read me:

Overview

This project classifies online purchase orders into high-risk or low-risk categories using a Random Forest Classifier. The solution involves data preprocessing(cleaning,wrangling,transformation), model training, prediction, and evaluation.

Files

- train_df.csv: Training dataset
- test_df.csv: Test dataset
- Capstone_midterm_project.py: Python script containing the code
- result.txt: Output file with predicted classifications
- report.pdf: Detailed description of the solution

Usage

1. Ensure you have the necessary files in the working directory.
2. Run the classification_code.py script to perform data preprocessing, model training, prediction, and evaluation.
3. The predicted classifications will be saved in predicted_classes.txt.
4. The script will also display confusion matrices and accuracy for validation and sanity check.

Requirements

- pandas
- matplotlib
- seaborn
- seaborn
- scikit-learn

Install the required packages using: bash pip install pandas matplotlib seaborn scikit-learn

In [256]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,recall_score,accuracy_score
import seaborn as sns
```

In [257]:
```
import pandas as pd

# Define the file paths
input_file = 'risk-train.txt'  # Replace with your input file path
output_file = 'output.csv'  # Replace with your desired output file path

# Read the text file
with open(input_file, 'r') as file:
    lines = file.readlines()

# Extract headers
headers = lines[0].strip().split(',')

# Extract data rows
data = []
for line in lines[1:]:
    row = line.strip().split(',')
    data.append(row)

# Create a DataFrame
df = pd.DataFrame(data, columns=headers)

# Handle any potential data type issues, such as converting date strings to datetime
# For example, convert 'B_BIRTHDATE' column to datetime, if necessary:
df['B_BIRTHDATE'] = pd.to_datetime(df['B_BIRTHDATE'], errors='coerce')
df.replace('?', pd.NA, inplace=True)
# Save to CSV
df.to_csv(output_file, index=False)

print(f"Data successfully converted to {output_file}")

```

```
Data successfully converted to output.csv
```

In [258]:
```
data1 = pd.read_csv('output.csv')
data1
```

Out[258]:

	ORDER_ID	CLASS	B_EMAIL	B_TELEFON	B_BIRTHDATE	FLAG_LRIDENTISCH	FLAG_NEWSLETTER	Z_METHODE	Z_CARD_ART	Z_CARD_VALID	...	FAIL_RPLZ	FAIL_RORT	FAIL_RPLZORTMATCH	SESSION_TIME	N
0	49917	no	yes	no	1973-01-17	yes	yes	check	NaN	5.2006	...	no	no	no	8	
1	49919	no	yes	yes	1970-12-08	no	no	credit_card	Visa	12.2007	...	yes	no	no	13	
2	49923	no	yes	no	1972-04-03	yes	no	check	NaN	12.2007	...	no	no	no	3	
3	49924	no	no	yes	1966-08-01	yes	no	check	NaN	1.2007	...	no	no	no	11	
4	49927	no	yes	yes	1969-12-21	yes	no	credit_card	Eurocard	12.2006	...	no	no	no	16	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	49821	no	yes	no	1981-03-03	yes	no	check	NaN	6.2005	...	no	no	no	10	
29996	49824	no	yes	no	1972-02-21	no	no	credit_card	Eurocard	5.2005	...	no	no	no	10	
29997	49825	no	yes	no	1980-06-11	no	no	credit_card	Eurocard	11.2006	...	no	no	no	6	
29998	49828	no	yes	no	1980-12-04	no	no	credit_card	Visa	4.2005	...	no	no	no	6	
29999	49829	no	yes	no	1958-03-15	no	no	credit_card	Eurocard	4.2006	...	no	no	no	11	

30000 rows × 44 columns

In [259]:
```
data1.isna().sum()
```

Out[259]:
```
ORDER_ID                0
CLASS                   0
B_EMAIL                 0
B_TELEFON               0
B_BIRTHDATE          2942
FLAG_LRIDENTISCH        0
FLAG_NEWSLETTER         0
Z_METHODE               0
Z_CARD_ART          18654
Z_CARD_VALID            0
Z_LAST_NAME         14808
VALUE_ORDER             0
WEEKDAY_ORDER           0
TIME_ORDER             20
AMOUNT_ORDER            0
ANUMMER_01              0
ANUMMER_02          22147
ANUMMER_03          26802
ANUMMER_04          28668
ANUMMER_05          29459
ANUMMER_06          29794
ANUMMER_07          29905
ANUMMER_08          29966
ANUMMER_09          29993
ANUMMER_10          30000
CHK_LADR                0
CHK_RADR                0
CHK_KTO                 0
CHK_CARD                0
CHK_COOKIE              0
CHK_IP                  0
FAIL_LPLZ               0
FAIL_LORT               0
FAIL_LPLZORTMATCH       0
FAIL_RPLZ               0
FAIL_RORT               0
FAIL_RPLZORTMATCH       0
SESSION_TIME            0
NEUKUNDE                0
AMOUNT_ORDER_PRE        0
VALUE_ORDER_PRE         0
DATE_LORDER         15856
MAHN_AKT            15856
MAHN_HOECHST        15856
dtype: int64
```

when i analysised the both columns they have the relation between both the columns i came to know that when there is a debitnote and check then there is Question mark ? so i replaced that with Noot applicable

In [260]:
```
data1.loc[data1['Z_METHODE'].isin(['check', 'debit_note']),'Z_CARD_ART']='Not Applicable'
```

In [261]:
```
data1[['Z_METHODE','Z_CARD_ART']]
```

Out[261]:

	Z_METHODE	Z_CARD_ART
0	check	Not Applicable
1	credit_card	Visa
2	check	Not Applicable
3	check	Not Applicable
4	credit_card	Eurocard
...	...	...
29995	check	Not Applicable
29996	credit_card	Eurocard
29997	credit_card	Eurocard
29998	credit_card	Visa
29999	credit_card	Eurocard

30000 rows × 2 columns

In [262]:
```
data1 = data1.dropna(subset=['B_BIRTHDATE'])
```

In [263]:
```
data1['B_BIRTHDATE'].isna().sum()
```

Out[263]:
```
0
```

In [264]:
```
data1['B_BIRTHDATE'] = pd.to_datetime(data1['B_BIRTHDATE'], errors='coerce')


data1['YEAR'] = data1['B_BIRTHDATE'].dt.year
data1['MONTH'] = data1['B_BIRTHDATE'].dt.month
data1['DAY'] = data1['B_BIRTHDATE'].dt.day
```

