Credit Card Fraud Detection

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_curve, roc_auc_score, auc
import warnings
warnings.filterwarnings('ignore')
```

Reading the Train and Test Datasets

In [3]:
```
df_train = pd.read_csv(r"C:\Users\HP\PGA 32\Machine Learning\CODSOFT\Credit Card Fraud Detecion\fraudTra
```

In [4]:
```
1 df_train.head()
```

Out[4]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	...	3
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Greens Suite 393	...	4
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale Suite 530	...	4
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jeremy	White	M	9443 Cynthia Court Apt. 038	...	4
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	M	408 Bradley Rest	...	3

5 rows × 23 columns

In [5]:
```
1 df_test = pd.read_csv(r"C:\Users\HP\PGA 32\Machine Learning\CODSOFT\Credit Card Fraud Detecion\fraudTest
```

In [6]:
```
1 df_test.head()
```

Out[6]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	Jeff	Elliott	M	351 Darlene Green	...
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne	Williams	F	3638 Marsh Union	...
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ashley	Lopez	F	9333 Valentine Point	...
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	Brian	Williams	M	32941 Krystal Mill Apt. 552	...
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	Nathan	Massey	M	5783 Evan Roads Apt. 465	...

5 rows × 23 columns

In [7]:
```
1 df_train.shape
```

Out[7]: (1296675, 23)

In [8]:
```
1 df_test.shape
```

Out[8]: (555719, 23)

In [9]:
```
1 df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype  
---  ------                 --------------    -----  
 0   Unnamed: 0             1296675 non-null  int64  
 1   trans_date_trans_time  1296675 non-null  object 
 2   cc_num                 1296675 non-null  int64  
 3   merchant               1296675 non-null  object 
 4   category               1296675 non-null  object 
 5   amt                    1296675 non-null  float64
 6   first                  1296675 non-null  object 
 7   last                   1296675 non-null  object 
 8   gender                 1296675 non-null  object 
 9   street                 1296675 non-null  object 
 10  city                   1296675 non-null  object 
 11  state                  1296675 non-null  object 
 12  zip                    1296675 non-null  int64  
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64  
 16  job                    1296675 non-null  object 
 17  dob                    1296675 non-null  object 
 18  trans_num              1296675 non-null  object 
 19  unix_time              1296675 non-null  int64  
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64  
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

In [10]:
```
1 df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   Unnamed: 0             555719 non-null  int64  
 1   trans_date_trans_time  555719 non-null  object 
 2   cc_num                 555719 non-null  int64  
 3   merchant               555719 non-null  object 
 4   category               555719 non-null  object 
 5   amt                    555719 non-null  float64
 6   first                  555719 non-null  object 
 7   last                   555719 non-null  object 
 8   gender                 555719 non-null  object 
 9   street                 555719 non-null  object 
 10  city                   555719 non-null  object 
 11  state                  555719 non-null  object 
 12  zip                    555719 non-null  int64  
 13  lat                    555719 non-null  float64
 14  long                   555719 non-null  float64
 15  city_pop               555719 non-null  int64  
 16  job                    555719 non-null  object 
 17  dob                    555719 non-null  object 
 18  trans_num              555719 non-null  object 
 19  unix_time              555719 non-null  int64  
 20  merch_lat              555719 non-null  float64
 21  merch_long             555719 non-null  float64
 22  is_fraud               555719 non-null  int64  
dtypes: float64(5), int64(6), object(12)
memory usage: 97.5+ MB
```

In [11]:
```
1 # df_train.drop(columns=['Unnamed: 0'],inplace=True)
2 # df_train.head()
```

In [12]:
```
# df_test.drop(columns=['Unnamed: 0'],inplace=True)
# df_test.head()
```

In [13]:
```
df_train.describe()
```

Out[13]:

	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop	unix_time	merch_la
count	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+0
mean	6.483370e+05	4.171920e+17	7.035104e+01	4.880067e+04	3.853762e+01	-9.022634e+01	8.882444e+04	1.349244e+09	3.853734e+0
std	3.743180e+05	1.308806e+18	1.603160e+02	2.689322e+04	5.075808e+00	1.375908e+01	3.019564e+05	1.284128e+07	5.109788e+0
min	0.000000e+00	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+01	1.325376e+09	1.902779e+0
25%	3.241685e+05	1.800429e+14	9.650000e+00	2.623700e+04	3.462050e+01	-9.679800e+01	7.430000e+02	1.338751e+09	3.473357e+0
50%	6.483370e+05	3.521417e+15	4.752000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.456000e+03	1.349250e+09	3.936568e+0
75%	9.725055e+05	4.642255e+15	8.314000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+04	1.359385e+09	4.195716e+0
max	1.296674e+06	4.992346e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.795030e+01	2.906700e+06	1.371817e+09	6.751027e+0

In [14]:
```
df_test.describe()
```

Out[14]:

	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop	unix_time	me
count	555719.000000	5.557190e+05	555719.000000	555719.000000	555719.000000	555719.000000	5.557190e+05	5.557190e+05	555719
mean	277859.000000	4.178387e+17	69.392810	48842.628015	38.543253	-90.231325	8.822189e+04	1.380679e+09	38
std	160422.401459	1.309837e+18	156.745941	26855.283328	5.061336	13.721780	3.003909e+05	5.201104e+06	5
min	0.000000	6.041621e+10	1.000000	1257.000000	20.027100	-165.672300	2.300000e+01	1.371817e+09	19
25%	138929.500000	1.800429e+14	9.630000	26292.000000	34.668900	-96.798000	7.410000e+02	1.376029e+09	34
50%	277859.000000	3.521417e+15	47.290000	48174.000000	39.371600	-87.476900	2.408000e+03	1.380762e+09	39
75%	416788.500000	4.635331e+15	83.010000	72011.000000	41.894800	-80.175200	1.968500e+04	1.385867e+09	41
max	555718.000000	4.992346e+18	22768.110000	99921.000000	65.689900	-67.950300	2.906700e+06	1.388534e+09	66

Duplicate Values

In [15]:
```
1 df_train.duplicated().sum()
```

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

In [16]:
```
1 df_test.duplicated().sum()
```

Out[16]:
```
0
```

Numerical Columns

In [17]:
```
1 df_train.dtypes[df_train.dtypes!="object"]
```

Out[17]:
```
Unnamed: 0      int64
cc_num          int64
amt           float64
zip             int64
lat           float64
long          float64
city_pop        int64
unix_time       int64
merch_lat     float64
merch_long    float64
is_fraud        int64
dtype: object
```

In [18]:
```
1 df_test.dtypes[df_test.dtypes!="object"]
```

Out[18]:
```
Unnamed: 0      int64
cc_num          int64
amt           float64
zip             int64
lat           float64
long          float64
city_pop        int64
unix_time       int64
merch_lat     float64
merch_long    float64
is_fraud        int64
dtype: object
```

Categorical Columns

In [19]:
```
1 df_train.dtypes[df_train.dtypes=="object"]
```

Out[19]:
```
trans_date_trans_time    object
merchant                 object
category                 object
first                    object
last                     object
gender                   object
street                   object
city                     object
state                    object
job                      object
dob                      object
trans_num                object
dtype: object
```

In [20]:
```
1 df_test.dtypes[df_test.dtypes=="object"]
```

Out[20]:
```
trans_date_trans_time    object
merchant                 object
category                 object
first                    object
last                     object
gender                   object
street                   object
city                     object
state                    object
job                      object
dob                      object
trans_num                object
dtype: object
```

