# Lending_Club_Data_Analysis

January 5, 2025

# 1 Home Loan Default Prediction

### 1.0.1 Deep Learning with Keras & TensorFlow

**Course End Project**

**Domain**: Finance
**Objective**: Predict loan defaults using historical data through a deep learning model.

---

### 1.0.2 Author

**Paul Badu Yakubu**

**Date**: January 2025

---

### 1.0.3 Project Highlights

- **Imbalanced Dataset Analysis**
- **Feature Engineering**
- **Deep Learning Model Implementation**
- **Performance Metrics: Sensitivity, ROC Curve, Accuracy**

---

"Empowering financial decisions through predictive analytics and AI."

## 1.1 Import Libraries

[41]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import confusion_matrix, accuracy_score, roc_curve, auc
```

## 2 0. Load and inspect the dataset

[78]:
```
data = pd.read_csv("loan_data.csv")
data.head()
```

[78]:
```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
3   8.10   712        2699.958333      33667        73.2               1
4  14.97   667        4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

[80]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   object
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64
 11  delinq.2yrs        9578 non-null   int64
 12  pub.rec            9578 non-null   int64
 13  not.fully.paid     9578 non-null   int64
```

```
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

[81]:
```
data.describe()
```

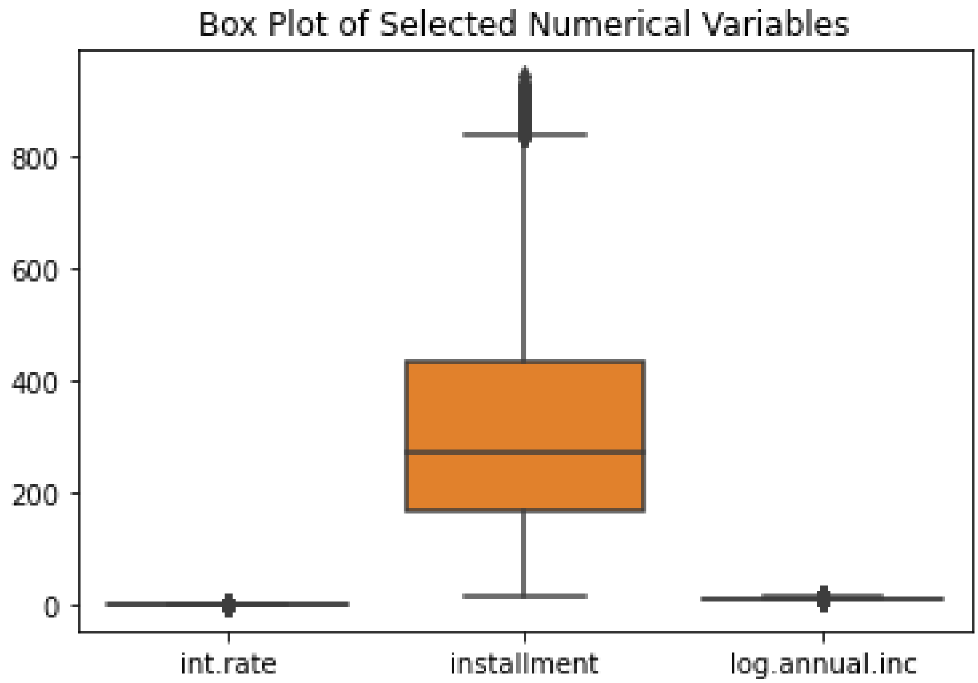
[81]:
```
       credit.policy     int.rate  installment  log.annual.inc          dti  \
count    9578.000000  9578.000000  9578.000000     9578.000000  9578.000000
mean        0.804970     0.122640   319.089413       10.932117    12.606679
std         0.396245     0.026847   207.071301        0.614813     6.883970
min         0.000000     0.060000    15.670000        7.547502     0.000000
25%         1.000000     0.103900   163.770000       10.558414     7.212500
50%         1.000000     0.122100   268.950000       10.928884    12.665000
75%         1.000000     0.140700   432.762500       11.291293    17.950000
max         1.000000     0.216400   940.140000       14.528354    29.960000

              fico  days.with.cr.line     revol.bal   revol.util  \
count  9578.000000        9578.000000  9.578000e+03  9578.000000
mean    710.846314        4560.767197  1.691396e+04    46.799236
std      37.970537        2496.930377  3.375619e+04    29.014417
min     612.000000         178.958333  0.000000e+00     0.000000
25%     682.000000        2820.000000  3.187000e+03    22.600000
50%     707.000000        4139.958333  8.596000e+03    46.300000
75%     737.000000        5730.000000  1.824950e+04    70.900000
max     827.000000       17639.958330  1.207359e+06   119.000000

       inq.last.6mths  delinq.2yrs      pub.rec  not.fully.paid
count     9578.000000  9578.000000  9578.000000     9578.000000
mean         1.577469     0.163708     0.062122        0.160054
std          2.200245     0.546215     0.262126        0.366676
min          0.000000     0.000000     0.000000        0.000000
25%          0.000000     0.000000     0.000000        0.000000
50%          1.000000     0.000000     0.000000        0.000000
75%          2.000000     0.000000     0.000000        0.000000
max         33.000000    13.000000     5.000000        1.000000
```

