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

In [2]:
```
#loading data
dataframe=pd.read_csv("onlinefraud.csv")
```

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

Out[3]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFra
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	

In [4]:
```
#check for null value
dataframe.isnull()
```

Out[4]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	is
0	False	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	
6362615	False	False	False	False	False	False	False	False	False	False	
6362616	False	False	False	False	False	False	False	False	False	False	
6362617	False	False	False	False	False	False	False	False	False	False	
6362618	False	False	False	False	False	False	False	False	False	False	
6362619	False	False	False	False	False	False	False	False	False	False	

6362620 rows × 11 columns

In [5]:
```
#methods to get columns in dataset
```

In [6]:
```
dataframe.columns
```

Out[6]:
```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
       'isFlaggedFraud'],
      dtype='object')
```

In [7]:
```
dataframe.columns.values
```

Out[7]:
```
array(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg',
       'newbalanceOrig', 'nameDest', 'oldbalanceDest', 'newbalanceDest',
       'isFraud', 'isFlaggedFraud'], dtype=object)
```

In [8]:
```
list(dataframe)
```

Out[8]:
```
['step',
 'type',
 'amount',
 'nameOrig',
 'oldbalanceOrg',
 'newbalanceOrig',
 'nameDest',
 'oldbalanceDest',
 'newbalanceDest',
 'isFraud',
 'isFlaggedFraud']
```

In [9]:
```
#getting to know about data
dataframe.info
```

Out[9]:
```
<bound method DataFrame.info of          step      type      amount     nameOrig  oldbalanceOrg  \
0           1   PAYMENT     9839.64  C1231006815      170136.00
1           1   PAYMENT     1864.28  C1666544295       21249.00
2           1  TRANSFER      181.00  C1305486145         181.00
3           1  CASH_OUT      181.00   C840083671         181.00
4           1   PAYMENT    11668.14  C2048537720       41554.00
...       ...       ...         ...          ...            ...
6362615   743  CASH_OUT   339682.13   C786484425      339682.13
6362616   743  TRANSFER  6311409.28  C1529008245     6311409.28
6362617   743  CASH_OUT  6311409.28  C1162922333     6311409.28
6362618   743  TRANSFER   850002.52  C1685995037      850002.52
6362619   743  CASH_OUT   850002.52  C1280323807      850002.52

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
...                 ...          ...             ...             ...      ...
6362615            0.00   C776919290            0.00       339682.13        1
6362616            0.00  C1881841831            0.00            0.00        1
6362617            0.00  C1365125890        68488.84      6379898.11        1
6362618            0.00  C2080388513            0.00            0.00        1
6362619            0.00   C873221189      6510099.11      7360101.63        1

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
...                 ...
6362615               0
6362616               0
6362617               0
6362618               0
6362619               0

[6362620 rows x 11 columns]>
```

In [10]:
```
dataframe.describe()
```

Out[10]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFrau
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+0
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06	1.290820e-03	2.514687e-0
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.590480e-02	1.585775e-0
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05	0.000000e+00	0.000000e+0
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06	0.000000e+00	0.000000e+0
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08	1.000000e+00	1.000000e+0

In [11]:
```
col=list(dataframe)
```

In [12]:
```
col
```

Out[12]:
```
['step',
 'type',
 'amount',
 'nameOrig',
 'oldbalanceOrg',
 'newbalanceOrig',
 'nameDest',
 'oldbalanceDest',
 'newbalanceDest',
 'isFraud',
 'isFlaggedFraud']
```

In [13]:
```
dataframe.isnull().sum()
```

Out[13]:
```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

In [14]:
```
#check for duplicate values
dataframe.duplicated()
```

Out[14]:
```
0          False
1          False
2          False
3          False
4          False
           ...
6362615    False
6362616    False
6362617    False
6362618    False
6362619    False
Length: 6362620, dtype: bool
```

In [15]:
```
dataframe.duplicated().sum()
```

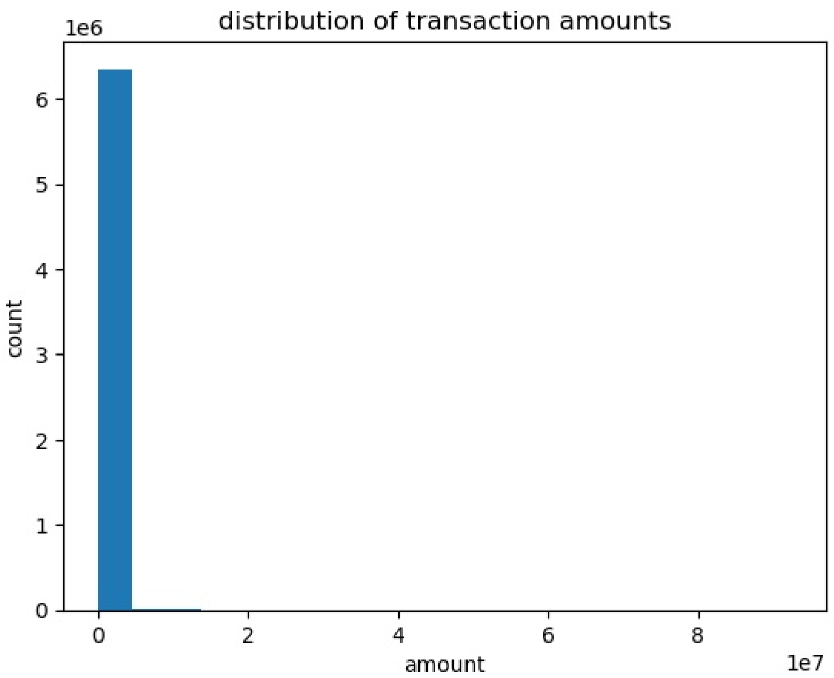
Out[15]:
```
0
```

Exploratory Data Analysis

In [16]:
```
#univariate analysis
```

In [17]:
```
#distrubution of transaction money
plt.hist(dataframe['amount'],bins=20)
plt.xlabel('amount')
plt.ylabel('count')
plt.title("distribution of transaction amounts")
```

Out[17]:
```
Text(0.5, 1.0, 'distribution of transaction amounts')
```

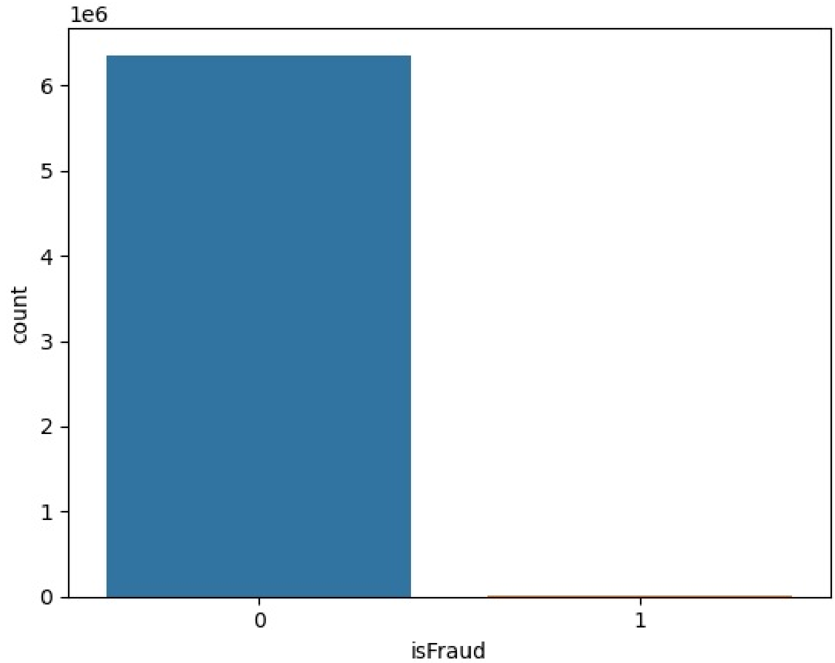


In [18]:
```
dataframe['isFraud'].value_counts()
#this will tell that in a column , what is the frequency of occurance of 1 and 0
#0 = not fraud; 1 = isfraud
```

Out[18]:
```
isFraud
0    6354407
1       8213
Name: count, dtype: int64
```

In [19]:
```
sns.countplot(x='isFraud',data=dataframe)
```