Missing Value Treatment

In [21]:
```
1 df_train.isnull().sum()
```

Out[21]:
```
Unnamed: 0              0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

In [22]:
```
1 df_test.isna().sum()
```

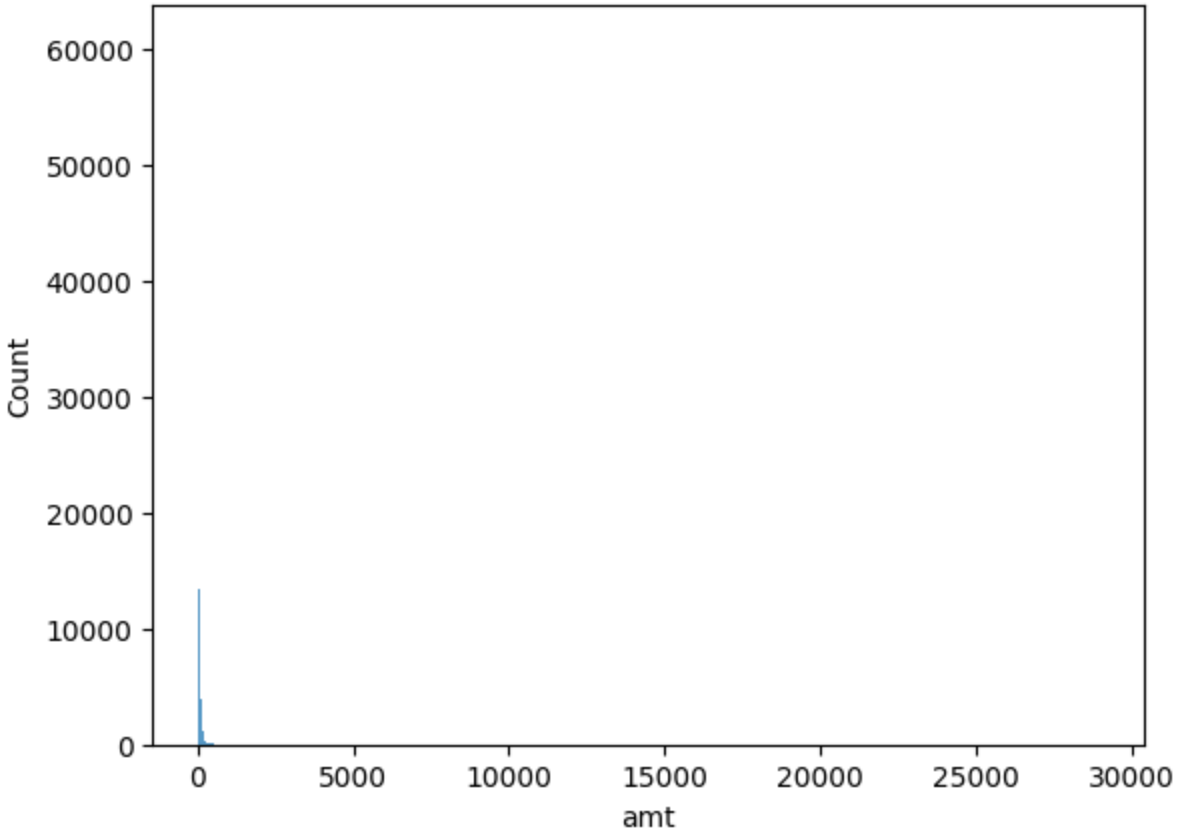
Out[22]:
```
Unnamed: 0              0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

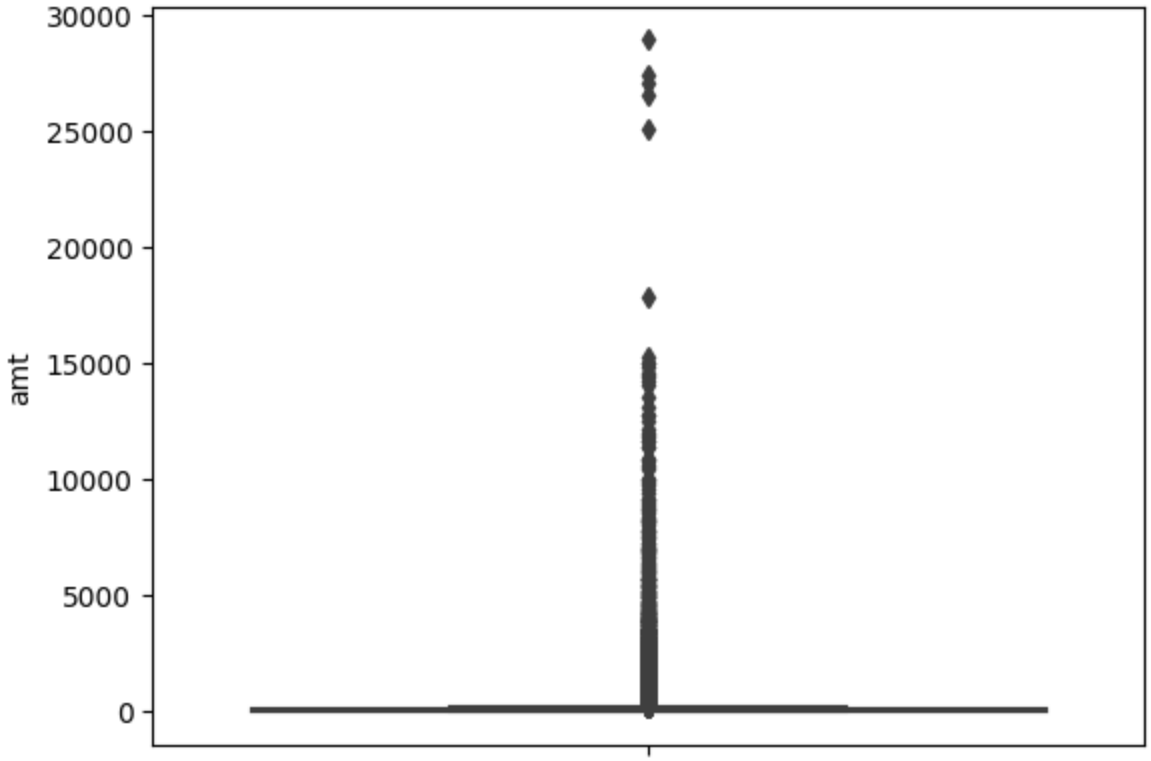
Visualizing minimum value, maximum value, variance, standard deviation, 1st, 10th, 25th, 50th, 75th, 90th, 99th Histogram and Barplot graphs of percentiles, interquartile range (IQR) of train data

In [23]:

```
def numerical(data,var,graph_plot=True):
    min_n=data[var].min()
    max_n=data[var].max()
    var_n=data[var].var()
    std_n=data[var].std()
    p1=data[var].quantile(.01)
    p10=data[var].quantile(.1)
    p25=data[var].quantile(.25)
    p50=data[var].quantile(.5)
    p75=data[var].quantile(.75)
    p90=data[var].quantile(.9)
    p99=data[var].quantile(.99)
    iqr=p75-p25

    if graph_plot==True:
        sns.histplot(data[var])
        plt.show()
        sns.boxplot(y=data[var])
        plt.show()

    results={"min":min_n,"max":max_n,"var":var_n,"std":std_n,
            "p1":p1,"p10":p10,"p25":p25,"p50":p50,"p75":p75,"p90":p90,"p99":p99}
    return results
```

