starter-code

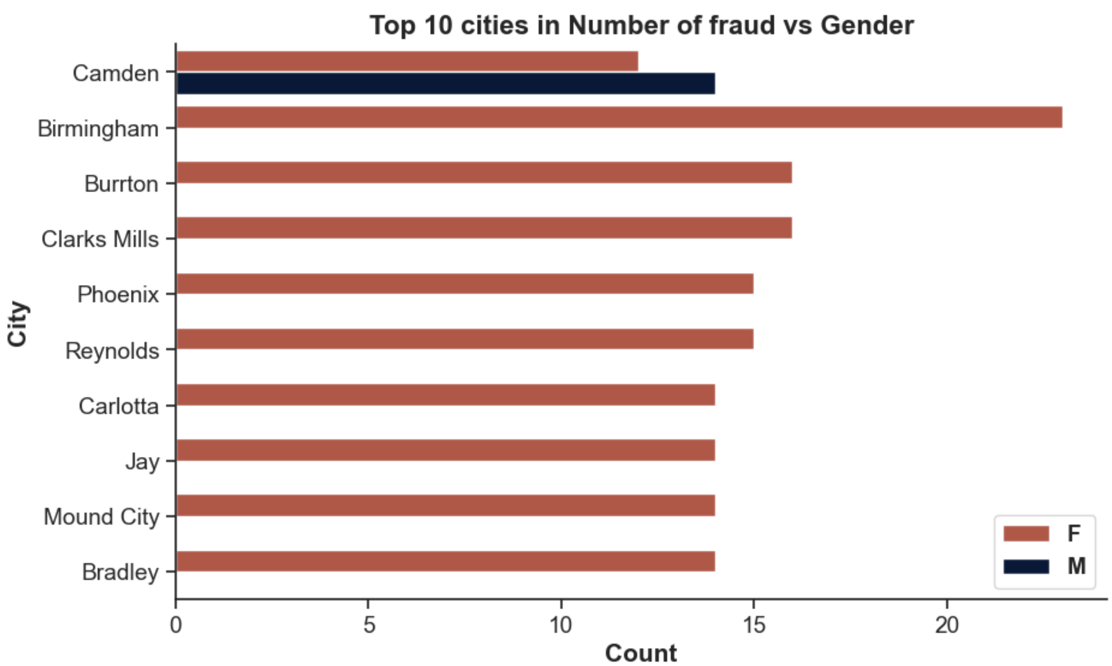
April 1, 2024

0.1 Exploration

[256]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df_train = pd.read_csv("./data/train.csv")
df_test = pd.read_csv("./data/test.csv")

df_combined = pd.concat([df_train, df_test], axis=0)

df = df_combined

df_fraud = df[df.is_fraud==1]

print('Shape for preprocessed train dataset: \n', df.shape)
print('Shape for only-fraud train dataset: \n', df_fraud.shape)

fig = plt.figure(figsize=(10,6))
# sns.set()
colors = ["#C04C36", "#00163E"]
custom_params = {"axes.spines.right": False, "axes.spines.top": False}
sns.set_theme(style="ticks", rc=custom_params)
sns.set_context("notebook", font_scale=1.25)
sns.set_palette(sns.color_palette(colors))

# Create count plot with region on the y-axis
g = sns.countplot(y = 'city',
                  data=df_fraud,
                  hue='gender',
                  width=0.8,
                  order=df_fraud.city.value_counts(sort=True, ascending=False).
 ↪head(10).index)

# Set title, label, legend
g.set_title('Top 10 cities in Number of fraud vs Gender', fontdict = {␣
 ↪'fontsize': 16, 'fontweight':'bold'})
```

```
g.set_xlabel('Count', fontsize=15, fontweight='bold')
g.set_ylabel('City', fontsize=15, fontweight='bold')
g.legend(prop={'weight':'bold'})

