The Second Part of the Assignment of IDS 2019-2020

Document your results as well as the way you obtained them in this jupyter notebook. Separate reports (pdf, word, etc.) are *not* required. However, it is necessary that you provide the python code leading to your results as well as textual answers to the assignment questions in this notebook. *DO NOT CLEAR THE OUTPUT of the notebook you are submitting!* In the cases that the result of an algorithm is pdf, jpg, etc, you should attach the result to this notebook file and refer to that in the text.

Next to the jupyter notebook, submit one zip-file containing all data sets that you are asked to submit. Make sure they are easily identifiable, i.e. use names as requested in the corresponding question.

Do not change the general structure of this notebook, but you can add further markdown or code cells to explain your solutions if necessary. In the end, submit this file and your created data sets in moodle.

Only one group member should upload your group's solution.*Make sure to include group members' names and matriculation numbers*. If your name and student id are not included in the report, you will not receive any points!

Hint 1: While answering the questions, you will get a better and better impression of the given data. **Ensure that all claims you make are supported by the presented facts!**

Hint 2: **Some of the tasks might need some time to run. Take this into account in your planning.**

Hint 3: RWTHonline allows for multiple submissions (each submission overwrites the previous ones). **Partial submissions are possible and encouraged.** This helps in case of technical problems of RWTHonline, which do seldomly happen.

Student Names and IDs:

1. **Bartosz Lipiński**
2. **Serjoscha Bende**
3.

Preprocessing of the Dataset (5 points)

The provided data set *air_pollution* contains hourly results of measuring the concentration of certain substances (CO, Benzene, NO2, particulate matter) and environmental conditions (temperature, relative humidity, traffic volume). This data was collected near a busy street in a city center by an automated device.

You should carry out some preprocessing steps before starting the analysis:

- **Select 90% of 'air_pollution' dataset by random sampling. Use one of the group member's student number as a seed.**
- **After completing this preprocessing step, export your final dataset as 'air_pollution_2.csv' dataset and use that for the next steps of the assignment.**
- **If it is not directly mentioned, you should always use your extracted (above-created) dataset.**
- **Important! Make sure that you submit your extracted dataset with your result in moodle.**

In [42]:

```
import pandas as pd

seed = 344493

csv = pd.read_csv("air_pollution.csv")

csv = csv.sample(frac=0.9, random_state=seed).sort_index()
csv.to_csv("air_pollution_2.csv", index=False)
```

Question 1 - Data Preprocessing and Data Quality (10 points)

For this question, use the extracted data set you created in the preprocessing step ('air_pollution_2.csv'), but without the features *Traffic_Volume* and *Particulate_Matter*. Remove those columns before answering the questions.

In [12]:

```
#import libraries and load data set
```

```
#import libraries and data set
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data1 = pd.read_csv("air_pollution_2.csv")
data1 = data1.drop(columns=["Traffic_Volume", "Particulate_Matter"])
```

In [13]:

```
data1.head()
```

Out[13]:

	Date	Time	CO	Benzene	NO2	Temperature	Humidity(%)
0	10.03.04	23:00:00	1.2	4.7	96	11.2	59.2
1	10.03.04	22:00:00	1.6	6.5	116	11.2	59.6
2	10.03.04	19:00:00	2.0	9.4	92	13.3	47.7
3	10.03.04	18:00:00	2.6	11.9	113	13.6	48.9
4	10.03.04	21:00:00	2.2	9.2	122	11.0	60.0

(a) We want to get a first impression of the data. To achieve this, compute and show the following:

- the column names (names of the features)
- the data type of each feature
- for time features: the minimum and maximum
- for numerical (non-time) features: the mean, standard deviation, minimum and maximum
- for categorical features: the number of classes and the value of the most frequent class

In [14]:

```
# No categorical features exist
statistics = pd.DataFrame({
    "data_type": [],
    "min": [],
    "max": [],
    "mean": [],
    "std": [],
    "num_classes": [],
    "most_frequent": []
})
statistics["data_type"] = data1.dtypes
statistics["min"] = data1.min()
statistics["max"] = data1.max()
statistics["mean"] = data1.mean()
statistics["std"] = data1.std()
dates = [x.split(".")[::-1] for x in data1["Date"]]
times = [list(map(int, x.split(":"))) for x in data1["Time"]]

datemin = ".".join(min(dates)[::-1])
datemax = ".".join(max(dates)[::-1])
timemin = ":".join([str(x) for x in min(times)])
timemax = ":".join([str(x) for x in max(times)])

statistics.at["Date", "min"] = datemin
statistics.at["Date", "max"] = datemax
statistics.at["Time", "min"] = timemin
statistics.at["Time", "max"] = timemax

statistics
```

Out[14]:

	data_type	min	max	mean	std	num_classes	most_frequent
Date	object	10.03.04	04.04.05	NaN	NaN	NaN	NaN
Time	object	0:0:0	23:0:0	NaN	NaN	NaN	NaN

	data_type	min	max	mean	std	num_classes	most_frequent
CO	float64	-200	11.9	-34.580418	77.961336	NaN	NaN
Benzene	float64	-200	63.7	1.646847	41.884722	NaN	NaN
NO2	int64	-200	340	57.350552	127.304172	NaN	NaN
Temperature	float64	-200	44.6	9.548628	43.711761	NaN	NaN
Humidity(%)	float64	-200	88.7	39.337585	51.790753	NaN	NaN

(b) Consider the features 'date' and 'time'. Combine them into a new column 'Datetime' using the datetime data type, and drop the old columns 'date' and 'time'.

Which of the two representations of date and time do you think is more suitable for most analysis applications?

Explanation:

Often you are interested in how a certain value develops during the day. For that the split representation is handy. But to get a real overview over how a timeseries develops over tim e, the combined datetime represenation is naturally better suited. Also, for the datetime d ata type there already exist a lot of methods to handle time ranges etc..

So the combined representation is more versatile and therefore better suited for most analy sis applications.

In [15]:

```
data1["Datetime"] = [pd.to_datetime(row["Date"] + " " + row["Time"], dayfirst=True) for index, row
in data1.iterrows()]
data1 = data1.drop(columns=["Date", "Time"])
data1.head()
```

Out[15]:

	CO	Benzene	NO2	Temperature	Humidity(%)	Datetime
0	1.2	4.7	96	11.2	59.2	2004-03-10 23:00:00
1	1.6	6.5	116	11.2	59.6	2004-03-10 22:00:00
2	2.0	9.4	92	13.3	47.7	2004-03-10 19:00:00
3	2.6	11.9	113	13.6	48.9	2004-03-10 18:00:00
4	2.2	9.2	122	11.0	60.0	2004-03-10 21:00:00

(c) For each feature corresponding to a measured value, provide a simple scatter plot showing the data points over time. Can you spot any obvious data quality issues, e.g. inconsistencies, implausible values or missing values (without researching on specific domain knowledge)?

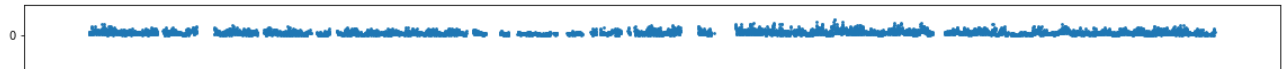
Hint: you may perform additional computations to verify your findings.