# 3 1. Exploratory Data Analysis

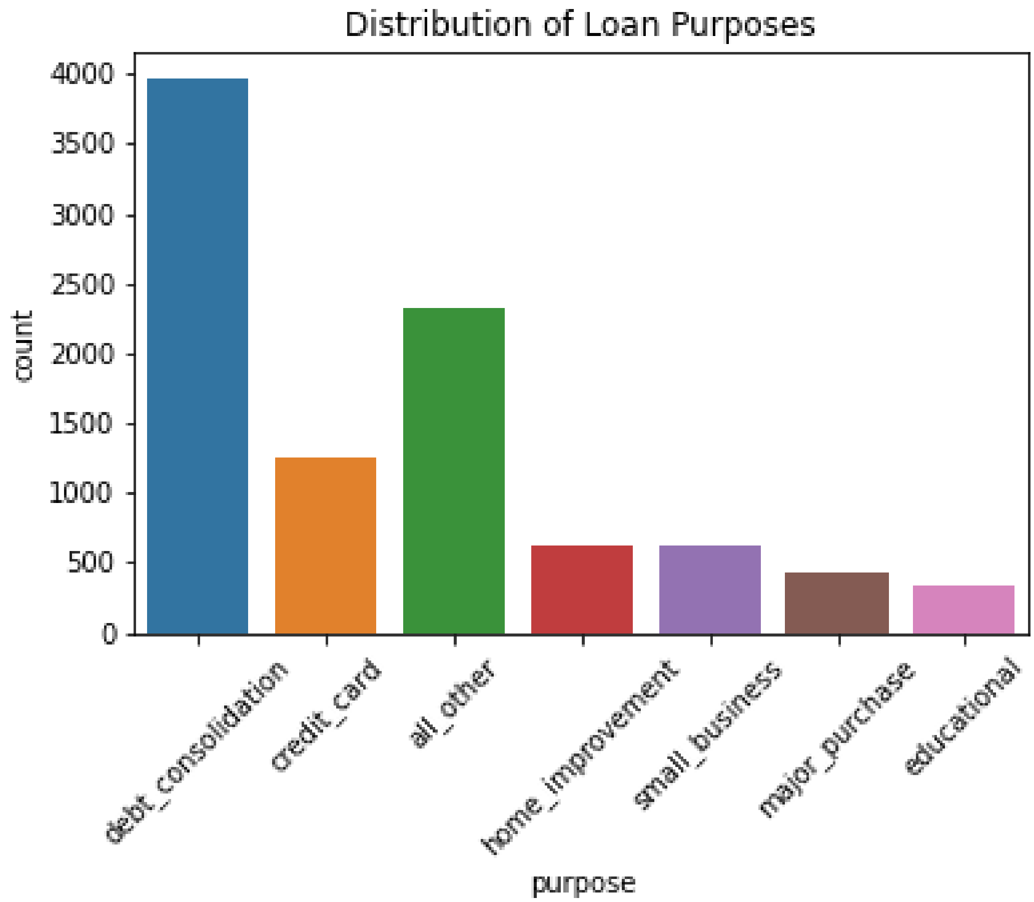
---

## 3.1 Visualize the distribution of numerical and categorical variables

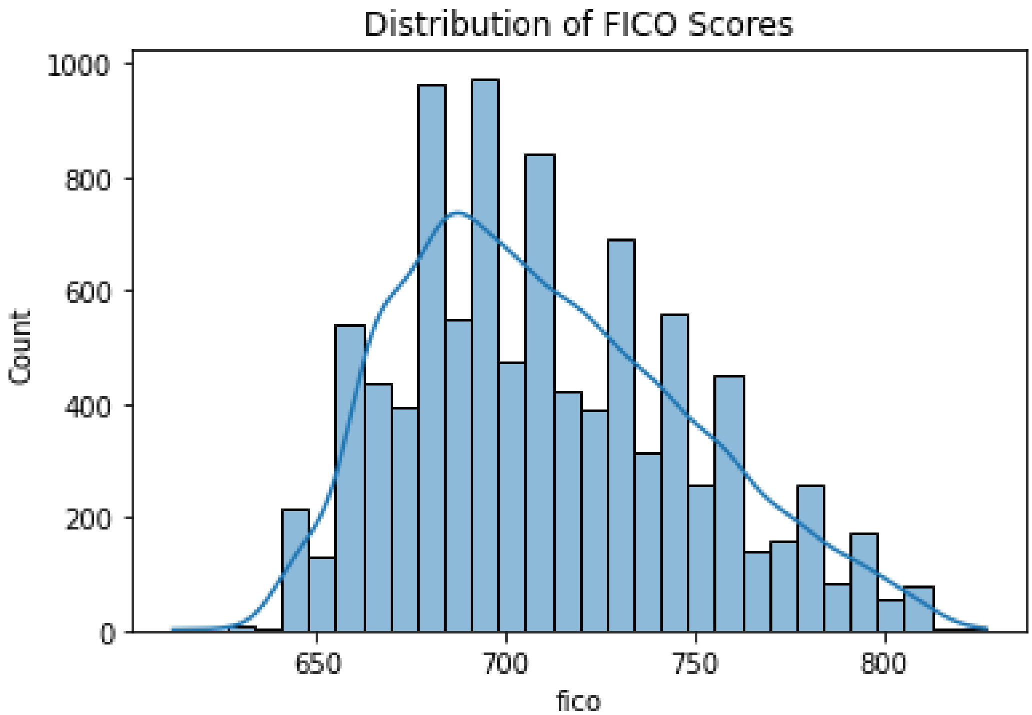
[82]:
```
sns.boxplot(data=data[['int.rate', 'installment', 'log.annual.inc']])
plt.title("Box Plot of Selected Numerical Variables")
plt.show()
```




[83]:
```
sns.countplot(data=data, x='purpose')
plt.title("Distribution of Loan Purposes")
plt.xticks(rotation=45)
plt.show()
```




[84]:
```
sns.histplot(data['fico'], kde=True, bins=30)
plt.title("Distribution of FICO Scores")
plt.show()
```