```
#this will show how much isFraud col is divided into its categories
#count shows count of occurances
plt.show()
```



In [20]:
```
#now break down transaction's that has been done
#as done above for isFraud col, do same for type of transactions col, and find occurances of each category
```

In [21]:
```
dataframe["type"].values #values of type col
```

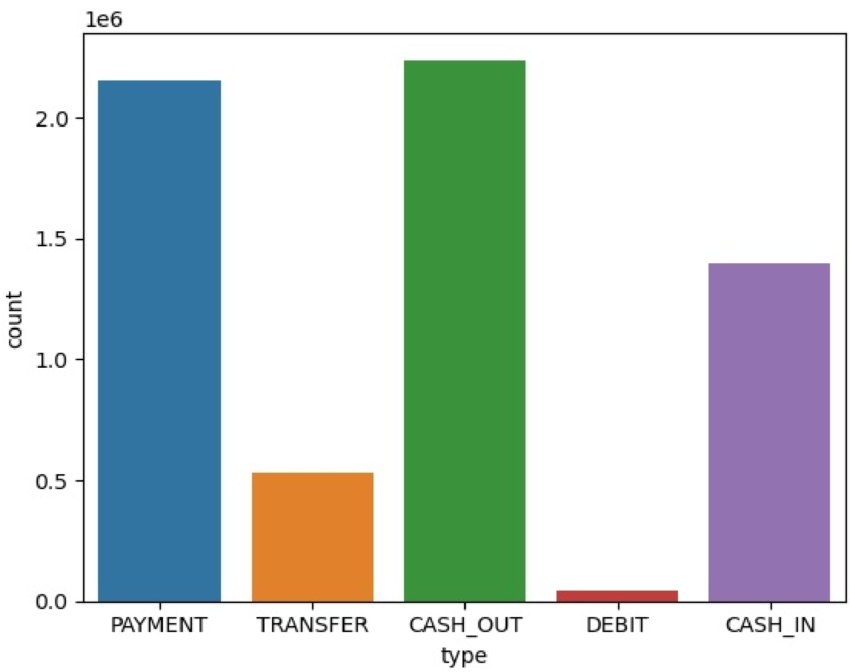
Out[21]:
```
array(['PAYMENT', 'PAYMENT', 'TRANSFER', ..., 'CASH_OUT', 'TRANSFER',
       'CASH_OUT'], dtype=object)
```

In [22]:
```
dataframe["type"].value_counts()
```

Out[22]:
```
type
CASH_OUT    2237500
PAYMENT     2151495
CASH_IN     1399284
TRANSFER     532909
DEBIT         41432
Name: count, dtype: int64
```

In [23]:
```
sns.countplot(x="type",data=dataframe)
```

Out[23]:
```
<Axes: xlabel='type', ylabel='count'>
```

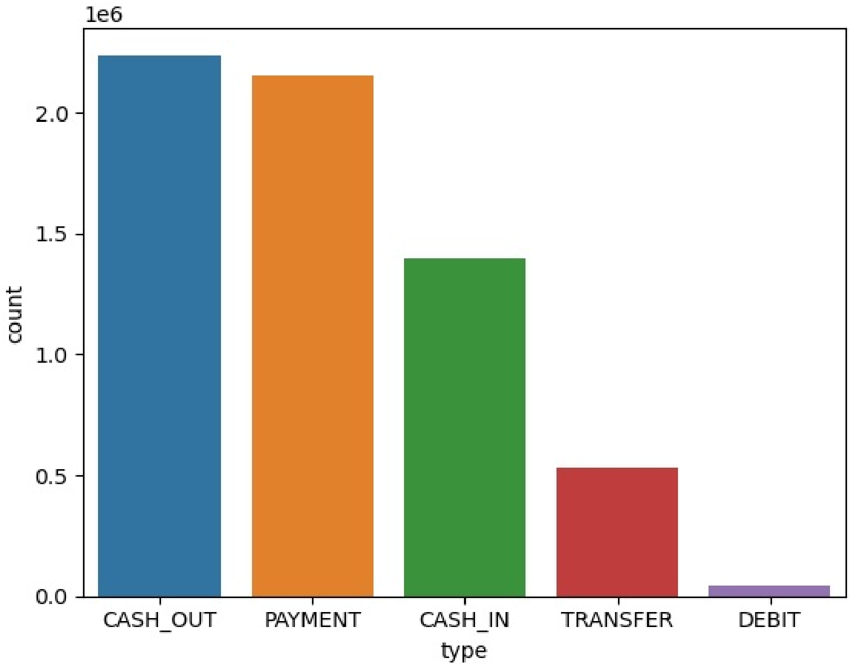


In [24]:
```
dataframe['type'].value_counts().index #list of unique values in decresing order
```

Out[24]:
```
Index(['CASH_OUT', 'PAYMENT', 'CASH_IN', 'TRANSFER', 'DEBIT'], dtype='object', name='type')
```

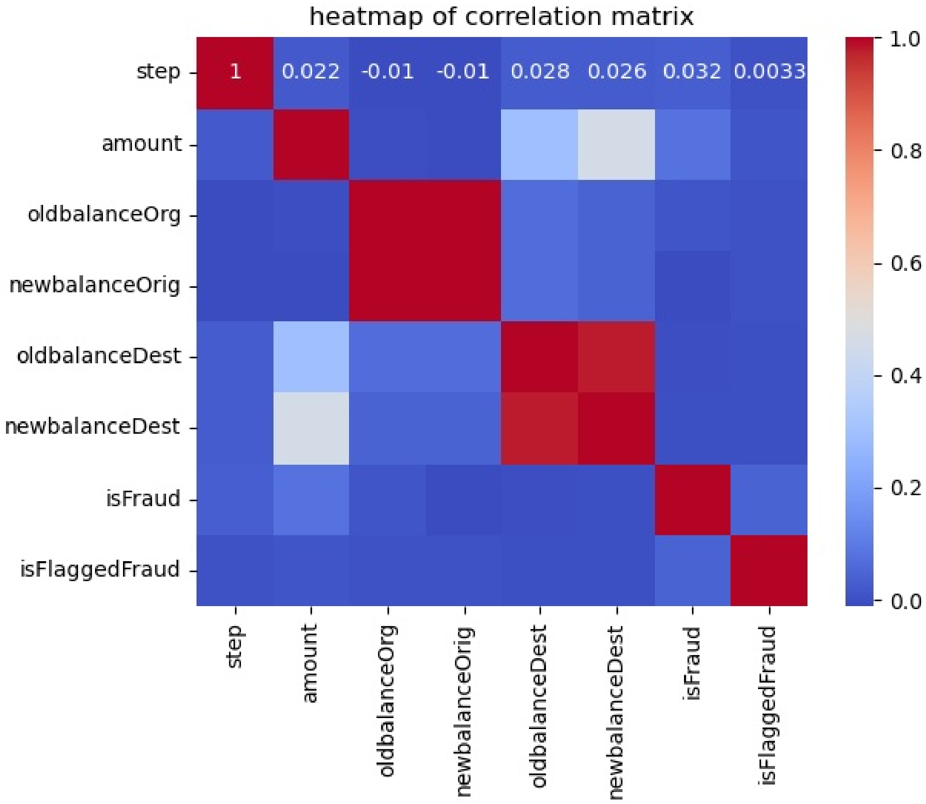
In [25]:
```
sns.countplot(x='type',data=dataframe,order=dataframe['type'].value_counts().index)
```

Out[25]: <Axes: xlabel='type', ylabel='count'>



In [26]:
```
#bi & multi variate
```

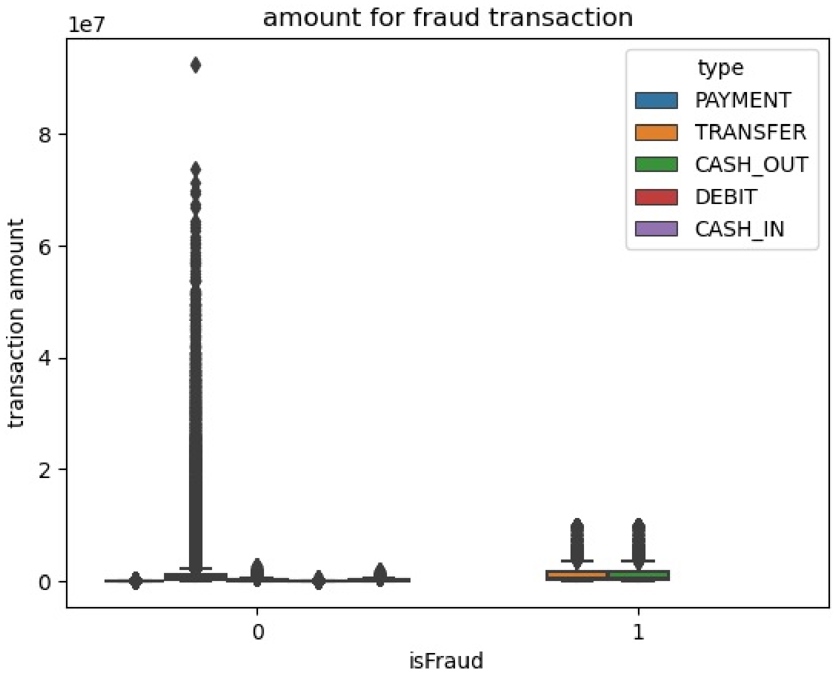
In [27]:
```
#heat map is used to get correlation b/w each numeric variable to each other
#we use this for quickly identify strong and weak correlations,dealing with large datasets,detect multi colline
correlation=dataframe.select_dtypes('number').corr()
sns.heatmap(correlation,cmap='coolwarm',annot=True)
plt.title('heatmap of correlation matrix')
plt.show()
```



In [28]:
```
list(dataframe)
```

Out[28]:
```
['step',
 'type',
 'amount',
 'nameOrig',
 'oldbalanceOrg',
 'newbalanceOrig',
 'nameDest',
 'oldbalanceDest',
 'newbalanceDest',
 'isFraud',
 'isFlaggedFraud']
```

In [29]:
```
#boxplot used to distribute conti. variable acc. to different categories, isfraud is category and transaction ar
sns.boxplot(x='isFraud',y='amount',data=dataframe,hue='type')
plt.xlabel('isFraud')
plt.ylabel('transaction amount')
plt.title('amount for fraud transaction')
plt.show()
#this graph will show that cash_out and transfer are the commonly used payment types
#used by scammers
```



Feature Engineering