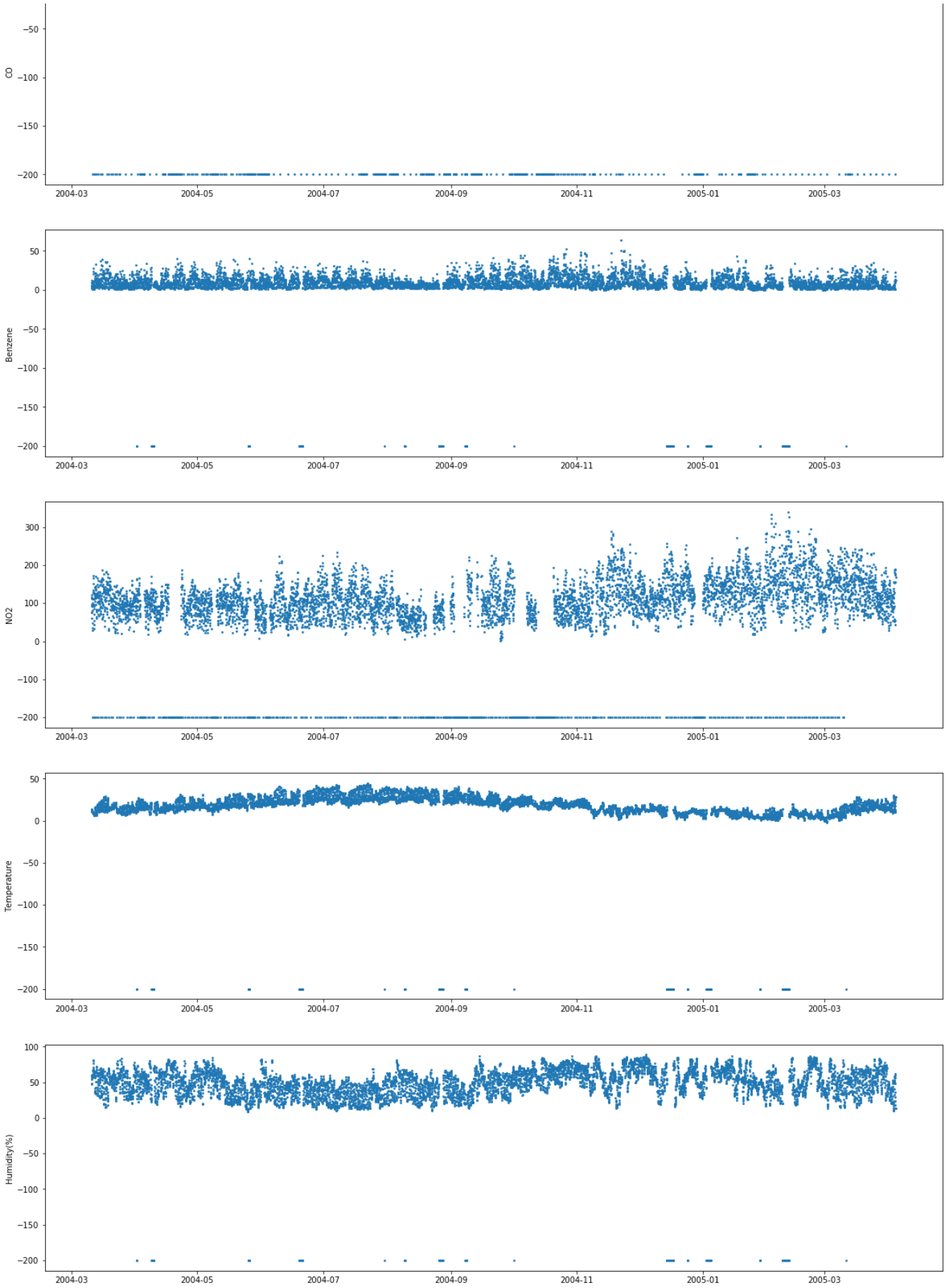
In [16]:

```
def plot_simple_scatter(data, x, features):
    plt.figure(figsize=(20,30))
    for i in range(len(features)):
        subnum = len(features) * 100 + 10 + i+1
        plt.subplot(subnum)
        f = features[i]
        plt.scatter(data[x], data[f], s=3)
        plt.ylabel(f)
    plt.show()
```

In [17]:

```
features = ["CO", "Benzene", "NO2", "Temperature", "Humidity(%)"]
plot_simple_scatter(data1, "Datetime", features)
```





Explanation:

What first strikes your eye when looking at the scatter plot are many data points with valu e -200. For all features this is an implausible value. Probably it indicates a measurement error.

For Temperature and Humidty the amount of -200 values is relatively low. The series of plau sible data points look more or less continuous

sible data points look more or less continuous.
For the other three features there are several periods of time where the stream of plausible data points is broken, probably due to missing or erronous (-200) values.

(d) We need to handle any implausible or missing data. In the lecture, several strategies to do so have been introduced, for example, deleting data rows that contain missing/implausible values or replacing them by a value derived from other data points.

In this question, consider implausible values to be the ones identified in question (c).

1) For all numerical features, compute and show mean, standard deviation, minimum and maximum, while ignoring the missing and implausible values. Also, print the total number of data rows.

2) Choose a strategy (or combination of strategies) to handle missing/implausible values. Create a cleaned data set with all those values handled accordingly.

3) For all numerical features, compute and show mean, standard deviation, minimum and maximum with respect to your cleaned data set. Also print the total number of data rows.

4) Motivate and explain your strategy and describe its (dis-)advantages compared to other options. Compare the computed statistical values before and after cleaning and briefly describe and evaluate any changes.

In [34]:

```
benzeneArr= []
coArr = []
no2Arr = []
humidityArr = []
temperatureArr = []

features = ['Benzene', 'CO', 'NO2', 'Humidity(%)', 'Temperature']
ignore = {}

for f in features:
    ignore[f] = []

for i in range(0,len(data1)):
    for f in features:
        if (data1[f][i] != -200):
            ignore[f].append(data1[f][i])

stats1 = pd.DataFrame(columns=['mean', 'std', 'min', 'max'], index=features)

for f in features:
    stats1['mean'][f] = np.mean(ignore[f])
    stats1['std'][f] = np.std(ignore[f])
    stats1['min'][f] = np.min(ignore[f])
    stats1['max'][f] = np.max(ignore[f])

print('1) Ignore missing/implausible values')
display(stats1)
print('Number of rows: {0}'.format(len(data1.index)))
```

1) Ignore missing/implausible values

	mean	std	min	max
Benzene	10.0789	7.48775	0.1	63.7
CO	2.14748	1.45581	0.1	11.9
NO2	112.855	48.3155	2	340
Humidity(%)	49.3458	17.2805	9.2	88.7
Temperature	18.3111	8.79916	-1.9	44.6

Number of rows: 8421

In [40]:

```
mask = (data1['CO'] >= 0) & (data1['Benzene'] >= 0) & (data1["NO2"] >= 0) & (data1["Temperature"] >
-40) & (data1["Humidity(%)"] >= 0)
cleaned1 = data1.loc[mask]

stats2 = pd.DataFrame(columns=['mean', 'std', 'min', 'max'], index=features)
stats2['mean']= cleaned1[features].mean()
stats2['std']= cleaned1[features].std()
stats2['min']= cleaned1[features].min()
stats2['max']= cleaned1[features].max()

print('3) Cleaned Dataset: Rows with implausible entries got deleted')
display(stats2)
print('Number of rows: {0}'.format(len(cleaned1.index)))
```

```
3) Cleaned Dataset: Rows with implausible entries got deleted
```

	mean	std	min	max
Benzene	10.545222	7.493666	0.2	63.7
CO	2.178169	1.443180	0.1	11.9
NO2	113.692246	47.391559	2.0	333.0
Humidity(%)	48.967728	17.388378	9.2	88.7
Temperature	17.751609	8.822277	-1.9	44.6

```
Number of rows: 6216
```

Explanation:

Our strategy was to delete rows with implausible/missing entries. This is a very easy strategy. But as we can see now, this deleted roughly 1/4 of the dataset. Chances are that we created a bias in the data because of all the values that are missing now.

Filling in missing data manually would have been too much of an effort, because of the high number of missing values. Using a mean or median for missing values would have been inappropriate, because we saw in the scatter plots, that the data is not normally distributed.

Probably it would have been appropriate to interpolate the values using some prediction technique (Linear Regression?).

Question 2 - Data Preprocessing and Advanced Visualization (15 points)

For this question, use the extracted data set you created in the preprocessing step ('air_pollution_2.csv').

(a) To create a suitable input for the following questions, modify the data set as listed below and then print the first 5 data rows:

- Remove the columns 'Traffic_Volume' and 'Particulate_Matter'
- Replace the columns 'Date' and 'Time' by a combined column 'Datetime' using the datetime data type
- Restrict the data to the timeframe between 2004-04-01 and 2005-03-31
- Drop all rows that contain a missing value or a value below -37

Hint: You may be able to reuse some of your code or data sets created above.

In [47]:

```
import pandas as pd
import math
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns

data2 = pd.read_csv("air_pollution_2.csv")

data2 = data2.drop(columns=["Traffic_Volume", "Particulate_Matter"])
data2["Datetime"] = [pd.to_datetime(row["Date"] + " " + row["Time"], dayfirst=True) for index, row
in data2.iterrows()]
data2 = data2.drop(columns=["Date", "Time"])
mask = (data2["Datetime"] >= "2004-04-01") & (data2["Datetime"] <= "2005-03-31")
data2 = data2[mask]
mask = (data2['CO'] >= 0) & (data2['Benzene'] >= 0) & (data2["NO2"] >= 0) & (data2["Temperature"]
>= -37) & (data2["Humidity(%)"] >= 0)
data2 = data2[mask]
data2 = data2.dropna()

data2.head()
```

Out[47]:

	CO	Benzene	NO2	Temperature	Humidity(%)	Datetime
444	0.7	2.4	49	10.2	66.6	2004-04-01 04:00:00
445	0.9	2.9	50	11.0	63.7	2004-04-01 05:00:00
446	1.1	4.1	55	10.7	67.2	2004-04-01 02:00:00
447	1.2	5.1	70	11.5	63.9	2004-04-01 01:00:00
448	1.6	6.8	88	19.7	38.4	2004-04-01 11:00:00

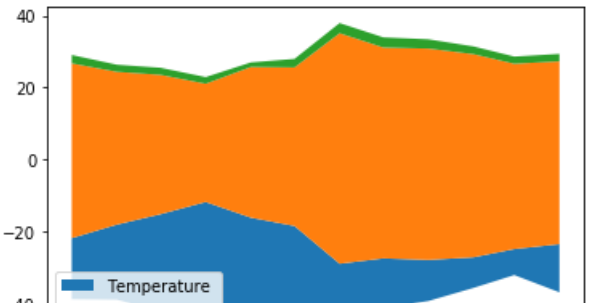
(b) For temperature, humidity and CO compute the mean value for each month. Create two stream graphs based on this newly computed data, that visualize the change in temperature, CO and humidity over the months:

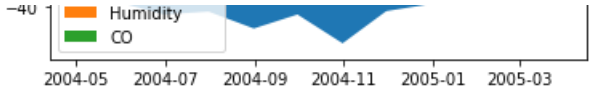
1) For the first stream graph, simply use the mean values you computed for each month.

