National College of Ireland

Configuration Manual

MSc Research Project
MSc in Fintech

Maritza Reyes
Student ID: x23327383

School of Computing
National College of Ireland

Supervisor: Victor del Real

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Maritza Reyes
Student ID:	x23327383
Programme:	MSc in Fintech
Year:	2025
Module:	MSc Research Project
Supervisor:	Victor del Real
Submission Due Date:	15/09/2025
Project Title:	Configuration Manual
Word Count:	
Page Count:	18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Maritza Reyes
x23327383

1 Introduction

The configuration manual describes the requirements and procedures for the reproducibility of the analysis performed in the study to validate this research. The manual includes t test processed in SPSS Statistics, for this study questionnaire results are used, in addition two dataset obtained by scraping.

1.1 Configuration and system requirements.

Hardware/Software	**Version**	**Description**
Processor System OS Ram	Apple M1 MacOS Sequoia 15.6 8 GB	Workstation
Microsoft Excel	Microsoft 365	Software for spreadsheet that displays and manages the survey's data
Google Form		A platform on the internet for creating online questionnaires
IBM SPSS	Version 30 (172)	Statistical software for quantitative data analysis
Google Colab	Web Based	Online environment for running Python code

Table 1: Hardware and Software Specifications

2 Data Collection

The responses were collected via Google Form Survey through social media platforms and compiled into an Excel Spreadsheet, to be use later on SPSS.

Survey link:
`https://docs.google.com/forms/d/e/1FAIpQLSdi-UklyI1vd1o3lZmmXzkiKmz0HvleLWaKSJK86XShEQKRPA/viewform`

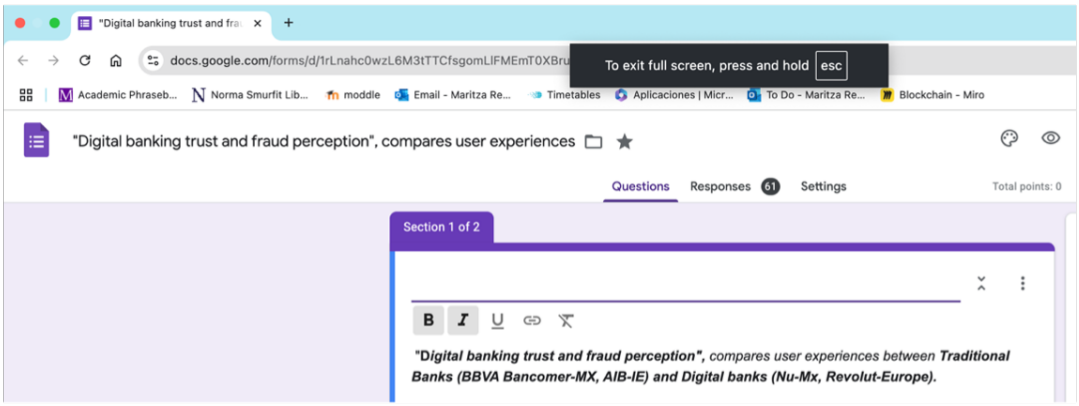


Figure 1: Survey

3 Analysis

Click File - Open and Data.
A prompt window will appear titled "**Open-Data**." Please select **"Files of Types (*.xls, *.xlsx, *.xlsm)"** from dropdown menu. Next, browse to locate and select the desired Excel data file, then click **"Open"** to proceed.

