Data Scientist Case Study

In the MyHammer platform we have one challenge. Often consumers submit the service request in a wrong serviceId category, mostly they get tempted to choose easy options such as “Sonstiges” (Others). For example, an actual painting job gets submitted to “Others” serviceId category. We have discovered the following issues with the current setup:

- jobs are in a wrong category
- tradesmen cannot find the proper job because of the above scenario.

This has a direct impact in revenue because the tradesmen cannot find a relevant job while searching. The task is to build an efficient classifier which can classify the wrong serviceId to a correct one. The goal is to help consumers to pick the right category by providing a good suggestion of a correct serviceId category based on the service request details.

Structure

- **Import libraries and setup Dataframes**
- **Prepare the Data (preprocessing)**
- **Choose the model**
- **Train the model**
- **Make prediction**
- **Summary of Results**

Import Libraries & Setup Data

In [1]:

```
import pandas as pd
import numpy as np
from numpy import array
from numpy import asarray
from numpy import zeros
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
plt.style.use('seaborn-whitegrid')
from imblearn.under_sampling import TomekLinks
import nltk as nltk
from nltk.corpus import stopwords
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.impute import SimpleImputer
import sklearn.metrics as metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from keras.preprocessing.sequence import pad_sequences
from keras.layers.core import  Dense
from keras.layers import  LSTM
from keras.models import Model
from keras.layers.embeddings import Embedding
from keras.preprocessing.text import Tokenizer
from keras.layers import Input
from keras.layers.merge import Concatenate
from sklearn.model_selection import train_test_split
```

In [2]:

```
# Import data
df = pd.read_excel("Data.xlsx")


# check the data
df.head(3)
#df_test.shape
```

Out[2]:

	id	title	description	createdAt	endedAt	serviceId	user_description	se
0	10870620	Badsanierung 39291 Lostau	Teilaufgaben: Fliesen verlegen, Heizung instal...	2019-05-12 10:10:45	2019-07-05 20:41:41	405110.0	Badsanierung Wanne Toilette Waschtisch verse...	
1	10877080	4 m¬≤ Privatfl√§che pflastern, Unterbau erstel...	Teilaufgaben: Privatfl√§che pflastern, Unterba...	2019-05-12 18:06:15	2019-07-02 21:47:02	104010.0	Durch einen neuen Durchbruch der Au√üenwand wu...	
2	10898310	Dusche nachr√ºsten, Heizk√∂rper tauschen, Wass...	- Dusche nachr√ºsten im Bad (EG) (inkl. Wasser...	2019-05-14 08:13:56	2019-07-07 10:01:03	405120.0	- Dusche nachr√ºsten im Bad (EG) (inkl. Wasser...	

3 rows × 22 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25770 entries, 0 to 25769
Data columns (total 22 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       25770 non-null  int64  
 1   title                    25768 non-null  object 
 2   description              25733 non-null  object 
 3   createdAt                25438 non-null  object 
 4   endedAt                  25438 non-null  object 
 5   serviceId                25438 non-null  object 
 6   user_description         530 non-null    object 
 7   service_based_form       25437 non-null  object 
 8   target_date              25438 non-null  object 
 9   reissuedCopyOf           25437 non-null  object 
 10  tradeClassificationType  25437 non-null  object 
 11  state                    25438 non-null  float64
 12  stateText                25438 non-null  object 
 13  sbf_form_text            24990 non-null  object 
 14  Unnamed: 14              3 non-null      object 
 15  Unnamed: 15              2 non-null      object 
 16  Unnamed: 16              2 non-null      object 
 17  Unnamed: 17              2 non-null      object 
 18  Unnamed: 18              2 non-null      object 
 19  Unnamed: 19              2 non-null      object 
 20  Unnamed: 20              1 non-null      object 
 21  Unnamed: 21              1 non-null      object 
dtypes: float64(1), int64(1), object(20)
memory usage: 4.3+ MB
```

Prepare the Data (preprocessing)

1. **Remove unneeded, irrelevant features**

In [4]:

```
df.drop(df.columns[[14,15,16,17,18,19,20,21]], axis=1, inplace=True)
df.drop(['createdAt','endedAt','id','sbf_form_text'], axis =1, inplace= True)

# check the data
df.head(3)
#df_test.shape
```

Out[4]:

	title	description	serviceId	user_description	service_based_form	target_date	reis
0	Badsanierung 39291 Lostau	Teilaufgaben: Fliesen verlegen, Heizung instal...	405110.0	Badsanierung Wanne Toilette Waschtisch verse...	1	In den n√§chsten 3 Monaten	
1	4 m¬≤ Privatfl√§che pflastern, Unterbau erstel...	Teilaufgaben: Privatfl√§che pflastern, Unterba...	104010.0	Durch einen neuen Durchbruch der Au√üenwand wu...	1	Zeitnah	
2	Dusche nachr√ºsten, Heizk√∂rper tauschen, Wass...	- Dusche nachr√ºsten im Bad (EG) (inkl. Wasser...	405120.0	- Dusche nachr√ºsten im Bad (EG) (inkl. Wasser...	0	Zeitnah	

In [5]:

```
# Check how our final data frame looks
print("Rows & Columns: ", df.shape, "\nAll columns if the df: ", df.columns.tolist(
```

