CODING :

```python
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss


from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,
classification_report,accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import tree
from matplotlib.colors import ListedColormap
import sys
import pandas as pd
DATA='/content/drive/MyDrive/bs140513_032310.csv'
dataset=pd.read_csv(DATA)
dataset.head(8)
ata_found_fraud = dataset.loc[dataset.fraud == 1]
data_non_fraud = dataset.loc[dataset.fraud == 0]
print("FRAUD")
print(data_found_fraud.head(4))
print("NOT FRAUD")
print(data_non_fraud.head(4))

#plotting fraud and non fraud data sets
sns.countplot(x="fraud",data=dataset)
plt.title("counting the no of fraud data in dataset")
plt.rcParams['figure.figsize'] = (6, 6)
plt.show()
#printing no of normal exmples vs the fradulent examples
print("Number of  examples: ",data_non_fraud.fraud.count())
print("Number of fradulent examples: ",data_found_fraud.fraud.count())
mf=data_found_fraud.groupby('category')['amount'].mean()
mnf=data_non_fraud.groupby('category')['amount'].mean()
md=dataset.groupby('category')['fraud'].mean()*100
cmpframe=pd.concat([mf,mnf,md],keys=["fraud","non-
fraud","percentage_fraud"],axis=1)
```

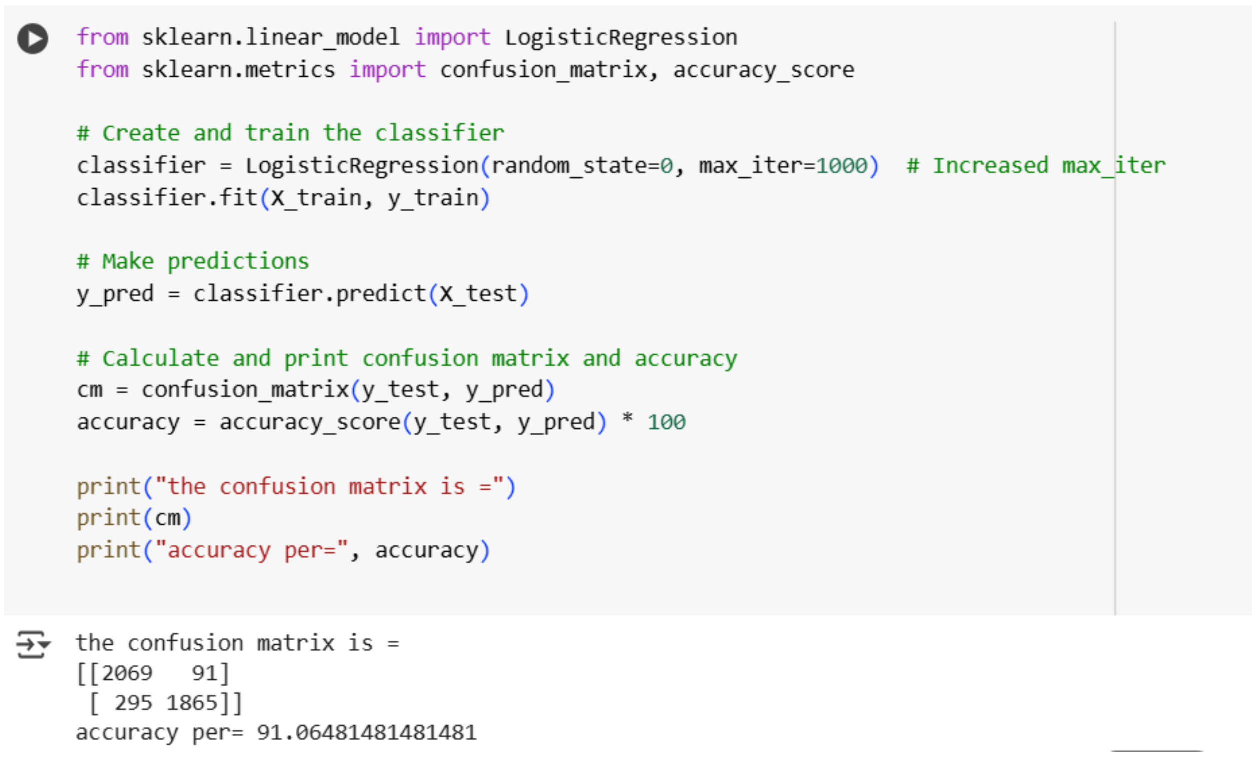
```
cmpframe.sort_values(by="non-fraud")
plt.hist(data_found_fraud.amount,alpha=0.5,label="fraud",bins=100,edgecolor="blac
k")
plt.hist(data_non_fraud.amount,alpha=0.5,label="non-
fraud",bins=100,edgecolor="black")
plt.xlim(0,1000)
plt.ylim(0,10000)

plt.title("Histogram for fradulent and non-fradulent amounts")
plt.legend()
plt.show()

dzip=dataset.zipcodeOri.nunique()
dmer=dataset.zipMerchant.nunique()

# erasing the zipCodeOri and zipMerchant values from the dataset"
dataset = dataset.drop(['zipcodeOri','zipMerchant'],axis=1)
print("Unique zipCodeOri values: ",dzip)
print("Unique zipMerchant values: ",dmer)
#changing datatype of object to categorical
col_cat=dataset.select_dtypes(include='object').columns
for i in col_cat:
  dataset[i]=dataset[i].astype('category')
#changing categorical values to numericals
dataset[col_cat]=dataset[col_cat].apply(lambda i:i.cat.codes)
dataset.head(5)
x=dataset.drop(['fraud'],axis=1)
y=dataset['fraud']
print(x)
print()
print(y)
from imblearn.under_sampling import NearMiss # Importing NearMiss here as well
nr = NearMiss()

# Convert 'customer' column to numerical before applying NearMiss
x['customer'] = pd.to_numeric(x['customer'], errors='coerce')

# Ensure all columns are numerical and handle NaN:
for col in x.columns:
    if x[col].dtype == 'object':
        x[col] = pd.to_numeric(x[col], errors='coerce')
