HW2: Final Project

Jarne van Dongen 411187097

Dataset

For my dataset I used a dataset about Credit card transactions from 2019, I found the original dataset on Kaggle. (https://www.kaggle.com/datasets/priyamchoksi/credit-card-transactions-dataset/data)

This dataset has more then 1.000.000 data points. This is a bit much for the scope of this project.
To make the dataset more usable, I wrote a Python program that get's 15.000 data points evenly spread across the original data.

This data gets saved in a new data set that I can use. This way I can use this smaller dataset but I still have data from a full year (2019).

My newly transferred dataset can be downloaded in Kaggle.
(https://www.kaggle.com/datasets/brogaming/15000-card-transactions-datamíng)

```python
import pandas as pd

# Variable for path to CSV file
csv_file = 'credit_card_transactions.csv'

# Look for an interval so I can evenly get 15.000 lines
total_lines = sum(1 for line in open(csv_file)) - 1
sample_size = 15000
interval = total_lines // sample_size

# Read the CSV but skip lines so we only get 15.000 lines
df = pd.read_csv(csv_file, skiprows=lambda x: x % interval != 0)

#delete the first row and add a new first row so the lines are numbered correctly
df = df.iloc[:, 1:]
df.insert(0, 'line_number', range(1, len(df) + 1))

#save the sampled file
df.to_csv('sampled_file.csv', index=False)
```

Figure 1: Python Program to select 15.000 evenly spread lines

Introduction to the dataset

Content

The dataset includes various attributes for each transaction, such as:

- **Transaction Details**: date and time, card number (hashed) , merchant, category, and amount.
- **Cardholder Information**: Name, gender, address, city, state, Occupation and date of birth.
- **Fraud status**: a value indicating fraud status.

Features

1. **line_number**: Unique number for each transaction. Interval Numerical
2. **trans_date_trans_time**: Timestamp of the transaction. Interval Numerical
3. **cc_num**: The credit card number used. Nominal Categorical
4. **merchant**: The merchant where the transaction occurred. Nominal Categorical
5. **category**: The type of thing/service that is bought. Nominal Categorical
6. **amt**: The amount spend. (In USD) Ratio Numerical
7. **first:** First name of the cardholder. Nominal Categorical
8. **last:** Last name of the cardholder. Nominal Categorical
9. **gender:** Gender of the cardholder. Nominal Categorical
10. **street:** Street name where the cardholder lives. Nominal Categorical
11. **city:** City where the cardholder lives. Nominal Categorical
12. **state:** State abbreviation where the cardholder lives. Nominal Categorical
13. **zip:** Zip code of the cardholder. Nominal Categorical
14. **lat:** Latitude of the address of the cardholder. Ratio Numerical
15. **long:** Longitude of the address of the cardholder. Ratio Numerical
16. **city_pop:** City population of the city of the cardholder. Ratio Numerical
17. **job**: The cardholder's job title. Nominal Categorical
18. **dob**: The cardholder's birth date. Interval Numerical
19. **trans_number**: A unique identifier for the transaction. Nominal Categorical
20. **unix_time:** The time the purchase is done standardized to unix time. (amount of seconds since 00:00:00 UTC, 1 January 1970) Ratio Numerical
21. **merch_lat:** Latitude of the address of the merchant. Ratio Numerical
22. **merch_long:** Longitude of the address of the merchant. Ratio Numerical
23. **is_fraud**: Indicator of whether the transaction is fraudulent. Nominal Categorical
24. **merch_zip:** Zip code of the merchant. Nominal Categorical

First 10 rows of the dataset

line_number	trans_date_trans_time	cc_num	merchant	category	amt	first
1	2019-01-01 00:42:26	4,51E+15	fraud_Macejkovic-Lesch	shopping_pos	8,57	Margaret
2	2019-01-01 01:31:53	5,02E+11	fraud_Weber and Sons	food_dining	98,24	Melissa
3	2019-01-01 02:14:41	4,39E+12	fraud_Wiza, Schaden and Stark	misc_pos	34,76	Charles
4	2019-01-01 03:04:28	4,59E+18	fraud_Hills-Olson	grocery_net	25,89	Amber
5	2019-01-01 03:57:43	2,13E+14	fraud_Ruecker, Beer and Collier	shopping_net	87,91	Craig
6	2019-01-01 04:45:43	4,51E+18	fraud_Rau and Sons	grocery_pos	124,7	Monica
7	2019-01-01 05:39:44	3,03E+13	fraud_Bartoletti-Wunsch	gas_transport	56,65	Christine
8	2019-01-01 06:24:59	3,51E+15	fraud_Corwin-Collins	gas_transport	79,27	Christine
9	2019-01-01 07:10:26	4,06E+18	fraud_Christiansen, Goyette and Schamberger	gas_transport	92,76	Patricia
10	2019-01-01 07:57:37	3,82E+13	fraud_Jast-McDermott	shopping_pos	127,2	Jesse

