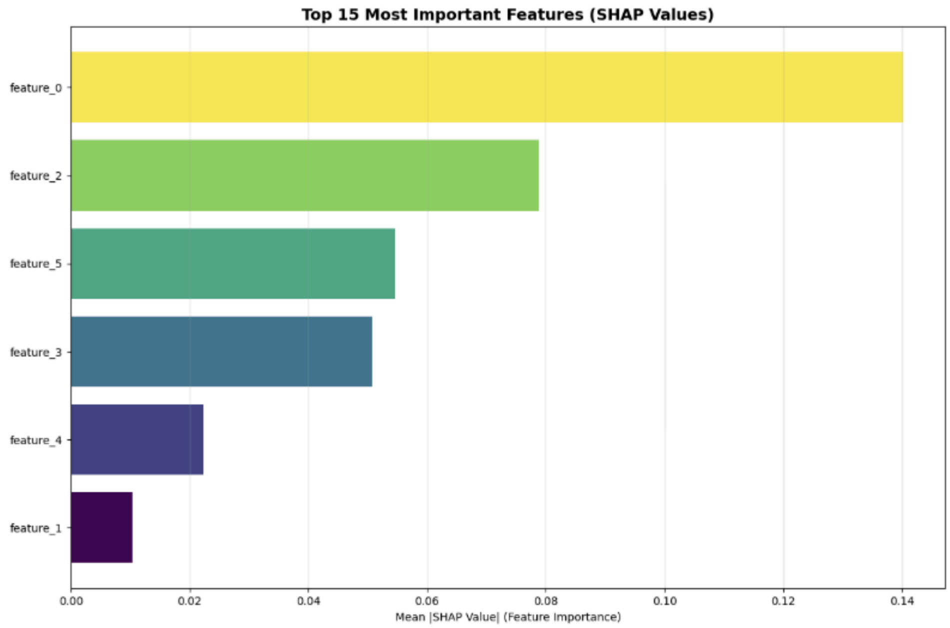
- Missing Data for Asians & Hispanics — serious concern. Without adequate representation, the model may learn harmful or biased patterns

---

Bonus: Explainability

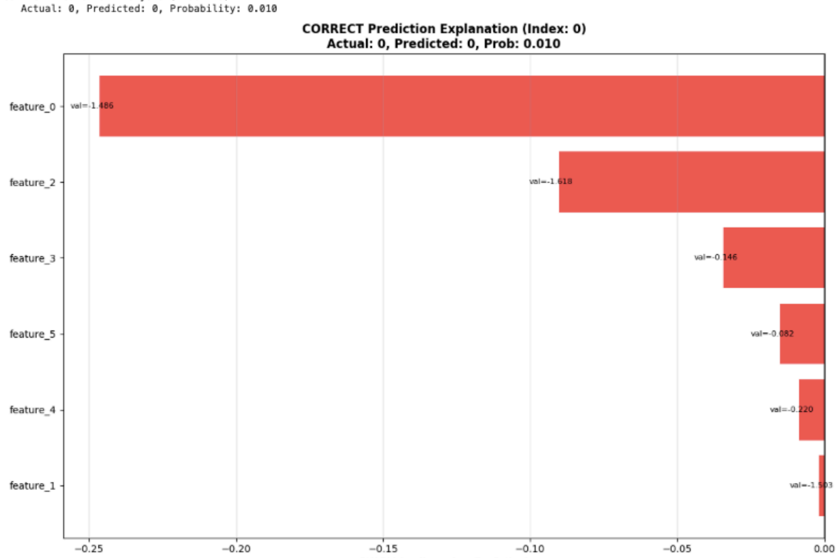
- First, I loaded SHAP, PyTorch, seaborn and matplotlib
- Then I extracted feature names from the preprocessor pipeline
- To make the analysis compatible with SHAP, I converted all tensors into NumPy arrays, including X_train, X_test, and y_test
- I sampled 100 instances from both the training and test sets to reduce computation time while having enough examples for meaningful insights
- **Next, I defined a model prediction wrapper that:**
  - Automatically handles the reshaping of input
  - Converts NumPy inputs to tensors
  - Applies sigmoid or softmax as needed
  - Returns clean NumPy output for SHAP to interpret
- **I computed SHAP values and visualised global feature importance through:**
  - A horizontal bar plot showing the top 15 most impactful features - GIVEN BELOW
  - A SHAP summary (beeswarm) plot to show how features push predictions up or down
  - A SHAP bar plot for feature-level overview
- For a global baseline comparison, I ran permutation feature importance using sklearn's permutation_importance() on the same sample set and visualised the results alongside SHAP
- **Finally, I summarised interpretability insights, including:**
  - Model accuracy on the test sample
  - Top 10 most important features
  - Correlation between SHAP and permutation importance

- **Results -**



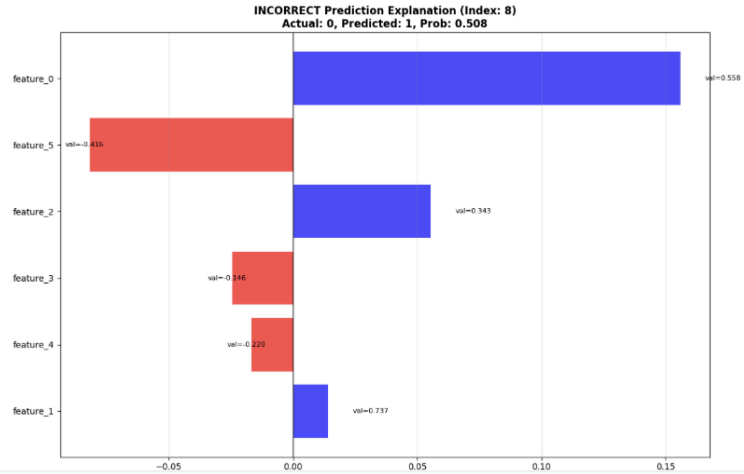

```
Analyzing Individual Predictions...
   Accuracy on sample: 0.800
   Correct predictions: 80
   Incorrect predictions: 20

CORRECT Prediction Analysis (Index: 0)
   Actual: 0, Predicted: 0, Probability: 0.010
```