# Show plot
plt.show()
```

```
Shape for preprocessed train dataset:
 (625184, 23)
Shape for only-fraud train dataset:
 (1877, 23)
```



0.2 Feature Extraction

[311]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from geopy.distance import great_circle
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

def calculate_age(dob, trans_date):
    age = trans_date.year - dob.year - ((trans_date.month, trans_date.day) <␣
↪(dob.month, dob.day))
    return age
```

```python
def calculate_distance(row):
    customer_loc = (row['lat'], row['long'])
    merchant_loc = (row['merch_lat'], row['merch_long'])
    return great_circle(customer_loc, merchant_loc).km

def cluster_locations(df):
    print("Performing location clustering...")
    # Scaling the location data
    scaler = StandardScaler()
    loc_df_scaled = scaler.fit_transform(df[['lat', 'long']])
    kmeans = KMeans(n_clusters=20, random_state=0, n_init='auto').fit(loc_df_scaled)
    df['location_cluster'] = kmeans.labels_
    print("Location clustering completed.")

    return df

def calculate_rfm_features(df, window_days):
    """Calculates frequency and average monetary value within a time window"""

    now = df['trans_date_trans_time'].max()
    window_start = now - pd.Timedelta(days=window_days)

    window_transactions = df[df['trans_date_trans_time'] >= window_start]

    rfm_features = window_transactions.groupby('cc_num')[['amt']].agg(['count', 'mean'])
    rfm_features.columns = [f'freq_{window_days}days', f'avg_amt_{window_days}days']

    return rfm_features

def calculate_merchant_risk_scores(df, window_days, delay_days):
    """Calculates terminal risk scores within time windows, with a delay"""

    now = df['trans_date_trans_time'].max()
    window_end = now - pd.Timedelta(days=delay_days)
    window_start = window_end - pd.Timedelta(days=window_days)

    window_transactions = df[(df['trans_date_trans_time'] >= window_start) & (df['trans_date_trans_time'] < window_end)]

    terminal_risk = window_transactions.groupby('merchant')['is_fraud'].mean()
    terminal_risk.name = f'risk_score_{window_days}days'

    return terminal_risk
```

```python
def calculate_category_risk_factors(df, window_days, delay_days):
    """Calculates category risk factors within time windows, with a delay"""

    now = df['trans_date_trans_time'].max()
    window_end = now - pd.Timedelta(days=delay_days)
    window_start = window_end - pd.Timedelta(days=window_days)

    window_transactions = df[(df['trans_date_trans_time'] >= window_start) & (df['trans_date_trans_time'] < window_end)]

    category_risk = window_transactions.groupby('category')['is_fraud'].mean()
    category_risk.name = f'category_risk_{window_days}days'

    return category_risk

def extract_time_category_features(df):
    # Convert transaction datetime to hour of the day
    df['trans_hour'] = df['trans_time'].apply(lambda x: x.hour)

    df['category_hour'] = df['category'].astype(str) + "_" + df['trans_hour'].astype(str)

    # To use these new interaction features in a model, you need to encode them
    # This encoding could be label encoding or one-hot encoding. Here's an example with label encoding:

    label_encoder = LabelEncoder()
    df['category_hour_encoded'] = label_encoder.fit_transform(df['category_hour'])

    return df

def calculate_odds_ratios(df, category_col='category_encoded'):
    # Compute the number of fraud and non-fraud transactions
    fraud_counts = df[df['is_fraud'] == 1][category_col].value_counts().sort_index()
    non_fraud_counts = df[df['is_fraud'] == 0][category_col].value_counts().sort_index()

    # Calculate the odds for each category
    odds = fraud_counts / non_fraud_counts

    # Calculate the overall odds of fraud
    overall_odds = df['is_fraud'].mean()

    # Calculate odds ratios (odds for each category / overall odds)
```

```