```
Rows & Columns:  (25770, 10)
All columns if the df:  ['title', 'description', 'serviceId', 'user_de
scription', 'service_based_form', 'target_date', 'reissuedCopyOf', 'tr
adeClassificationType', 'state', 'stateText']
```

2. **Looking deeply in our data to determine data type of each feature**

In [6]:

```
print("Unique values for service_based_form: ", df.service_based_form.unique())
print("Unique values for reissuedCopyOf: ", df.reissuedCopyOf.unique())
print("Unique values for tradeClassificationType: ", df.tradeClassificationType.uni
print("Unique values for target_date: ", df.target_date.unique())
print("Unique values for state : ", df.state.unique())
print("Unique values for stateText : ", df.stateText.unique())
print("Unique values for serviceId : ", df.serviceId.unique())
```

```
Unique values for service_based_form:  [1 0 nan datetime.datetime(202
0, 12, 13, 12, 58, 1)
 datetime.datetime(2020, 11, 29, 14, 51, 14)]
Unique values for reissuedCopyOf:  [0 nan 10200185 10671805 8876504 10
379245 11526527 13943333
 'Wunschtermin: 27.11.2020' '105080.0']
Unique values for tradeClassificationType:  ['1.0' '4.0' '2.0' nan 946
667 0 'Innerhalb der n√§chsten 30 Tage']
Unique values for target_date:  ['In den n√§chsten 3 Monaten' 'Zeitna
h' nan 'In 3 bis 6 Monaten'
 'Innerhalb der n√§chsten 30 Tage' 'In mehr als 6 Monaten'
 'Wunschtermin: 11.06.2020' 'Wunschtermin: 28.06.2020'
 'Wunschtermin: 21.09.2019' 'Wunschtermin: 19.12.2019'
 'Wunschtermin: 15.11.2019' 'Wunschtermin: 29.10.2019'
 'Wunschtermin: 11.10.2019' 'Wunschtermin: 25.01.2020'
 'Wunschtermin: 06.12.2019' 'Wunschtermin: 06.01.2020'
 'Wunschtermin: 13.04.2020' 'Wunschtermin: 24.05.2020'
 'Wunschtermin: 21.03.2020' 'Wunschtermin: 12.11.2020'
 'Wunschtermin: 12.12.2020' 'Wunschtermin: 07.12.2020'
 'Wunschtermin: 18.01.2021' 'Wunschtermin: 02.12.2020'
 'Wunschtermin: 23.11.2020' 'Wunschtermin: 08.12.2020'
 'Wunschtermin: 22.01.2021' 'Wunschtermin: 13.02.2021'
 'Wunschtermin: 30.11.2020' 'Wunschtermin: 27.11.2020'
 'Wunschtermin: 22.11.2020' 'Wunschtermin: 15.02.2021'
 'Wunschtermin: 20.11.2020' 'Wunschtermin: 25.11.2020'
 'Wunschtermin: 04.01.2021' 'Wunschtermin: 04.12.2020'
 'Wunschtermin: 20.01.2021' 'Wunschtermin: 03.05.2021'
 'Wunschtermin: 29.01.2021' 'Wunschtermin: 13.01.2021'
 'Wunschtermin: 15.03.2021' 'Wunschtermin: 28.12.2020'
 'Wunschtermin: 26.11.2020' 'Wunschtermin: 03.12.2020'
 'Wunschtermin: 23.12.2020' 'Wunschtermin: 21.11.2020'
 'Wunschtermin: 29.05.2021' 'Wunschtermin: 14.12.2020'
 'Wunschtermin: 06.02.2021' 'Wunschtermin: 24.11.2020'
 'Wunschtermin: 01.07.2021' 'Wunschtermin: 05.01.2021'
 'Wunschtermin: 12.07.2021' 'Wunschtermin: 01.02.2021'
 'Wunschtermin: 10.12.2020' 'Wunschtermin: 18.12.2020'
 'Wunschtermin: 15.12.2020' 'Wunschtermin: 07.01.2021'
 'Wunschtermin: 30.06.2021' 'Wunschtermin: 04.10.2021'
 'Wunschtermin: 11.01.2021' 1 'Wunschtermin: 17.05.2021'
 'Wunschtermin: 16.12.2020' 'Wunschtermin: 01.04.2021'
 'Wunschtermin: 19.12.2020' 'Wunschtermin: 22.12.2020'
 'Wunschtermin: 28.01.2021' 'Wunschtermin: 21.06.2021'
 'Wunschtermin: 27.01.2021' 'Wunschtermin: 31.01.2021'
 'Wunschtermin: 01.06.2021' 'Wunschtermin: 15.01.2021'
 'Wunschtermin: 09.12.2020' 'Wunschtermin: 21.01.2021'
 'Wunschtermin: 05.12.2020' 'Wunschtermin: 12.01.2021'
 'Wunschtermin: 24.12.2020' 'Wunschtermin: 31.03.2021'
 'Wunschtermin: 17.12.2020' 'Wunschtermin: 28.11.2020'
 'Wunschtermin: 14.01.2021' 'Wunschtermin: 21.12.2020'
 'Wunschtermin: 02.01.2021' 'Wunschtermin: 01.01.2021'
 'Wunschtermin: 08.01.2021' 'Wunschtermin: 31.05.2021'
```