```
/var/folders/j0/z5v95_qd44z83cv5p8l90k8c0000gn/T/ipykernel_93411/1310581256.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data1['B_BIRTHDATE'] = pd.to_datetime(data1['B_BIRTHDATE'], errors='coerce')
/var/folders/j0/z5v95_qd44z83cv5p8l90k8c0000gn/T/ipykernel_93411/1310581256.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data1['YEAR'] = data1['B_BIRTHDATE'].dt.year
/var/folders/j0/z5v95_qd44z83cv5p8l90k8c0000gn/T/ipykernel_93411/1310581256.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data1['MONTH'] = data1['B_BIRTHDATE'].dt.month
/var/folders/j0/z5v95_qd44z83cv5p8l90k8c0000gn/T/ipykernel_93411/1310581256.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data1['DAY'] = data1['B_BIRTHDATE'].dt.day
```

In [265]:
```
data1[['ANUMMER_01','ANUMMER_02',1ANUMMER_03','ANUMMER_04','ANUMMER_05','ANUMMER_06','ANUMMER_07','ANUMMER_08','ANUMMER_09','ANUMMER_10']].head(15)
```

Out[265]:

	ANUMMER_01	ANUMMER_02	ANUMMER_03	ANUMMER_04	ANUMMER_05	ANUMMER_06	ANUMMER_07	ANUMMER_08	ANUMMER_09	ANUMMER_10
0	406811	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	600953	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	406310	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	307359	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	200767	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	405897	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	303950	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	400124	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	508801	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	400914	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	106590	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	201402	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	207610	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	201373	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	407610	102089.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

there are number of NAN Values in the columns from ANUMMER_02-10 so i am planning to drop this columns

In [266]:
```
data1.drop(columns={'ANUMMER_02','ANUMMER_03','ANUMMER_04','ANUMMER_05','ANUMMER_06','ANUMMER_07','ANUMMER_08','ANUMMER_09','ANUMMER_10'},inplace=True)
```

```
/var/folders/j0/z5v95_qd44z83cv5p8l90k8c0000gn/T/ipykernel_93411/2056935221.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data1.drop(columns={'ANUMMER_02','ANUMMER_03','ANUMMER_04','ANUMMER_05','ANUMMER_06','ANUMMER_07','ANUMMER_08','ANUMMER_09','ANUMMER_10'},inplace=True)
```

In [267]:
```
data1.isna().sum()
```

Out[267]:
```
ORDER_ID                0
CLASS                   0
B_EMAIL                 0
B_TELEFON               0
B_BIRTHDATE             0
FLAG_LRIDENTISCH        0
FLAG_NEWSLETTER         0
Z_METHODE               0
Z_CARD_ART              0
Z_CARD_VALID            0
Z_LAST_NAME         13387
VALUE_ORDER             0
WEEKDAY_ORDER           0
TIME_ORDER             17
AMOUNT_ORDER            0
ANUMMER_01              0
CHK_LADR                0
CHK_RADR                0
CHK_KTO                 0
CHK_CARD                0
CHK_COOKIE              0
CHK_IP                  0
FAIL_LPLZ               0
FAIL_LORT               0
FAIL_LPLZORTMATCH       0
FAIL_RPLZ               0
FAIL_RORT               0
FAIL_RPLZORTMATCH       0
SESSION_TIME            0
NEUKUNDE                0
AMOUNT_ORDER_PRE        0
VALUE_ORDER_PRE         0
DATE_LORDER         14290
MAHN_AKT            14290
MAHN_HOECHST        14290
YEAR                    0
MONTH                   0
DAY                     0
dtype: int64
```

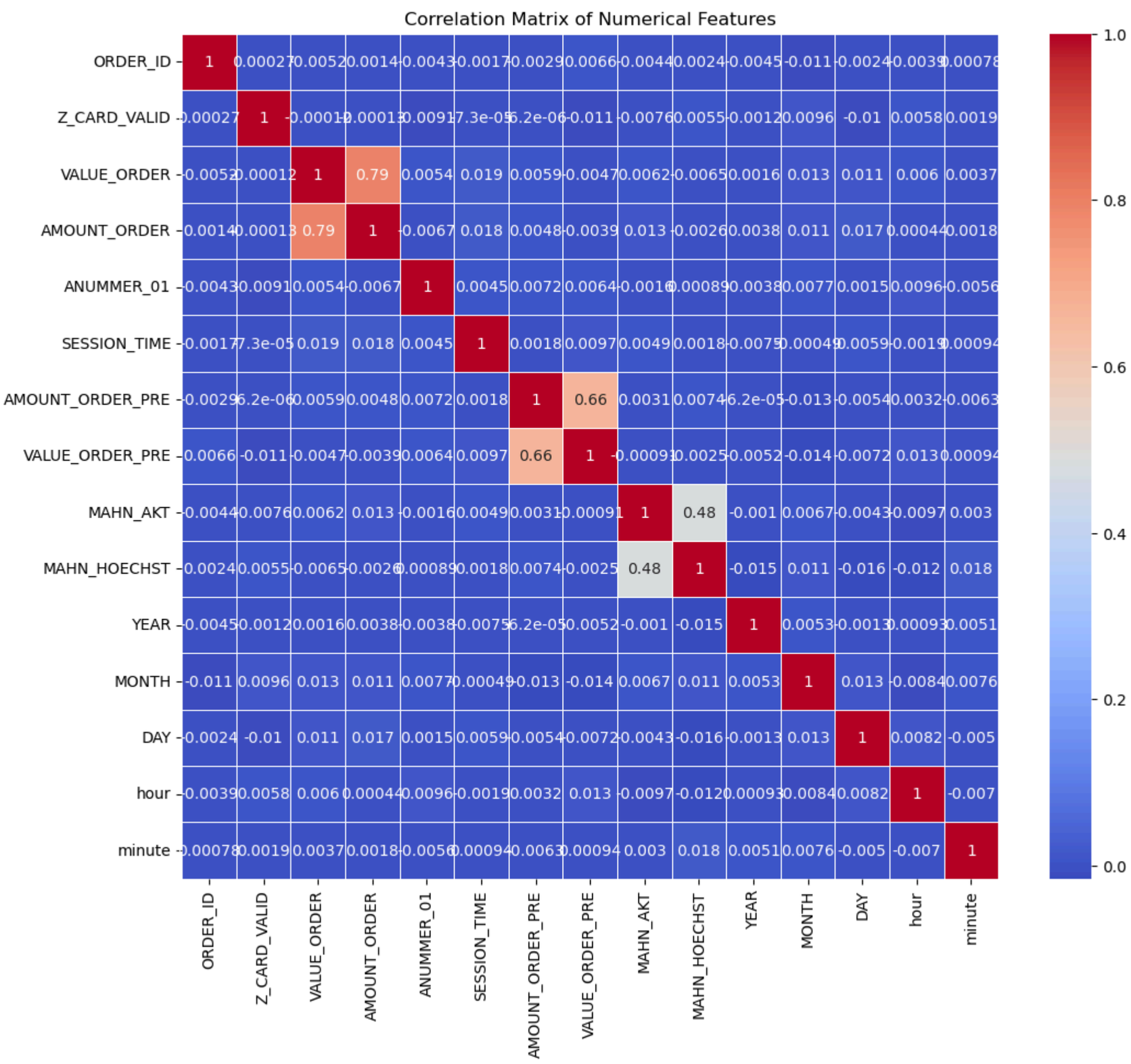
In [268]:
```
data1.loc[data1['NEUKUNDE'] == 'yes', 'DATE_LORDER'] = None

```

In [269]:

```
import pandas as pd

# Check for duplicate columns
duplicate_columns = data1.columns.duplicated()
print("Duplicate columns:", data1.columns[duplicate_columns])

