5500 Final Project Group 3

In [1]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.utils import class_weight

# Deep learning libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv1D, MaxPooling1D, Flatten, LSTM, SimpleRNN
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Load datasets (replace with your actual file paths)
try:
    train_data = pd.read_csv('trainset.csv')
    test_data = pd.read_csv('testset.csv')
except FileNotFoundError:
    print("Please make sure the files 'trainset.csv' and 'testset.csv' are in the correct directory.")
    raise

# Step 1: Data Exploration
print("\n=== Data Exploration ===\n")
print("Training set shape:", train_data.shape)
print("Test set shape:", test_data.shape)
print("\nTraining set class distribution:")
print(train_data['Subscribed'].value_counts())
print("\nTest set class distribution:")
```

```
print(test_data['Subscribed'].value_counts())

# Visualize class distribution
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
train_data['Subscribed'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Training Set Class Distribution')
plt.subplot(1, 2, 2)
test_data['Subscribed'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Test Set Class Distribution')
plt.tight_layout()
plt.show()

# Explore categorical features
categorical_features = ['job', 'marital', 'education', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcom
plt.figure(figsize=(20, 25))
for i, feature in enumerate(categorical_features, 1):
    plt.subplot(5, 2, i)
    sns.countplot(x=feature, hue='Subscribed', data=train_data, palette='viridis')
    plt.title(f'{feature} Distribution')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Explore numerical features
numerical_features = ['age', 'duration', 'campaign', 'pdays', 'nr.employed']
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
    plt.title(f'{feature} Distribution')
plt.tight_layout()
plt.show()

# Step 2: Data Preprocessing
print("\n=== Data Preprocessing ===\n")

# Combine train and test for consistent preprocessing
combined = pd.concat([train_data, test_data], axis=0)

# Handle 'unknown' values - replace with mode or appropriate value
for column in categorical_features:
```

```
    mode_val = combined[column].mode()[0]
    combined[column] = combined[column].replace('unknown', mode_val)

# Convert pdays=999 to -1 (indicator for not previously contacted)
combined['pdays'] = combined['pdays'].replace(999, -1)

# Encode categorical variables
label_encoders = {}
for column in categorical_features:
    le = LabelEncoder()
    combined[column] = le.fit_transform(combined[column])
    label_encoders[column] = le

# Encode target variable
target_encoder = LabelEncoder()
combined['Subscribed'] = target_encoder.fit_transform(combined['Subscribed'])

# Split back into train and test
train_data = combined.iloc[:len(train_data)]
test_data = combined.iloc[len(train_data):]

# Separate features and target
X_train = train_data.drop('Subscribed', axis=1)
y_train = train_data['Subscribed']
X_test = test_data.drop('Subscribed', axis=1)
y_test = test_data['Subscribed']

# Scale numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Handle class imbalance
class_weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights = dict(enumerate(class_weights))

# Reshape data for CNN and RNN
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)

# Step 3: Model Development and Training
print("\n=== Model Development and Training ===\n")
```

```
# MLP Model
def build_mlp():
    model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

mlp_model = build_mlp()
print("MLP Model Summary:")
mlp_model.summary()

# CNN Model
def build_cnn():
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_reshaped.shape[1], 1)),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

cnn_model = build_cnn()
print("\nCNN Model Summary:")
cnn_model.summary()

# LSTM Model
def build_lstm():
    model = Sequential([
        LSTM(64, input_shape=(X_train_reshaped.shape[1], 1), return_sequences=True),
```

```
        Dropout(0.3),
        LSTM(32),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

lstm_model = build_lstm()
print("\nLSTM Model Summary:")
lstm_model.summary()

# RNN Model
def build_rnn():
    model = Sequential([
        SimpleRNN(64, input_shape=(X_train_reshaped.shape[1], 1)),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

rnn_model = build_rnn()
print("\nRNN Model Summary:")
rnn_model.summary()

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train MLP
print("\nTraining MLP Model...")
mlp_history = mlp_model.fit(X_train_scaled, y_train,
                           validation_split=0.2,
                           epochs=50,
                           batch_size=64,
                           class_weight=class_weights,
```

```
                           callbacks=[early_stopping],
                           verbose=1)

# Train CNN
print("\nTraining CNN Model...")
cnn_history = cnn_model.fit(X_train_reshaped, y_train,
                           validation_split=0.2,
                           epochs=50,
                           batch_size=64,
                           class_weight=class_weights,
                           callbacks=[early_stopping],
                           verbose=1)

# Train LSTM
print("\nTraining LSTM Model...")
lstm_history = lstm_model.fit(X_train_reshaped, y_train,
                             validation_split=0.2,
                             epochs=50,
                             batch_size=64,
                             class_weight=class_weights,
                             callbacks=[early_stopping],
                             verbose=1)

# Train RNN
print("\nTraining RNN Model...")
rnn_history = rnn_model.fit(X_train_reshaped, y_train,
                           validation_split=0.2,
                           epochs=50,
                           batch_size=64,
                           class_weight=class_weights,
                           callbacks=[early_stopping],
                           verbose=1)

# Step 4: Model Evaluation and Comparison
print("\n=== Model Evaluation and Comparison ===\n")

def evaluate_model(model, X_test, y_test, model_name):
    y_pred_prob = model.predict(X_test)
    y_pred = (y_pred_prob > 0.5).astype(int)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
```

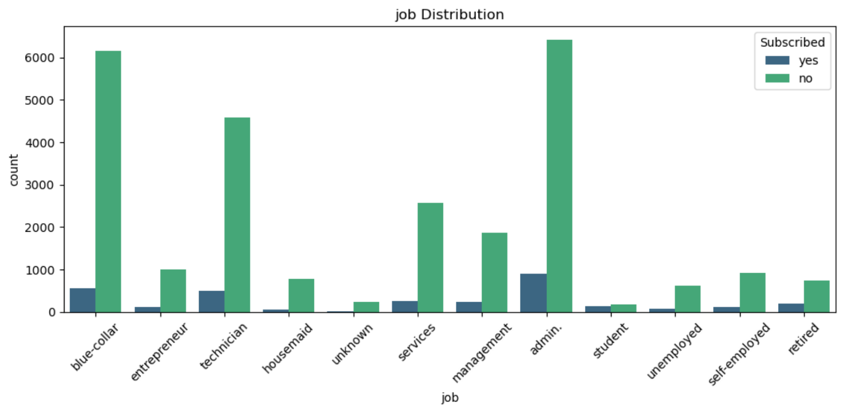
```
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"{model_name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    # Plot confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['No', 'Yes'],
                yticklabels=['No', 'Yes'])
    plt.title(f'{model_name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    return accuracy, precision, recall, f1

# Evaluate MLP
mlp_metrics = evaluate_model(mlp_model, X_test_scaled, y_test, 'MLP')

# Evaluate CNN
cnn_metrics = evaluate_model(cnn_model, X_test_reshaped, y_test, 'CNN')

# Evaluate LSTM
lstm_metrics = evaluate_model(lstm_model, X_test_reshaped, y_test, 'LSTM')

# Evaluate RNN
rnn_metrics = evaluate_model(rnn_model, X_test_reshaped, y_test, 'RNN')

# Plot training history for all models
def plot_history(history, model_name):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epoch')
```

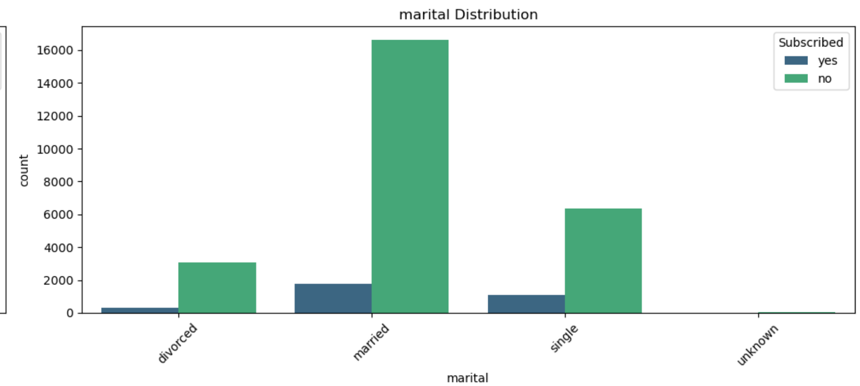
```
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.tight_layout()
    plt.show()

plot_history(mlp_history, 'MLP')
plot_history(cnn_history, 'CNN')
plot_history(lstm_history, 'LSTM')
plot_history(rnn_history, 'RNN')

# Compare all models' performance
metrics_df = pd.DataFrame({
    'Model': ['MLP', 'CNN', 'LSTM', 'RNN'],
    'Accuracy': [mlp_metrics[0], cnn_metrics[0], lstm_metrics[0], rnn_metrics[0]],
    'Precision': [mlp_metrics[1], cnn_metrics[1], lstm_metrics[1], rnn_metrics[1]],
    'Recall': [mlp_metrics[2], cnn_metrics[2], lstm_metrics[2], rnn_metrics[2]],
    'F1 Score': [mlp_metrics[3], cnn_metrics[3], lstm_metrics[3], rnn_metrics[3]]
})

print("\n=== Model Performance Comparison ===")
print(metrics_df)

# Visual comparison of model metrics
plt.figure(figsize=(15, 8))
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
for i, metric in enumerate(metrics, 1):
    plt.subplot(2, 2, i)
    sns.barplot(x='Model', y=metric, data=metrics_df)
    plt.title(metric)
    plt.ylim(0, 1)
plt.tight_layout()
plt.show()

# Step 5: Final Test Results
```

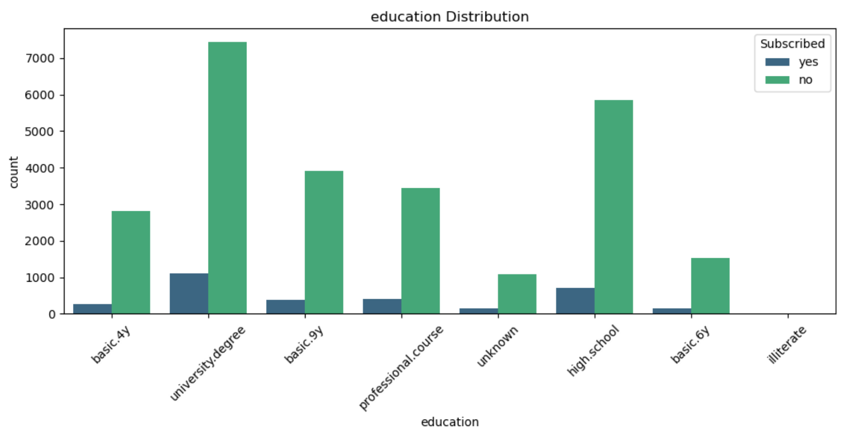
```
print("\n=== Final Test Results ===\n")
print("The models have been evaluated on the test set. The performance metrics are summarized above.")
print("The best performing model based on F1 Score is:", metrics_df.loc[metrics_df['F1 Score'].idxmax(), 'Model'])
```

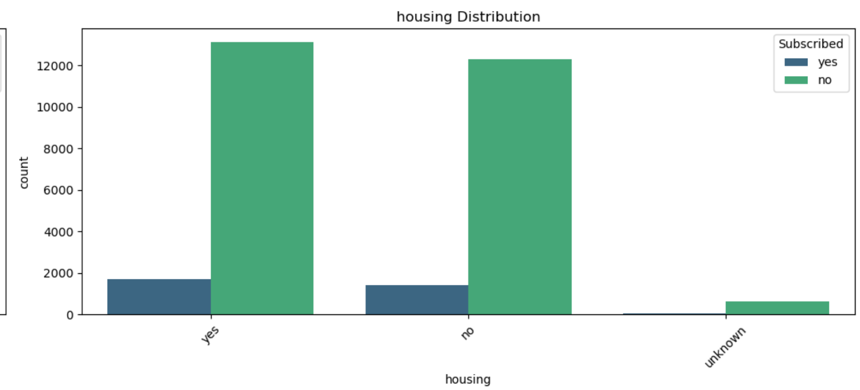
```
=== Data Exploration ===

Training set shape: (29271, 15)
Test set shape: (11917, 15)

Training set class distribution:
Subscribed
no     26075
yes     3196
Name: count, dtype: int64