last	gender	street	city	state	zip	lat	long
Williams	F	165 Jerry Meadows Suite 460	Surrency	GA	31563	31,6489	-82,1982
Phillips	F	5069 Scott Pass Apt. 654	Meadville	MS	39653	31,4285	-90,8578
Rodriguez	M	240 Tracy Forges	Easton	KS	66020	39,3391	-95,0999
Lewis	F	6296 John Keys Suite 858	Pembroke Township	IL	60958	41,0646	-87,5917
Franco	M	9242 Vanessa Ramp Apt. 525	Smithfield	IL	61477	40,4855	-90,2856
Cohen	F	864 Reynolds Plains	Uledi	PA	15484	39,8936	-79,7856
Johnson	F	8011 Chapman Tunnel Apt. 568	Blairsden-Graeagle	CA	96103	39,8127	-120,641
Burns	F	343 Hannah Parkway	Comfort	WV	25049	38,1372	-81,5962
Mendoza	F	1683 Davidson Freeway	Mendon	UT	84325	41,71	-111,982
Roberts	M	8415 Vaughn Squares Apt. 788	Acworth	NH	3601	43,196	-72,3001

city_pop	job	dob	trans_num	unix_time	merch_lat	merch_long	is_fraud	merch_zipcode
1324	Engineer, technical sales	1926-07-12	70ed1a189a6bce479e564b120f2379d9	1325378546	32,023302	-82,58345	0	30473
2799	Therapist, horticultural	1961-01-21	6658882e9274e3d6fc3dffa5593b8ad6	1325381513	30,906565	-90,486622	0	70444
1442	Air broker	1982-05-20	e6e3f462abfffdf22115360310622e20	1325384081	38,78936	-95,405907	0	66047
2135	Psychotherapist, child	2004-05-08	76c33ab500644d7795159d1a0eaa243c	1325387068	40,888445	-87,409615	0	47922
631	Futures trader	1973-02-14	3ff25009cf03ea68e805219047373cb2	1325390263	40,55495	-89,50587	0	61535
328	Tree surgeon	1983-07-25	12edfc2306c9e242adccce59a838cbac	1325393143	39,088298	-79,538272	0	26271
1725	Chartered legal executive (England and Wales)	1967-05-27	c198762abf80d696e56eaa0646632488	1325396384	39,25212	-120,809112	0	95959
630	Fine artist	1959-07-30	b47ec7498c7c19de4801a850b99c433f	1325399099	38,686653	-80,803493	0	26624
2078	Scientist, audiological	1963-06-13	caedb2e95360b4615a85193357810e38	1325401826	42,599951	-112,758179	0	83271
477	Naval architect	1988-04-15	df2704afe7f9631383e6de2cd787eba0	1325404657	43,32428	-73,067471	0	5739

Table 2: First 10 rows of my dataset.

Target/application

Application 1:
This dataset is interesting for a couple of reasons. First, it contains a large amount of cardholder data, which allows us to analyse purchasing patterns. We can determine valuable information about consumer behaviour by looking at who and where money is spend.
→ I assume that people from bigger populated cities have a bigger spending pattern then people in smaller cities.

Application 2:
There is also data about the product/service being purchased. This can us help determine, when a specific category of products is bought the most and by who. Understanding and analysing this data can help us with targeted advertisement.
→ I assume that in the summer vacation there are more experience based (services) spendings and in the winter (holiday season) more items (goods).

Application 3:
Another key data point is the data point about fraud. By using machine learning or other data analysing techniques we could possibly find fraudulent transactions. This could help people that will otherwise lose money by this transactions.
→ I hope to find a clear feature in the data that can help me categorize a transaction into fraudulent transactions or honest transactions.

Data preprocessing

First I check if there is any data missing in my dataset.

```
import pandas as pd

df = pd.read_csv('Data\sampled_file.csv')

missing_data = df.isnull().sum()

print("Missing data points in each column:")
print(missing_data[missing_data > 0])
```

Figure 2: Code to check if there is any data missing.

After running this code I found out that the “merch_zipcode” feature has 2302 rows that are not filled in. Because we also have the city that the product is bought in the postal code is not that important for our analysis. I will discard the “merch_zipcode” column entirely.