# Drop duplicate columns (if any)
data1 = data1.loc[:, ~data1.columns.duplicated()]

# Rename 'TIME_ORDER' to 'time_order' for consistency
data1.rename(columns={'TIME_ORDER': 'time_order'}, inplace=True)

# Convert time_order to datetime
data1['time_order'] = pd.to_datetime(data1['time_order'], format='%H:%M', errors='coerce')

# Extract hour and minute components from the datetime formatted 'time_order'
data1['hour'] = data1['time_order'].dt.hour
data1['minute'] = data1['time_order'].dt.minute

# Define function to categorize period of day
def get_period_of_day(hour):
    if 6 <= hour < 12:
        return 'morning'
    elif 12 <= hour < 18:
        return 'afternoon'
    elif 18 <= hour < 24:
        return 'evening'
    else:
        return 'night'

# Apply function to categorize period of day
data1['period_of_day'] = data1['hour'].apply(get_period_of_day)

print(data1)

```

```
Duplicate columns: Index([], dtype='object')
       ORDER_ID CLASS B_EMAIL B_TELEFON B_BIRTHDATE FLAG_LRIDENTISCH  \
0         49917    no     yes         no  1973-01-17              yes
1         49919    no     yes        yes  1970-12-08               no
2         49923    no     yes         no  1972-04-03              yes
3         49924    no      no        yes  1966-08-01              yes
4         49927    no     yes        yes  1969-12-21              yes
...         ...   ...     ...        ...         ...              ...
29995     49821    no     yes         no  1981-03-03              yes
29996     49824    no     yes         no  1972-02-21               no
29997     49825    no     yes         no  1980-06-11               no
29998     49828    no     yes         no  1980-12-04               no
29999     49829    no     yes         no  1958-03-15               no

      FLAG_NEWSLETTER    Z_METHODE      Z_CARD_ART  Z_CARD_VALID  ...  \
0                 yes        check  Not Applicable        5.2006  ...
1                  no  credit_card            Visa       12.2007  ...
2                  no        check  Not Applicable       12.2007  ...
3                  no        check  Not Applicable        1.2007  ...
4                  no  credit_card        Eurocard       12.2006  ...
...               ...          ...             ...           ...  ...
29995              no        check  Not Applicable        6.2005  ...
29996              no  credit_card        Eurocard        5.2005  ...
29997              no  credit_card        Eurocard       11.2006  ...
29998              no  credit_card            Visa        4.2005  ...
29999              no  credit_card        Eurocard        4.2006  ...

     VALUE_ORDER_PRE  DATE_LORDER MAHN_AKT MAHN_HOECHST  YEAR  MONTH DAY  \
0               0.00         None      NaN          NaN  1973      1  17
1               0.00         None      NaN          NaN  1970     12   8
2               0.00         None      NaN          NaN  1972      4   3
3              75.72    5/12/2002      0.0          0.0  1966      8   1
4               0.00         None      NaN          NaN  1969     12  21
...              ...          ...      ...          ...   ...    ...  ..
29995          40.83     3/9/2002      0.0          3.0  1981      3   3
29996          52.84     6/3/2001      0.0          0.0  1972      2  21
29997           0.00         None      NaN          NaN  1980      6  11
29998          42.92     7/1/2001      0.0          0.0  1980     12   4
29999          27.29    1/30/2001      0.0          0.0  1958      3  15

       hour minute period_of_day
0       9.0   13.0       morning
1      17.0   36.0     afternoon
2      11.0   13.0       morning
3       2.0    7.0         night
4      23.0   46.0       evening
...     ...    ...           ...
29995  18.0   21.0       evening
29996  15.0   51.0     afternoon
29997  10.0   37.0       morning
29998   1.0   52.0         night
29999  20.0   34.0       evening

[27058 rows x 41 columns]
```

In [270]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data1' is your DataFrame containing the training data
numerical_cols = data1.select_dtypes(include=[np.number]).columns

# Calculate the correlation matrix
corr_matrix = data1[numerical_cols].corr()

# Plot a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Features')
plt.show()

# Identify features with high correlation (e.g., correlation > 0.9)
high_corr_pairs = corr_matrix.abs().unstack().sort_values(ascending=False).drop_duplicates()
high_corr_pairs = high_corr_pairs[high_corr_pairs > 0.9]

print("Highly Correlated Feature Pairs:\n", high_corr_pairs)

```




```
Highly Correlated Feature Pairs:
 ORDER_ID  ORDER_ID    1.0
dtype: float64
```

In [271]:
```
data1.drop(columns={'CHK_COOKIE','CHK_IP','FAIL_LPLZ','FAIL_RPLZ','FAIL_RORT','FAIL_RPLZORTMATCH'},inplace=True)
```

In [272]:
```
data1.shape
```

Out[272]: (27058, 35)

In [273]:
```
mean_value = data1['MAHN_AKT'].mean()
data1['MAHN_AKT'].fillna(mean_value, inplace=True)
```

In [274]:
```
mean_value = data1['MAHN_HOECHST'].mean()
data1['MAHN_HOECHST'].fillna(mean_value, inplace=True)
```

In [275]:
```
data1.loc[data1['NEUKUNDE'] == 'yes', 'DATE_LORDER'] = 'Not Applicable'

```

In [276]:
```
data1.drop(columns={'Z_LAST_NAME'},inplace=True)
```

In [277]:
```
data1.isna().sum()
```

Out[277]:
```
ORDER_ID               0
CLASS                  0
B_EMAIL                0
B_TELEFON              0
B_BIRTHDATE            0
FLAG_LRIDENTISCH       0
FLAG_NEWSLETTER        0
Z_METHODE              0
Z_CARD_ART             0
Z_CARD_VALID           0
VALUE_ORDER            0
WEEKDAY_ORDER          0
time_order            17
AMOUNT_ORDER           0
ANUMMER_01             0
CHK_LADR               0
CHK_RADR               0
CHK_KTO                0
CHK_CARD               0
FAIL_LORT              0
FAIL_LPLZORTMATCH      0
SESSION_TIME           0
NEUKUNDE               0
AMOUNT_ORDER_PRE       0
VALUE_ORDER_PRE        0
DATE_LORDER          725
MAHN_AKT               0
MAHN_HOECHST           0
YEAR                   0
MONTH                  0
DAY                    0
hour                  17
minute                17
period_of_day          0
dtype: int64
```

In [278]:
```
data1.dropna(axis=1, inplace=True)
```

In [279]:
```
train_df = data1
```

In [280]:
```
categorical_cols = train_df.select_dtypes(include=['object']).columns
numerical_cols = train_df.select_dtypes(include=[np.number]).columns
print(categorical_cols)
print(numerical_cols)
```

```
Index(['CLASS', 'B_EMAIL', 'B_TELEFON', 'FLAG_LRIDENTISCH', 'FLAG_NEWSLETTER',
       'Z_METHODE', 'Z_CARD_ART', 'WEEKDAY_ORDER', 'CHK_LADR', 'CHK_RADR',
       'CHK_KTO', 'CHK_CARD', 'FAIL_LORT', 'FAIL_LPLZORTMATCH', 'NEUKUNDE',
       'period_of_day'],
      dtype='object')
Index(['ORDER_ID', 'Z_CARD_VALID', 'VALUE_ORDER', 'AMOUNT_ORDER', 'ANUMMER_01',
       'SESSION_TIME', 'AMOUNT_ORDER_PRE', 'VALUE_ORDER_PRE', 'MAHN_AKT',
       'MAHN_HOECHST', 'YEAR', 'MONTH', 'DAY'],
      dtype='object')
```