Test set class distribution:
Subscribed
no     10473
yes     1444
Name: count, dtype: int64
```

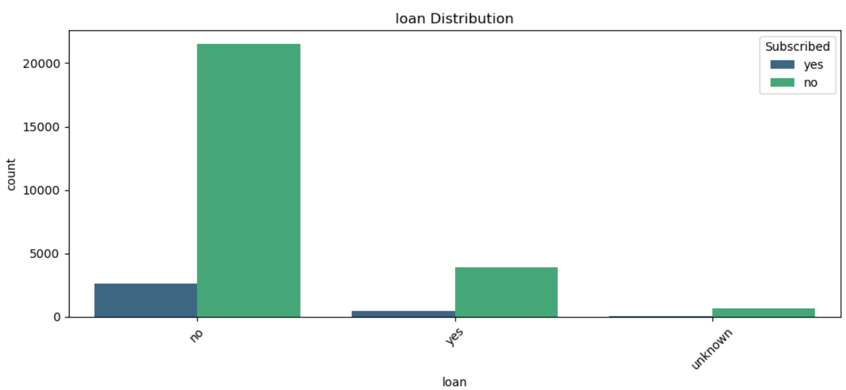

Training Set Class Distribution
25000
20000
15000
10000
5000
0
no
yes
Subscribed
Test Set Class Distribution
10000
8000
6000
4000
2000
0
no
yes
Subscribed

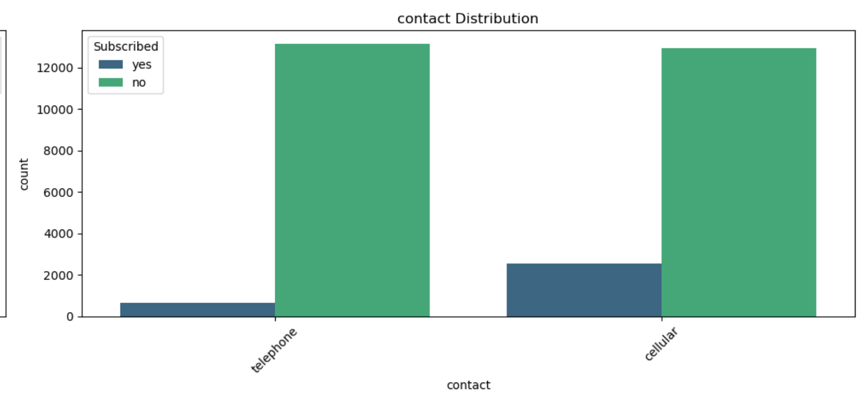

job Distribution
Subscribed
yes
no
count
0
1000
2000
3000
4000
5000
6000
blue-collar
entrepreneur
technician
housemaid
unknown
services
management
admin.
student
unemployed
self-employed
retired
job

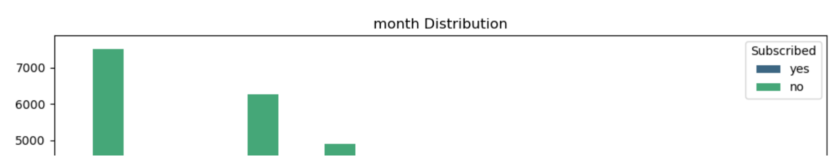

marital Distribution
Subscribed
yes
no
count
0
2000
4000
6000
8000
10000
12000
14000
16000
divorced
married
single
unknown
marital

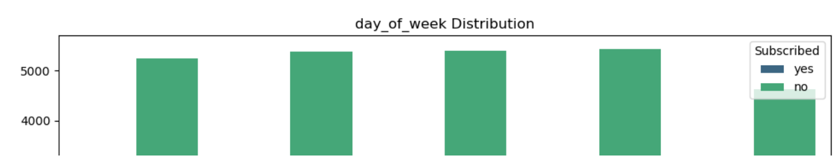

education Distribution
Subscribed
yes
no
count
0
1000
2000
3000
4000
5000
6000
7000
basic.4y
university.degree
basic.9y
professional.course
unknown
high.school
basic.6y
illiterate
education

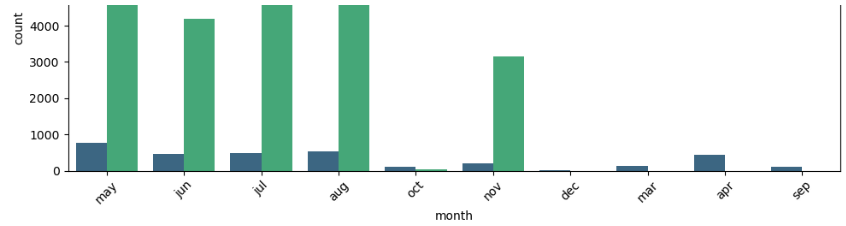

housing Distribution
Subscribed
yes
no
count
0
2000
4000
6000
8000
10000
12000
yes
no
unknown
housing

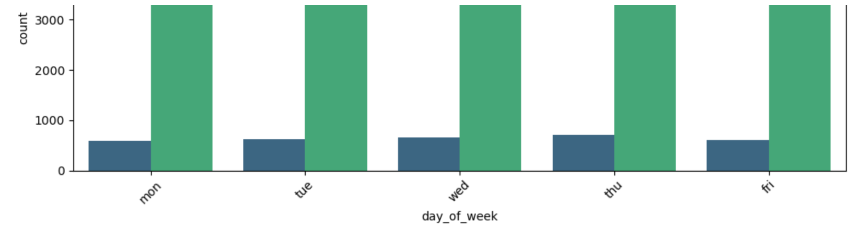

loan Distribution
Subscribed
yes
no
count
0
5000
10000
15000
20000
no
yes
unknown
loan

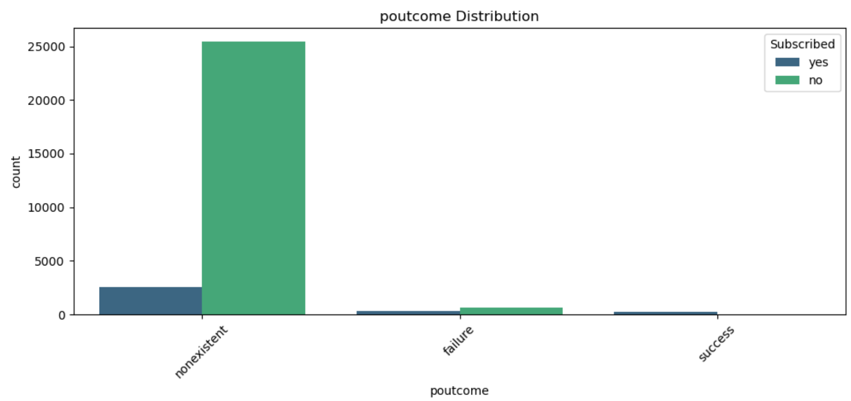

contact Distribution
Subscribed
yes
no
count
0
2000
4000
6000
8000
10000
12000
telephone
cellular
contact


month Distribution
Subscribed
yes
no
5000
6000
7000


day_of_week Distribution
Subscribed
yes
no
4000
5000

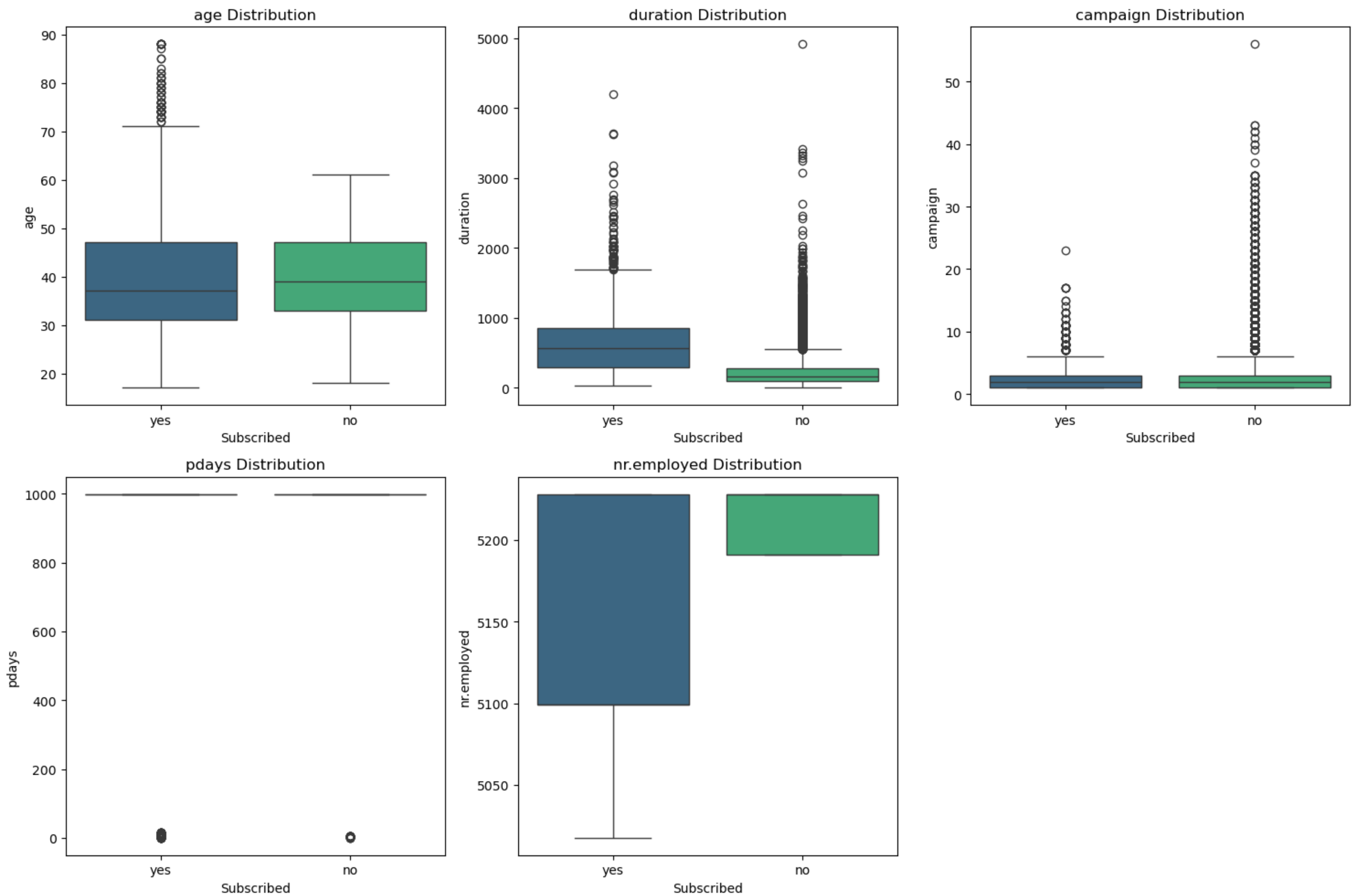







```
C:\Users\Tom\AppData\Local\Temp\ipykernel_17180\2116035293.py:67: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
C:\Users\Tom\AppData\Local\Temp\ipykernel_17180\2116035293.py:67: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
C:\Users\Tom\AppData\Local\Temp\ipykernel_17180\2116035293.py:67: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
C:\Users\Tom\AppData\Local\Temp\ipykernel_17180\2116035293.py:67: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
```

```
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
C:\Users\Tom\AppData\Local\Temp\ipykernel_17180\2116035293.py:67: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
```



=== Data Preprocessing ===

=== Model Development and Training ===

```
C:\Users\Tom\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape
`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first l
ayer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
MLP Model Summary:
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	960
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 1)	33

Total params: 3,073 (12.00 KB)

Trainable params: 3,073 (12.00 KB)

Non-trainable params: 0 (0.00 B)

```
CNN Model Summary:
```

```
C:\Users\Tom\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 12, 64)	256
max_pooling1d (MaxPooling1D)	(None, 6, 64)	0
flatten (Flatten)	(None, 384)	0
dense_3 (Dense)	(None, 32)	12,320
dropout_2 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 1)	33

Total params: 12,609 (49.25 KB)

Trainable params: 12,609 (49.25 KB)

Non-trainable params: 0 (0.00 B)

```
LSTM Model Summary:
C:\Users\Tom\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:200: UserWarning: Do not pass an `input_shape`/`
input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first laye
r in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 14, 64)	16,896
dropout_3 (Dropout)	(None, 14, 64)	0
lstm_1 (LSTM)	(None, 32)	12,416
dropout_4 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 1)	33

Total params: 29,345 (114.63 KB)

Trainable params: 29,345 (114.63 KB)

Non-trainable params: 0 (0.00 B)

```
RNN Model Summary:
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 64)	4,224
dropout_5 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 32)	2,080
dropout_6 (Dropout)	(None, 32)	0
dense_7 (Dense)	(None, 1)	33

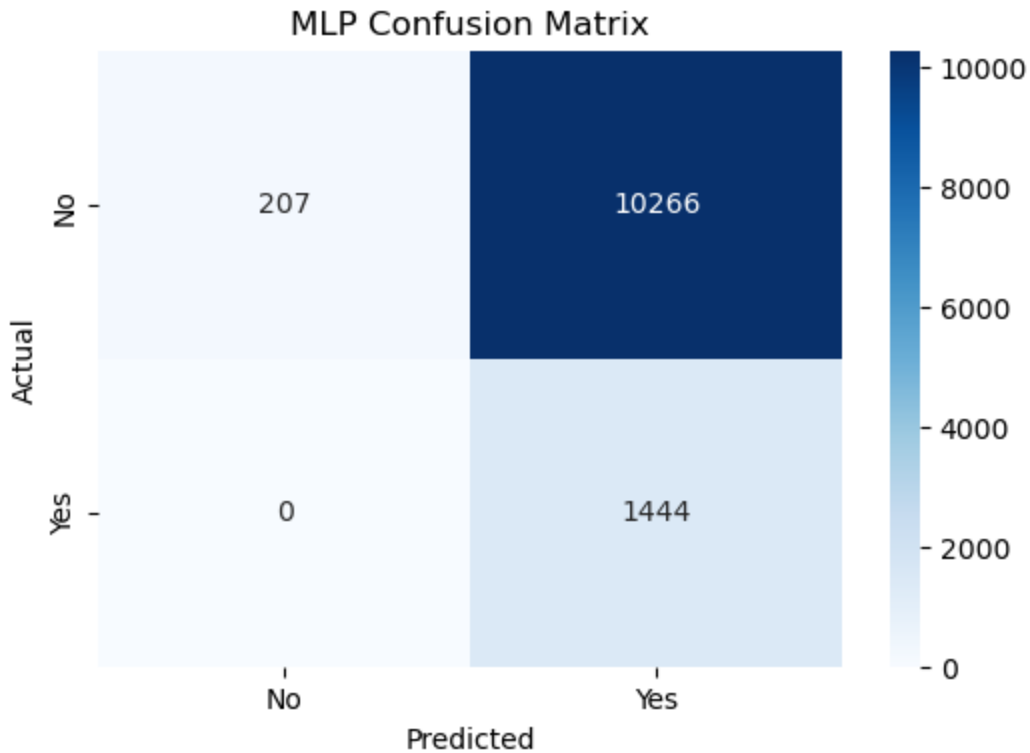
Total params: 6,337 (24.75 KB)

Trainable params: 6,337 (24.75 KB)

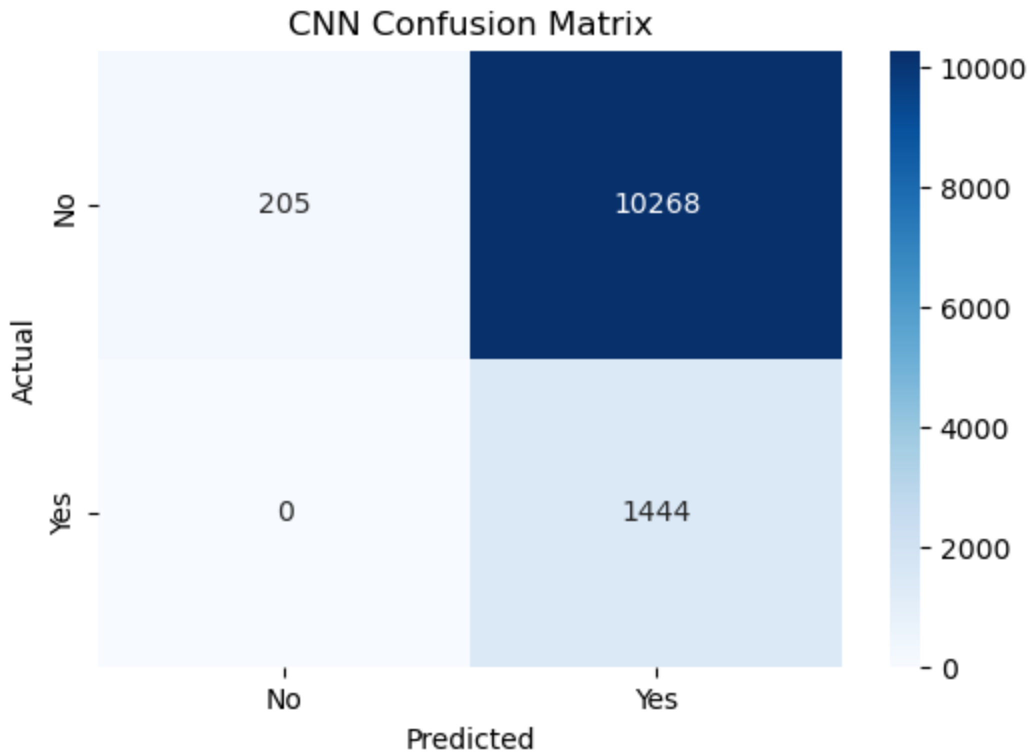
Non-trainable params: 0 (0.00 B)

```
Training MLP Model...
Epoch 1/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.7181 - loss: 0.5234 - val_accuracy: 0.7522 - val_loss: 0.5797
Epoch 2/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9186 - loss: 0.2326 - val_accuracy: 0.7339 - val_loss: 0.6220
Epoch 3/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9196 - loss: 0.2105 - val_accuracy: 0.6499 - val_loss: 0.9073
Epoch 4/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9171 - loss: 0.1929 - val_accuracy: 0.5908 - val_loss: 1.1683
Epoch 5/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9178 - loss: 0.1831 - val_accuracy: 0.4941 - val_loss: 1.7673
Epoch 6/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9172 - loss: 0.1785 - val_accuracy: 0.4634 - val_loss: 2.6099

Training CNN Model...
Epoch 1/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.7965 - loss: 0.3702 - val_accuracy: 0.8167 - val_loss: 1.0615
Epoch 2/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9227 - loss: 0.2183 - val_accuracy: 0.7667 - val_loss: 1.3201
Epoch 3/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9260 - loss: 0.1889 - val_accuracy: 0.5974 - val_loss: 1.8204
Epoch 4/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9254 - loss: 0.1769 - val_accuracy: 0.5132 - val_loss: 2.3651
Epoch 5/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9264 - loss: 0.1661 - val_accuracy: 0.4893 - val_loss: 3.0741
Epoch 6/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9273 - loss: 0.1587 - val_accuracy: 0.4811 - val_loss: 3.2162