Now I will clean up my data, some columns are not necessary to keep for our analytics, so we will discard this columns. The columns I will drop are:

- first -> I want my data to make general conclusions about groups of people, so we don’t need specific names of people. This will also make our data more anonymous.
- last
- cc_num
- street -> I want to look at the city/state where people buy their products, I am not interested in the specific street.
- zip -> Because we already have the city name in our dataset, we don’t need the zip codes anymore.
- (merch_)lat -> I won’t use this super specific address details.
- (merch_)long
- trans_num -> We can identify the transaction by the line_number so we don’t need this transaction number separately

You could say I should also drop my “trans_date_trans_time” column because we have the unix time saved in our data frame. I won’t do this because I will need the months to determine if a transaction is done in the summer or winter. By having the datetime stamp this will be much easier.

```
df = df.drop(columns=['merch_zipcode'])
df = df.drop(columns=['first'])
df = df.drop(columns=['last'])
df = df.drop(columns=['street'])
df = df.drop(columns=['zip'])
df = df.drop(columns=['lat'])
df = df.drop(columns=['long'])
df = df.drop(columns=['merch_lat'])
df = df.drop(columns=['merch_long'])
df = df.drop(columns=['trans_num'])
df = df.drop(columns=['cc_num'])
```

Figure 3: Drop columns that are not necessary.

I have the date of birth in my table, it's more practical for our analysis to have the age of the costumer by hand. We add an age column to our dataset using the "dob" column. After adding this age column we can drop the "day of birth" column entirely. This is called feature engineering.

```
def calculate_age(dob):
    dob = datetime.strptime(dob, '%Y-%m-%d')
    today = datetime.today()
    age = today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))
    return age
df['age'] = df['dob'].apply(calculate_age)
df = df.drop(columns=['dob'])
```

Figure 4: Calculate the age out of the date of birth.

Eventually I also want to get rid of some outliers in my dataset. This outliers could influence the mean or other statistics that I will eventually use in my analysis. To calculate the outliers I used the Z_scores, every data that has a higher Z-score then 3 will get discarded. This is done for some important numerical features.

```
numerical_features = ['amt', 'city_pop', 'age']

z_scores = np.abs(stats.zscore(df[numerical_features]))
outliers = (z_scores > 3).any(axis=1)
df_cleaned = df[~outliers]
```

Figure 5: Get rid of outliers using the Z_scores.

The preprocessed dataset can be found here:
https://www.kaggle.com/datasets/brogaming/preprocessed-dataset-transactions-dataming/data

Preprocessing for pattern mining (Application 1):

I want to use pattern mining to see what kind of categories of goods or services are bought more in big cities against small cities.

For this application we firstly need to make a column that contains if a city is big or small, I determined that a big city is a city from 20.000 people (this dataset contains American cities).

```
df['Big_city'] = df['city_pop'].apply(lambda x: True if x > 20000 else False)
```

Figure 6: Make a new column where big cities are True and small cities are False.

Because we want to get useful association rules we should have more then one category per transaction. Our original dataset only has one category per transaction. I really want to test out the Apriori algorithm so I made some code to randomly add a random amount of categories to each transaction. We can then get useful information out of the association rules.

```
def add_random_categories(transaction, transaction_data, max_categories=4):
    all_categories = transaction_data['category'].explode().unique()
    remaining_categories = list(set(all_categories) - set(transaction))

    num_categories_to_add = min(max_categories - len(transaction), len(remaining_categories))
    categories_to_add = random.sample(remaining_categories, num_categories_to_add)
    transaction.extend(categories_to_add)

    return transaction

transaction_data['category'] = transaction_data['category'].apply(
    lambda x: add_random_categories(x, transaction_data, max_categories=4)
)
```

Figure 7: Add random categories to each transaction so I can use Apriori algorithm.

After that I one-hot encode the category column so it becomes a binary matrix. This is necessary for the Apriori algorithm, that can only accept True or False values.

This preprocessed dataset can be found on Kaggle:
https://www.kaggle.com/datasets/brogaming/preprocessed-dataset-for-apriori

Preprocessing for Decision three model (Application 2):

We want to look if services or goods are most popular in the winter or summer. Before we van find this information we have to do some feature engineering to get the necessary columns.

Out of the "trans_date_trans_time" column I can get the month that the transaction is done, after I determine what months are winter and summer months and make two extra columns.