As we have categorical data in the dataset and machine learning algorithms perform on only numerical values, we need to convert our categorical data into numerical format. For that we would perform One Hot encoding to convert categorical data to that format which can be fed as an input to an algorithm

In [30]:
```
#one hot encoding
encoded_data=pd.get_dummies(dataframe,columns=['type'],prefix='type')
encoded_data.head()
```

Out[30]:

	step	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlagge
0	1	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	
1	1	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	
2	1	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	
3	1	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	
4	1	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	

In [31]:
```
list(encoded_data)
```

Out[31]:
```
['step',
 'amount',
 'nameOrig',
 'oldbalanceOrg',
 'newbalanceOrig',
 'nameDest',
 'oldbalanceDest',
 'newbalanceDest',
 'isFraud',
 'isFlaggedFraud',
 'type_CASH_IN',
 'type_CASH_OUT',
 'type_DEBIT',
 'type_PAYMENT',
 'type_TRANSFER']
```

In [32]:
```
target_var=dataframe['isFraud'] #select target variable
#target varible is what we aim to predict
```

In [33]:
```
target_var
```

Out[33]:
```
0          0
1          0
2          1
3          1
4          0
          ..
6362615    1
6362616    1
6362617    1
6362618    1
6362619    1
Name: isFraud, Length: 6362620, dtype: int64
```

In [34]:
```
#select feature columns
feature_col=['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest','type_CASH_IN','type_C
feature_var=encoded_data[feature_col]
#feature variable are attributes of data that we use to make predictions
```

In [35]:
```
feature_var.head()
```

Out[35]:

	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	type_CASH_IN	type_CASH_OUT	type_DEBIT	type_
0	9839.64	170136.0	160296.36	0.0	0.0	False	False	False	
1	1864.28	21249.0	19384.72	0.0	0.0	False	False	False	
2	181.00	181.0	0.00	0.0	0.0	False	False	False	
3	181.00	181.0	0.00	21182.0	0.0	False	True	False	
4	11668.14	41554.0	29885.86	0.0	0.0	False	False	False	

Splitting of dataset into test and train in 5:5 ratio

In [36]:
```
from sklearn.model_selection import train_test_split
```

In [37]:
```
x_train,x_test,y_train,y_test=train_test_split(feature_var,target_var,test_size=0.5,random_state=41)
```

In [38]:
```
print('x_train',x_train.shape)
print('x_test',x_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)
```