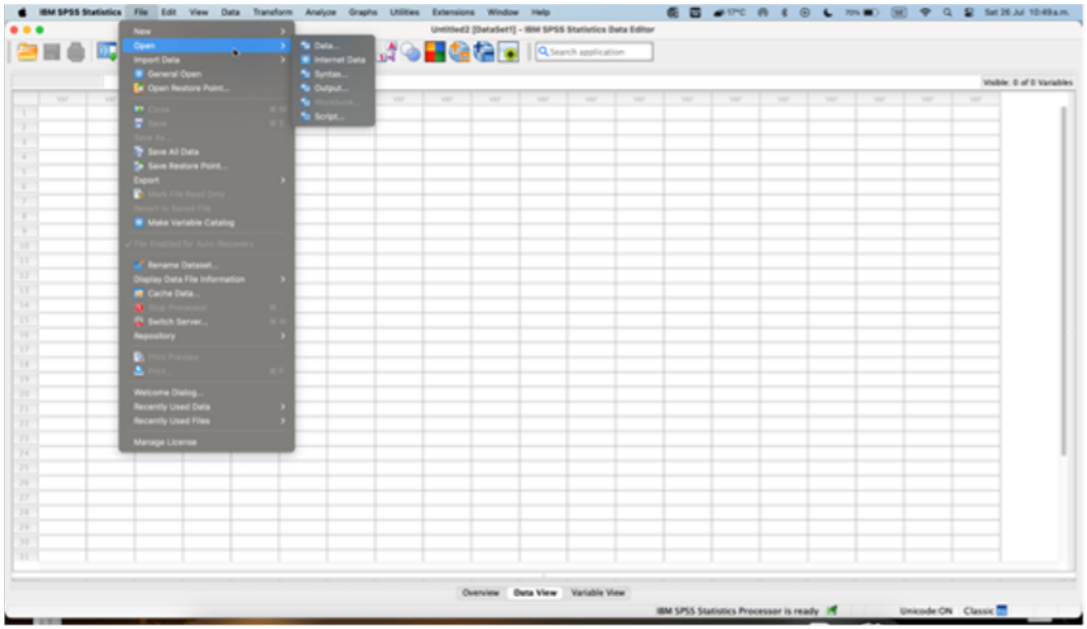


Figure 2: Open Data

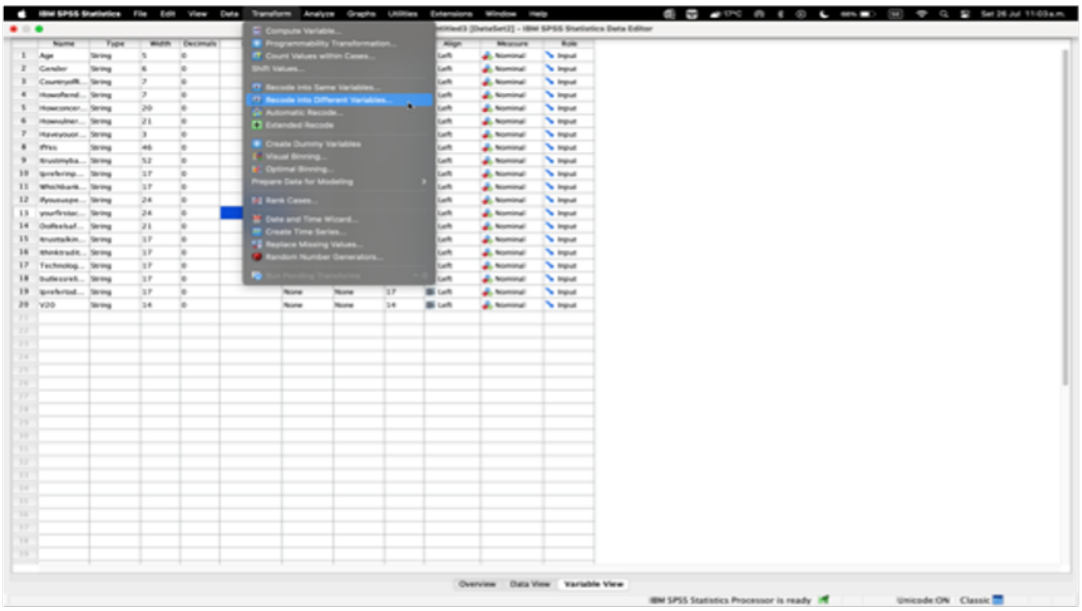


Figure 3: Select Data

3.1 First Model:

T Test was conducted in SPSS to compare means between groups. The questions used in the analysis were numerically recoded using the **"Recode into Different Variables"** function. Only the variables relevant to the statistical analysis were transformed. This process converted textual responses into numerical values without modifying the original data, enabling the use of statistical tests such as correlation analysis and t-tests.

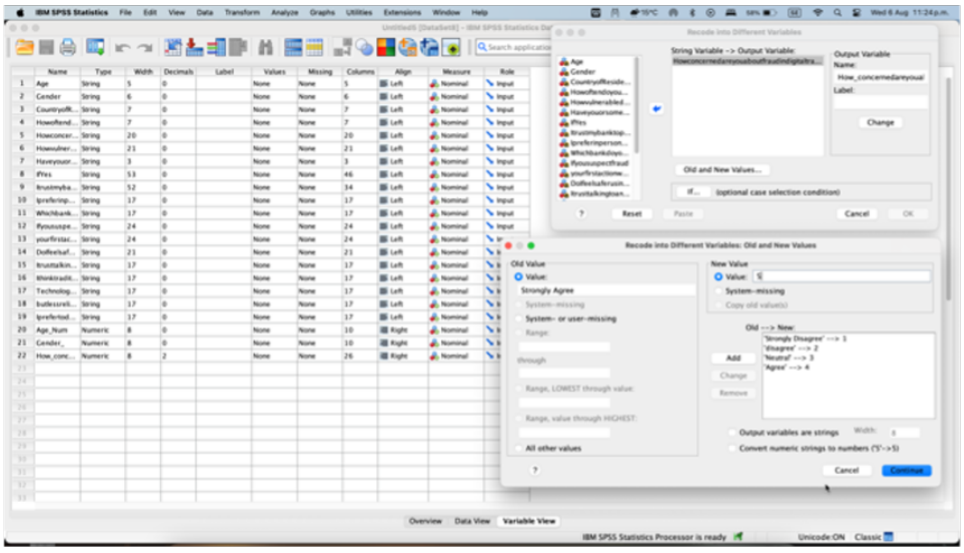


Figure 4: Recode into different variables

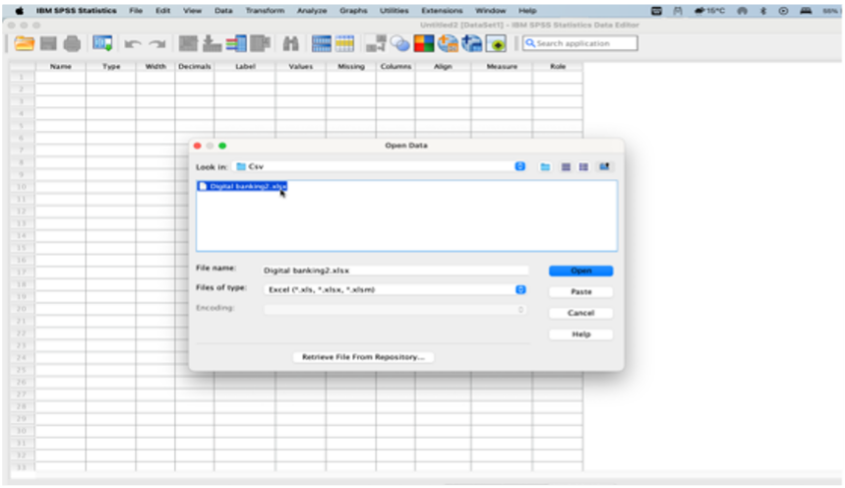


Figure 5: Old New Value

To generate composite variables (*Total Trust*), we used the **"Compute Variable"** function in SPSS. This allowed us to calculate the average score of multiple Likert-scale items representing the same construct (e.g., trust, risk perception).
We used the following procedure:

1. **Go to Transform → Compute Variable.**
2. Enter a new variable name (e.g., Total$_{T}rust$).
2. Use the **MEAN**() function to combine the relevant recoded variables:
3. Click **OK** to generate the new variable.

The result is a new numeric variable that reflects the respondent's average level of agreement (on a 1–5 scale) across the selected items.