2) For the second stream graph, first normalize the computed mean values by mapping them in dividually to an interval between 0 and 1, that is, apply Min-max normalization.

In [48]:

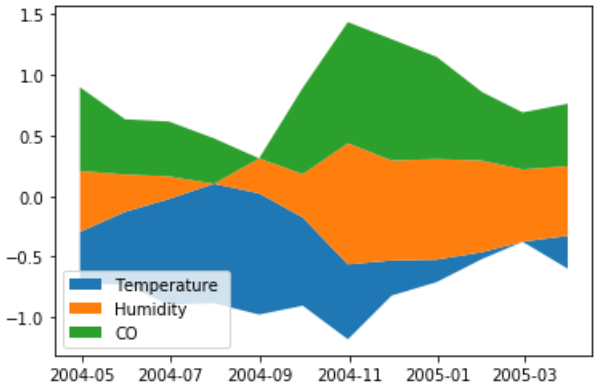
```
monthly_mean = data2.resample("M", on="Datetime").mean()

# the values for our x-axis
x = monthly_mean.index
# the values that will be stacked on top of each other
y1 = monthly_mean["Temperature"]
y2 = monthly_mean["Humidity(%)"]
y3 = monthly_mean["CO"]

# the labels for y1, y2 and y3
labels = ["Temperature ", "Humidity", "CO"]

#stacking our values vertically
y = np.vstack([y1, y2, y3])

fig, ax = plt.subplots()
#modifying the axis
ax.stackplot(x, y1, y2, y3, labels=labels, baseline="wiggle")
ax.legend(loc='lower left')
plt.show()
```





In [49]:

```
from sklearn import preprocessing

x = monthly_mean.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
monthly_mean_norm = pd.DataFrame(x_scaled, index=monthly_mean.index, columns=monthly_mean.columns)

# the values for our x-axis
x = monthly_mean_norm.index
# the values that will be stacked on top of each other
y1 = monthly_mean_norm["Temperature"]
y2 = monthly_mean_norm["Humidity(%)"]
y3 = monthly_mean_norm["CO"]

# the labels for y1, y2 and y3
labels = ["Temperature ", "Humidity", "CO"]

#stacking our values vertically
y = np.vstack([y1, y2, y3])

fig, ax = plt.subplots()
#modifying the axis
ax.stackplot(x, y1, y2, y3, labels=labels, baseline="wiggle")
ax.legend(loc='lower left')
plt.show()
```



(c) Answer the following questions based on the stream graphs created in question 2(a) and briefly explain how you derived the answer. Which of the two stream graphs do you consider more adequate to obtain the answers?

1) In which month is the average CO lowest?

2) Is there any correlation between humidity and CO?

3) Between which months is temperature increasing?

Explanation:

1) Average CO is lowest in September 2004.

2) There is a correlation between humidity and CO. They reach their peak in the same month (2004-11) and also their minima occur to similar times (2004-08 - 2004-09). Apart from that also the measurements follow a similar relative trend.

3) Temperature rises from 2004-04 on till 2004-08 and again between 2005-02 and 2005-03

We used the graph of the normalized means. The normalization makes for comparable graph sizes that which allows for easier detection of correlation and (local) minima and maxima

(d) We want to create a heat map that visualizes the CO measured for different combinations of humidity and temperature. The heatmap should have 12 columns and 12 rows. The shown CO value should be the *median* of all values for the combination of humidity and temperature.

Modify the data as needed. Use a binning strategy of your choice for converting numerical data to categorical data. Motivate and explain all your choices and modifications.

In [51]:

```
discretizer = preprocessing.KBinsDiscretizer(n_bins=[12,12], encode='ordinal', strategy = 'quantile
')
discretizer.fit(data2[["Humidity(%)", "Temperature"]])
binned = discretizer.transform(data2[["Humidity(%)", "Temperature"]])
bins = discretizer.bin_edges_

data2_binned = data2.copy()
data2_binned["Humidity(%)"] = [b[0] for b in binned]
data2_binned["Temperature"] = [b[1] for b in binned]

table = pd.pivot_table(data2_binned, values='CO', index='Humidity(%)', columns='Temperature', aggfu
nc=np.median)
table = table.rename(index= lambda x: "{:.1f} - {:.1f}".format(bins[0][int(x)], bins[0][int(x+1)]))
table = table.rename(columns= lambda x: "{:.1f} - {:.1f}".format(bins[1][int(x)], bins[1][int(x+1)])
)
```

(e) Use the modified data to create a heat map as specified in question part (d). Answer the following questions based on that heat map and briefly explain how you derived your answer:

1) **For which combination of humidity and temperature values is the median CO highest?**

2) **How do you explain empty fields in your heat map?**

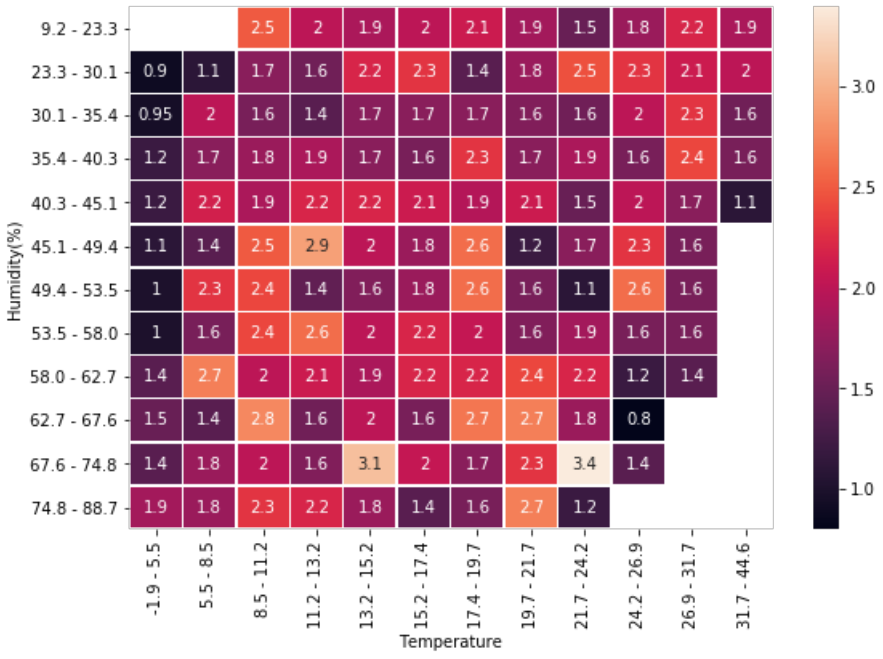
3) **Is CO correlating with temperature, humidity, or both?**

In [52]:

```
f, ax = plt.subplots(figsize=(9, 6))
sns.heatmap(table, annot=True, linewidths=.5, ax=ax)
```

Out[52]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb968a71cd0>
```



Explanation:

Explanation:

1) Median CO is highest for Humidity in range (67.6-74.8) and Temperature in range (21.7-24.2). This is determined by looking for the brightest field in the heatmap (highest annotation value).

2) These are fields for which no data points exist, e.g. there was no measurement taken with this combination of temperature and humidity.

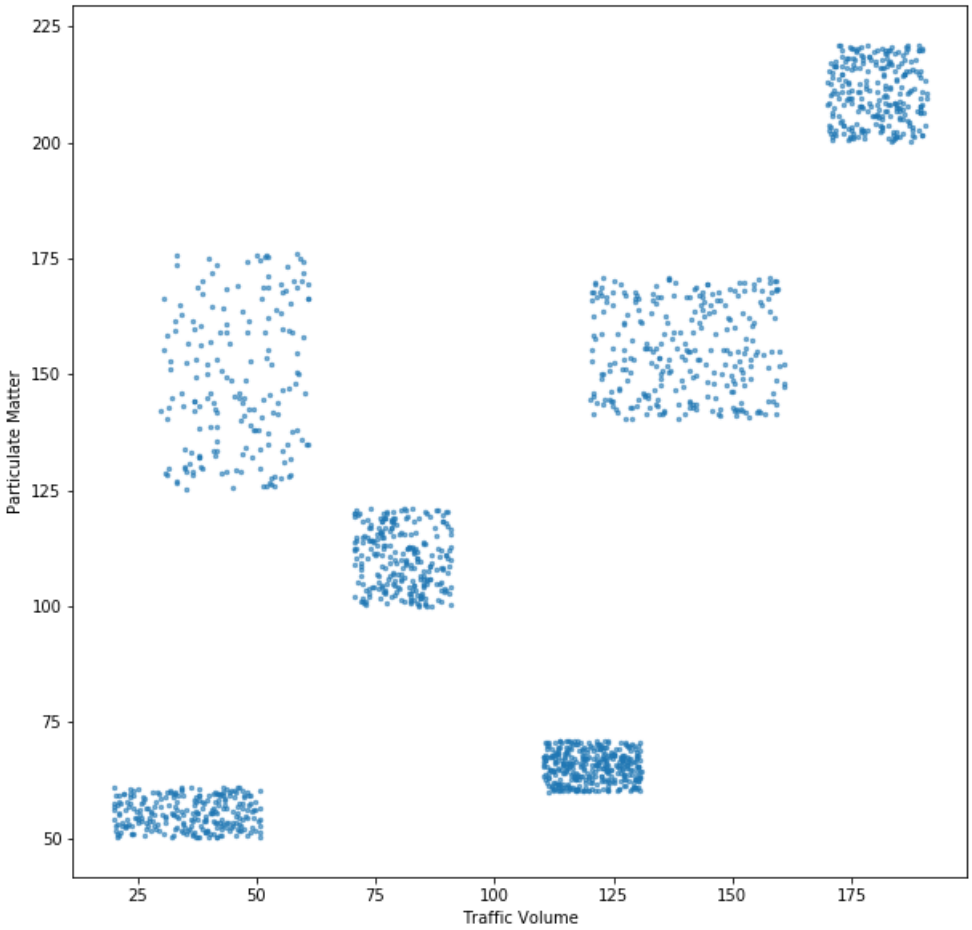
3) No, the heatmap shows no such correlation. If there was a correlation, the heatmap would show a kind of color gradient in some direction(s).

Question 3 - Clustering (15 points):

(a) For this question, use the extracted data set you created in the preprocessing step ('air_pollution_2.csv'). Use a scatter diagram to find the relation between the 'Traffic_Volume' and 'Particulate_Matter' columns.

In [55]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN

data3 = pd.read_csv('air_pollution_2.csv')

plt.figure(figsize=(10,10))
plt.scatter(data3['Traffic_Volume'], data3['Particulate_Matter'], s=6, alpha=0.6)
plt.xlabel('Traffic Volume')
plt.ylabel('Particulate Matter')
plt.show()
```