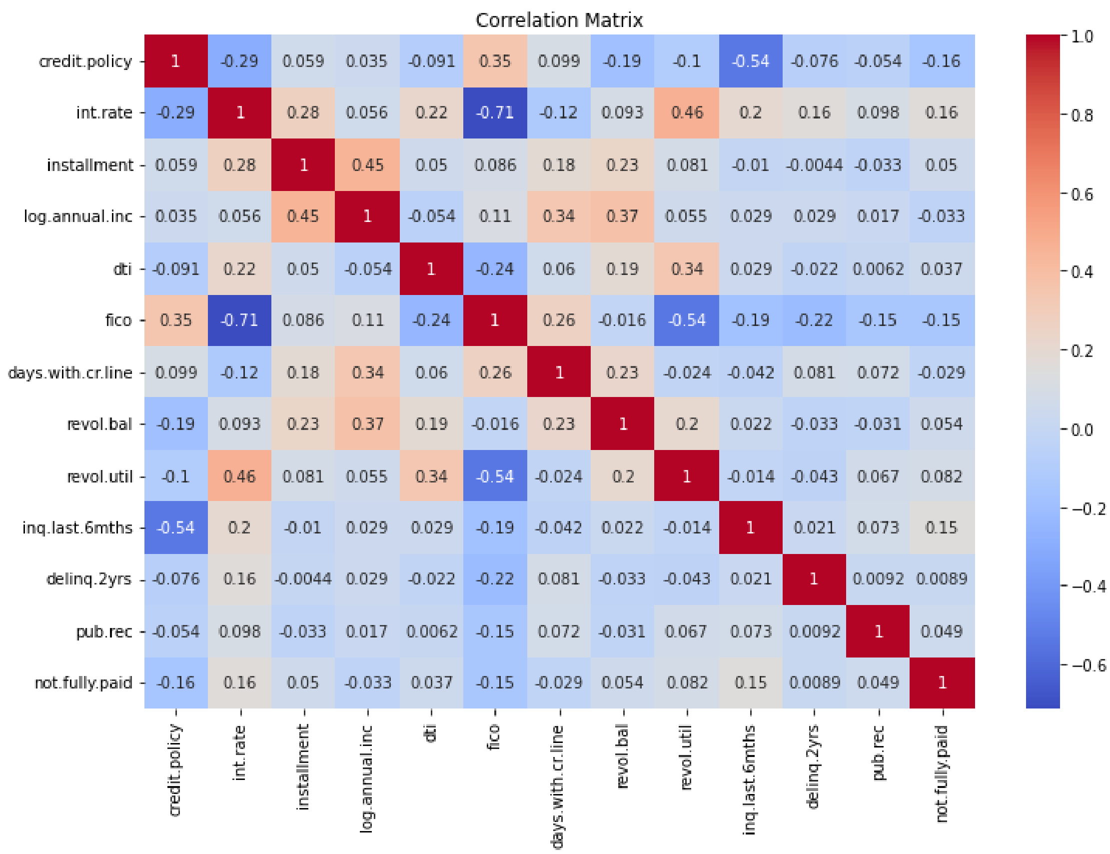



# 4 2. Feature Engineering

---

## 4.1 Check correlation between numerical features and drop highly correlated ones

[85]:
```
correlation_matrix = data.corr(numeric_only=True)
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```




[86]:
```
data = data.drop(columns=['not.fully.paid'])
```

[87]:
```
data.columns
```

[87]:
```
Index(['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec'],
      dtype='object')
```

# 5 3. One-hot encode categorical variables

[88]:
```
data.head()
```

[88]:
```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732
```

```
     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
3   8.10   712        2699.958333      33667        73.2               1
4  14.97   667        4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec
0            0        0
1            0        0
2            0        0
3            0        0
4            1        0
```

[89]:
```
data = pd.get_dummies(data, columns=['purpose'], drop_first=True)
```

[90]:
```
data.head()
```

[90]:
```
   credit.policy  int.rate  installment  log.annual.inc    dti  fico  \
0              1    0.1189       829.10       11.350407  19.48   737
1              1    0.1071       228.22       11.082143  14.29   707
2              1    0.1357       366.86       10.373491  11.63   682
3              1    0.1008       162.34       11.350407   8.10   712
4              1    0.1426       102.92       11.299732  14.97   667

   days.with.cr.line  revol.bal  revol.util  inq.last.6mths  delinq.2yrs  \
0        5639.958333      28854        52.1               0            0
1        2760.000000      33623        76.7               0            0
2        4710.000000       3511        25.6               1            0
3        2699.958333      33667        73.2               1            0
4        4066.000000       4740        39.5               0            1

   pub.rec  purpose_credit_card  purpose_debt_consolidation  \
0        0                    0                           1
1        0                    1                           0
2        0                    0                           1
3        0                    0                           1
4        0                    1                           0

   purpose_educational  purpose_home_improvement  purpose_major_purchase  \
0                    0                         0                       0
1                    0                         0                       0
2                    0                         0                       0
3                    0                         0                       0
4                    0                         0                       0
```

```
   purpose_small_business
0                       0
1                       0
2                       0
3                       0
4                       0
```

[92]:
```
data.columns
```

[92]:
```
Index(['credit.policy', 'int.rate', 'installment', 'log.annual.inc', 'dti',
       'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'purpose_credit_card',
       'purpose_debt_consolidation', 'purpose_educational',
       'purpose_home_improvement', 'purpose_major_purchase',
       'purpose_small_business'],
      dtype='object')
```

# 6 4. Prepare Data for Modeling

---

## 6.1 a. Create a feature dataset and a target dataset (x and y)

[93]:
```
X = data.drop(columns=['credit.policy'])
y = data['credit.policy']
```

## 6.2 b. Train-test split with a test size of 25%

[94]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42, stratify=y)
```

## 6.3 c. Normalize the train and test features using MinMaxScaler

[95]:
```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

# 7 5. Modeling

---

## 7.1 a. Build the Deep Learning Model

[96]:
```
model = Sequential([
    Dense(64, activation='relu', input_dim=X_train_scaled.shape[1]),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
```

## 7.2 b. Compile the model

[97]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

## 7.3 c. Train the model

[98]:
```
history = model.fit(X_train_scaled, y_train, epochs=50, validation_split=0.1,␣
 ↪batch_size=32)
```

```
Epoch 1/50
202/202 [==============================] - 1s 2ms/step - loss: 0.4874 -
accuracy: 0.7896 - val_loss: 0.3795 - val_accuracy: 0.8248
Epoch 2/50
202/202 [==============================] - 0s 926us/step - loss: 0.3880 -
accuracy: 0.8308 - val_loss: 0.3197 - val_accuracy: 0.8776
Epoch 3/50
202/202 [==============================] - 0s 947us/step - loss: 0.3445 -
accuracy: 0.8646 - val_loss: 0.2925 - val_accuracy: 0.8804
Epoch 4/50
202/202 [==============================] - 0s 946us/step - loss: 0.3114 -
accuracy: 0.8778 - val_loss: 0.2671 - val_accuracy: 0.8943
Epoch 5/50
202/202 [==============================] - 0s 922us/step - loss: 0.2905 -
accuracy: 0.8869 - val_loss: 0.2606 - val_accuracy: 0.8999
Epoch 6/50
202/202 [==============================] - 0s 911us/step - loss: 0.2793 -
accuracy: 0.8948 - val_loss: 0.2485 - val_accuracy: 0.9013
Epoch 7/50
202/202 [==============================] - 0s 909us/step - loss: 0.2777 -
accuracy: 0.8911 - val_loss: 0.2480 - val_accuracy: 0.9054
Epoch 8/50
202/202 [==============================] - 0s 925us/step - loss: 0.2696 -
accuracy: 0.8967 - val_loss: 0.2463 - val_accuracy: 0.8999
Epoch 9/50
202/202 [==============================] - 0s 924us/step - loss: 0.2645 -
accuracy: 0.8951 - val_loss: 0.2422 - val_accuracy: 0.9026
```

```
Epoch 10/50
202/202 [==============================] - 0s 900us/step - loss: 0.2648 -
accuracy: 0.9016 - val_loss: 0.2446 - val_accuracy: 0.9054
Epoch 11/50
202/202 [==============================] - 0s 919us/step - loss: 0.2601 -
accuracy: 0.8990 - val_loss: 0.2384 - val_accuracy: 0.9054
Epoch 12/50
202/202 [==============================] - 0s 907us/step - loss: 0.2608 -
accuracy: 0.9015 - val_loss: 0.2392 - val_accuracy: 0.9013
Epoch 13/50
202/202 [==============================] - 0s 905us/step - loss: 0.2557 -
accuracy: 0.9015 - val_loss: 0.2357 - val_accuracy: 0.9124
Epoch 14/50
202/202 [==============================] - 0s 900us/step - loss: 0.2540 -
accuracy: 0.9028 - val_loss: 0.2298 - val_accuracy: 0.9096
Epoch 15/50
202/202 [==============================] - 0s 912us/step - loss: 0.2509 -
accuracy: 0.9045 - val_loss: 0.2274 - val_accuracy: 0.9082
Epoch 16/50
202/202 [==============================] - 0s 936us/step - loss: 0.2509 -
accuracy: 0.9019 - val_loss: 0.2267 - val_accuracy: 0.9082
Epoch 17/50
202/202 [==============================] - 0s 932us/step - loss: 0.2406 -
accuracy: 0.9036 - val_loss: 0.2228 - val_accuracy: 0.9082
Epoch 18/50
202/202 [==============================] - 0s 917us/step - loss: 0.2427 -
accuracy: 0.9062 - val_loss: 0.2273 - val_accuracy: 0.9054
Epoch 19/50
202/202 [==============================] - 0s 904us/step - loss: 0.2444 -
accuracy: 0.9049 - val_loss: 0.2246 - val_accuracy: 0.9068
Epoch 20/50
202/202 [==============================] - 0s 909us/step - loss: 0.2409 -
accuracy: 0.9033 - val_loss: 0.2213 - val_accuracy: 0.9082
Epoch 21/50
202/202 [==============================] - 0s 920us/step - loss: 0.2396 -
accuracy: 0.9107 - val_loss: 0.2182 - val_accuracy: 0.9096
Epoch 22/50
202/202 [==============================] - 0s 896us/step - loss: 0.2374 -
accuracy: 0.9078 - val_loss: 0.2165 - val_accuracy: 0.9082
Epoch 23/50
202/202 [==============================] - 0s 894us/step - loss: 0.2342 -
accuracy: 0.9117 - val_loss: 0.2181 - val_accuracy: 0.9152
Epoch 24/50
202/202 [==============================] - 0s 913us/step - loss: 0.2309 -
accuracy: 0.9090 - val_loss: 0.2140 - val_accuracy: 0.9096
Epoch 25/50
202/202 [==============================] - 0s 968us/step - loss: 0.2291 -
accuracy: 0.9100 - val_loss: 0.2149 - val_accuracy: 0.9054
```