In [281]:
```
# 1. Convert object columns to categories
categorical_cols = ['CLASS', 'B_EMAIL', 'B_TELEFON', 'FLAG_LRIDENTISCH', 'FLAG_NEWSLETTER', 'Z_METHODE',
                    'Z_CARD_ART', 'WEEKDAY_ORDER', 'CHK_LADR', 'CHK_RADR', 'CHK_KTO', 'CHK_CARD',
                    'FAIL_LORT', 'FAIL_LPLZORTMATCH', 'NEUKUNDE', 'period_of_day']

for col in categorical_cols:
    data1[col] = data1[col].astype('category')
```

In [282]:
```
data1[col]
```

Out[282]:
```
0         morning
1       afternoon
2         morning
3           night
4         evening
           ...
29995     evening
29996   afternoon
29997     morning
29998       night
29999     evening
Name: period_of_day, Length: 27058, dtype: category
Categories (4, object): ['afternoon', 'evening', 'morning', 'night']
```

In [283]:
```
data1['AGE'] = data1['B_BIRTHDATE'].apply(lambda x: pd.Timestamp('now').year - pd.to_datetime(x).year)
data1 = data1.drop(['B_BIRTHDATE', 'YEAR', 'MONTH', 'DAY'], axis=1)
```

In [284]:
```
numeric_cols = data1.select_dtypes(include=['float64', 'int64']).columns
for col in numeric_cols:
    data1[col].fillna(data1[col].median(), inplace=True)
```

In [285]:
```
numeric_cols
```

Out[285]:
```
Index(['ORDER_ID', 'Z_CARD_VALID', 'VALUE_ORDER', 'AMOUNT_ORDER', 'ANUMMER_01',
       'SESSION_TIME', 'AMOUNT_ORDER_PRE', 'VALUE_ORDER_PRE', 'MAHN_AKT',
       'MAHN_HOECHST', 'AGE'],
      dtype='object')
```

In [286]:
```
data1 = pd.get_dummies(data1, drop_first=True)

train_df =  pd.get_dummies(data1, drop_first=True)
```

In [287]:
```
data1 =  pd.get_dummies(train_df, drop_first=True)
```

In [288]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

continuous_cols = ['VALUE_ORDER', 'SESSION_TIME', 'AMOUNT_ORDER', 'AMOUNT_ORDER_PRE', 'VALUE_ORDER_PRE', 'MAHN_AKT', 'MAHN_HOECHST', 'AGE']

```

In [289]:
```
data1.dtypes
```

Out[289]:
```
ORDER_ID                      int64
Z_CARD_VALID                float64
VALUE_ORDER                 float64
AMOUNT_ORDER                  int64
ANUMMER_01                    int64
SESSION_TIME                  int64
AMOUNT_ORDER_PRE              int64
VALUE_ORDER_PRE             float64
MAHN_AKT                    float64
MAHN_HOECHST                float64
AGE                           int64
CLASS_yes                     uint8
B_EMAIL_yes                   uint8
B_TELEFON_yes                 uint8
FLAG_LRIDENTISCH_yes          uint8
FLAG_NEWSLETTER_yes           uint8
Z_METHODE_credit_card         uint8
Z_METHODE_debit_card          uint8
Z_METHODE_debit_note          uint8
Z_CARD_ART_Eurocard           uint8
Z_CARD_ART_Not Applicable     uint8
Z_CARD_ART_Visa               uint8
Z_CARD_ART_debit_card         uint8
WEEKDAY_ORDER_Monday          uint8
WEEKDAY_ORDER_Saturday        uint8
WEEKDAY_ORDER_Sunday          uint8
WEEKDAY_ORDER_Thursday        uint8
WEEKDAY_ORDER_Tuesday         uint8
WEEKDAY_ORDER_Wednesday       uint8
CHK_LADR_yes                  uint8
CHK_RADR_yes                  uint8
CHK_KTO_yes                   uint8
CHK_CARD_yes                  uint8
FAIL_LORT_yes                 uint8
FAIL_LPLZORTMATCH_yes         uint8
NEUKUNDE_yes                  uint8
period_of_day_evening         uint8
period_of_day_morning         uint8
period_of_day_night           uint8
dtype: object
```

In [290]:
```
data1[continuous_cols] = scaler.fit_transform(data1[continuous_cols])
```

In [291]:
```
train_df = data1
train_df.to_csv('cleaned_train.csv', index=False)
```

In [292]:
```
# Assuming X contains features and y contains the target variable 'CLASS_yes'
X = data1.drop('CLASS_yes', axis=1)  # Features
y = data1['CLASS_yes']  # Target
```

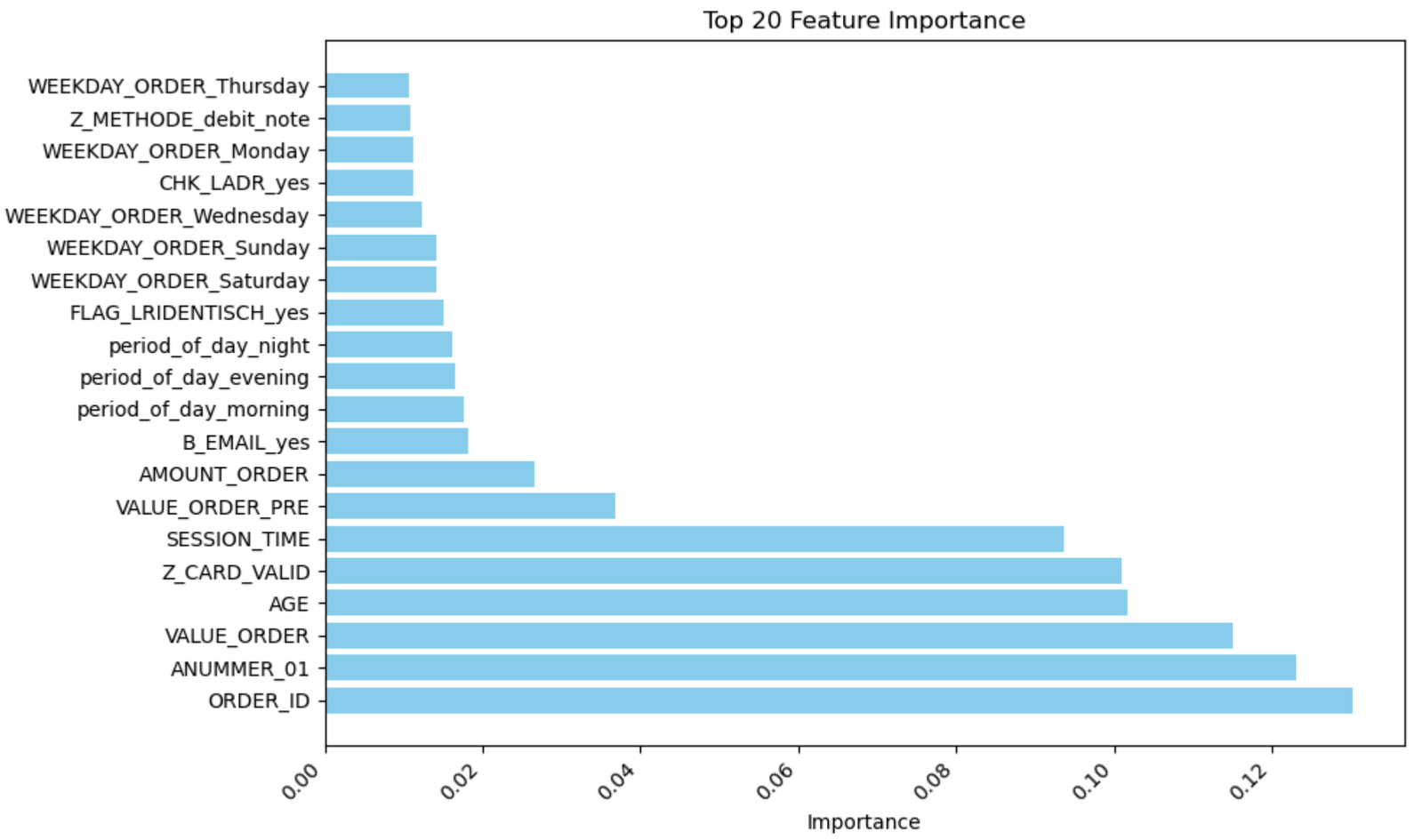
In [293]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize RandomForestClassifier (you can choose another classifier as needed)
clf = RandomForestClassifier(random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print classification report for detailed metrics
print(classification_report(y_test, y_pred))

