Importing libraries

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

EDA

In [2]:
```
train_path = "C:/Users/Megha Sharma/Desktop/MEGHA/MSC DATA SCIENCE/INTERNSHIP/Co

train = pd.read_csv(train_path)
```

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

Out[3]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	10
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	22
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	4
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	4

5 rows × 23 columns

In [4]:
```
train.shape
```

Out[4]: (1296675, 23)

In [5]:
```
train.columns
```

Out[5]:
```
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 [6]:
```
train.describe()
```

Out[6]:

	Unnamed: 0	cc_num	amt	zip	lat	
count	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.29667
mean	6.483370e+05	4.171920e+17	7.035104e+01	4.880067e+04	3.853762e+01	-9.02263
std	3.743180e+05	1.308806e+18	1.603160e+02	2.689322e+04	5.075808e+00	1.37590
min	0.000000e+00	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.65672
25%	3.241685e+05	1.800429e+14	9.650000e+00	2.623700e+04	3.462050e+01	-9.67980
50%	6.483370e+05	3.521417e+15	4.752000e+01	4.817400e+04	3.935430e+01	-8.74769
75%	9.725055e+05	4.642255e+15	8.314000e+01	7.204200e+04	4.194040e+01	-8.01580
max	1.296674e+06	4.992346e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.79503

In [7]:
```
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 [8]:
```
for col in train.columns:
    print(col, train[col].isnull().sum())
```

```
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
```

In [9]:
```
exit_counts = train["is_fraud"].value_counts()
print("Yes: ",exit_counts[1])
print("No: ",exit_counts[0])
```

