In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('FastagFraudDetection.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Geo
0	1	1/6/2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	Large	350	120	1
1	2	1/7/2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	Small	120	100	1
2	3	1/8/2023 18:25	Motorcycle	NaN	D-104	Regular	Small	0	0	1
3	4	1/9/2023 2:05	Truck	FTG-044-LMN-322	C-103	Regular	Large	350	120	1
4	5	1/10/2023 6:35	Van	FTG-505-DEF-652	B-102	Express	Medium	140	100	1

In [4]:
```
df.tail()
```

Out[4]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid
4995	4996	1/1/2023 22:18	Truck	FTG-445-EDC-765	C-103	Regular	Large	330	330
4996	4997	1/17/2023 13:43	Van	FTG-446-LMK-432	B-102	Express	Medium	125	125
4997	4998	2/5/2023 5:08	Sedan	FTG-447-PLN-109	A-101	Regular	Medium	115	115
4998	4999	2/20/2023 20:34	SUV	FTG-458-VFR-876	B-102	Express	Large	145	145
4999	5000	3/10/2023 0:59	Bus	FTG-459-WSX-543	C-103	Regular	Large	330	125

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Transaction_ID             0
Timestamp                  0
Vehicle_Type               0
FastagID                 549
TollBoothID                0
Lane_Type                  0
Vehicle_Dimensions         0
Transaction_Amount         0
Amount_paid                0
Geographical_Location      0
Vehicle_Speed              0
Vehicle_Plate_Number       0
Fraud_indicator            0
dtype: int64
```

In [6]:
```
df.describe()
```

Out[6]:

	Transaction_ID	Transaction_Amount	Amount_paid	Vehicle_Speed
count	5000.000000	5000.00000	5000.000000	5000.000000
mean	2500.500000	161.06200	141.261000	67.851200
std	1443.520003	112.44995	106.480996	16.597547
min	1.000000	0.00000	0.000000	10.000000
25%	1250.750000	100.00000	90.000000	54.000000
50%	2500.500000	130.00000	120.000000	67.000000
75%	3750.250000	290.00000	160.000000	82.000000
max	5000.000000	350.00000	350.000000	118.000000

In [7]:
```
df.dtypes
```

Out[7]:
```
Transaction_ID            int64
Timestamp                object
Vehicle_Type             object
FastagID                 object
TollBoothID              object
Lane_Type                object
Vehicle_Dimensions       object
Transaction_Amount        int64
Amount_paid               int64
Geographical_Location    object
Vehicle_Speed             int64
Vehicle_Plate_Number     object
Fraud_indicator          object
dtype: object
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Transaction_ID         5000 non-null   int64
 1   Timestamp              5000 non-null   object
 2   Vehicle_Type           5000 non-null   object
 3   FastagID               4451 non-null   object
 4   TollBoothID            5000 non-null   object
 5   Lane_Type              5000 non-null   object
 6   Vehicle_Dimensions     5000 non-null   object
 7   Transaction_Amount     5000 non-null   int64
 8   Amount_paid            5000 non-null   int64
 9   Geographical_Location  5000 non-null   object
 10  Vehicle_Speed          5000 non-null   int64
 11  Vehicle_Plate_Number   5000 non-null   object
 12  Fraud_indicator        5000 non-null   object
dtypes: int64(4), object(9)
memory usage: 507.9+ KB
```

In [9]:
```
df.nunique()
```

Out[9]:
```
Transaction_ID           5000
Timestamp                4423
Vehicle_Type                7
FastagID                 4451
TollBoothID                 6
Lane_Type                   2
Vehicle_Dimensions          3
Transaction_Amount         20
Amount_paid                23
Geographical_Location       5
Vehicle_Speed              85
Vehicle_Plate_Number     5000
Fraud_indicator             2
dtype: int64
```

In [10]:
```
df.shape
```

Out[10]:
```
(5000, 13)
```

In [11]:
```
df.duplicated().sum()
```

Out[11]:
```
0
```

In [12]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [13]:
```
df['FastagID'].unique()
```

Out[13]:
```
array(['FTG-001-ABC-121', 'FTG-002-XYZ-451', nan, ..., 'FTG-447-PLN-109',
       'FTG-458-VFR-876', 'FTG-459-WSX-543'], dtype=object)
```

In [14]:
```
df['FastagID'].value_counts()
```

Out[14]:
```
FastagID
FTG-001-ABC-121    1
FTG-524-CDE-098    1
FTG-531-ZAS-987    1
FTG-530-LPO-210    1
FTG-528-WSX-876    1
                  ..
FTG-414-HIJ-567    1
FTG-647-KLM-890    1
FTG-880-NOP-123    1
FTG-113-QRS-456    1
FTG-459-WSX-543    1
Name: count, Length: 4451, dtype: int64
```

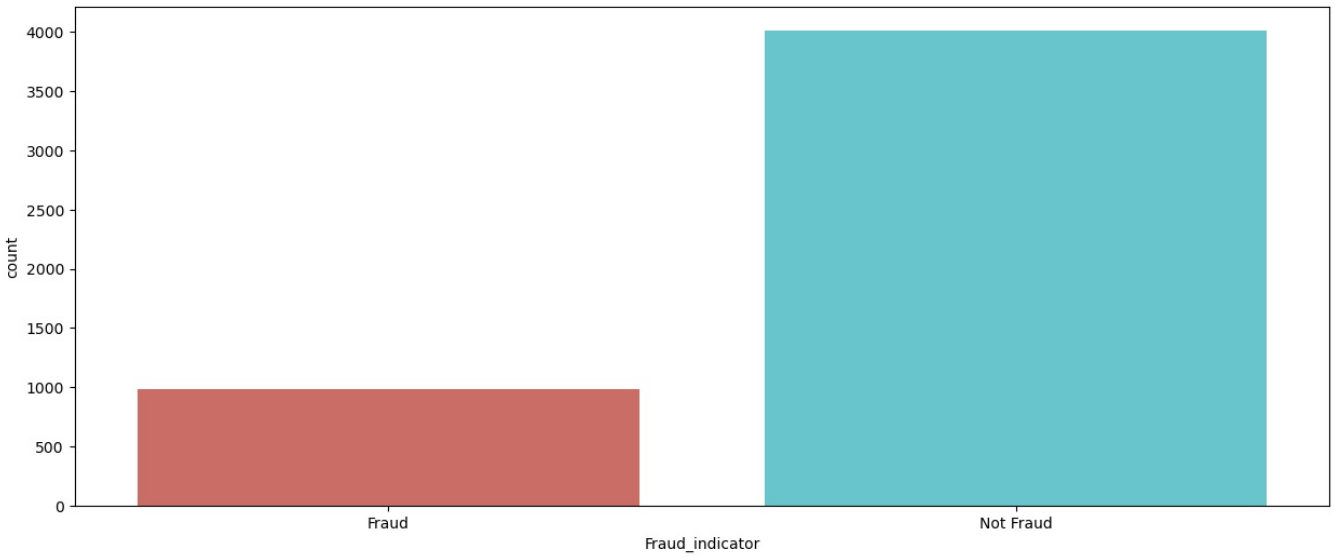
In [15]:
```
df['Fraud_indicator'].unique()
```

Out[15]:
```
array(['Fraud', 'Not Fraud'], dtype=object)
```

In [16]:
```
df['Fraud_indicator'].value_counts()
```

Out[16]:
```
Fraud_indicator
Not Fraud    4017
Fraud         983
Name: count, dtype: int64
```

In [17]:
```
plt.figure(figsize=(15, 6))
sns.countplot(x='Fraud_indicator', data=df, palette='hls')
plt.show()
```



In [18]:
```
df.dropna(inplace=True)
```

In [19]:
```
df.head()
```

Out[19]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Geo
0	1	1/6/2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	Large	350	120	1
1	2	1/7/2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	Small	120	100	1
3	4	1/9/2023 2:05	Truck	FTG-044-LMN-322	C-103	Regular	Large	350	120	1
4	5	1/10/2023 6:35	Van	FTG-505-DEF-652	B-102	Express	Medium	140	100	1
5	6	1/11/2023 10:00	Sedan	FTG-066-GHI-987	A-101	Regular	Medium	160	100	1

In [20]:
```
df.shape
```

Out[20]:
```
(4451, 13)
```

In [21]:
```
df.isnull().sum()
```

Out[21]:
```
Transaction_ID           0
Timestamp                0
Vehicle_Type             0
FastagID                 0
TollBoothID              0
Lane_Type                0
Vehicle_Dimensions       0
Transaction_Amount       0
Amount_paid              0
Geographical_Location    0
Vehicle_Speed            0
Vehicle_Plate_Number     0
Fraud_indicator          0
dtype: int64
```

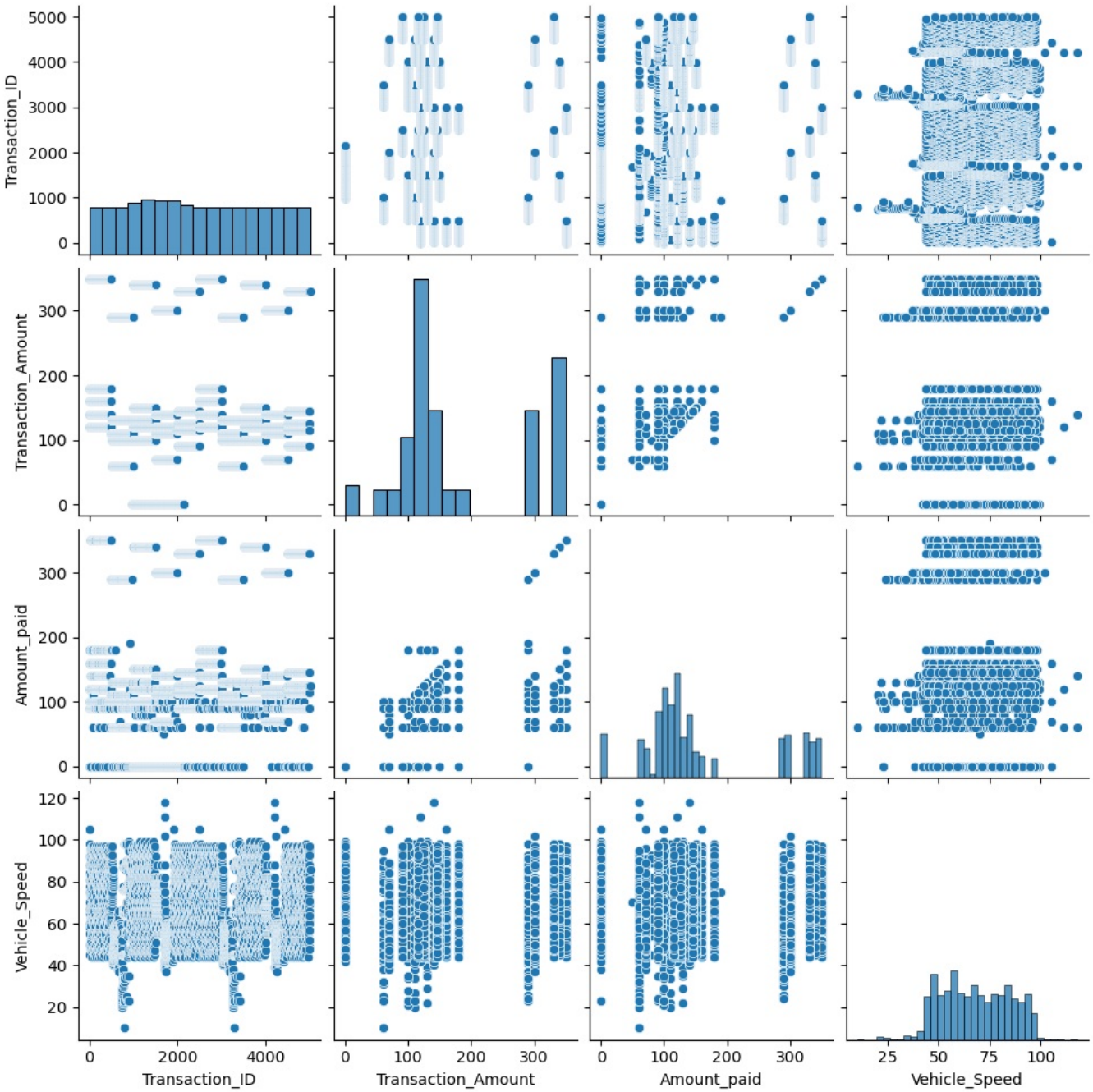
In [22]:
```
import plotly.express as px

fig = px.bar(df, x="Fraud_indicator", y= df.index)
fig.show()
```

In [23]:
```
value_counts = df['Fraud_indicator'].value_counts()
fig = px.pie(names=value_counts.index, values=value_counts.values)
fig.update_layout(
    title='Pie Chart of Fraud_indicator',
    title_x=0.5
)
```

```
fig.show()
```

In [24]:
```
sns.pairplot(df)
```

Out[24]:
```
<seaborn.axisgrid.PairGrid at 0x1d5b0112a90>
```



In [33]:
```
sns.boxplot(x='Fraud_indicator', y='Vehicle_Speed', data=df)
plt.show()
```

In [35]:
```
df[['Transaction_Amount', 'Amount_paid', 'Vehicle_Speed']].corr()
```

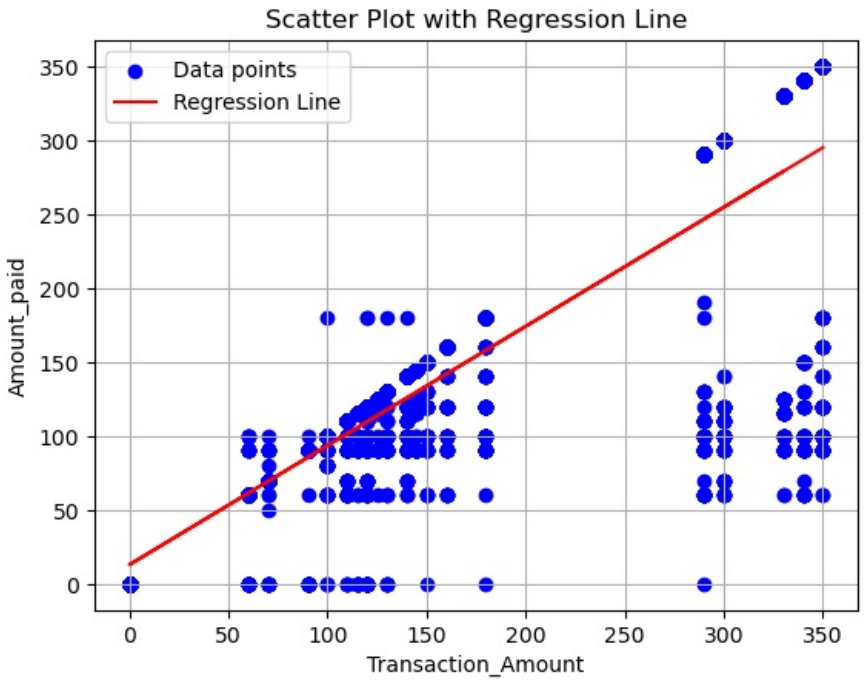
Out[35]:

	Transaction_Amount	Amount_paid	Vehicle_Speed
Transaction_Amount	1.000000	0.831275	0.061599
Amount_paid	0.831275	1.000000	0.043446
Vehicle_Speed	0.061599	0.043446	1.000000

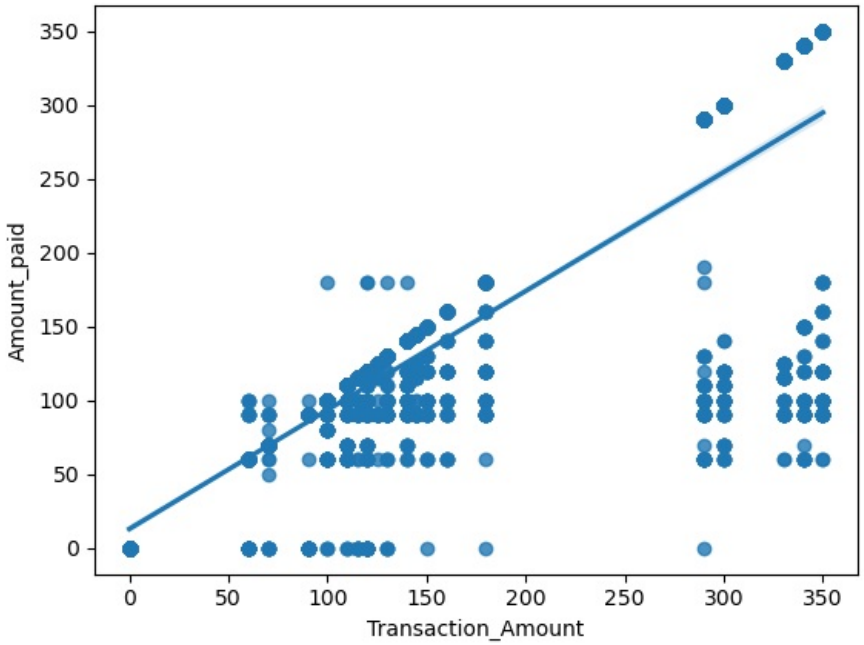
In [38]:
```
correlation = df['Transaction_Amount'].corr(df['Amount_paid'])
correlation
```

Out[38]:
```
0.8312749747685372
```

In [40]:
```
# Calculate the regression line
coefficients = np.polyfit(df['Transaction_Amount'], df['Amount_paid'], 1)
regression_line = np.polyval(coefficients, df['Transaction_Amount'])

# Plot a scatter plot with the regression line
plt.scatter(df['Transaction_Amount'], df['Amount_paid'], color='blue', label='Data points')
plt.plot(df['Transaction_Amount'], regression_line, color='red', label='Regression Line')
plt.title('Scatter Plot with Regression Line')
plt.xlabel('Transaction_Amount')
plt.ylabel('Amount_paid')
plt.legend()
plt.grid(True)
plt.show()
```



In [41]:
```
sns.regplot(x='Transaction_Amount', y='Amount_paid', data=df)
plt.show()
```



In [43]:
```
df = df.copy()
```

```
df.head(3)
```

Out[43]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Geo
0	1	1/6/2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	Large	350	120	1
1	2	1/7/2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	Small	120	100	1
3	4	1/9/2023 2:05	Truck	FTG-044-LMN-322	C-103	Regular	Large	350	120	1

In [44]:
```
X = df[['Transaction_Amount','Amount_paid']]
y = df['Fraud_indicator']
```

In [45]:
```
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)
```

In [46]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [47]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
```

In [48]:
```
df.head()
```

Out[48]:

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Geo
0	1	1/6/2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	Large	350	120	1
1	2	1/7/2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	Small	120	100	1
3	4	1/9/2023 2:05	Truck	FTG-044-LMN-322	C-103	Regular	Large	350	120	1
4	5	1/10/2023 6:35	Van	FTG-505-DEF-652	B-102	Express	Medium	140	100	1
5	6	1/11/2023 10:00	Sedan	FTG-066-GHI-987	A-101	Regular	Medium	160	100	1

In [49]:
```
#Neural network model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
WARNING:tensorflow:From C:\Users\Acer\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.
sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead
.

WARNING:tensorflow:From C:\Users\Acer\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name tf.get_def
ault_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

In [50]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                96

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 641 (2.50 KB)
Trainable params: 641 (2.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [51]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\Acer\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309: The nam
e tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

In [52]:
```
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
WARNING:tensorflow:From C:\Users\Acer\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.
ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Acer\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The
name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_fu
nctions instead.

89/89 [==============================] - 3s 11ms/step - loss: 0.5047 - accuracy: 0.7788 - val_loss: 0.4317 - va
l_accuracy: 0.7739
Epoch 2/10
89/89 [==============================] - 0s 5ms/step - loss: 0.3707 - accuracy: 0.8294 - val_loss: 0.3243 - val
_accuracy: 0.8638
Epoch 3/10
89/89 [==============================] - 0s 5ms/step - loss: 0.2643 - accuracy: 0.8912 - val_loss: 0.2259 - val
_accuracy: 0.9242
Epoch 4/10
89/89 [==============================] - 0s 5ms/step - loss: 0.1900 - accuracy: 0.9305 - val_loss: 0.1768 - val
_accuracy: 0.9312
Epoch 5/10
89/89 [==============================] - 0s 5ms/step - loss: 0.1507 - accuracy: 0.9501 - val_loss: 0.1464 - val
_accuracy: 0.9607
Epoch 6/10
89/89 [==============================] - 1s 6ms/step - loss: 0.1239 - accuracy: 0.9579 - val_loss: 0.1288 - val
_accuracy: 0.9719
Epoch 7/10
89/89 [==============================] - 1s 8ms/step - loss: 0.1063 - accuracy: 0.9733 - val_loss: 0.1185 - val
_accuracy: 0.9817
Epoch 8/10
89/89 [==============================] - 1s 6ms/step - loss: 0.0952 - accuracy: 0.9810 - val_loss: 0.1096 - val
_accuracy: 0.9817
Epoch 9/10
89/89 [==============================] - 0s 5ms/step - loss: 0.0862 - accuracy: 0.9814 - val_loss: 0.1058 - val
_accuracy: 0.9817
Epoch 10/10
89/89 [==============================] - 1s 6ms/step - loss: 0.0800 - accuracy: 0.9821 - val_loss: 0.1014 - val
_accuracy: 0.9817
```

Out[52]:
```
<keras.src.callbacks.History at 0x1d5bd481b10>
```

In [53]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
y_pred_prob = model.predict(X_test)

# Convert probabilities to binary predictions
y_pred = np.round(y_pred_prob)

# Print accuracy metrics
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 accuracy metrics
print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1))
```

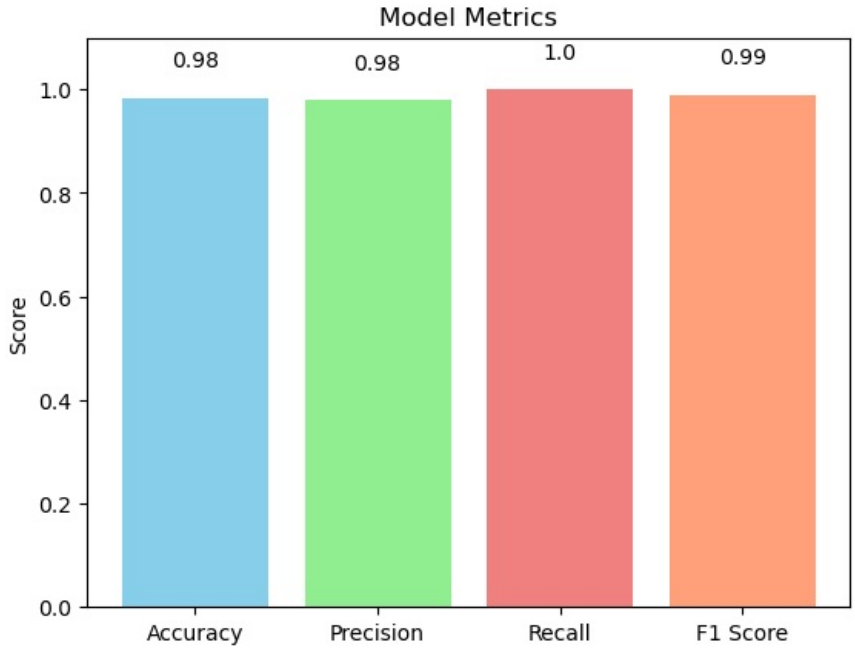
```
28/28 [==============================] - 0s 6ms/step
Accuracy: 98.43%
Precision: 0.98
Recall: 1.00
F1 Score: 0.99
```

In [54]:
```
print(y_pred_prob[:5])
y_pred[:5]
```

```
[[0.9332529 ]
 [0.92760545]
 [0.94306105]
 [0.9372957 ]
 [0.16856828]]
```

Out[54]:
```
array([[1.],
       [1.],
       [1.],
       [1.],
       [0.]], dtype=float32)
```

In [55]:
```
import matplotlib.pyplot as plt

metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
values = [accuracy, precision, recall, f1]

# Define custom colors for bars and text
bar_colors = ['skyblue', 'lightgreen', 'lightcoral', 'lightsalmon']
plt.bar(metrics, values, color=bar_colors)

# Adding values on top of each bar
for i, v in enumerate(values):
    plt.text(i, v + 0.05, str(round(v, 2)), ha='center', va='bottom')

plt.ylabel('Score')
plt.title('Model Metrics')
plt.ylim(0, 1.1)
plt.show()
```



In [ ]: