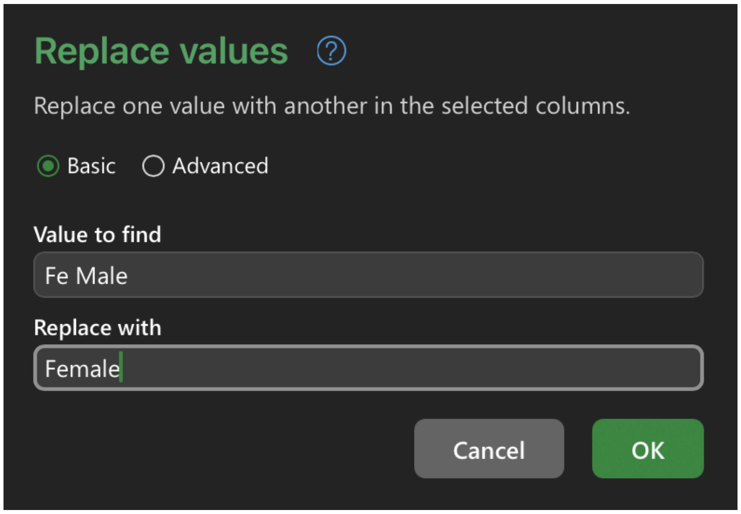
Open PowerQuery
Step 1: Replace Value Fe Male to Female



Step 2: Assign Value to ProductPitched Tier (ordinal)

```
if[ProductPitched]="Basic" then 1
else if [ProductPitched] = "Standard" then 2
else if [ProductPitched] = "Deluxe" then 3
else if [ProductPitched] = "Super Deluxe" then 4
else 5
```

Step 3: Cap Pitch Duration and replace column
if [DurationOfPitch] > 100 then 100 else [DurationOfPitch]

Step4: Num of Adults Visiting
[NumberOfPersonVisiting] - [NumberOfChildrenVisiting]
Step5: Income Per Person
[MonthlyIncome] / [NumberOfPersonVisiting]
Step 6: Pitch Satisfaction Score
=[PitchSatisfactionScore]/[[PitchDuration_Cap]]
Step7:
One-Hot Encoding: Converted text columns like `Occupation`, `Gender`, `MaritalStatus`, `Designation`, and `TypeofContact` into binary columns (e.g., `Gender_Male`: 1 or 0).

Using "1, 2, 3" for categories without a natural rank (like Marital Status) misleads the model because it treats the numbers as having mathematical value, not just as labels. If you code Married=1, Single=2, and Divorced=3, the model assumes "Divorced" is numerically greater than "Married" and might even calculate that the "average" of a Married and Divorced person is a Single person (since $(1+3)/2=2$). This false hierarchy forces the model to find patterns based on non-existent logic, whereas splitting them into separate True/False columns (One-Hot Encoding) treats each category as distinct and equal, allowing the AI to learn the actual impact of each specific status independently.

structure and logic of the `assignment5training.py` script. The goal of this script is to train a Deep Learning model (using TensorFlow/Keras) to predict whether a customer will purchase a tourism package (`ProdTaken`).

Part 0: The Helper Function (`split_csv_data`)

What it does

This is a custom function designed to physically separate the raw data into two distinct CSV files before any processing begins.

- **Inputs:** It takes a source file, output filenames, and a `fraction` (set to 0.10 or 10%).
- **Logic:**
    1. It loads the full dataset.
    2. It uses `df.sample(frac=...)` to randomly pick 10% of the rows.
    3. It uses `df.drop(...)` to create a dataset of the remaining 90%.
    4. It saves both as new CSV files.
- **Why is this important?** In Machine Learning, **Data Leakage** is a major problem. By saving the "Test Set" to a completely separate file (`class5_test_data.csv`) at the very beginning, we guarantee that the model *never* sees this data during training. It serves as the "Final Exam."

