1. Importing the libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, precision_score, recall_score, f1_score
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
```

2. Loading the csv train data

```
df = pd.read_csv('fraudTrain.csv')
test_df = pd.read_csv('fraudTest.csv')
```

3. Data Inspection

```
df.shape
```

```
(1296675, 23)
```

```
df.sample(5)
```

{"type":"dataframe"}

```
df['is_fraud'].value_counts()
```

```
is_fraud
0    1289169
1       7506
Name: count, dtype: int64
```

```
print(df.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
None
```

##4. Finding Missing Values

```
print(df.describe())
```

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

```
                long      city_pop     unix_time     merch_lat
merch_long  \
count  1.296675e+06  1.296675e+06  1.296675e+06  1.296675e+06
1.296675e+06
mean  -9.022634e+01  8.882444e+04  1.349244e+09  3.853734e+01 -
9.022646e+01
std    1.375908e+01  3.019564e+05  1.284128e+07  5.109788e+00
1.377109e+01
min   -1.656723e+02  2.300000e+01  1.325376e+09  1.902779e+01 -
1.666712e+02
25%   -9.679800e+01  7.430000e+02  1.338751e+09  3.473357e+01 -
9.689728e+01
50%   -8.747690e+01  2.456000e+03  1.349250e+09  3.936568e+01 -
8.743839e+01
75%   -8.015800e+01  2.032800e+04  1.359385e+09  4.195716e+01 -
8.023680e+01
max   -6.795030e+01  2.906700e+06  1.371817e+09  6.751027e+01 -
6.695090e+01

           is_fraud
count  1.296675e+06
mean   5.788652e-03
std    7.586269e-02
min    0.000000e+00
25%    0.000000e+00
50%    0.000000e+00
75%    0.000000e+00
max    1.000000e+00
```

```python
print(df.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
dtype: int64
```

```
df.duplicated().sum()
```

```
0
```

5. Analysing the correlation

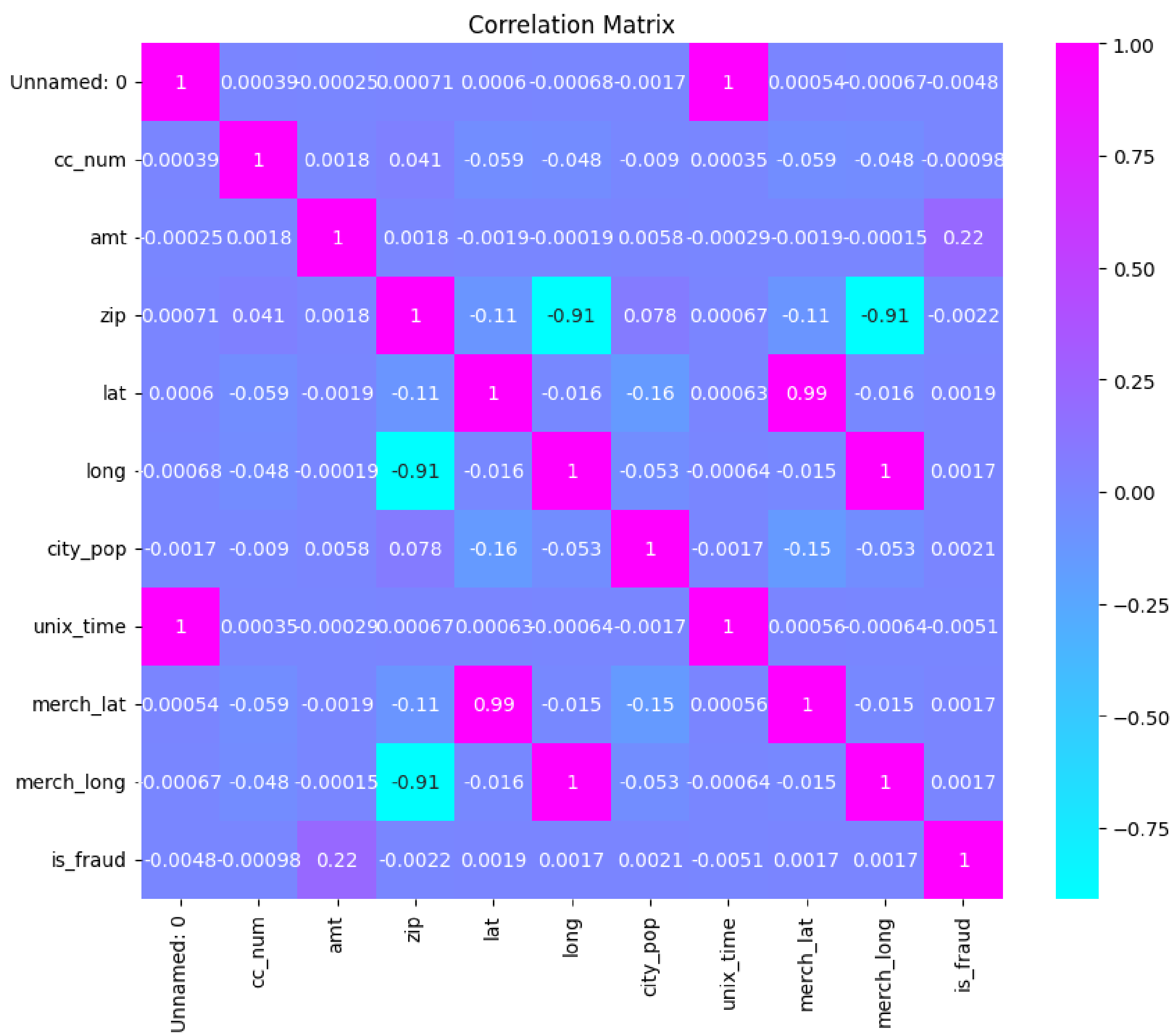
```
df.corr(numeric_only=True)['is_fraud']
```

```
Unnamed: 0   -0.004767
cc_num       -0.000981
amt           0.219404
zip          -0.002162
lat           0.001894
long          0.001721
city_pop      0.002136
unix_time    -0.005078
merch_lat     0.001741
merch_long    0.001721
is_fraud      1.000000
Name: is_fraud, dtype: float64
```

```
plt.figure(figsize=(10, 8))
for col in df.columns:
  if df[col].dtype == 'object':
    try:
      df[col] = pd.to_datetime(df[col])
    except:
      pass


sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='cool')
plt.title('Correlation Matrix')
plt.show()
```

```
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
```

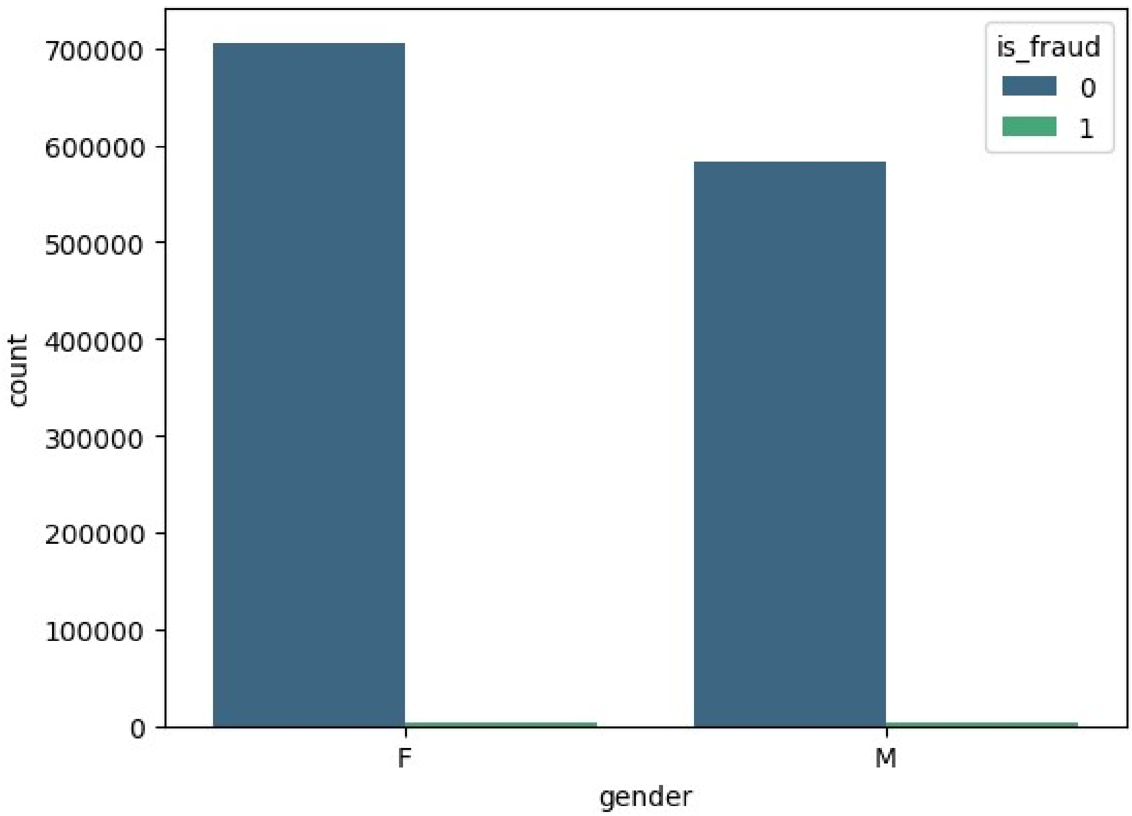
```
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
<ipython-input-11-c996b71c5b20>:5: UserWarning: Could not infer
format, so each element will be parsed individually, falling back to
`dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col])
```



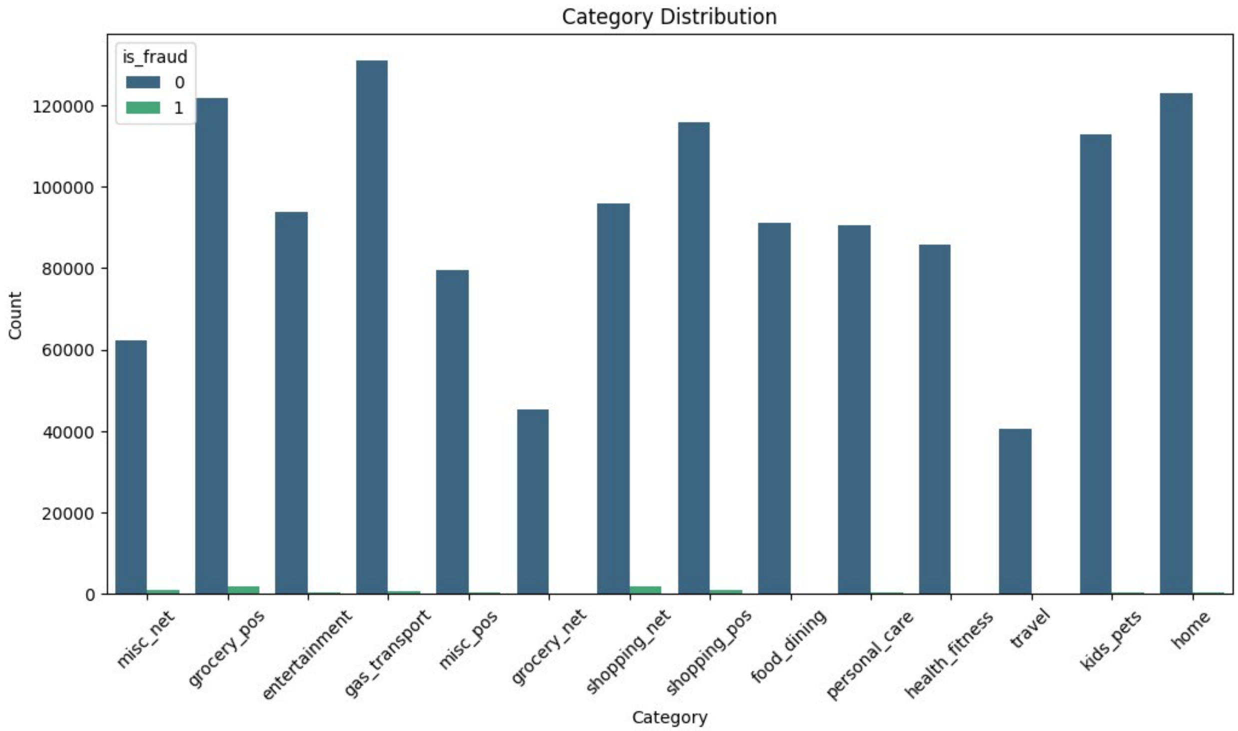
1. Raw Data Visualization

Categorical

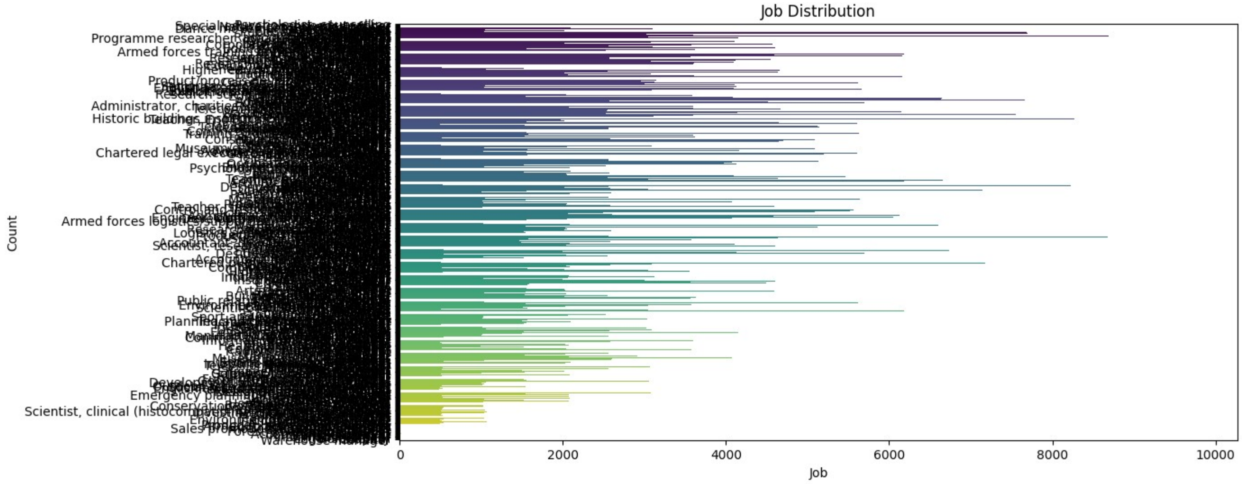
```
sns.countplot(x='gender', hue='is_fraud', data=df, palette='viridis')
<Axes: xlabel='gender', ylabel='count'>
```