If the month of the transaction is a winter months it will add a "1" to that column, and visa versa for the summer months.

```
df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'])
df['month'] = df['trans_date_trans_time'].dt.month
winter_months = [10, 11, 12, 1, 2, 3]
summer_months = [4, 5, 6, 7, 8, 9]
df['winter'] = df['month'].apply(lambda x: 1 if x in winter_months else 0)
df['summer'] = df['month'].apply(lambda x: 1 if x in summer_months else 0)
```

Figure 8: Make two columns with winter and summer month encoding.

For the services and goods columns I first determined which categories fit in what field and then did the same process with filling in 1's and 0's in the corresponding columns.

```
services = ['food_dining', 'travel', 'personal_care', 'health_fitness', 'entertainment']
goods = ['shopping_pos', 'grocery_net', 'shopping_net', 'grocery_pos', 'gas_transport', 'misc_pos', 'misc_net', 'kids_pets', 'home']
df['Service'] = df['category'].apply(lambda x: 1 if x in services else 0)
df['Goods'] = df['category'].apply(lambda x: 1 if x in goods else 0)
```

Figure 9: Make two columns with service and goods encoding

We can save this four new columns and have the necessary data for our Decision three.

Data can be found on Kaggle:
https://www.kaggle.com/datasets/brogaming/preprocessed-dataset-for-decision-three

Preprocessing for Classification model (Application 3):

Before we can train our classification model we should do some more preprocessing on our data. Most machine learning models can only accept numerical values and so every categorical string value has to be encoded into integers.

To do this we can use label encoding and/or one hot encoding. Label encoding will give every unique value in a column a number label, this will result in one column for each column that has to be encoded. But it can cause for trouble because the AI model can think that the relation between the numbers has a specific meaning while that's not the case.

One-hot encoding changes every unique value in a column to it's own column and add binary values to it (1 where the column is True and 0 where it's False). This can cause for a lot of columns if we one hot encode a column with lots of unique values.

We choose to use one hot encoding on columns that don't have a lot of unique values and label encoding on the other ones.

Every numerical value has to be normalized, this so the model doesn't have to work with different scales and different order of values.

```
categorical_columns_onehot = ['gender', 'category']
categorical_columns_label = ['merchant', 'city','state', 'job']
numerical_columns = ['line_number', 'amt', 'city_pop','unix_time', 'age']

scaler = StandardScaler()
encoder = OneHotEncoder()
label = LabelEncoder()

for col in categorical_columns_label:
    data[col] = label.fit_transform(data[col])

scaled_data = scaler.fit_transform(data[numerical_columns])
scaled_df = pd.DataFrame(scaled_data, columns=numerical_columns)

encoded_data_onehot = encoder.fit_transform(data[categorical_columns_onehot]).toarray()
encoded_df_onehot = pd.DataFrame(encoded_data_onehot, columns=encoder.get_feature_names_out(categorical_columns_onehot))

combined_df = pd.concat([scaled_df, encoded_df_onehot, data[categorical_columns_label], data['is_fraud']], axis=1)
```

Figure 10: Preprocessing for the classification model.

The data from this preprocessing can be found on Kaggle:
https://www.kaggle.com/datasets/brogaming/preprocessed-ai-dataset-transactions-datamining

Tools/methods

Tools

I will be using Python for the data mining techniques I will do at this dataset. I chose Python because it's well-known for it's data manipulation, analytics and machine learning possibilities.

Python has a lot of useful libraries like:

- Panda's for data structures and manipulation
- scikit-learn for Machine learning techniques
- sciPy for statistical calculations
- NumPy for mathematical practices
- Mlxtend for pattern mining algorithms
- MatPlotLib for visualisation of the results
- …

All data will be read and saved from and to a CSV file.

Code can be found on my GitHub:
https://github.com/jarne2703/Datamining_Final_project

Methods

Application 1:

To check if people are buying more in bigger cities, I will use the **Apriori algorithm**. This is a method for discovering patterns in large datasets. It can identify frequent itemsets and the relationships between them. By using the Apriori algorithm to my transaction data, I can discover trends in product purchases in cities with high populations and see if they differ from smaller cities.

```
frequent_itemsets = apriori(transaction_df, min_support=0.05, use_colnames=True, max_len=5,)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6, num_itemsets=len(df))
```

Figure 11: Frequent itemsets and association rules calculation.

I used the Apriori function from the mlxtend library.
min_support=0.05: This parameter sets the minimum support threshold. Itemsets must appear in at least 5% of the transactions to be considered frequent. I choice 5% because it's a nice general balance between a too low or too high minimum.
use_colnames=True: Uset the original item names from the dataset.
max_len=5: Limits the maximum size of itemsets considered (up to 5 items per itemset)
This was chosen because we don't want itemsets to be too big, then the fact that they are together wouldn't be that useful to know.