(b) From the previous question you found the relationship between two columns which are suitable for applying clustering methods. Find the two arrays corresponding to these to columns and drop nan values from them. Apply the k_means

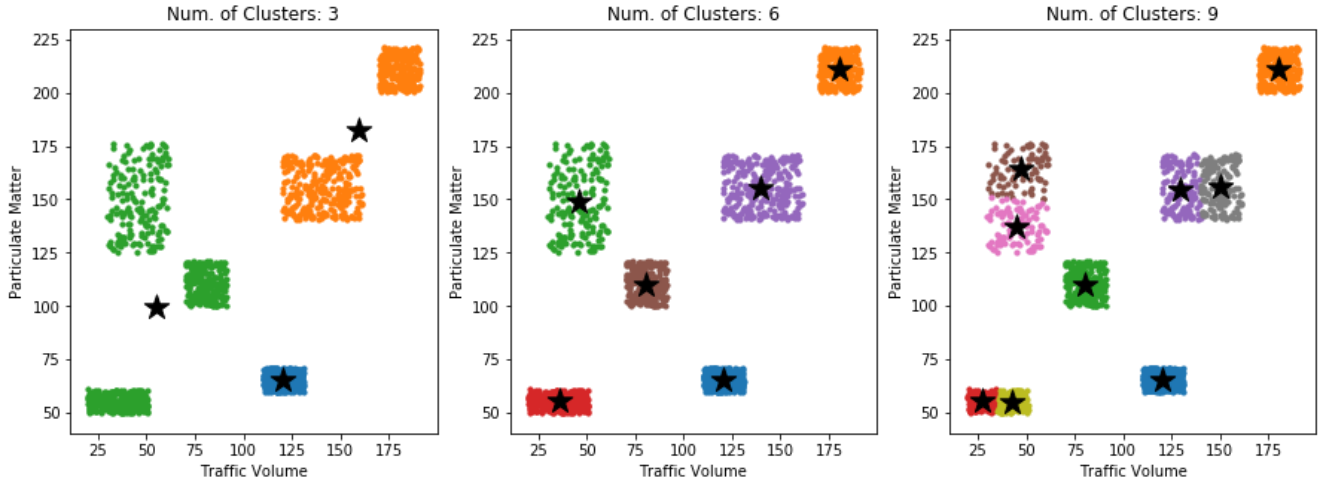
method for clustering the two mentioned columns. Use three different number of clusters for classifying.

In [56]:

```
columns = zip(data3['Traffic_Volume'].values, data3['Particulate_Matter'].values)
arr = np.array(list(filter(lambda x: not np.isnan(x[0]) and not np.isnan(x[1]), columns)))
```

In [57]:

```
cluster_nums = [3, 6, 9]

def plot_k_means(data, cluster_nums):
    plt.figure(figsize = (15,5))

    for i, cn in enumerate(cluster_nums):
        subnum = 100 + len(cluster_nums) * 10 + i+1
        title = "Num. of Clusters: {:d}".format(cn)
        plots = plt.subplot(subnum, title=title)
        kmeans = KMeans(cn).fit(data)
        clusters = kmeans.labels_
        centroids = kmeans.cluster_centers_

        for cluster_index in range(cn) :
            sub_set = np.array([data[i] for i in range(len(data)) if clusters[i] == cluster_index])
            if len(sub_set) == 0 :
                continue
            plots.scatter(sub_set[:,0], sub_set[:,1], s = 10, c = "C{:d}".format(cluster_index))
        plots.scatter(centroids[:,0], centroids[:,1], marker = '*', s = 300, c = 'k')
        plt.xlabel('Traffic Volume')
        plt.ylabel('Particulate Matter')

    plt.show()

plot_k_means(arr, cluster_nums)
```

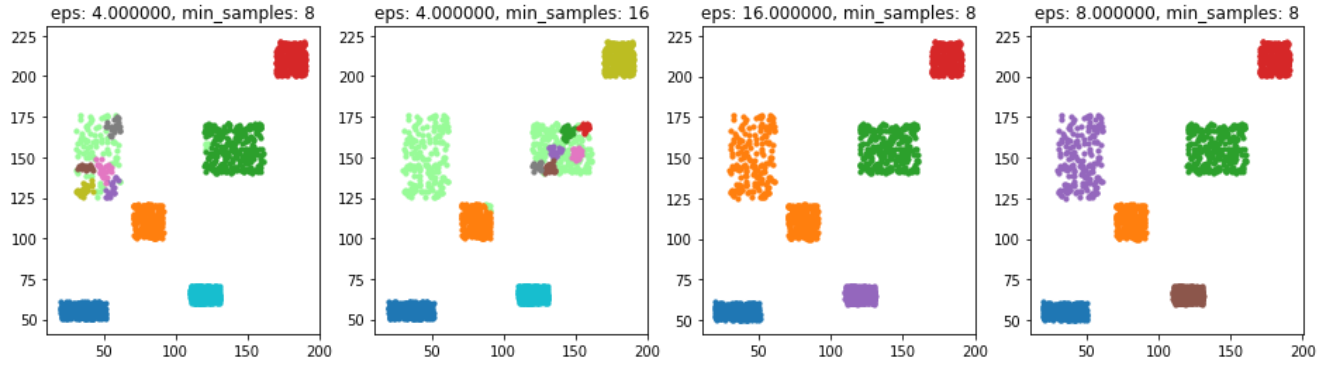


(c) Apply the DBSCAN method for clustering the two mentioned columns. Use three different eps and min_samples for clustering.

In [58]:

```
eps_samples= [[4,8], [4,16], [16,8], [8,8]]

def plot_dbscan(data, eps_samples):
    plt.figure(figsize = (16,4))

    for i, es in enumerate(eps_samples):
        subnum = 100 + len(eps_samples) * 10 + i+1
        title = "eps: {:f}, min_samples: {:d}".format(es[0], es[1])
        plots = plt.subplot(subnum, title=title)

        dbscan = DBSCAN(eps = es[0], min_samples=es[1]).fit(data)
        clusters = dbscan.labels_
        cluster_indexs = np.unique(clusters).tolist()
```

```
        for cluster_index in cluster_indexs :
            sub_set = np.array([data[i] for i in range(len(data)) if clusters[i] == cluster_index])
            if len(sub_set) == 0 :
                continue
            if cluster_index < 0:
                color = 'palegreen'
            else:
                color = "C{:d}".format(cluster_index)
            plots.scatter(sub_set[:,0], sub_set[:,1], s = 10, c = color)

    plt.show()

plot_dbscan(arr, eps_samples)
```



(d) Compare k_means and DBSCAN results. Which of these methods is more suitable for clustering this data? Why?

Explanation:

k_means delivers easy and good results on this data. Even when the number of clusters is set incorrectly, the found clusters seem logic (cluster_num too small: Several "real clusters" get grouped together, cluster_num too high: "real clusters" get split up.

DBSCAN initially has more problems to find cluster the data correctly. Parameters cannot directly be found by looking at a visual representation of the data. Wrongly set parameters split or merge clusters where it is not appropriate. Also data that belongs to a cluster gets wrongly labeled as noise. But with a bit of trial and error it is possible to find good parameters for a correct clustering.

k_means is more suitable for this data. On one hand the number of clusters is clear to see, which makes it easy to tune the algorithm to find the right clusters. On the other hand there are zero outliers, which could present an obstacle for the k_means algorithm.