```
plt.figure(figsize=(12, 6))
sns.countplot(x=df['category'], hue=df['is_fraud'], palette='viridis')
plt.xticks(rotation=45)
plt.title('Category Distribution')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```



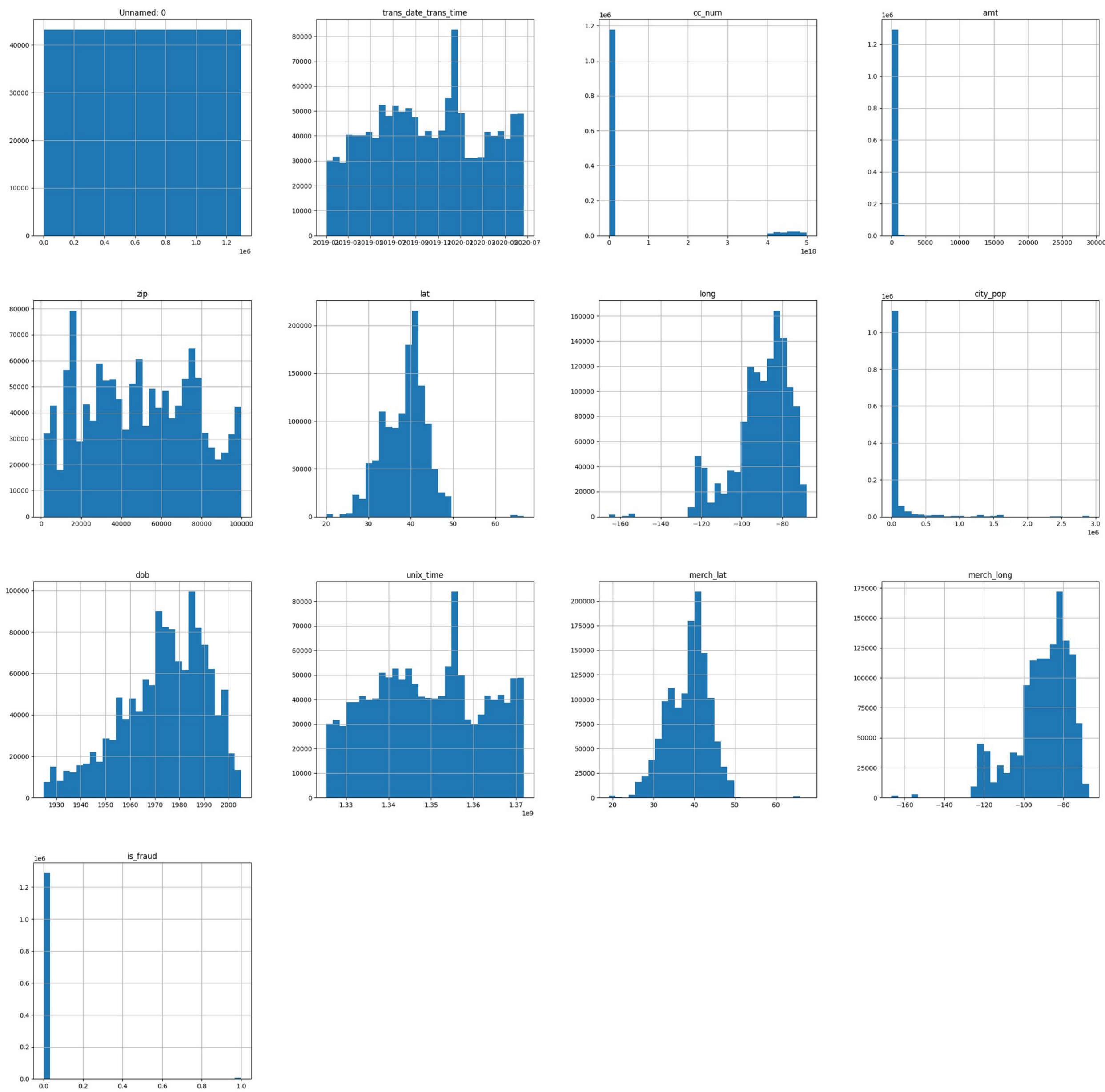

```
#### Results Show no correlation if used without extracting
featureswith the target since its mostly
##different for each entery are : marchant , first, last,street, city,
job
##(vo kehte hain na chor ko to bss chori karni hai they are unbiased
unlike normal people)

plt.figure(figsize=(12, 6))
sns.countplot(df['job'], palette='viridis')
plt.title('Job Distribution')
plt.xlabel('Job')
plt.ylabel('Count')
plt.show()

<ipython-input-14-261c9f1fbdde>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(df['job'], palette='viridis')
```



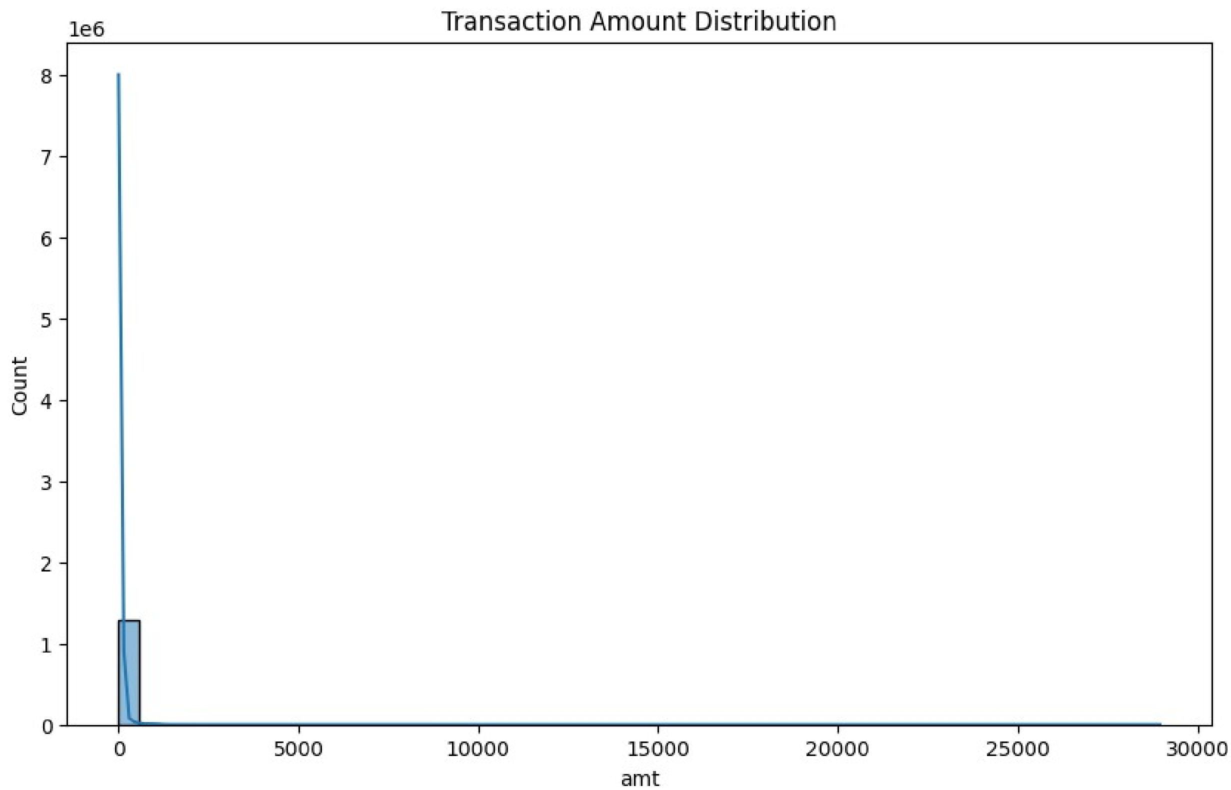
Numerical Data Analysis

```
df.hist(bins=30, figsize=(30, 30))
```

```
array([[<Axes: title={'center': 'Unnamed: 0'}>,
        <Axes: title={'center': 'trans_date_trans_time'}>,
        <Axes: title={'center': 'cc_num'}>,
        <Axes: title={'center': 'amt'}>],
       [<Axes: title={'center': 'zip'}>, <Axes: title={'center':
'lat'}>,
        <Axes: title={'center': 'long'}>,
        <Axes: title={'center': 'city_pop'}>],
       [<Axes: title={'center': 'dob'}>,
        <Axes: title={'center': 'unix_time'}>,
        <Axes: title={'center': 'merch_lat'}>,
        <Axes: title={'center': 'merch_long'}>],
       [<Axes: title={'center': 'is_fraud'}>, <Axes: >, <Axes: >,
        <Axes: >]], dtype=object)
```