Part 1: Execution of the Split

split_csv_data(..., fraction=0.10, random_state=42)

- **`random_state=42`**: This ensures that every time you run the code, it picks the *exact same* random rows. This makes your results reproducible.
- **Outcome**: You now have `class5_test_data.csv` (10% unseen) and `class5_remaining_data.csv` (90% for training).

Part 2: Loading Training Data

df = pd.read_csv(remaining_file_name)

- **Crucial Step**: The script intentionally loads the **remaining** 90% file, not the original source. This ensures we are only training on the data meant for learning.

Part 3: Preprocessing

Dropping Columns

- `Index`: Identifier columns do not contain patterns helping prediction, so they are removed.
- `ProductPitched`: This is dropped to prevent **Data Leakage**. If we know which product was pitched, it might be too highly correlated with the result, preventing the model from learning *customer* characteristics.

Defining X and y

- `y` **(Target)**: `ProdTaken` (Did they buy? 0 = No, 1 = Yes).
- `X` **(Features)**: All other columns.
- `.astype(float)`: Neural networks require mathematical inputs. This converts any boolean (True/False) data into 1.0 and 0.0.

Part 4: Train / Validation Split

X_train, X_test, y_train, y_test = train_test_split(..., test_size=0.25, stratify=y)

Even though we already set aside a "Final Exam" file (Part 1), we need an internal "Quiz" to evaluate the model while we are building it.

- `test_size=0.25`: 25% of the *remaining* data is set aside for internal validation.
- `stratify=y`: This is vital for imbalanced data. It ensures that if 18% of people bought the product in the total set, exactly 18% of people in the train set and 18% in the test set also bought it. It keeps the proportions balanced.

Part 5: Scaling

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

Neural Networks are very sensitive to the size of numbers.

- **StandardScaler**: Transforms data so the mean is 0 and standard deviation is 1.
- **Why?**: Without this, a column like "Income" (values like 50,000) would overpower a column like "Age" (values like 30).
- `joblib.dump`: Saves the math used for scaling so we can apply the *exact same math* to new data later.

Part 6: Class Weights (Currently Commented Out)

If active, `compute_class_weight` calculates how rare the "Buy" (1) event is. It assigns a higher penalty if the model makes a mistake on a "Buy" prediction, forcing the AI to pay more attention to buyers.

Part 7: The Model Architecture

This is a **Deep Neural Network (DNN)** using the Keras Sequential API.

The Layers

The structure is a "Funnel" shape, getting smaller as it goes deeper:

1. **Dense(512)**: 512 neurons. A wide layer to capture complex patterns.
2. **Dense(256) -> Dense(128) -> ... -> Dense(16)**: Gradually compressing the information to find the most essential features.

Key Components

- **`activation='relu'`**: The mathematical function that allows the network to learn non-linear patterns (decisions that aren't just straight lines).
- **`Dropout(0.4)` / `Dropout(0.2)`**:
  - This is a regularization technique.
  - `0.4` means "Randomly turn off 40% of the neurons in this layer during every training step."
  - **Why?** It prevents the model from memorizing the data (Overfitting) and forces it to learn robust general rules.
- **Output Layer (`Dense(1, activation='sigmoid')`)**:
  - 1 Neuron: Because we want one answer (Probability of buying).
  - `sigmoid`: Smashes the output between 0 and 1. (e.g., 0.85 means 85% chance of buying).

Part 8: Training with Callbacks

model.fit(..., epochs=150, validation_split=0.2, ...)

- **`epochs=150`**: The model sees the data 150 times.
- **`validation_split=0.2`**: During training, it holds back another 20% of the training data to check its performance at the end of every epoch.