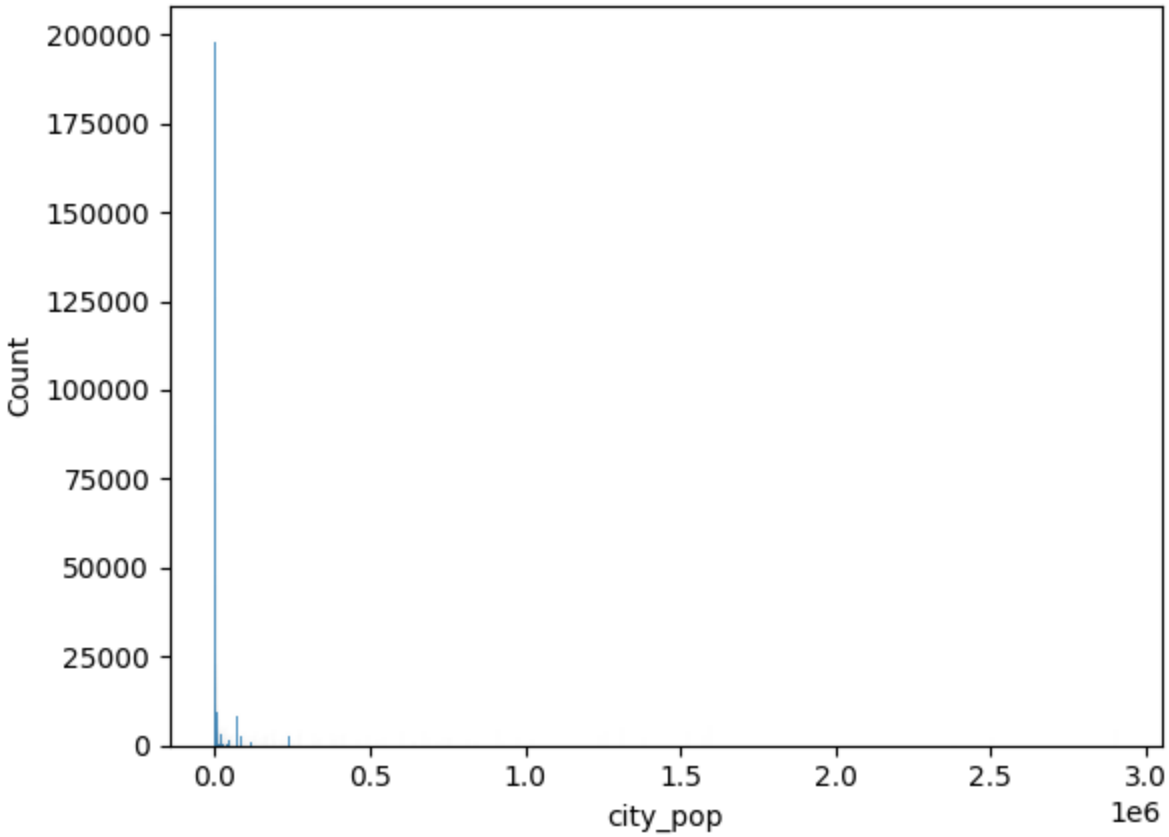
In [24]:
```
1 numerical(data=df_train, var="amt")
```

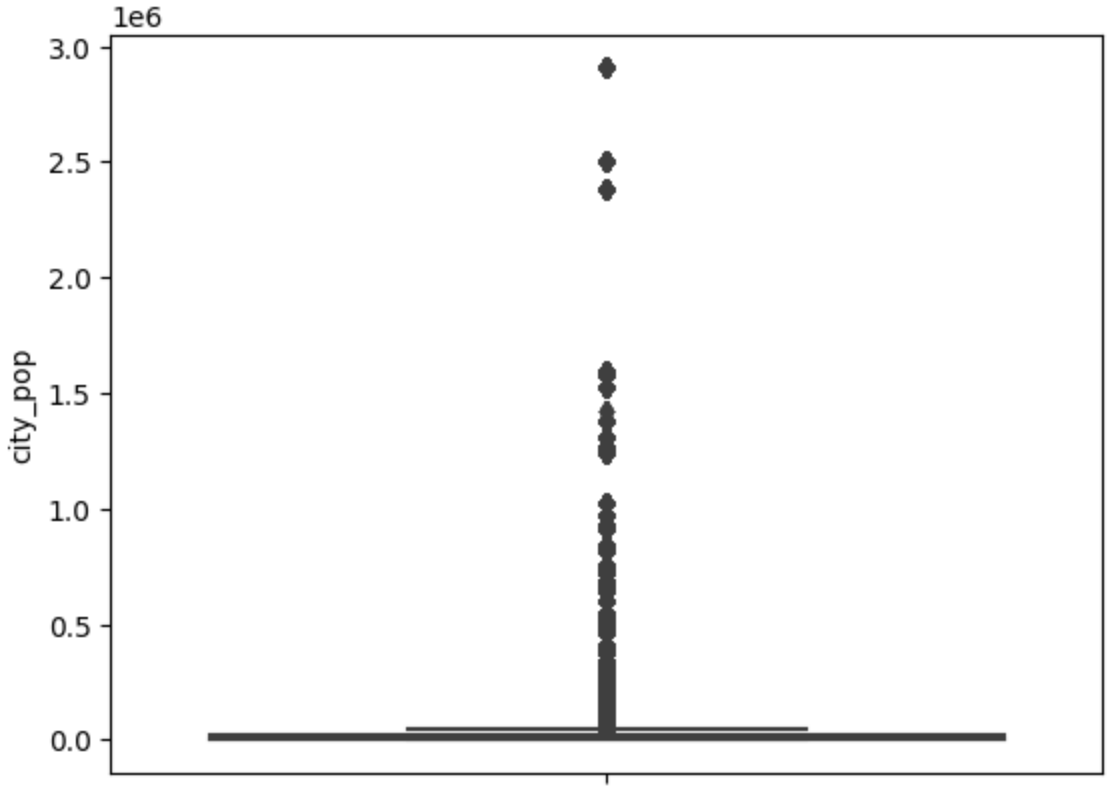




Out[24]:
```
{'min': 1.0,
 'max': 28948.9,
 'var': 25701.232223266023,
 'std': 160.3160385715229,
 'p1': 1.26,
 'p10': 4.11,
 'p25': 9.65,
 'p50': 47.52,
 'p75': 83.14,
 'p90': 136.67,
 'p99': 545.9926000000002}
```

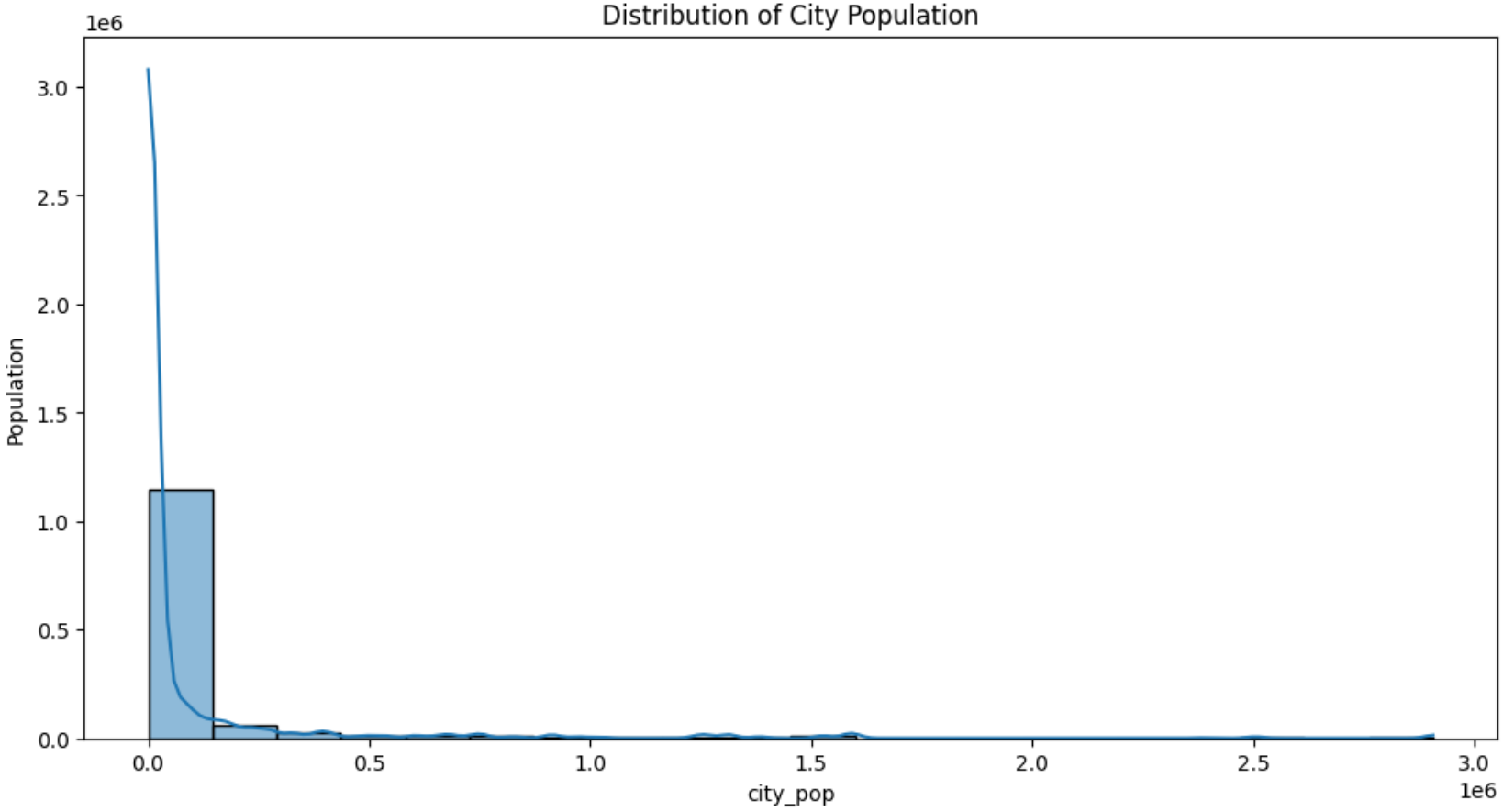
In [25]:
```
1 numerical(data=df_train, var="city_pop")
```