The Association_rules function gives us the rules based on the frequent itemsets we found.
metric="confidence": The metric used to evaluate the strength of the rules. Confidence measures the probability that item B is purchased when item A is purchased.
min_threshold=0.6: The minimum threshold for the chosen metric (confidence). Only rules with confidence values greater than or equal to 0.6 will be considered.
num_itemsets=len(df): The number of itemsets to consider when generating rules. We set it on the length of the dataset.

I want to see what the difference is in patterns between the big and small cities so I also made a function to filter the dataset into a subset using a variable and value. The subset is getting the same frequent_itemsets and rules function to calculate the association rules.

```
def run_apriori_on_subset(transaction_data, city_value, city_column):
    subset_data = transaction_data[transaction_data[city_column] == city_value]
    frequent_itemsets_subset = apriori(subset_data, min_support=0.05, use_colnames=True)
    rules_subset = association_rules(frequent_itemsets_subset, metric="confidence", min_threshold=0.6, num_itemsets=len(subset_data))

    return frequent_itemsets_subset, rules_subset

frequent_itemsets_big, rules_big = run_apriori_on_subset(transaction_df, True, 'Big_city')
frequent_itemsets_small, rules_small = run_apriori_on_subset(transaction_df, False, 'Big_city'
```

Figure 12: Frequent itemsets and association rules calculation on subsets.

Application 2:

For this application, I will use a **decision tree** to predict whether products are mostly bought in the winter or summer. A decision tree works by splitting data based on the most important features. By having good splits we can classify items into different categories. We can train the model (choosing the splits) by using the data in our dataset.

```
X = df[['winter', 'summer']]
y = df['Service']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

clf = DecisionTreeClassifier(class_weight='balanced', random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Figure 13: Training and predictions with decision tree.

I define the X and Y labels for my Decision three. I use service so I can always classify that and after I can do the same for goods.

I use the train_test_split function from the sklearn.model_selection to split the data into training and test data.
I use a test_size of 30%, so that I can use 70% (most of the data) to train the model. The random_state is a seed for the randomness so that the result of this split stays the same between runs. This is convenient during the testing and developing of the model.

I used the DecisionTreeClassifier function from sklearn.tree to train the decision tree model. The class_weight='balanced' variable is used to help our imbalanced dataset get more balanced. The model will assign higher weights to unrepresented data values so that these values will also be considered in the training process. My dataset is a bit unbalanced because there are more goods sold then Services, that's why I used this variable.

.fit, will fit the model (train) on the training data.

After I predict some things with the decision three using the test data. And the I print the classification report and confusion matrix to see if the model is good.

Application 3:

To see if a transaction is fraud, I will use the **Random Forest** algorithm. This method creates multiple decision trees and combines their results to make more accurate predictions. It's great for detecting fraud because it can handle a lot of features in the data. Using these features, the model will classify transactions as fraudulent or not.

```python
X = data.drop('is_fraud', axis=1)
y = data['is_fraud']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(n_estimators=20, random_state=42)
model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

Figure 14: Random forest training and accuracy calculations.

I use all available features in the data for classifying if the transaction is fraud or not. The Y-label is fraud.

I use the same train_test_split to split my data into train and testing data. Here I use a test_size of 20% because this data is much more, so I will still have plenty of test data, but I can use more data to actually train the model. This will make it more accurate.

n_estimators=20: The model uses 20 decision trees in the forest, balancing performance and computational cost. The moment I go above 20 decision threes the model's training accuracy will become near / to 100%. This clearly shows that the model is overfitting on the data, and that's not good.

After training I predict and check with my testing data and print the accuracy.

Results

Application 1:

If I print the Top Frequent itemsets generated by the apriori algorithm we can see the following results:

Top Frequent Itemsets (Big City):
(gas_transport)
(shopping_pos)
(kids_pets, grocer_pos, misc_pos)
(personal_care, grocer_pos)
(entertainment)

Top Frequent Itemsets (Small City):
(misc_net, health_fitness)
(grocery_pos, gas_transport)
(kids_pets, grocer_pos)
(home)

Most itemsets are very easy to explain. Gas_transport in big cities, because in America there is not a lot of good public transportation, so a lot of people have to buy gas every day to go to work. Other itemsets like entertainment and shopping are very logical in big cities because malls are located here so people from smaller cities will come here to do this transactions.