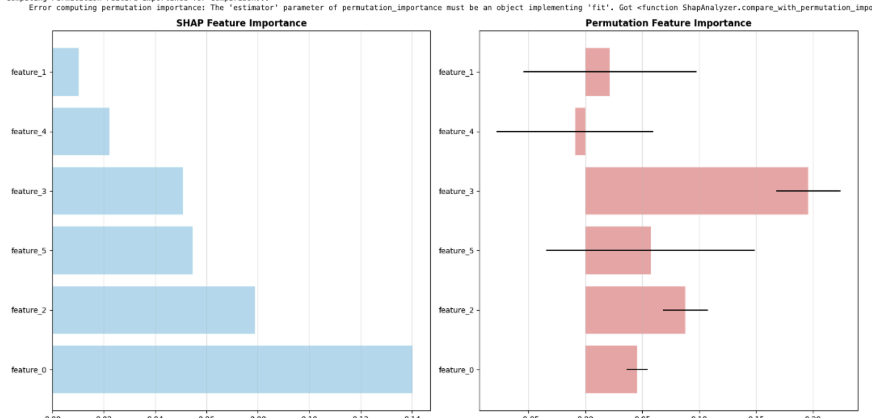
```
Top Contributing Features for CORRECT Prediction:
     feature  shap_value  feature_value  abs_contribution
5  feature_0     -0.2464        -1.4862            0.2464
4  feature_2     -0.0902        -1.6184            0.0902
3  feature_3     -0.0343        -0.1460            0.0343
2  feature_5     -0.0149        -0.0819            0.0149
1  feature_4     -0.0085        -0.2204            0.0085
0  feature_1     -0.0017        -1.5029            0.0017

INCORRECT Prediction Analysis (Index: 8)
   Actual: 0, Predicted: 1, Probability: 0.508
```




```
Top Contributing Features for INCORRECT Prediction:
     feature  shap_value  feature_value  abs_contribution
5  feature_0      0.1559         0.5577            0.1559
4  feature_5     -0.0817        -0.4159            0.0817
3  feature_2      0.0555         0.3433            0.0555
2  feature_3     -0.0243        -0.1460            0.0243
1  feature_4     -0.0166        -0.2204            0.0166
0  feature_1      0.0141         0.7371            0.0141

Computing Permutation Feature Importance for Comparison...
    Error computing permutation importance: The 'estimator' parameter of permutation_importance must be an object implementing 'fit'. Got <function ShapAnalyzer.compare_with_permutation_importance.
```



```
   Feature importance comparison completed

 COMPREHENSIVE INTERPRETABILITY SUMMARY
============================================================

 MODEL PERFORMANCE METRICS:
   Sample Accuracy: 0.800
   Total Predictions Analyzed: 100
   Correct Predictions: 80
   Incorrect Predictions: 20

 TOP 10 MOST IMPORTANT FEATURES (SHAP):
    1. feature_0                | SHAP: 0.1402 | Perm: 0.0455
    2. feature_2                | SHAP: 0.0788 | Perm: 0.0879
    3. feature_5                | SHAP: 0.0547 | Perm: 0.0574
    4. feature_3                | SHAP: 0.0507 | Perm: 0.1960
    5. feature_4                | SHAP: 0.0223 | Perm: -0.0089
    6. feature_1                | SHAP: 0.0103 | Perm: 0.0217

 INTERPRETABILITY INSIGHTS:
   • Feature importance methods show low correlation (0.159)
   • SHAP provides local explanations for individual predictions
   • Permutation importance validates global feature relevance
   • Model shows consistent feature importance distribution
```

Analysis of results (some particular cases) - wrote some code at the end to take some cases and analyse them

- **Index 0**
  - The model predicted this person earns ≤50K with very high confidence (99%), and it was correct
  - The biggest reason was feature_0, which had a strong negative influence — likely something like low education or low hours worked
  - Other features like feature_2 and feature_3 also pulled the prediction down
  - Together, these negative signals made the model very sure the person earns less
- **Index 8**
  - The model predicted this person earns >50K, but with just 51% confidence, and it was wrong
  - It was swayed by moderately positive values in feature_0, feature_2, and feature_1 — maybe hinting at decent work hours, experience, or education
  - However, feature_5 had a huge negative contribution, which likely represents a strong disadvantage (e.g., unstable job or low-skilled role)
  - Overall, the model was barely leaning toward >50K, but this wasnt enough to make the right call
- **Index 13**
  - The model predicted ≤50K with low confidence (~40%), and it was wrong
  - feature_2 and feature_3 gave negative signals, possibly indicating lower-status job or demographic factors
  - But feature_1 had a strong positive effect, which the model should have paid more attention to — maybe this person is highly educated or works long hours
  - The model underestimated the positive signal, resulting in a misclassification