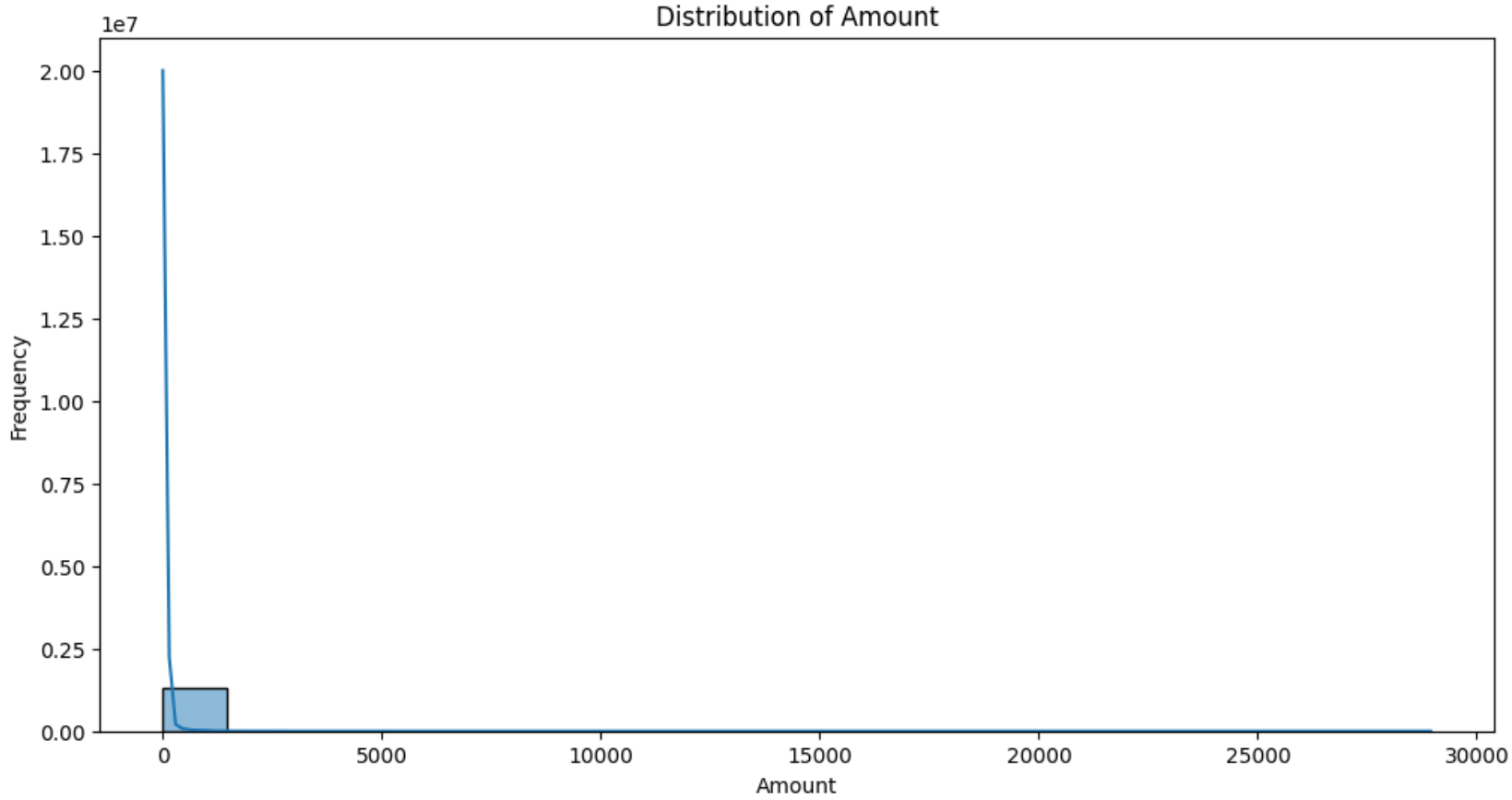


Out[25]:
```
{'min': 23,
 'max': 2906700,
 'var': 91177643760.35764,
 'std': 301956.360688689,
 'p1': 53.0,
 'p10': 260.0,
 'p25': 743.0,
 'p50': 2456.0,
 'p75': 20328.0,
 'p90': 186140.0,
 'p99': 1577385.0}
```

In [26]:
```
plt.figure(figsize=(12, 6))
sns.histplot(df_train['city_pop'], bins=20, kde=True)
plt.xlabel('city_pop')
plt.ylabel('Population')
plt.title('Distribution of City Population')
plt.show()
```

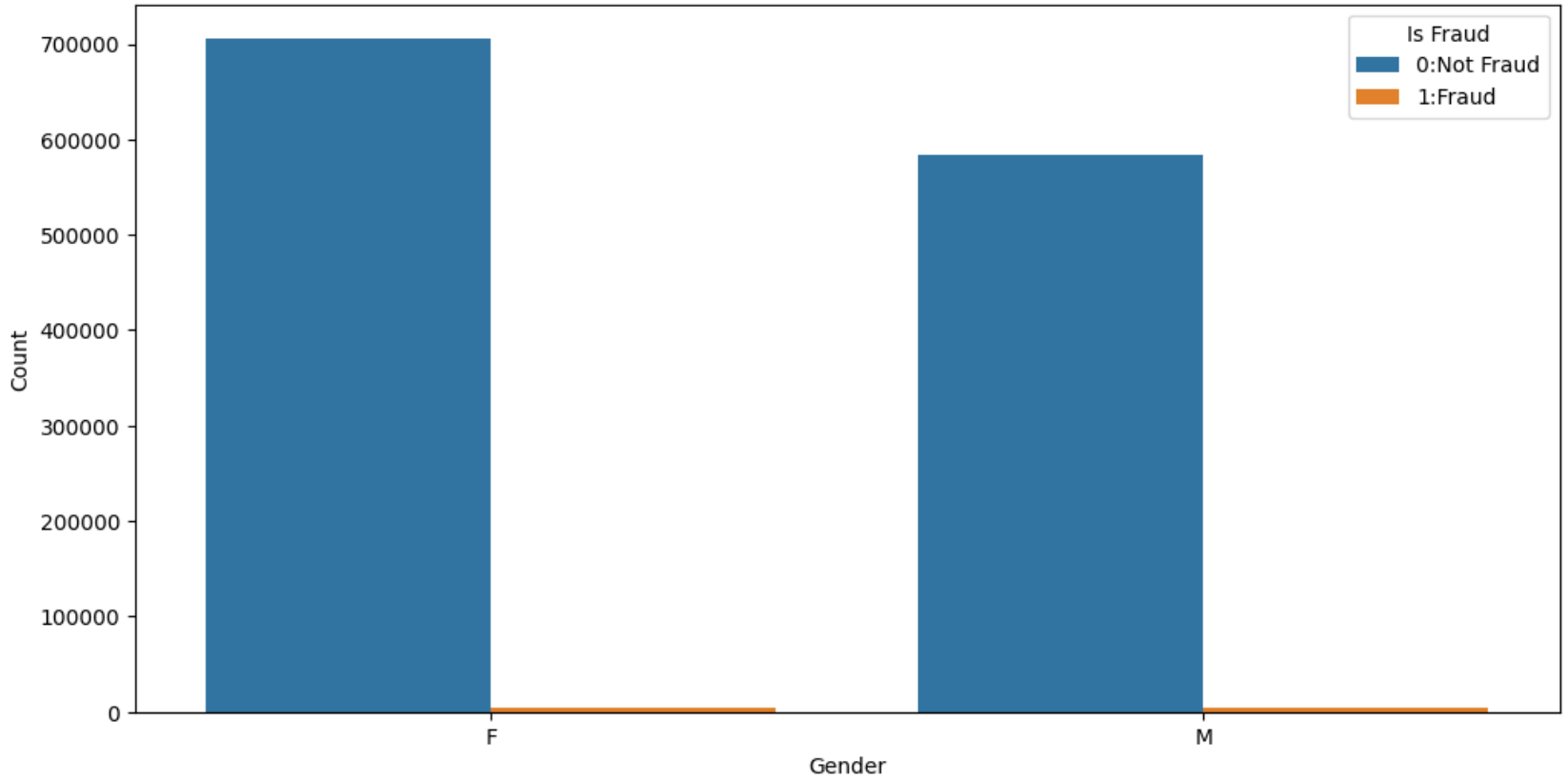


In [27]:
```
plt.figure(figsize=(12, 6))
sns.histplot(df_train['amt'], bins=20, kde=True)
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.title('Distribution of Amount')
plt.show()
```



The resulting plot shows the count of fraudulent and non-fraudulent transactions for each gender.