    odds_ratios = odds / overall_odds

    # Replace infinite values with a large number
    odds_ratios = odds_ratios.replace(np.inf, np.finfo(float).max)

    return odds_ratios

def process(df):
    print("Starting data processing...")
    # Convert to datetime just once
    df['dob'] = pd.to_datetime(df['dob'], format='%d/%m/%Y')
    df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'], format='%d/%m/%Y %H:%M')

    # Extracting date and time for each row
    df['trans_date'] = df['trans_date_trans_time'].dt.date
    df['trans_time'] = df['trans_date_trans_time'].dt.time

    print("Cleaning data and calculating basic statistics...")

    # Transaction amount statistics for each cardholder
    agg_funcs = ['mean', 'std', 'min', 'max']
    cardholder_stats = df.groupby('cc_num')['amt'].agg(agg_funcs).rename(columns=dict(zip(agg_funcs, ['mean', 'std', 'min', 'max'])))
    df = df.join(cardholder_stats, on='cc_num', rsuffix='_cardholder')

    # Encode categorical features
    categorical_features = ['merchant', 'city', 'state', 'job']
    df[categorical_features] = df[categorical_features].apply(LabelEncoder().fit_transform)

    df['category_encoded'] = LabelEncoder().fit_transform(df['category'])
    df['gender_encoded'] = LabelEncoder().fit_transform(df['gender'])

    # Calculate odds ratios for each category
    odds_ratios = calculate_odds_ratios(df, category_col='category_encoded')
    df['category_odds_ratio'] = df['category_encoded'].map(odds_ratios).fillna(1)

    print("Calculating age at transaction and distances...")
    # Age at transaction
    df['age_at_transaction'] = df.apply(lambda x: calculate_age(x['dob'], x['trans_date']), axis=1)

    # Geographical feature: Distance between customer and merchant
    df['cust_merch_distance'] = df.apply(calculate_distance, axis=1)
```

```python
    df['trans_weekend'] = df['trans_date_trans_time'].dt.weekday >= 5
    df['trans_night'] = df['trans_date_trans_time'].dt.hour.apply(lambda x: 1␣
↪if 0 <= x < 6 or 20 <= x < 24 else 0)

    for window_days in [1, 7, 30]:
        rfm_features = calculate_rfm_features(df.copy(), window_days)
        df = df.merge(rfm_features, how='left', on='cc_num')

        terminal_risk = calculate_merchant_risk_scores(df.copy(), window_days,␣
↪delay_days=7)
        df = df.merge(terminal_risk, how='left', on='merchant')

        category_risk = calculate_category_risk_factors(df.copy(), window_days,␣
↪delay_days=7)
        df = df.merge(category_risk, how='left', on='category')

    # Perform location clustering
    df = cluster_locations(df)

    df = extract_time_category_features(df)

    df.drop(columns=['first', 'last', 'street', 'gender',␣
↪'trans_date_trans_time', 'trans_date', 'trans_time', 'cc_num'], inplace=True)

    print("Data processing completed.")

    return df

# Load the dataset
trainingSet = pd.read_csv("./data/train.csv")

# Process the DataFrame
print("Processing training set...")
train_processed = process(trainingSet)
print("Training set processed.")

# Load test set
submissionSet = pd.read_csv("./data/test.csv")

# Merge on Id so that the test set can have feature columns as well
testX= pd.merge(train_processed, submissionSet, left_on='Id', right_on='Id')
testX = testX.drop(columns=['is_fraud_x'])
testX = testX.rename(columns={'is_fraud_y': 'is_fraud'})

# The training set is where the score is not null
trainX =  train_processed[train_processed['is_fraud'].notnull()]
```

```python
# Save the datasets with the new features for easy access later
testX.to_csv("./data/X_test.csv", index=False)
trainX.to_csv("./data/X_train.csv", index=False)
```

```
Processing training set…
Starting data processing…
Cleaning data and calculating basic statistics…
Calculating age at transaction and distances…
Performing location clustering…
Location clustering completed.
Data processing completed.
Training set processed.
```

