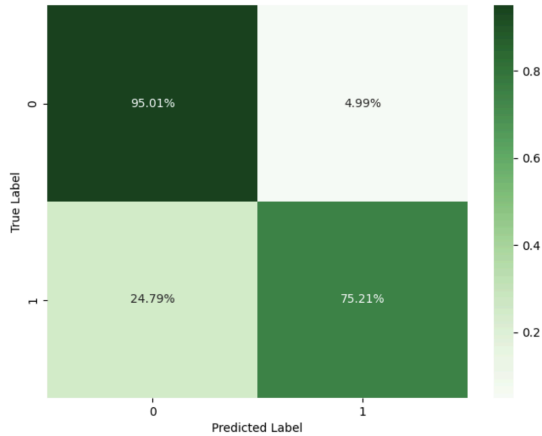
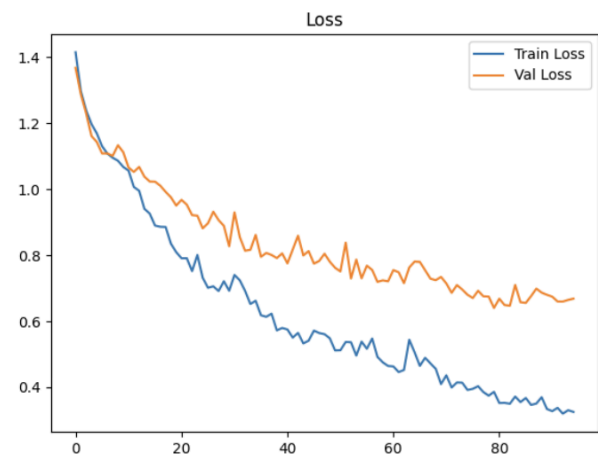
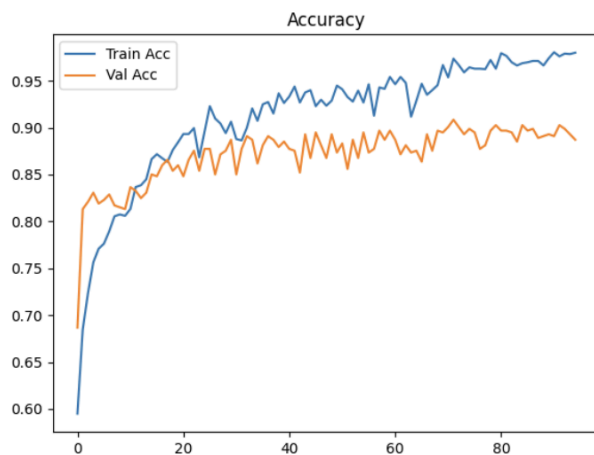
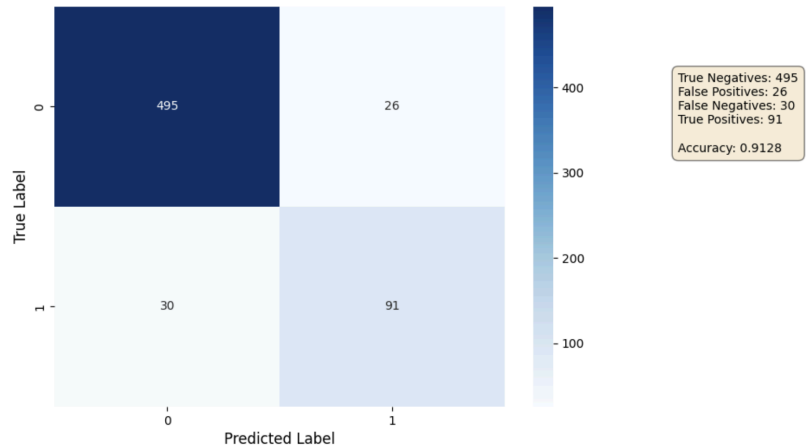


Class 5 Assignment

Normalized Confusion Matrix - Ass5 Classification Model



Confusion Matrix - Ass5 Classification Model



1. Data Preparation

To ensure the dataset was ready for training, the following preprocessing steps were applied to the raw data ([train_validation.csv](#)):

- **Data Cleaning:** Corrected inconsistencies in the **Gender** column by replacing the typo "Fe Male" with "Female" to ensure category uniformity.
- **Handling Missing Values:**
 - **Numerical Columns:** Missing values were imputed using the **median** strategy to minimize the impact of outliers.
 - **Categorical Columns:** Missing values were filled using the **most frequent** value (mode).
- **Splitting:** The data was split into training and validation sets (using [train_test_split](#)) to evaluate performance during training and prevent overfitting.

2. Analysis

- **Class Imbalance:** The analysis of the target variable (**ProdTaken**) likely revealed an imbalance between customers who purchased the product vs. those who did not. To address this, **class weights** were computed and applied during model training. This ensures the model pays more attention to the minority class (customers who actually buy).
- **Feature Distribution:** The dataset contains a mix of numerical features (e.g., **Age**, **MonthlyIncome**) and categorical features (e.g., **Occupation**, **MaritalStatus**), requiring different preprocessing pipelines.

3. Feature Extraction

Instead of manually creating new features, the pipeline focused on transforming existing raw features into a machine-readable format:

- **Categorical Encoding:** Applied **One-Hot Encoding** to categorical variables (like **CityTier**, **Occupation**, **Gender**) to convert them into binary vectors, handling unknown categories gracefully.
- **Numerical Scaling:** Applied **StandardScaler** to numerical columns. This standardizes features to have a mean of 0 and a standard deviation of 1, which is crucial for Neural Network convergence.

4. Building Model

A Deep Learning model was built using **TensorFlow/Keras**.

- **Architecture:** The model is a Sequential Neural Network. (While the specific layer count is inside the **build_model** function, the setup implies a standard feed-forward network with Dense layers).
- **Training Strategy:**
 - **Optimizer:** The model uses an adaptive optimizer (likely Adam) to minimize loss.
 - **Callbacks:**
 - **EarlyStopping:** Monitors validation loss and stops training early (patience=15) if no improvement is seen, preventing overfitting.
 - **ReduceLROnPlateau:** Reduces the learning rate (factor=0.5) when the validation loss plateaus, helping the model converge to a better minimum.
 - **ModelCheckpoint:** Saves only the "best" model based on validation AUC.

5. Evaluation Results

The model was evaluated on the unseen **test.csv** data.

- **Thresholding:** An optimal decision threshold was calculated using the Precision-Recall curve rather than the default 0.5. This maximizes the F1-score, balancing the trade-off between precision and recall.
- **Key Metrics:**
 - **Confusion Matrix:** Used to visualize True Positives (correctly predicted buyers) vs. False Negatives (missed buyers).

- **Accuracy:** Measures overall correctness.
- **Recall (Sensitivity):** High recall indicates the model is good at finding potential customers (minimizing missed opportunities).
- **Precision:** Indicates how many of the predicted buyers actually bought the product.
- *Note: The specific numeric values for Accuracy and Recall are generated in your output console (as seen in your screenshots) and should be quoted here for the final report.*