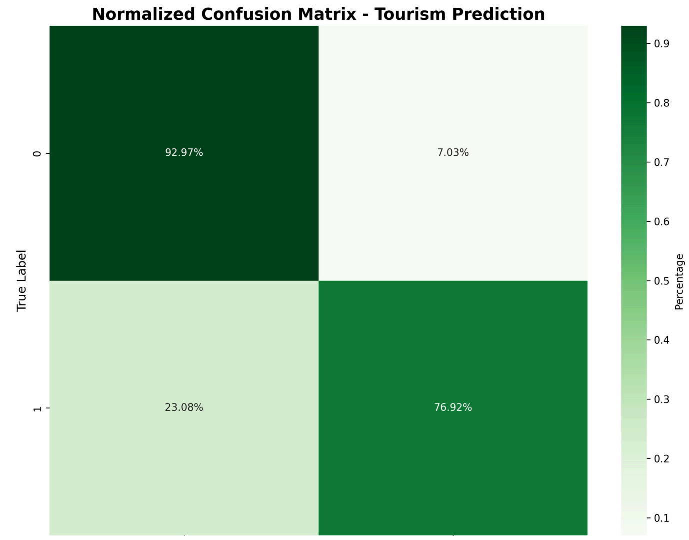
Callbacks (The "Smart" Controls)

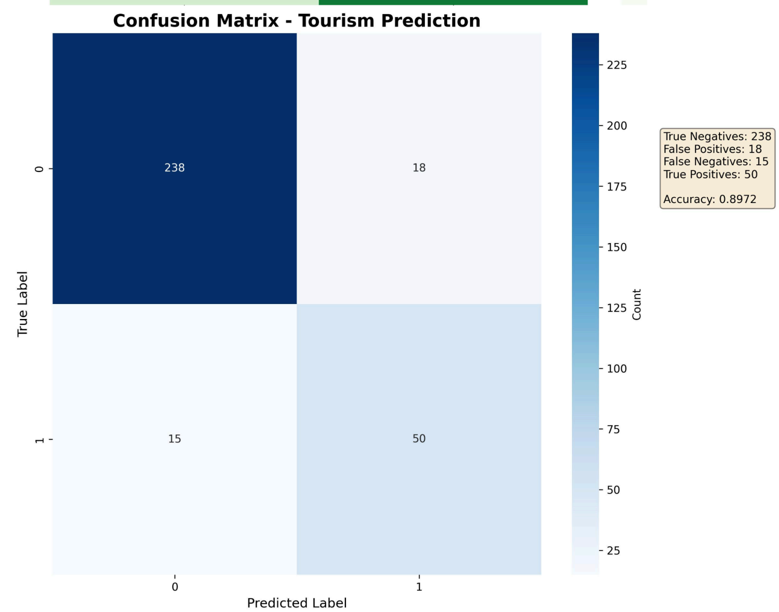
1. **`EarlyStopping`**:

    - Monitors `accuracy`.
    - If the accuracy doesn't improve for 10 epochs (`patience=10`), it stops training immediately.
    - `restore_best_weights=True`: It reverts the model back to its "best" state, ignoring the last 10 bad epochs.
2. **`ReduceLROnPlateau`**:
    - If the model gets stuck based on what it monitors, it cuts the `learning_rate` in half (`factor=0.5`).
    - This allows the model to make smaller, more precise adjustments to try and find a better solution.

Part 9 & 10: Evaluation and Saving

- **`model.evaluate`**: Checks the final accuracy on the `X_test` data (the 25% internal split).
- **`classification_report`**: Shows Precision (Accuracy of positive predictions) and Recall (How many of the actual positives were found).
- **`model.save`**: Saves the trained brain of the AI to a `.keras` file so it can be loaded later without retraining.





==
INFERENCE RESULTS
==

Accuracy: 0.8972
AUC Score: 0.9053

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.93	0.94	256
1	0.74	0.77	0.75	65
accuracy			0.90	321

macro avg	0.84	0.85	0.84	321
weighted avg	0.90	0.90	0.90	321

Confusion Matrix:

```
[[238  18]
 [ 15  50]]
```