Training LSTM Model...
Epoch 1/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 13s 23ms/step - accuracy: 0.4916 - loss: 0.5874 - val_accuracy: 0.6767 - val_loss: 0.979
2
Epoch 2/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.8739 - loss: 0.2883 - val_accuracy: 0.6372 - val_loss: 1.1410
Epoch 3/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.8942 - loss: 0.2622 - val_accuracy: 0.6835 - val_loss: 1.0677
Epoch 4/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.9135 - loss: 0.2405 - val_accuracy: 0.6962 - val_loss: 1.0065
Epoch 5/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.9171 - loss: 0.2283 - val_accuracy: 0.6989 - val_loss: 1.0192
Epoch 6/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 8s 23ms/step - accuracy: 0.9203 - loss: 0.2191 - val_accuracy: 0.6994 - val_loss: 1.0459
```

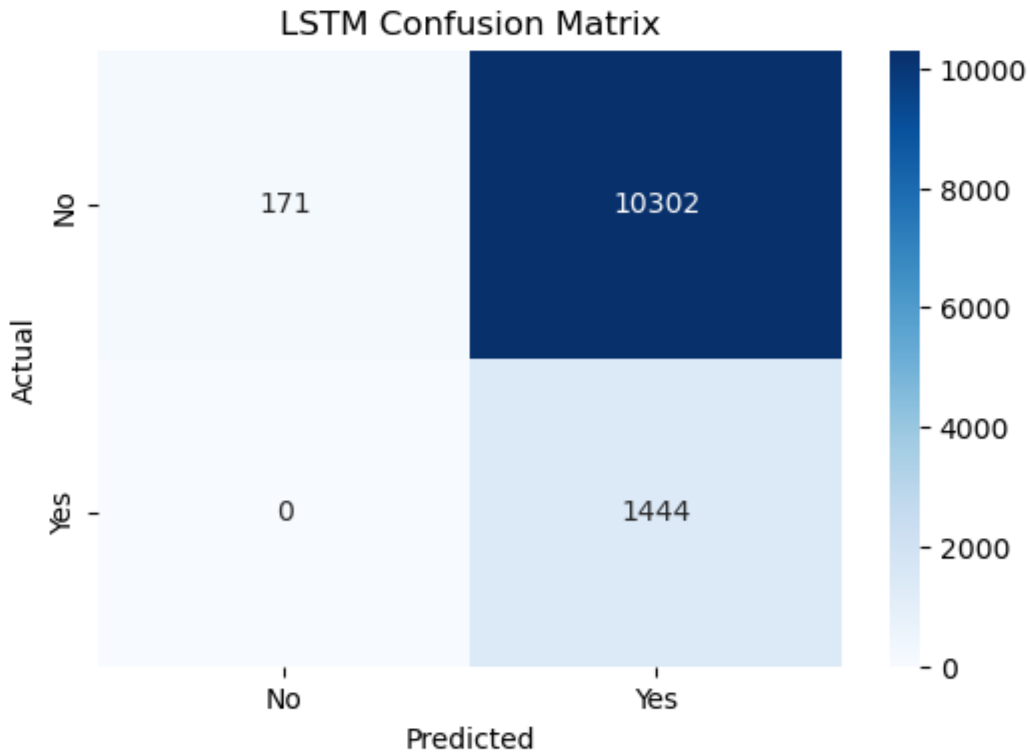
```
Training RNN Model...
Epoch 1/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.8590 - loss: 0.3479 - val_accuracy: 0.7289 - val_loss: 0.5845
Epoch 2/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9136 - loss: 0.2147 - val_accuracy: 0.7392 - val_loss: 0.6096
Epoch 3/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9131 - loss: 0.2064 - val_accuracy: 0.6227 - val_loss: 0.8852
Epoch 4/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9132 - loss: 0.1894 - val_accuracy: 0.5474 - val_loss: 1.2928
Epoch 5/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9171 - loss: 0.1757 - val_accuracy: 0.5095 - val_loss: 1.8977
Epoch 6/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9172 - loss: 0.1708 - val_accuracy: 0.4883 - val_loss: 2.0044

=== Model Evaluation and Comparison ===

373/373 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
MLP Performance:
Accuracy: 0.1385
Precision: 0.1233
Recall: 1.0000
F1 Score: 0.2196
```



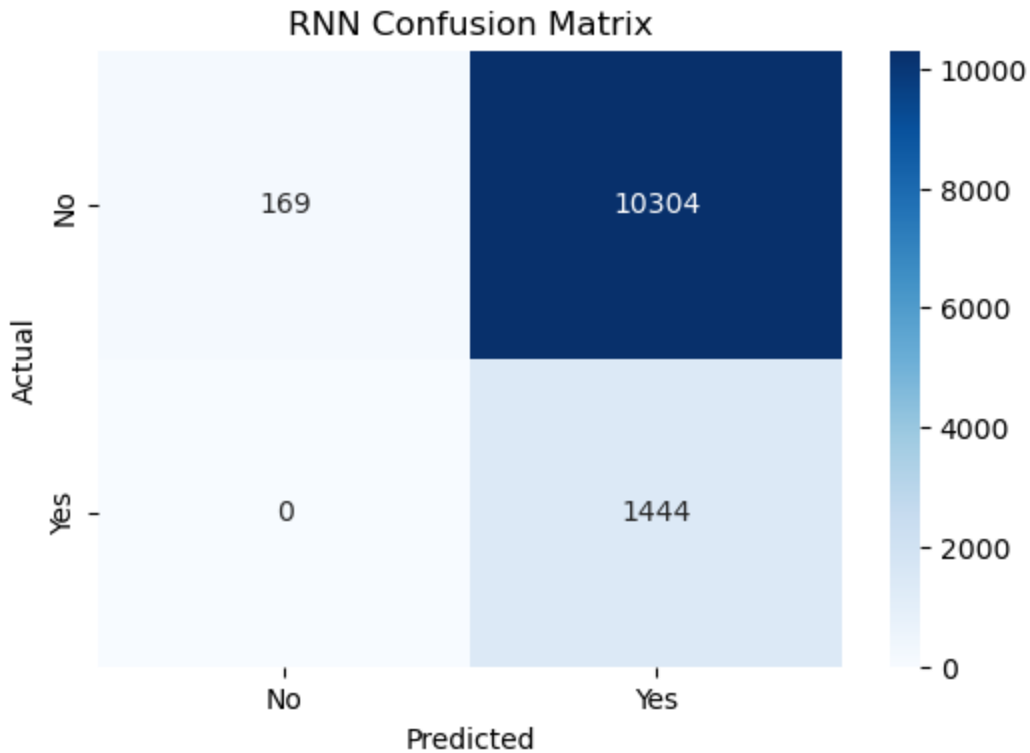

```
373/373 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
CNN Performance:
Accuracy: 0.1384
Precision: 0.1233
Recall: 1.0000
F1 Score: 0.2195
```

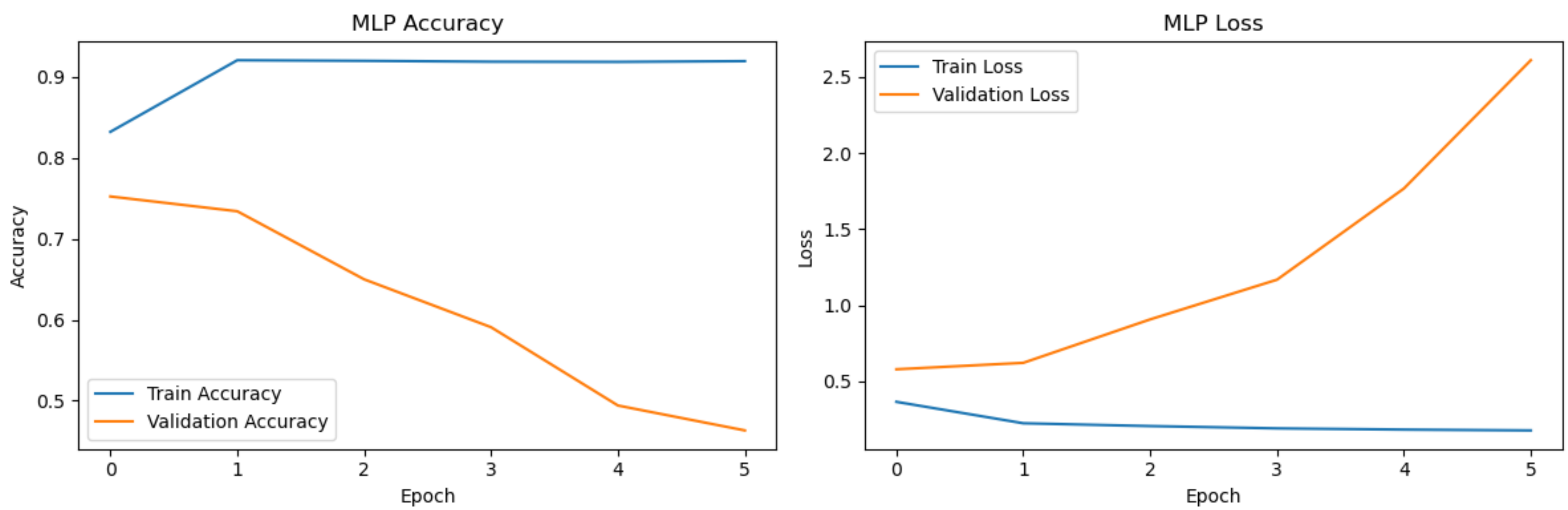


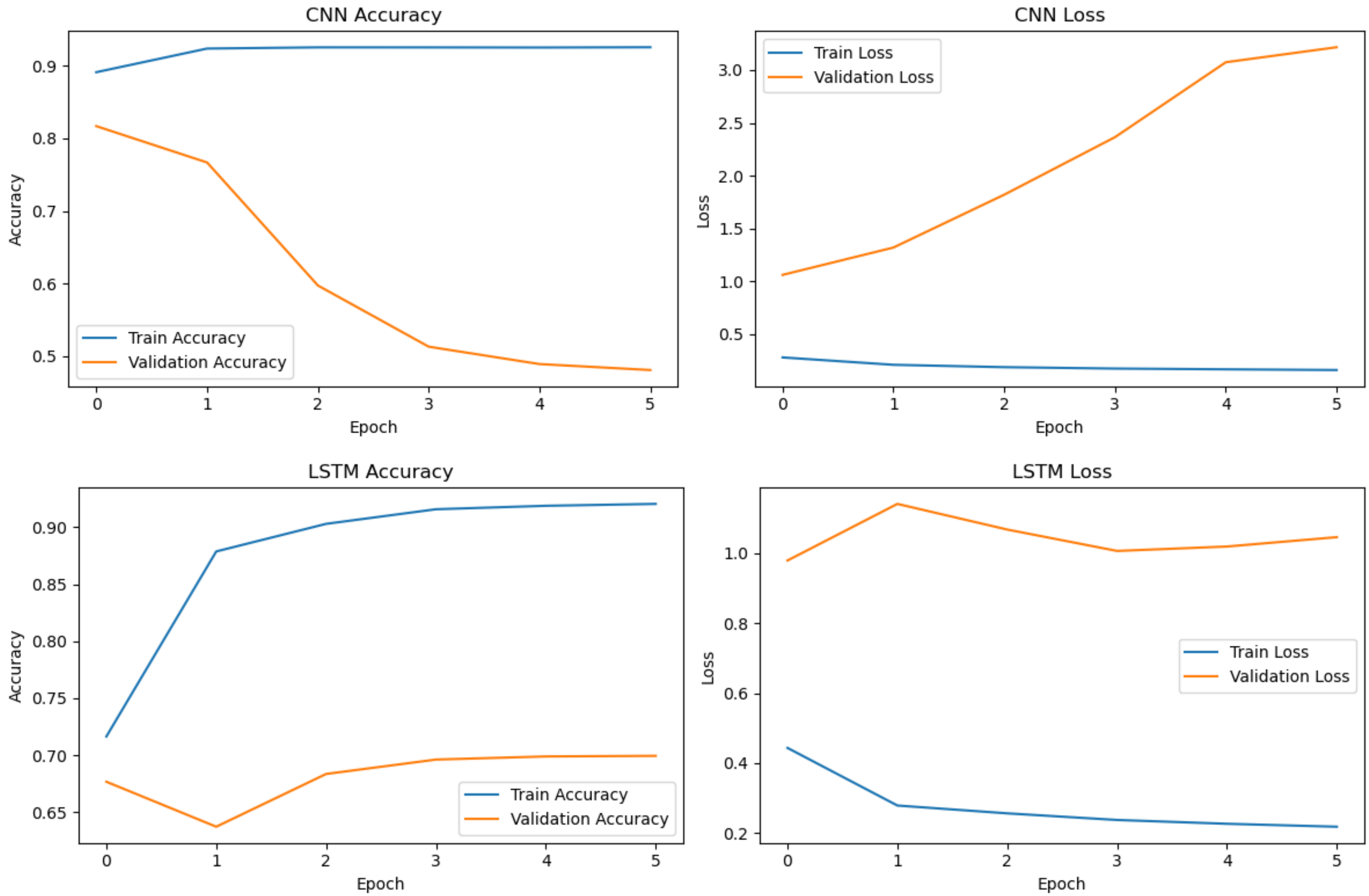

```
373/373 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
LSTM Performance:
Accuracy: 0.1355
Precision: 0.1229
Recall: 1.0000
F1 Score: 0.2190
```

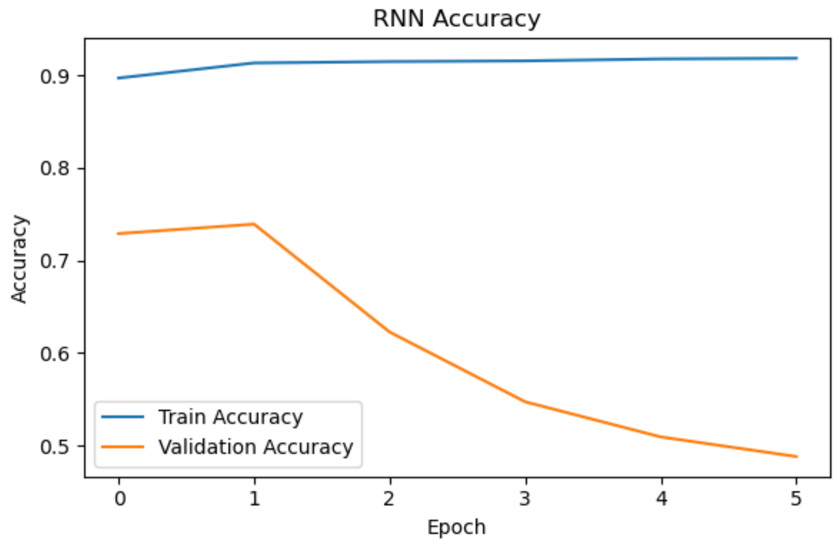


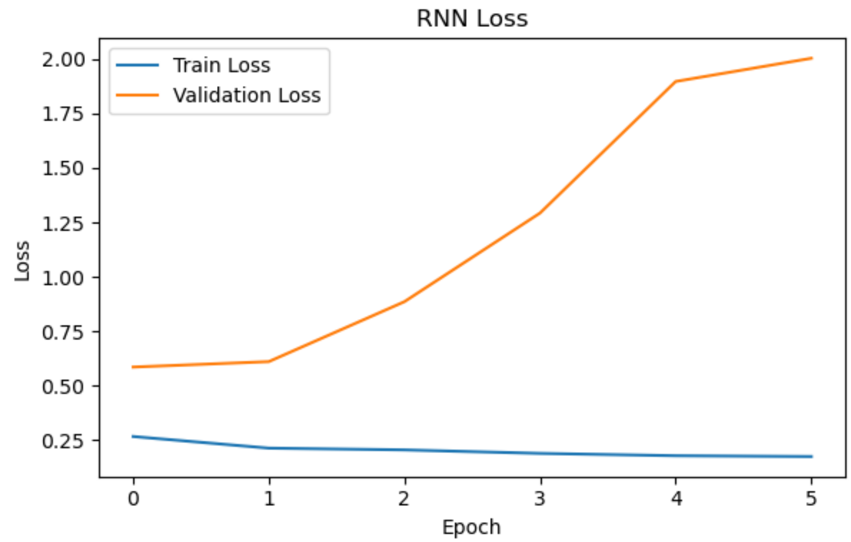