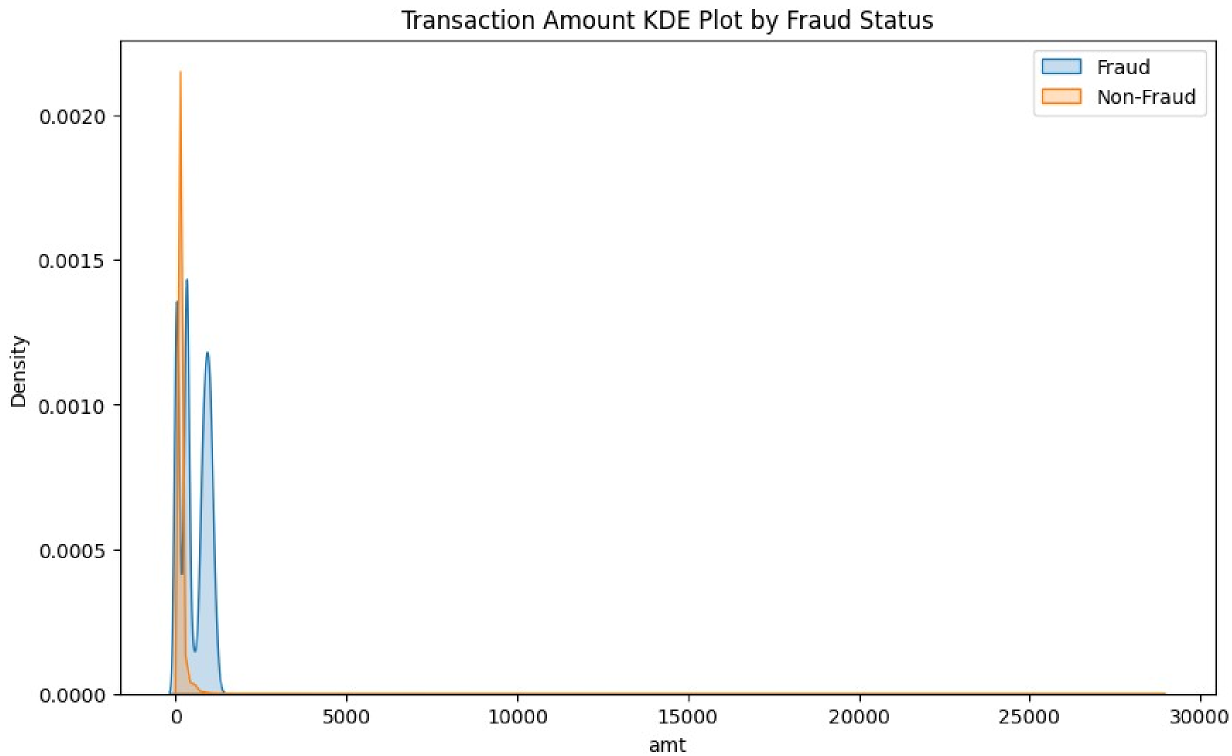



```python
plt.figure(figsize=(10, 6))
sns.histplot(df['amt'], bins=50, kde=True)
plt.title('Transaction Amount Distribution')
plt.show()
```

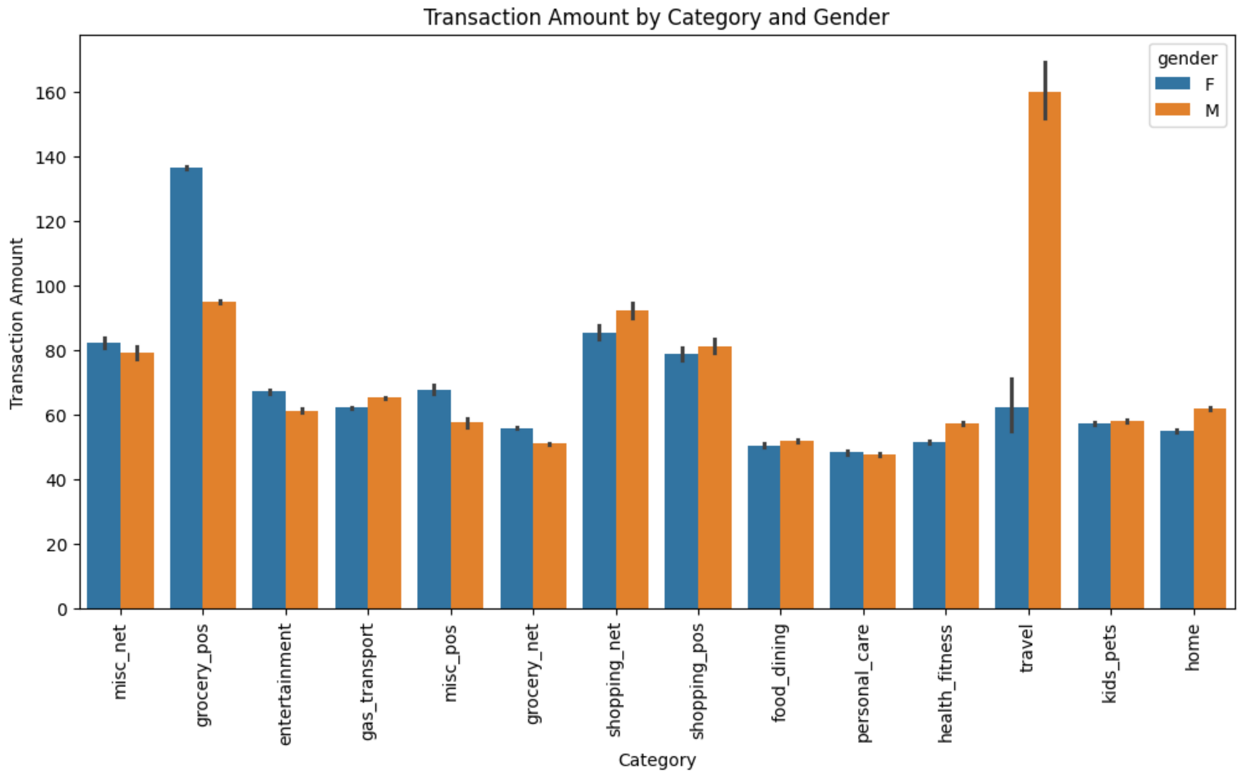



```
plt.figure(figsize=(10, 6))
sns.kdeplot(df[df['is_fraud'] == 1]['amt'], label='Fraud', shade=True)
sns.kdeplot(df[df['is_fraud'] == 0]['amt'], label='Non-Fraud',
shade=True)
plt.title('Transaction Amount KDE Plot by Fraud Status')
plt.legend()
plt.show()
```

```
<ipython-input-17-de8859d2b4a4>:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[df['is_fraud'] == 1]['amt'], label='Fraud',
shade=True)
<ipython-input-17-de8859d2b4a4>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[df['is_fraud'] == 0]['amt'], label='Non-Fraud',
shade=True)
```




```
plt.figure(figsize=(12, 6))
sns.barplot(x='category', y='amt', hue='gender', data=df)
plt.title('Transaction Amount by Category and Gender')
plt.xlabel('Category')
plt.ylabel('Transaction Amount')
plt.xticks(rotation=90)
plt.show()
```