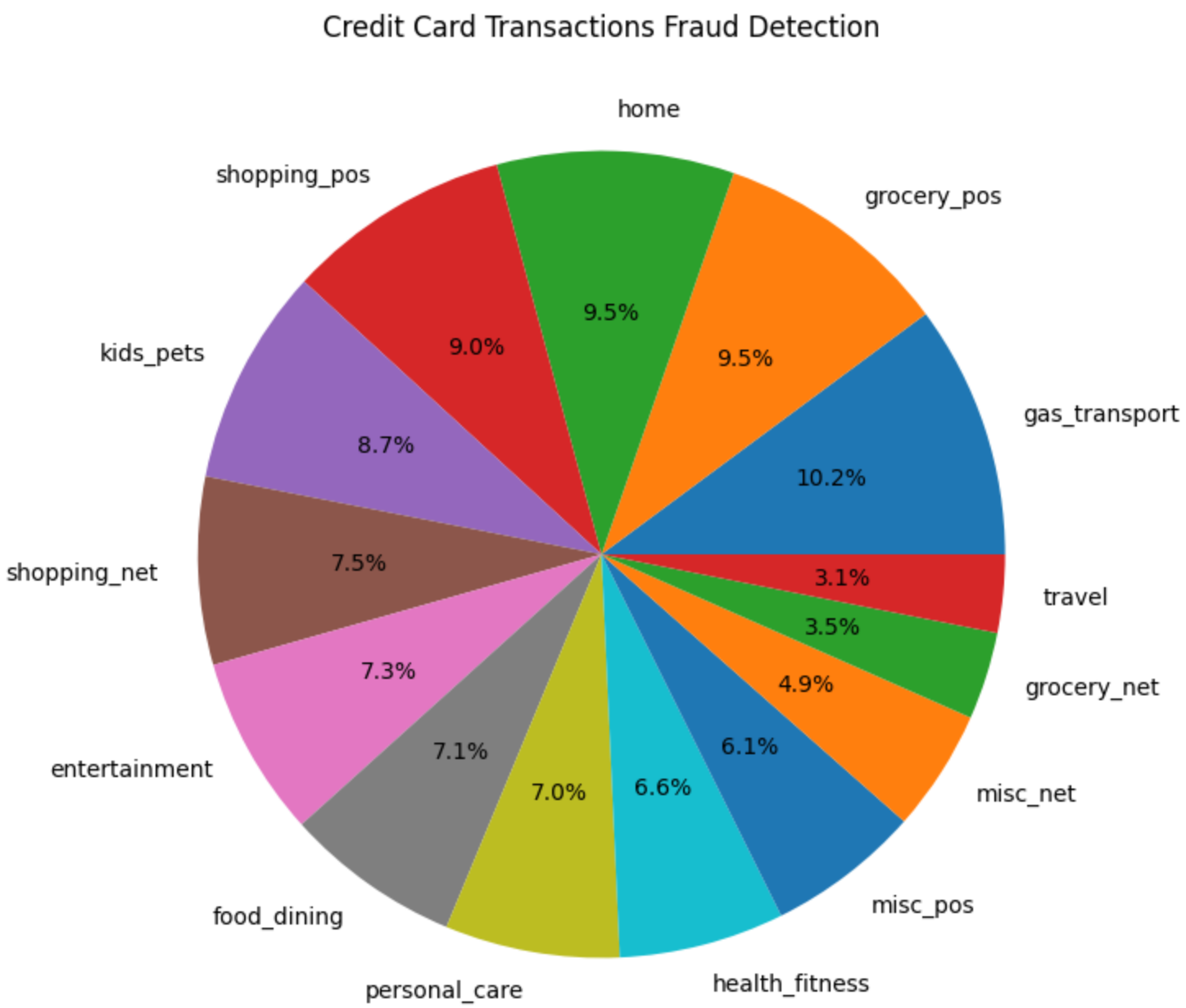
In [28]:
```
plt.figure(figsize=(12, 6))
sns.countplot(x='gender', hue='is_fraud', data=df_train)
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Is Fraud', labels=['0:Not Fraud', '1:Fraud'])
plt.show()
```



The resulting pie chart clearly shows the distribution of credit card transactions across different categories, helping to identify the proportion of transactions in each category.

In [29]:

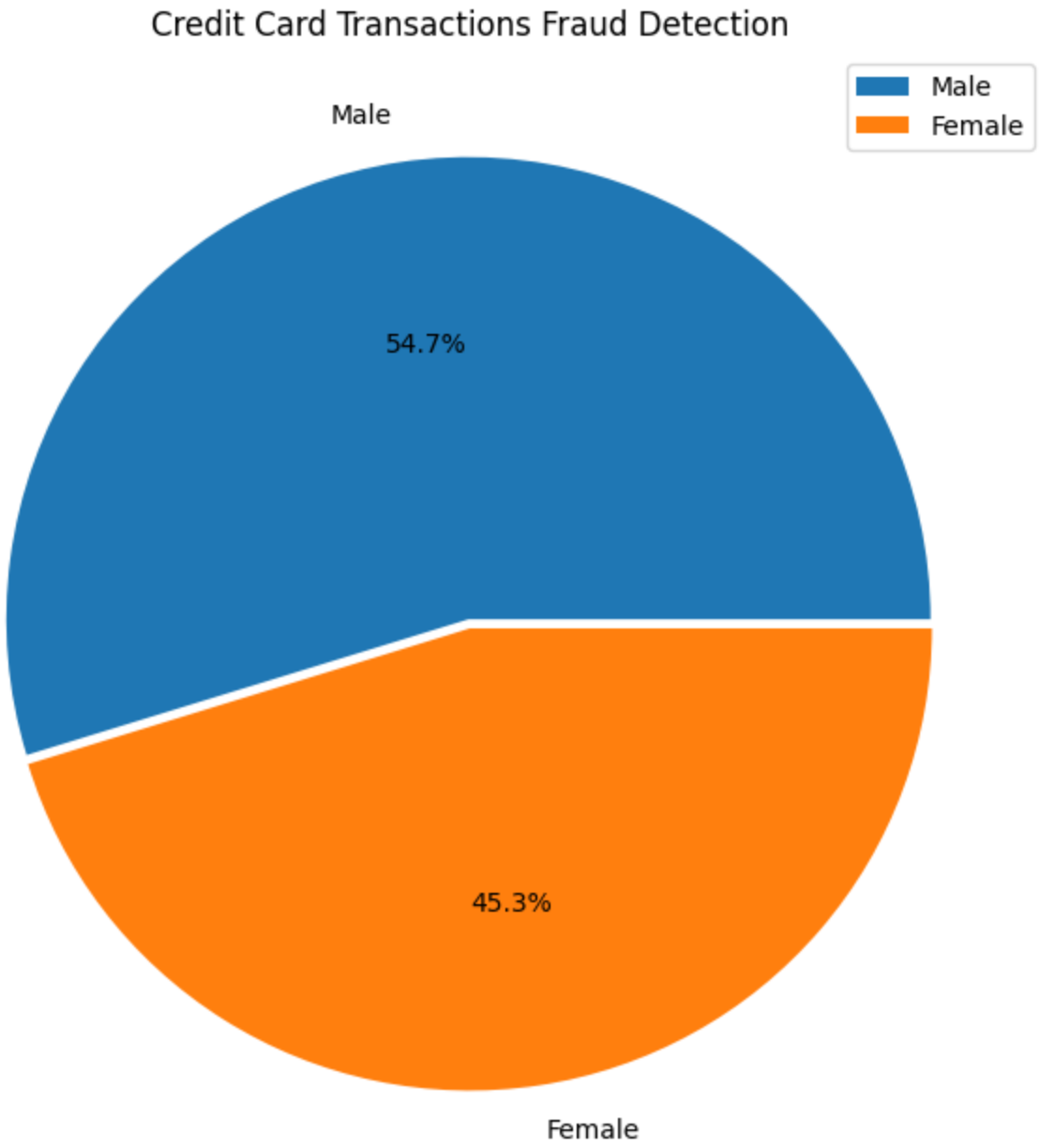
```
plt.figure(figsize = (10, 8))
plt.title('Credit Card Transactions Fraud Detection')
plt.pie(df_train['category'].value_counts(), labels = ["gas_transport", "grocery_pos", "home", "shopping
plt.show()
```