0.3 Creating your model

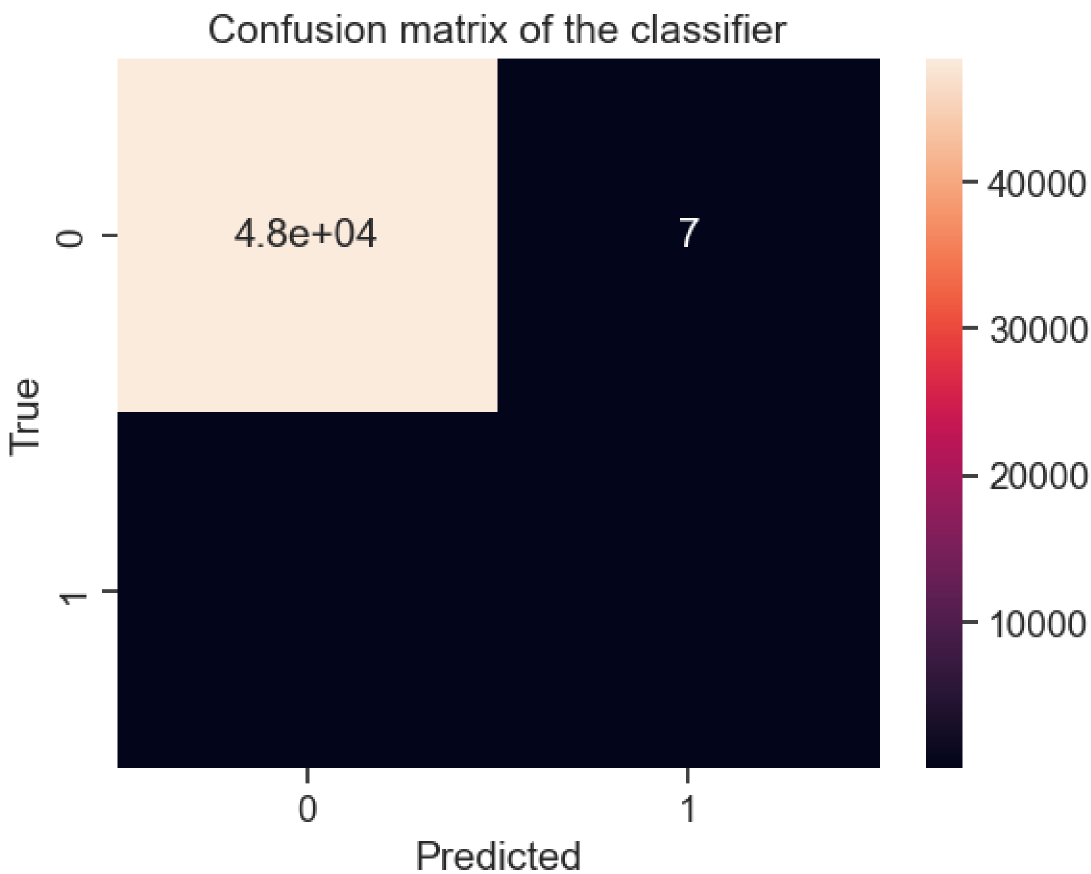
[312]:
```python
import pickle
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from xgboost import XGBClassifier

# Load training set with new features into DataFrame
X_train = pd.read_csv("./data/X_train.csv")

# Split training set into training and testing set
X_train, X_test, Y_train, Y_test = train_test_split(
        X_train.drop(['is_fraud', 'Id'], axis=1),
        X_train['is_fraud'],
        test_size=0.1,
        random_state=42
    )

# This is where you can do more feature selection
X_train_processed = X_train._get_numeric_data()
print(X_train_processed.columns)
X_test_processed = X_test._get_numeric_data()

# Define XGBoost model (you might want to tune these)
xgb_model = XGBClassifier(random_state=0, use_label_encoder=False,␣
 ↪eval_metric='logloss')

# Learn the model
xgb_model.fit(X_train_processed, Y_train)
```

```python
# Pickle model
with open('xgboost_model.obj', 'wb') as f:
    pickle.dump(xgb_model, f)

# Evaluate on the testing set
Y_test_predictions = xgb_model.predict(X_test_processed)
print("Accuracy on testing set = ", accuracy_score(Y_test, Y_test_predictions))
print("F1 score on testing set = ", f1_score(Y_test, Y_test_predictions))

# Plot a confusion matrix
cm = confusion_matrix(Y_test, Y_test_predictions)
sns.heatmap(cm, annot=True)
plt.title('Confusion matrix of the classifier')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Index(['merchant', 'amt', 'city', 'state', 'zip', 'lat', 'long', 'city_pop',
       'job', 'unix_time', 'merch_lat', 'merch_long', 'mean', 'std', 'min',
       'max', 'category_encoded', 'gender_encoded', 'category_odds_ratio',
       'gender_odds_ratio', 'age_at_transaction', 'cust_merch_distance',
       'trans_weekend', 'trans_night', 'freq_1days', 'avg_amt_1days',
       'risk_score_1days', 'category_risk_1days', 'freq_7days',
       'avg_amt_7days', 'risk_score_7days', 'category_risk_7days',
       'freq_30days', 'avg_amt_30days', 'risk_score_30days',
       'category_risk_30days', 'location_cluster', 'trans_hour',
       'category_hour_encoded'],
      dtype='object')
```

```
`use_label_encoder` is deprecated in 1.7.0.
```

```
Accuracy on testing set =  0.9993830461070209
F1 score on testing set =  0.9019607843137255
```