```
 'Wunschtermin: 12.02.2021' 'Wunschtermin: 01.03.2021'
 'Wunschtermin: 31.07.2021' 'Wunschtermin: 01.12.2020'
 'Wunschtermin: 25.01.2021' 'Wunschtermin: 30.12.2020'
 'Wunschtermin: 08.02.2021' 'Wunschtermin: 19.01.2021'
 'Wunschtermin: 26.02.2021' 'Wunschtermin: 02.02.2021'
 'Wunschtermin: 05.06.2021' 'Wunschtermin: 31.12.2020'
 'Wunschtermin: 10.01.2021' 'Wunschtermin: 02.07.2021'
 'Wunschtermin: 30.01.2021' 'Wunschtermin: 19.07.2021'
 'Wunschtermin: 16.01.2021' 'Wunschtermin: 11.12.2020'
 'Wunschtermin: 23.01.2021' 'Wunschtermin: 08.03.2021'
 'Wunschtermin: 06.04.2021' 'Wunschtermin: 05.03.2021'
 'Wunschtermin: 09.01.2021' 'Wunschtermin: 29.11.2020'
 'Wunschtermin: 27.02.2021' 'Wunschtermin: 29.12.2020'
 'Wunschtermin: 23.07.2021' 'Wunschtermin: 29.03.2021'
 'Wunschtermin: 13.12.2020' 'Wunschtermin: 01.05.2021'
 'Wunschtermin: 23.02.2021' 'Wunschtermin: 06.01.2021'
 'Wunschtermin: 04.02.2021' 'Wunschtermin: 22.02.2021'
 'Wunschtermin: 19.02.2021' 'Wunschtermin: 20.02.2021'
 'Wunschtermin: 22.03.2021' 'Wunschtermin: 28.08.2021'
 'Wunschtermin: 14.02.2021' 'Wunschtermin: 06.12.2020'
 'Wunschtermin: 16.02.2021' 'Wunschtermin: 03.01.2021'
 'Wunschtermin: 05.02.2021' 'Wunschtermin: 11.06.2021'
 'Wunschtermin: 02.08.2021' 'Wunschtermin: 18.02.2021'
 'Wunschtermin: 15.04.2021' 'Wunschtermin: 09.08.2021'
 'Wunschtermin: 01.08.2021' 'Wunschtermin: 25.02.2021'
 'Wunschtermin: 10.02.2021' 'Wunschtermin: 15.07.2021'
 'Wunschtermin: 13.03.2021' 'Wunschtermin: 01.09.2021'
 'Wunschtermin: 04.03.2021' 'Wunschtermin: 05.04.2021'
 'Wunschtermin: 30.04.2021' 'Wunschtermin: 02.09.2021'
 'Wunschtermin: 01.12.2021' 'Wunschtermin: 26.01.2021' '405120.0'
 datetime.datetime(2020, 12, 13, 14, 51, 14) 'Wunschtermin: 17.02.202
1'
 'Wunschtermin: 11.02.2021' 'Wunschtermin: 26.04.2021'
 'Wunschtermin: 10.06.2021' 'Wunschtermin: 03.03.2021'
 'Wunschtermin: 10.03.2021' 'Wunschtermin: 25.06.2021'
 'Wunschtermin: 20.12.2020' 'Wunschtermin: 27.12.2020'
 'Wunschtermin: 24.02.2021' 'Wunschtermin: 28.02.2021'
 'Wunschtermin: 06.03.2021' 'Wunschtermin: 17.01.2021'
 'Wunschtermin: 20.03.2021' 'Wunschtermin: 03.02.2021']
Unique values for state :  [0.00000000e+00            nan 4.00000000e+
00 4.81700928e+08]
Unique values for stateText :  ['active' nan 0 115776824 505833]
Unique values for serviceId :  ['405110.0' '104010.0' '405120.0' '1040
40.0' '405140.0' '404040.0'
 '109000.0' nan '703030.0' '105080.0' '107010.0' '107070.0' '412120.0'
 '412030.0' '101020.0' '409000.0' '504000.0' '108020.0' '106040.0'
 '108130.0' '703010.0' '109010.0' '402070.0' '405020.0' '107050.0'
 '101120.0' '601000.0' '403040.0' '407100.0' '402030.0' '410050.0'
 '405050.0' '411020.0' '101030.0' '408020.0' '304010.0' '404020.0'
 '407040.0' '101110.0' '107090.0' '405090.0' '402020.0' '102010.0'
 '412090.0' '701000.0' '107080.0' '101060.0' '107020.0' '106020.0'
 '412100.0' '101010.0' '102080.0' '412010.0' '404050.0' '108030.0'
 '412020.0' '102030.0' '412080.0' '411100.0' '406020.0' '412040.0'
 '403010.0' '104030.0' '304080.0' '403060.0' '411080.0' '101070.0'
 '702020.0' '401000.0' '403100.0' '101140.0' '411040.0' '402120.0'
 '402050.0' '102050.0' '411060.0' '410030.0' '405010.0' '102060.0'
 '101080.0' '411010.0' '412110.0' '410010.0' '407010.0' '407020.0'
 '107060.0' '101050.0' '304020.0' '304030.0' '107030.0' '503000.0'
 '502000.0' '202000.0' '608000.0' '304050.0' '402110.0' '107100.0'
 '412130.0' '108120.0' '402060.0' '704000.0' '407090.0' '102070.0'
 '404010.0' '605000.0' '108070.0' '105040.0' '804010.0' '105020.0'
```