```
x_train (3181310, 10)
x_test (3181310, 10)
y_train (3181310,)
y_test (3181310,)
```

Analysis of Algorithm

Random Forest

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,f1_score
```

In [40]:
```
random_forest_model=RandomForestClassifier(n_estimators=100)
```

In [41]:
```
random_forest_model.fit(x_train,y_train)
```

Out[41]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [42]:
```
random_forest_pred=random_forest_model.predict(x_test)
```

In [43]:
```
print(y_test.head().tolist())
```

```
[0, 0, 0, 0, 0]
```

In [44]:
```
print(random_forest_pred[:5])
```

```
[0 0 0 0 0]
```

as we are dealing with the binary classification we will be using accuracy score of both the algorithms to see which one is more effective. We can also use other measures like precision , F1-score,etc.

In [45]:
```
print(accuracy_score(random_forest_pred,y_test)*100)
```

```
99.96674325985208
```

Logistic Regression

In [46]:
```
from sklearn.linear_model import LogisticRegression
```

In [47]:
```
logistic_model= LogisticRegression()
```

In [48]:
```
logistic_model.fit(x_train,y_train)
```

Out[48]:

```
▾ LogisticRegression
LogisticRegression()
```

In [49]:
```
logistic_model_pred=logistic_model.predict(x_test)
```

In [50]:
```
print(y_test.head().tolist())
```

```
[0, 0, 0, 0, 0]
```

In [51]:
```
print(logistic_model_pred[:5])
```

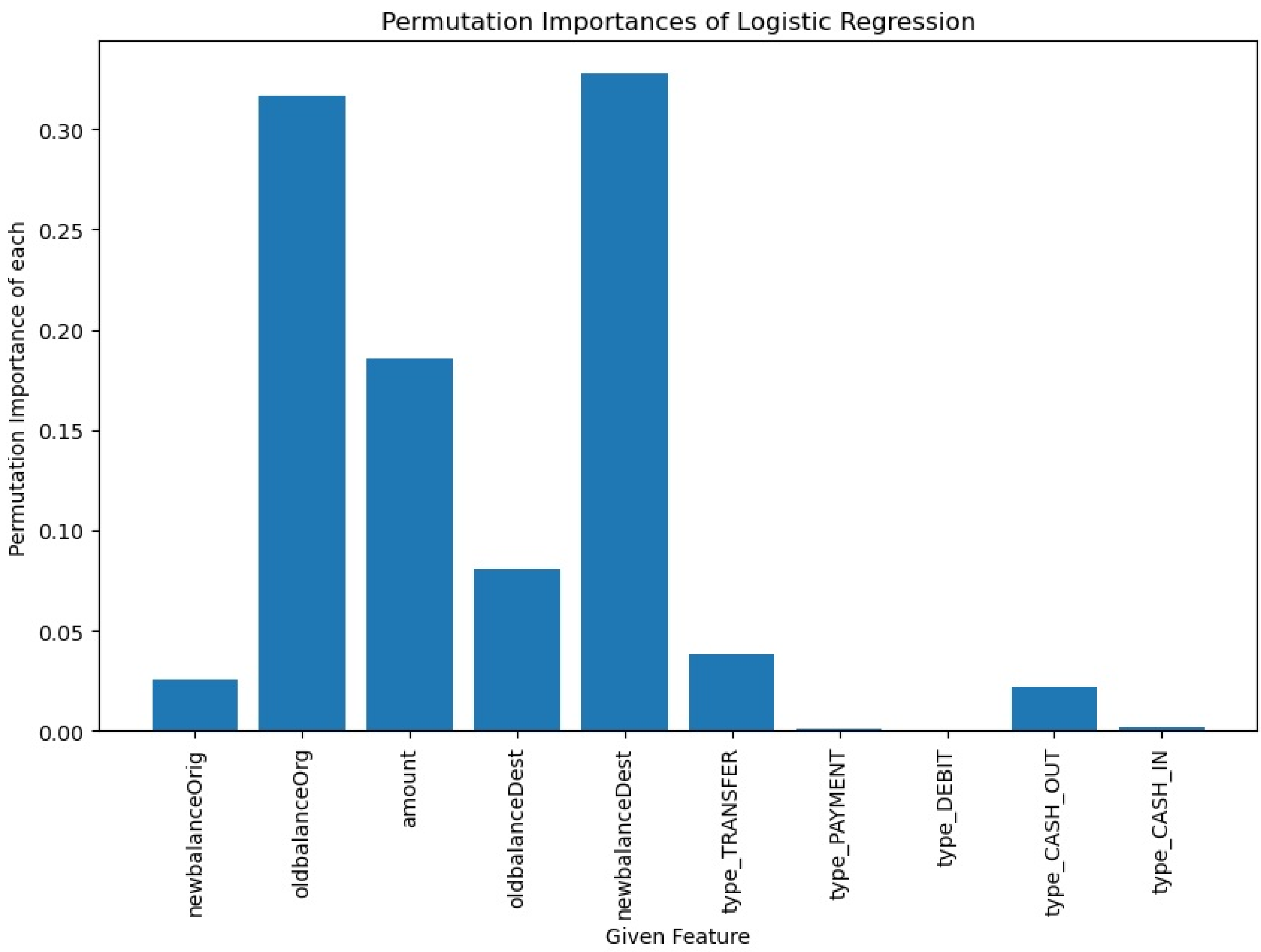
```
[0 0 0 0 0]
```

Permutation Importance of Logistic Regression

In [65]:
```
from sklearn.inspection import permutation_importance
```

In [66]:
```
# Compute permutation importance
per_result = permutation_importance(logistic_model, x_train, y_train, n_repeats=10, random_state=42)
feature_imp_permutation = per_result.importances_mean
ind=np.argsort(feature_imp_permutation)[::-1]
feature_name=x_train.columns
#n_repeats - controls the number of times each feature is shuffled and evaluated.
#random_state parameter is used to set the random seed for reproducibility
```

In [67]:
```
#now use visualization
plt.figure(figsize=(10,6))
plt.bar(range(x_train.shape[1]),important[ind])
plt.xticks(range(x_train.shape[1]), feature_name[ind], rotation='vertical')
plt.xlabel('Given Feature')
plt.ylabel('Permutation Importance of each')
plt.title('Permutation Importances of Logistic Regression')
plt.show()
```



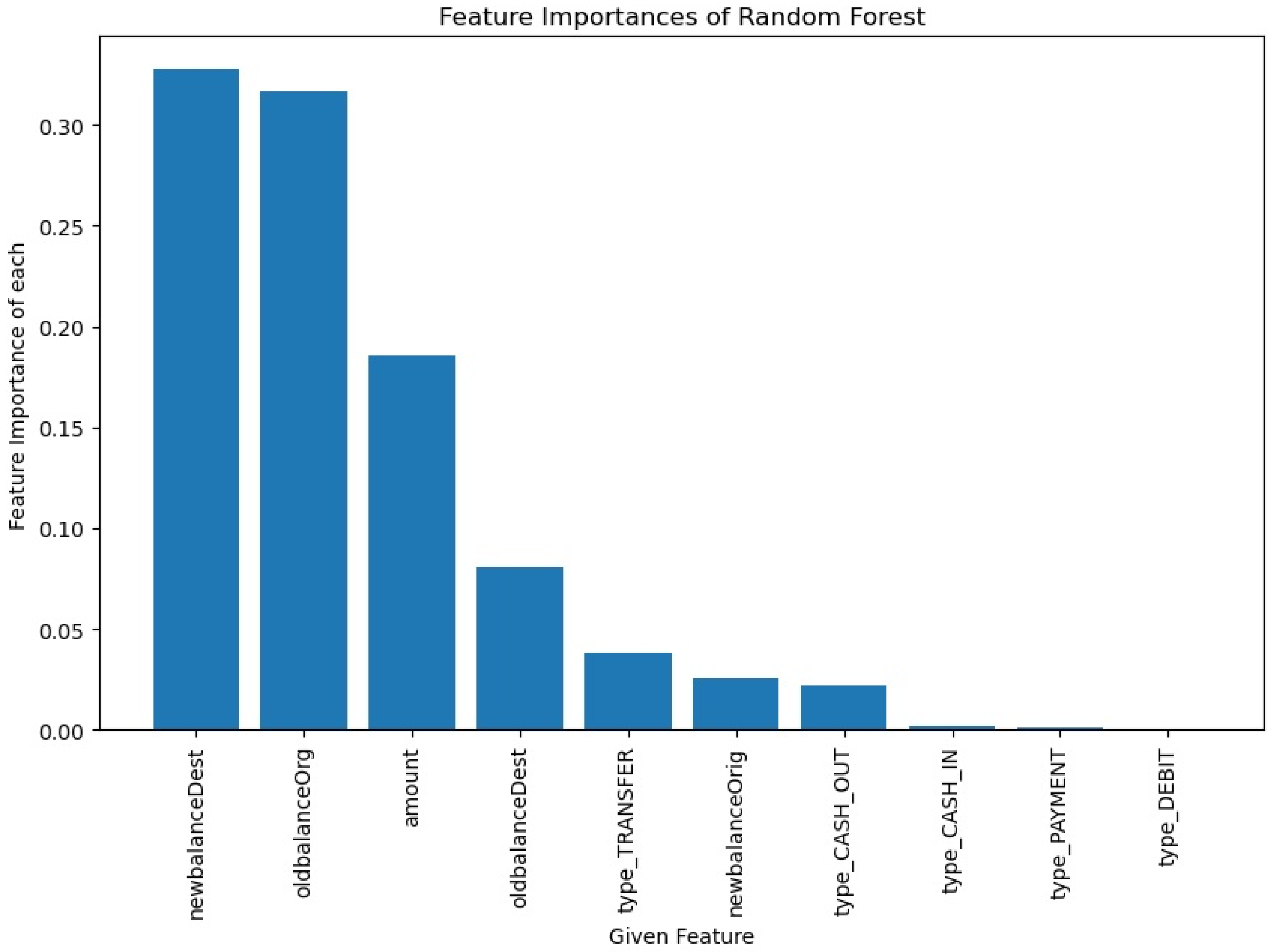
Evaluation Metrics of logistic Regression