In the small cities we see a lot of necessary (mostly weekly) purchases. This because people want to buy them quickly and close to home. The items that are together can be found in grocery stores, so that's why the items are probably together a lot.
(If you remember the preprocessing I added categories randomly, so the conclusions from this are of course arbitrary and just for example purpose only)

In the Network diagram we can see that a lot of different categories are connected with each other. This can also be lead to the random generation of the categories.

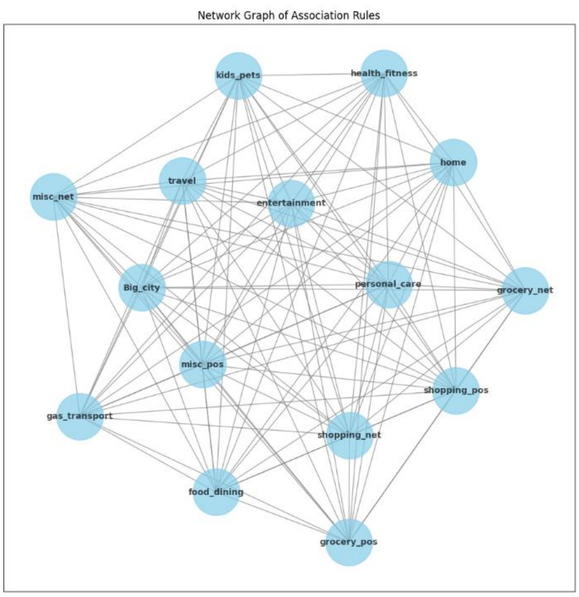

Figure 15: Network visualising the connections between categories.

I also plotted the Lift against the Confidence for the small city and big city rules. We can see that as the Lift improves, the Confidence also increases, but it never really gets super high.

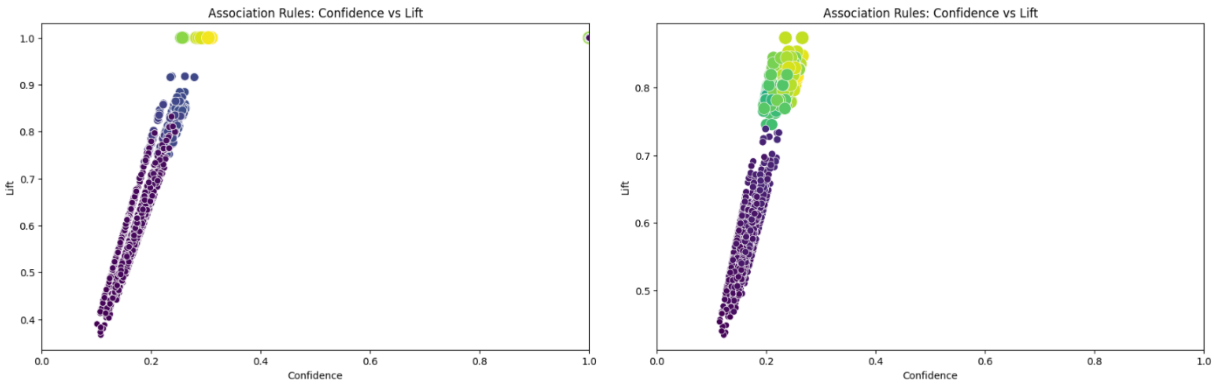

Figure 16: confidence vs lift graph for big cities (Left) and small cities (right).

Lift shows how much more likely two items are to be bought together than by chance. A Lift value greater than 1 indicates a positive relationship between the items (i.e., they occur together more often than expected), while a value of 1 means the items are independent, and a value less than 1 means they occur together less often than expected.

This pattern of moderate Lift and Confidence might suggest that there isn't a strong or reliable association between the items across both big and small cities. This could be a result of randomness in the dataset, and there might not be a clear pattern to be found.

If this is true, the rules we generated may not be reliable enough to generate trustworthy insights or solutions, as they don't show strong and consistent patterns that could be used for prediction or decision-making.

This is a result of the randomness added in the preprocessing step.
I made an Apriori algorithm and found that there are not really relationships that can be trusted in my dataset. Which you expected because of the preprocessing we did.

Application 2:

After developing the decision three we unfortunately have to conclude that the services / goods don't really have a clear correlation with the summer or winter months. I wanted to proof this by getting a successful decision tree and so showing to advertisers what the best target time is to launch their advertising.

Unfortunately my decision tree only has an accuracy of 0.69. It's better then randomly guessing (that would be 50%) but I think this has to do with a unbalance in the amount of data to service based purchases.

We can see this even more clear in the confusion matrix:

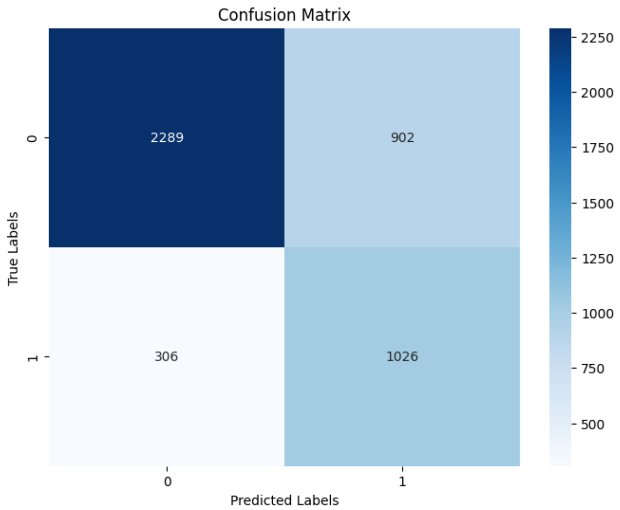

Figure 17: Confusion Matrix of the decision three.

The model gives a lot of False Positives and is therefore not useful. My earlier prediction of using the period a transaction is done is therefor not enough to determine if a transaction is a service or a good.

When I visualize my Decision tree we can see that the Gini scores are relatively high, this also shows us that the data is not homogenous, meaning there is still a mix of classes (transactions for goods and services) within each split. The algorithm can't find a good split because there is no clear pattern in this while looking at the winter or summer months.

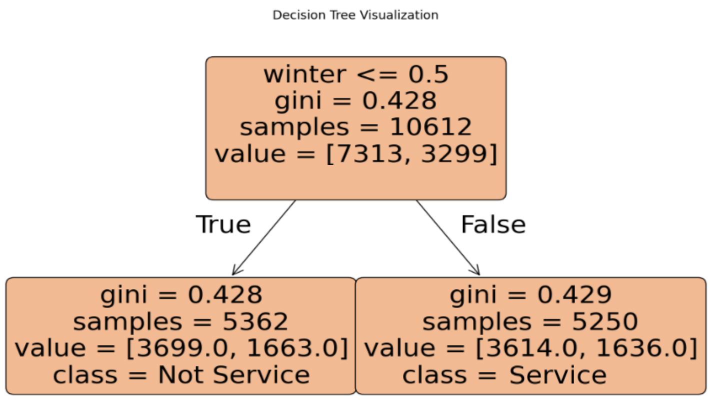

Figure 18: Visualisation of decision tree

Application 3:

Firstly when we print our accuracy we get:

> *Training Accuracy: 0.9696701846965699*
> *Testing Accuracy: 0.9480435212660732*

This is a very nice result, it shows us that it's not yet over overfitted to the training data and still gives a high accuracy on the testing data. We can also see this in the confusing matrix and ROC curve, we have no false positives or false negatives in our test predictions. The area under the ROC curve indicates the performance of the model, we can clearly see that this is about 80% percent and more which means a very good model.

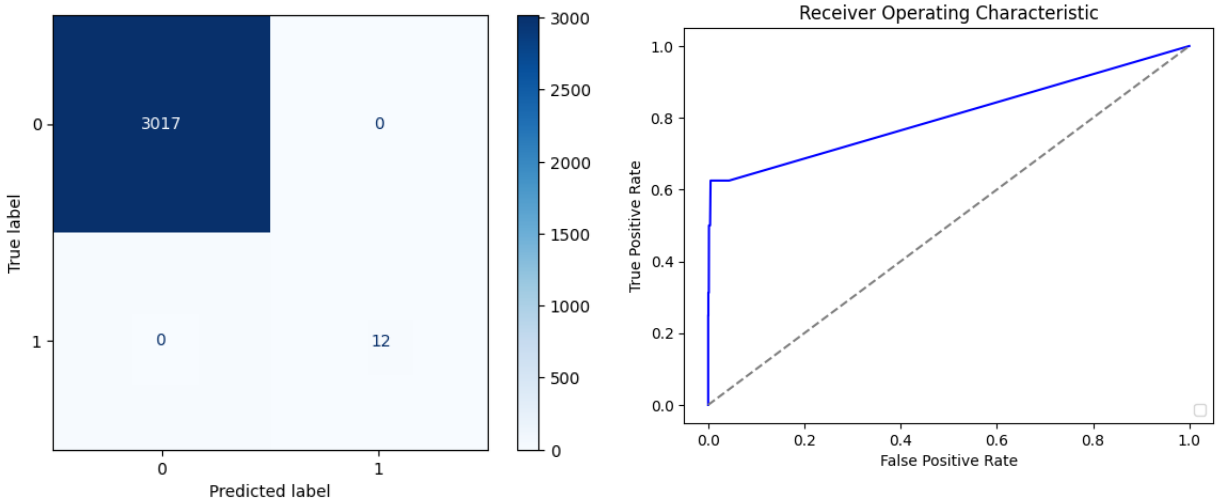

Figure 19 and 20: Confusion matrix and ROC graph of classification model.