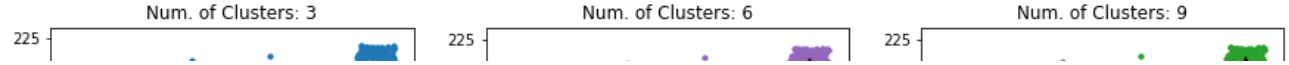
(e) Add noise to the two mentioned columns. For adding noise, first find the range of these two columns and add a random number of 30 samples of noise in the range of each column to that. Plot the scatter diagram again. Repeat k_means and DBSCAN algorithms for clustering. Which of these methods is more suitable for clustering this data? Why?

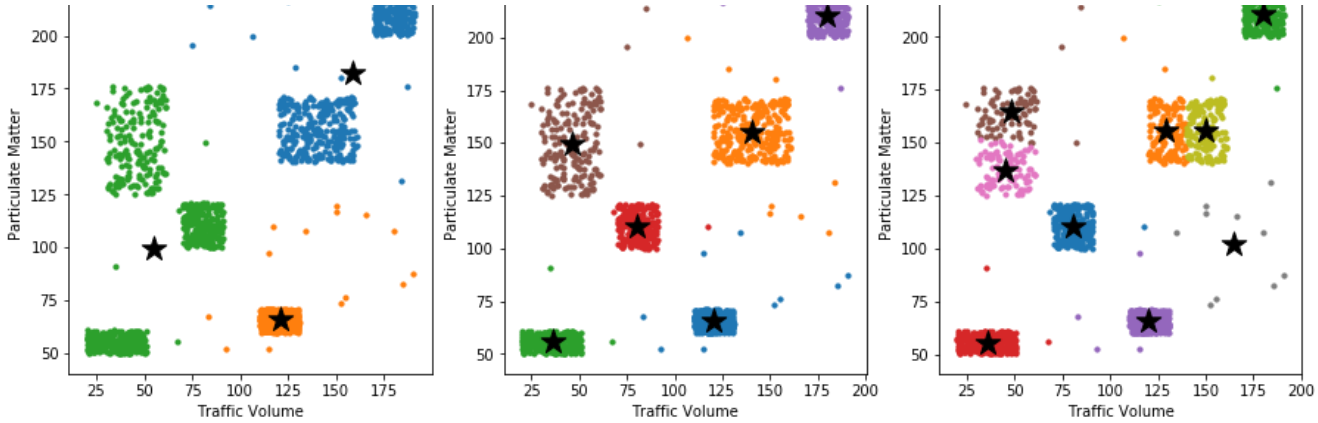
In [61]:

```
rng = np.random.default_rng()
randoms = rng.random((30,2))
arr_min = arr.min(0)
arr_max = arr.max(0)
noise = randoms * (arr_max - arr_min) + arr_min

arr2 = np.concatenate((arr, noise))
```

In [62]:

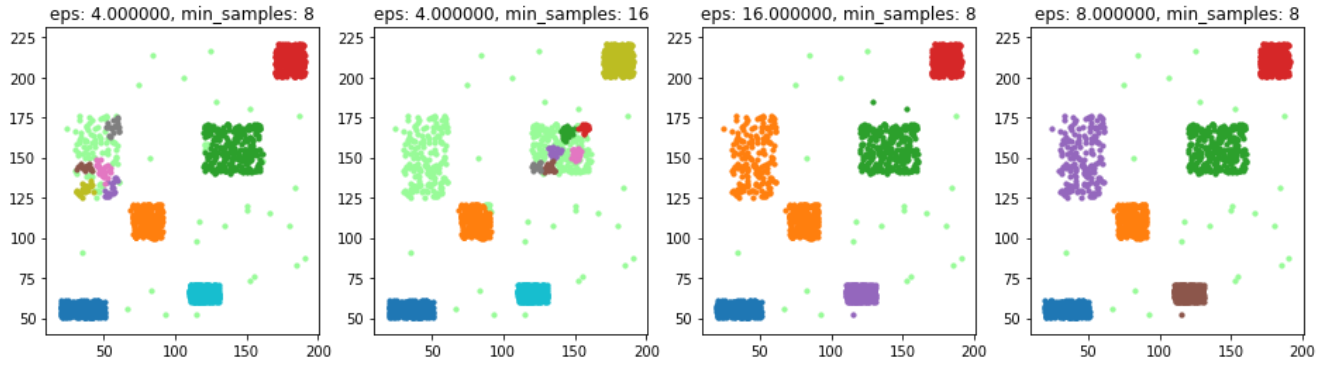
```
cluster_nums = [3, 6, 9]
plot_k_means(arr2, cluster_nums)
```





In [63]:

```
eps_samples= [[4,8], [4,16], [16,8], [8,8]]
plot_dbscan(arr2, eps_samples)
```



Explanation:

Now with noise added, DBSCAN is more suitable for clustering. It is able to detect outliers and mark them as noise, whereas k_means assigns every data point to a cluster, which is not always reasonable.

Question 4 - Frequent itemsets and association rules (15 points):

(a) You should carry out some preprocessing steps before starting the analysis:

- **Select 90% of 'applications' dataset by random sampling. Use one of the group member's student number as a seed.**
- **After completing this preprocessing step, export your final dataset as 'applications_2.csv' dataset and use that for the next steps of the assignment.**
- **Important! Make sure that you submit your extracted dataset with your result in moodle.**

In [65]:

```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules as arule

seed = 344493

csv4 = pd.read_csv("applications.csv", header=None, low_memory=False)

csv4 = csv4.sample(frac=0.9, random_state=seed).sort_index()
csv4.to_csv("applications_2.csv", header=None, index=None)
csv4.head()
```

Out[65]:

	0	1	2	3	4	5	6	
0	A_SUBMITTED	A_PARTLYSUBMITTED	A_PREACCEPTED	W_Completeren aanvraag	W_Completeren aanvraag	A_ACCEPTED	O_SELECTED	A_F
1	A_SUBMITTED	A_PARTLYSUBMITTED	A_PREACCEPTED	W_Completeren aanvraag	W_Completeren aanvraag	W_Completeren aanvraag	W_Completeren aanvraag	A_A
2	A_SUBMITTED	A_PARTLYSUBMITTED	A_PREACCEPTED	W_Completeren aanvraag	W_Completeren aanvraag	W_Completeren aanvraag	W_Completeren aanvraag	W_Co
3	A_SUBMITTED	A_PARTLYSUBMITTED	A_DECLINED	NaN	NaN	NaN	NaN	
4	A_SUBMITTED	A_PARTLYSUBMITTED	A_DECLINED	NaN	NaN	NaN	NaN	

5 rows × 175 columns

(b) Find the most frequent itemsets with the support of more than 0.4 by using the Apriori algorithm (Hint: When you are creating rows to make the data_set, you should use none_empty strings).

In [67]:

```
raw_data4 = list([[item for item in x if not pd.isnull(item)] for x in csv4.values])
```

In [68]:

```
te = TransactionEncoder()
te_ary = te.fit(raw_data4).transform(raw_data4)
data4 = pd.DataFrame(te_ary, columns = te.columns_)

data4.head()
```

Out[68]:

	A_ACCEPTED	A_ACTIVATED	A_APPROVED	A_CANCELLED	A_DECLINED	A_FINALIZED	A_PARTLYSUBMITTED	A_PREACCEPTE
0	True	True	True	False	False	True	True	Tru
1	True	True	True	False	False	True	True	Tru
2	True	True	True	False	False	True	True	Tru
3	False	False	False	False	True	False	True	Fals
4	False	False	False	False	True	False	True	Fals

5 rows × 24 columns

In [69]:

```
frequent_itemsets = apriori(data4, min_support = 0.4, use_colnames = True)
frequent_itemsets.sort_values(by=['support'], ascending=False)
```

Out[69]:

	support	itemsets
3	1.000000	(A_SUBMITTED)
8	1.000000	(A_PARTLYSUBMITTED, A_SUBMITTED)
1	1.000000	(A_PARTLYSUBMITTED)
0	0.584904	(A_DECLINED)
5	0.584904	(A_PARTLYSUBMITTED, A_DECLINED)
6	0.584904	(A_SUBMITTED, A_DECLINED)
13	0.584904	(A_PARTLYSUBMITTED, A_SUBMITTED, A_DECLINED)
12	0.562404	(W_Completeren aanvraag, A_SUBMITTED)
17	0.562404	(W_Completeren aanvraag, A_SUBMITTED, A_PREACC...
16	0.562404	(W_Completeren aanvraag, A_SUBMITTED, A_PARTLY...
15	0.562404	(A_PARTLYSUBMITTED, W_Completeren aanvraag, A_...

	support	itemsets
14	0.562404	(A_PARTLYSUBMITTED, A_SUBMITTED, A_PREACCEPTED)
9	0.562404	(A_PARTLYSUBMITTED, W_Completeren aanvraag)
11	0.562404	(W_Completeren aanvraag, A_PREACCEPTED)
10	0.562404	(A_SUBMITTED, A_PREACCEPTED)
7	0.562404	(A_PARTLYSUBMITTED, A_PREACCEPTED)
4	0.562404	(W_Completeren aanvraag)
2	0.562404	(A_PREACCEPTED)
18	0.562404	(W_Completeren aanvraag, A_SUBMITTED, A_PREACC...