```

```
Accuracy: 0.95
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      5123
           1       0.50      0.00      0.01       289

    accuracy                           0.95      5412
   macro avg       0.72      0.50      0.49      5412
weighted avg       0.92      0.95      0.92      5412
```

In [294]:
```
print(confusion_matrix(y_test, y_pred))
```

```
[[5122    1]
 [ 288    1]]
```

In [295]:
```
feature_importances = clf.feature_importances_

# Create a DataFrame to visualize feature importance
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

top_n = 20
top_features = feature_importance_df.head(top_n)

# Plotting feature importance
plt.figure(figsize=(10, 6))
plt.barh(top_features['Feature'], top_features['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.title('Feature Importance')
plt.title('Top 20 Feature Importance')
plt.xticks(rotation=45, ha='right')

plt.tight_layout()  # Adjust layout to make room for rotated labels
plt.show()
```



In [296]:
```
import pandas as pd

# Specify the file paths
input_file = 'risk-test.txt'   # Replace with the path to your input TSV file
output_file = 'orders.csv'  # Replace with the desired path for your output CSV file

# Read the TSV file
df = pd.read_csv(input_file, delimiter='\t')

# Handle missing values (optional)
# For example, replace "?" with NaN
df.replace('?', pd.NA, inplace=True)

# Save the DataFrame to a CSV file
df.to_csv(output_file, index=False)

print(f"File converted successfully from {input_file} to {output_file}")
```

```
File converted successfully from risk-test.txt to orders.csv
```

In [297]:
```
df.shape
```

Out[297]:
```
(20000, 43)
```

In [298]:
```
# Replace 'Z_CARD_ART' with 'Not Applicable' where 'Z_METHODE' is 'check' or 'debit_note'
df.loc[df['Z_METHODE'].isin(['check', 'debit_note']), 'Z_CARD_ART'] = 'Not Applicable'

```

In [299]:
```
df[['Z_METHODE','Z_CARD_ART']]
```

Out[299]:

	Z_METHODE	Z_CARD_ART
0	credit_card	Visa
1	credit_card	Eurocard
2	check	Not Applicable
3	credit_card	Eurocard
4	debit_note	Not Applicable
...	...	...
19995	check	Not Applicable
19996	credit_card	Eurocard
19997	credit_card	Visa
19998	debit_card	debit_card
19999	debit_note	Not Applicable

20000 rows × 2 columns

In [300]:
```
df = df.dropna(subset=['B_BIRTHDATE'])
```

In [301]:
```
df['B_BIRTHDATE'].isna().sum()
```

Out[301]:
```
0
```

In [302]:
```
# Convert 'B_BIRTHDATE' to datetime with errors='coerce'
df['B_BIRTHDATE'] = pd.to_datetime(df['B_BIRTHDATE'], errors='coerce')

# Extract year, month, and day from 'B_BIRTHDATE'
df['YEAR'] = df['B_BIRTHDATE'].dt.year
df['MONTH'] = df['B_BIRTHDATE'].dt.month
df['DAY'] = df['B_BIRTHDATE'].dt.day

```

In [303]:
```
df.drop(columns={'ANUMMER_02','ANUMMER_03','ANUMMER_04','ANUMMER_05','ANUMMER_06','ANUMMER_07','ANUMMER_08','ANUMMER_09','ANUMMER_10'},inplace=True)
```

In [304]:
```
# Set 'DATE_LORDER' to None where 'NEUKUNDE' is 'yes'
df.loc[df['NEUKUNDE'] == 'yes', 'DATE_LORDER'] = None

```

In [305]:
```
import pandas as pd

# Check for duplicate columns
duplicate_columns = df.columns.duplicated()
print("Duplicate columns:", df.columns[duplicate_columns])

# Drop duplicate columns (if any)
df = df.loc[:, ~df.columns.duplicated()]

# Rename 'TIME_ORDER' to 'time_order' for consistency
df.rename(columns={'TIME_ORDER': 'time_order'}, inplace=True)

# Convert time_order to datetime
df['time_order'] = pd.to_datetime(df['time_order'], format='%H:%M', errors='coerce')

# Extract hour and minute components from the datetime formatted 'time_order'
df['hour'] = df['time_order'].dt.hour
df['minute'] = df['time_order'].dt.minute

# Define function to categorize period of day
def get_period_of_day(hour):
    if 6 <= hour < 12:
        return 'morning'
    elif 12 <= hour < 18:
        return 'afternoon'
    elif 18 <= hour < 24:
        return 'evening'
    else:
        return 'night'

# Apply function to categorize period of day
df['period_of_day'] = df['hour'].apply(get_period_of_day)

print(df)