[308]:
```
X_train_processed.head()
```

[308]:
```
        merchant     amt  city  state    zip      lat     long  city_pop  job  \
230198        70   37.81   690     47  54559  46.4959 -90.4383       795  446
201736       271   16.46   542      2  71960  34.4596 -93.6743      1383  265
426780       100  103.05   733     22  55080  45.6675 -93.2433      2607  254
17706        178   94.31   243     30   7022  40.8170 -74.0000     13835  354
180397        33  133.62   123     47  53924  43.4987 -90.2796      1360  302

         unix_time  …  avg_amt_7days  risk_score_7days  category_risk_7days  \
230198  1375853554  …      85.199149               0.0             0.000000
201736  1387011410  …      71.865179               0.0             0.001496
426780  1375130700  …      45.662535               0.0             0.001482
17706   1377325280  …      60.025455               0.0             0.000000
180397  1378635449  …      87.870625               0.0             0.001948

        freq_30days  avg_amt_30days  risk_score_30days  category_risk_30days  \
230198        207.0       93.963188           0.010526              0.001185
201736        202.0      131.916040           0.007353              0.006904
```

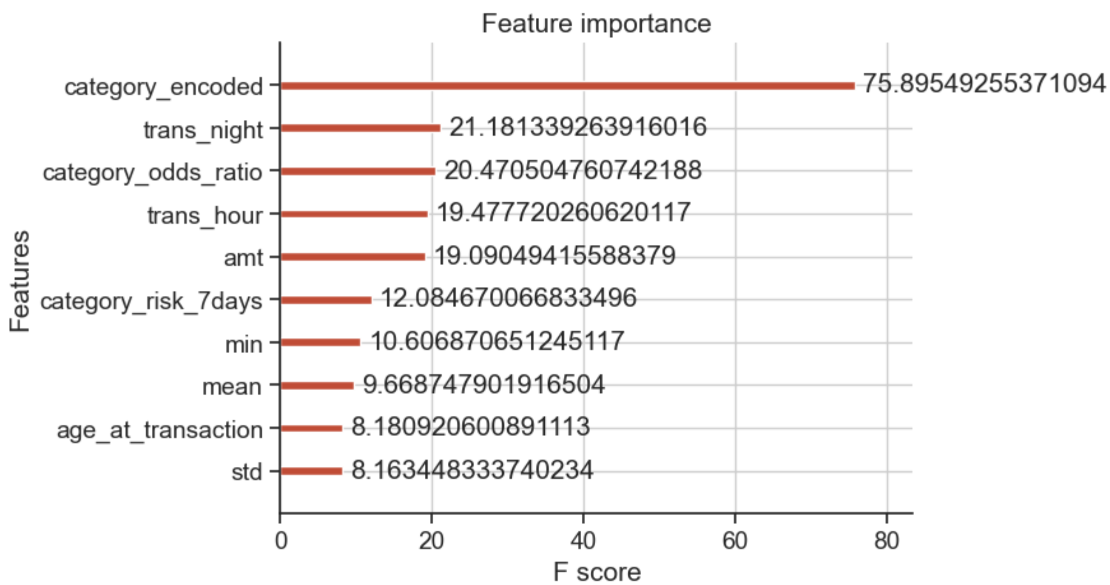
```
426780     236.0   72.610085     0.000000         0.001551
17706      161.0   55.785217     0.000000         0.000639
180397     159.0   98.946667     0.005464         0.003979

        location_cluster  trans_hour  category_hour_encoded
230198                 0           5                    175
201736                11           8                    154
426780                 0          20                    116
17706                  6           6                     20
180397                 9          10                    218

[5 rows x 40 columns]
```

[313]:

```python
# After fitting your XGBoost model (xgb_model)
importances = xgb_model.feature_importances_
feature_names = X_train_processed.columns  # Replace with the names of your␣
↪features

sorted_features = [feature for _, feature in sorted(zip(importances,␣
↪feature_names), reverse=True)]
for i in sorted_features:
    print(i)

import matplotlib.pyplot as plt
import xgboost as xgb

# If you use 'gain' as your importance_type:
xgb.plot_importance(xgb_model, importance_type='gain', max_num_features=10)
plt.show()
```

```
category_encoded
trans_night
category_odds_ratio
trans_hour
amt
category_risk_7days
min
mean
age_at_transaction
std
freq_30days
category_risk_30days
gender_encoded
max
risk_score_30days
city_pop
freq_1days
```

```
category_hour_encoded
unix_time
avg_amt_30days
freq_7days
avg_amt_7days
avg_amt_1days
risk_score_7days
lat
state
trans_weekend
long
job
city
location_cluster
zip
merch_long
cust_merch_distance
merch_lat
merchant
risk_score_1days
gender_odds_ratio
category_risk_1days
```