(c) Find the most frequent itemsets with the support of more than 0.4 by using the Apriori algorithm having more than 2 members.

In [70]:

```
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets[ (frequent_itemsets['length'] > 2) ]
```

Out[70]:

	support	itemsets	length
13	0.584904	(A_PARTLYSUBMITTED, A_SUBMITTED, A_DECLINED)	3
14	0.562404	(A_PARTLYSUBMITTED, A_SUBMITTED, A_PREACCEPTED)	3
15	0.562404	(A_PARTLYSUBMITTED, W_Completeren aanvraag, A_...	3
16	0.562404	(W_Completeren aanvraag, A_SUBMITTED, A_PARTLY...	3
17	0.562404	(W_Completeren aanvraag, A_SUBMITTED, A_PREACC...	3
18	0.562404	(W_Completeren aanvraag, A_SUBMITTED, A_PREACC...	4

(d) Find the itemsets having min_confidence=0.3 and min_lift=1.2. Print support, confidence and lift of filtered rules in one table.

In [73]:

```
rules = arule(frequent_itemsets)
rules[ (rules['confidence'] >= 0.3) & (rules['lift'] >= 1.2) ][['antecedents', 'consequents', 'supp
ort', 'confidence', 'lift']]
```

Out[73]:

	antecedents	consequents	support	confidence	lift
7	(W_Completeren aanvraag)	(A_PREACCEPTED)	0.562404	1.0	1.77808
8	(A_PREACCEPTED)	(W_Completeren aanvraag)	0.562404	1.0	1.77808
16	(A_PARTLYSUBMITTED, W_Completeren aanvraag)	(A_PREACCEPTED)	0.562404	1.0	1.77808
17	(A_PARTLYSUBMITTED, A_PREACCEPTED)	(W_Completeren aanvraag)	0.562404	1.0	1.77808
19	(W_Completeren aanvraag)	(A_PARTLYSUBMITTED, A_PREACCEPTED)	0.562404	1.0	1.77808
20	(A_PREACCEPTED)	(A_PARTLYSUBMITTED, W_Completeren aanvraag)	0.562404	1.0	1.77808
24	(W_Completeren aanvraag, A_SUBMITTED)	(A_PREACCEPTED)	0.562404	1.0	1.77808
26	(A_SUBMITTED, A_PREACCEPTED)	(W_Completeren aanvraag)	0.562404	1.0	1.77808
27	(W_Completeren aanvraag)	(A_SUBMITTED, A_PREACCEPTED)	0.562404	1.0	1.77808
28	(A_PREACCEPTED)	(W_Completeren aanvraag, A_SUBMITTED)	0.562404	1.0	1.77808
30	(A_PARTLYSUBMITTED, W_Completeren aanvraag, A_...	(A_PREACCEPTED)	0.562404	1.0	1.77808
32	(A_PARTLYSUBMITTED, A_SUBMITTED, A_PREACCEPTED)	(W_Completeren aanvraag)	0.562404	1.0	1.77808
33	(W_Completeren aanvraag, A_SUBMITTED)	(A_PARTLYSUBMITTED, A_PREACCEPTED)	0.562404	1.0	1.77808

	antecedents	consequents	support	confidence	lift
35	(A_PARTLYSUBMITTED, W_Completeren aanvraag)	(A_SUBMITTED, A_PREACCEPTED)	0.562404	1.0	1.77808
36	(A_SUBMITTED, A_PREACCEPTED)	(A_PARTLYSUBMITTED, W_Completeren aanvraag)	0.562404	1.0	1.77808
37	(A_PARTLYSUBMITTED, A_PREACCEPTED)	(W_Completeren aanvraag, A_SUBMITTED)	0.562404	1.0	1.77808
38	(W_Completeren aanvraag)	(A_PARTLYSUBMITTED, A_SUBMITTED, A_PREACCEPTED)	0.562404	1.0	1.77808
39	(A_PREACCEPTED)	(A_PARTLYSUBMITTED, W_Completeren aanvraag, A_...	0.562404	1.0	1.77808

Question 5 - Text Mining (15 points)

Among the datasets given for this assignment you will find the files "pg_train" and "pg_test". These two files contain a labeled corpus, already splitted in training and test set. The corpus consists of sentences from several novels, labeled with the name of the author. The first task is text classification: you will train a set of classifiers that predict the author of a piece of text.

(a) Perform preprocessing on the training corpus (all lowercase, no punctuation, tokenization, stemming, stopword removal) and obtain a binary document-term matrix; train a logistic classifier with the author as target.

In [76]:

```
import pandas as pd

train = pd.read_csv("pg_train.csv", sep='#', header=None, encoding='ISO-8859-1', names=['author', '
text'])
test = pd.read_csv("pg_test.csv", sep='#', header=None, encoding='ISO-8859-1', names=['author', 'te
xt'])
train.head()
```

Out[76]:

	author	text
0	austen	[Emma by Jane Austen 1816] VOLUME I CHAPTER ...
1	austen	Sixteen years had Miss Taylor been in Mr . Woo...
2	austen	It was Miss Taylor ' s loss which first brough...
3	austen	The want of Miss Taylor would be felt every ho...
4	austen	He could not meet her in conversation , ration...

In [77]:

```
import nltk
from sklearn.base import BaseEstimator, TransformerMixin
from nltk.stem.snowball import SnowballStemmer, PorterStemmer

nltk.download('stopwords')

class Stemmer(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.stemmer = SnowballStemmer('english', ignore_stopwords=True)

    def fit(self, X, y=None):
        return self

    def transform(self, X, *_):
        new_corpus=[' '.join([self.stemmer.stem(word) for word in text.split(' ')]) for text in X]
        return new_corpus
```

```
[nltk_data] Downloading package stopwords to /home/serjo/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [78]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier
```

```
from sklearn.linear_model import SGDClassifier

from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

#Stemmer performs all lowercase and stemming, CountVectorizer does tokenization, no punctuation an
d stopword removal
bindocterm_clf = Pipeline([('stem', Stemmer()),('vect', CountVectorizer(stop_words='english', binar
y=True)), ('clf-svm', SGDClassifier(loss='log'))])
bindocterm_clf = bindocterm_clf.fit(train['text'], train['author'])
```

(b) Perform preprocessing on the training corpus (all lowercase, no punctuation, tokenization, stemming, stopword removal) and obtain a document-term matrix of counts; train a logistic classifier with the author as target.

In [79]:

```
#Stemmer performs all lowercase and stemming, CountVectorizer does tokenization, no punctuation an
d stopword removal
docterm_clf = Pipeline([('stem', Stemmer()),('vect', CountVectorizer(stop_words='english')), ('clf-
svm', SGDClassifier(loss='log'))])
docterm_clf = docterm_clf.fit(train['text'], train['author'])
```

(c) Perform preprocessing on the training corpus (all lowercase, no punctuation, tokenization, stemming, stopword removal) and obtain a tf-idf scores document-term matrix; train a logistic classifier.

In [80]:

```
from sklearn.feature_extraction.text import TfidfTransformer

#Stemmer performs all lowercase and stemming, CountVectorizer does tokenization, no punctuation an
d stopword removal
tfidf_clf = Pipeline([('stem', Stemmer()),
                      ('vect', CountVectorizer(stop_words='english')),
                      ('tf-idf', TfidfTransformer()),
                      ('clf-svm', SGDClassifier(loss='log'))])

tfidf_clf = tfidf_clf.fit(train['text'], train['author'])
```

(d) Perform preprocessing on the training corpus (all lowercase, no punctuation, tokenization, stemming, stopword removal) and obtain a doc2vec embedding in order to reduce the dimension of the document vector to 300; use the doc2vec model you just trained to convert the training set to a set of document vectors; train a logistic classifier with the author as target.

In [81]:

```
import gensim
from gensim.parsing.preprocessing import remove_stopwords
stemmer = SnowballStemmer('english', ignore_stopwords=True)

#simple_preprocess does no punctuation, tokenization, stemmer does all lowercase and stemming
def preprocess_text(text):
    return gensim.utils.simple_preprocess(remove_stopwords(' '.join([stemmer.stem(word) for word in
text.split(' ')])))

train_docs = [preprocess_text(text) for text in train['text']]
test_docs = [preprocess_text(text) for text in test['text']]
```

In [82]:

```
train_tagged = []
test_tagged = []

for i, t in enumerate(train_docs):
    train_tagged.append(gensim.models.doc2vec.TaggedDocument(words=t, tags=[train['author'][i]]))

for i, t in enumerate(test_docs):
    test_tagged.append(gensim.models.doc2vec.TaggedDocument(words=t, tags=[test['author'][i]]))
```

In [83]:

```
# Building the vocabulary
from gensim.models import Doc2Vec
import multiprocessing

cores = multiprocessing.cpu_count()

doc2vec_model = Doc2Vec(dm=0, vector_size=300, min_count=2, epochs=40, workers=cores)
doc2vec_model.build_vocab(train_tagged)
```

In [84]:

```
doc2vec_model.train(train_tagged, total_examples=doc2vec_model.corpus_count, epochs=doc2vec_model.e
pochs)
```

In [85]:

```
# Building the feature vector for the classifier
def vec_for_learning(model, docs):
    doc2vec_vectors = [model.infer_vector(doc.words) for doc in docs]
    targets = [doc.tags[0] for doc in docs]
    return doc2vec_vectors, targets
```

In [86]:

```
# Translating docs into vectors for training and test set
X_train, y_train = vec_for_learning(doc2vec_model, train_tagged)
X_test, y_test = vec_for_learning(doc2vec_model, test_tagged)
```

In [92]:

```
# Training a classification model
from sklearn.linear_model import LogisticRegression

doc2vec_clf Perform preprocessing on the training corpus (all lowercase, no punctuation, tokenizati
on, stemming, stopword removal) and obtain a doc2vec embedding in order to reduce the dimension of
the document vector to 300; use the doc2vec model you just trained to convert the training set to a
set of document vectors; train a logistic cl= SGDClassifier(loss="log")
doc2vec_clf.fit(X_train, y_train)
```

Out[92]:

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=1000,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

(e) Predict the classification with the four models on the test data.