```
373/373 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
RNN Performance:
Accuracy: 0.1354
Precision: 0.1229
Recall: 1.0000
F1 Score: 0.2189
```


RNN Confusion Matrix
No
Yes
Actual
169
10304
0
1444
No
Yes
Predicted
10000
8000
6000
4000
2000
0

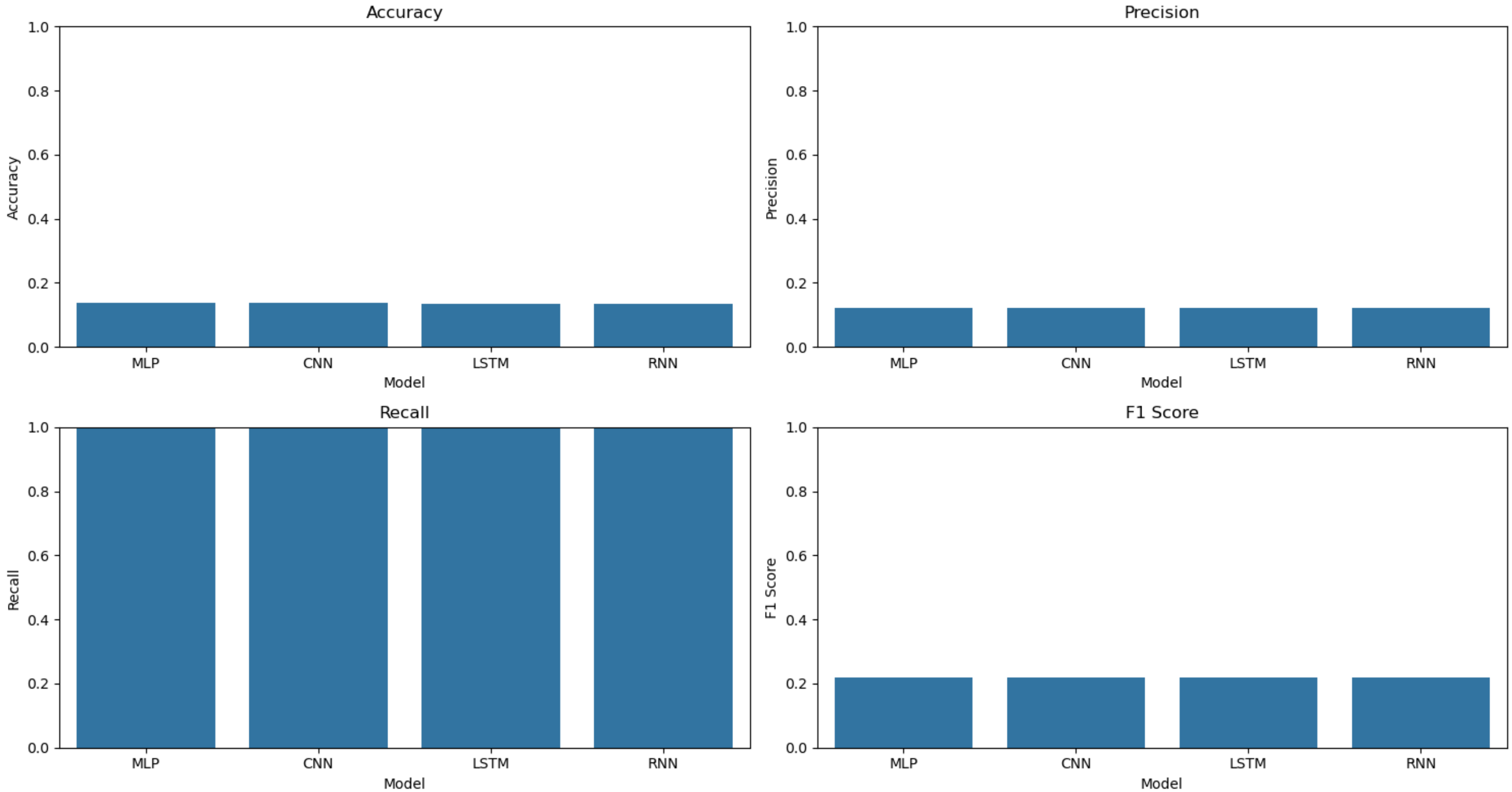

MLP Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy
0.9
0.8
0.7
0.6
0.5
0
1
2
3
4
5
MLP Loss
Loss
Epoch
Train Loss
Validation Loss
2.5
2.0
1.5
1.0
0.5
0
1
2
3
4
5


CNN Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy
CNN Loss
Loss
Epoch
Train Loss
Validation Loss
LSTM Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy
LSTM Loss
Loss
Epoch
Train Loss
Validation Loss






```
=== Model Performance Comparison ===
  Model  Accuracy  Precision  Recall  F1 Score
0   MLP  0.138542   0.123313     1.0  0.219553
1   CNN  0.138374   0.123292     1.0  0.219520
2  LSTM  0.135521   0.122935     1.0  0.218954
3   RNN  0.135353   0.122915     1.0  0.218921
```

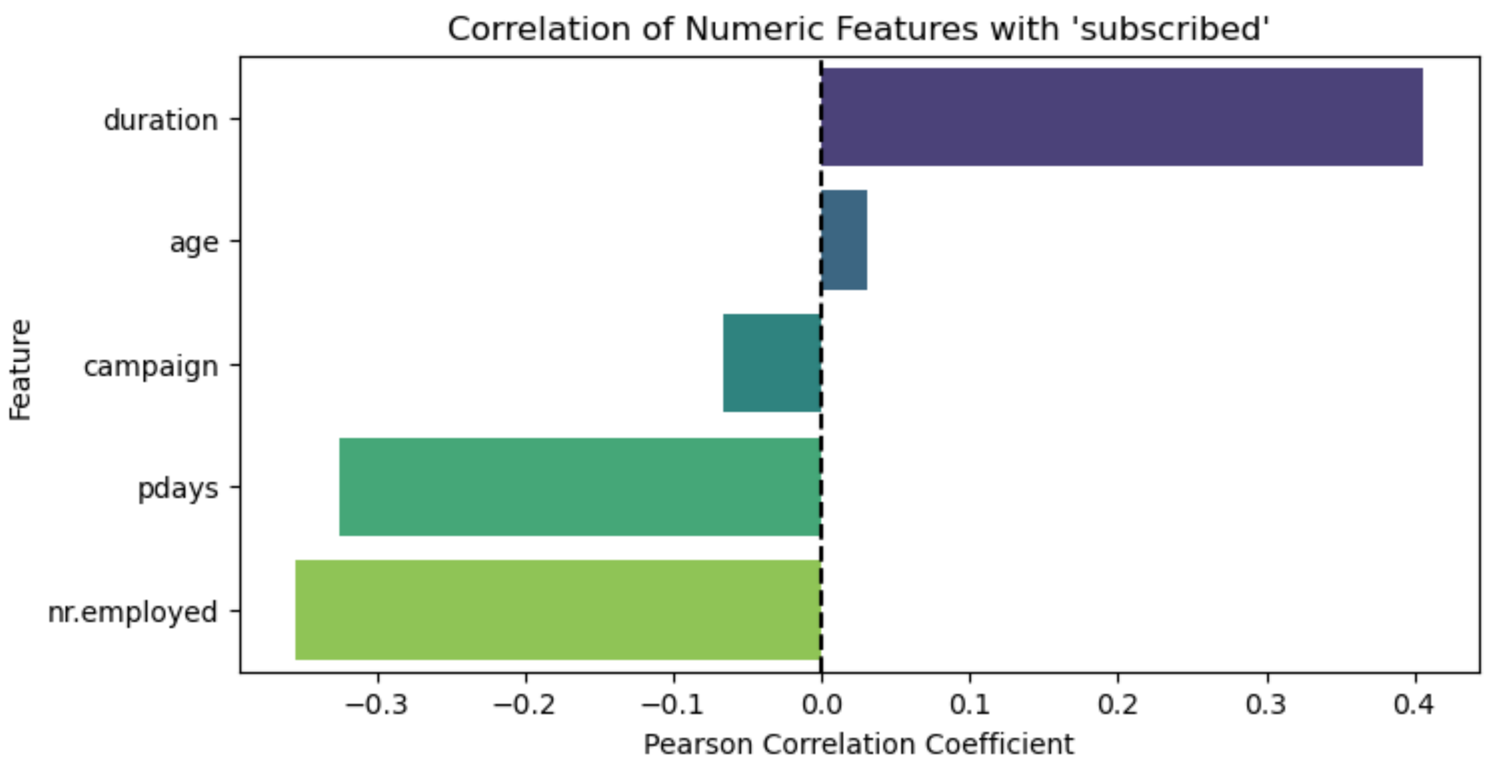



```
=== Final Test Results ===

The models have been evaluated on the test set. The performance metrics are summarized above.
The best performing model based on F1 Score is: MLP
```

In [21]:
```
# Calculate correlations for numeric features vs. target
numeric_data = data[numerical_features + ["Subscribed"]]
numeric_data["Subscribed"] = numeric_data["Subscribed"].map({"no": 0, "yes": 1})  # Encode target

correlation_matrix = numeric_data.corr(method="pearson")
target_correlations = correlation_matrix["Subscribed"].drop("Subscribed").sort_values(ascending=False)

# Plot correlation coefficients
plt.figure(figsize=(8, 4))
sns.barplot(x=target_correlations.values, y=target_correlations.index, palette="viridis")
```

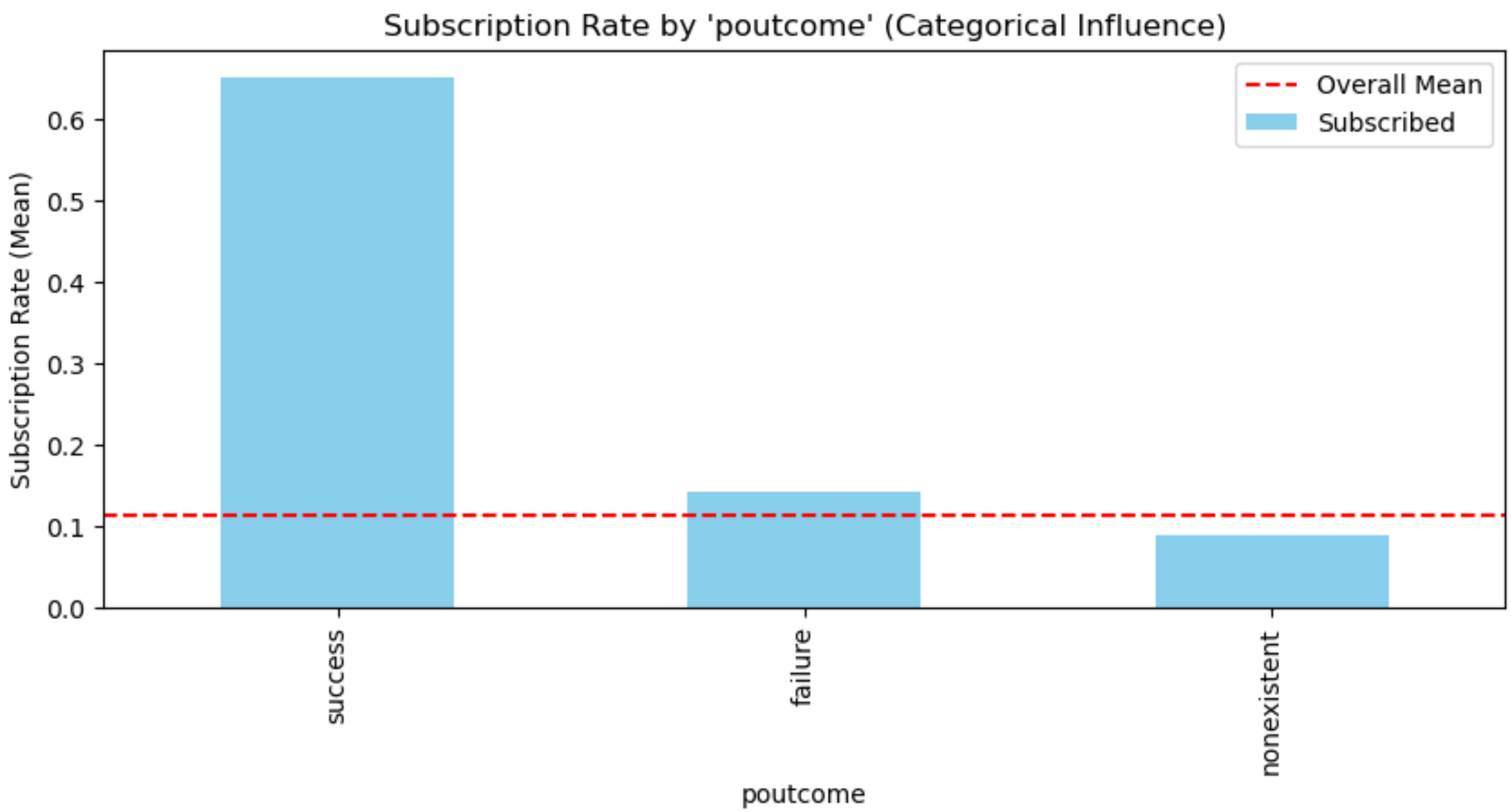
```
plt.title("Correlation of Numeric Features with 'subscribed'")
plt.xlabel("Pearson Correlation Coefficient")
plt.ylabel("Feature")
plt.axvline(0, color="k", linestyle="--")
plt.show()
```

```
C:\Users\Tom\AppData\Local\Temp\ipykernel_28516\3013124865.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returnin
g-a-view-versus-a-copy
  numeric_data["Subscribed"] = numeric_data["Subscribed"].map({"no": 0, "yes": 1})  # Encode target
C:\Users\Tom\AppData\Local\Temp\ipykernel_28516\3013124865.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.barplot(x=target_correlations.values, y=target_correlations.index, palette="viridis")
```



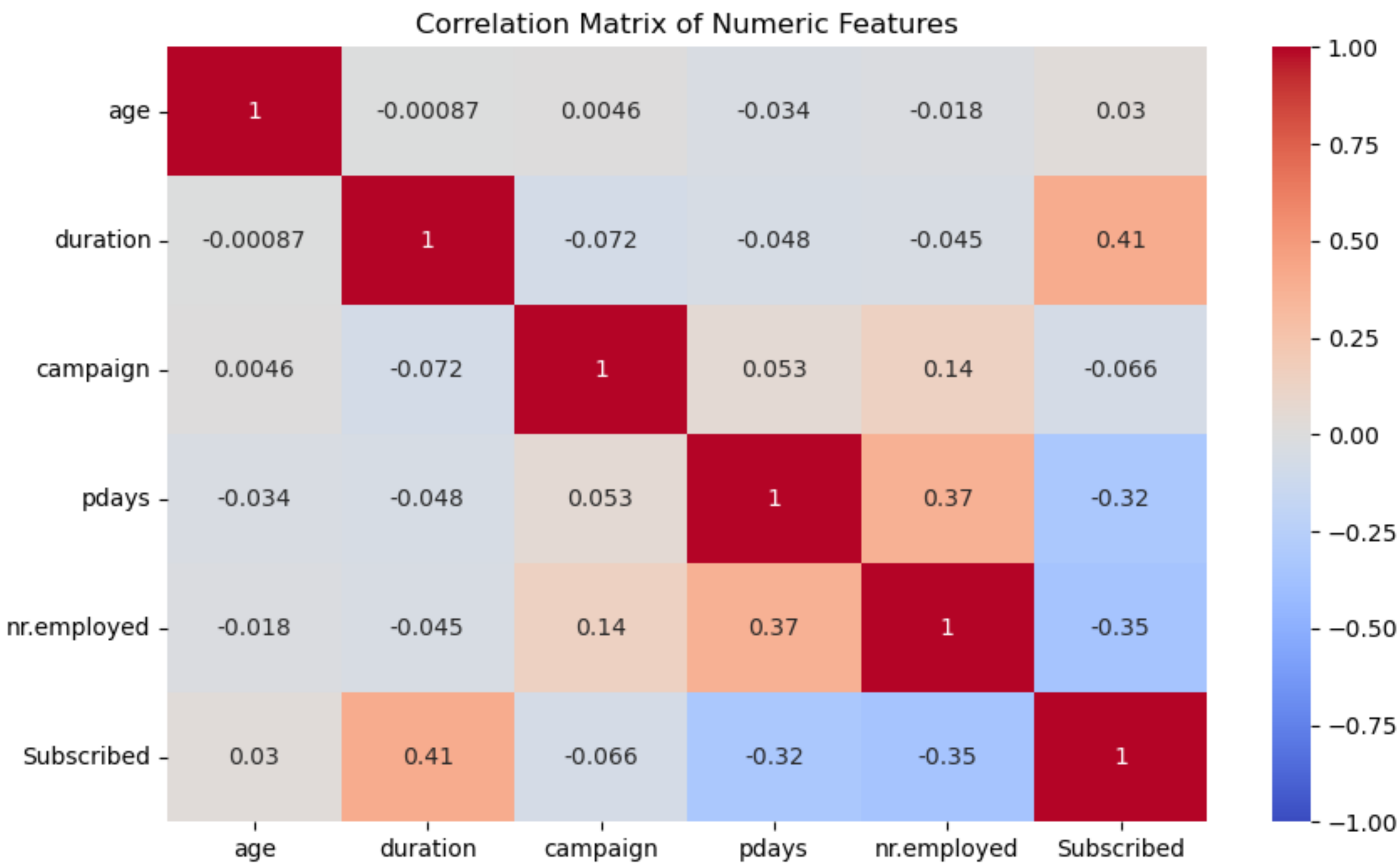
In [23]:
```
# Target-encode categorical features for correlation-like analysis
categorical_data = data[categorical_features + ["Subscribed"]]
categorical_data["Subscribed"] = categorical_data["Subscribed"].map({"no": 0, "yes": 1})

# Calculate mean subscription rate per category (proxy for correlation)
mean_effects = {}
for col in categorical_features:
    mean_effects[col] = categorical_data.groupby(col)["Subscribed"].mean().sort_values(ascending=False)