If I cross validate the test data to get a more accurate result we also get a good result:

> *Cross-Validation Accuracy Scores:*
> *[0.6857896 0.957124 0.96538259 0.89406332 0.97505277]*
> *Mean CV Accuracy: 0.91*

I wanted to find a feature that can be specifically used to determine if the transaction is fraud or not. We can easily see this from our model using some SKLearn functions.

```python
feature_importances = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': model.feature_importances_
}).sort_values(by='Importance', ascending=False)

plt.barh(feature_importances['Feature'], feature_importances['Importance'])
plt.xlabel("Importance")
plt.ylabel("Features")
plt.title("Feature Importances")
plt.gca().invert_yaxis()
plt.show()
```

Figure 21: Code to find importance of the features.

We get a nice graph from this code where we can clearly see that the amt (amount spend in the transaction) is the most important feature to classify if the transaction is fraud or not.
It's also nice to see some categories where the importance for the model is practically zero, so in this categories there are rarely fraudulent transactions.
We can also see that city is pretty important to look at too, this can give us a view on where most fraudulent transactions are taking place.

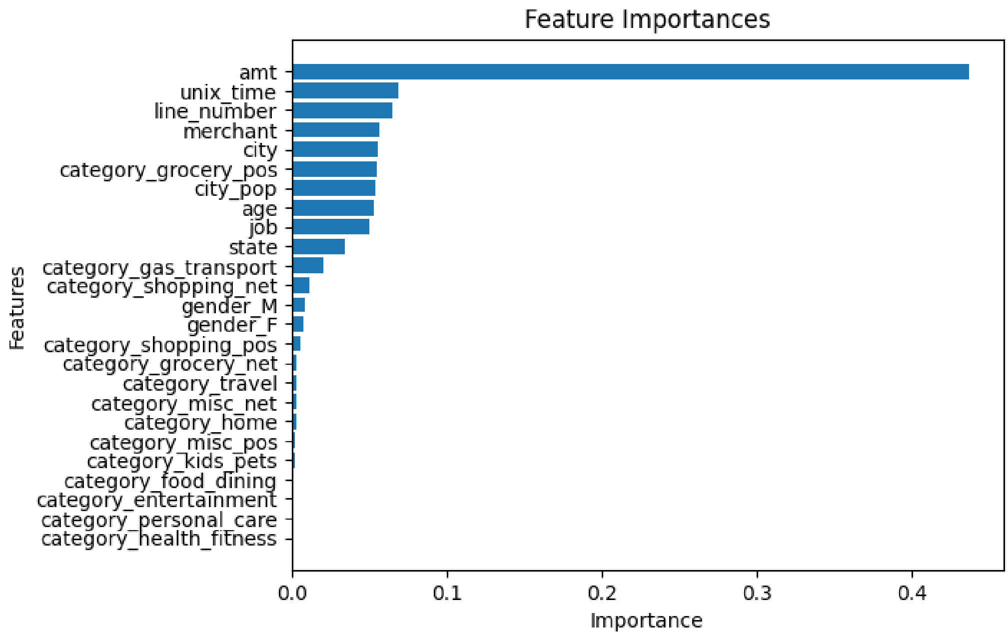

Figure 22: Features ordered by importance for determining if a transaction is fraudulent or not.

Difficulties

I had a lot of difficulties with my data preprocessing. I knew what things where theoretically available from the class, but to actually implement them was very difficult.

I also found it hard to find good metrics and ways to show if my models / association rules where good. I didn't really know how to evaluate them properly.

I watched a lot of data mining / Ai modelling video's and did some Python courses on DataCamp to help me solve my issues.

Feedback

I found the class difficult and challenging, because we didn't have any physical class. The online classes where a bit confusing and not really organized to me.
The project was interesting but because we didn't get any feedback on our Mid-Term yet and we could choose almost everything ourselves it was difficult to know if we were on the right track and what the professor actually expects from us.

Difference from proposal

I didn't have big differences from the proposal. I only pre-processed my data and used it to tackle the problem I described in my proposal.