The resulting pie chart visualizes the distribution of credit card transactions between male and

In [30]:

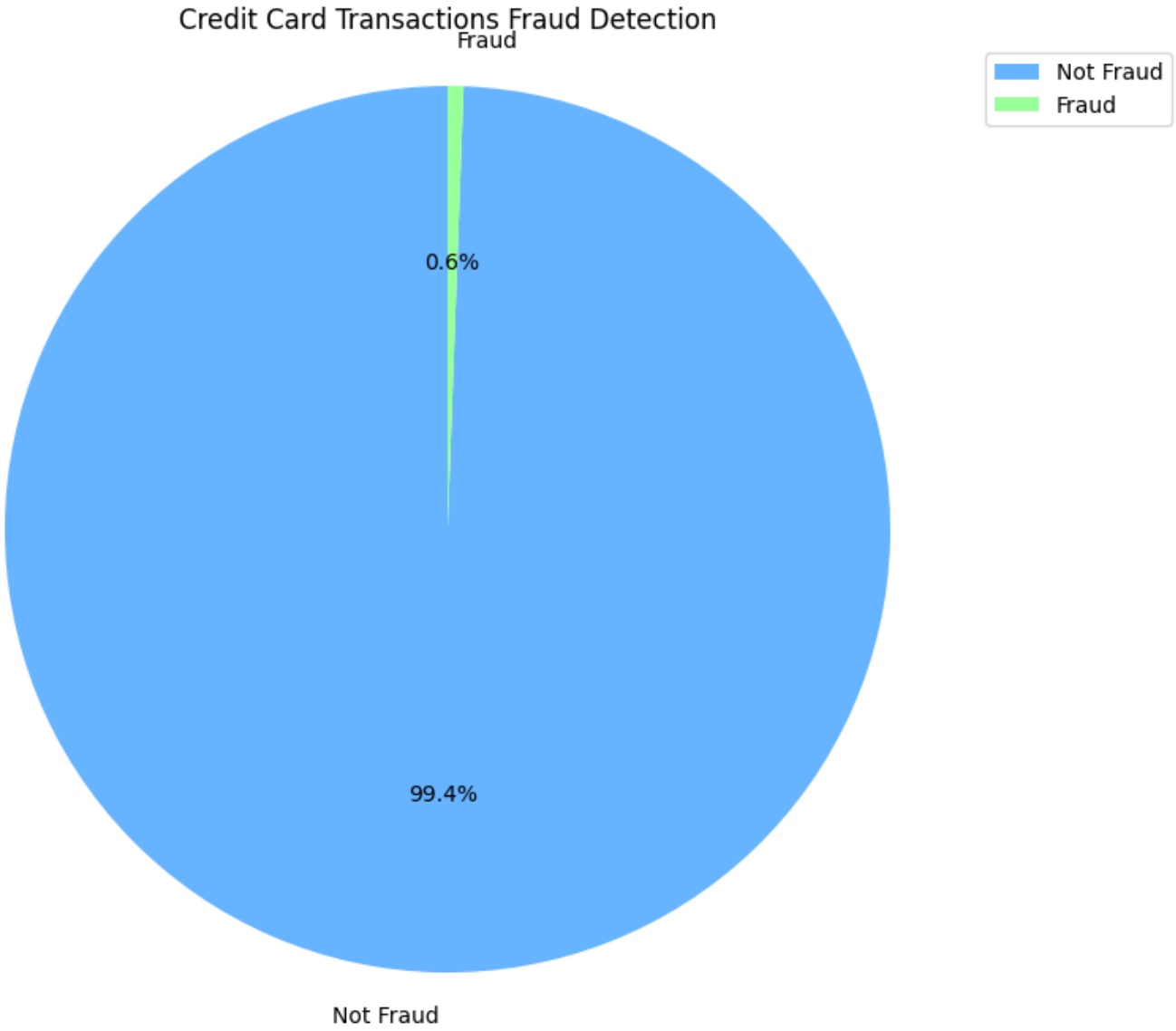
```
plt.figure(figsize = (13, 8))
plt.title('Credit Card Transactions Fraud Detection')
plt.pie(df_train['gender'].value_counts(), labels = ['Male', 'Female'], explode = (0.02, 0.0), autopct =
plt.legend(loc = 'best')
plt.show()
```



The resulting pie chart provides a visual representation of the distribution of fraud and non-

In [31]:

```
plt.figure(figsize=(12, 8))
plt.title('Credit Card Transactions Fraud Detection')

# Assuming df_train['is_fraud'] contains binary values (0 or 1)
fraud_counts = df_train['is_fraud'].value_counts()
labels = ['Not Fraud', 'Fraud']

plt.pie(fraud_counts, labels=labels, autopct='%1.1f%%', startangle=90, colors=['#66b3ff', '#99ff99'])
plt.axis('equal')
plt.legend(loc='best')
plt.show()
```


Credit Card Transactions Fraud Detection
Fraud
Not Fraud
Fraud
0.6%
99.4%
Not Fraud

In [32]:
```
1 df_train.columns
```

Out[32]:
```
Index(['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category',
       'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip',
       'lat', 'long', 'city_pop', 'job', 'dob', 'trans_num', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud'],
      dtype='object')
```

In [33]:
```
1 df_test.columns
```

Out[33]:
```
Index(['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category',
       'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip',
       'lat', 'long', 'city_pop', 'job', 'dob', 'trans_num', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud'],
      dtype='object')
```

Convert to Datetime, and Extract Hour, Day of Week, and Month

In [34]:
```
1 df_train['trans_date_trans_time'] = pd.to_datetime(df_train['trans_date_trans_time'])
2 df_train['hour'] = df_train['trans_date_trans_time'].dt.hour
3 df_train['day_of_week'] = df_train['trans_date_trans_time'].dt.dayofweek
4 df_train['month'] = df_train['trans_date_trans_time'].dt.month
```

In [35]:
```
1 df_test['trans_date_trans_time'] = pd.to_datetime(df_test['trans_date_trans_time'])
2 df_test['hour'] = df_test['trans_date_trans_time'].dt.hour
3 df_test['day_of_week'] = df_test['trans_date_trans_time'].dt.dayofweek
4 df_test['month'] = df_test['trans_date_trans_time'].dt.month
```

Dropping Unnecessary Columns, One-Hot Encoding and Mapping Gender to Numeric Values3

In [36]:
```
df_train = df_train.drop(['Unnamed: 0', 'trans_date_trans_time', 'merchant', 'first', 'last', 'street',
df_train = pd.get_dummies(df_train, columns=['category'], drop_first=True)


df_train['gender'] = df_train['gender'].map({'M': 0, 'F': 1})
```

In [37]:
```
df_test = df_test.drop(['Unnamed: 0', 'trans_date_trans_time', 'merchant', 'first', 'last', 'street', 'c
df_test = pd.get_dummies(df_test, columns=['category'], drop_first=True)


df_test['gender'] = df_test['gender'].map({'M': 0, 'F': 1})
```

In [38]:
```
df_train.head()
```

Out[38]:

	cc_num	amt	gender	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud	...	category_grocery
0	2703186189652095	4.97	1	36.0788	-81.1781	3495	1325376018	36.011293	-82.048315	0	...	
1	630423337322	107.23	1	48.8878	-118.2105	149	1325376044	49.159047	-118.186462	0	...	
2	38859492057661	220.11	0	42.1808	-112.2620	4154	1325376051	43.150704	-112.154481	0	...	
3	3534093764340240	45.00	0	46.2306	-112.1138	1939	1325376076	47.034331	-112.561071	0	...	
4	375534208663984	41.96	0	38.4207	-79.4629	99	1325376186	38.674999	-78.632459	0	...	

5 rows × 26 columns

In [39]:
```
for column in df_train.columns:
    if df_train[column].dtype == bool:
        df_train[column] = df_train[column].astype(int)

```

In [40]:
```
# corr = df_train.corr()
# sns.heatmap(df_train,cmap='coolwarm',annot=True)
# plt.show()
```

In [41]:
```
df_test.head()
```

Out[41]:

	cc_num	amt	gender	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud	...	category_grocery_
0	2291163933867244	2.86	0	33.9659	-80.9355	333497	1371816865	33.986391	-81.200714	0	...	F
1	3573030041201292	29.84	1	40.3207	-110.4360	302	1371816873	39.450498	-109.960431	0	...	F
2	3598215285024754	41.28	1	40.6729	-73.5365	34496	1371816893	40.495810	-74.196111	0	...	F
3	3591919803438423	60.05	0	28.5697	-80.8191	54767	1371816915	28.812398	-80.883061	0	...	F
4	3526826139003047	3.19	0	44.2529	-85.0170	1126	1371816917	44.959148	-85.884734	0	...	F

5 rows × 26 columns

In [42]:
```
for column in df_test.columns:
    if df_test[column].dtype == bool:
        df_test[column] = df_test[column].astype(int)

```

In [43]:
```
# corr = df_test.corr()
# sns.heatmap(df_test,cmap='coolwarm',annot=True)
# plt.show()
```

Feature selection

Let's use X as our feature matrix and Y as our target variable to train and test a machine learning model for fraud detection.

In [44]:
```
X = df_train.drop(['is_fraud'], axis=1)
y = df_train['is_fraud']
```

In [45]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Standardize the features (important for Logistic Regression)

In [46]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Logistic Regression

In [47]:
```
logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)
```

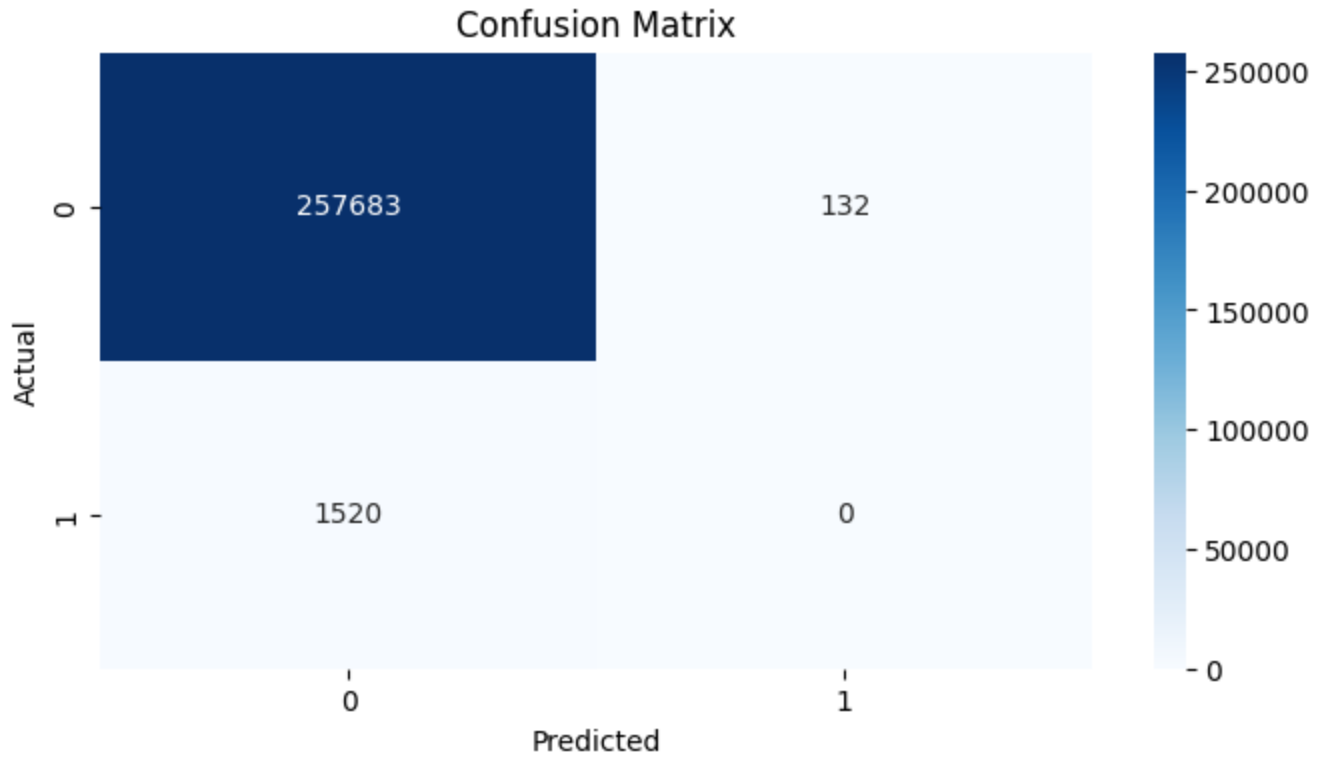
Out[47]:
```
LogisticRegression
LogisticRegression()
```

In [48]:
```
logistic_predictions = logistic_model.predict(X_test_scaled)
```

In [49]:

```
# Calculate the Accuracy score and Classification Report
accuracy = accuracy_score(y_test, logistic_predictions)
report = classification_report(y_test, logistic_predictions)
conf_matrix = confusion_matrix(y_test, logistic_predictions)

# Print the Accuracy score, Confusion Matrix, Classification Report
print(f"Logistic Regression Accuracy Score: {accuracy*100:.2f}")
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{report}')

# Data Visualization
plt.figure(figsize=(8, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Logistic Regression Accuracy Score: 99.36
Confusion Matrix:
[[257683    132]
 [  1520      0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    257815
           1       0.00      0.00      0.00      1520

    accuracy                           0.99    259335
   macro avg       0.50      0.50      0.50    259335
weighted avg       0.99      0.99      0.99    259335
```