# Plot top influential categories (e.g., for 'poutcome')
plt.figure(figsize=(10, 4))
mean_effects["poutcome"].plot(kind="bar", color="skyblue")
plt.title("Subscription Rate by 'poutcome' (Categorical Influence)")
plt.ylabel("Subscription Rate (Mean)")
plt.axhline(y=categorical_data["Subscribed"].mean(), color="r", linestyle="--", label="Overall Mean")
plt.legend()
plt.show()
```

```
C:\Users\Tom\AppData\Local\Temp\ipykernel_28516\901812262.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returnin
g-a-view-versus-a-copy
  categorical_data["Subscribed"] = categorical_data["Subscribed"].map({"no": 0, "yes": 1})
```

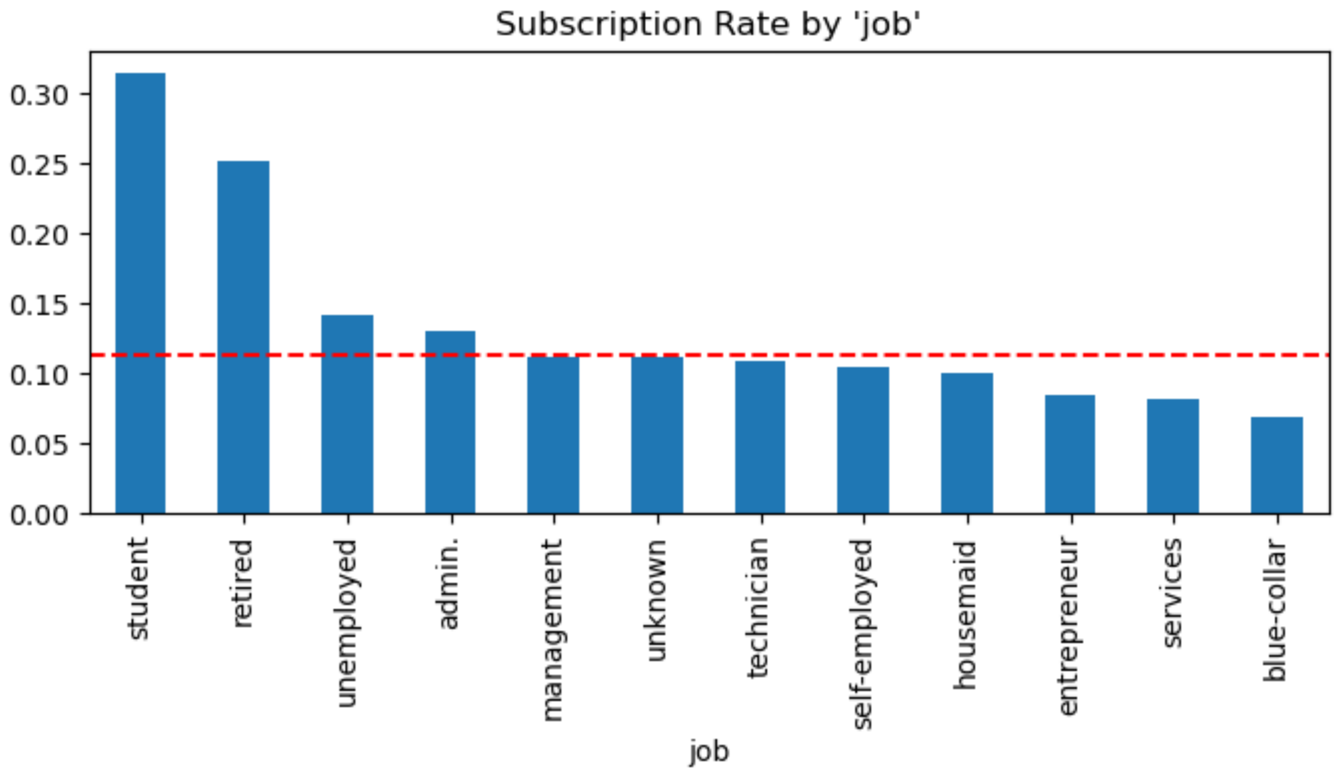


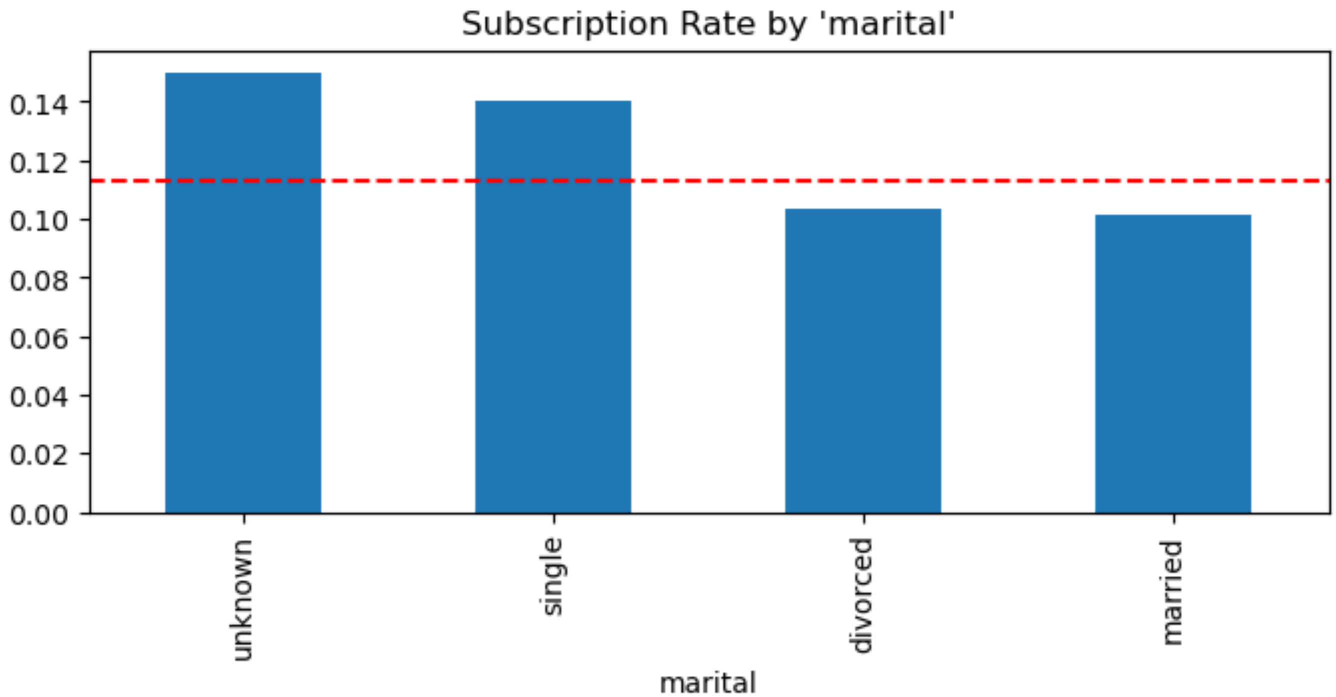
In [25]:

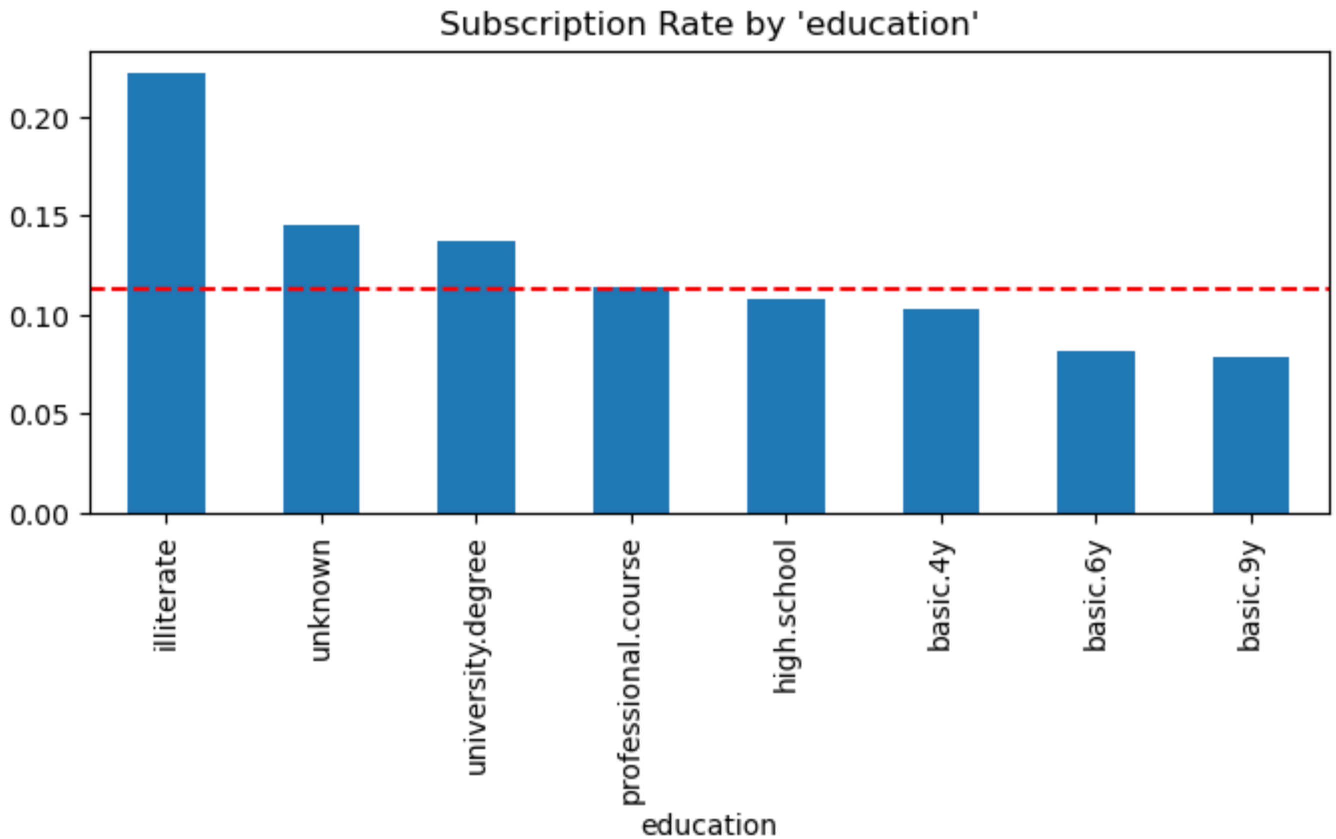
```
# Plot full correlation matrix (numeric features only)
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", vmin=-1, vmax=1, center=0)
plt.title("Correlation Matrix of Numeric Features")
plt.show()
```

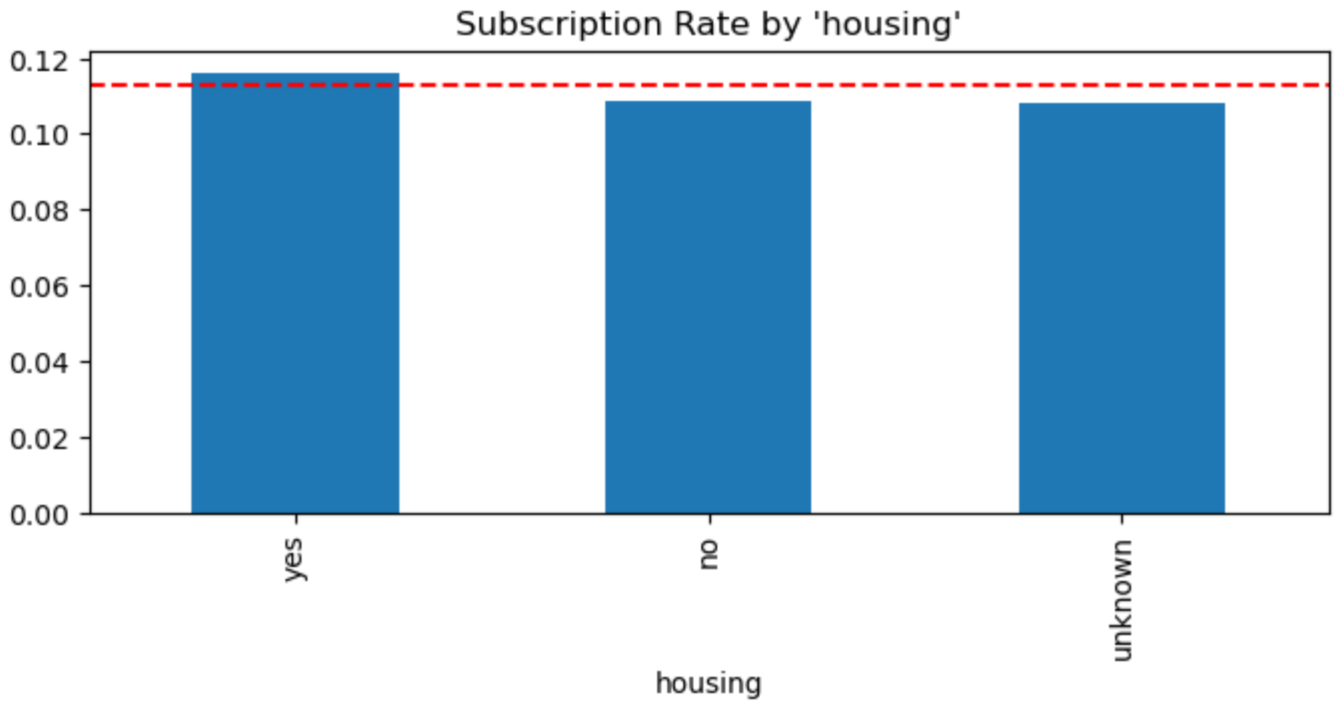


In [27]:
```
# Generate plots for all categorical variables
for col in categorical_features:
    plt.figure(figsize=(8, 3))
    mean_effects[col].plot(kind="bar", title=f"Subscription Rate by '{col}'")
    plt.axhline(y=categorical_data["Subscribed"].mean(), color="r", linestyle="--")
    plt.show()
```


Subscription Rate by 'job'
0.30
0.25
0.20
0.15
0.10
0.05
0.00
student
retired
unemployed
admin.
management
unknown
technician
self-employed
housemaid
entrepreneur
services
blue-collar
job

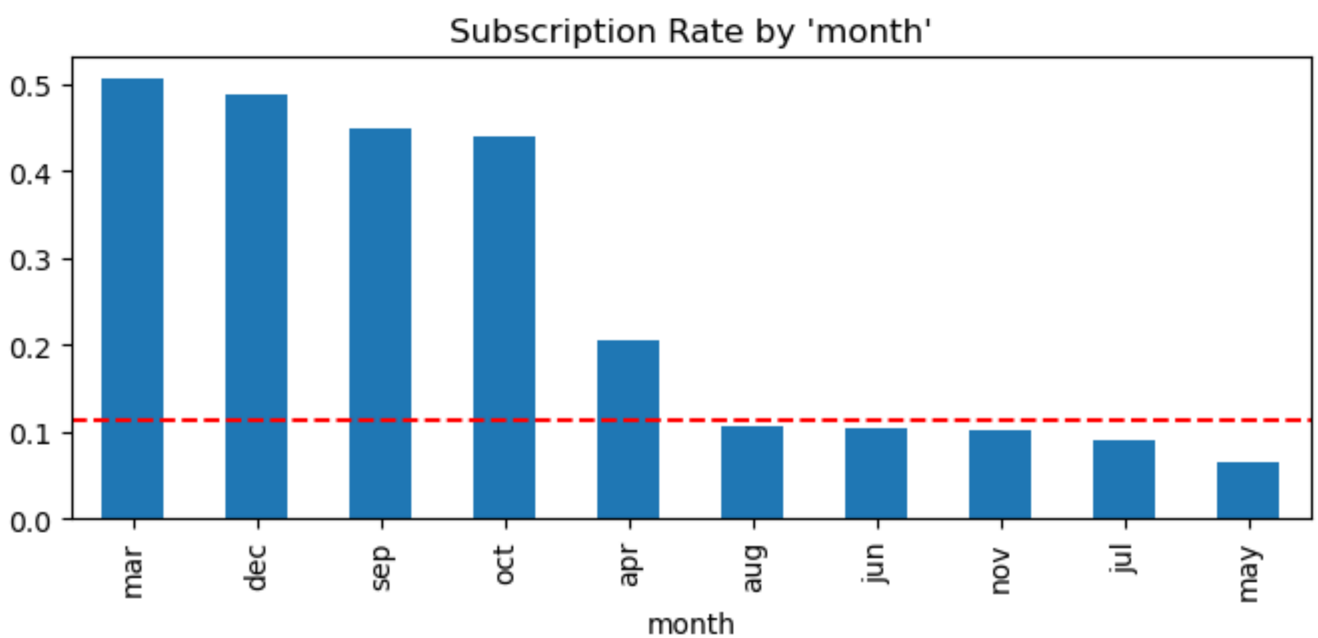

Subscription Rate by 'marital'
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
unknown
single
divorced
married
marital

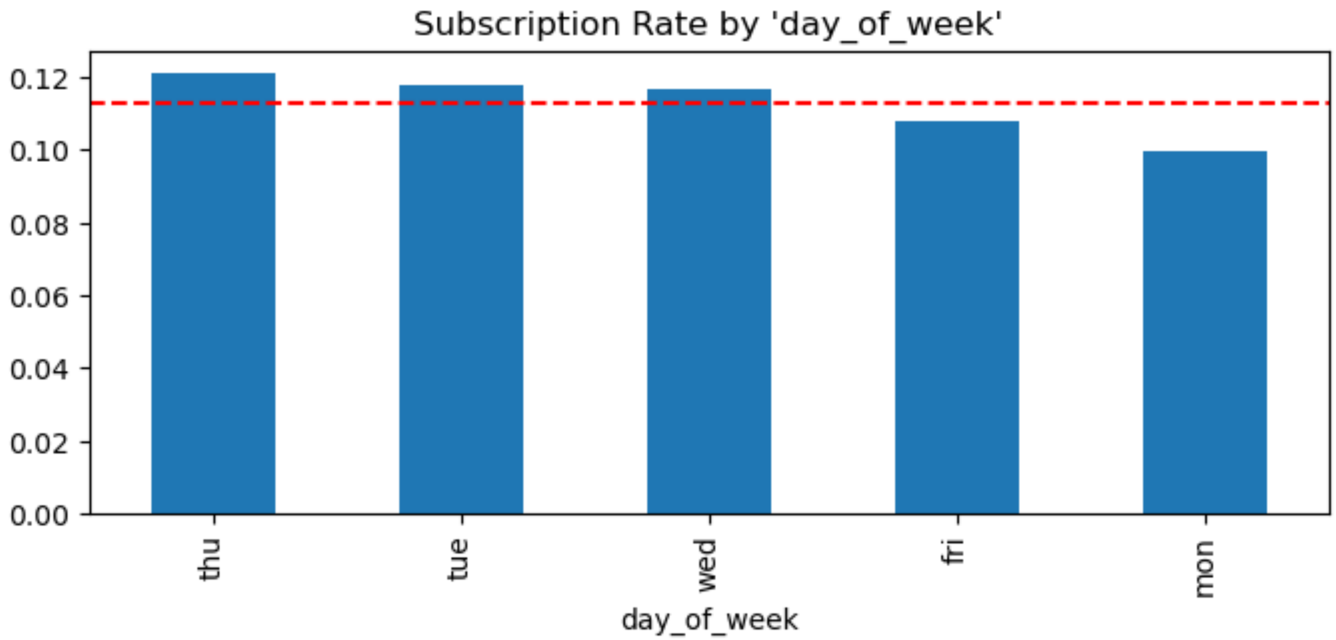

Subscription Rate by 'education'
0.20
0.15
0.10
0.05
0.00
illiterate
unknown
university.degree
professional.course
high.school
basic.4y
basic.6y
basic.9y
education

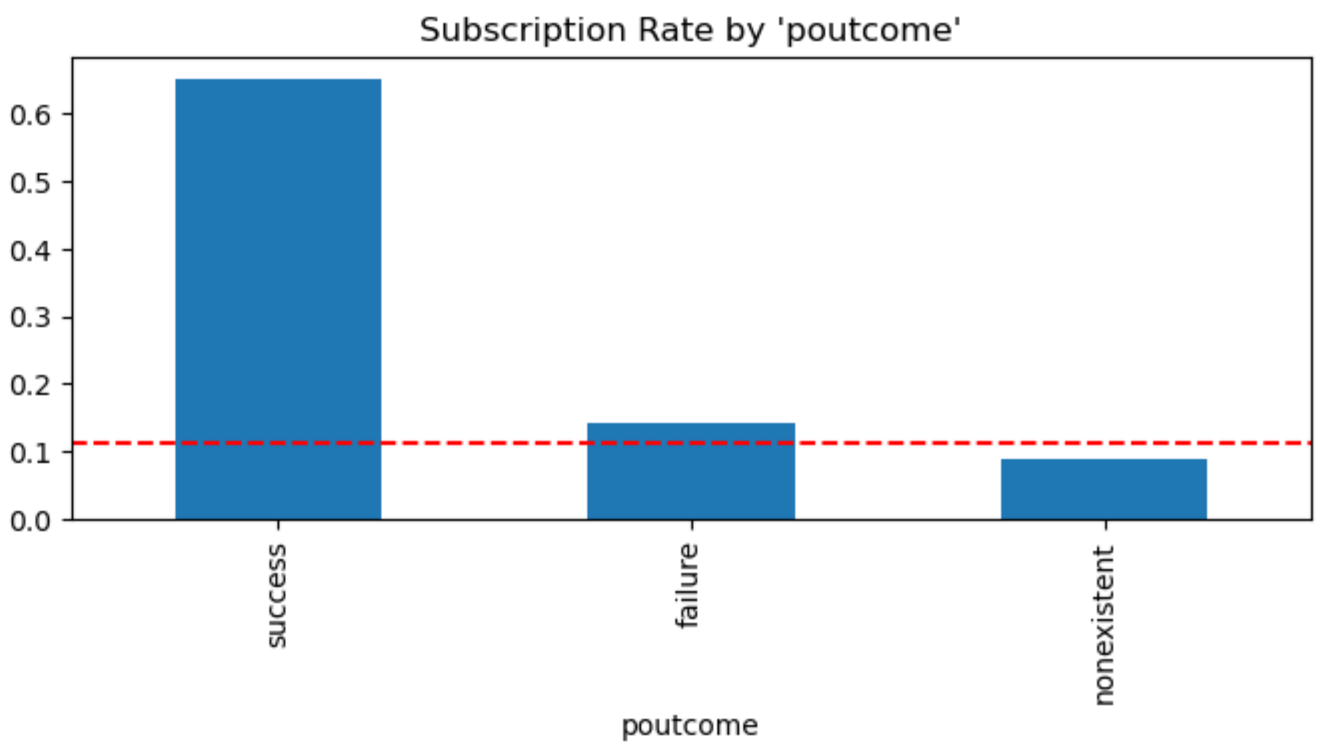

Subscription Rate by 'housing'
0.12
0.10
0.08
0.06
0.04
0.02
0.00
yes
no
unknown
housing


Subscription Rate by 'loan'
0.10
0.08
0.06
0.04
0.02
0.00
no
yes
unknown
loan


Subscription Rate by 'contact'
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
cellular
telephone
contact


Subscription Rate by 'month'
0.5
0.4
0.3
0.2
0.1
0.0
mar
dec
sep
oct
apr
aug
jun
nov
jul
may
month


Subscription Rate by 'day_of_week'
0.12
0.10
0.08
0.06
0.04
0.02
0.00
thu
tue
wed
fri
mon
day_of_week


Subscription Rate by 'poutcome'
0.6
0.5
0.4
0.3
0.2
0.1
0.0
success
failure
nonexistent
poutcome

In [1]:
```
# MLP, CNN and a Linear enhanced model for comparison

