```python
import pandas as pd
import numpy as np

file_path = r"C:\Users\utkar\LFB Lithium and Electric Vehicle fire
data.xlsx"

df = pd.read_excel(file_path)

df.head()
```

```
    IncidentNumber  CalendarYear MonthText                   Type  \
0  000285-01012017          2017       Jan  Other Lithium Battery
1  000641-02012017          2017       Jan  Other Lithium Battery
2  005511-13012017          2017       Jan                    Car
3  006064-15012017          2017       Jan                    Car
4  007635-18012017          2017       Jan              e-scooter

  IgnitionSourcePower                IgnitionSource  \
0         Electricity  Other appliance or equipment
1             Unknown                     Not known
2         Electricity         Batteries, generators
3         Electricity         Batteries, generators
4         Electricity         Batteries, generators

              ItemFirstIgnited VehiclePowerType VehicleManufacturer  \
0                   Other item              NaN                 NaN
1  Plastic - raw material only              NaN                 NaN
2   Rubber - raw material only              NaN            Mercedes
3  Plastic - raw material only              NaN            Mercedes
4            Wiring insulation              NaN                 NaN

       IncGeo_BoroughName                         IncGeo_WardName
IncGeo_WardCode  \
0    BARKING AND DAGENHAM                             Thames View
E05014068
1                HOUNSLOW                 Osterley & Spring Grove
E05013626
2                  NEWHAM  Plaistow West & Canning Town East
E05013920
3  HAMMERSMITH AND FULHAM                             Ravenscourt
E05013746
4               ISLINGTON                                 Mildmay
E05013709

  Lithium batteries mentioned in report?
0                                    Yes
1                                    Yes
2                                     No
3                                     No
4                                     No
```

```
df.isnull().sum()

IncidentNumber                             0
CalendarYear                               0
MonthText                                  0
Type                                       0
IgnitionSourcePower                       12
IgnitionSource                            12
ItemFirstIgnited                          44
VehiclePowerType                        1047
VehicleManufacturer                      849
IncGeo_BoroughName                         0
IncGeo_WardName                            6
IncGeo_WardCode                            6
Lithium batteries mentioned in report?     0
dtype: int64

df.columns

Index(['IncidentNumber', 'CalendarYear', 'MonthText', 'Type',
       'IgnitionSourcePower', 'IgnitionSource', 'ItemFirstIgnited',