```

```
Duplicate columns: Index([], dtype='object')
       ORDER_ID B_EMAIL B_TELEFON B_BIRTHDATE FLAG_LRIDENTISCH  \
0         49916     yes        no  1979-07-22               no
1         49918      no        no  1973-02-05               no
2         49920     yes        no  1970-07-19              yes
3         49921     yes        no  1985-04-09              yes
4         49922      no       yes  1963-04-07               no
...         ...     ...       ...         ...              ...
19995     49820     yes        no  1965-04-14              yes
19996     49822     yes        no  1975-08-08               no
19997     49823     yes        no  1949-05-11               no
19998     49826     yes        no  1976-04-14               no
19999     49827     yes        no  1976-10-21               no

      FLAG_NEWSLETTER    Z_METHODE      Z_CARD_ART  Z_CARD_VALID Z_LAST_NAME  \
0                  no  credit_card            Visa        2.2005         yes
1                  no  credit_card       Eurocard        4.2005         yes
2                  no        check  Not Applicable        8.2005        <NA>
3                  no  credit_card       Eurocard        7.2006         yes
4                  no   debit_note  Not Applicable        5.2007         yes
...               ...         ...             ...           ...         ...
19995              no        check  Not Applicable        7.2005        <NA>
19996              no  credit_card       Eurocard        3.2007         yes
19997              no  credit_card           Visa       10.2007         yes
19998              no   debit_card     debit_card        5.2005         yes
19999              no   debit_note  Not Applicable       12.2007         yes

       ...  VALUE_ORDER_PRE DATE_LORDER MAHN_AKT  MAHN_HOECHST  YEAR MONTH  \
0      ...             0.00        None     <NA>          <NA>  1979     7
1      ...            30.31    9/2/2004        0             0  1973     2
2      ...             0.00        None     <NA>          <NA>  1970     7
3      ...            54.07  12/24/2003        0             1  1985     4
4      ...             0.00        None     <NA>          <NA>  1963     4
...    ...              ...         ...      ...           ...   ...   ...
19995  ...            80.99   2/10/2003        0             0  1965     4
19996  ...             0.00        None     <NA>          <NA>  1975     8
19997  ...             0.00        None     <NA>          <NA>  1949     5
19998  ...             0.00        <NA>     <NA>          <NA>  1976     4
19999  ...           104.93   12/3/2001        0             2  1976    10

      DAY  hour minute period_of_day
0      22   6.0   42.0       morning
1       5  14.0   44.0     afternoon
2      19  18.0   40.0       evening
3       9  14.0   22.0     afternoon
4       7   1.0   54.0         night
...    ..   ...    ...           ...
19995  14  14.0   12.0     afternoon
19996   8   5.0   29.0         night
19997  11  23.0   55.0       evening
19998  14  13.0    8.0     afternoon
19999  21   6.0   49.0       morning

[17946 rows x 40 columns]
```

In [306]:
```
df
```

Out[306]:

	ORDER_ID	B_EMAIL	B_TELEFON	B_BIRTHDATE	FLAG_LRIDENTISCH	FLAG_NEWSLETTER	Z_METHODE	Z_CARD_ART	Z_CARD_VALID	Z_LAST_NAME	...	VALUE_ORDER_PRE	DATE_LORDER	MAHN_AKT	MAHN_I
0	49916	yes	no	1979-07-22	no	no	credit_card	Visa	2.2005	yes	...	0.00	None	<NA>	
1	49918	no	no	1973-02-05	no	no	credit_card	Eurocard	4.2005	yes	...	30.31	9/2/2004	0	
2	49920	yes	no	1970-07-19	yes	no	check	Not Applicable	8.2005	<NA>	...	0.00	None	<NA>	
3	49921	yes	no	1985-04-09	yes	no	credit_card	Eurocard	7.2006	yes	...	54.07	12/24/2003	0	
4	49922	no	yes	1963-04-07	no	no	debit_note	Not Applicable	5.2007	yes	...	0.00	None	<NA>	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
19995	49820	yes	no	1965-04-14	yes	no	check	Not Applicable	7.2005	<NA>	...	80.99	2/10/2003	0	
19996	49822	yes	no	1975-08-08	no	no	credit_card	Eurocard	3.2007	yes	...	0.00	None	<NA>	
19997	49823	yes	no	1949-05-11	no	no	credit_card	Visa	10.2007	yes	...	0.00	None	<NA>	
19998	49826	yes	no	1976-04-14	no	no	debit_card	debit_card	5.2005	yes	...	0.00	<NA>	<NA>	
19999	49827	yes	no	1976-10-21	no	no	debit_note	Not Applicable	12.2007	yes	...	104.93	12/3/2001	0	

17946 rows × 40 columns

In [307]:
```
df.drop(columns={'CHK_COOKIE','CHK_IP','FAIL_LPLZ','FAIL_RPLZ','FAIL_RORT','FAIL_RPLZORTMATCH'},inplace=True)
```

In [308]:
```
df.shape
```

Out[308]: (17946, 34)

In [309]:
```

unique_values = df['MAHN_AKT'].unique()
print(unique_values)


df['MAHN_AKT'] = pd.to_numeric(df['MAHN_AKT'], errors='coerce')


mean_value = df['MAHN_AKT'].mean()


df['MAHN_AKT'].fillna(mean_value, inplace=True)


```

```
[<NA> '0' '1' '2' '3']
```

In [310]:
```
# Check unique values in 'MAHN_AKT' column
unique_values = df['MAHN_HOECHST'].unique()
print(unique_values)

# Example of how to handle the mean calculation
# Replace non-numeric values with NaN
df['MAHN_HOECHST'] = pd.to_numeric(df['MAHN_HOECHST'], errors='coerce')

# Calculate mean after converting to numeric
mean_value = df['MAHN_HOECHST'].mean()

# Fill NaN values with mean_value
df['MAHN_HOECHST'].fillna(mean_value, inplace=True)

# Proceed with further preprocessing or analysis

```

```
[<NA> '0' '1' '2' '3']
```

In [311]:
```
# Set 'DATE_LORDER' to 'Not Applicable' where 'NEUKUNDE' is 'yes'
df.loc[df['NEUKUNDE'] == 'yes', 'DATE_LORDER'] = 'Not Applicable'

```

In [312]:
```
df.drop(columns={'Z_LAST_NAME'},inplace=True)
```

In [313]:
```
df.dropna(axis=1, inplace=True)
```

In [314]:
```
df.isna().sum()
```

Out[314]:
```
ORDER_ID             0
B_EMAIL              0
B_TELEFON            0
B_BIRTHDATE          0
FLAG_LRIDENTISCH     0
FLAG_NEWSLETTER      0
Z_METHODE            0
Z_CARD_ART           0
Z_CARD_VALID         0
VALUE_ORDER          0
WEEKDAY_ORDER        0
AMOUNT_ORDER         0
ANUMMER_01           0
CHK_LADR             0
CHK_RADR             0
CHK_KTO              0
CHK_CARD             0
FAIL_LORT            0
FAIL_LPLZORTMATCH    0
SESSION_TIME         0
NEUKUNDE             0
AMOUNT_ORDER_PRE     0
VALUE_ORDER_PRE      0
MAHN_AKT             0
MAHN_HOECHST         0
YEAR                 0
MONTH                0
DAY                  0
period_of_day        0
dtype: int64
```

In [315]:
```
test_df = df
```

In [316]:
```
test_df.shape
```

Out[316]: (17946, 29)