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.utils import class_weight

# Deep learning libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Load datasets (replace with your actual file paths)
try:
    train_data = pd.read_csv('trainset.csv')
    test_data = pd.read_csv('testset.csv')
except FileNotFoundError:
    print("Please make sure the files 'trainset.csv' and 'testset.csv' are in the correct directory.")
    raise

# Step 1: Data Exploration
print("\n=== Data Exploration ===\n")
print("Training set shape:", train_data.shape)
print("Test set shape:", test_data.shape)
print("\nTraining set class distribution:")
print(train_data['Subscribed'].value_counts())
print("\nTest set class distribution:")
print(test_data['Subscribed'].value_counts())
```

```
# Visualize class distribution
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
train_data['Subscribed'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Training Set Class Distribution')
plt.subplot(1, 2, 2)
test_data['Subscribed'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Test Set Class Distribution')
plt.tight_layout()
plt.show()

# Explore categorical features
categorical_features = ['job', 'marital', 'education', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcom
plt.figure(figsize=(20, 25))
for i, feature in enumerate(categorical_features, 1):
    plt.subplot(5, 2, i)
    sns.countplot(x=feature, hue='Subscribed', data=train_data, palette='viridis')
    plt.title(f'{feature} Distribution')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Explore numerical features
numerical_features = ['age', 'duration', 'campaign', 'pdays', 'nr.employed']
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
    plt.title(f'{feature} Distribution')
plt.tight_layout()
plt.show()

# Step 2: Data Preprocessing
print("\n=== Data Preprocessing ===\n")

# Combine train and test for consistent preprocessing
combined = pd.concat([train_data, test_data], axis=0)

# Handle 'unknown' values - replace with mode or appropriate value
for column in categorical_features:
    mode_val = combined[column].mode()[0]
    combined[column] = combined[column].replace('unknown', mode_val)
```

```
# Convert pdays=999 to -1 (indicator for not previously contacted)
combined['pdays'] = combined['pdays'].replace(999, -1)

# Encode categorical variables
label_encoders = {}
for column in categorical_features:
    le = LabelEncoder()
    combined[column] = le.fit_transform(combined[column])
    label_encoders[column] = le

# Encode target variable
target_encoder = LabelEncoder()
combined['Subscribed'] = target_encoder.fit_transform(combined['Subscribed'])

# Split back into train and test
train_data = combined.iloc[:len(train_data)]
test_data = combined.iloc[len(train_data):]

# Separate features and target
X_train = train_data.drop('Subscribed', axis=1)
y_train = train_data['Subscribed']
X_test = test_data.drop('Subscribed', axis=1)
y_test = test_data['Subscribed']

# Scale numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Handle class imbalance
class_weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights = dict(enumerate(class_weights))

# Reshape data for CNN
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)

# Step 3: Model Development and Training
print("\n=== Model Development and Training ===\n")

# MLP Model
```

```
def build_mlp():
    model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

mlp_model = build_mlp()
print("MLP Model Summary:")
mlp_model.summary()

# CNN Model
def build_cnn():
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_reshaped.shape[1], 1)),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

cnn_model = build_cnn()
print("\nCNN Model Summary:")
cnn_model.summary()

# Linear Model (a simple model that's generally not good for this type of data)
def build_linear():
    model = Sequential([
        Dense(1, activation='sigmoid', input_shape=(X_train_scaled.shape[1],))
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
```

```
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

linear_model = build_linear()
print("\nLinear Model Summary:")
linear_model.summary()

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train MLP
print("\nTraining MLP Model...")
mlp_history = mlp_model.fit(X_train_scaled, y_train,
                           validation_split=0.2,
                           epochs=50,
                           batch_size=64,
                           class_weight=class_weights,
                           callbacks=[early_stopping],
                           verbose=1)

# Train CNN
print("\nTraining CNN Model...")
cnn_history = cnn_model.fit(X_train_reshaped, y_train,
                           validation_split=0.2,
                           epochs=50,
                           batch_size=64,
                           class_weight=class_weights,
                           callbacks=[early_stopping],
                           verbose=1)

# Train Linear
print("\nTraining Linear Model...")
linear_history = linear_model.fit(X_train_scaled, y_train,
                                 validation_split=0.2,
                                 epochs=50,
                                 batch_size=64,
                                 class_weight=class_weights,
                                 callbacks=[early_stopping],
                                 verbose=1)

# Step 4: Model Evaluation and Comparison
```

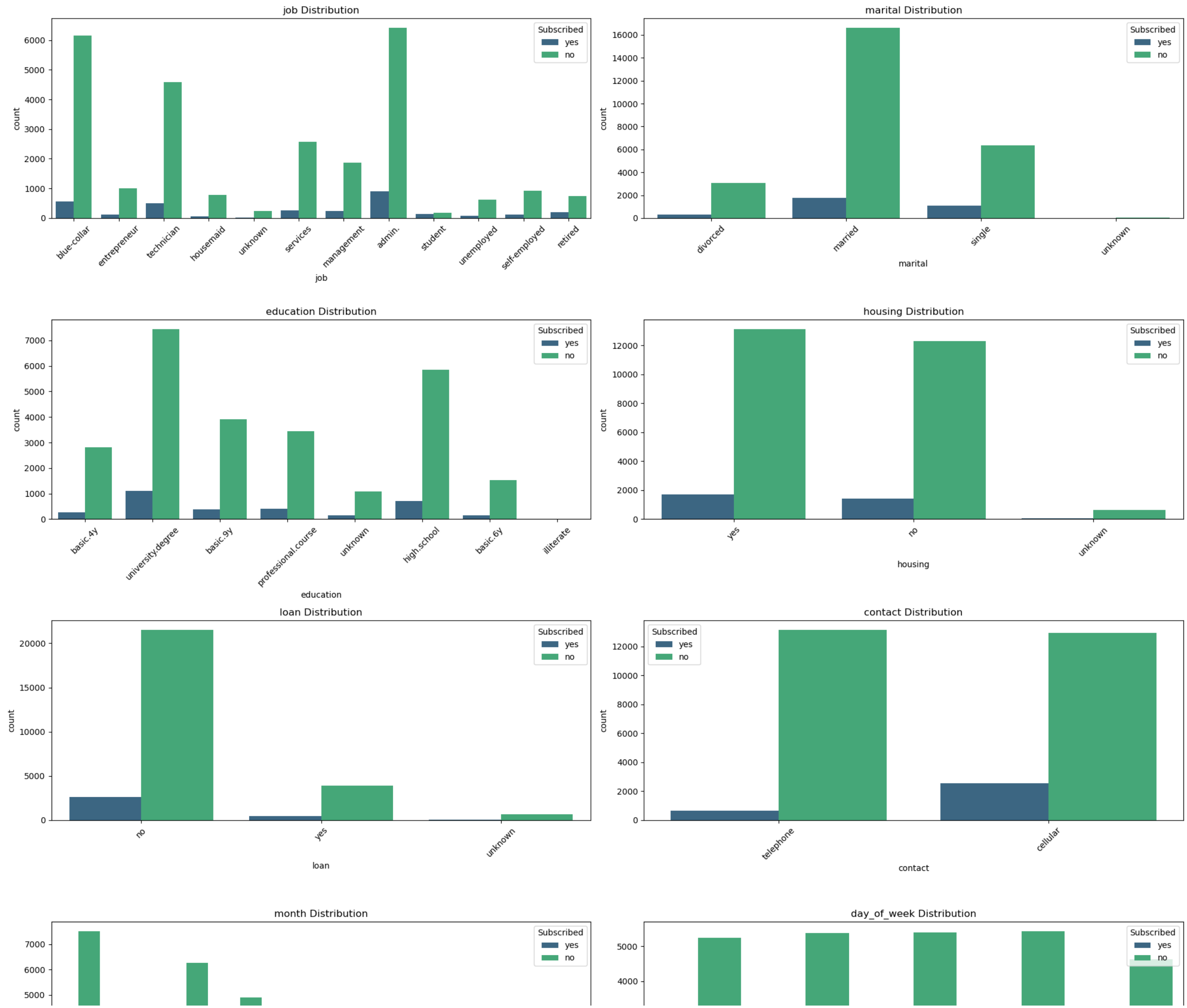
```
print("\n=== Model Evaluation and Comparison ===\n")

def evaluate_model(model, X_test, y_test, model_name):
    y_pred_prob = model.predict(X_test)
    y_pred = (y_pred_prob > 0.5).astype(int)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"{model_name} Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    # Plot confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['No', 'Yes'],
                yticklabels=['No', 'Yes'])
    plt.title(f'{model_name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    return accuracy, precision, recall, f1