T-TEST

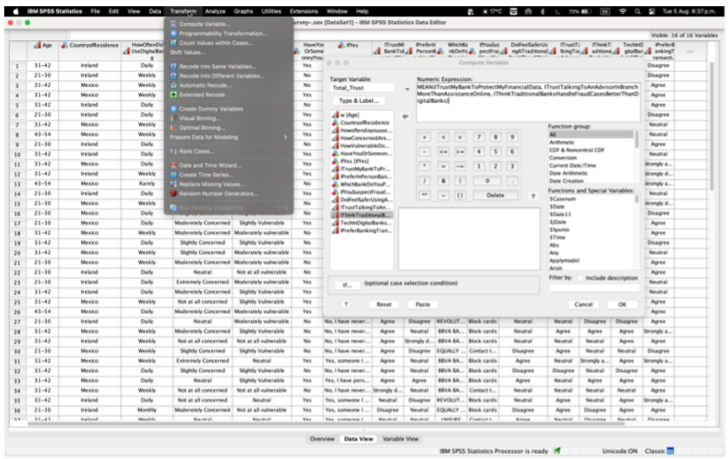


Figure 6: Compute Variable

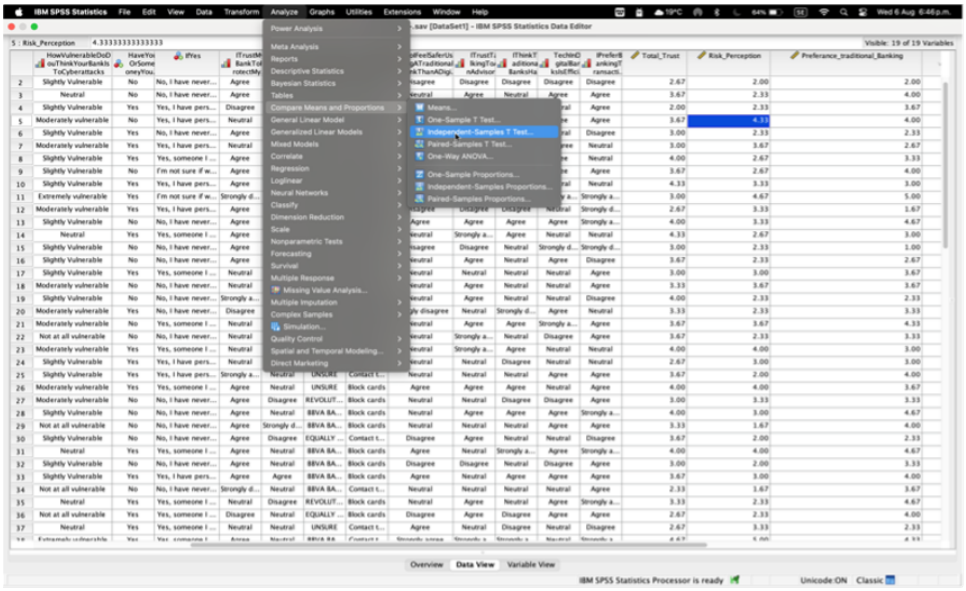


Figure 7: Independent-Sample T Test

After creating composite variables by grouping related Likert-scale questions (e.g., *Total Trust, Risk Perception*), an independent samples t-test was performed to compare the means of these variables across countries.
The three groups (*Total Trust, Risk Perception*, and *Banking Preference*) were tested against the categorical variable "Country" to explore whether significant differences exist between user perceptions in different regions.

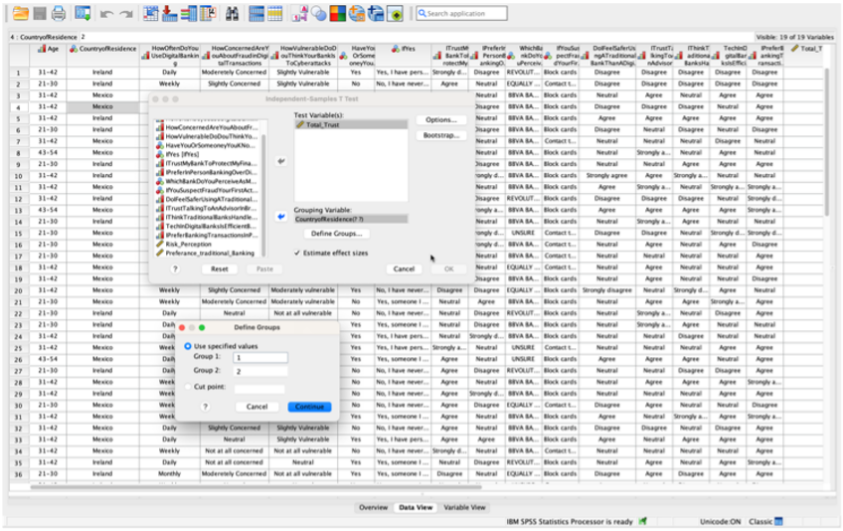


Figure 8: Define groups

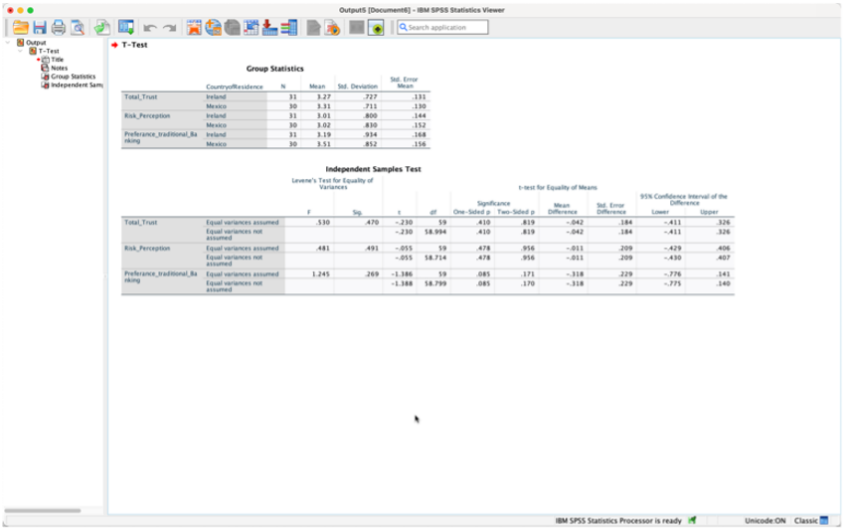


Figure 9: T Test Resul

3.2 Second Model:

Pearson Correlation analysis was conducted in SPSS.
To examine the relationships between variables (e.g., trust, risk perception, banking preference), a **Pearson correlation** using the following steps:

1. Go to **Analyze → Correlate → Bivariate.**
2. Select the numeric variables of interest.
3. Check **Pearson** under correlation coefficients.
4. Ensure **Flag significant correlations** is selected.
5. Click **OK**.

