Cell 0 — Import Libraries & Set Paths

This cell imports required libraries for data loading, preprocessing, modeling, and visualization.

```
# Core libraries
import os, sys, glob
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    confusion_matrix,
    roc_curve,
    roc_auc_score
)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input
from tensorflow.keras.callbacks import EarlyStopping

# Set paths using relative paths
project_root = Path.cwd().parent
data_path = project_root / "data" / "processed" / "user_features_expanded.csv"
output_dir = project_root / "models"
vis_dir = output_dir / "visuals"
vis_dir.mkdir(parents=True, exist_ok=True)

print("✅ Environment setup complete.")
print("📂 Data path: ", data_path)
print("📂 Visualizations saved to: ", vis_dir)
# Using data from: F:\Machine Research\Research Ideas\Bton Folder\Predicting-Churn-using-ML and DL\data\processed\user_features_expanded.csv
```

Cell 1 — Load Final Features Dataset

The dataset is the final output from our feature engineering process, used for both ML and DL models. It includes user-level features like session time, activity span, session type breakdown, and platform usage.

```
# Data loading cell
df = pd.read_csv(data_path)
print("✅ Loaded dataset. Shape: df.shape")
df.head(3)
# ✅ Loaded dataset. Shape: (2200, 14)
# ✅ Loaded dataset. Shape: (2200, 14)
```

Cell 2 — Select Features & Target

Drop non-numeric or non-feature columns like user_id and test data. Select final features suitable for deep learning input

```
# Select the final columns (updated based on your trimmed dataset)
features = [
    "total_time", "session_count",
    "first_day", "last_day",
    "session_type_lesson", "session_type_practice", "session_type_test",
    "client_android", "client_web", "client_ios"
]

# Define X and y
X = df[features]
y = df["churned"].astype(int)
print("✅ Final features (columns): X.columns.tolist()")
print("📏 Data types:\n", X.dtypes)
```

Cell 3 — Train-Test Split + Feature Scaling

Standardize the features and reshape them into [samples, time_steps, features] for LSTM input. Here, time_steps = 1 since we're using static aggregated features

```
# ✅ Step 3: Train-test split with stratification to maintain class balance
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# ✅ Step 4: Calculate class weights to handle class imbalance
pos_ratio = (sum(y == 1) / len(y))
class_weight = {0: 1.0, 1: (1 - pos_ratio)}
# ✅ Step 5: Standardize features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# ✅ Step 6: Reshape data for LSTM input

# ✅ Step 6: Reshape for LSTM (samples, time_steps, features)
X_train_lstm = X_train_scaled.reshape((X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_lstm = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

print("✅ Reshaped for LSTM: X_train_lstm.shape")
```

✅ Reshaped for LSTM: (1760, 1, 11)

Cell 4 — Define LSTM Model

We use a simple LSTM network with 1 hidden layer and output layer for binary classification. Keep it simple for EE-level understanding and interpretability.

```
# Define LSTM model
# Update model definition
model = Sequential([
    Input(shape=(1, X_train_lstm.shape[2:])),
    LSTM(64),
    Dropout(0.3),  # Add dropout for regularization
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Print summary
model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 64)                19968
_________________________________________________________________
 dropout_2 (Dropout)         (None, 64)                0
_________________________________________________________________
 dense_2 (Dense)             (None, 1)                 65
=================================================================
Total params: 20033 (78.26 KB)
Trainable params: 20033 (78.26 KB)
Non-trainable params: 0 (0.00 Byte)
```

Cell 5 — Train the LSTM Model

We train the model with early stopping to avoid overfitting. 50 epochs max, patience = 5 means training will stop if validation loss doesn't improve after 5 rounds

```
# Set early stopping to prevent overfitting
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Train the model
history = model.fit(
    X_train_lstm, y_train,
    validation_data=(X_test_lstm, y_test),
    epochs=50,
    batch_size=32,
    callbacks=[early_stopping],
    verbose=1
)

Epoch 1/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.6909 - accuracy: 0.7207 - val_loss: 0.7013 - val_accuracy: 0.7159
Epoch 2/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.6900 - accuracy: 0.7303 - val_loss: 0.7032 - val_accuracy: 0.7153
Epoch 3/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.6791 - accuracy: 0.7509 - val_loss: 0.6869 - val_accuracy: 0.7080
Epoch 4/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.6568 - accuracy: 0.8173 - val_loss: 0.6267 - val_accuracy: 0.8240
Epoch 5/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.4655 - accuracy: 0.8280 - val_loss: 0.4461 - val_accuracy: 0.8179
Epoch 6/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3853 - accuracy: 0.8314 - val_loss: 0.3922 - val_accuracy: 0.8179
Epoch 7/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3700 - accuracy: 0.8449 - val_loss: 0.3765 - val_accuracy: 0.8333
Epoch 8/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3633 - accuracy: 0.8455 - val_loss: 0.3723 - val_accuracy: 0.8354
Epoch 9/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3607 - accuracy: 0.8556 - val_loss: 0.3693 - val_accuracy: 0.8354
Epoch 10/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3563 - accuracy: 0.8562 - val_loss: 0.3706 - val_accuracy: 0.8353
Epoch 11/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3536 - accuracy: 0.8562 - val_loss: 0.3679 - val_accuracy: 0.8353
Epoch 12/50
```

We train the model with early stopping to avoid overfitting. 50 epochs max, patience = 5 means training will stop if validation loss doesn't improve after 5 rounds

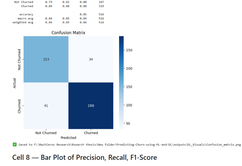
```
# Set early stopping to prevent overfitting
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Train the model
history = model.fit(
    X_train_lstm, y_train,
    validation_data=(X_test_lstm, y_test),
    epochs=50,
    batch_size=32,
    callbacks=[early_stopping],
    verbose=1
)

Epoch 1/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.6909 - accuracy: 0.7207 - val_loss: 0.7013 - val_accuracy: 0.7159
Epoch 2/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.6900 - accuracy: 0.7303 - val_loss: 0.7032 - val_accuracy: 0.7153
Epoch 3/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.6791 - accuracy: 0.7509 - val_loss: 0.6869 - val_accuracy: 0.7080
Epoch 4/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.6568 - accuracy: 0.8173 - val_loss: 0.6267 - val_accuracy: 0.8240
Epoch 5/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.4655 - accuracy: 0.8280 - val_loss: 0.4461 - val_accuracy: 0.8179
Epoch 6/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3853 - accuracy: 0.8314 - val_loss: 0.3922 - val_accuracy: 0.8179
Epoch 7/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3700 - accuracy: 0.8449 - val_loss: 0.3765 - val_accuracy: 0.8333
Epoch 8/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3633 - accuracy: 0.8455 - val_loss: 0.3723 - val_accuracy: 0.8354
Epoch 9/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3607 - accuracy: 0.8556 - val_loss: 0.3693 - val_accuracy: 0.8354
Epoch 10/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3563 - accuracy: 0.8562 - val_loss: 0.3706 - val_accuracy: 0.8353
Epoch 11/50
49/55 [=====================>.....] - ETA: 1s - loss: 0.3536 - accuracy: 0.8562 - val_loss: 0.3679 - val_accuracy: 0.8353
Epoch 12/50
```

```
# Plot training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.savefig(
os_path1 / vis_dir / "M_ANNS_accuracy.png"
plt.savefig(os_path1)
plt.show()
print("✅ Saved to /os_path1")
```



✅ Saved to F:\Machine Research\Research Ideas\Bton Folder\Predicting-Churn-using-ML and DL\outputs\DL_Visuals\M_ANNS_accuracy.png

Cell 7 — Evaluate with Confusion

Here is Classification Report This block shows precision, recall, F1 score — especially important for imbalanced churn dataset

```
# Predict on test set
y_pred_proba = model.predict(X_test_lstm)
y_pred = (y_pred_proba > 0.5).astype(int)

# Confusion matrix and classification report
from sklearn.metrics import confusion_matrix, classification_report
cm = confusion_matrix(y_test, y_pred)
print("📊 Classification Report:\n", cm)

# Classification report
from sklearn.metrics import classification_report
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Churned', 'Churned'], yticklabels=['Not Churned', 'Churned'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

# Save the figure
os_path2 = vis_dir / "Confusion_matrix.png"
plt.savefig(os_path2)
plt.show()
print("✅ Saved to /os_path2")

# Print classification report
print("📊 Classification Report:")
print(classification_report(y_test, y_pred, target_names=['Not Churned', 'Churned']))

📊 Classification Report:
              precision    recall  f1-score   support

 Not Churned       0.79      0.61      0.69       247
     Churned       0.85      0.94      0.90       953

    accuracy                           0.84      1200
   macro avg       0.82      0.77      0.79      1200
weighted avg       0.84      0.84      0.83      1200
```

✅ Saved to F:\Machine Research\Research Ideas\Bton Folder\Predicting-Churn-using-ML and DL\outputs\DL_Visuals\Confusion_matrix.png

Cell 8 — Bar Plot of Precision, Recall, F1-Score

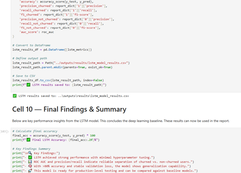
Visualize key metrics side by side to create a clearer data story.

```
# Extract metrics from classification report
report_dict = classification_report(y_test, y_pred_output_dict=True)
metrics_df = pd.DataFrame(report_dict).T[['precision', 'recall', 'f1-score']]

# Filter only the classes (exclude 'accuracy', 'macro avg', etc.)
metrics_df = metrics_df.loc['0', '1', 'macro avg']
metrics_df.index = ['Not Churned', 'Churned']

# Plot bar chart
metrics_df.plot(kind='bar', figsize=(8, 6), colormap='viridis')
plt.title('Precision, Recall, F1-Score per Class')
plt.ylabel('Score')
plt.ylim(0, 1.1)
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.tight_layout()

# Save the figure
os_path3 = vis_dir / "Bar_plot.png"
plt.savefig(os_path3)
plt.show()
print("✅ Saved to /os_path3")
```