# Evaluate MLP
mlp_metrics = evaluate_model(mlp_model, X_test_scaled, y_test, 'MLP')

# Evaluate CNN
cnn_metrics = evaluate_model(cnn_model, X_test_reshaped, y_test, 'CNN')

# Evaluate Linear
linear_metrics = evaluate_model(linear_model, X_test_scaled, y_test, 'Linear')

# Plot training history for all models
def plot_history(history, model_name):
    plt.figure(figsize=(12, 4))
```

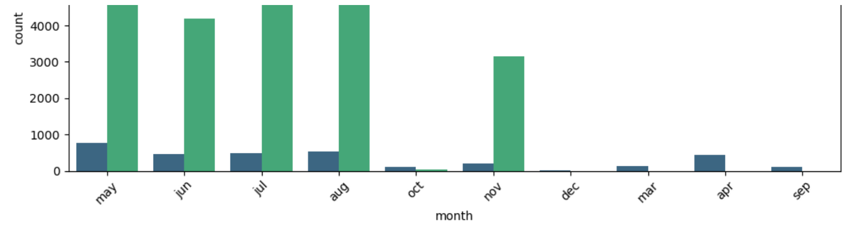
```
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.tight_layout()
    plt.show()

plot_history(mlp_history, 'MLP')
plot_history(cnn_history, 'CNN')
plot_history(linear_history, 'Linear')

# Compare all models' performance
metrics_df = pd.DataFrame({
    'Model': ['MLP', 'CNN', 'Linear'],
    'Accuracy': [mlp_metrics[0], cnn_metrics[0], linear_metrics[0]],
    'Precision': [mlp_metrics[1], cnn_metrics[1], linear_metrics[1]],
    'Recall': [mlp_metrics[2], cnn_metrics[2], linear_metrics[2]],
    'F1 Score': [mlp_metrics[3], cnn_metrics[3], linear_metrics[3]]
})

print("\n=== Model Performance Comparison ===")
print(metrics_df)

# Visual comparison of model metrics
plt.figure(figsize=(15, 8))
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
for i, metric in enumerate(metrics, 1):
    plt.subplot(2, 2, i)
    sns.barplot(x='Model', y=metric, data=metrics_df)
    plt.title(metric)
    plt.ylim(0, 1)
```

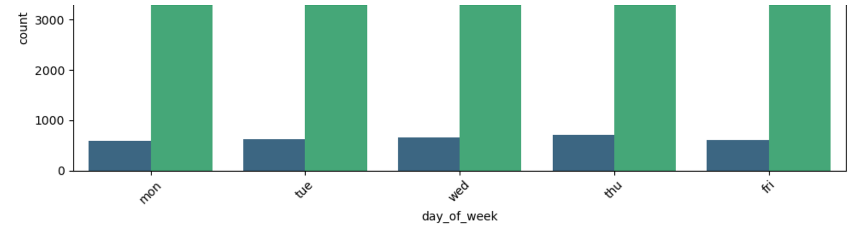
```python
plt.tight_layout()
plt.show()

# Step 5: Final Test Results
print("\n=== Final Test Results ===\n")
print("The models have been evaluated on the test set. The performance metrics are summarized above.")
print("The best performing model based on F1 Score is:", metrics_df.loc[metrics_df['F1 Score'].idxmax(), 'Model'])
```

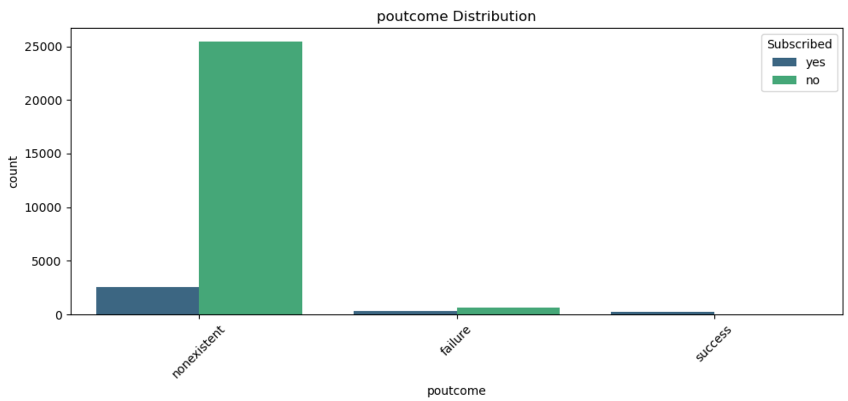
```
=== Data Exploration ===

Training set shape: (29271, 15)
Test set shape: (11917, 15)

Training set class distribution:
Subscribed
no     26075
yes     3196
Name: count, dtype: int64

Test set class distribution:
Subscribed
no     10473
yes     1444
Name: count, dtype: int64
```


Training Set Class Distribution
25000
20000
15000
10000
5000
0
no
yes
Subscribed
Test Set Class Distribution
10000
8000
6000
4000
2000
0
no
yes
Subscribed


job Distribution
Subscribed
yes
no
count
job
blue-collar
entrepreneur
technician
housemaid
unknown
services
management
admin.
student
unemployed
self-employed
retired
marital Distribution
marital
divorced
married
single
unknown
education Distribution
education
basic.4y
university.degree
basic.9y
professional.course
unknown
high.school
basic.6y
illiterate
housing Distribution
housing
yes
no
unknown
loan Distribution
loan
no
yes
unknown
contact Distribution
contact
telephone
cellular
month Distribution
day_of_week Distribution

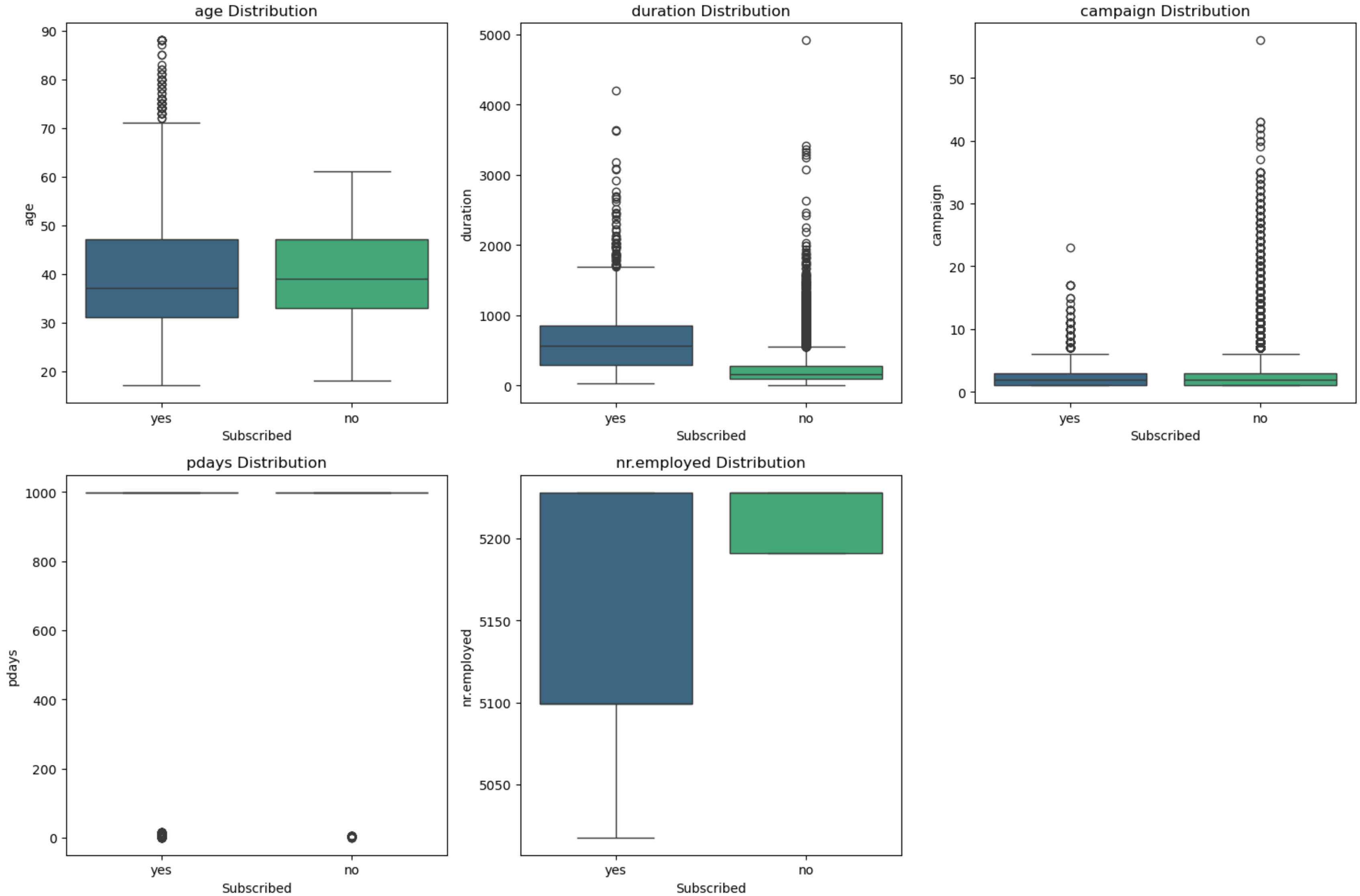







```
C:\Users\Tom\AppData\Local\Temp\ipykernel_11852\2599279041.py:69: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
C:\Users\Tom\AppData\Local\Temp\ipykernel_11852\2599279041.py:69: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
C:\Users\Tom\AppData\Local\Temp\ipykernel_11852\2599279041.py:69: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
C:\Users\Tom\AppData\Local\Temp\ipykernel_11852\2599279041.py:69: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
```

```
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
C:\Users\Tom\AppData\Local\Temp\ipykernel_11852\2599279041.py:69: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x='Subscribed', y=feature, data=train_data, palette='viridis')
```




```
=== Data Preprocessing ===


=== Model Development and Training ===
```

```
C:\Users\Tom\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape
`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first l
ayer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
MLP Model Summary:
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	960
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 1)	33

Total params: 3,073 (12.00 KB)

Trainable params: 3,073 (12.00 KB)

Non-trainable params: 0 (0.00 B)

```
CNN Model Summary:
```

```
C:\Users\Tom\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 12, 64)	256
max_pooling1d (MaxPooling1D)	(None, 6, 64)	0
flatten (Flatten)	(None, 384)	0
dense_3 (Dense)	(None, 32)	12,320
dropout_2 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 1)	33

Total params: 12,609 (49.25 KB)

Trainable params: 12,609 (49.25 KB)

Non-trainable params: 0 (0.00 B)

Linear Model Summary:

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 1)	15

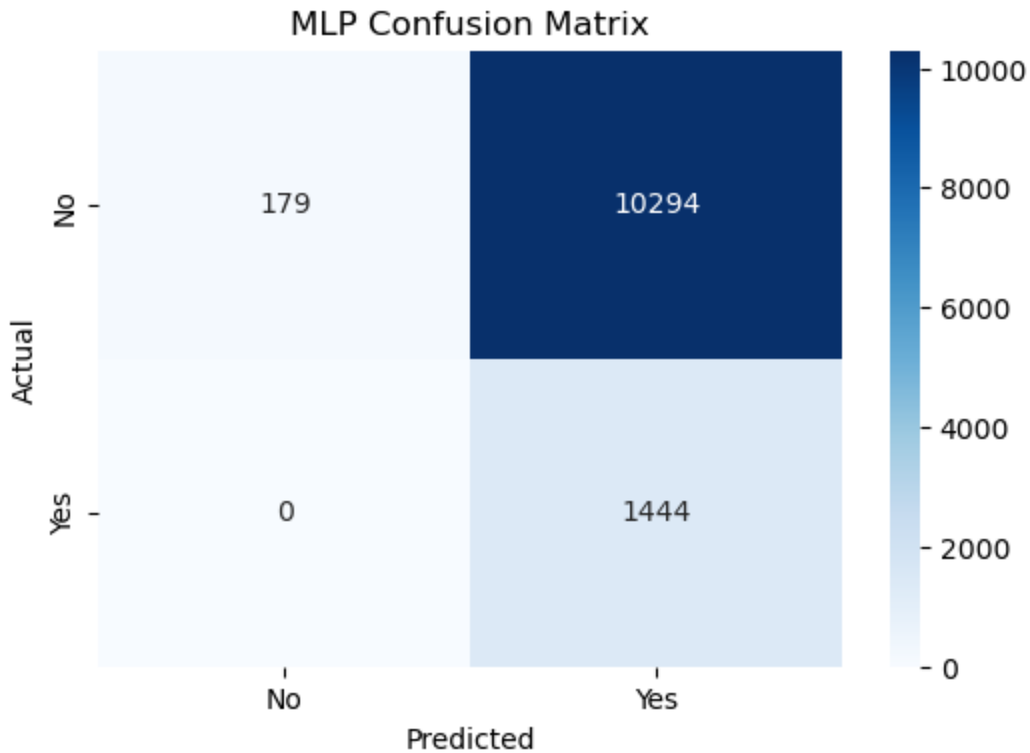
Total params: 15 (60.00 B)

Trainable params: 15 (60.00 B)

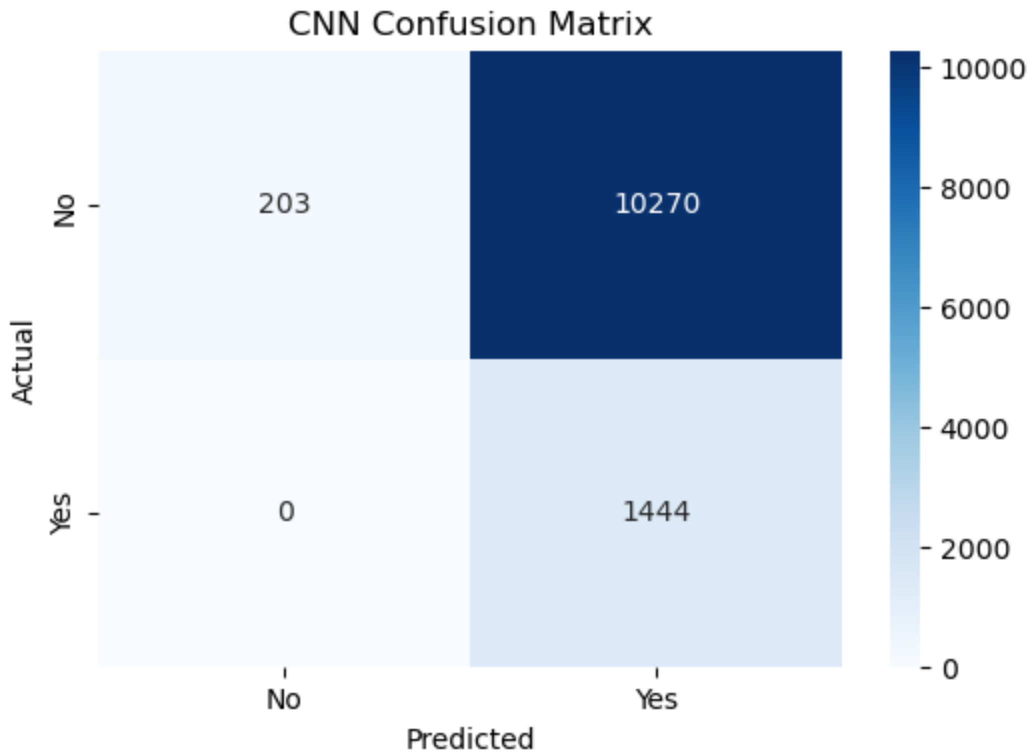
Non-trainable params: 0 (0.00 B)

```
Training MLP Model...
Epoch 1/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.7760 - loss: 0.4491 - val_accuracy: 0.7689 - val_loss: 0.5842
Epoch 2/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9131 - loss: 0.2270 - val_accuracy: 0.7209 - val_loss: 0.8300
Epoch 3/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9133 - loss: 0.2022 - val_accuracy: 0.6555 - val_loss: 1.1295
Epoch 4/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9162 - loss: 0.1870 - val_accuracy: 0.5221 - val_loss: 1.6430
Epoch 5/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9212 - loss: 0.1763 - val_accuracy: 0.4639 - val_loss: 2.6601
Epoch 6/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9221 - loss: 0.1710 - val_accuracy: 0.4576 - val_loss: 3.5997

Training CNN Model...
Epoch 1/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8303 - loss: 0.3637 - val_accuracy: 0.8029 - val_loss: 1.0018
Epoch 2/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9247 - loss: 0.2087 - val_accuracy: 0.7740 - val_loss: 1.2481
Epoch 3/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9247 - loss: 0.1915 - val_accuracy: 0.6319 - val_loss: 1.5834
Epoch 4/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9230 - loss: 0.1773 - val_accuracy: 0.5255 - val_loss: 1.9201
Epoch 5/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9241 - loss: 0.1718 - val_accuracy: 0.4864 - val_loss: 2.2493
Epoch 6/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9235 - loss: 0.1648 - val_accuracy: 0.4675 - val_loss: 2.5130