```
 '410040.0' '405100.0' '404080.0' '412060.0' '406030.0' '802010.0'
 '405070.0' '108010.0' '407070.0' '105030.0' '102020.0' '607000.0'
 '408030.0' '804020.0' '304060.0' '108080.0' '407050.0' '802030.0'
 '411090.0' '108050.0' '108110.0' '702010.0' '108150.0' '405130.0'
 '403030.0' '108140.0' '402040.0' '702030.0' '602000.0' '105010.0'
 '402100.0' '705000.0' '411050.0' '101040.0' '106030.0' '102040.0'
 '405060.0' '412050.0' '804030.0' '105060.0' '412070.0' '107120.0'
 '304040.0' '304090.0' '410020.0' '402090.0' '802020.0' '501000.0'
 '703040.0' '802040.0' '108100.0' '703020.0' '103040.0' '407030.0'
 '404060.0' '304100.0' '402010.0' '406040.0' '108040.0' '108060.0'
 '801000.0' '302000.0' '103020.0' '201000.0' '407060.0' '930020.0'
 '107160.0' '406010.0' '803000.0' '107110.0' '103010.0' '408010.0'
 '404030.0' '104020.0' '603000.0' '606000.0' '403080.0' '702040.0'
 '403050.0' '411070.0' '407080.0' '204000.0' '405080.0' '107040.0'
 '405030.0' '411030.0' '403020.0' '708010.0' '804040.0' '107170.0'
 '604000.0' '408040.0' '105050.0' '706030.0' '720010.0' '402080.0'
 '403070.0' '108090.0' '910060.0' '405040.0' '105070.0' '404070.0'
 '920040.0' '106010.0' '101130.0' '301010.0' '101100.0' '103030.0'
 datetime.datetime(2020, 12, 4, 14, 35, 58) '303000.0' '107140.0'
 '101090.0' '720050.0' '301020.0' '107130.0' '304070.0' '920010.0'
 '720040.0' '203000.0' '403090.0' '301030.0' '716090.0' '930010.0'
 '706050.0' '720030.0' '910010.0' '716070.0' '709030.0' '920020.0'
 '910030.0' '107180.0' '107150.0' '706060.0' '709020.0' '716060.0'
 'Wandhalterung im WC (2 L√∂cher)  ' '700 m¬≤ = ca. 600 m¬≥ 4.' '71001
0.0'
 '910040.0' '720020.0' '910020.0']
```

We can consider the following feature as categorial ones, since some data pattern are repeating for them

- **service_based_form**: include numerical and datatime data
- **reissuedCopyOf**: include numerical and text data
- **tradeClassificationType**: include numerical and text data
- **tradeClassificationType**: include numerical and text data
- **target_date**: include text data
- **state**: include numerical data
- **stateText**: include numerical and text data
- **serviceId**: include numerical and text data

Converting categorial features to numerical one

- service_based_form
- reissuedCopyOf
- tradeClassificationType
- state
- target_date
- stateText
- serviceId

In [7]:

```
from sklearn import preprocessing
df['serviceId'] = df['serviceId'].astype('str')
le = preprocessing.LabelEncoder()
df['serviceId'] = le.fit_transform(df.serviceId.values)


df['service_based_form'] = df['service_based_form'].astype('str')
le = preprocessing.LabelEncoder()
df['service_based_form'] = le.fit_transform(df.service_based_form.values)

df['reissuedCopyOf'] = df['reissuedCopyOf'].astype('str')
le = preprocessing.LabelEncoder()
df['reissuedCopyOf'] = le.fit_transform(df.reissuedCopyOf.values)

df['state'] = df['state'].astype('str')
le = preprocessing.LabelEncoder()
df['state'] = le.fit_transform(df.state.values)

df['tradeClassificationType'] = df['tradeClassificationType'].astype('str')
le = preprocessing.LabelEncoder()
df['tradeClassificationType'] = le.fit_transform(df.tradeClassificationType.values)

df['target_date'] = df['target_date'].astype('str')
le = preprocessing.LabelEncoder()
df['target_date'] = le.fit_transform(df.target_date.values)


df['stateText'] = df['stateText'].astype('str')
le = preprocessing.LabelEncoder()
df['stateText'] = le.fit_transform(df.stateText.values)
```

In [8]:

```
# check the data
df.head(3)
#df_test.shape
```

Out[8]:

	title	description	serviceId	user_description	service_based_form	target_date	reis
0	Badsanierung 39291 Lostau	Teilaufgaben: Fliesen verlegen, Heizung instal...	138	Badsanierung Wanne Toilette Waschtisch verse...	1	4	
1	4 m¬≤ Privatfl√§che pflastern, Unterbau erstel...	Teilaufgaben: Privatfl√§che pflastern, Unterba...	26	Durch einen neuen Durchbruch der Au√üenwand wu...	1	162	
2	Dusche nachr√°sten, Heizk√∂rper tauschen, Wass...	- Dusche nachr√°sten im Bad (EG) (inkl. Wasser...	139	- Dusche nachr√°sten im Bad (EG) (inkl. Wasser...	0	162	

3. **Checking missing values in dataset**

In [9]:

```
# Check where we find NaN values

tab_info = pd.DataFrame(df.dtypes).T.rename(index={0:'column Type'})
tab_info = tab_info.append(pd.DataFrame(df.isnull().sum()).T.rename(index={0:'null
tab_info = tab_info.append(pd.DataFrame(df.isnull().sum()/df.shape[0]*100).T.
                                                    rename(index={0: 'null values (%)'}))
tab_info
```

Out[9]:

	title	description	serviceId	user_description	service_based_form	target_date
column Type	object	object	int64	object	int64	int64
null values (nb)	2	37	0	25240	0	0
null values (%)	0.00776096	0.143578	0	97.9433	0	0

In [10]:

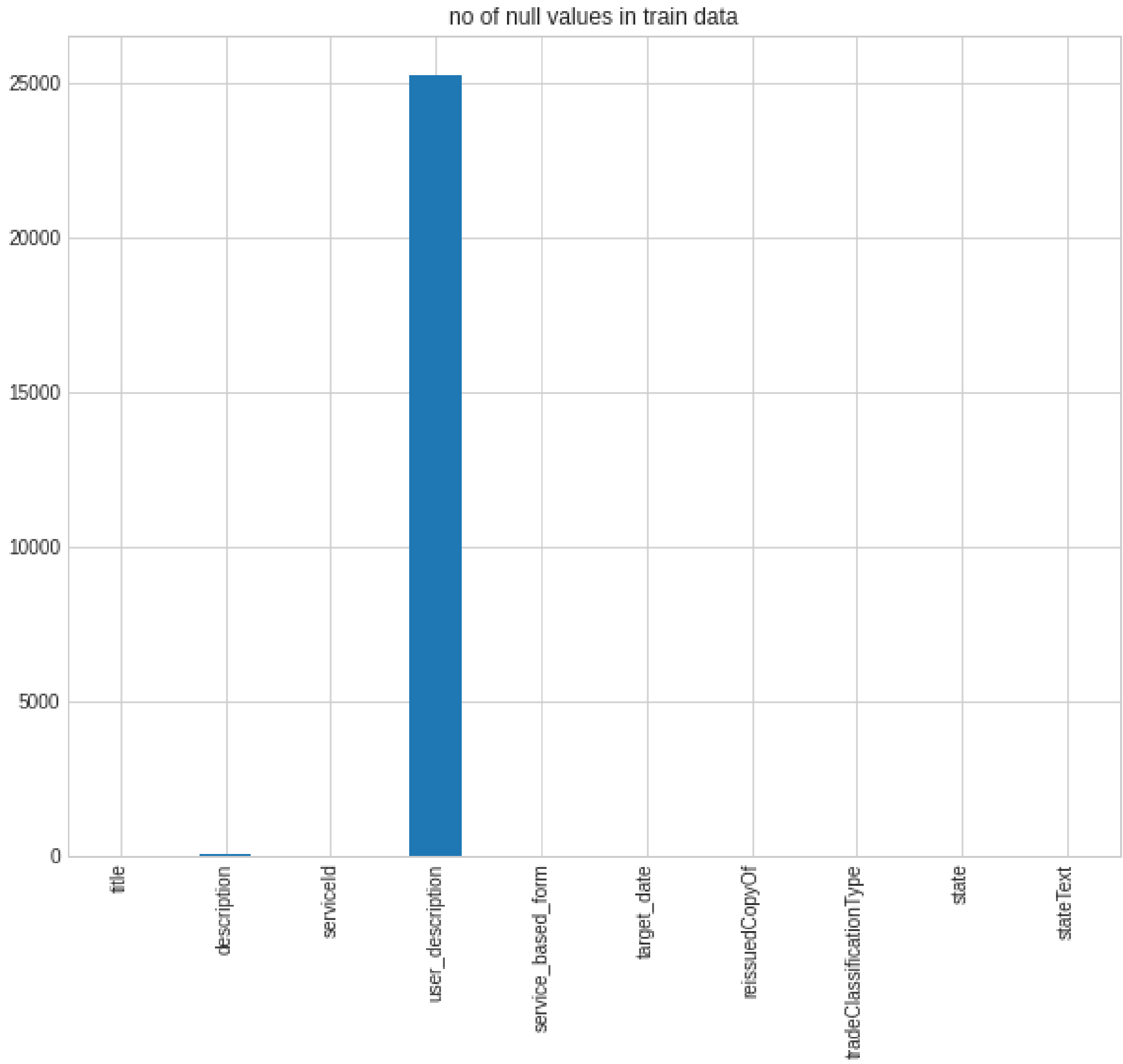
```
df.isnull().sum()
```

Out[10]:

```
title                         2
description                  37
serviceId                     0
user_description          25240
service_based_form            0
target_date                   0
reissuedCopyOf                0
tradeClassificationType       0
state                         0
stateText                     0
dtype: int64
```

In [11]:

```
df.isna().sum().plot(kind="bar",figsize=(10, 8))
plt.title("no of null values in train data")
plt.show()
```



Imputing missing values

- drop col **user_description** due to having more that 70% missing values
- col **title** only has two missing values so we remove these two rows
- imputing categorical features & numerical values with more frequency values in feature
- imputing text features with constant text **nothing**

In [12]:

```
df.drop(['user_description'], axis=1, inplace=True)

imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
df["serviceId"] = imp.fit_transform(df[["serviceId"]])
df["service_based_form"] = imp.fit_transform(df[["service_based_form"]])
df["target_date"] = imp.fit_transform(df[["target_date"]])
df["reissuedCopyOf"] = imp.fit_transform(df[["reissuedCopyOf"]])
df["tradeClassificationType"] = imp.fit_transform(df[["tradeClassificationType"]])
df["state"] = imp.fit_transform(df[["state"]])
df["stateText"] = imp.fit_transform(df[["stateText"]])

imp = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value="nothing
df["description"] = imp.fit_transform(df[["description"]])
```

In [13]:

```
# Check how our final data frame looks
print("Rows & Columns before remove nan value rows: ", df.shape)
df.dropna(inplace=True)
# Check how our final data frame looks
print("Rows & Columns before remove nan value rows: ", df.shape)
```

```
Rows & Columns before remove nan value rows:  (25770, 9)
Rows & Columns before remove nan value rows:  (25768, 9)
```

In [14]:

```
# check the data
df.tail(3)
#df_test.shape
```

Out[14]:

	title	description	serviceId	service_based_form	target_date	reissuedCopyOf
25767	Glaswand f√ºr Dusche ohne Haltestange	Ich h√§tte gerne eine Glaswand 1 m breit und ...	160	1	6	0
25768	Birken f√§llen in 14621 Sch√∂nwalde-Glien	Teilaufgaben: Baum f√§llen Anzahl B√§ume: +5 A...	44	1	162	0
25769	80 m¬≤ PVC-Boden verlegen + Material vorhanden...	Teilaufgaben: PVC-Boden verlegen Geb√§udeart: ...	103	1	28	0

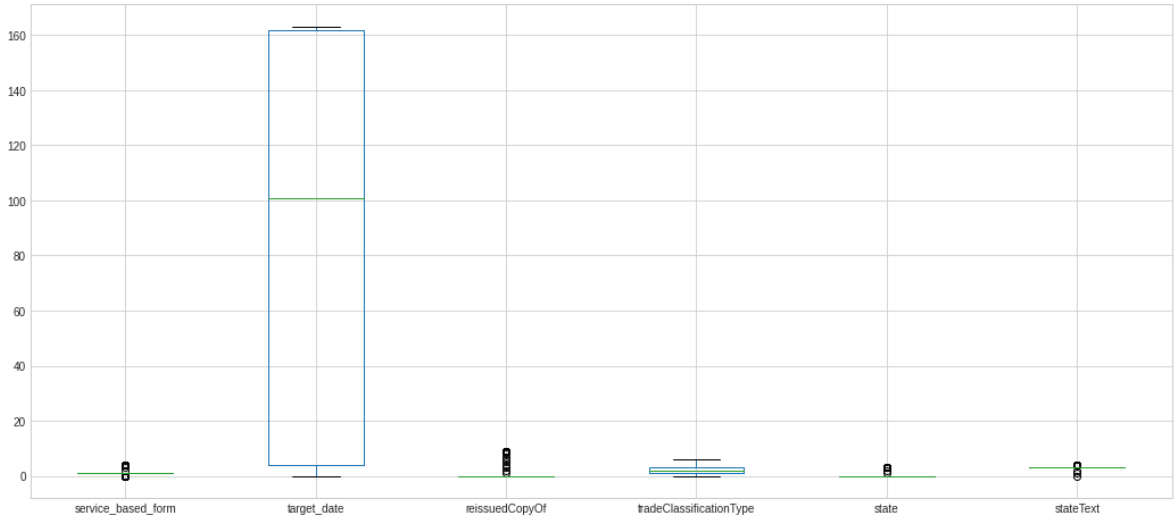
4- Checking for outliers in data

In [15]:

```
df.plot(x = 'serviceId', kind = 'box',figsize=(18, 8))
```

Out[15]:

```
<AxesSubplot:>
```



From box-plot it is clear we have outliers in serveral features. We need to remove these outliers as it affect the accuracy of the result.

Removing outliers from all columens in Data

In [16]:

```
from scipy import stats

def drop_numerical_outliers(df, z_thresh=3):
    # Constrains will contain `True` or `False` depending on if it is a value below
    constrains = df.select_dtypes(include=[np.number]).apply(lambda x: np.abs(stats
   # Drop (inplace) values set to be rejected
    df.drop(df.index[~constrains], inplace=True)
drop_numerical_outliers(df)
```

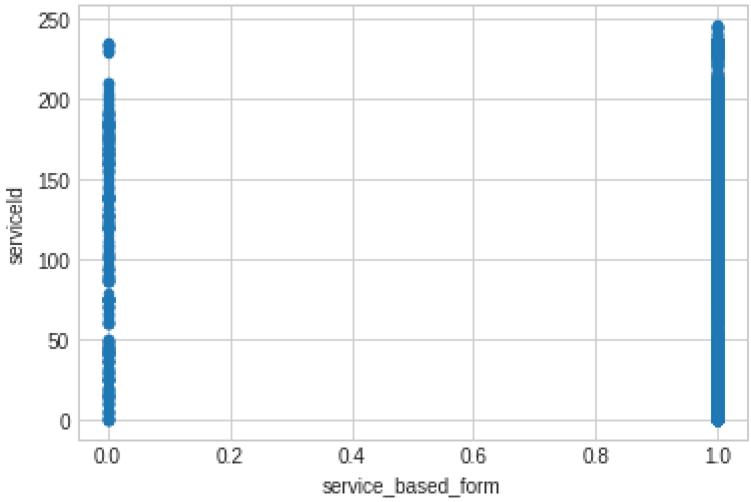
5. **Checking distribution of features**

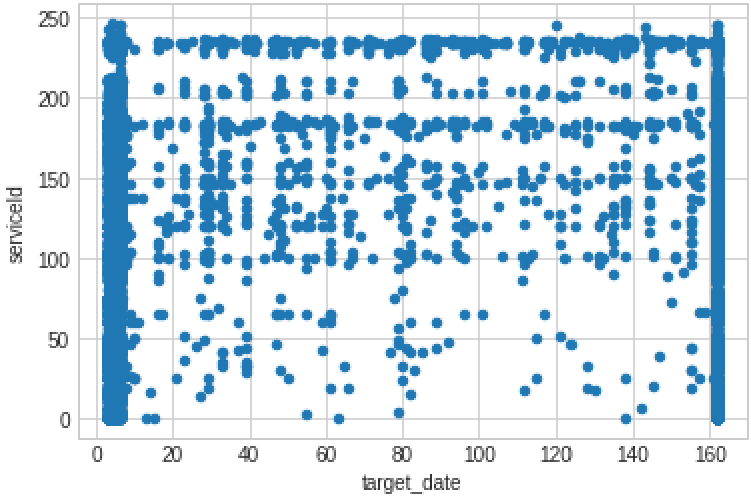
We need to check distribution of the numberical features and remove the one that has

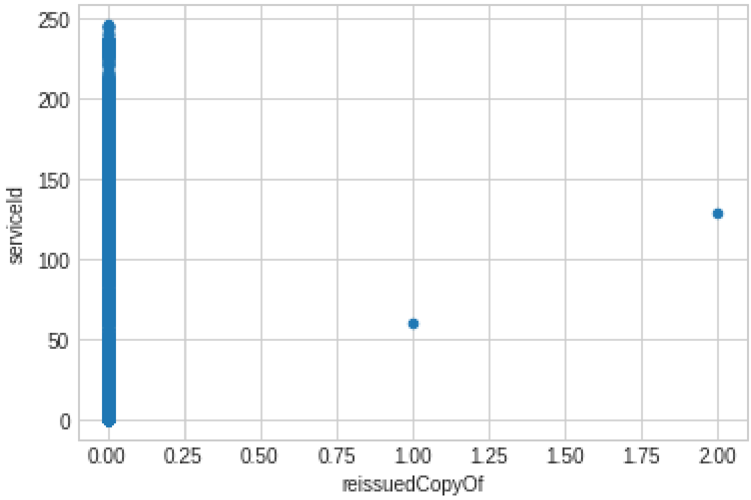
In [17]:

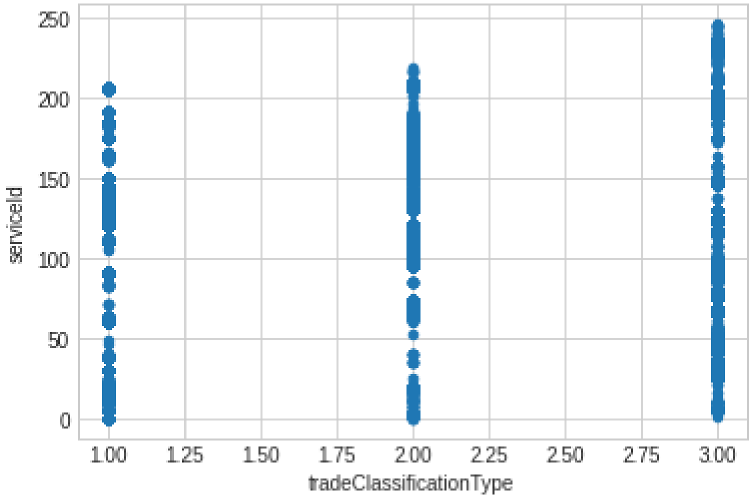
```
fig = plt.figure()

df.plot.scatter(x = 'service_based_form', y = 'serviceId')


df.plot.scatter(x = 'target_date', y = 'serviceId')


df.plot.scatter(x = 'reissuedCopyOf', y = 'serviceId')


df.plot.scatter(x = 'tradeClassificationType', y = 'serviceId')


df.plot.scatter(x = 'state', y = 'serviceId')


df.plot.scatter(x = 'stateText', y = 'serviceId')