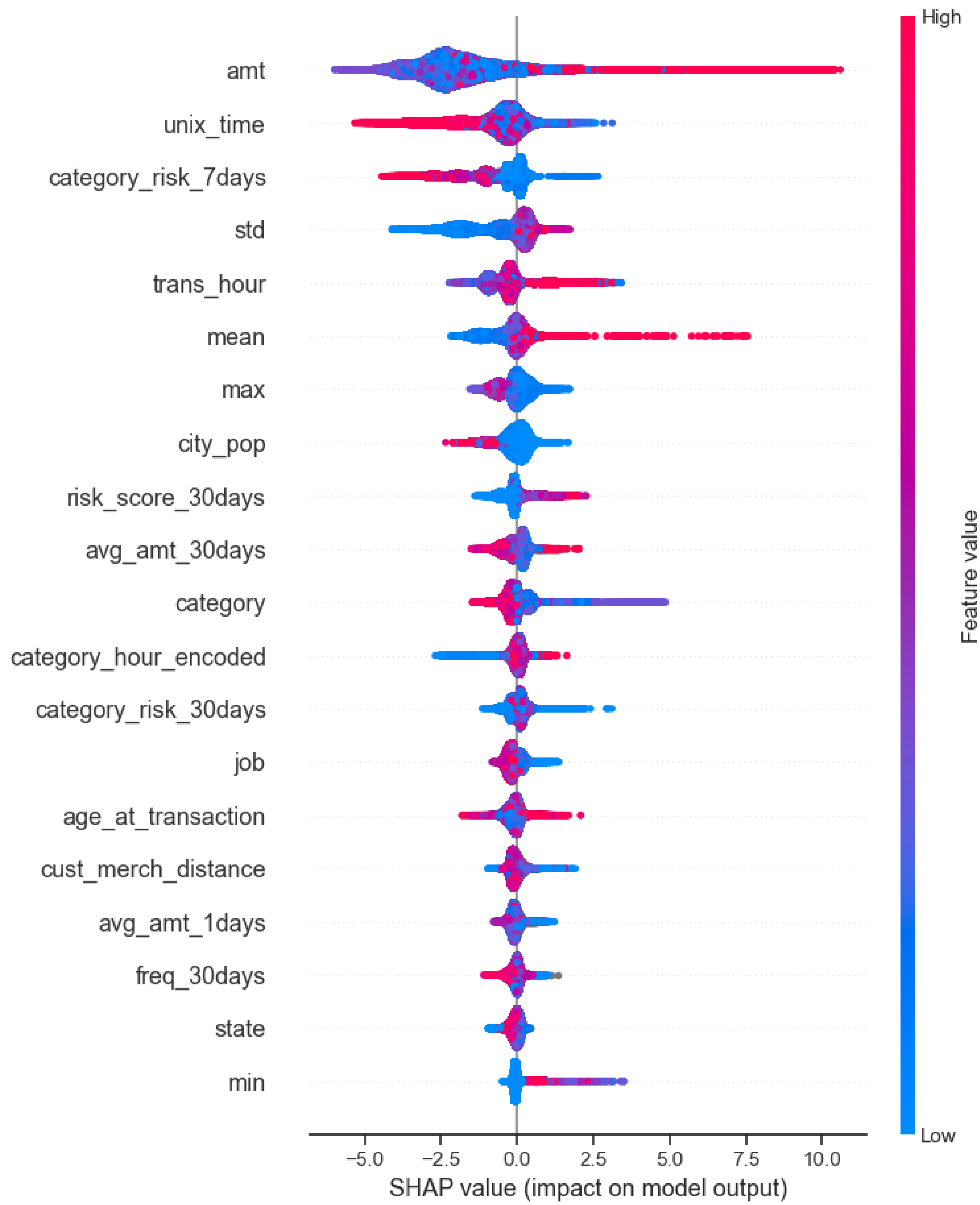


[280]:
```
import shap

# Fit an explainer
explainer = shap.TreeExplainer(xgb_model)
```

```
test_IDs = df['Id'].copy()


X_shap = X_train_processed.copy()

shap_values = explainer.shap_values(X_shap)

print(shap_values.shape)
print(X_train_processed.shape)

shap.summary_plot(shap_values, X_train_processed)
```

```
(437628, 36)
(437628, 36)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be
ignored
```