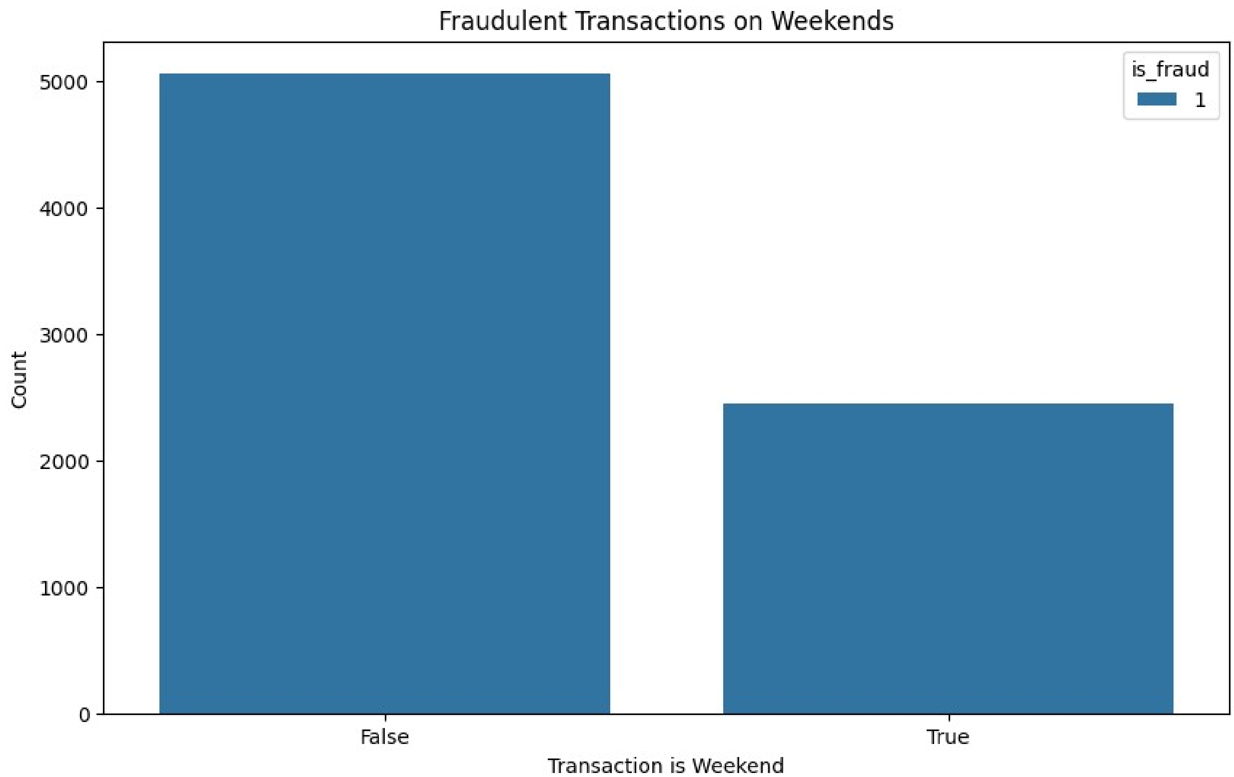
6.Feature Engineering and Visualization

```
#making features out of trans_date_trans_time

df.loc[:, 'trans_date_trans_time'] =
pd.to_datetime(df['trans_date_trans_time'], errors='coerce')


df = df.dropna(subset=['trans_date_trans_time'])


df.loc[:, 'trans_hour'] = df['trans_date_trans_time'].dt.hour
df.loc[:, 'trans_day'] = df['trans_date_trans_time'].dt.day
df.loc[:, 'trans_month'] = df['trans_date_trans_time'].dt.month
df.loc[:, 'trans_dayofweek'] =
df['trans_date_trans_time'].dt.dayofweek
df.loc[:, 'trans_is_weekend'] =
df['trans_date_trans_time'].dt.dayofweek >= 5


df[['trans_date_trans_time', 'trans_hour', 'trans_day', 'trans_month',
'trans_dayofweek', 'trans_is_weekend']].head()
```


```
{"summary":"{\n  \"name\": \"df[['trans_date_trans_time',
'trans_hour', 'trans_day', 'trans_month', 'trans_dayofweek',
'trans_is_weekend']]\",\n  \"rows\": 5,\n  \"fields\": [\n    {\n
\"column\": \"trans_date_trans_time\",\n      \"properties\": {\n
```

```
\"dtype\": \"date\",\n        \"min\": \"2019-01-01 00:00:18\",\n
\"max\": \"2019-01-01 00:03:06\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"2019-01-01 00:00:44\",\n          \"2019-
01-01 00:03:06\",\n          \"2019-01-01 00:00:51\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"trans_hour\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"trans_day\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"trans_month\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"trans_dayofweek\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"trans_is_weekend\",\n
\"properties\": {\n        \"dtype\": \"boolean\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          false\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


```python
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.countplot(x='trans_is_weekend', hue='is_fraud',
data=df[df['is_fraud'] == True])
plt.title('Fraudulent Transactions on Weekends')
plt.xlabel('Transaction is Weekend')
plt.ylabel('Count')
plt.show()
```



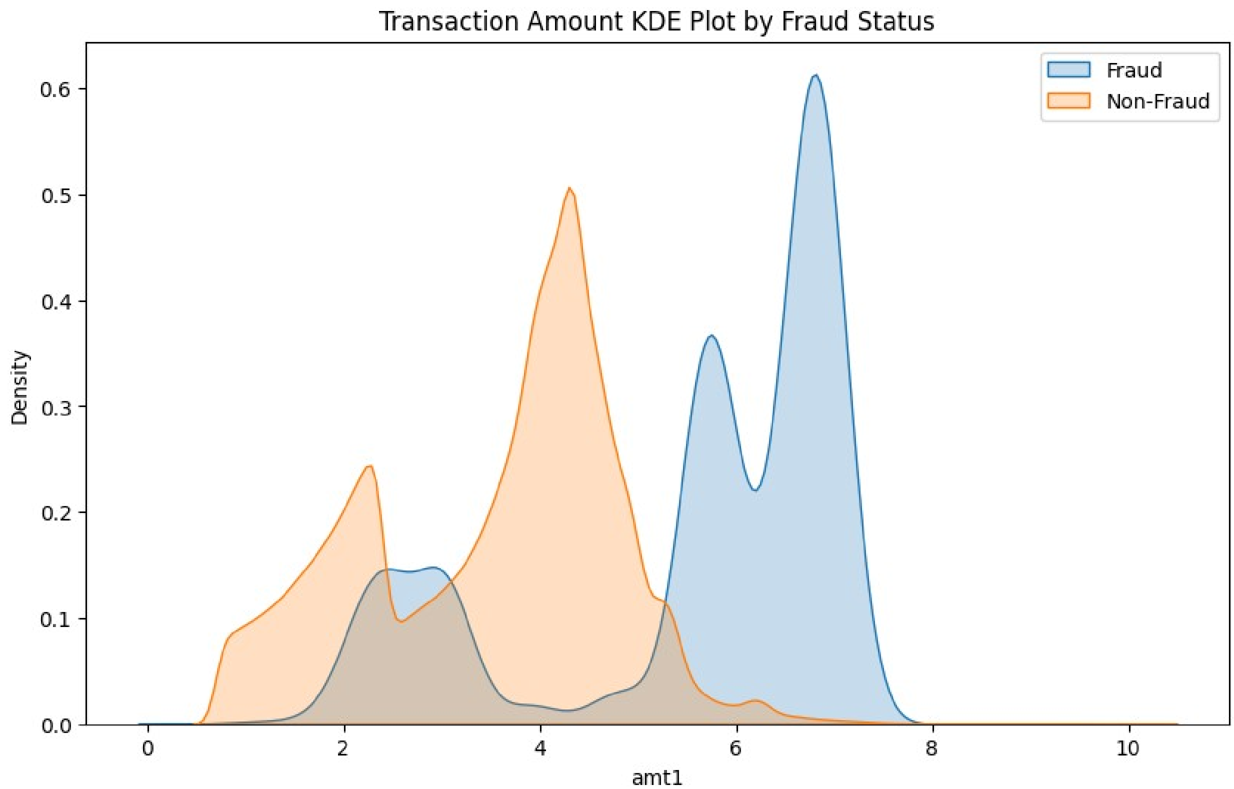

```
test_df['trans_date_trans_time'] =
pd.to_datetime(test_df['trans_date_trans_time'], errors='coerce')


test_df = test_df.dropna(subset=['trans_date_trans_time'])


test_df.loc[:, 'trans_hour'] =
test_df['trans_date_trans_time'].dt.hour
test_df.loc[:, 'trans_day'] = test_df['trans_date_trans_time'].dt.day
test_df.loc[:, 'trans_month'] =
test_df['trans_date_trans_time'].dt.month
test_df.loc[:, 'trans_dayofweek'] =
test_df['trans_date_trans_time'].dt.dayofweek
test_df.loc[:, 'trans_is_weekend'] =
test_df['trans_date_trans_time'].dt.dayofweek >= 5


test_df[['trans_date_trans_time', 'trans_hour', 'trans_day',
'trans_month', 'trans_dayofweek', 'trans_is_weekend']].head()
```


```
{"summary":"{\n  \"name\": \"test_df[['trans_date_trans_time',
'trans_hour', 'trans_day', 'trans_month', 'trans_dayofweek',
'trans_is_weekend']]\",\n  \"rows\": 5,\n  \"fields\": [\n    {\n
\"column\": \"trans_date_trans_time\",\n      \"properties\": {\n
```

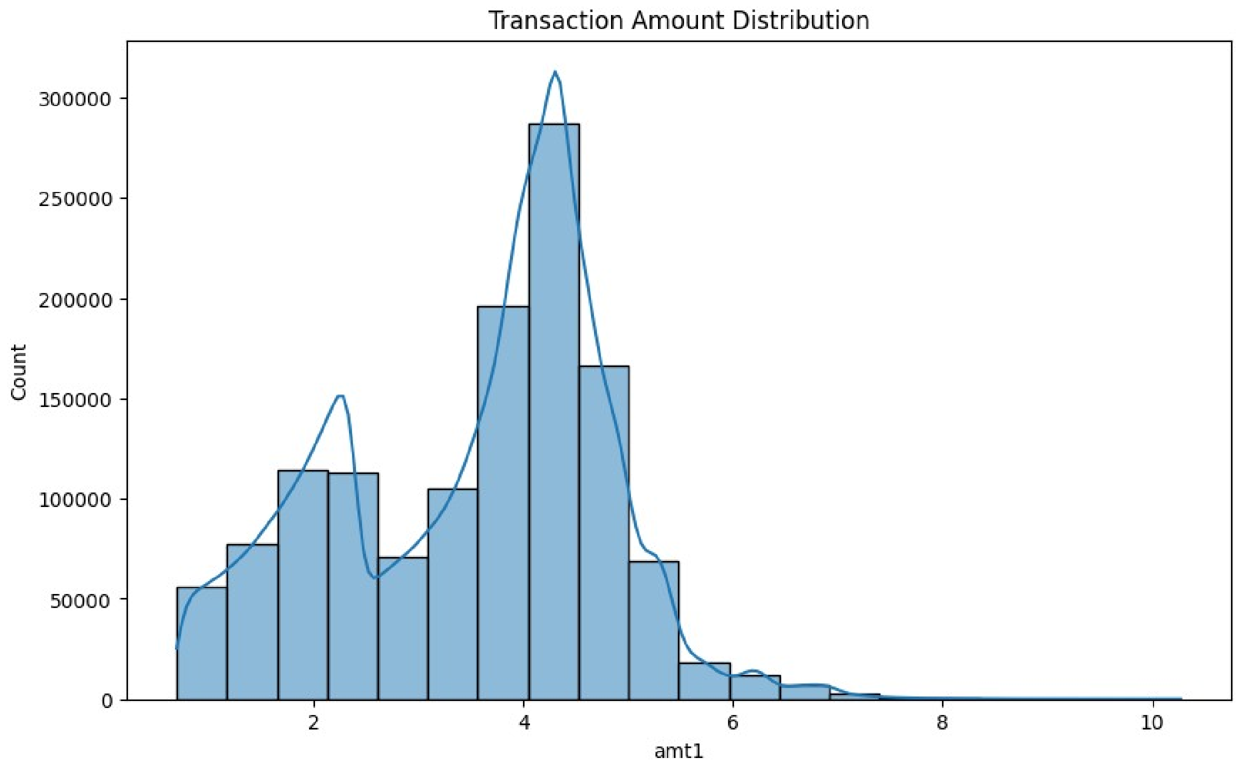
```
\"dtype\": \"date\",\n        \"min\": \"2020-06-21 12:14:25\",\n        \"max\": \"2020-06-21 12:15:17\",\n        \"num_unique_values\": 5,\n        \"samples\": [\n          \"2020-06-21 12:14:33\",\n          \"2020-06-21 12:15:17\",\n          \"2020-06-21 12:14:53\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"trans_hour\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          12\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"trans_day\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          21\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"trans_month\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          6\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"trans_dayofweek\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          6\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"trans_is_weekend\",\n      \"properties\": {\n        \"dtype\": \"boolean\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          true\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```python
#removing the skew from amt

df['amt1'] = np.log1p(df['amt'])

test_df['amt1'] = np.log1p(test_df['amt'])