       'VehiclePowerType', 'VehicleManufacturer',
'IncGeo_BoroughName',
       'IncGeo_WardName', 'IncGeo_WardCode',
       'Lithium batteries mentioned in report?'],
      dtype='object')

df.drop(columns=[
    'IncidentNumber',
    'MonthText',
    'IncGeo_BoroughName',
    'IncGeo_WardName',
    'IncGeo_WardCode'
], inplace=True)

df.columns

Index(['CalendarYear', 'Type', 'IgnitionSourcePower',
'IgnitionSource',
       'ItemFirstIgnited', 'VehiclePowerType', 'VehicleManufacturer',
       'Lithium batteries mentioned in report?'],
      dtype='object')

df.isnull().sum()

CalendarYear                               0
Type                                       0
IgnitionSourcePower                       12
IgnitionSource                            12
ItemFirstIgnited                          44
VehiclePowerType                        1047
```

```
VehicleManufacturer                                849
Lithium batteries mentioned in report?               0
dtype: int64
```

```
df['IgnitionSourcePower'] =
df['IgnitionSourcePower'].fillna('Unknown')
df['IgnitionSource'] = df['IgnitionSource'].fillna('Unknown')
df['ItemFirstIgnited'] = df['ItemFirstIgnited'].fillna('Unknown')
df['VehiclePowerType'] = df['VehiclePowerType'].fillna('Unknown')
df['VehicleManufacturer'] =
df['VehicleManufacturer'].fillna('Unknown')

df['Target'] = df['Lithium batteries mentioned in
report?'].apply(lambda x: 1 if x == 'Yes' else 0)
df.drop(columns=['Lithium batteries mentioned in report?'],
inplace=True)

df.head()
```

```
   CalendarYear                   Type IgnitionSourcePower  \
0          2017  Other Lithium Battery         Electricity
1          2017  Other Lithium Battery             Unknown
2          2017                    Car         Electricity
3          2017                    Car         Electricity
4          2017              e-scooter         Electricity

                 IgnitionSource             ItemFirstIgnited
VehiclePowerType  \
0  Other appliance or equipment                   Other item
Unknown
1                     Not known  Plastic - raw material only
Unknown
2          Batteries, generators   Rubber - raw material only
Unknown
3          Batteries, generators  Plastic - raw material only
Unknown
4          Batteries, generators            Wiring insulation
Unknown

  VehicleManufacturer  Target
0             Unknown       1
1             Unknown       1
2            Mercedes       0
3            Mercedes       0
4             Unknown       0
```

```
df.isnull().sum()
```

```
CalendarYear              0
Type                      0
IgnitionSourcePower       0
```

```
IgnitionSource         0
ItemFirstIgnited       0
VehiclePowerType       0
VehicleManufacturer    0
Target                 0
dtype: int64

from sklearn.preprocessing import LabelEncoder