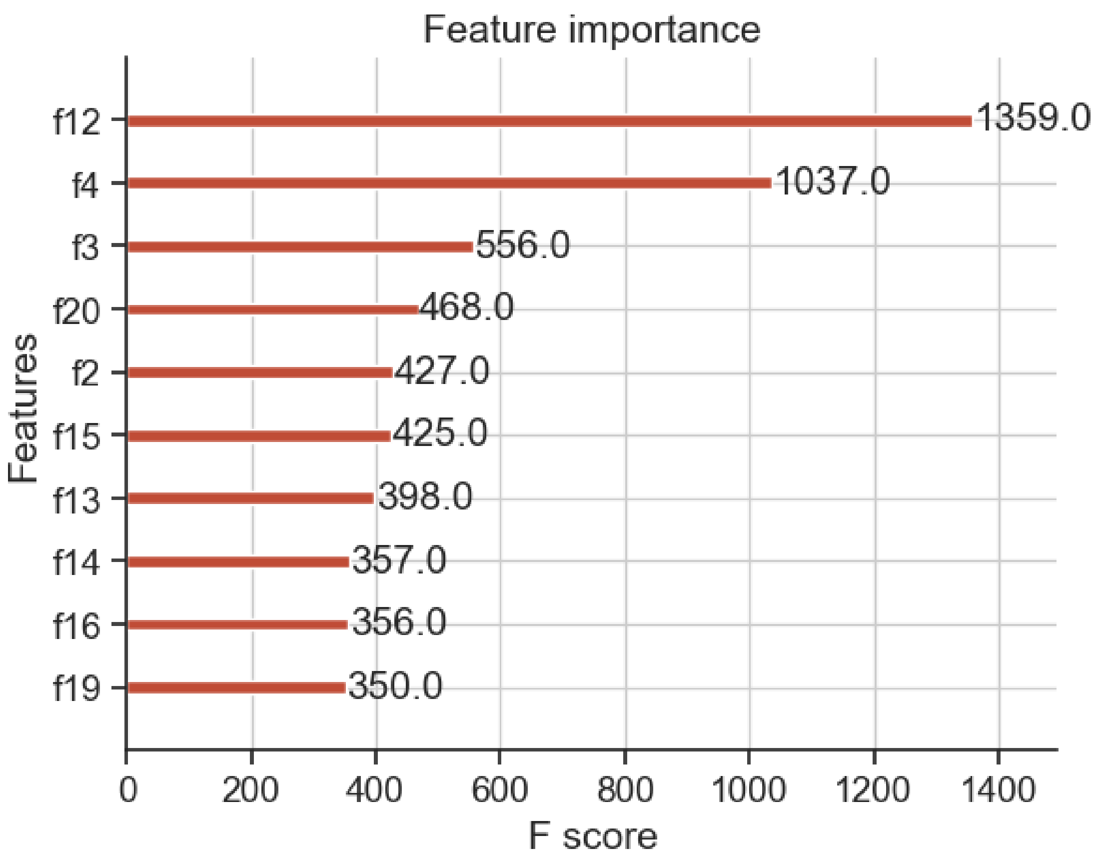
0.4 Create the Kaggle submission

[303]:
```
X_submission = pd.read_csv("./data/X_test.csv")
test_IDs = X_submission['Id']
X_submission = X_submission.drop(columns=['is_fraud', 'Id'])
X_submission_processed = X_submission._get_numeric_data()
```

```
X_submission['is_fraud'] = xgb_model.predict(X_submission_processed)
X_submission.is_fraud = X_submission.is_fraud.astype(int)
X_submission['Id'] = test_IDs
submission = X_submission[['Id', 'is_fraud']]
submission.to_csv("./data/submission.csv", index=False)
```

Now you can upload the `submission.csv` to kaggle

[291]:
```
import xgboost as xgb
import matplotlib.pyplot as plt

# Load the model
# Replace with the correct path or code to load your model
xgb_model = pickle.load(open('XGBoost_best_model.pkl', 'rb'))

# Plot feature importances
xgb.plot_importance(xgb_model, max_num_features=10)  # Adjust to display the␣
 ↪number of top features you want to see
plt.show()

# If your categories are one-hot encoded, their feature importances will be␣
 ↪scattered across multiple features,
# and you might need to sum these importances to get the overall importance for␣
 ↪the category feature.
```


Feature importance
f12
f4
f3
f20
f2
f15
f13
f14
f16
f19
Features
1359.0
1037.0
556.0
468.0
427.0
425.0
398.0
357.0
356.0
350.0
0
200
400
600
800
1000
1200
1400
F score