Training Linear Model...
Epoch 1/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6206 - loss: 0.7897 - val_accuracy: 0.1851 - val_loss: 1.0380
Epoch 2/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6812 - loss: 0.4970 - val_accuracy: 0.2789 - val_loss: 0.8548
Epoch 3/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7867 - loss: 0.3875 - val_accuracy: 0.5658 - val_loss: 0.7259
Epoch 4/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8897 - loss: 0.3319 - val_accuracy: 0.6941 - val_loss: 0.6467
Epoch 5/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9122 - loss: 0.2986 - val_accuracy: 0.7534 - val_loss: 0.5945
Epoch 6/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9210 - loss: 0.2769 - val_accuracy: 0.7855 - val_loss: 0.5578
Epoch 7/50
```

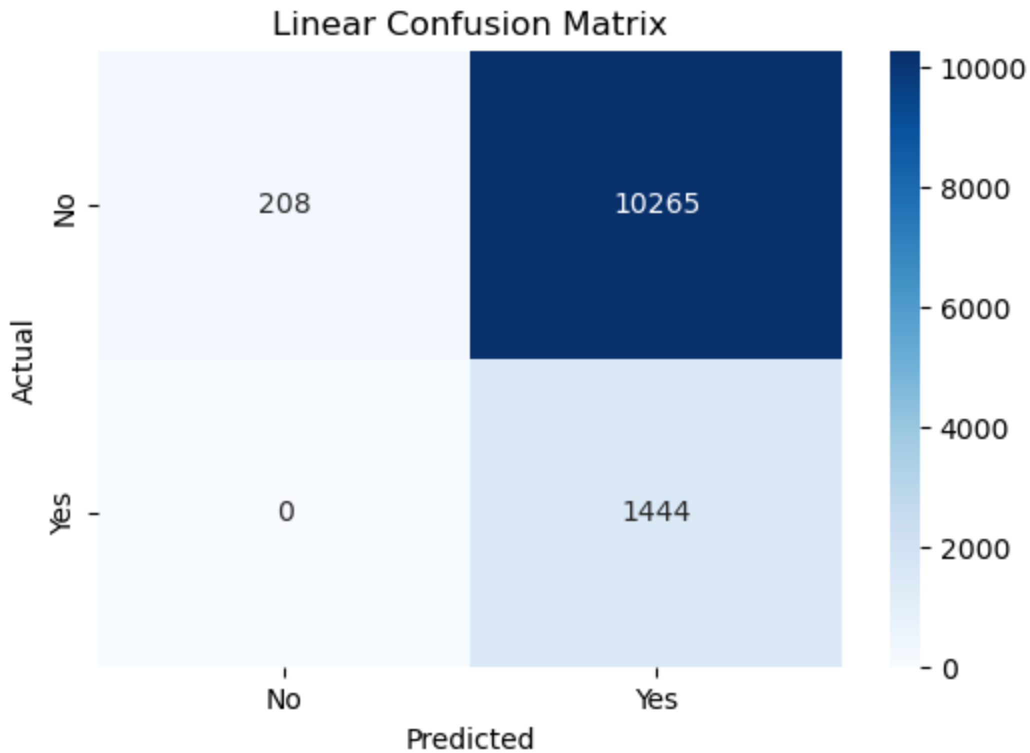
```
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9253 - loss: 0.2623 - val_accuracy: 0.8046 - val_loss: 0.5311
Epoch 8/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9282 - loss: 0.2521 - val_accuracy: 0.8162 - val_loss: 0.5114
Epoch 9/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9286 - loss: 0.2450 - val_accuracy: 0.8256 - val_loss: 0.4968
Epoch 10/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9297 - loss: 0.2399 - val_accuracy: 0.8292 - val_loss: 0.4862
Epoch 11/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9301 - loss: 0.2364 - val_accuracy: 0.8331 - val_loss: 0.4786
Epoch 12/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9305 - loss: 0.2339 - val_accuracy: 0.8359 - val_loss: 0.4733
Epoch 13/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9308 - loss: 0.2321 - val_accuracy: 0.8374 - val_loss: 0.4700
Epoch 14/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9311 - loss: 0.2308 - val_accuracy: 0.8372 - val_loss: 0.4680
Epoch 15/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9313 - loss: 0.2299 - val_accuracy: 0.8362 - val_loss: 0.4673
Epoch 16/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9313 - loss: 0.2293 - val_accuracy: 0.8360 - val_loss: 0.4676
Epoch 17/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9312 - loss: 0.2289 - val_accuracy: 0.8331 - val_loss: 0.4686
Epoch 18/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9311 - loss: 0.2285 - val_accuracy: 0.8307 - val_loss: 0.4703
Epoch 19/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9311 - loss: 0.2283 - val_accuracy: 0.8261 - val_loss: 0.4727
Epoch 20/50
366/366 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9311 - loss: 0.2281 - val_accuracy: 0.8210 - val_loss: 0.4756

=== Model Evaluation and Comparison ===

373/373 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
MLP Performance:
Accuracy: 0.1362
Precision: 0.1230
Recall: 1.0000
F1 Score: 0.2191
```

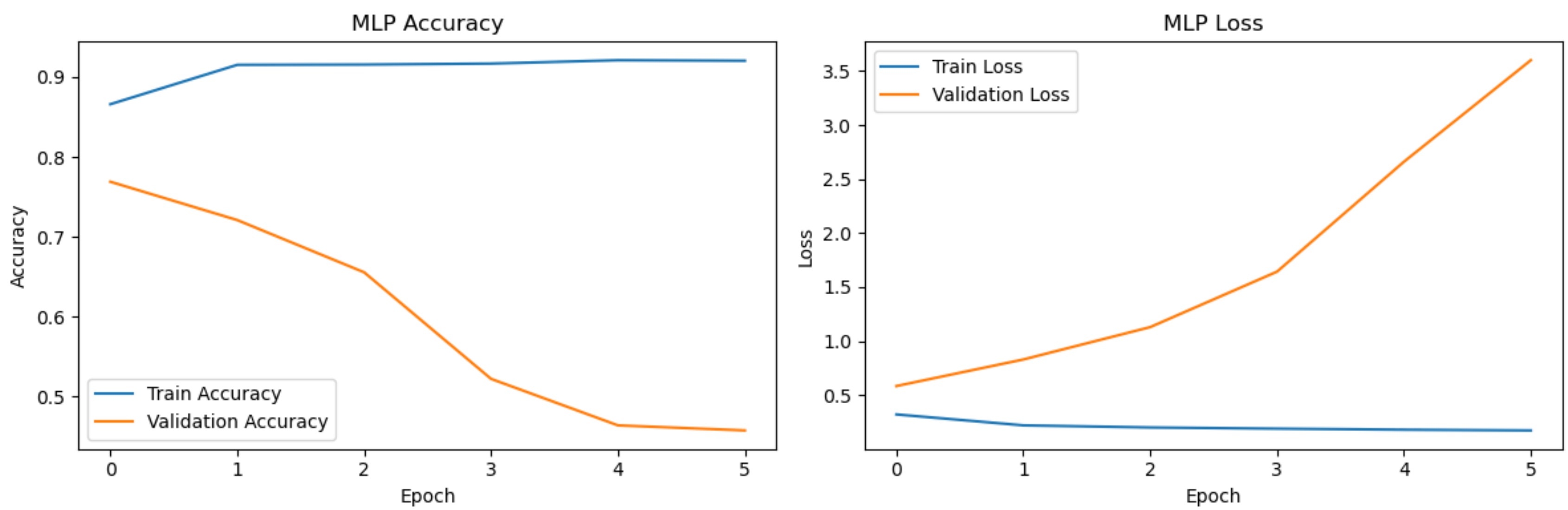


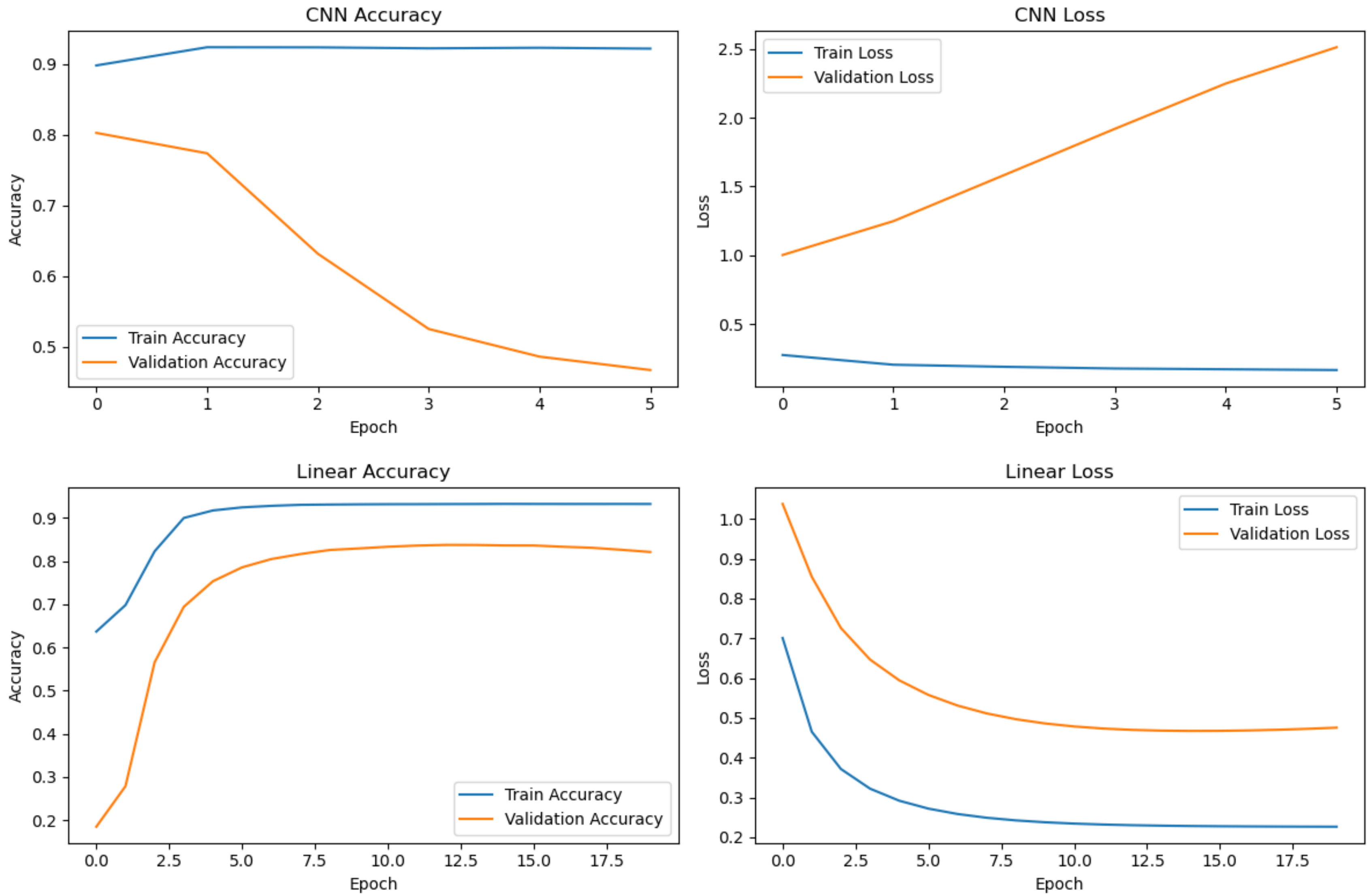

```
373/373 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
CNN Performance:
Accuracy: 0.1382
Precision: 0.1233
Recall: 1.0000
F1 Score: 0.2195
```



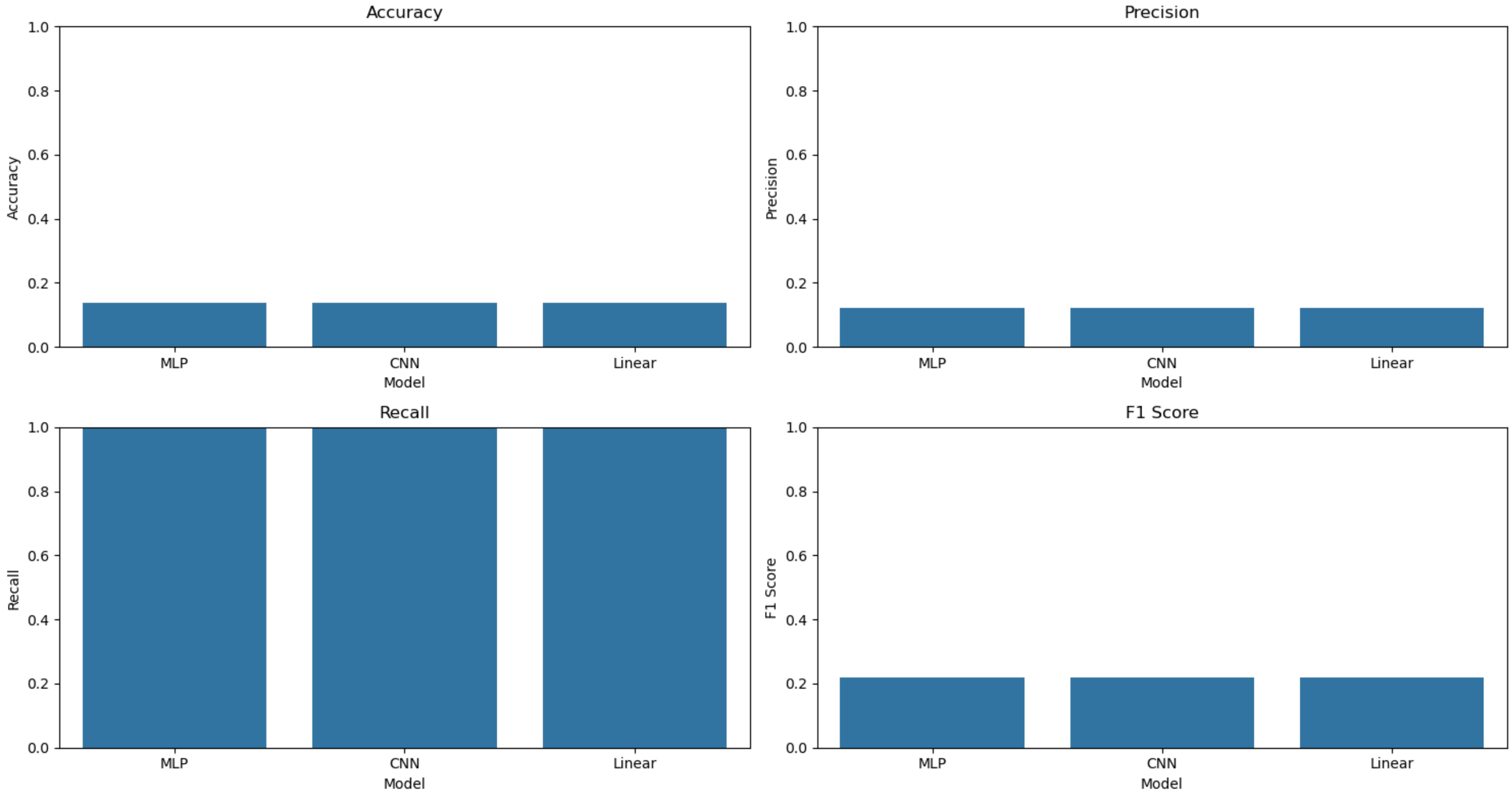

```
373/373 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Linear Performance:
Accuracy: 0.1386
Precision: 0.1233
Recall: 1.0000
F1 Score: 0.2196
```


Linear Confusion Matrix
Actual
No
Yes
208
10265
0
1444
No
Yes
Predicted
10000
8000
6000
4000
2000
0


MLP Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy
MLP Loss
Loss
Epoch
Train Loss
Validation Loss




```
=== Model Performance Comparison ===
    Model  Accuracy  Precision  Recall  F1 Score
0     MLP  0.136192   0.123019     1.0  0.219087
1     CNN  0.138206   0.123271     1.0  0.219486
2  Linear  0.138625   0.123324     1.0  0.219570
```




```
=== Final Test Results ===

The models have been evaluated on the test set. The performance metrics are summarized above.
The best performing model based on F1 Score is: Linear
```

In [ ]:

In [ ]: