In [11]:
```
#a. Import required libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
```

In [28]:
```
#b. Upload / access the dataset
dataset = pd.read_csv("creditcard.csv")
dataset
```

Out[28]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5

284807 rows × 31 columns

In [12]:
```
# Preprocess the data (if needed)
# Normalize the data to have a mean of 0 and a standard deviation of 1
scaler = StandardScaler()
X = scaler.fit_transform(dataset.drop("Class", axis=1))
y = dataset["Class"]
```

In [13]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [14]:
```
# Build and train the Autoencoder model
input_dim = X_train.shape[1]
```

In [15]:
```
#c. Encoder converts it into latent representation
```

In [17]:
```
# Encoder
encoder = models.Sequential([
    layers.Input(shape=(input_dim,)),
    layers.Dense(32, activation='relu'),
    layers.Dense(16, activation='relu')
])
```

In [ ]:
```
#d. Decoder networks convert it back to the original input
```

In [18]:
```
# Decoder
decoder = models.Sequential([
    layers.Input(shape=(16,)),
    layers.Dense(32, activation='relu'),
    layers.Dense(input_dim, activation='linear')  # Using 'linear' acti
])

# Autoencoder
autoencoder = models.Sequential([
    encoder,
    decoder
])
```

In [ ]:
```
#e . Compile the models with Optimizer, Loss, and Evaluation Metrics
```

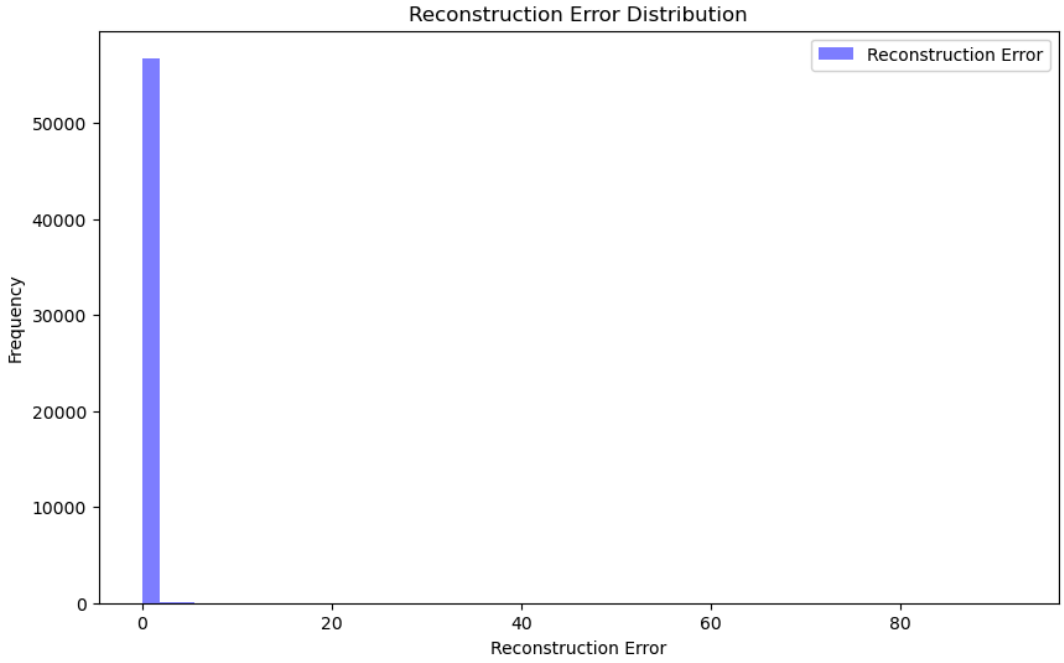
In [19]:
```
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, shuffle=Tru
```

```
WARNING:tensorflow:From C:\Users\shrey\AppData\Roaming\Python\Pyth
on311\site-packages\keras\src\optimizers\__init__.py:309: The name
tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Op
timizer instead.

Epoch 1/10
WARNING:tensorflow:From C:\Users\shrey\AppData\Roaming\Python\Pyth
on311\site-packages\keras\src\utils\tf_utils.py:492: The name tf.r
agged.RaggedTensorValue is deprecated. Please use tf.compat.v1.rag
ged.RaggedTensorValue instead.

7121/7121 [==============================] - 30s 4ms/step - loss:
0.3291 - val_loss: 0.1998
Epoch 2/10
7121/7121 [==============================] - 26s 4ms/step - loss:
0.1791 - val_loss: 0.1571
Epoch 3/10
7121/7121 [==============================] - 26s 4ms/step - loss:
0.1481 - val_loss: 0.1341
```

In [20]:
```
# Detect anomalies and tune the threshold
y_pred = autoencoder.predict(X_test)
mse = np.mean(np.power(X_test - y_pred, 2), axis=1)
```

```
1781/1781 [==============================] - 5s 2ms/step
```

In [21]:
```
# Visualize the reconstruction error distribution
plt.figure(figsize=(10, 6))
plt.hist(mse, bins=50, alpha=0.5, color='b', label='Reconstruction Erro
plt.xlabel("Reconstruction Error")
plt.ylabel("Frequency")
plt.legend()
plt.title("Reconstruction Error Distribution")
plt.show()
```



In [22]:
```
# Threshold tuning (iterate and adjust as needed)
thresholds = np.arange(0.1, 1.0, 0.1)  # Adjust the step size as needed

for threshold in thresholds:
    anomalies = mse > threshold
```

In [30]:
```
# Count the number of anomalies
num_anomalies = np.sum(anomalies)
print(f"Threshold: {threshold:.1f}, Number of anomalies: {num_anomalies
```

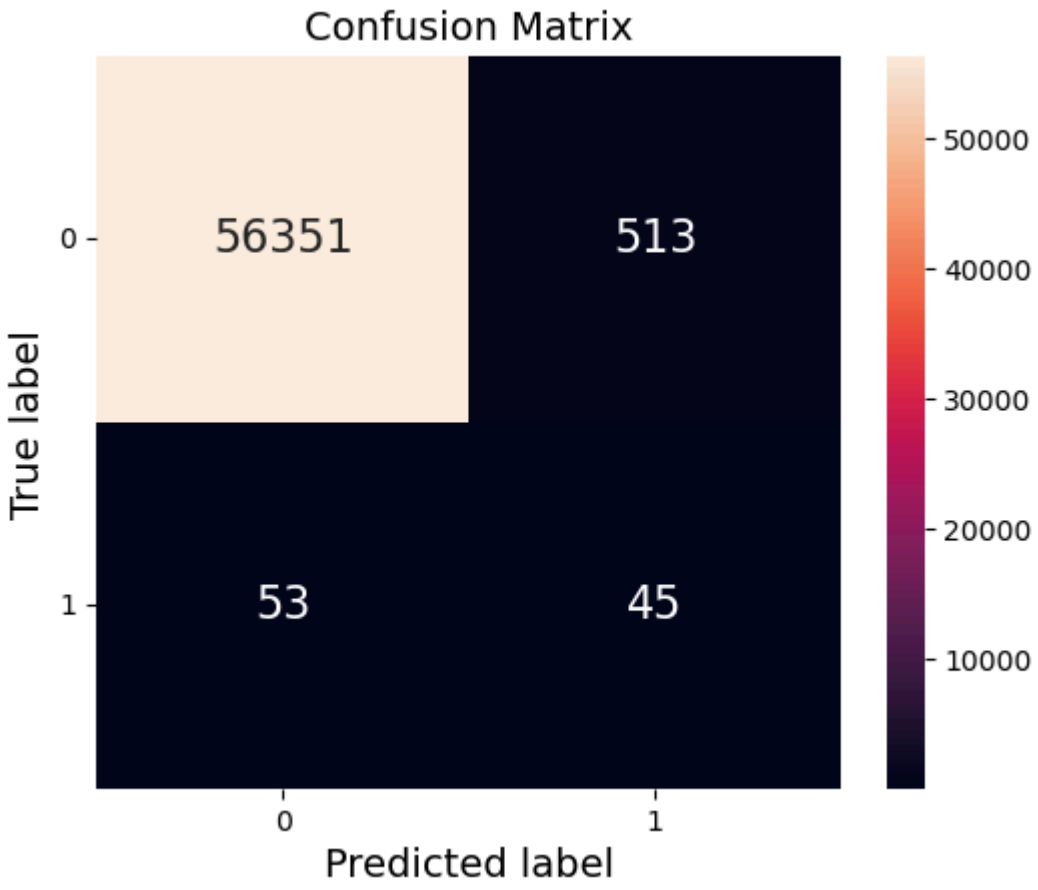
```
Threshold: 0.9, Number of anomalies: 558
```

In [24]:
```
# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, anomalies))

print("\nClassification Report:")
print(classification_report(y_test, anomalies))
```

```
Confusion Matrix:
[[56351   513]
 [   53    45]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56864
           1       0.08      0.46      0.14        98

    accuracy                           0.99     56962
   macro avg       0.54      0.73      0.57     56962
weighted avg       1.00      0.99      0.99     56962
```

In [25]:
```
import seaborn as sns
```

In [26]:
```
plt.figure(figsize = (6, 4.75))
sns.heatmap(confusion_matrix(y_test, anomalies), annot = True, annot_kw
plt.xticks([0.5, 1.5],  rotation = 'horizontal')
plt.yticks([0.5, 1.5],  rotation = 'horizontal')
plt.xlabel("Predicted label", fontsize = 14)
plt.ylabel("True label", fontsize = 14)
plt.title("Confusion Matrix", fontsize = 14)
plt.grid(False)
plt.show()
```



In [ ]:
```

```