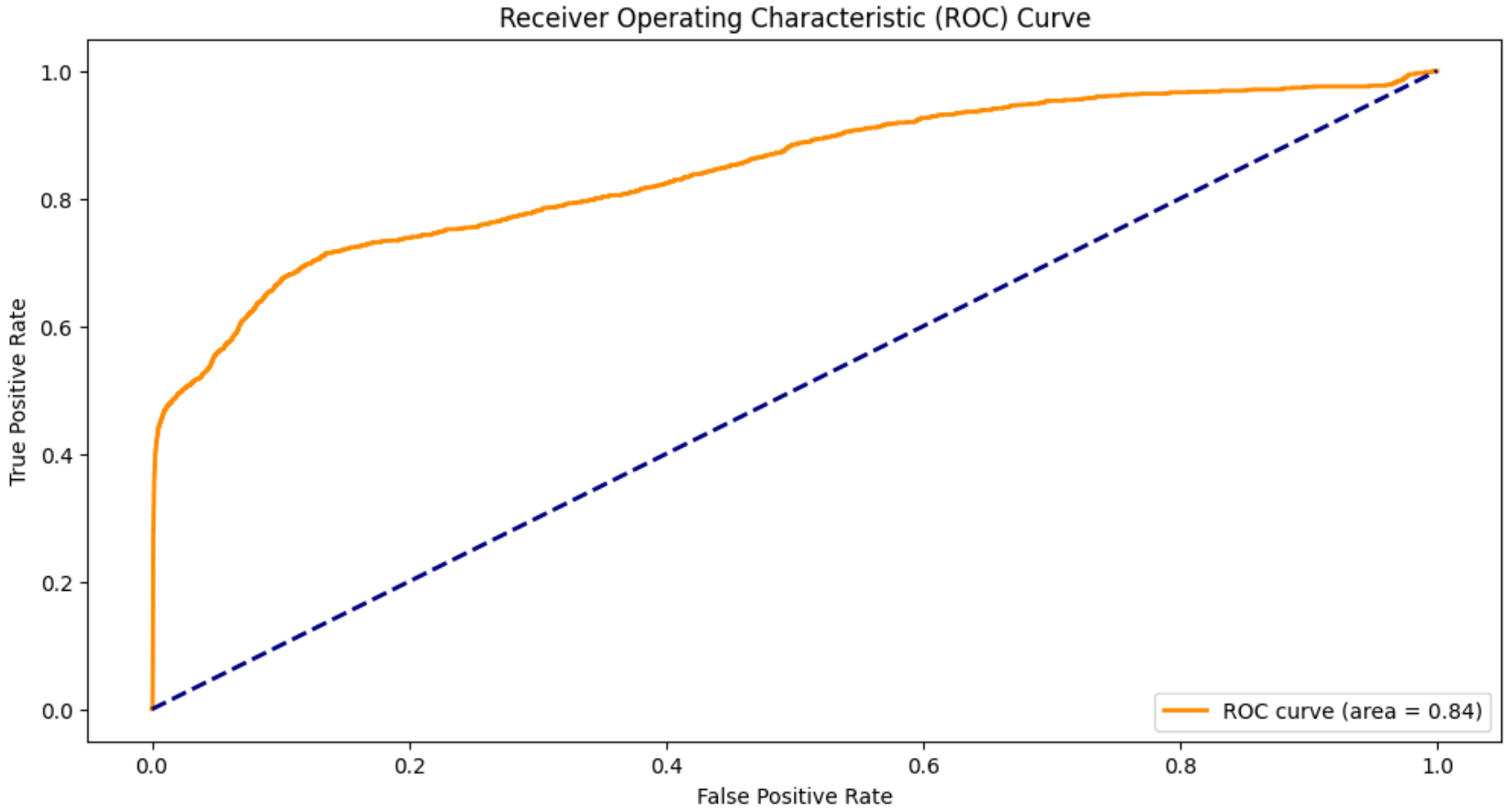


In [50]:
```
1 print(f'The Train_accuracy: {logistic_model.score(X_train_scaled, y_train)*100:.2f}')
2 print(f'The Test_accuracy: {logistic_model.score(X_test_scaled, y_test)*100:.2f}')
```

```
The Train_accuracy: 99.37
The Test_accuracy: 99.36
```

In [51]:

```
# Get predicted probabilities for the positive class
y_probs = logistic_model.predict_proba(X_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc_lr = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(12, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc_lr))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

print(f'AUC: {roc_auc_lr*100:.2f}')
```




```
AUC: 84.35
```

Decision Tree

In [52]:
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train_scaled, y_train)
```

Out[52]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [53]:
```
dt_predictions_test = dt_model.predict(X_test_scaled)
```

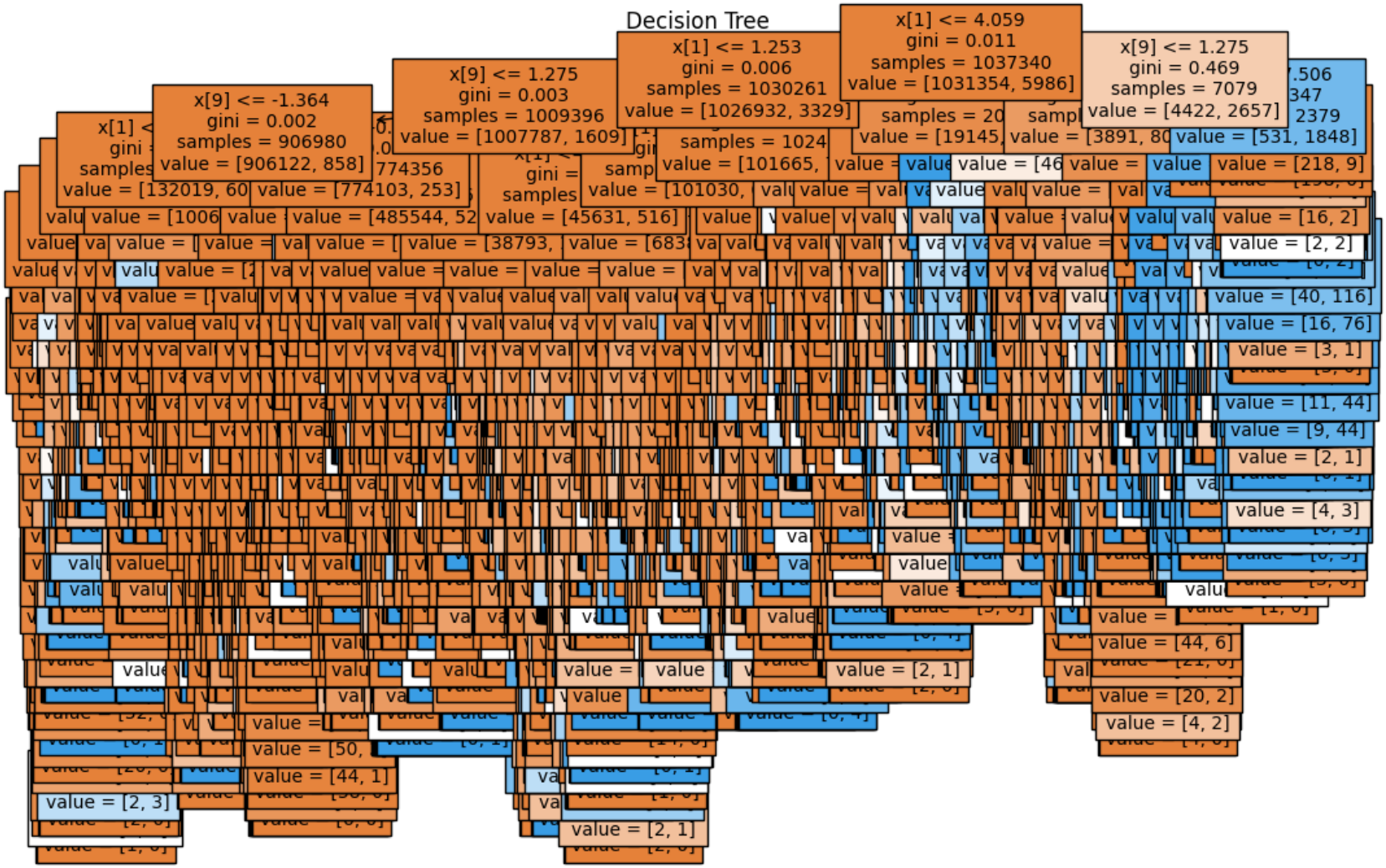
In [54]:
```
print(f'The Train_accuracy: {dt_model.score(X_train_scaled, y_train)*100:.2f}')
print(f'The Test_accuracy: {dt_model.score(X_test_scaled, y_test)*100:.2f}')
```

```
The Train_accuracy: 100.00
The Test_accuracy: 99.70
```

Plot Tree

In [55]:

```
from sklearn.tree import plot_tree
from sklearn import tree

plt.figure(figsize=(12, 8))
tree.plot_tree(dt_model,  filled=True, fontsize=10)
plt.title("Decision Tree")
plt.show()
```



Random Forest Classifier

In [56]:
```
rf_model = RandomForestClassifier()
rf_model.fit(X_train_scaled, y_train)
```

Out[56]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [57]:
```

rf_predictions_test = rf_model.predict(X_test_scaled)
```

In [59]:
```
print(f'The Train_accuracy: {rf_model.score(X_train_scaled, y_train)*100:.2f}')
print(f'The Test_accuracy: {rf_model.score(X_test_scaled, y_test)*100:.2f}')
```

```
The Train_accuracy: 100.00
The Test_accuracy: 99.81
```