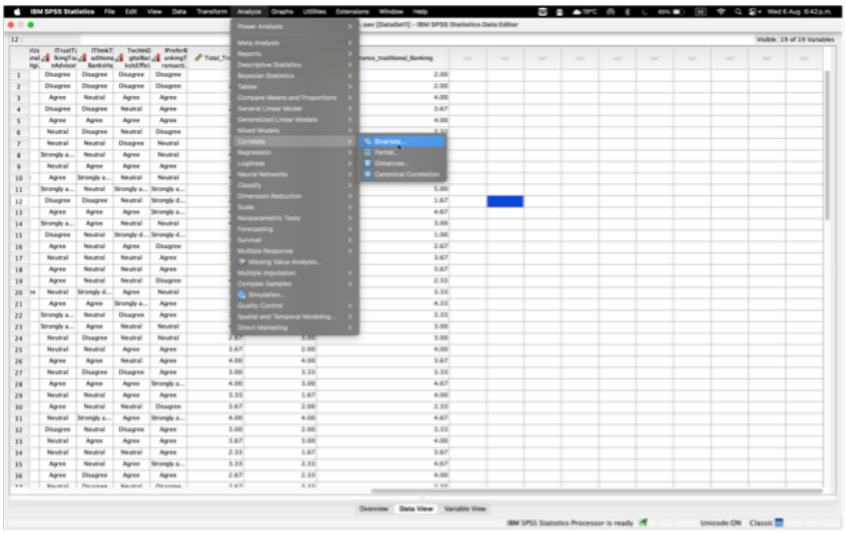


Figure 10: Correlate - Bivariate

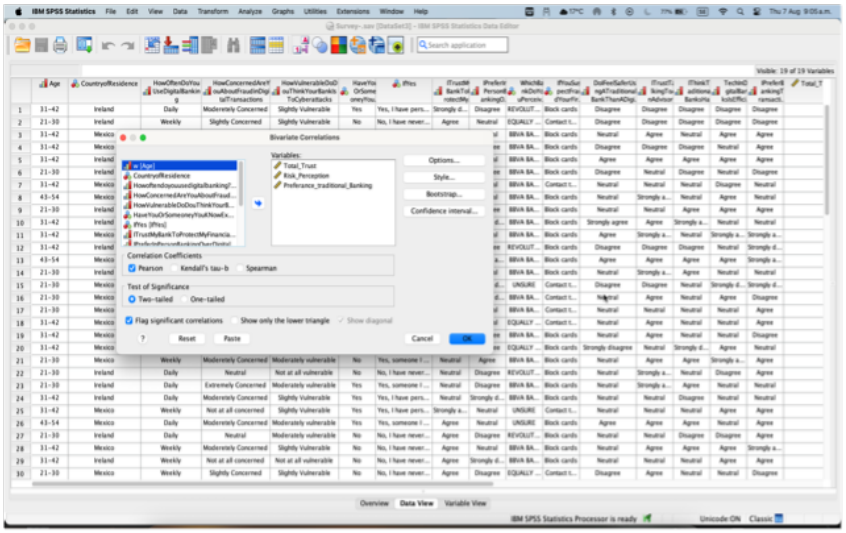


Figure 11: Select Groups to correlate

Correlations

		Total_Trust	Risk_Perception	Preferance_traditional_Banking
Total_Trust	Pearson Correlation	1	.354**	.349**
	Sig. (2-tailed)		.005	.006
	N	61	61	61
Risk_Perception	Pearson Correlation	.354**	1	.292*
	Sig. (2-tailed)	.005		.022
	N	61	61	61
Preferance_traditional_Banking	Pearson Correlation	.349**	.292*	1
	Sig. (2-tailed)	.006	.022	
	N	61	61	61

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Figure 12: Pearson Correlation Results

3.3 Third Model:

Was performed in Google Colab (Python)

3.3.1 Scraping Web

As part of the qualitative component, sentiment analysis was conducted on user reviews from digital banking platforms. The data was first collected using web scraping techniques via Google search results, targeting user feedback on BBVA and Revolut. The extracted data was stored in .csv files and included multiple fields, but only two variables were used for the analysis: **score**, representing the numerical rating, and **content**, containing the textual review. Natural Language Processing (NLP) methods were then applied to classify the emotional tone of the reviews as **positive, neutral, or negative**. This approach complements the survey results by capturing real user experiences and perceptions in an unstructured format.

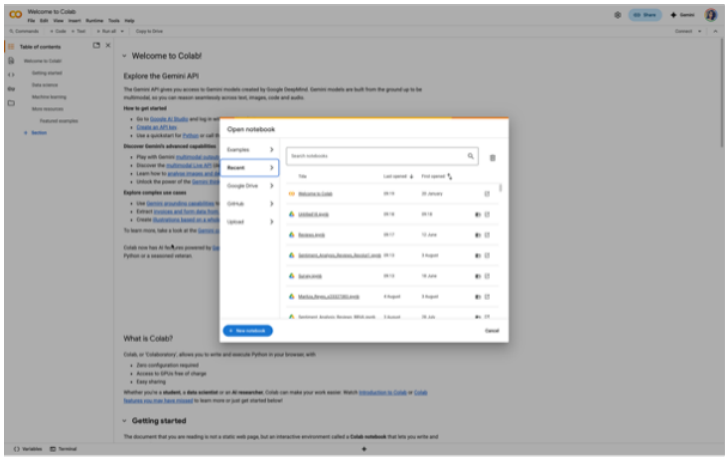


Figure 13: Open New Notebook

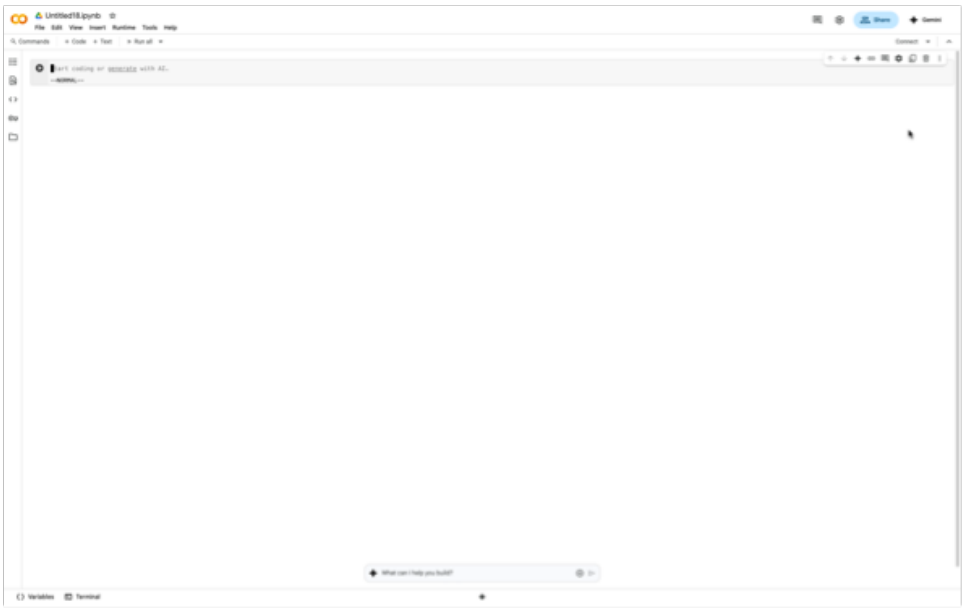


Figure 14: Connect

Figure 15: Scraping Step

3.3.2 Sentiment Analysis

Open colab and NewBook and Upload csv file.

