```python
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
from sklearn.metrics import confusion_matrix,
classification_report

df = pd.read_csv('../creditcard data/creditcard.csv')
classes = df['Class'].value_counts()
print(classes)
normal_share = round((classes[0]/df['Class'].count()*
100),2)
print("Normal share =", normal_share)
fraud_share = round((classes[1]/df['Class'].count()*
100),2)
print("Fraud share=", fraud_share)
sns.countplot(x='Class', data=df)
plt.title('Number of fraudulent vs non-fraudulent
transactions')
#plt.show()

# Creating fraudulent dataframe
data_fraud = df[df['Class'] == 1]
# Creating non fraudulent dataframe
data_non_fraud = df[df['Class'] == 0]
# Distribution plot
plt.figure(figsize=(8,5))
ax = sns.distplot(data_fraud['Amount'],label='
fraudulent',hist=False)
ax = sns.distplot(data_non_fraud['Amount'],label='non
 fraudulent',hist=False)
ax.set(xlabel='Transaction Amount')
#plt.show()

# Putting feature variables into X
```

```python
X = df.drop(['Class'], axis=1)
# Putting target variable to y
y = df['Class']
# Splitting data into train and test set 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=100)

# Standardization method
# Instantiate the Scaler
scaler = StandardScaler()
# Fit the data into scaler and transform
X_train['Amount'] = scaler.fit_transform(X_train[['Amount']])
print(X_train.head())

# Transform the test set
X_test['Amount'] = scaler.transform(X_test[['Amount']])
print(X_test.head())

# Plotting the distribution of the variables (skewness) of all the columns
cols = X_train.columns
k = 0
plt.figure(figsize=(17, 28))
for col in cols:
 k = k + 1
 plt.subplot(6, 5, k)
 sns.distplot(X_train[col])
 plt.title(col + ' Skewness: ' + str(X_train[col].skew()))
plt.tight_layout()
#plt.show()

# Instantiate the powertransformer
pt = PowerTransformer(method='yeo-johnson', standardize=True,
copy=False)
# Fit and transform the PT on training data
X_train[cols] = pt.fit_transform(X_train)
# Transform the test set
```

```
X_test[cols] = pt.transform(X_test)
# Plotting the distribution of the variables (
skewness) of all the columns
k=0
plt.figure(figsize=(17,28))
for col in cols :
 k=k+1
 plt.subplot(6, 5,k)
 sns.distplot(X_train[col])
 plt.title(col+' '+str(X_train[col].skew()))
#plt.show()

# instantiating the random over sampler
ros = RandomOverSampler()
# resampling X, y
X_train_ros, y_train_ros = ros.fit_resample(X_train
, y_train)
# Before sampling class distribution
print('Before sampling class distribution:-',
Counter(y_train))
# new class distribution
print('New class distribution:-', Counter(
y_train_ros))

input_shape = (X_train_ros.shape[1], 1)
print(input_shape)

inputs = tf.keras.layers.Input(input_shape)
l1 = tf.keras.layers.LSTM(64, activation = 'tanh')(
inputs)
l1 = tf.keras.layers.Dropout(0.20)(l1)
outputs = tf.keras.layers.Dense(1,activation = '
sigmoid')(l1)
model = tf.keras.Model(inputs=[inputs],outputs=[
outputs])
model.compile(optimizer='rmsprop',loss='
binary_crossentropy',metrics=['accuracy'])
model.summary()

checkpoint_dir = '../models'
callbacks = [tf.keras.callbacks.EarlyStopping(
```

```python
monitor='val_loss',min_delta=0.001,patience=20,verbose=1),
            tf.keras.callbacks.ModelCheckpoint(os.path.join(checkpoint_dir, 'Fraud_Detection_Model.keras'),verbose=1,save_best_only=True)]
results = model.fit(X_train_ros.to_numpy().reshape(X_train_ros.shape[0],X_train_ros.shape[1],1),y_train_ros,validation_split = 0.20,epochs=100,callbacks = callbacks)

model.evaluate(X_test.values.reshape(X_test.shape[0],X_test.shape[1],1),y_test,verbose = 1)

# Prediction on the train set
y_pred = model.predict(X_train.to_numpy().reshape(X_train.shape[0],X_train.shape[1],1))
y_pred = (y_pred > 0.5).astype(int)
print("Confusion Matrix:")
con_list = confusion_matrix(y_train,y_pred)
print("\t\tPositive\t Negative\n")
print("Positive\t",con_list[0][0],"(TP)","\t",con_list[0][1],"(FP)\n")
print("Negative\t",con_list[1][0],"(FN)","\t",con_list[1][1],"(TN)\n\n")
print(classification_report(y_train,y_pred))

tp=con_list[0][0]
fp=con_list[0][1]
fn=con_list[1][0]
tn=con_list[1][1]
negative=fp+tn
positive=tp+fn

recall = tp/(tp+fn)
print("recall = tp/(tp+fn) : ", recall)
#precision
precision = tp/(tp+fp)
print("precision = tp/(tp+fp) : ", precision)
#sensitivity
sensitivity = tp/(tp+fn)
print("sensitivity = tp/(tp+fn) : ", sensitivity)
```

```python
#accuracy
accuracy = (tp + tn) / (negative + positive)
print("accuracy = (tp+tn)/(negative + positive) : "
, accuracy * 100)
# f1 score
f1_score = (2*precision*recall)/(precision+recall)
print("f1 score = (2*precision*recall)/(precision+
recall) : ", f1_score)

# Prediction on the test set
y_pred = model.predict(X_test.to_numpy().reshape(
X_test.shape[0],X_test.shape[1],1))
y_pred = (y_pred > 0.5).astype(int)
print("Confusion Matrix:")
con_list = confusion_matrix(y_test,y_pred)
print("\t\tPositive\t Negative\n")
print("Positive\t",con_list[0][0],"(TP)","\t",
con_list[0][1],"(FP)\n")
print("Negative\t",con_list[1][0],"(FN)","\t",
con_list[1][1],"(TN)\n\n")
print(classification_report(y_test,y_pred))

tp=con_list[0][0]
fp=con_list[0][1]
fn=con_list[1][0]
tn=con_list[1][1]
negative=fp+tn
positive=tp+fn

recall = tp/(tp+fn)
print("recall = tp/(tp+fn) : ", recall)
#precision
precision = tp/(tp+fp)
print("precision = tp/(tp+fp) : ", precision)
#sensitivity
sensitivity = tp/(tp+fn)
print("sensitivity = tp/(tp+fn) : ", sensitivity)
#accuracy
accuracy = (tp + tn) / (negative + positive)
print("accuracy = (tp+tn)/(negative + positive) : "
, accuracy * 100)
```

```python
# f1 score
f1_score = (2*precision*recall)/(precision+recall)
print("f1 score = (2*precision*recall)/(precision+recall) : ", f1_score)
```