    # Fill NaN with a suitable value (e.g., -1, 0, or the mean)
    x[col] = x[col].fillna(-1).astype('int64') # Filling NaN with -1 before
conversion
```

```
X_train_miss, y_train_miss = nr.fit_resample(x, y)  # changed fit_sample to fit_resample
print('the shape of train_X: {}'.format(X_train_miss.shape))
print('the shape of train_y: {} \n'.format(y_train_miss.shape))
print("counts of label '1': {}".format(sum(y_train_miss == 1)))
print("counts of label '0': {}".format(sum(y_train_miss == 0)))
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score

# Create and train the classifier
classifier = LogisticRegression(random_state=0, max_iter=1000)  # Increased max_iter
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Calculate and print confusion matrix and accuracy
cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred) * 100

print("the confusion matrix is =")
print(cm)
print("accuracy per=", accuracy)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn_cls = KNeighborsClassifier(n_neighbors=5, p=2)
knn_cls.fit(X_train, y_train)
y_pred = knn_cls.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

# Print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy * 100)

from sklearn.tree import DecisionTreeClassifier


classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```python
print("accuracy per=",accuracy_score(y_test,y_pred)*100)
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n_estimators=100,
        max_depth=8,random_state=42,verbose=1,
                     class_weight="balanced")
rf.fit(X_train,y_train)
y_pred=rf.predict(X_test)
print("the confusion matrix is =")
print(cm)
print("accuracy per=",accuracy_score(y_test,y_pred)*100)
#Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_resampled)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_resampled,
test_size=0.3, random_state=42)

# Train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# -----------------------
# ✅ Batch Prediction
# -----------------------
predictions = model.predict(X_test)
output_labels = ["Yes" if pred == 1 else "No" for pred in predictions]
print("Sample batch predictions (Yes = Fraud, No = Not Fraud):")
print(output_labels[:20])  # Show first 20

# -----------------------
# ? Single Transaction Prediction

# Create sample input with all expected features
sample_input = pd.DataFrame([{
    'step': 100,
    'customer': 0,      # Add a dummy or representative value for 'customer'
    'category': 2,      # Number from label encoding
    'amount': 150.0,
    'gender': 1,        # 0 = F, 1 = M
    'age': 3,
    'merchant': 0       # Add a dummy or representative value for 'merchant'
}])
```

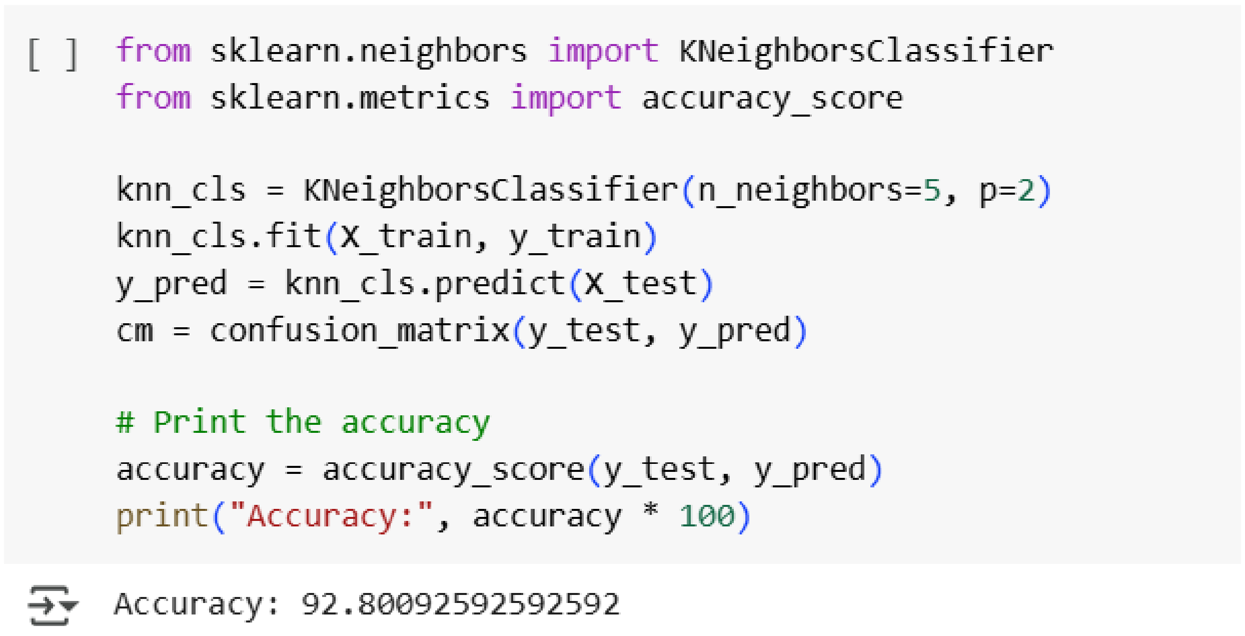
```
# Ensure the columns are in the same order as the training data X_scaled
# Get the column names from X_resampled which was used to fit the scaler
expected_columns = X_resampled.columns.tolist() # Get the actual column order
from X_resampled
sample_input = sample_input[expected_columns]


# Scaling the input
sample_input_scaled = scaler.transform(sample_input)

# Prediction
single_pred = model.predict(sample_input_scaled)[0]
result = "Yes" if single_pred == 1 else "No"
print(f"\nIs the single transaction fraudulent? {result}")
```

OUTPUTS:



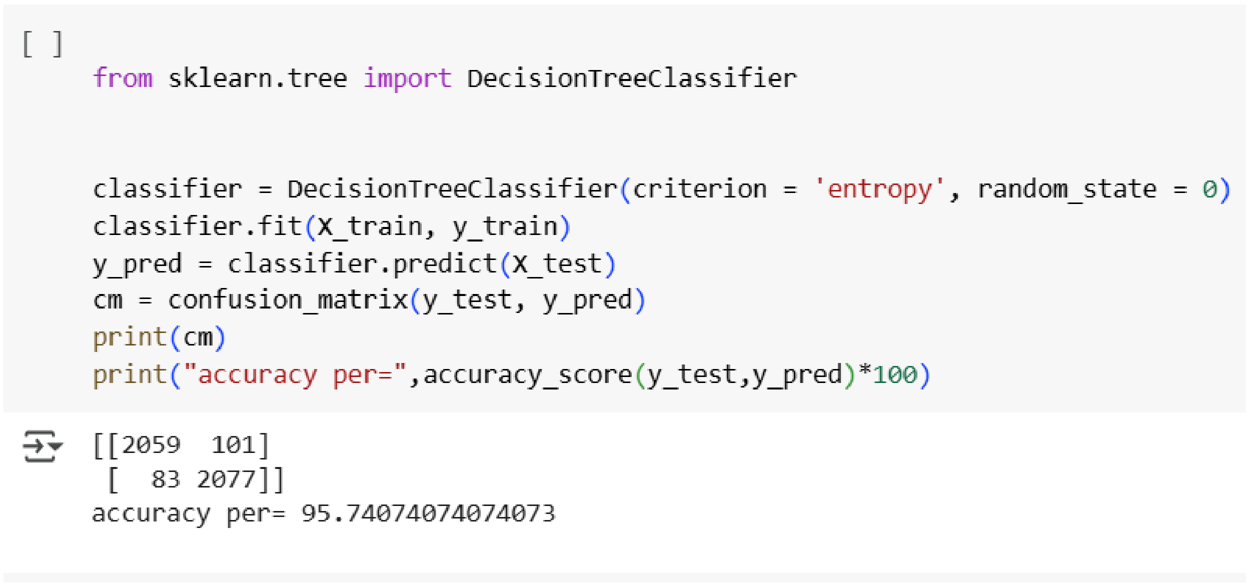
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score

# Create and train the classifier
classifier = LogisticRegression(random_state=0, max_iter=1000)  # Increased max_iter
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Calculate and print confusion matrix and accuracy
cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred) * 100

print("the confusion matrix is =")
print(cm)
print("accuracy per=", accuracy)
```

```
the confusion matrix is =
[[2069   91]
 [ 295 1865]]
accuracy per= 91.06481481481481
```



```
[ ]  from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score

     knn_cls = KNeighborsClassifier(n_neighbors=5, p=2)
     knn_cls.fit(X_train, y_train)
     y_pred = knn_cls.predict(X_test)
     cm = confusion_matrix(y_test, y_pred)

     # Print the accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print("Accuracy:", accuracy * 100)
```

```
Accuracy: 92.80092592592592
```



```
[ ]
     from sklearn.tree import DecisionTreeClassifier


     classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
     classifier.fit(X_train, y_train)
     y_pred = classifier.predict(X_test)
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     print("accuracy per=",accuracy_score(y_test,y_pred)*100)
```

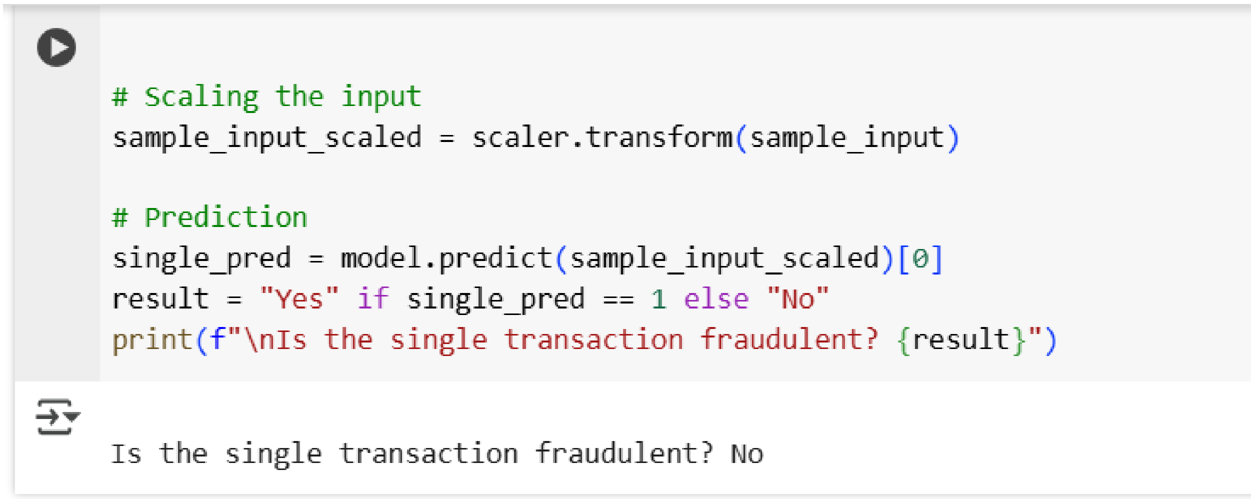
```
[[2059  101]
 [  83 2077]]
accuracy per= 95.74074074074073
```

```
[ ]
    from sklearn.tree import DecisionTreeClassifier


    classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print("accuracy per=",accuracy_score(y_test,y_pred)*100)
```

```
[[2059  101]
 [  83 2077]]
accuracy per= 95.74074074074073
```

```
[ ] from sklearn.ensemble import RandomForestClassifier

    rf=RandomForestClassifier(n_estimators=100,
            max_depth=8,random_state=42,verbose=1,
                          class_weight="balanced")
    rf.fit(X_train,y_train)
    y_pred=rf.predict(X_test)
    print("the confusion matrix is =")
    print(cm)
    print("accuracy per=",accuracy_score(y_test,y_pred)*100)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.3s
the confusion matrix is =
[[2059  101]
 [  83 2077]]
accuracy per= 96.66666666666667
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.7s finished
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```



```python
# Scaling the input
sample_input_scaled = scaler.transform(sample_input)

# Prediction
single_pred = model.predict(sample_input_scaled)[0]
result = "Yes" if single_pred == 1 else "No"
print(f"\nIs the single transaction fraudulent? {result}")
```

```
Is the single transaction fraudulent? No
```