In [96]:

```
# Your code

bindocterm_pred = bindocterm_clf.predict(test['text'])
docterm_pred = docterm_clf.predict(test['text'])
tfidf_pred = tfidf_clf.predict(test['text'])
doc2vec_pred = doc2vec_clf.predict(X_test)
```

(f) Obtain confusion matrices for the four different models.

In [97]:

```
# Your code
import numpy as np
true_y = np.array(test['author'])

print('Binary Docterm')
display(pd.crosstab(true_y, np.array(bindocterm_pred), rownames=['True'], colnames=['Predicted'], m
```

```
argins=True))
print('Docterm')
display(pd.crosstab(true_y, np.array(docterm_pred), rownames=['True'], colnames=['Predicted'], marg
ins=True))
print('tf-idf')
display(pd.crosstab(true_y, np.array(tfidf_pred), rownames=['True'], colnames=['Predicted'], margin
s=True))
print('doc2vec')
display(pd.crosstab(true_y, np.array(doc2vec_pred), rownames=['True'], colnames=['Predicted'], marg
ins=True))
```

```
Binary Docterm
```

Predicted True	austen	chesterton	shakespeare	All
austen	680	6	3	689
chesterton	16	496	3	515
shakespeare	0	0	301	301
All	696	502	307	1505

```
Docterm
```

Predicted True	austen	chesterton	shakespeare	All
austen	679	6	4	689
chesterton	22	490	3	515
shakespeare	0	0	301	301
All	701	496	308	1505

```
tf-idf
```

Predicted True	austen	chesterton	shakespeare	All
austen	682	6	1	689
chesterton	10	505	0	515
shakespeare	0	2	299	301
All	692	513	300	1505

```
doc2vec
```

Predicted True	austen	chesterton	shakespeare	All
austen	687	2	0	689
chesterton	19	494	2	515
shakespeare	1	0	300	301
All	707	496	302	1505

(g) Obtain accuracy and f1-score for the four different models.

In [98]:

```
# Classification performance metrics
from sklearn.metrics import accuracy_score, f1_score

def score_pred(pred):
```

```
    return [accuracy_score(test['author'], pred), f1_score(test['author'], pred, average='weighted'
)]

scores = pd.DataFrame({
    "Binary Docterm": score_pred(bindocterm_pred),
    "Docterm": score_pred(docterm_pred),
    "tf-idf": score_pred(tfidf_pred),
    "doc2vec": score_pred(doc2vec_pred)
}, index=['Accuracy', 'F1'])

scores
```

Out[98]:

	Binary Docterm	Docterm	tf-idf	doc2vec
Accuracy	0.981395	0.976744	0.987375	0.984053
F1	0.981351	0.976669	0.987374	0.984005

(h) Briefly comment on the quality of the predictions for the four models.

Explanation: All 4 models show comparabel, high performance. Interestingly Binary Docterm performs better than normal Docterm. tf-idf delivers the best result, close to 99% accuracy, followed by doc2vec.

(i) For the two authors Austen and Chesterton separately, build a bigram language model. You should use both training and test data to build the model. Do not perform stemming or stopword removal for this task, but do use the other preprocessing steps described in (a). Use both right and left padding, and manage unknown terms by using a dedicated token.

In [99]:

```
from nltk.lm.preprocessing import pad_both_ends
from nltk.util import ngrams
from nltk.lm import Vocabulary

combined = pd.concat([train, test])
austen_texts = combined[combined['author'] == 'austen']['text']
chesterton_texts = combined[combined['author'] == 'chesterton']['text']

austen_preprocessed = [ gensim.utils.simple_preprocess(text) for text in austen_texts ]
chesterton_preprocessed = [ gensim.utils.simple_preprocess(text) for text in chesterton_texts ]

austen_padded2 = [ list(pad_both_ends(text, n=2)) for text in austen_preprocessed ]
chesterton_padded2 = [ list(pad_both_ends(text, n=2)) for text in chesterton_preprocessed ]

austen_bigrams = [ ngrams(doc, 2) for doc in austen_padded2 ]
austen_vocab2 = [ word for doc in austen_padded2 for word in doc ]

chesterton_bigrams = [ ngrams(doc, 2) for doc in chesterton_padded2 ]
chesterton_vocab2 = [ word for doc in chesterton_padded2 for word in doc ]
```

In [100]:

```
from nltk.lm import MLE
lm_austen2 = MLE(2)
lm_austen2.fit(austen_bigrams, austen_vocab2)

lm_chesterton2 = MLE(2)
lm_chesterton2.fit(chesterton_bigrams, chesterton_vocab2)
```

(j) For each author, use the correspondent language models from (i) to generate, using MLE, a sentence of fifteen words starting from each of the following terms:

In [101]:

```
def print_sentences(n, start, models, random_seed=None):
    for label, model in models:
        print('-- {0} --\n'.format(label))
```

```
        for s in start:
            gen = ' '.join(model.generate(n-1, text_seed=s, random_seed=random_seed))
            print('{0} {1}'.format(s[-1], gen))
        print('\n')
```

In [102]:

```
start = [['she'], ['they'], ['it']] # context text also has to be preprocessed

print_sentences(15, start, [('Austen Bigrams', lm_austen2), ('Chesterton Bigrams', lm_chesterton2)
], 344493)
```

```
-- Austen Bigrams --

she believed half year lightened too diffident of its size of great deal under the
they are better was not speak do the letter was safely covered with what she
it amiss for you made so easily removed her these their departure arrived together mrs


-- Chesterton Bigrams --

she always been walking on the departing detective in their parents because it what rich
they are buzzing down said the departing detective in their parents because it what rich
it and cocoa this man was confused things it was set forth in voice peculiarly
```

(k) Now, computer another two models (for Austen and Chesterton separately), build a trigram model with the same data as in (i). Use both right and left padding, and manage unknown terms by using a dedicated token.

In [103]:

```
austen_padded3 = [ list(pad_both_ends(text, n=3)) for text in austen_preprocessed ]
chesterton_padded3 = [ list(pad_both_ends(text, n=3)) for text in chesterton_preprocessed ]

austen_trigrams = [ ngrams(doc, 3) for doc in austen_padded3]
austen_vocab3 = [ word for doc in austen_padded3 for word in doc ]

chesterton_trigrams = [ ngrams(doc, 3) for doc in chesterton_padded3]
chesterton_vocab3 = [ word for doc in chesterton_padded3 for word in doc ]

lm_austen3 = MLE(3)
lm_austen3.fit(austen_trigrams, austen_vocab3)

lm_chesterton3 = MLE(4)
lm_chesterton3.fit(chesterton_trigrams, chesterton_vocab3)
```

(l) For each author, use the correspondent language models from (k) to generate, using MLE, a sentence of fifteen words starting from each of the same terms as question (j):

In [104]:

```
start = [['<s>', 'she'], ['<s>' ,'they'], ['<s>', 'it']]

print_sentences(15, start, [('Austen Trigrams', lm_austen3), ('Chesterton Trigrams',
lm_chesterton3)], 344493)
```

```
-- Austen Trigrams --

she cared for there really is engaged from morning to the excellence of the party
they are as she listened to anew on this subject before had known your engagement
it had been walking some time but was not unnatural for me however got very


-- Chesterton Trigrams --

she flung back three of these carriages stood rank of such gloves and there the
they are charging the mob he said three minutes off yes said macian with singular
it fastens up to michael the sound of the tower that seemed to talk reasonably
```

(m) Comment on the quality of the models and generated text. Which model performs better? In general, which differences are there in using trigrams as opposed to bigrams?

Explanation:

Question 6 - Process Mining (15 points)

For this part, refer to the online docs of pm4py (https://pm4py.fit.fraunhofer.de/). Important: if you did not do it in the instruction, you should make sure to have the latest pm4py version: to get it is sufficient to type `pip install pm4py --upgrade`. In this section of the assignment you will be working with a real life event log, derived from the activity of the help desk department of an Italian software company. The data is contained in the file "event_log.xes".