In [317]:
```
categorical_cols = test_df.select_dtypes(include=['object']).columns
numerical_cols = test_df.select_dtypes(include=[np.number]).columns
print(categorical_cols)
print(numerical_cols)
```

```
Index(['B_EMAIL', 'B_TELEFON', 'FLAG_LRIDENTISCH', 'FLAG_NEWSLETTER',
       'Z_METHODE', 'Z_CARD_ART', 'WEEKDAY_ORDER', 'CHK_LADR', 'CHK_RADR',
       'CHK_KTO', 'CHK_CARD', 'FAIL_LORT', 'FAIL_LPLZORTMATCH', 'NEUKUNDE',
       'period_of_day'],
      dtype='object')
Index(['ORDER_ID', 'Z_CARD_VALID', 'VALUE_ORDER', 'AMOUNT_ORDER', 'ANUMMER_01',
       'SESSION_TIME', 'AMOUNT_ORDER_PRE', 'VALUE_ORDER_PRE', 'MAHN_AKT',
       'MAHN_HOECHST', 'YEAR', 'MONTH', 'DAY'],
      dtype='object')
```

In [318]:
```
# 1. Convert object columns to categories
categorical_cols = ['B_EMAIL','B_TELEFON', 'FLAG_LRIDENTISCH', 'FLAG_NEWSLETTER', 'Z_METHODE',
                    'Z_CARD_ART', 'WEEKDAY_ORDER', 'CHK_LADR', 'CHK_RADR', 'CHK_KTO', 'CHK_CARD',
                    'FAIL_LORT', 'FAIL_LPLZORTMATCH', 'NEUKUNDE', 'period_of_day']

for col in categorical_cols:
    df[col] = df[col].astype('category')
```

In [319]:
```
df[col]
```

Out[319]:
```
0         morning
1       afternoon
2         evening
3       afternoon
4           night
          ...
19995   afternoon
19996       night
19997     evening
19998   afternoon
19999     morning
Name: period_of_day, Length: 17946, dtype: category
Categories (4, object): ['afternoon', 'evening', 'morning', 'night']
```

In [320]:
```
# Calculate age based on 'B_BIRTHDATE'
df['AGE'] = df['B_BIRTHDATE'].apply(lambda x: pd.Timestamp('now').year - pd.to_datetime(x).year)

# Drop 'B_BIRTHDATE', 'YEAR', 'MONTH', 'DAY' columns
df = df.drop(['B_BIRTHDATE', 'YEAR', 'MONTH', 'DAY'], axis=1)

```

In [321]:
```
import pandas as pd

# Assuming df is your DataFrame
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Fill missing values with median for numeric columns
for col in numeric_cols:
    df[col].fillna(df[col].median(), inplace=True)

```

In [322]:
```
numeric_cols
```

Out[322]:
```
Index(['ORDER_ID', 'Z_CARD_VALID', 'VALUE_ORDER', 'AMOUNT_ORDER', 'ANUMMER_01',
       'SESSION_TIME', 'AMOUNT_ORDER_PRE', 'VALUE_ORDER_PRE', 'MAHN_AKT',
       'MAHN_HOECHST', 'AGE'],
      dtype='object')
```

In [323]:
```
df = pd.get_dummies(df, drop_first=True)
```

In [324]:
```
df
```

Out[324]:

	ORDER_ID	Z_CARD_VALID	VALUE_ORDER	AMOUNT_ORDER	ANUMMER_01	SESSION_TIME	AMOUNT_ORDER_PRE	VALUE_ORDER_PRE	MAHN_AKT	MAHN_HOECHST	...	CHK_LADR_yes	CHK_RADR_yes	CHK_KT
0	49916	2.2005	64.5	1	608801	6	0	0.00	0.123892	0.429838	...	0	0	
1	49918	4.2005	74.3	4	400124	9	1	30.31	0.000000	0.000000	...	0	0	
2	49920	8.2005	42.8	1	406284	5	0	0.00	0.123892	0.429838	...	0	0	
3	49921	7.2006	42.8	1	403772	13	1	54.07	0.000000	1.000000	...	0	0	
4	49922	5.2007	6.5	1	202410	8	0	0.00	0.123892	0.429838	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
19995	49820	7.2005	32.8	1	200329	17	1	80.99	0.000000	0.000000	...	0	0	
19996	49822	3.2007	34.5	1	300925	9	0	0.00	0.123892	0.429838	...	0	0	
19997	49823	10.2007	9.8	1	405378	9	0	0.00	0.123892	0.429838	...	0	0	
19998	49826	5.2005	130.0	3	404921	12	0	0.00	0.123892	0.429838	...	0	0	
19999	49827	12.2007	5.2	1	202062	10	1	104.93	0.000000	2.000000	...	0	0	

17946 rows × 38 columns

In [325]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

continuous_cols = ['VALUE_ORDER', 'SESSION_TIME', 'AMOUNT_ORDER', 'AMOUNT_ORDER_PRE', 'VALUE_ORDER_PRE', 'MAHN_AKT', 'MAHN_HOECHST', 'AGE']

```

In [326]:
```
df.dtypes
```

Out[326]:
```
ORDER_ID                      int64
Z_CARD_VALID                float64
VALUE_ORDER                 float64
AMOUNT_ORDER                  int64
ANUMMER_01                    int64
SESSION_TIME                  int64
AMOUNT_ORDER_PRE              int64
VALUE_ORDER_PRE             float64
MAHN_AKT                    float64
MAHN_HOECHST                float64
AGE                           int64
B_EMAIL_yes                   uint8
B_TELEFON_yes                 uint8
FLAG_LRIDENTISCH_yes          uint8
FLAG_NEWSLETTER_yes           uint8
Z_METHODE_credit_card         uint8
Z_METHODE_debit_card          uint8
Z_METHODE_debit_note          uint8
Z_CARD_ART_Eurocard           uint8
Z_CARD_ART_Not Applicable     uint8
Z_CARD_ART_Visa               uint8
Z_CARD_ART_debit_card         uint8
WEEKDAY_ORDER_Monday          uint8
WEEKDAY_ORDER_Saturday        uint8
WEEKDAY_ORDER_Sunday          uint8
WEEKDAY_ORDER_Thursday        uint8
WEEKDAY_ORDER_Tuesday         uint8
WEEKDAY_ORDER_Wednesday       uint8
CHK_LADR_yes                  uint8
CHK_RADR_yes                  uint8
CHK_KTO_yes                   uint8
CHK_CARD_yes                  uint8
FAIL_LORT_yes                 uint8
FAIL_LPLZORTMATCH_yes         uint8
NEUKUNDE_yes                  uint8
period_of_day_evening         uint8
period_of_day_morning         uint8
period_of_day_night           uint8
dtype: object
```

In [327]:
```
df[continuous_cols] = scaler.fit_transform(df[continuous_cols])

```

In [328]:
```
test_df = df
test_df.to_csv('cleaned_test.csv', index=False)

```

In [329]:
```
train_df.dtypes
```