#KDE
plt.figure(figsize=(10, 6))
sns.kdeplot(df[df['is_fraud'] == 1]['amt1'], label='Fraud', 
shade=True)
sns.kdeplot(df[df['is_fraud'] == 0]['amt1'], label='Non-Fraud', 
fill=True)
plt.title('Transaction Amount KDE Plot by Fraud Status')
plt.legend()
plt.show()
```

```
<ipython-input-24-1554c9388404>:3: FutureWarning: 

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[df['is_fraud'] == 1]['amt1'], label='Fraud', 
shade=True)
```




```
plt.figure(figsize=(10, 6))
sns.histplot(df['amt1'], bins=20, kde=True)
plt.title('Transaction Amount Distribution')
plt.show()
```




```python
import numpy as np


def haversine(lat1, lon1, lat2, lon2):
    """
    Calculate the great circle distance between two points
    on the Earth (specified in decimal degrees).
    """

    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    # Haversine formula
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat / 2)**2 + np.cos(lat1) * np.cos(lat2) *
np.sin(dlon / 2)**2
    c = 2 * np.arcsin(np.sqrt(a))


    r = 6371
    return c * r

# Apply the Haversine function to calculate the customer-merchant
distance
```

```
df['cust_merch_distance'] = df.apply(lambda row: haversine(row['lat'],
row['long'], row['merch_lat'], row['merch_long']), axis=1)

#customer to merchant distance
correlation = df['cust_merch_distance'].corr(df['is_fraud'])
print(f"Correlation between cust_merch_distance and is_fraud:
{correlation}")
```

```
Correlation between cust_merch_distance and is_fraud:
0.0004027341540144967
```

```
test_df['cust_merch_distance'] = test_df.apply(lambda row:
haversine(row['lat'], row['long'], row['merch_lat'],
row['merch_long']), axis=1)

#transaction frequency

df['transaction_count_per_card'] = df.groupby('cc_num')
['trans_num'].transform('count')

test_df['transaction_count_per_card'] = test_df.groupby('cc_num')
['trans_num'].transform('count')

df['dob'] = pd.to_datetime(df['dob'])


df['trans_date_trans_time'] =
pd.to_datetime(df['trans_date_trans_time'])

# Calculate age
df['age'] = df['trans_date_trans_time'].dt.year - df['dob'].dt.year


df['age'] -= (df['trans_date_trans_time'].dt.month <
df['dob'].dt.month) | (
    (df['trans_date_trans_time'].dt.month == df['dob'].dt.month) &
(df['trans_date_trans_time'].dt.day < df['dob'].dt.day)
)


test_df['dob'] = pd.to_datetime(test_df['dob'])


test_df['trans_date_trans_time'] =
pd.to_datetime(test_df['trans_date_trans_time'])


test_df['age'] = test_df['trans_date_trans_time'].dt.year -
test_df['dob'].dt.year


test_df['age'] -= (test_df['trans_date_trans_time'].dt.month <
```

```
test_df['dob'].dt.month) | (
    (test_df['trans_date_trans_time'].dt.month ==
test_df['dob'].dt.month) &
    (test_df['trans_date_trans_time'].dt.day < test_df['dob'].dt.day)
)

#encoding gender
df['gender'] = df['gender'].map({'M': 1, 'F': 0})

#encoding gender
test_df['gender'] = test_df['gender'].map({'M': 1, 'F': 0})

#encoding gender
df['trans_is_weekend'] = df['trans_is_weekend'].map({True: 1, False:
0})

#encoding gender
test_df['trans_is_weekend'] = test_df['trans_is_weekend'].map({True:
1, False: 0})

#Transaction Velocity: Calculate the velocity of transactions over
time (e.g., the number of transactions in a short period).
df['velocity'] = df.groupby('cc_num')
['trans_date_trans_time'].diff().dt.total_seconds().fillna(0)

test_df['velocity'] = test_df.groupby('cc_num')
['trans_date_trans_time'].diff().dt.total_seconds().fillna(0)
```

##7. Feature Selection

```
columns_to_drop = ['unix_time', 'trans_date_trans_time', 'Unnamed: 0',
'trans_num', 'cc_num', 'merchant', 'job', 'street', 'city', 'state',
'zip' , 'dob', 'first','last']
columns_to_drop_existing = [col for col in columns_to_drop if col in
df.columns]

df = df.drop(columns=columns_to_drop_existing)

columns_to_drop = ['unix_time', 'trans_date_trans_time', 'Unnamed: 0',
'trans_num', 'cc_num', 'merchant', 'job', 'street', 'city', 'state',
'zip', 'dob', 'first', 'last']
test_df = test_df.drop(columns=columns_to_drop)

df.shape
```

```
(1296675, 19)
```

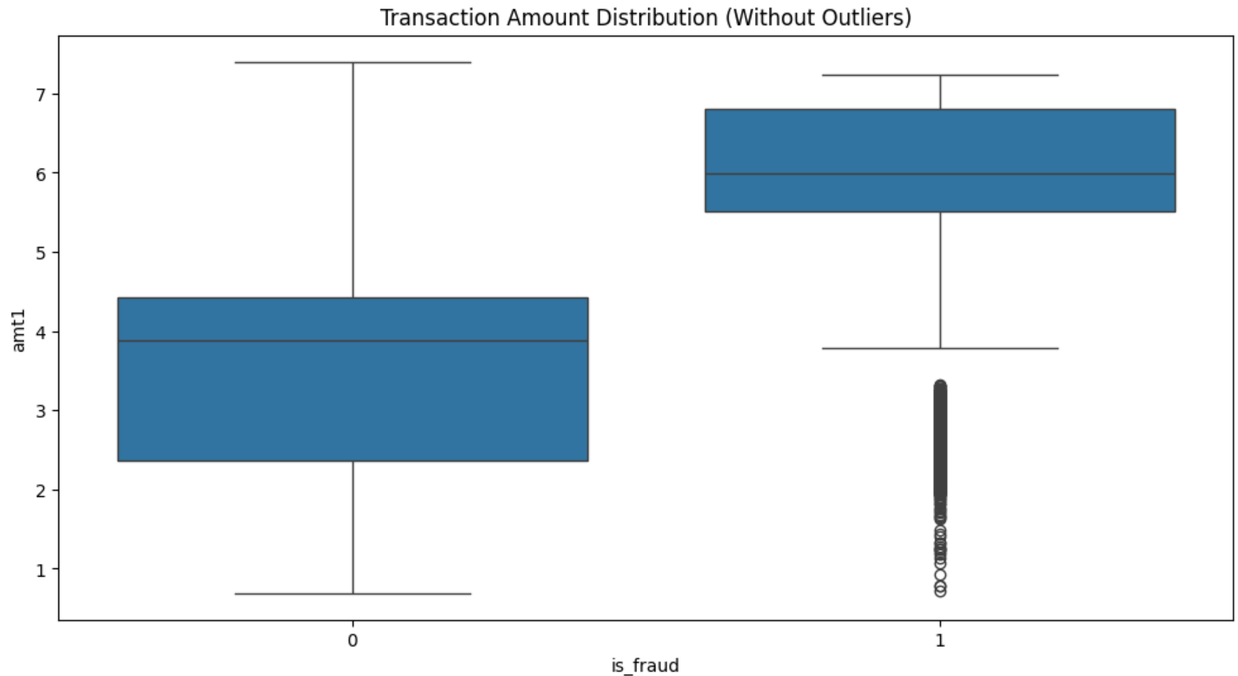
##8. Outlier detection using Z SCORE

```
features = ['amt1']
```

```python
def detect_outliers_z_score(df, features,  threshold=3):
    """Detect unique outlier indices in specified columns of a 
DataFrame using Z-scores.

    This function identifies data points that are considered outliers 
based on the Z-score,
    which measures how many standard deviations a data point is from 
the mean. Outliers are
    those data points with a Z-score greater than the specified 
threshold."""

    outlier_set = set()

    for feature in features:
        # Calculate the Z-scores
        mean = df[feature].mean()
        std_dev = df[feature].std()
        z_scores = abs(df[feature] - mean) / std_dev

        # Adding outlier indices
        outlier_indices = df[z_scores > threshold].index
        outlier_set.update(outlier_indices)

    print("Total Outliers:", len(outlier_set))

    return sorted(outlier_set)

# Detecting Outliers
outlier_indices = detect_outliers_z_score(df, features, threshold=3)

# Removing Outliers
df = df.drop(index=outlier_indices, axis=0).reset_index(drop=True)
```

```
Total Outliers: 1058
```

```python
# Visualzing Features after removal of outlier using Z-Score
plt.figure(figsize=(12, 6))
sns.boxplot(x='is_fraud', y='amt1', data=df)
plt.title('Transaction Amount Distribution (Without Outliers)')
plt.show()
```




```
features = ['amt1']


def detect_outliers_z_score(test_df, features,  threshold=3):


    outlier_set = set()

    for feature in features:

        mean = test_df[feature].mean()
        std_dev = test_df[feature].std()
        z_scores = abs(test_df[feature] - mean) / std_dev


        outlier_indices = test_df[z_scores > threshold].index
        outlier_set.update(outlier_indices)

    print("Total Outliers:", len(outlier_set))

    return sorted(outlier_set)
```

##9.Data Preprocssing Pipeline

```
numeric_features = ['city_pop', 'trans_hour', 'trans_day',
'trans_month', 'trans_dayofweek',
                    'transaction_count_per_card', 'age', 'velocity',
'lat', 'long', 'merch_lat',
```

```
                    'merch_long', 'amt1', 'cust_merch_distance']
categorical_features = ['category', 'gender']

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse=True))
])

df[numeric_features] =
numeric_transformer.fit_transform(df[numeric_features])

# One-hot encode the categorical features
df_encoded = pd.get_dummies(df[categorical_features],
drop_first=True).astype(int)