label_encoders = {}
for col in df.columns:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le

X = df.drop(columns=['Target'])
y = df['Target']

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, random_state=42)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from tensorflow.keras.models import Sequential
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from tensorflow.keras.layers import Dense, Dropout

model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

D:\anaconda\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 layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

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

history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=16,
    validation_split=0.2
)

Epoch 1/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.5759 - loss:
0.6540 - val_accuracy: 0.6667 - val_loss: 0.5672
Epoch 2/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7585 - loss:
0.4964 - val_accuracy: 0.7143 - val_loss: 0.5478
Epoch 3/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7671 - loss:
0.4895 - val_accuracy: 0.7381 - val_loss: 0.5354
Epoch 4/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7792 - loss:
0.4753 - val_accuracy: 0.7421 - val_loss: 0.5244
Epoch 5/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7935 - loss:
0.4480 - val_accuracy: 0.7540 - val_loss: 0.5153
Epoch 6/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8119 - loss:
0.4319 - val_accuracy: 0.7619 - val_loss: 0.5046
Epoch 7/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7982 - loss:
0.4152 - val_accuracy: 0.7540 - val_loss: 0.4894
Epoch 8/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7987 - loss:
0.4094 - val_accuracy: 0.7540 - val_loss: 0.4773
Epoch 9/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8313 - loss:
0.3887 - val_accuracy: 0.7500 - val_loss: 0.4663
Epoch 10/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8390 - loss:
0.3683 - val_accuracy: 0.7619 - val_loss: 0.4572
Epoch 11/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8424 - loss:
0.3740 - val_accuracy: 0.7659 - val_loss: 0.4520
Epoch 12/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8352 - loss:
0.3822 - val_accuracy: 0.7698 - val_loss: 0.4449
Epoch 13/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8410 - loss:
0.3596 - val_accuracy: 0.7659 - val_loss: 0.4407
Epoch 14/50
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8158 - loss:
0.3583 - val_accuracy: 0.7738 - val_loss: 0.4318
Epoch 15/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8623 - loss:
0.3260 - val_accuracy: 0.7778 - val_loss: 0.4253
Epoch 16/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8447 - loss:
0.3447 - val_accuracy: 0.7778 - val_loss: 0.4198
Epoch 17/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8434 - loss:
0.3371 - val_accuracy: 0.7817 - val_loss: 0.4161
Epoch 18/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8588 - loss:
0.3090 - val_accuracy: 0.7778 - val_loss: 0.4175
Epoch 19/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8359 - loss:
0.3532 - val_accuracy: 0.7778 - val_loss: 0.4089
Epoch 20/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8607 - loss:
0.3322 - val_accuracy: 0.7817 - val_loss: 0.4139
Epoch 21/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8341 - loss:
0.3224 - val_accuracy: 0.7778 - val_loss: 0.4099
Epoch 22/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8673 - loss:
0.3073 - val_accuracy: 0.7897 - val_loss: 0.4081
Epoch 23/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8578 - loss:
0.3069 - val_accuracy: 0.7778 - val_loss: 0.4002
Epoch 24/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8383 - loss:
0.3310 - val_accuracy: 0.7817 - val_loss: 0.3964
Epoch 25/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8544 - loss:
0.3178 - val_accuracy: 0.7778 - val_loss: 0.3976
Epoch 26/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8795 - loss:
0.2784 - val_accuracy: 0.7817 - val_loss: 0.3960
Epoch 27/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8478 - loss:
0.3260 - val_accuracy: 0.7817 - val_loss: 0.3907
Epoch 28/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8632 - loss:
0.3109 - val_accuracy: 0.7817 - val_loss: 0.3937
Epoch 29/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8497 - loss:
0.3123 - val_accuracy: 0.7738 - val_loss: 0.3968
Epoch 30/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8594 - loss:
```

```
0.3217 - val_accuracy: 0.7817 - val_loss: 0.3989
Epoch 31/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8663 - loss:
0.3020 - val_accuracy: 0.7778 - val_loss: 0.3914
Epoch 32/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8444 - loss:
0.3111 - val_accuracy: 0.7778 - val_loss: 0.3909
Epoch 33/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8551 - loss:
0.3219 - val_accuracy: 0.7857 - val_loss: 0.3855
Epoch 34/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8580 - loss:
0.2879 - val_accuracy: 0.7778 - val_loss: 0.3871
Epoch 35/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8665 - loss:
0.2910 - val_accuracy: 0.7857 - val_loss: 0.3894
Epoch 36/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8502 - loss:
0.2917 - val_accuracy: 0.7817 - val_loss: 0.3878
Epoch 37/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8526 - loss:
0.3108 - val_accuracy: 0.7778 - val_loss: 0.3963
Epoch 38/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8617 - loss:
0.2905 - val_accuracy: 0.7857 - val_loss: 0.3985
Epoch 39/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8493 - loss:
0.3046 - val_accuracy: 0.7778 - val_loss: 0.3889