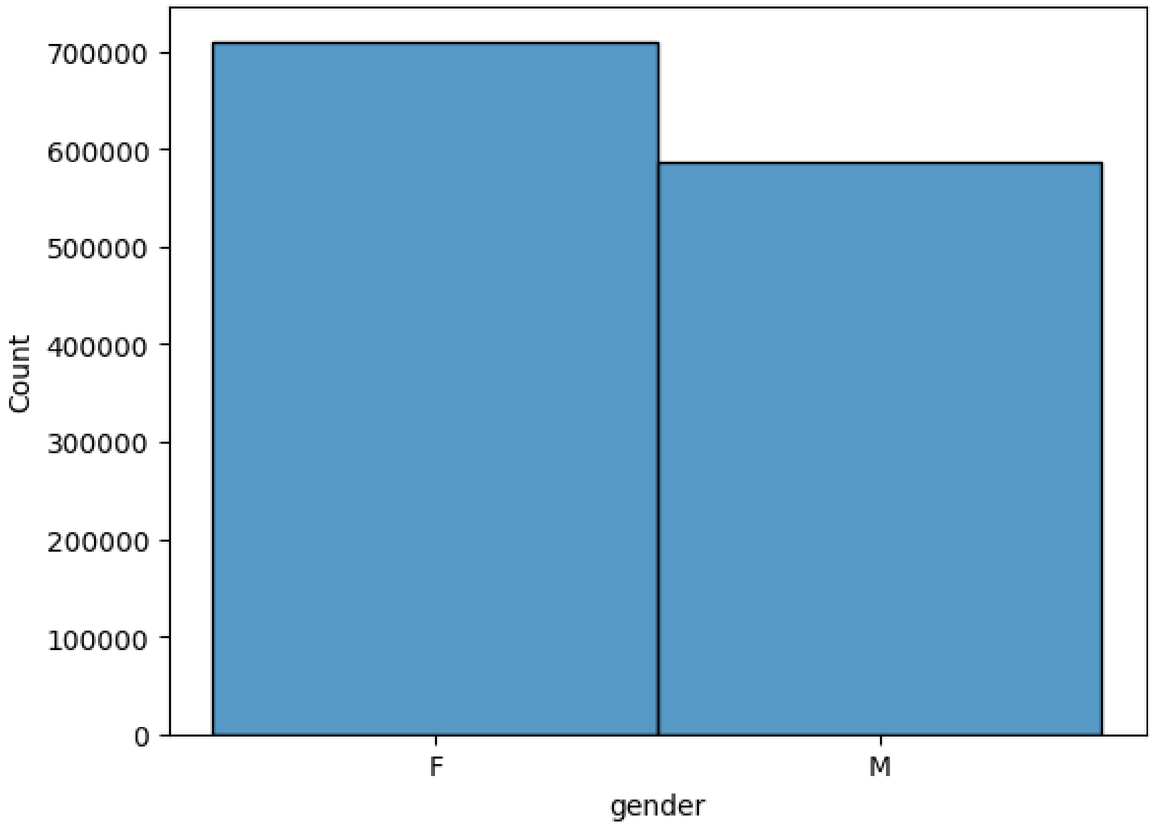
```
Yes:  7506
No:  1289169
```

In [10]:
```
fraudulent_data = train[train['is_fraud'] > 0]

gender_counts = train['gender'].value_counts()

male_fraud_count = fraudulent_data[fraudulent_data['gender'] == 'M'].shape[0]
female_fraud_count = fraudulent_data[fraudulent_data['gender'] == 'F'].shape[0]

print("male:",  gender_counts['M'])
print("female",  gender_counts['F'])
print("male_fraud:", male_fraud_count)
print("female_fraud:", female_fraud_count)
```

```
male: 586812
female 709863
male_fraud: 3771
female_fraud: 3735
```

In [11]:
```
sns.histplot(data=train['gender'] )
```

Out[11]:
```
<Axes: xlabel='gender', ylabel='Count'>
```



Data processing

In [12]:
```
# Drop unnecessary columns
train.drop(columns=['Unnamed: 0','cc_num','first', 'last', 'street', 'city', 'st
```

In [13]:
```
train.head()
```

Out[13]:

	trans_date_trans_time	merchant	category	amt	gender	lat	lon
0	2019-01-01 00:00:18	fraud_Rippin, Kub and Mann	misc_net	4.97	F	36.0788	-81.178
1	2019-01-01 00:00:44	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	F	48.8878	-118.210
2	2019-01-01 00:00:51	fraud_Lind-Buckridge	entertainment	220.11	M	42.1808	-112.262
3	2019-01-01 00:01:16	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	M	46.2306	-112.113
4	2019-01-01 00:03:06	fraud_Keeling-Crist	misc_pos	41.96	M	38.4207	-79.462

In [14]:
```
# Convert categorical variables to numerical using Label Encoding
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
newdata = train.apply(LabelEncoder().fit_transform)
```

In [15]:
```
newdata.head()
```

Out[15]:

	trans_date_trans_time	merchant	category	amt	gender	lat	long	city_pop	job
0	0	514	8	397	0	291	693	458	370
1	1	241	4	10623	0	964	60	43	428
2	2	390	0	21906	1	736	88	486	307
3	3	360	2	4400	1	931	91	367	328
4	4	297	9	4096	1	398	753	22	116

In [16]:
```
# Splitting data into train and test sets
from sklearn.model_selection import train_test_split
X = newdata.drop("is_fraud", axis=1)
y = newdata["is_fraud"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [17]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[17]:
```
((1037340, 12), (259335, 12), (1037340,), (259335,))
```

Training

a. LogisticRegression

In [18]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
print("Accuracy Score: ", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report: \n", classification_report(y_test, y_pred))
```

```
C:\Users\Megha Sharma\AppData\Roaming\Python\Python39\site-packages\sklearn\linea
r_model\_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy Score:  0.9934332041567856
Confusion Matrix:
 [[257525    290]
 [  1413    107]]
Classification Report:
               precision    recall  f1-score   support

           0       0.99      1.00      1.00    257815
           1       0.27      0.07      0.11      1520

    accuracy                           0.99    259335
   macro avg       0.63      0.53      0.55    259335
weighted avg       0.99      0.99      0.99    259335
```