In [68]:
```
#getting accuracy, precision f1 score for logistic regression
logistic_accuracy=accuracy_score(y_test,logistic_model_pred)*100
logistic_precision=precision_score(y_test,logistic_model_pred)*100
logistic_f1=f1_score(y_test,logistic_model_pred)*100
print("logistic regression")
print("Accuracy",logistic_accuracy)
print("Precision",logistic_precision)
print("F1-score",logistic_f1)
print()
```

```
logistic regression
Accuracy 99.77166638900327
Precision 34.17876866434854
F1-score 47.920848867221096
```

feature importance for random forest model

In [52]:
```
#we perform this to see the importance of various features
import matplotlib .pyplot as plt
important= random_forest_model.feature_importances_
#this function will give list of feature importance score for each feature
indices=np.argsort(important)[::-1]
feature_name=x_train.columns
#now use visualization
plt.figure(figsize=(10,6))
plt.bar(range(x_train.shape[1]),important[indices])
plt.xticks(range(x_train.shape[1]), feature_name[indices], rotation='vertical')
plt.xlabel('Given Feature')
plt.ylabel('Feature Importance of each')
plt.title('Feature Importances of Random Forest')
plt.show()
```



Classification Metrices of Random Forest

In [64]:
```
#getting accuracy, precision f1 score for random forest
random_f_accuracy=accuracy_score(y_test,random_forest_pred)*100
random_f_precision=precision_score(y_test,random_forest_pred)*100
random_f_f1=f1_score(y_test,random_forest_pred)*100
print("Random Forest")
print("Accuracy",random_f_accuracy)
print("Precision",random_f_precision)
print("F1-score",random_f_f1)
print()
```

```
Random Forest
Accuracy 99.96674325985208
Precision 97.15151515151516
F1-score 85.83668005354752
```

Decision Tress

In [53]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [54]:
```
tree_classifier_model = DecisionTreeClassifier(criterion='gini', max_depth=None)
```

In [55]:
```
tree_classifier_model.fit(x_train,y_train)
```

Out[55]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [56]:
```
tree_classifier_model_pred=tree_classifier_model.predict(x_test)
```

In [57]:
```
print(y_test.head().tolist())
```

```
[0, 0, 0, 0, 0]
```

In [58]:
```
print(tree_classifier_model_pred[:5])
```

```
[0 0 0 0 0]
```

In [59]:
```
print(accuracy_score(tree_classifier_model_pred,y_test)*100)
```

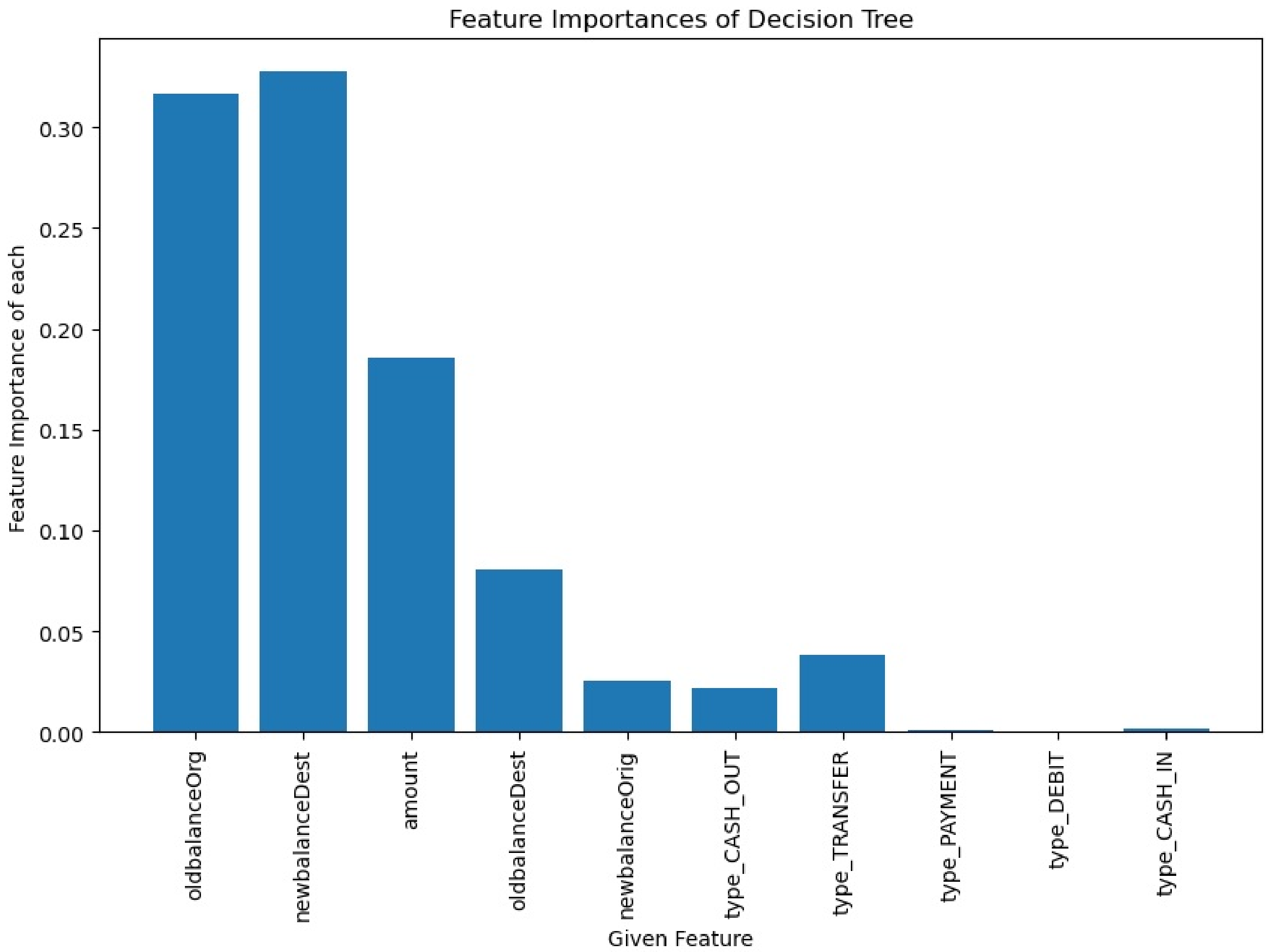
```
99.9710182283399
```

Feature Importance of Decision Tree

In [60]:
```
#we perform this to see the importance of various features
import matplotlib .pyplot as plt
imp_tree= tree_classifier_model.feature_importances_
```

In [61]:
```
#this function will give list of feature importance score for each feature
ind_tree=np.argsort(imp_tree)[::-1]
feature_name=x_train.columns
```