Epoch 40/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8578 - loss:
0.3105 - val_accuracy: 0.7817 - val_loss: 0.3922
Epoch 41/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8626 - loss:
0.2935 - val_accuracy: 0.7817 - val_loss: 0.3926
Epoch 42/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8642 - loss:
0.3224 - val_accuracy: 0.7778 - val_loss: 0.3911
Epoch 43/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8652 - loss:
0.2907 - val_accuracy: 0.7738 - val_loss: 0.3917
Epoch 44/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8318 - loss:
0.3415 - val_accuracy: 0.7738 - val_loss: 0.3912
Epoch 45/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8599 - loss:
0.3054 - val_accuracy: 0.7817 - val_loss: 0.3907
Epoch 46/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8822 - loss:
0.2662 - val_accuracy: 0.7857 - val_loss: 0.3864
```

```
Epoch 47/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8501 - loss:
0.3012 - val_accuracy: 0.7857 - val_loss: 0.3860
Epoch 48/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8802 - loss:
0.2873 - val_accuracy: 0.7738 - val_loss: 0.3851
Epoch 49/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8821 - loss:
0.2896 - val_accuracy: 0.7857 - val_loss: 0.3827
Epoch 50/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8788 - loss:
0.2975 - val_accuracy: 0.7698 - val_loss: 0.3850
```

```
import matplotlib.pyplot as plt
import seaborn as sns

y_pred = (model.predict(X_test) > 0.5).astype("int32")
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```

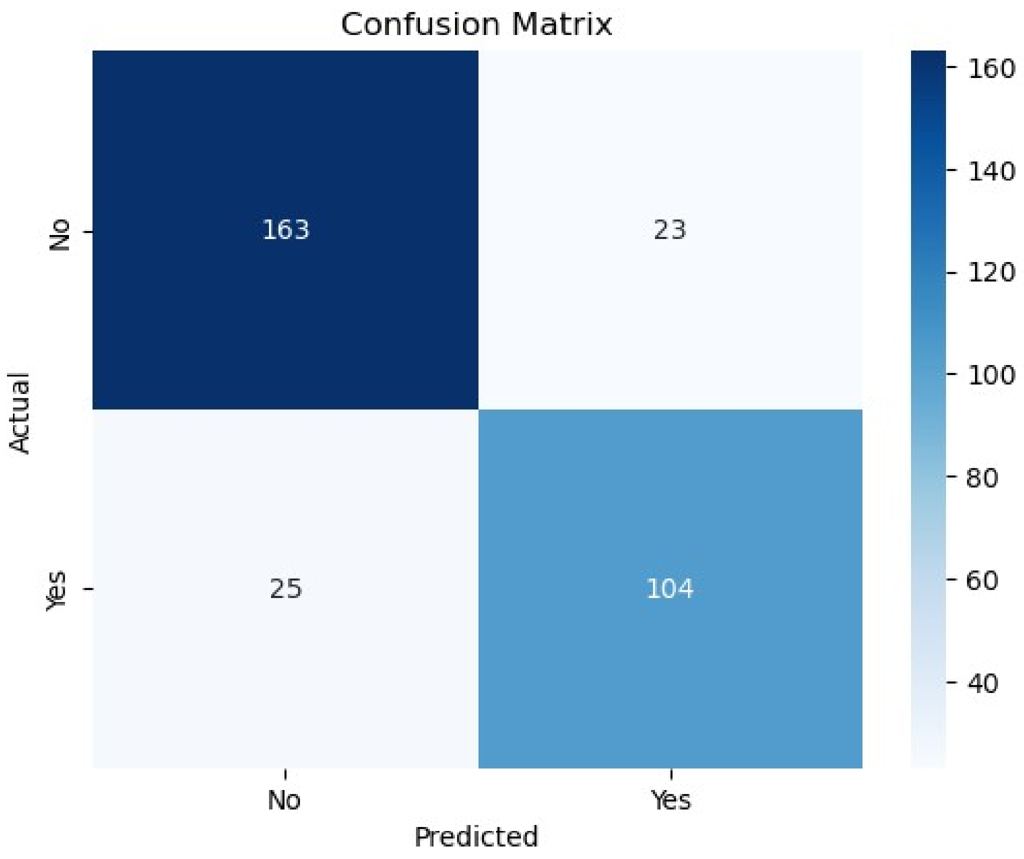
```
print("\n□ Accuracy:", accuracy_score(y_test, y_pred))
```

```

□ Accuracy: 0.8476190476190476
```

```
print("\n□ Classification Report:\n", classification_report(y_test,
y_pred))
```

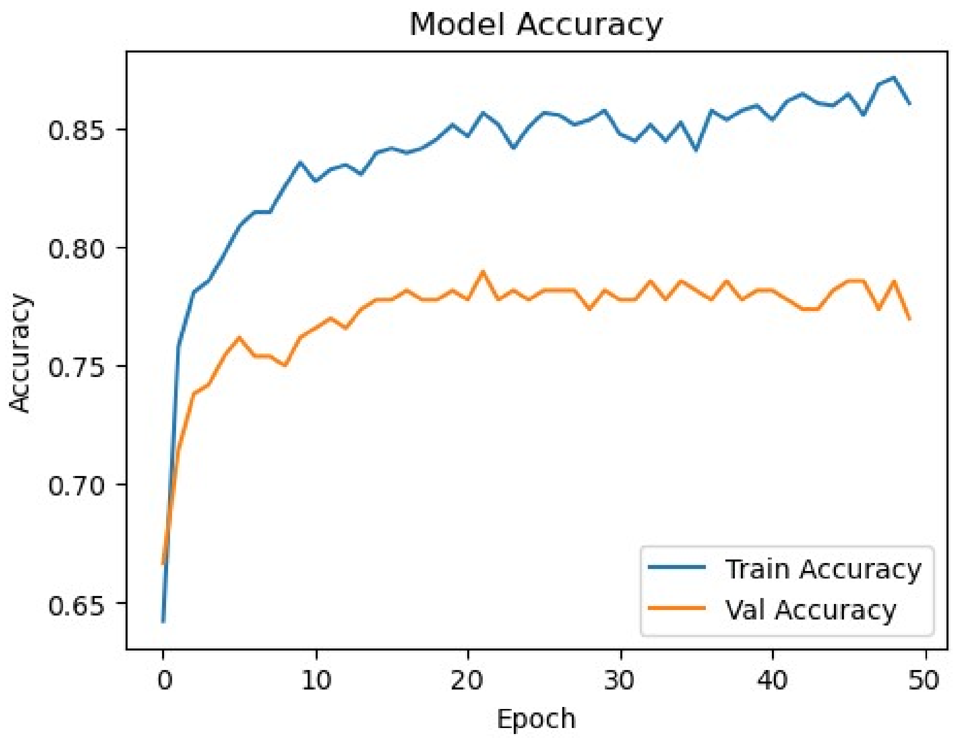
```

□ Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.88      0.87       186
           1       0.82      0.81      0.81       129

    accuracy                           0.85       315
   macro avg       0.84      0.84      0.84       315
weighted avg       0.85      0.85      0.85       315
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["No",
"Yes"], yticklabels=["No", "Yes"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

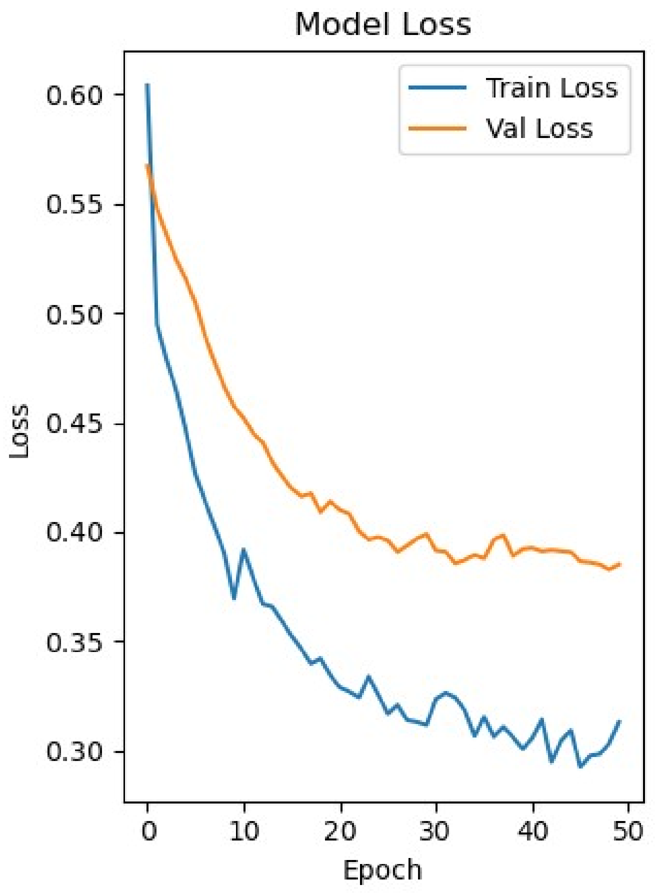


```
plt.figure(figsize=(12, 4))

# Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title("Model Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x20199ccd6d0>
```



```python
# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title("Model Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

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


Model Loss
Train Loss
Val Loss
Loss
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
10
20
30
40
50
Epoch