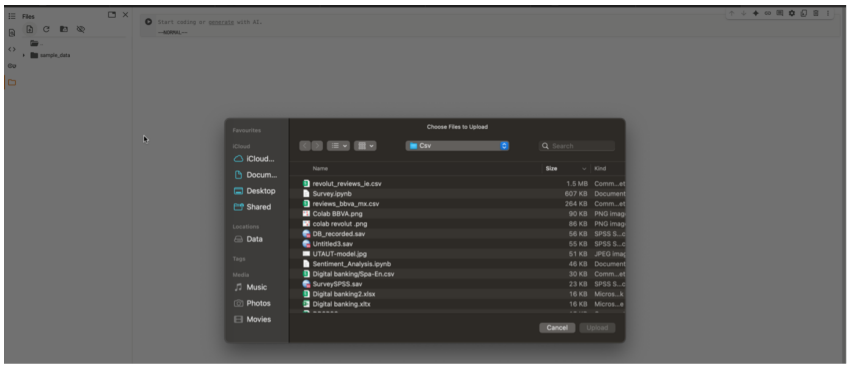


Figure 16: Upload csv

Steps to follow:

1.- Install required libraries

```
#1

!pip install deep-translator
!pip install textblob
!python -m textblob.download_corpora
!pip install langdetect
```

Figure 17: Install Libraries

2.- Import all libraries

```
#2

# Standar Libraries

import pandas as pd
import numpy as np
import re
import string
import time
import threading
import concurrent.futures
from collections import defaultdict
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import os

# Text processing
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from deep_translator import GoogleTranslator
from textblob import TextBlob
from langdetect import detect
from tqdm import tqdm

# Machine Learning

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import TfidfVectorizer

# Traditional ML models

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC


# Deep Learning

from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical

# Enable tqdm progress bars for pandas
tqdm.pandas()


# Download NLTK data
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)
nltk.download('wordnet', quiet=True)
```

Figure 18: Import Libraries

3.- Load the dataset to be analyzed

Figure 19: Load Dataset

4.- Define functions(ML.DL)

We define custom functions to handle language detection, translation, sentiment scoring (polarity and subjectivity), and text cleaning. These functions allow automated processing of reviews in different languages and prepare the text for sentiment analysis by translating, classifying emotions, and cleaning unnecessary elements like punctuation and stopwords.

Figure 20: Define functions (ML,DL)

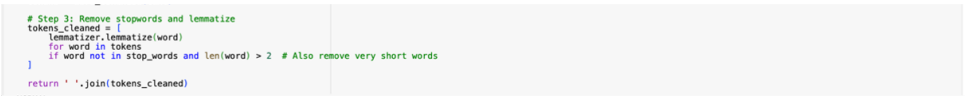


Figure 21: Define functions (ML,DL)

5.-Data cleaning(ML.DL) The dataset needs to have all errors removed and missing values addressed and inconsistent formats standardized.

Figure 22: Data Cleaning (ML,DL)

6.- Translate (ML.DL) The NLP tools require English translation of texts for proper analysis

Figure 23: Translate (ML.DL)

7.- Preprocessing(ML.DL)

The text needs to be cleaned and standardized for analysis which includes converting it to lowercase and removing symbols

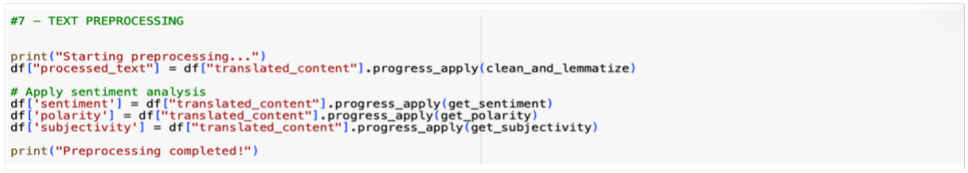


Figure 24: Preprocessing(ML.DL)

8.- Class balancing(ML. DL)

In this step, we check the distribution of sentiment labels (positive, neutral, negative) and reduce the number of samples in overrepresented classes to ensure all categories have similar size. This helps prevent biased model predictions.

```
[9] #8 - Class Balancing (undersampling majority class)

    # Check class distribution before balancing
    print("Sentiment distribution before balancing:")
    print(df["sentiment"].value_counts())  # FIXED: was "content", should be "sentiment"
    print(f"Total samples: {len(df)}")

    # Show percentage distribution
    print("\nPercentage distribution:")
    print(df["sentiment"].value_counts(normalize=True) * 100)

    # Separate data by sentiment
    positive_reviews = df[df["sentiment"] == "Positive"]  # FIXED: was "content"
    negative_reviews = df[df["sentiment"] == "Negative"]  # FIXED: was "content"
    neutral_reviews = df[df["sentiment"] == "Neutral"]    # FIXED: was "content"

    print(f"\nClass sizes before balancing:")
    print(f"Positive: {len(positive_reviews)}")
    print(f"Negative: {len(negative_reviews)}")
    print(f"Neutral: {len(neutral_reviews)}")

    # Set target sample size (based on minority class)
    min_samples = min(len(positive_reviews), len(negative_reviews), len(neutral_reviews))
    print(f"\nTarget sample size per class: {min_samples}")

    # Check if balancing is actually needed
    max_samples = max(len(positive_reviews), len(negative_reviews), len(neutral_reviews))
    imbalance_ratio = max_samples / min_samples

    print(f"Imbalance ratio: {imbalance_ratio:.2f}")

    if imbalance_ratio < 2.0:
        print("Dataset is relatively balanced. Balancing may not be necessary.")
    else:
        print("Dataset is imbalanced. Proceeding with balancing...")

    # Undersample each class to the same size
    balanced_df = pd.concat([
        positive_reviews.sample(min_samples, random_state=42),
        negative_reviews.sample(min_samples, random_state=42),
        neutral_reviews.sample(min_samples, random_state=42)
    ])

    # Shuffle the balanced dataset
    balanced_df = balanced_df.sample(frac=1, random_state=42).reset_index(drop=True)

    # Show balanced class distribution
    print(f"\nSentiment distribution after balancing:")
    print(balanced_df["sentiment"].value_counts())  # FIXED: was "content"
    print(f"Total balanced samples: {len(balanced_df)}")

    # Verify all columns are preserved
    print(f"\nColumns in balanced dataset: {balanced_df.columns.tolist()}")

    # Show data reduction
    original_size = len(df)
    balanced_size = len(balanced_df)
    reduction_percent = ((original_size - balanced_size) / original_size) * 100

    print(f"\nData reduction: {original_size:,} -> {balanced_size:,} ({reduction_percent:.1f}% reduction)")

    # Quick verification of balanced data
    print(f"\nFirst 5 samples from balanced dataset:")
    print(balanced_df[['sentiment', 'processed_text']].head())

    # Update df to use balanced dataset for further processing
    df_balanced = balanced_df.copy()
    print("\nBalanced dataset ready for machine learning!")
```

Figure 25: Class balancing(ML. DL)

9.- Tokenization and padding(DL)

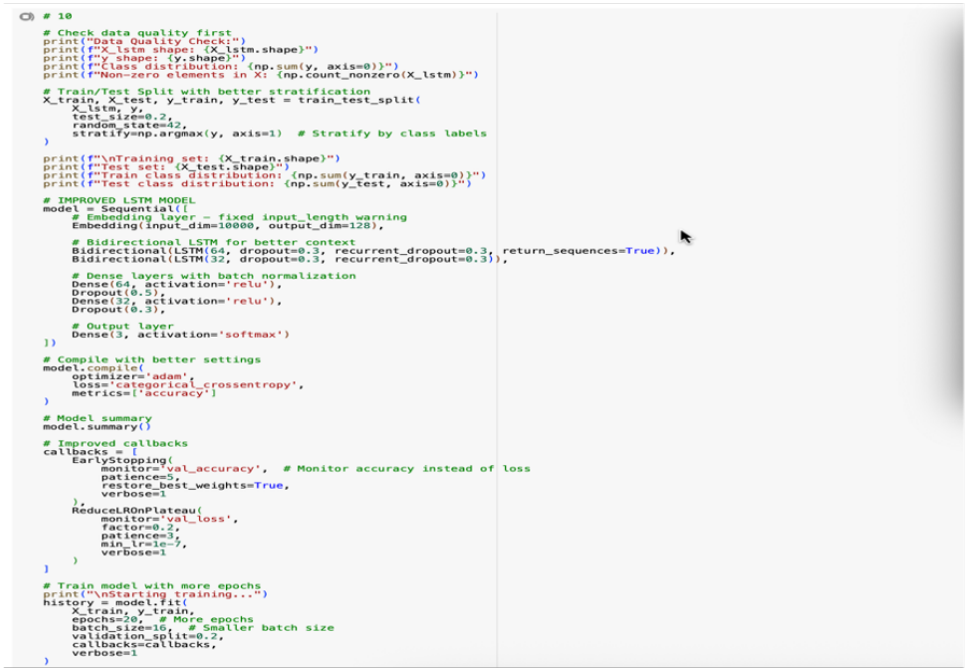
Text must be converted into numeric sequences and padded to equal length for LSTM processing.

```
#9 - Tokenization and Padding

# Initialize tokenizer
tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")
tokenizer.fit_on_texts(df_balanced["processed_text"])

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(df_balanced["processed_text"])

# Pad sequences
X_lstm = pad_sequences(sequences, padding='post', maxlen=100)

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(df_balanced["sentiment"])
y = to_categorical(y_encoded)

# Verify
print(f"X_lstm shape: {X_lstm.shape}")
print(f"y shape: {y.shape}")
print("Tokenization completed!")
```

Figure 26: Tokenization and padding(DL)

10.-Train/ test LSTM model(DL)

The data needs to be divided into training and test sets and the LSTM model should be defined.



```
# 10

# Check data quality first
print("Data Quality Check:")
print(f"X_lstm shape: {X_lstm.shape}")
print(f"y shape: {y.shape}")
print(f"Class distribution: {np.sum(y, axis=0)}")
print(f"Non-zero elements in X: {np.count_nonzero(X_lstm)}")

# Train/Test Split with better stratification
X_train, X_test, y_train, y_test = train_test_split(
    X_lstm, y,
    test_size=0.2,
    random_state=42,
    stratify=np.argmax(y, axis=1)  # Stratify by class labels
)

print(f"\nTraining set: {X_train.shape}")
print(f"Test set: {X_test.shape}")
print(f"Train class distribution: {np.sum(y_train, axis=0)}")
print(f"Test class distribution: {np.sum(y_test, axis=0)}")

# IMPROVED LSTM MODEL
model = Sequential([
    # Embedding layer - fixed input_length warning
    Embedding(input_dim=10000, output_dim=128),

    # Bidirectional LSTM for better context
    Bidirectional(LSTM(64, dropout=0.3, recurrent_dropout=0.3, return_sequences=True)),
    Bidirectional(LSTM(32, dropout=0.3, recurrent_dropout=0.3)),

    # Dense layers with batch normalization
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.3),

    # Output layer
    Dense(3, activation='softmax')
])

# Compile with better settings
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Model summary
model.summary()

# Improved callbacks
callbacks = [
    EarlyStopping(
        monitor='val_accuracy',  # Monitor accuracy instead of loss
        patience=5,
        restore_best_weights=True,
        verbose=1
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        patience=3,
        min_lr=1e-7,
        verbose=1
    )
]

# Train model with more epochs
print("\nStarting training...")
history = model.fit(
    X_train, y_train,
    epochs=20,  # More epochs
    batch_size=16,  # Smaller batch size
    validation_split=0.2,
    callbacks=callbacks,
    verbose=1
)
```

Figure 27: Train: test LSTM model(DL)

```
)

# Evaluate model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"\nFinal Test Accuracy: {test_accuracy:.4f}")

# Detailed evaluation
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

print("\nClassification Report:")
print(classification_report(y_test_classes, y_pred_classes,
                            target_names=label_encoder.classes_))

# Save model in new format
model.save('improved_lstm_sentiment_model.keras')
print("Model saved as 'improved_lstm_sentiment_model.keras'")

# Check if model is actually learning
print(f"\nTraining History Summary:")
print(f"Final training accuracy: {history.history['accuracy'][-1]:.4f}")
print(f"Final validation accuracy: {history.history['val_accuracy'][-1]:.4f}")
print(f"Best validation accuracy: {max(history.history['val_accuracy']):.4f}")

# Plot training history

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Figure 28: Train: test LSTM model(DL)

11.- Train and evaluate models (ML)

```python
print("="*60)
print("TRAINING TRADITIONAL ML MODELS")
print("="*60)

# Use balanced dataset from step #8
X_text = df_balanced['processed_text']
y_labels = df_balanced['sentiment']

print(f"Training on {len(X_text)} balanced samples")
print(f"Class distribution: {y_labels.value_counts().to_dict()}")

# 1. TF-IDF VECTORIZATION
print("\n1. TF-IDF VECTORIZATION")
print("-" * 25)

# Create TF-IDF vectors
tfidf = TfidfVectorizer(
    max_features=5000,    # Top 5000 words
    min_df=2,             # Word must appear at least 2 times
    max_df=0.95,          # Ignore words in >95% of documents
    stop_words='english', # Remove English stop words
    ngram_range=(1, 2)    # Unigrams and bigrams
)

X_tfidf = tfidf.fit_transform(X_text)
print(f"TF-IDF matrix shape: {X_tfidf.shape}")

# 2. TRAIN-TEST SPLIT
print("\n2. TRAIN-TEST SPLIT")
print("-" * 20)

X_train_tfidf, X_test_tfidf, y_train_trad, y_test_trad = train_test_split(
    X_tfidf, y_labels,
    test_size=0.2,
    random_state=42,
    stratify=y_labels
)

print(f"Training set: {X_train_tfidf.shape}")
print(f"Test set: {X_test_tfidf.shape}")

# 3. TRAIN MODELS
print("\n3. TRAINING MODELS")
print("-" * 20)

models = {}
results = {}

# Logistic Regression
print("\nTraining Logistic Regression...")
start_time = time.time()
lr_model = LogisticRegression(random_state=42, max_iter=1000)
lr_model.fit(X_train_tfidf, y_train_trad)
lr_time = time.time() - start_time

lr_pred = lr_model.predict(X_test_tfidf)
lr_accuracy = accuracy_score(y_test_trad, lr_pred)

models['Logistic Regression'] = lr_model
results['Logistic Regression'] = {
    'accuracy': lr_accuracy,
    'time': lr_time,
    'predictions': lr_pred
}

print(f"Logistic Regression - Accuracy: {lr_accuracy:.4f}, Time: {lr_time:.2f}s")
```

Figure 29: Train and evaluate models (ML)

```
[ ] # Random Forest
    print("\nTraining Random Forest...")
    start_time = time.time()
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
    rf_model.fit(X_train_tfidf, y_train_trad)
    rf_time = time.time() - start_time

    rf_pred = rf_model.predict(X_test_tfidf)
    rf_accuracy = accuracy_score(y_test_trad, rf_pred)

    models['Random Forest'] = rf_model
    results['Random Forest'] = {
        'accuracy': rf_accuracy,
        'time': rf_time,
        'predictions': rf_pred
    }

    print(f"Random Forest - Accuracy: {rf_accuracy:.4f}, Time: {rf_time:.2f}s")

    # SVM
    print("\nTraining SVM...")
    start_time = time.time()
    svm_model = SVC(kernel='linear', random_state=42)
    svm_model.fit(X_train_tfidf, y_train_trad)
    svm_time = time.time() - start_time

    svm_pred = svm_model.predict(X_test_tfidf)
    svm_accuracy = accuracy_score(y_test_trad, svm_pred)

    models['SVM'] = svm_model
    results['SVM'] = {
        'accuracy': svm_accuracy,
        'time': svm_time,
        'predictions': svm_pred
    }

    print(f"SVM - Accuracy: {svm_accuracy:.4f}, Time: {svm_time:.2f}s")

    # 4. RESULTS SUMMARY
    print("\n" + "="*60)
    print("TRADITIONAL ML RESULTS SUMMARY")
    print("="*60)

    results_df = pd.DataFrame({
        'Model': list(results.keys()),
        'Accuracy': [results[model]['accuracy'] for model in results.keys()],
        'Training Time (s)': [results[model]['time'] for model in results.keys()]
    })

    results_df = results_df.sort_values('Accuracy', ascending=False)
    print(results_df.to_string(index=False))

    # 5. DETAILED CLASSIFICATION REPORTS
    print("\n5. DETAILED CLASSIFICATION REPORTS")
    print("-" * 40)

    for model_name in results.keys():
        print(f"\n{model_name.upper()} Classification Report:")
        print("-" * 30)
        print(classification_report(y_test_trad, results[model_name]['predictions']))

    # 6. BEST TRADITIONAL MODEL
    best_traditional_model = max(results.keys(), key=lambda x: results[x]['accuracy'])
    best_accuracy = results[best_traditional_model]['accuracy']

    print(f"\nBEST TRADITIONAL MODEL: {best_traditional_model}")
    print(f"Best Accuracy: {best_accuracy:.4f}")


    # Save for comparison with LSTM
    traditional_ml_results = {
        'results_summary': results_df,
        'best_model': best_traditional_model,
        'best_accuracy': best_accuracy,
        'tfidf_vectorizer': tfidf,
        'trained_models': models
    }
```

Figure 30: Train and evaluate models (ML)

```
with open('traditional_ml_results.pkl', 'wb') as f:
    pickle.dump(traditional_ml_results, f)

print("Traditional ML results saved for comparison!")
print(f"Next: Compare with LSTM (76% accuracy)")

print("\n" + "="*60)
print("TRADITIONAL ML TRAINING COMPLETED")
print("="*60)
```

Figure 31: Train and evaluate models (ML)

12.- Model Comparison

```
print("="*70)
print("COMPREHENSIVE MODEL COMPARISON")
print("="*70)

# 1. COLLECT ALL RESULTS
print("\n1. COLLECTING ALL MODEL RESULTS")
print("-" * 35)

# LSTM Results (from your training)
lstm_accuracy = 0.76  # Replace with your actual LSTM test accuracy
lstm_training_time = 25 * 60  # ~25 minutes in seconds

# Traditional ML Results (from previous step)
try:
    with open('traditional_ml_results.pkl', 'rb') as f:
        traditional_results = pickle.load(f)

    # Extract results
    traditional_df = traditional_results['results_summary']
    print("Traditional ML results loaded successfully")

except:
    # Fallback: Manual input (replace with your actual results)
    print("Traditional ML results file not found. Using placeholder values.")
    traditional_df = pd.DataFrame({
        'Model': ['Logistic Regression', 'Random Forest', 'SVM'],
        'Accuracy': [0.72, 0.70, 0.74],  # Replace with actual results
        'Training Time (s)': [3, 15, 45]  # Replace with actual times
    })

# 2. CREATE COMPREHENSIVE COMPARISON
print("\n2. COMPREHENSIVE COMPARISON TABLE")
print("-" * 40)

# Combine all results
all_results = []

# Add LSTM results
all_results.append({
    'Model': 'LSTM (Deep Learning)',
    'Accuracy': lstm_accuracy,
    'Training Time (s)': lstm_training_time,
    'Training Time (min)': lstm_training_time / 60,
    'Model Type': 'Deep Learning',
    'Complexity': 'High',
    'Interpretability': 'Low'
})