In [62]:
```
#now use visualization
plt.figure(figsize=(10,6))
plt.bar(range(x_train.shape[1]),important[ind_tree])
plt.xticks(range(x_train.shape[1]), feature_name[ind_tree], rotation='vertical')
plt.xlabel('Given Feature')
plt.ylabel('Feature Importance of each')
plt.title('Feature Importances of Decision Tree')
plt.show()
```



Classification Metrices of Decision tree

In [69]:
```
#getting accuracy, precision f1 score for random forest
tree_f_accuracy=accuracy_score(y_test,tree_classifier_model_pred)*100
tree_f_precision=precision_score(y_test,tree_classifier_model_pred)*100
tree_f_f1=f1_score(y_test,tree_classifier_model_pred)*100
print("Decision Tree")
print("Accuracy",tree_f_accuracy)
print("Precision",tree_f_precision)
print("F1-score",tree_f_f1)
print()
```

```
Decision Tree
Accuracy 99.9710182283399
Precision 89.9017199017199
F1-score 88.81067961165049
```

Naive Bayes

In [70]:
```
from sklearn.naive_bayes import GaussianNB
```

In [71]:
```
nb_classifier_model = GaussianNB()
```

In [72]:
```
nb_classifier_model.fit(x_train,y_train)
```
Out[72]:
```
▾ GaussianNB
GaussianNB()
```

In [73]:
```
nb_classifier_model_pred=nb_classifier_model.predict(x_test)
```

In [74]:
```
print(y_test.head().tolist())
```
```
[0, 0, 0, 0, 0]
```

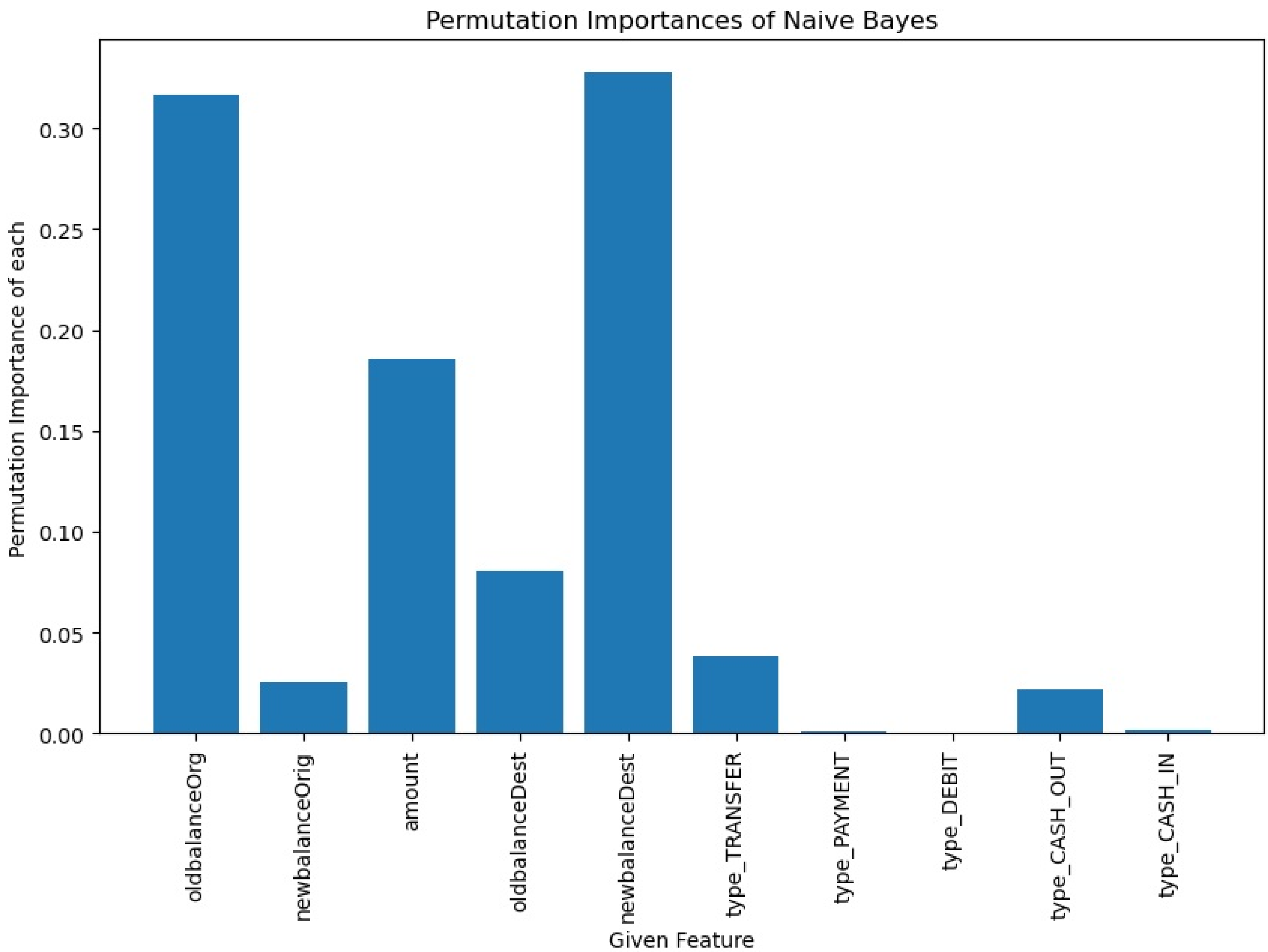
In [75]:
```
print(nb_classifier_model_pred[:5])
```
```
[0 0 0 0 0]
```

In [77]:
```
print(accuracy_score(nb_classifier_model_pred,y_test)*100)
```
```
99.52727021258538
```

Feature Importance as a Permutation Importance for Naive Bayes

In [78]:
```
# Compute permutation importance
Naive_imp = permutation_importance(nb_classifier_model, x_train, y_train)
naive_imp_permutation = Naive_imp.importances_mean
ind_naive=np.argsort(naive_imp_permutation)[::-1]
feature_name=x_train.columns
```

In [79]:
```
#now use visualization
plt.figure(figsize=(10,6))
plt.bar(range(x_train.shape[1]),important[ind_naive])
plt.xticks(range(x_train.shape[1]), feature_name[ind_naive], rotation='vertical')
plt.xlabel('Given Feature')
plt.ylabel('Permutation Importance of each')
plt.title('Permutation Importances of Naive Bayes')
plt.show()
```