b. Decision Tree

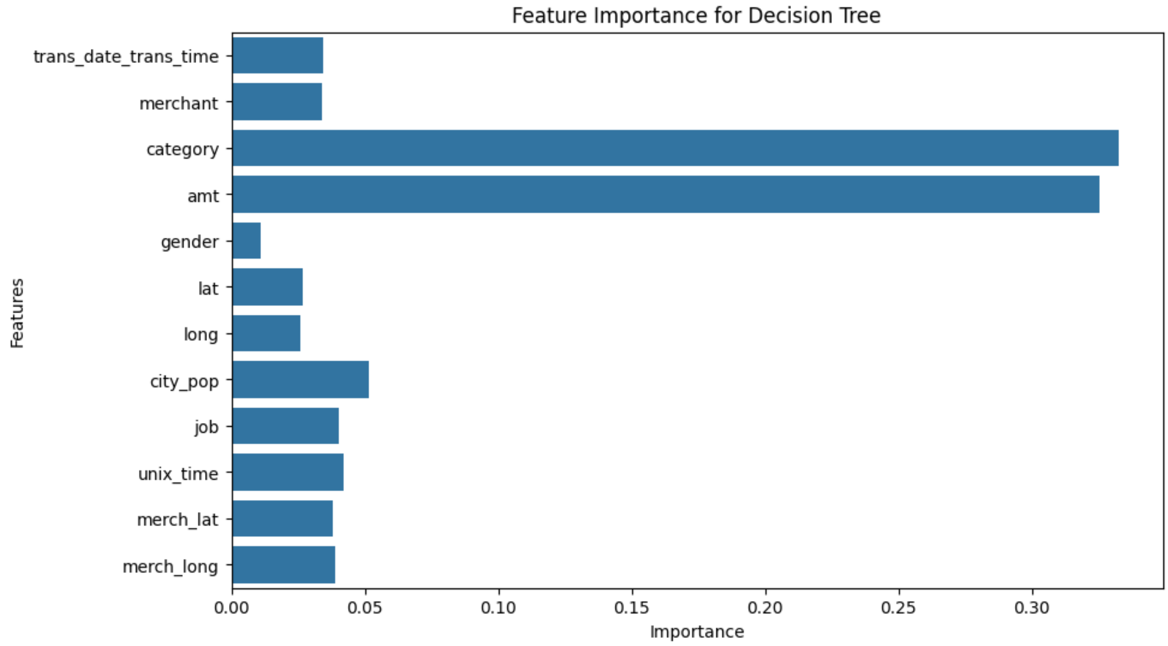
In [19]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
print("Accuracy Score: ", accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Classification Report: \n", classification_report(y_test, y_pred))
```

```
Accuracy Score:  0.9959164786858696
Confusion Matrix:
 [[257250    565]
 [   494   1026]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    257815
           1       0.64      0.68      0.66      1520

    accuracy                           1.00    259335
   macro avg       0.82      0.84      0.83    259335
weighted avg       1.00      1.00      1.00    259335
```

In [20]:
```
from sklearn.metrics import roc_auc_score
print('Logistic Regression: ', roc_auc_score(y_test, log_reg.predict_proba(X_tes
print('Decision Tree: ', roc_auc_score(y_test, dt.predict_proba(X_test)[:, 1]))
```

```
Logistic Regression:  0.8159288330983967
Decision Tree:  0.8364042530496675
```

In [21]:
```
# Feature Importance for Decision Tree
feature_importance = dt.feature_importances_
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importance, y=X.columns)
plt.title("Feature Importance for Decision Tree")
plt.xlabel("Importance")
plt.ylabel("Features")
plt.show()
```


Feature Importance for Decision Tree
trans_date_trans_time
merchant
category
amt
gender
lat
long
city_pop
job
unix_time
merch_lat
merch_long
Features
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Importance

```
In [ ]:
```