```
Z_CARD_ART_Not Applicable     uint8
Z_CARD_ART_Visa               uint8
Z_CARD_ART_debit_card         uint8
WEEKDAY_ORDER_Monday          uint8
WEEKDAY_ORDER_Saturday        uint8
WEEKDAY_ORDER_Sunday          uint8
WEEKDAY_ORDER_Thursday        uint8
WEEKDAY_ORDER_Tuesday         uint8
WEEKDAY_ORDER_Wednesday       uint8
CHK_LADR_yes                  uint8
CHK_RADR_yes                  uint8
CHK_KTO_yes                   uint8
CHK_CARD_yes                  uint8
FAIL_LORT_yes                 uint8
FAIL_LPLZORTMATCH_yes         uint8
NEUKUNDE_yes                  uint8
period_of_day_evening         uint8
period_of_day_morning         uint8
period_of_day_night           uint8
dtype: object
```

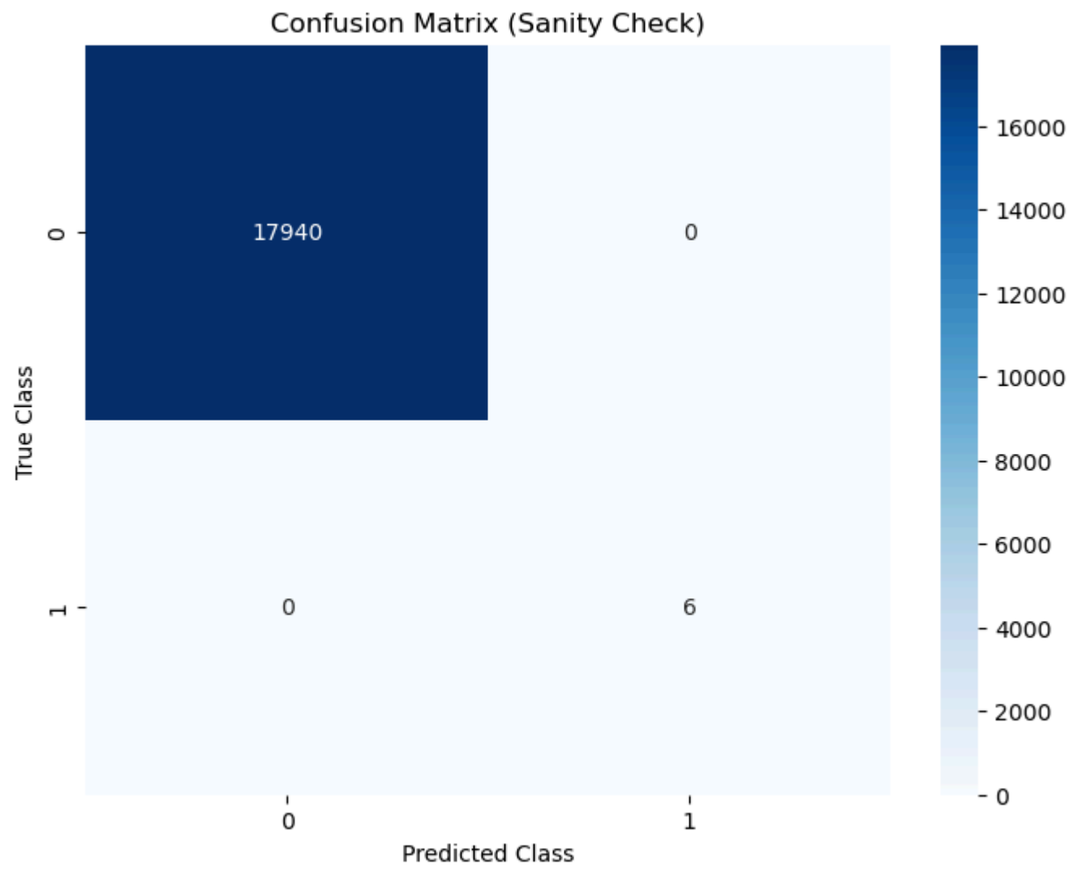
In [330]:
```
test_df.dtypes
```

Out[330]:
```
ORDER_ID                      int64
Z_CARD_VALID                float64
VALUE_ORDER                 float64
AMOUNT_ORDER                float64
ANUMMER_01                    int64
SESSION_TIME                float64
AMOUNT_ORDER_PRE            float64
VALUE_ORDER_PRE             float64
MAHN_AKT                    float64
MAHN_HOECHST                float64
AGE                         float64
B_EMAIL_yes                   uint8
B_TELEFON_yes                 uint8
FLAG_LRIDENTISCH_yes          uint8
FLAG_NEWSLETTER_yes           uint8
Z_METHODE_credit_card         uint8
Z_METHODE_debit_card          uint8
Z_METHODE_debit_note          uint8
Z_CARD_ART_Eurocard           uint8
Z_CARD_ART_Not Applicable     uint8
Z_CARD_ART_Visa               uint8
Z_CARD_ART_debit_card         uint8
WEEKDAY_ORDER_Monday          uint8
WEEKDAY_ORDER_Saturday        uint8
WEEKDAY_ORDER_Sunday          uint8
WEEKDAY_ORDER_Thursday        uint8
WEEKDAY_ORDER_Tuesday         uint8
WEEKDAY_ORDER_Wednesday       uint8
CHK_LADR_yes                  uint8
CHK_RADR_yes                  uint8
CHK_KTO_yes                   uint8
CHK_CARD_yes                  uint8
FAIL_LORT_yes                 uint8
FAIL_LPLZORTMATCH_yes         uint8
NEUKUNDE_yes                  uint8
period_of_day_evening         uint8
period_of_day_morning         uint8
period_of_day_night           uint8
dtype: object
```

In [331]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix

X_test_new = test_df

# Ensure X_test_new has the same columns as X_train after preprocessing
X_test_new = pd.get_dummies(X_test_new)  # One-hot encoding for categorical variables
X_test_new = X_test_new.reindex(columns=X.columns, fill_value=0)

# Predict using the trained classifier
y_pred_new = clf.predict(X_test_new)

# Prepare results dataframe with ORDER_ID and predicted CLASS
result_df = pd.DataFrame({'ORDER_ID': test_df['ORDER_ID'], 'CLASS': y_pred_new})

# Save predictions to a CSV file (optional)
# result_df.to_csv('predicted_classes.csv', index=False)

# Print shape of result_df for sanity check
print(result_df.shape)

# Calculate accuracy (sanity check using predicted labels)
# Since we don't have true labels for test set, this is just for illustration
accuracy_sanity = accuracy_score(y_pred_new, y_pred_new)  # Using predictions as "true" labels
print(f'Accuracy (Sanity Check): {accuracy_sanity:.1f}')

# Generate confusion matrix (sanity check)
conf_matrix_sanity = confusion_matrix(y_pred_new, y_pred_new)
print('Confusion Matrix (Sanity Check):')
print(conf_matrix_sanity)

# Plot confusion matrix (sanity check)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_sanity, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion Matrix (Sanity Check)')
plt.show()

```

```
(17946, 2)
Accuracy (Sanity Check): 1.0
Confusion Matrix (Sanity Check):
[[17940     0]
 [    0     6]]
```



In [332]:

```
result_df
```

Out[332]:

	ORDER_ID	CLASS
0	49916	0
1	49918	0
2	49920	0
3	49921	0
4	49922	0
...	...	...
19995	49820	0
19996	49822	0
19997	49823	0
19998	49826	0
19999	49827	0

17946 rows × 2 columns

In [333]:

```
# Assuming result_df is your Pandas DataFrame

# Specify the file path for the output text file
output_file = 'result.txt'

# Convert DataFrame to text file
result_df.to_csv(output_file, sep='\t', index=False)

print(f"DataFrame successfully saved as {output_file}")

```

```
DataFrame successfully saved as result.txt
```