Classification Metrices of Nave Bayes

In [82]:
```
#getting accuracy, precision f1 score for naive bayes
naive_f_accuracy=accuracy_score(y_test,nb_classifier_model_pred)*100
naive_f_precision=precision_score(y_test,nb_classifier_model_pred)*100
naive_f_f1=f1_score(y_test,nb_classifier_model_pred)*100
```

```
print("Naive Bayes")
print("Accuracy",naive_f_accuracy)
print("Precision",naive_f_precision)
print("F1-score",naive_f_f1)
print()
```

```
Naive Bayes
Accuracy 99.52727021258538
Precision 5.712655855268519
F1-score 8.527461833221823
```

Comparison of all models

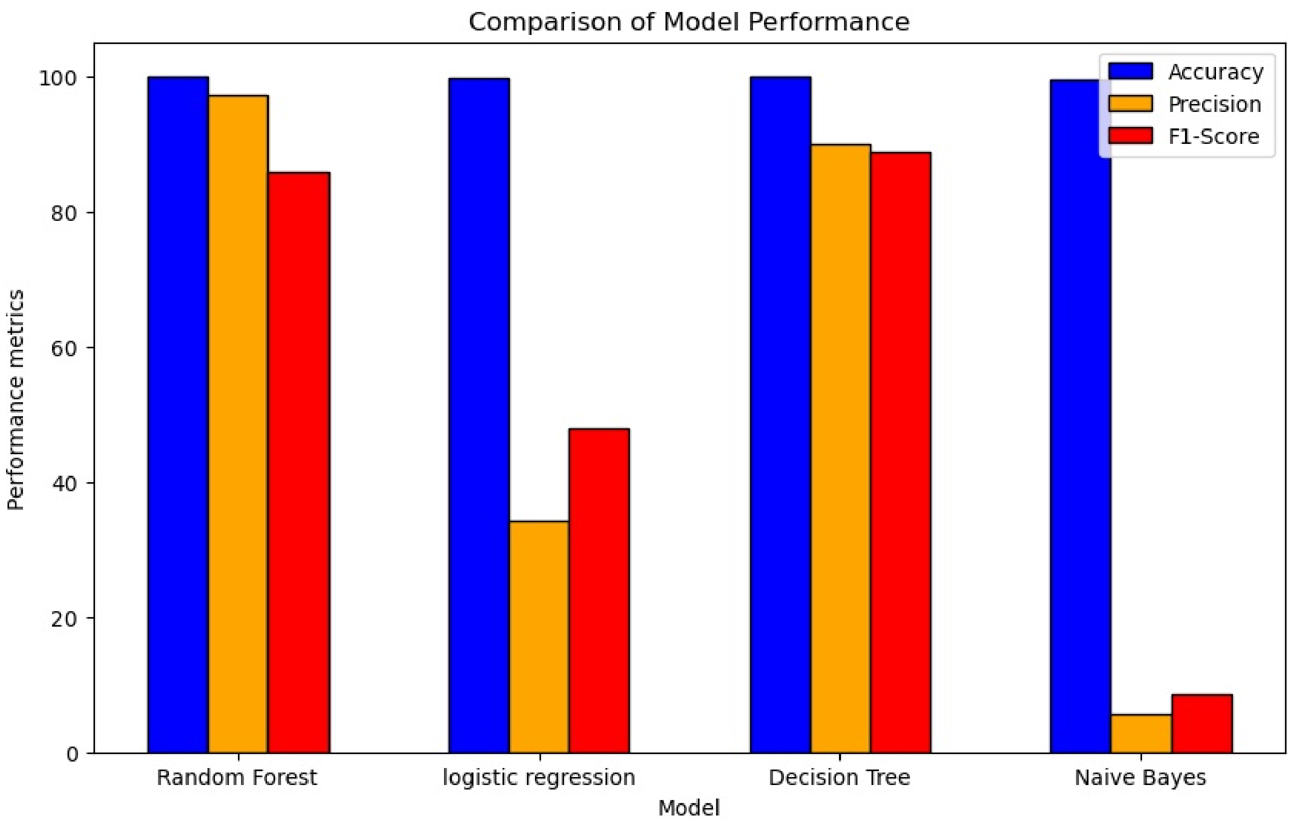
In [83]:
```
#comparing performance metrics of classification models
models=['Random Forest','logistic regression','Decision Tree','Naive Bayes']
b_width=0.35 #bar width
a1=np.arange(len(models))#bar position on x axis
random_forest_values=[random_f_accuracy,random_f_precision,random_f_f1]
logistic_model_values=[logistic_accuracy,logistic_precision,logistic_f1]
tree_model_values=[tree_f_accuracy,tree_f_precision,tree_f_f1]
naive_model_values=[naive_f_accuracy,naive_f_precision,naive_f_f1]
```

In [84]:
```
models=['Random Forest','logistic regression','Decision Tree','Naive Bayes']
b_width=0.2 #bar width
accuracies=[random_f_accuracy,logistic_accuracy,tree_f_accuracy,naive_f_accuracy]
precision=[random_f_precision,logistic_precision,tree_f_precision,naive_f_precision]
f1_score=[random_f_f1,logistic_f1,tree_f_f1,naive_f_f1]

a1 = np.arange(len(models))
a2 = [i + b_width for i in a1]
a3 = [i + b_width for i in a2]


plt.figure(figsize=(10, 6))
plt.bar(a1, accuracies, color='blue', width=b_width, edgecolor='black', label='Accuracy')
plt.bar(a2, precision, color='orange', width=b_width, edgecolor='black', label='Precision')
plt.bar(a3, f1_score, color='red', width=b_width, edgecolor='black', label='F1-Score')
plt.xticks([a + b_width for a in range(len(models))], models)


plt.xlabel('Model')
plt.ylabel('Performance metrics')
plt.title('Comparison of Model Performance')
plt.legend()
plt.show()
```



From the observed output, we have derived the following insights

Random Forest Accuracy 99.96674325985208 Precision 97.0659407138536 F1-score 85.84804708400215 logistic regression Accuracy 99.77166638900327 Precision 34.17876866434854 F1-score 47.920848867221096

Naive Bayes Accuracy 99.52727021258538 Precision 5.712655855268519 F1-score 8.527461833221823

Decision Tree Accuracy 99.9710182283399 Precision 89.9017199017199 F1-score 88.81067961165049

From above we can see accuracy,precision and f1-score of classification models, and on giving insights based upon accuracy of all we found that Decision Tree can predict fraudent data more accurately than any other algorithms. Naive Bayes has the lowest accuracy among all