# Add Traditional ML results
for _, row in traditional_df.iterrows():
    all_results.append({
        'Model': row['Model'],
        'Accuracy': row['Accuracy'],
        'Training Time (s)': row['Training Time (s)'],
        'Training Time (min)': row['Training Time (s)'] / 60,
        'Model Type': 'Traditional ML',
        'Complexity': 'Medium' if 'Forest' in row['Model'] else 'Low',
        'Interpretability': 'High' if 'Logistic' in row['Model'] else 'Medium'
    })

# Create comparison DataFrame
comparison_df = pd.DataFrame(all_results)
comparison_df = comparison_df.sort_values('Accuracy', ascending=False)

print("COMPLETE MODEL COMPARISON:")
print("="*80)
display_df = comparison_df[['Model', 'Accuracy', 'Training Time (min)', 'Model Type', 'Complexity']].copy()
display_df['Accuracy'] = display_df['Accuracy'].round(4)
display_df['Training Time (min)'] = display_df['Training Time (min)'].round(2)
print(display_df.to_string(index=False))

# 3. VISUAL COMPARISONS
print("\n3. VISUAL COMPARISONS")
print("-" * 25)

# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
```

Figure 32: Model comparison

```python
# Accuracy Comparison
ax1 = axes[0, 0]
bars1 = ax1.bar(comparison_df['Model'], comparison_df['Accuracy'],
                color=['red' if 'LSTM' in model else 'skyblue' for model in comparison_df['Model']])
ax1.set_title('Model Accuracy Comparison')
ax1.set_ylabel('Accuracy')
ax1.set_ylim(0, 1)
plt.setp(ax1.get_xticklabels(), rotation=45, ha='right')

# Add accuracy values on bars
for bar, acc in zip(bars1, comparison_df['Accuracy']):
    ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{acc:.3f}', ha='center', va='bottom')

# Training Time Comparison (log scale)
ax2 = axes[0, 1]
bars2 = ax2.bar(comparison_df['Model'], comparison_df['Training Time (min)'],
                color=['red' if 'LSTM' in model else 'lightgreen' for model in comparison_df['Model']])
ax2.set_title('Training Time Comparison')
ax2.set_ylabel('Training Time (minutes)')
ax2.set_yscale('log')
plt.setp(ax2.get_xticklabels(), rotation=45, ha='right')

# Add time values on bars
for bar, time_val in zip(bars2, comparison_df['Training Time (min)']):
    ax2.text(bar.get_x() + bar.get_width()/2, bar.get_height() * 1.1,
             f'{time_val:.1f}m', ha='center', va='bottom')

# Accuracy vs Training Time Scatter
ax3 = axes[1, 0]
colors = ['red' if 'LSTM' in model else 'blue' for model in comparison_df['Model']]
scatter = ax3.scatter(comparison_df['Training Time (min)'], comparison_df['Accuracy'],
                      c=colors, s=100, alpha=0.7)
ax3.set_xlabel('Training Time (minutes)')
ax3.set_ylabel('Accuracy')
ax3.set_title('Accuracy vs Training Time')
ax3.set_xscale('log')

# Add model labels
for i, model in enumerate(comparison_df['Model']):
    ax3.annotate(model.split()[0],
                 (comparison_df['Training Time (min)'].iloc[i], comparison_df['Accuracy'].iloc[i]),
                 xytext=(5, 5), textcoords='offset points', fontsize=8)

# Model Type Summary
ax4 = axes[1, 1]
model_type_summary = comparison_df.groupby('Model Type')['Accuracy'].agg(['mean', 'max', 'count'])
x_pos = range(len(model_type_summary))
bars4 = ax4.bar(x_pos, model_type_summary['mean'],
                color=['red', 'skyblue'], alpha=0.7)
ax4.set_title('Average Accuracy by Model Type')
ax4.set_ylabel('Average Accuracy')
ax4.set_xticks(x_pos)
ax4.set_xticklabels(model_type_summary.index)

# Add values on bars
for bar, val in zip(bars4, model_type_summary['mean']):
    ax4.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{val:.3f}', ha='center', va='bottom')

plt.tight_layout()
plt.show()

# 4. DETAILED ANALYSIS
print("\n4. DETAILED ANALYSIS")
print("-" * 25)

best_accuracy_model = comparison_df.iloc[0]
fastest_model = comparison_df.loc[comparison_df['Training Time (s)'].idxmin()]
best_traditional = comparison_df[comparison_df['Model Type'] == 'Traditional ML'].iloc[0]

print(f"BEST OVERALL ACCURACY:")
print(f"   Model: {best_accuracy_model['Model']}")
print(f"   Accuracy: {best_accuracy_model['Accuracy']:.4f}")
print(f"   Training Time: {best_accuracy_model['Training Time (min)']:.1f} minutes")

print(f"\n⚡ FASTEST TRAINING:")
print(f"   Model: {fastest_model['Model']}")
print(f"   Training Time: {fastest_model['Training Time (min)']:.1f} minutes")
print(f"   Accuracy: {fastest_model['Accuracy']:.4f}")
```

Figure 33: Model comparison

```python
# 5. RECOMMENDATIONS
print("\n5. RECOMMENDATIONS")
print("-" * 20)

accuracy_diff = best_accuracy_model['Accuracy'] - best_traditional['Accuracy']
time_diff = best_accuracy_model['Training Time (min)'] / best_traditional['Training Time (min)']

print(f"\nLSTM vs Best Traditional ML:")
print(f"   Accuracy difference: {accuracy_diff:.4f} ({accuracy_diff*100:.1f}%)")
print(f"   Time difference: {time_diff:.1f}x slower")

if accuracy_diff > 0.05:  # 5% better
    print(f"\nRECOMMENDATION: LSTM")
    print(f"   Reason: Significantly better accuracy (+{accuracy_diff*100:.1f}%)")
elif accuracy_diff > 0.02:  # 2% better
    print(f"\nRECOMMENDATION: Depends on use case")
    print(f"   LSTM: Better accuracy but slower training")
    print(f"   Traditional ML: Faster training, good enough accuracy")
else:
    print(f"\nRECOMMENDATION: {best_traditional['Model']}")
    print(f"   Reason: Similar accuracy but much faster training")

# 6. SAVE COMPARISON RESULTS
print("\n6. SAVING COMPARISON RESULTS")
print("-" * 35)

comparison_df.to_csv('model_comparison_results.csv', index=False)
print("Comparison results saved to 'model_comparison_results.csv'")

print("\n" + "="*70)
print("MODEL COMPARISON COMPLETED")
print("="*70)
```

Figure 34: Model comparison