# Dropping the original categorical columns from the dataframe
df.drop(columns=categorical_features, inplace=True)

# Concatenate the encoded columns to the original dataframe
df = pd.concat([df, df_encoded], axis=1)

df.shape
```

```
(1295617, 31)
```

```
df.sample(5)
```

{"type":"dataframe"}

```
df['is_fraud'].value_counts()
```

```
is_fraud
0    1288111
1       7506
Name: count, dtype: int64
```

```
# One-hot encode the categorical features
df_encoded1 = pd.get_dummies(test_df[categorical_features],
drop_first=True).astype(int)

# Dropping the original categorical columns from the dataframe
test_df.drop(columns=categorical_features, inplace=True)

# Concatenate the encoded columns to the original dataframe
test_df = pd.concat([test_df, df_encoded1], axis=1)

test_df.shape
```

```
(555719, 31)
```

```
X_train = df.drop(columns='is_fraud')
y_train = df['is_fraud']
X_test = test_df.drop(columns='is_fraud')
y_test = test_df['is_fraud']
```

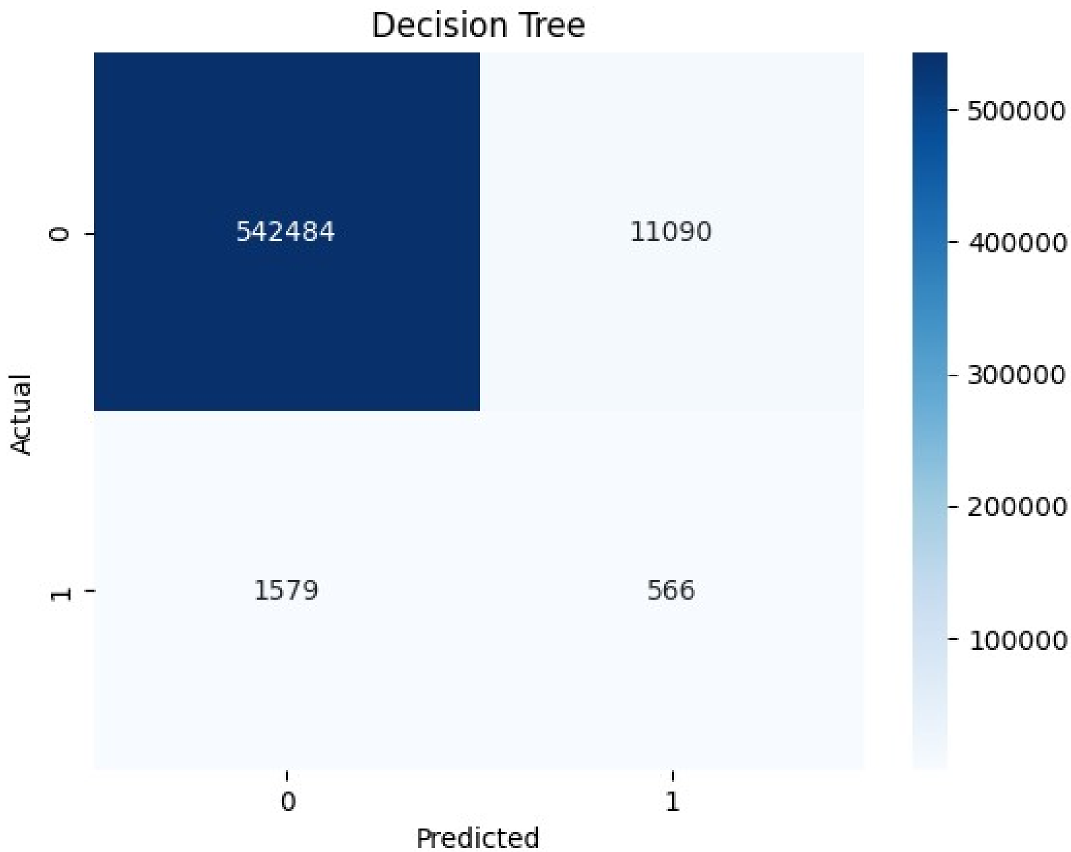
10.Decision Tree

```
from sklearn.metrics import (accuracy_score,
                             f1_score,
                             precision_score,
                             recall_score,
                             RocCurveDisplay,
                             precision_recall_curve,
                             average_precision_score,
                             roc_auc_score,
                             roc_curve, auc,
confusion_matrix,classification_report)

# Decision Tree

from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth =7, random_state=42)
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=7, random_state=42)
```

```
y_pred_dt = dt.predict(X_test)
acc_dt = accuracy_score(y_test, y_pred_dt)
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

print(acc_dt)
sns.heatmap(conf_matrix_dt, annot=True, cmap='Blues', fmt='g')
plt.title('Decision Tree')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
0.977202507022434

Text(50.722222222222214, 0.5, 'Actual')
```




```
print(classification_report(y_test, y_pred_dt))

              precision    recall  f1-score   support

           0       1.00      0.98      0.99    553574
           1       0.05      0.26      0.08      2145

    accuracy                           0.98    555719
   macro avg       0.52      0.62      0.54    555719
weighted avg       0.99      0.98      0.98    555719


from sklearn.tree import plot_tree

from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 40,20

plot_tree(dt)

[Text(0.44040697674418605, 0.9375, 'x[0] <= 712.97\ngini = 0.012\
nsamples = 1295617\nvalue = [1288111, 7506]'),
 Text(0.21608527131782945, 0.8125, 'x[13] <= -2.119\ngini = 0.006\
nsamples = 1287765\nvalue = [1283608, 4157]'),
 Text(0.20833333333333334, 0.6875, 'gini = 0.0\nsamples = 354\nvalue =
```

```
[0, 354]'),
 Text(0.2238372093023256, 0.6875, 'x[0] <= 259.04\ngini = 0.006\
nsamples = 1287411\nvalue = [1283608, 3803]'),
 Text(0.1182170542635659, 0.5625, 'x[6] <= 1.276\ngini = 0.003\
nsamples = 1259902\nvalue = [1258117, 1785]'),
 Text(0.06201550387596899, 0.4375, 'x[6] <= -1.364\ngini = 0.002\
nsamples = 1132312\nvalue = [1131367, 945]'),
 Text(0.031007751937984496, 0.3125, 'x[11] <= -0.241\ngini = 0.008\
nsamples = 165509\nvalue = [164844, 665]'),
 Text(0.015503875968992248, 0.1875, 'x[18] <= 0.5\ngini = 0.031\
nsamples = 40128\nvalue = [39504, 624]'),
 Text(0.007751937984496124, 0.0625, 'gini = 0.007\nsamples = 39631\
nvalue = [39484, 147]'),
 Text(0.023255813953488372, 0.0625, 'gini = 0.077\nsamples = 497\
nvalue = [20, 477]'),
 Text(0.046511627906976744, 0.1875, 'x[0] <= 254.935\ngini = 0.001\
nsamples = 125381\nvalue = [125340, 41]'),
 Text(0.03875968992248062, 0.0625, 'gini = 0.001\nsamples = 125291\
nvalue = [125256, 35]'),
 Text(0.05426356589147287, 0.0625, 'gini = 0.124\nsamples = 90\nvalue
= [84, 6]'),
 Text(0.09302325581395349, 0.3125, 'x[0] <= 24.525\ngini = 0.001\
nsamples = 966803\nvalue = [966523, 280]'),
 Text(0.07751937984496124, 0.1875, 'x[18] <= 0.5\ngini = 0.001\
nsamples = 362553\nvalue = [362328, 225]'),
 Text(0.06976744186046512, 0.0625, 'gini = 0.001\nsamples = 362406\
nvalue = [362254, 152]'),
 Text(0.08527131782945736, 0.0625, 'gini = 0.5\nsamples = 147\nvalue =
[74, 73]'),
 Text(0.10852713178294573, 0.1875, 'x[11] <= 1.486\ngini = 0.0\
nsamples = 604250\nvalue = [604195, 55]'),
 Text(0.10077519379844961, 0.0625, 'gini = 0.0\nsamples = 598065\
nvalue = [598023, 42]'),
 Text(0.11627906976744186, 0.0625, 'gini = 0.004\nsamples = 6185\
nvalue = [6172, 13]'),
 Text(0.1744186046511628, 0.4375, 'x[0] <= 233.525\ngini = 0.013\
nsamples = 127590\nvalue = [126750, 840]'),
 Text(0.15503875968992248, 0.3125, 'x[15] <= -0.621\ngini = 0.012\
nsamples = 126938\nvalue = [126165, 773]'),
 Text(0.13953488372093023, 0.1875, 'x[13] <= -1.705\ngini = 0.036\
nsamples = 21331\nvalue = [20938, 393]'),
 Text(0.13178294573643412, 0.0625, 'gini = 0.277\nsamples = 555\nvalue
= [463, 92]'),
 Text(0.14728682170542637, 0.0625, 'gini = 0.029\nsamples = 20776\
nvalue = [20475, 301]'),
 Text(0.17054263565891473, 0.1875, 'x[15] <= 0.646\ngini = 0.007\
nsamples = 105607\nvalue = [105227, 380]'),
 Text(0.16279069767441862, 0.0625, 'gini = 0.005\nsamples = 90602\
nvalue = [90378, 224]'),
```

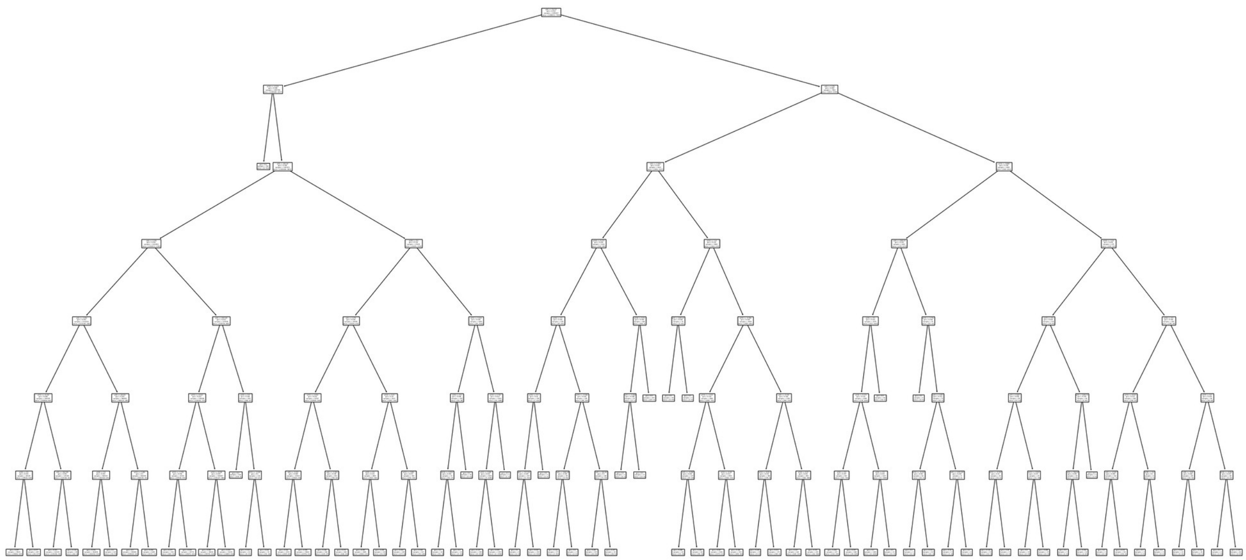
```
 Text(0.17829457364341086, 0.0625, 'gini = 0.021\nsamples = 15005\
nvalue = [14849, 156]'),
 Text(0.1937984496124031, 0.3125, 'x[22] <= 0.5\ngini = 0.184\nsamples
= 652\nvalue = [585, 67]'),
 Text(0.18604651162790697, 0.1875, 'gini = 0.0\nsamples = 518\nvalue =
[518, 0]'),
 Text(0.20155038759689922, 0.1875, 'x[15] <= -0.635\ngini = 0.5\
nsamples = 134\nvalue = [67, 67]'),
 Text(0.1937984496124031, 0.0625, 'gini = 0.314\nsamples = 41\nvalue =
[8, 33]'),
 Text(0.20930232558139536, 0.0625, 'gini = 0.464\nsamples = 93\nvalue
= [59, 34]'),
 Text(0.32945736434108525, 0.5625, 'x[20] <= 0.5\ngini = 0.136\
nsamples = 27509\nvalue = [25491, 2018]'),
 Text(0.27906976744186046, 0.4375, 'x[6] <= 1.276\ngini = 0.036\
nsamples = 25900\nvalue = [25426, 474]'),
 Text(0.24806201550387597, 0.3125, 'x[11] <= 2.304\ngini = 0.014\
nsamples = 22747\nvalue = [22589, 158]'),
 Text(0.23255813953488372, 0.1875, 'x[6] <= -1.511\ngini = 0.009\
nsamples = 21916\nvalue = [21820, 96]'),
 Text(0.2248062015503876, 0.0625, 'gini = 0.042\nsamples = 2074\nvalue
= [2029, 45]'),
 Text(0.24031007751937986, 0.0625, 'gini = 0.005\nsamples = 19842\
nvalue = [19791, 51]'),
 Text(0.26356589147286824, 0.1875, 'x[6] <= -1.364\ngini = 0.138\
nsamples = 831\nvalue = [769, 62]'),
 Text(0.2558139534883721, 0.0625, 'gini = 0.346\nsamples = 166\nvalue
= [129, 37]'),
 Text(0.2713178294573643, 0.0625, 'gini = 0.072\nsamples = 665\nvalue
= [640, 25]'),
 Text(0.31007751937984496, 0.3125, 'x[14] <= 0.23\ngini = 0.18\
nsamples = 3153\nvalue = [2837, 316]'),
 Text(0.29457364341085274, 0.1875, 'x[22] <= 0.5\ngini = 0.121\
nsamples = 2803\nvalue = [2621, 182]'),
 Text(0.2868217054263566, 0.0625, 'gini = 0.083\nsamples = 2685\nvalue
= [2568, 117]'),
 Text(0.3023255813953488, 0.0625, 'gini = 0.495\nsamples = 118\nvalue
= [53, 65]'),
 Text(0.32558139534883723, 0.1875, 'x[11] <= 1.865\ngini = 0.473\
nsamples = 350\nvalue = [216, 134]'),
 Text(0.3178294573643411, 0.0625, 'gini = 0.063\nsamples = 183\nvalue
= [177, 6]'),
 Text(0.3333333333333333, 0.0625, 'gini = 0.358\nsamples = 167\nvalue
= [39, 128]'),
 Text(0.3798449612403101, 0.4375, 'x[11] <= 1.616\ngini = 0.078\
nsamples = 1609\nvalue = [65, 1544]'),
 Text(0.3643410852713178, 0.3125, 'x[16] <= 0.5\ngini = 0.498\nsamples
= 103\nvalue = [55, 48]'),
 Text(0.35658914728682173, 0.1875, 'x[14] <= 0.259\ngini = 0.361\
```

```
nsamples = 72\nvalue = [55, 17]'),
 Text(0.3488372093023256, 0.0625, 'gini = 0.2\nsamples = 62\nvalue =
[55, 7]'),
 Text(0.3643410852713178, 0.0625, 'gini = 0.0\nsamples = 10\nvalue =
[0, 10]'),
 Text(0.37209302325581395, 0.1875, 'gini = 0.0\nsamples = 31\nvalue =
[0, 31]'),
 Text(0.3953488372093023, 0.3125, 'x[13] <= 1.747\ngini = 0.013\
nsamples = 1506\nvalue = [10, 1496]'),
 Text(0.3875968992248062, 0.1875, 'x[11] <= 1.646\ngini = 0.012\
nsamples = 1505\nvalue = [9, 1496]'),
 Text(0.3798449612403101, 0.0625, 'gini = 0.137\nsamples = 108\nvalue
= [8, 100]'),
 Text(0.3953488372093023, 0.0625, 'gini = 0.001\nsamples = 1397\nvalue
= [1, 1396]'),
 Text(0.40310077519379844, 0.1875, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.6647286821705426, 0.8125, 'x[6] <= 1.276\ngini = 0.489\
nsamples = 7852\nvalue = [4503, 3349]'),
 Text(0.5242248062015504, 0.6875, 'x[6] <= -1.364\ngini = 0.325\
nsamples = 5022\nvalue = [3995, 1027]'),
 Text(0.4786821705426357, 0.5625, 'x[14] <= 0.173\ngini = 0.499\
nsamples = 1034\nvalue = [541, 493]'),
 Text(0.44573643410852715, 0.4375, 'x[24] <= 0.5\ngini = 0.256\
nsamples = 578\nvalue = [87, 491]'),
 Text(0.4263565891472868, 0.3125, 'x[16] <= 0.5\ngini = 0.04\nsamples
= 339\nvalue = [7, 332]'),
 Text(0.4186046511627907, 0.1875, 'x[0] <= 717.535\ngini = 0.029\
nsamples = 337\nvalue = [5, 332]'),
 Text(0.4108527131782946, 0.0625, 'gini = 0.5\nsamples = 4\nvalue =
[2, 2]'),
 Text(0.4263565891472868, 0.0625, 'gini = 0.018\nsamples = 333\nvalue
= [3, 330]'),
 Text(0.43410852713178294, 0.1875, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.46511627906976744, 0.3125, 'x[15] <= -0.584\ngini = 0.445\
nsamples = 239\nvalue = [80, 159]'),
 Text(0.4496124031007752, 0.1875, 'x[0] <= 1046.58\ngini = 0.165\
nsamples = 99\nvalue = [9, 90]'),
 Text(0.4418604651162791, 0.0625, 'gini = 0.134\nsamples = 97\nvalue =
[7, 90]'),
 Text(0.4573643410852713, 0.0625, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.4806201550387597, 0.1875, 'x[0] <= 924.89\ngini = 0.5\nsamples
= 140\nvalue = [71, 69]'),
 Text(0.4728682170542636, 0.0625, 'gini = 0.481\nsamples = 107\nvalue
= [43, 64]'),
 Text(0.4883720930232558, 0.0625, 'gini = 0.257\nsamples = 33\nvalue =
[28, 5]'),
```

```
 Text(0.5116279069767442, 0.4375, 'x[14] <= 0.23\ngini = 0.009\
nsamples = 456\nvalue = [454, 2]'),
 Text(0.5038759689922481, 0.3125, 'x[1] <= -1.068\ngini = 0.188\
nsamples = 19\nvalue = [17, 2]'),
 Text(0.49612403100775193, 0.1875, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]'),
 Text(0.5116279069767442, 0.1875, 'gini = 0.0\nsamples = 17\nvalue =
[17, 0]'),
 Text(0.5193798449612403, 0.3125, 'gini = 0.0\nsamples = 437\nvalue =
[437, 0]'),
 Text(0.5697674418604651, 0.5625, 'x[13] <= -1.8\ngini = 0.232\
nsamples = 3988\nvalue = [3454, 534]'),
 Text(0.5426356589147286, 0.4375, 'x[0] <= 1209.47\ngini = 0.028\
nsamples = 70\nvalue = [1, 69]'),
 Text(0.5348837209302325, 0.3125, 'gini = 0.0\nsamples = 69\nvalue =
[0, 69]'),
 Text(0.5503875968992248, 0.3125, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.5968992248062015, 0.4375, 'x[15] <= -0.006\ngini = 0.209\
nsamples = 3918\nvalue = [3453, 465]'),
 Text(0.5658914728682171, 0.3125, 'x[13] <= -1.113\ngini = 0.132\
nsamples = 2682\nvalue = [2491, 191]'),
 Text(0.5503875968992248, 0.1875, 'x[3] <= -0.288\ngini = 0.432\
nsamples = 127\nvalue = [87, 40]'),
 Text(0.5426356589147286, 0.0625, 'gini = 0.488\nsamples = 45\nvalue =
[19, 26]'),
 Text(0.5581395348837209, 0.0625, 'gini = 0.283\nsamples = 82\nvalue =
[68, 14]'),
 Text(0.5813953488372093, 0.1875, 'x[15] <= -0.44\ngini = 0.111\
nsamples = 2555\nvalue = [2404, 151]'),
 Text(0.5736434108527132, 0.0625, 'gini = 0.067\nsamples = 1636\nvalue
= [1579, 57]'),
 Text(0.5891472868217055, 0.0625, 'gini = 0.184\nsamples = 919\nvalue
= [825, 94]'),
 Text(0.627906976744186, 0.3125, 'x[6] <= -0.191\ngini = 0.345\
nsamples = 1236\nvalue = [962, 274]'),
 Text(0.6124031007751938, 0.1875, 'x[15] <= 0.011\ngini = 0.031\
nsamples = 384\nvalue = [378, 6]'),
 Text(0.6046511627906976, 0.0625, 'gini = 0.426\nsamples = 13\nvalue =
[9, 4]'),
 Text(0.6201550387596899, 0.0625, 'gini = 0.011\nsamples = 371\nvalue
= [369, 2]'),
 Text(0.6434108527131783, 0.1875, 'x[14] <= 0.23\ngini = 0.431\
nsamples = 852\nvalue = [584, 268]'),
 Text(0.6356589147286822, 0.0625, 'gini = 0.267\nsamples = 642\nvalue
= [540, 102]'),
 Text(0.6511627906976745, 0.0625, 'gini = 0.331\nsamples = 210\nvalue
= [44, 166]'),
 Text(0.8052325581395349, 0.6875, 'x[13] <= 0.888\ngini = 0.295\
```

```
nsamples = 2830\nvalue = [508, 2322]'),
 Text(0.7209302325581395, 0.5625, 'x[0] <= 1258.72\ngini = 0.202\
nsamples = 2451\nvalue = [279, 2172]'),
 Text(0.6976744186046512, 0.4375, 'x[25] <= 0.5\ngini = 0.163\nsamples
= 2372\nvalue = [212, 2160]'),
 Text(0.689922480620155, 0.3125, 'x[14] <= 0.173\ngini = 0.141\
nsamples = 2338\nvalue = [178, 2160]'),
 Text(0.6744186046511628, 0.1875, 'x[16] <= 0.5\ngini = 0.297\nsamples
= 921\nvalue = [167, 754]'),
 Text(0.6666666666666666, 0.0625, 'gini = 0.494\nsamples = 237\nvalue
= [132, 105]'),
 Text(0.6821705426356589, 0.0625, 'gini = 0.097\nsamples = 684\nvalue
= [35, 649]'),
 Text(0.7054263565891473, 0.1875, 'x[29] <= 0.5\ngini = 0.015\nsamples
= 1417\nvalue = [11, 1406]'),
 Text(0.6976744186046512, 0.0625, 'gini = 0.0\nsamples = 1406\nvalue =
[0, 1406]'),
 Text(0.7131782945736435, 0.0625, 'gini = 0.0\nsamples = 11\nvalue =
[11, 0]'),
 Text(0.7054263565891473, 0.3125, 'gini = 0.0\nsamples = 34\nvalue =
[34, 0]'),
 Text(0.7441860465116279, 0.4375, 'x[13] <= -1.791\ngini = 0.258\
nsamples = 79\nvalue = [67, 12]'),
 Text(0.7364341085271318, 0.3125, 'gini = 0.0\nsamples = 6\nvalue =
[0, 6]'),
 Text(0.751937984496124, 0.3125, 'x[4] <= 1.307\ngini = 0.151\nsamples
= 73\nvalue = [67, 6]'),
 Text(0.7364341085271318, 0.1875, 'x[15] <= -0.674\ngini = 0.061\
nsamples = 64\nvalue = [62, 2]'),
 Text(0.7286821705426356, 0.0625, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.7441860465116279, 0.0625, 'gini = 0.031\nsamples = 63\nvalue =
[62, 1]'),
 Text(0.7674418604651163, 0.1875, 'x[0] <= 1379.225\ngini = 0.494\
nsamples = 9\nvalue = [5, 4]'),
 Text(0.7596899224806202, 0.0625, 'gini = 0.32\nsamples = 5\nvalue =
[1, 4]'),
 Text(0.7751937984496124, 0.0625, 'gini = 0.0\nsamples = 4\nvalue =
[4, 0]'),
 Text(0.8895348837209303, 0.5625, 'x[14] <= -1.15\ngini = 0.478\
nsamples = 379\nvalue = [229, 150]'),
 Text(0.8410852713178295, 0.4375, 'x[27] <= 0.5\ngini = 0.409\nsamples
= 115\nvalue = [33, 82]'),
 Text(0.813953488372093, 0.3125, 'x[7] <= 0.047\ngini = 0.495\nsamples
= 49\nvalue = [27, 22]'),
 Text(0.7984496124031008, 0.1875, 'x[15] <= -0.155\ngini = 0.32\
nsamples = 20\nvalue = [16, 4]'),
 Text(0.7906976744186046, 0.0625, 'gini = 0.124\nsamples = 15\nvalue =
[14, 1]'),
```

```
 Text(0.8062015503875969, 0.0625, 'gini = 0.48\nsamples = 5\nvalue =
[2, 3]'),
 Text(0.8294573643410853, 0.1875, 'x[11] <= 2.555\ngini = 0.471\
nsamples = 29\nvalue = [11, 18]'),
 Text(0.8217054263565892, 0.0625, 'gini = 0.0\nsamples = 10\nvalue =
[0, 10]'),
 Text(0.8372093023255814, 0.0625, 'gini = 0.488\nsamples = 19\nvalue =
[11, 8]'),
 Text(0.8682170542635659, 0.3125, 'x[0] <= 1294.73\ngini = 0.165\
nsamples = 66\nvalue = [6, 60]'),
 Text(0.8604651162790697, 0.1875, 'x[0] <= 764.66\ngini = 0.091\
nsamples = 63\nvalue = [3, 60]'),
 Text(0.8527131782945736, 0.0625, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.8682170542635659, 0.0625, 'gini = 0.062\nsamples = 62\nvalue =
[2, 60]'),
 Text(0.875968992248062, 0.1875, 'gini = 0.0\nsamples = 3\nvalue = [3,
0]'),
 Text(0.937984496124031, 0.4375, 'x[16] <= 0.5\ngini = 0.382\nsamples
= 264\nvalue = [196, 68]'),
 Text(0.9069767441860465, 0.3125, 'x[15] <= 0.627\ngini = 0.112\
nsamples = 168\nvalue = [158, 10]'),
 Text(0.8914728682170543, 0.1875, 'x[5] <= -2.305\ngini = 0.083\
nsamples = 162\nvalue = [155, 7]'),
 Text(0.8837209302325582, 0.0625, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.8992248062015504, 0.0625, 'gini = 0.072\nsamples = 161\nvalue
= [155, 6]'),
 Text(0.9224806201550387, 0.1875, 'x[28] <= 0.5\ngini = 0.5\nsamples =
6\nvalue = [3, 3]'),
 Text(0.9147286821705426, 0.0625, 'gini = 0.0\nsamples = 3\nvalue =
[3, 0]'),
 Text(0.9302325581395349, 0.0625, 'gini = 0.0\nsamples = 3\nvalue =
[0, 3]'),
 Text(0.9689922480620154, 0.3125, 'x[27] <= 0.5\ngini = 0.478\nsamples
= 96\nvalue = [38, 58]'),
 Text(0.9534883720930233, 0.1875, 'x[8] <= -0.48\ngini = 0.444\
nsamples = 48\nvalue = [32, 16]'),
 Text(0.9457364341085271, 0.0625, 'gini = 0.463\nsamples = 22\nvalue =
[8, 14]'),
 Text(0.9612403100775194, 0.0625, 'gini = 0.142\nsamples = 26\nvalue =
[24, 2]'),
 Text(0.9844961240310077, 0.1875, 'x[14] <= -0.862\ngini = 0.219\
nsamples = 48\nvalue = [6, 42]'),
 Text(0.9767441860465116, 0.0625, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.9922480620155039, 0.0625, 'gini = 0.159\nsamples = 46\nvalue =
[4, 42]')]
```

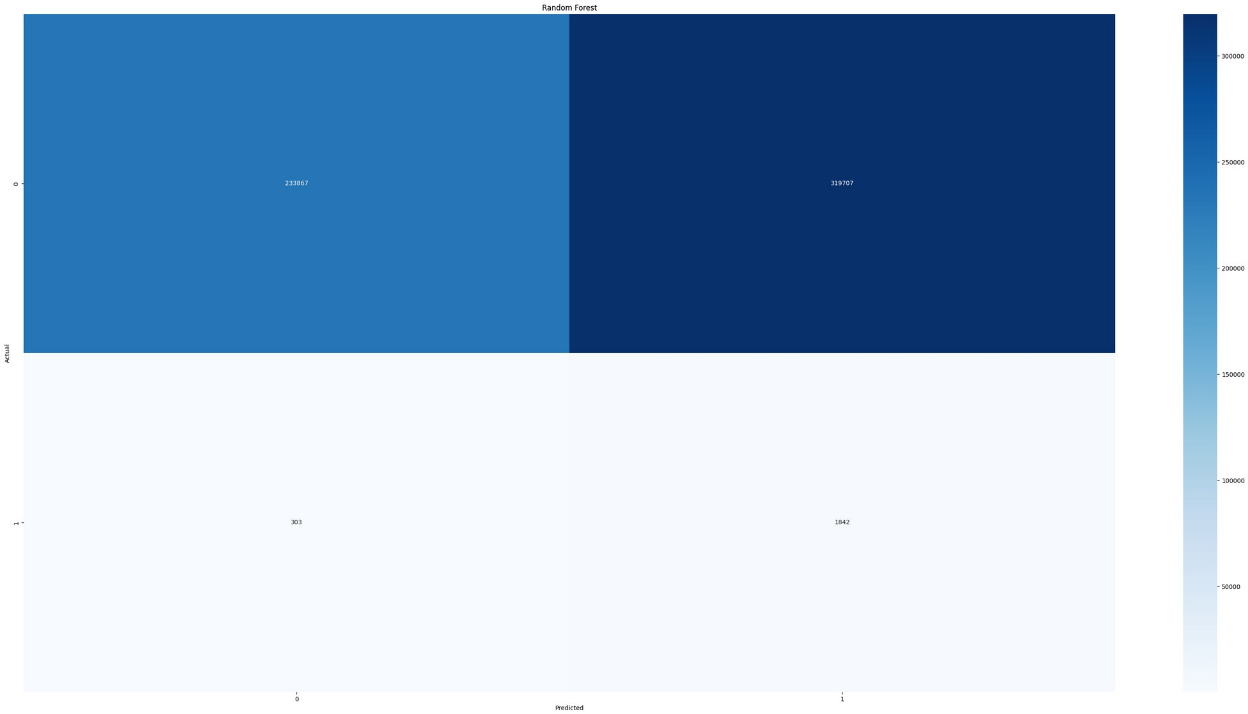


##11. Random Forest

```
from sklearn.ensemble import RandomForestClassifier


rf = RandomForestClassifier(random_state=42, max_depth=15)

rf.fit(X_train, y_train)


y_pred_rf = rf.predict(X_test)

acc_rf = accuracy_score(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

print(acc_rf)
sns.heatmap(conf_matrix_rf, annot=True, cmap='Blues', fmt='g')
plt.title('Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
print(classification_report(y_test, y_pred_rf))
```

```
0.42415141465380884
              precision    recall  f1-score   support

           0       1.00      0.42      0.59    553574
           1       0.01      0.86      0.01      2145

    accuracy                           0.42    555719
   macro avg       0.50      0.64      0.30    555719
```

```
weighted avg       0.99      0.42      0.59    555719
```



##12. XGBOOST

```python
import matplotlib.pyplot as plt
# XGBOOST
xgb = XGBClassifier(random_state=42)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
acc_xgb = accuracy_score(y_test, y_pred_xgb)
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)
print(acc_xgb)
sns.heatmap(conf_matrix_xgb, annot=True, cmap='Blues', fmt='g')
plt.title('XGBOOST')
plt.xlabel('Predicted')
plt.ylabel('Actual')
print(classification_report(y_test, y_pred_xgb))
```

```
0.9905383836075426
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    553574
           1       0.23      0.63      0.34      2145

    accuracy                           0.99    555719
   macro avg       0.62      0.81      0.67    555719
```

```
weighted avg       1.00      0.99      0.99    555719
```