(a) Use the provided event log and identify the least frequent variant and the most frequent variant.

In [105]:

```
from pm4py.objects.log.importer.xes import factory as xes_importer

log = xes_importer.import_log('event_log.xes')
```

In [117]:

```
from pm4py.statistics.traces.log import case_statistics
variants_count = case_statistics.get_variant_statistics(log)
variants_count = sorted(variants_count, key=lambda x: x['count'], reverse=True)
most_frequent = variants_count[0]
least_frequent = [v for v in variants_count if v['count'] == 1]

print('Most frequent variant: {0}'.format(most_frequent))
```

```
Most frequent variant: {'variant': 'Assign seriousness,Take in charge ticket,Resolve
ticket,Closed', 'count': 2366}
```

There are several variants that occur only 1 time and therefore share the least frequent variant.

In [146]:

```
least_frequent[0:5]
```

Out[146]:

```
[{'variant': 'Wait,Resolve ticket,Closed', 'count': 1},
 {'variant': 'Take in charge ticket,Wait,Wait,Resolve ticket,Closed',
  'count': 1},
 {'variant': 'Take in charge ticket,Wait,Resolve ticket,Take in charge ticket,Wait,Resolve
ticket,Closed',
  'count': 1},
 {'variant': 'Take in charge ticket,Take in charge ticket,Take in charge ticket,Resolve
ticket,Closed',
  'count': 1},
 {'variant': 'Take in charge ticket,Take in charge ticket,Resolve ticket,Closed',
  'count': 1}]
```

(b) Remove all the variants that contain less than 1% of the traces in the log. Create a new event log without these variants.

In [131]:

```
from pm4py.algo.filtering.log.variants import variants_filter
boundary = int(len(log)*0.01)
filtered_variants = [v['variant'] for v in variants_count if v['count'] >= boundary]
filtered_log = variants_filter.apply(log, filtered_variants)
```

(c) Use Inductive miner algorithm to discover the process model based on you new event log (the filtered log without the infrequent variants of question (b)).

In [143]:

```
from pm4py.algo.discovery.inductive import factory as inductive_miner

tree = inductive_miner.apply_tree(filtered_log)

# Mining for a Petri net
net, initial_marking, final_marking = inductive_miner.apply(filtered_log)
```

(d) Perform the token replay conformance checking using your discovered model (c) and the original event log. Does your process model fit the log?

In [151]:

```
from pm4py.algo.conformance.tokenreplay import factory as token_based_replay_factory

token_replay_result = token_based_replay_factory.apply(log, net, initial_marking, final_marking)

fit_traces = 0
unfit_traces = 0
for r in token_replay_result:
    if r['trace_is_fit']:
        fit_traces += 1
    else:
        unfit_traces += 1

fit_traces_quota = fit_traces/(fit_traces + unfit_traces)

print('Fit traces: {0}\nUnfit Traces: {1}\nQuota of fit traces: {2}'.format(fit_traces,
unfit_traces, fit_traces_quota))
```

```
Fit traces: 2989
Unfit Traces: 1591
Quota of fit traces: 0.6526200873362445
```

Explanation:

The model fits less than 2/3 of the traces, which is not a good score. As a consequence the process model does not fit the log well.

(e) Calculate the fitness of the model in (c).

In [195]:

```
from pm4py.evaluation.replay_fitness import factory as replay_fitness_factory

fitness_tokenbasedreplay = replay_fitness_factory.apply(log, net, initial_marking, final_marking)
print(fitness_tokenbasedreplay)
```

```
{'percFitTraces': 62.99126637554585, 'averageFitness': 0.9387764189103065}
```

(f) Are there any deviations between the process model and the event log? If so, where and why?

In [161]:

```
deviations = [t['transitions_with_problems'] for t in token_replay_result if
t['transitions_with_problems'] != []]
deviations[0:10]
```

Out[161]:

```
[[Take in charge ticket],
 [Wait],
 [Closed],
 [Wait],
 [Wait],
 [Wait],
 [Wait]]
```

```
[Wait],
[Closed],
[Wait],
[Wait]]
```

Explanation: There are deviations (obviously, because there are only 63% fit traces), of which most occur at the transition 'Wait'

(g) Now use the original event log and remove the two most frequent variants, and discover the model based on your new event log (the filtered log without two most frequent variants).

In [190]:

```
filtered_variants2 = [v['variant'] for v in variants_count[2:]]
filtered_log2 = variants_filter.apply(log, filtered_variants2)
```

In [191]:

```
tree2 = inductive_miner.apply_tree(filtered_log2)

# Mining for a Petri net
net2, initial_marking2, final_marking2 = inductive_miner.apply(filtered_log2)
```

(h) Perform the token replay conformance checking using the newly discovered model of question (g) and the original event log. Does your process model fit the log?

In [192]:

```
token_replay_result2 = token_based_replay_factory.apply(log, net2, initial_marking2, final_marking2
)

fit_traces2 = 0
unfit_traces2 = 0
for r in token_replay_result2:
    if r['trace_is_fit']:
        fit_traces2 += 1
    else:
        unfit_traces2 += 1

fit_traces_quota2 = fit_traces2/(fit_traces2 + unfit_traces2)

print('Fit traces: {0}\nUnfit Traces: {1}\nQuota of fit traces: {2}'.format(fit_traces2,
unfit_traces2, fit_traces_quota2))
```

```
Fit traces: 4204
Unfit Traces: 376
Quota of fit traces: 0.9179039301310044
```

Explanation: With roughly 92% fitting traces, the model fits the log well.

(i) Calculate the fitness of the model in (g).

In [196]:

```
fitness_tokenbasedreplay2 = replay_fitness_factory.apply(log, net2, initial_marking2,
final_marking2)
print(fitness_tokenbasedreplay2)
```

```
{'percFitTraces': 0.2183406113537118, 'averageFitness': 0.752955974156862}
```

(j) Are there any deviations in the process model? If so, where and why?

In [164]:

```
deviations2 = [t['transitions_with_problems'] for t in token_replay_result2 if
t['transitions_with_problems'] != []]
```

```
deviations2[0:10]
```

Out[164]:

```
[[Closed],
 [Closed],
 [Closed],
 [Closed],
 [Closed],
 [Closed],
 [Closed],
 [Closed],
 [Closed],
 [Closed]]
```

Explanation: Again there are deviations, but this time they most often occur at transition 'Closed'

(k) Use the complete event log (original event log) and discover your process model using Inductive Miner.

In [170]:

```
tree3 = inductive_miner.apply_tree(log)

# Mining for a Petri net
net3, initial_marking3, final_marking3 = inductive_miner.apply(log)
```

(l) Do the token replay conformance checking using your newly discovered model and the original event log. Does your process model fit the log?

In [171]:

```
token_replay_result3 = token_based_replay_factory.apply(log, net3, initial_marking3, final_marking3
)

fit_traces3 = 0
unfit_traces3 = 0
for r in token_replay_result3:
    if r['trace_is_fit']:
        fit_traces3 += 1
    else:
        unfit_traces3 += 1

fit_traces_quota3= fit_traces3/(fit_traces3 + unfit_traces3)

print('Fit traces: {0}\nUnfit Traces: {1}\nQuota of fit traces: {2}'.format(fit_traces3,
unfit_traces3, fit_traces_quota3))
```

```
Fit traces: 4204
Unfit Traces: 376
Quota of fit traces: 0.9179039301310044
```

Explanation: The model fits the log exactly as well as the previous model, which is pretty good, but not 100%. No new information has been added to the model by the two most frequent trace variants.

(m) How are these three discovered process models different from each other? Which model is the best fitting to the original log? Why?

Explanation: Model 2 and Model 3 are the same, while the first model is a much simpler model, which is due to the fact that the special, not so frequent variants were not considered in the model generation. The model 2/3 is the one that fits the original log better, but it also is much more complex due to all the special traces that were considered in the generation.

(n) Visualize the model discovered in question (k) enriched with frequency information. Subsequently, visualize that same model enriched with performance information.

In [180]:

```
from pm4py.visualization.petrinet import factory as pn_vis_factory
parameters = {"format":"png"}
```

```
parameters = {"format":"png"}
gviz = pn_vis_factory.apply(net3, initial_marking3, final_marking3, parameters=parameters, variant=
"frequency", log=log)
pn_vis_factory.save(gviz, "inductive_frequency.png")
gviz
```

Out[180]:

In [181]:

```
parameters = {"format":"png"}
gviz = pn_vis_factory.apply(net3, initial_marking3, final_marking3, parameters=parameters, variant=
"performance", log=log)
pn_vis_factory.save(gviz, "inductive_performance.png")
gviz
```

Out[181]:

(o) If you were the process owner and you had more resources to hire employees, on which parts of the process would you assign them to maximize efficiency?

Explanation: The transition 'take in charge ticket' takes a very long time and we would try to speed up this process. It should not take so long for the company to take charge of an incoming ticket.

(p) Add your further comments and considerations on the frequency and performance informations. Does the process have a "happy path", and is it shown in the enhanced process models? Where are the major bottlenecks in this process? Is there any rework?

Explanation: Yes, the process has a happy path, which can be easily seen in the frequency visualization by following arrows with a value > 4000.