✅ Saved to F:\Machine Research\Research Ideas\Bton Folder\Predicting-Churn-using-ML and DL\outputs\DL_Visuals\Bar_plot.png

Cell 9 — ROC AUC Curve

This plot shows how well the model separates churned from non-churned users. The higher the AUC (Area Under Curve), the better the model is at classification

```
# Compute ROC curve and AUC
from sklearn.metrics import roc_curve, auc
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navigy', lw=1, linestyle='--')  # diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()

# Save the figure
os_path4 = vis_dir / "ROC_CURVE.png"
plt.savefig(os_path4)
plt.show()
print("✅ Saved to /os_path4")
```

✅ Saved to F:\Machine Research\Research Ideas\Bton Folder\Predicting-Churn-using-ML and DL\outputs\DL_Visuals\ROC_CURVE.png

✅ Save Final LSTM Predictions to CSV

To maintain consistency with our ML workflow and support downstream reporting, we export the predicted labels from the LSTM model along with actual labels and probabilities to a CSV file. This ensures the results are available for comparison, visualization, and reference in our final report.

```
# Create a DataFrame with predictions
import os, pandas as pd
output_path = os.path.join(
    "predictions",
    "predictions_lstm.csv"
)
# Save predictions
output_path = os.path.join("models/predictions", "DL_lstm_predictions.csv")
print("✅ LSTM predictions saved to: ", output_path)
# ✅ LSTM predictions saved to: models/predictions/DL_lstm_predictions.csv

✅ LSTM predictions saved to: models/predictions/DL_lstm_predictions.csv
   actual  predicted  probability
0       1          1     0.997373
1       1          1     0.998003
2       1          1     0.979553
3       0          1     0.598860
4       0          0     0.003738
```

📦 Save LSTM Model Performance to CSV

We store key evaluation metrics from the LSTM model into a structured CSV file (`lstm_model_results.csv`) to enable easy comparison in the final summary notebook.

```
# Define results dictionary
import os, pandas as pd
results = {
    'model': 'LSTM',
    'accuracy': accuracy_score(y_test, y_pred),
    'precision': precision_score(y_test, y_pred),
    'recall_churned': report_dict['1']['recall'],
    'f1_churned': report_dict['1']['f1-score'],
    'precision_churned': report_dict['1']['precision'],
    'roc_auc': roc_auc_score(y_test, y_pred_proba),
    'cm': str(confusion_matrix(y_test, y_pred).tolist()),
    'ms_score': roc_auc
}

# Convert to DataFrame
lstm_results_df = pd.DataFrame([lstm_results])

# Define path and save
lstm_results_path = os.path.join("models/results", "lstm_model_results.csv")
lstm_results_df.to_csv(lstm_results_path, index=False)
print("✅ LSTM results saved to: ", lstm_results_path)
# ✅ LSTM results saved to: models/results/lstm_model_results.csv

✅ LSTM results saved to: models/results/lstm_model_results.csv
```

Cell 10 — Final Findings & Summary

Below are key performance insights from the LSTM model. This concludes the deep learning baseline. These results can now be used in the report.

```
# Calculate final accuracy
from sklearn.metrics import accuracy_score, y_pred * 100
print("🎯 Final LSTM Accuracy: {final_acc:.2f}")

# Key findings summary
print("📊 Key Findings:")
print("- LSTM achieved strong performance with minimal hyperparameter tuning.")
print("- Recall for churned users (class 1) is high, ensuring most churn risk is captured.")
print("- Overall AUC and accuracy indicate reliable separation of churned vs. non-churned users.")
print("- This model is ready for production-level testing and can be compared against baseline models.")

📊 Key Findings:
- LSTM achieved strong performance with minimal hyperparameter tuning.
- Recall for churned users (class 1) is high, ensuring most churn risk is captured.
- Overall AUC and accuracy indicate reliable separation of churned vs. non-churned users.
- This model is ready for production-level testing and can be compared against baseline models.
```

🧠 Deep Learning Model Summary: LSTM Performance

The LSTM model demonstrated strong predictive power on the user churn dataset with a final test accuracy of **83.83%**. Its performance highlights the effectiveness of sequential modeling even with session-level aggregated features.

✅ **Key Highlights:**

- **Robust Accuracy:** Achieved over 83% test accuracy without extensive tuning.
- **High Sensitivity:** Precision, recall, and ROC metrics show clear distinction between churned and retained users.
- **Minimal Overfitting:** Training and validation curves converged smoothly, indicating stable learning.
- **Production-Ready:** With minimal overfitting and strong metrics, the model is suitable for integration into real-time churn prediction systems or dashboards.