```
Epoch 26/50
202/202 [==============================] - 0s 919us/step - loss: 0.2267 -
accuracy: 0.9110 - val_loss: 0.2137 - val_accuracy: 0.9054
Epoch 27/50
202/202 [==============================] - 0s 916us/step - loss: 0.2247 -
accuracy: 0.9115 - val_loss: 0.2130 - val_accuracy: 0.9124
Epoch 28/50
202/202 [==============================] - 0s 914us/step - loss: 0.2253 -
accuracy: 0.9137 - val_loss: 0.2160 - val_accuracy: 0.9040
Epoch 29/50
202/202 [==============================] - 0s 913us/step - loss: 0.2199 -
accuracy: 0.9121 - val_loss: 0.2083 - val_accuracy: 0.9124
Epoch 30/50
202/202 [==============================] - 0s 906us/step - loss: 0.2262 -
accuracy: 0.9109 - val_loss: 0.2082 - val_accuracy: 0.9110
Epoch 31/50
202/202 [==============================] - 0s 902us/step - loss: 0.2117 -
accuracy: 0.9199 - val_loss: 0.2047 - val_accuracy: 0.9124
Epoch 32/50
202/202 [==============================] - 0s 916us/step - loss: 0.2138 -
accuracy: 0.9155 - val_loss: 0.2043 - val_accuracy: 0.9082
Epoch 33/50
202/202 [==============================] - 0s 916us/step - loss: 0.2136 -
accuracy: 0.9179 - val_loss: 0.2011 - val_accuracy: 0.9138
Epoch 34/50
202/202 [==============================] - 0s 909us/step - loss: 0.2150 -
accuracy: 0.9141 - val_loss: 0.2003 - val_accuracy: 0.9082
Epoch 35/50
202/202 [==============================] - 0s 899us/step - loss: 0.2067 -
accuracy: 0.9199 - val_loss: 0.1954 - val_accuracy: 0.9193
Epoch 36/50
202/202 [==============================] - 0s 922us/step - loss: 0.2042 -
accuracy: 0.9225 - val_loss: 0.1987 - val_accuracy: 0.9193
Epoch 37/50
202/202 [==============================] - 0s 906us/step - loss: 0.2071 -
accuracy: 0.9179 - val_loss: 0.1951 - val_accuracy: 0.9166
Epoch 38/50
202/202 [==============================] - 0s 909us/step - loss: 0.2020 -
accuracy: 0.9251 - val_loss: 0.1943 - val_accuracy: 0.9207
Epoch 39/50
202/202 [==============================] - 0s 922us/step - loss: 0.2005 -
accuracy: 0.9216 - val_loss: 0.1919 - val_accuracy: 0.9193
Epoch 40/50
202/202 [==============================] - 0s 924us/step - loss: 0.2029 -
accuracy: 0.9199 - val_loss: 0.1938 - val_accuracy: 0.9166
Epoch 41/50
202/202 [==============================] - 0s 933us/step - loss: 0.1996 -
accuracy: 0.9199 - val_loss: 0.1909 - val_accuracy: 0.9235
```

```
Epoch 42/50
202/202 [==============================] - 0s 918us/step - loss: 0.1972 -
accuracy: 0.9239 - val_loss: 0.1866 - val_accuracy: 0.9179
Epoch 43/50
202/202 [==============================] - 0s 923us/step - loss: 0.1936 -
accuracy: 0.9279 - val_loss: 0.1821 - val_accuracy: 0.9207
Epoch 44/50
202/202 [==============================] - 0s 921us/step - loss: 0.1891 -
accuracy: 0.9273 - val_loss: 0.1802 - val_accuracy: 0.9263
Epoch 45/50
202/202 [==============================] - 0s 910us/step - loss: 0.1877 -
accuracy: 0.9262 - val_loss: 0.1798 - val_accuracy: 0.9277
Epoch 46/50
202/202 [==============================] - 0s 916us/step - loss: 0.1857 -
accuracy: 0.9274 - val_loss: 0.1748 - val_accuracy: 0.9291
Epoch 47/50
202/202 [==============================] - 0s 908us/step - loss: 0.1858 -
accuracy: 0.9296 - val_loss: 0.1783 - val_accuracy: 0.9305
Epoch 48/50
202/202 [==============================] - 0s 914us/step - loss: 0.1794 -
accuracy: 0.9322 - val_loss: 0.1734 - val_accuracy: 0.9291
Epoch 49/50
202/202 [==============================] - 0s 910us/step - loss: 0.1768 -
accuracy: 0.9335 - val_loss: 0.1679 - val_accuracy: 0.9332
Epoch 50/50
202/202 [==============================] - 0s 913us/step - loss: 0.1749 -
accuracy: 0.9338 - val_loss: 0.1634 - val_accuracy: 0.9346
```

### 7.4 d. Print model summary

[99]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                1152

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 3265 (12.75 KB)
```

```
Trainable params: 3265 (12.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

### 7.5 e. Evaluate the Loss and Accuracy for the test set

[100]:
```python
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
```

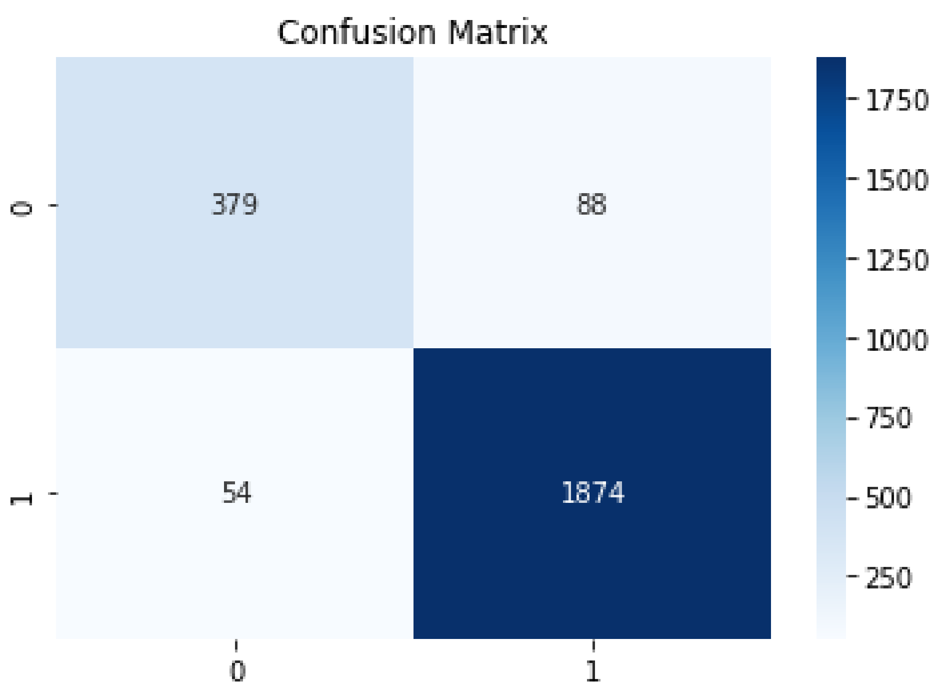
```
75/75 [==============================] - 0s 672us/step - loss: 0.1575 -
accuracy: 0.9407
Test Loss: 0.15752221643924713, Test Accuracy: 0.9407098293304443
```

### 7.6 f. Predictions and metrics

[101]:
```python
y_pred = (model.predict(X_test_scaled) > 0.5).astype("int32")
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
75/75 [==============================] - 0s 598us/step
Confusion Matrix:
[[ 379   88]
 [  54 1874]]
```

[102]:
```python
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



# 8 THANK YOU