plt.show()
```

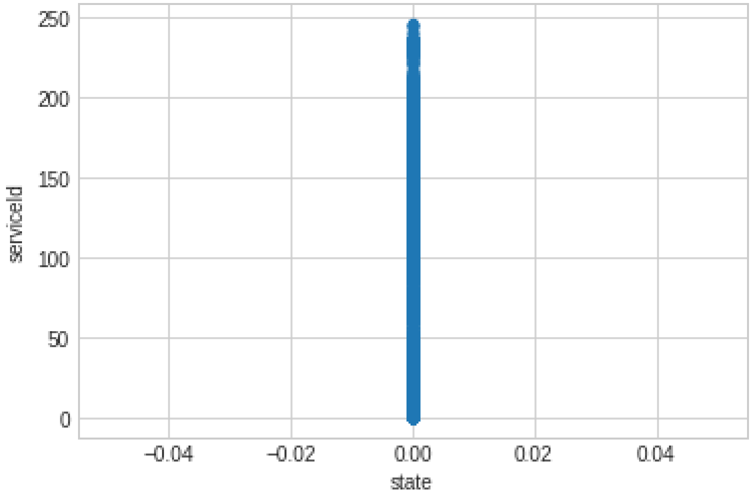
```
<Figure size 432x288 with 0 Axes>
```

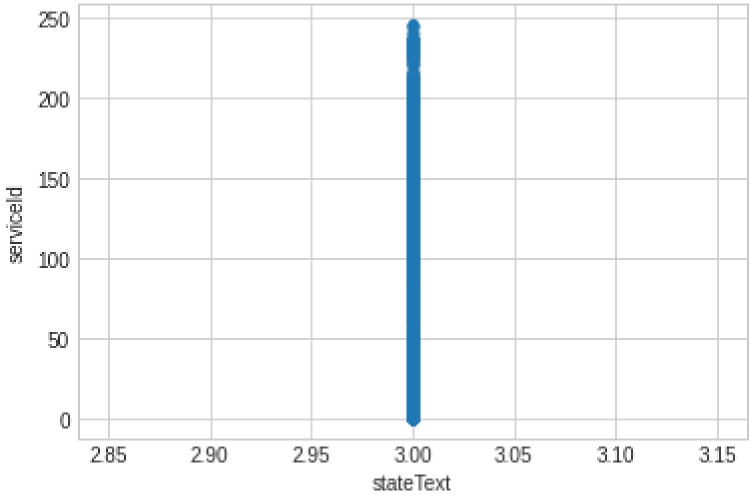



serviceId
target_date
0
50
100
150
200
250
0
20
40
60
80
100
120
140
160


serviceId
reissuedCopyOf
0
50
100
150
200
250
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


serviceId
tradeClassificationType
0
50
100
150
200
250
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


serviceId
state
0
50
100
150
200
250
−0.04
−0.02
0.00
0.02
0.04


serviceId
stateText
0
50
100
150
200
250
2.85
2.90
2.95
3.00
3.05
3.10
3.15

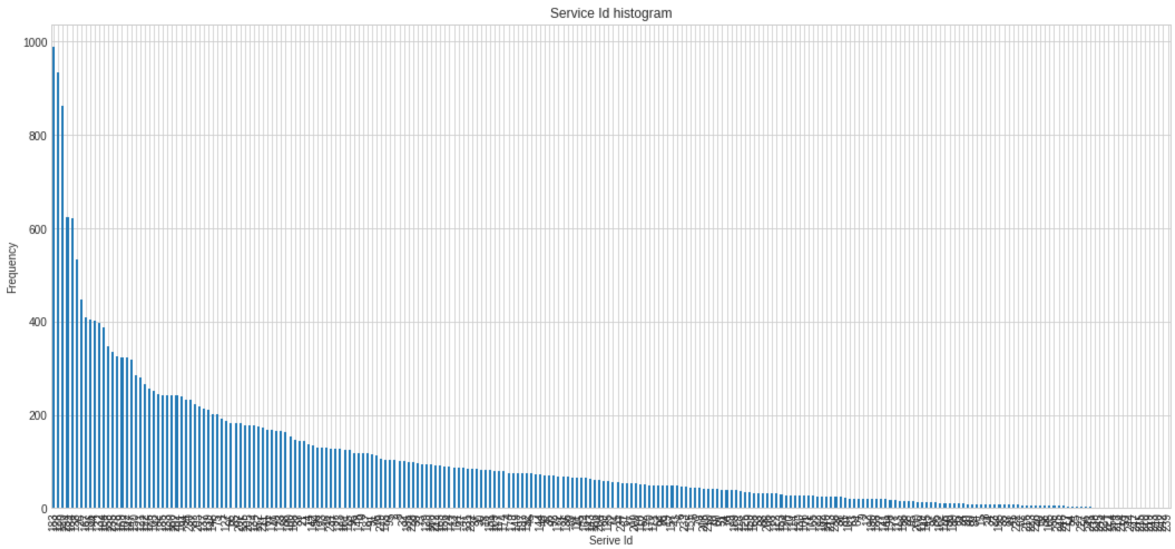
From above scatter plot we can see the below features does have a static distirbution, in other words, all of data points just take one value for them. These features cannot help us in classification models so we remove them.

- StateTest
- State
- reissuedCopyOf

6. **Checking Target feature to know if we have a balanced classification problem**

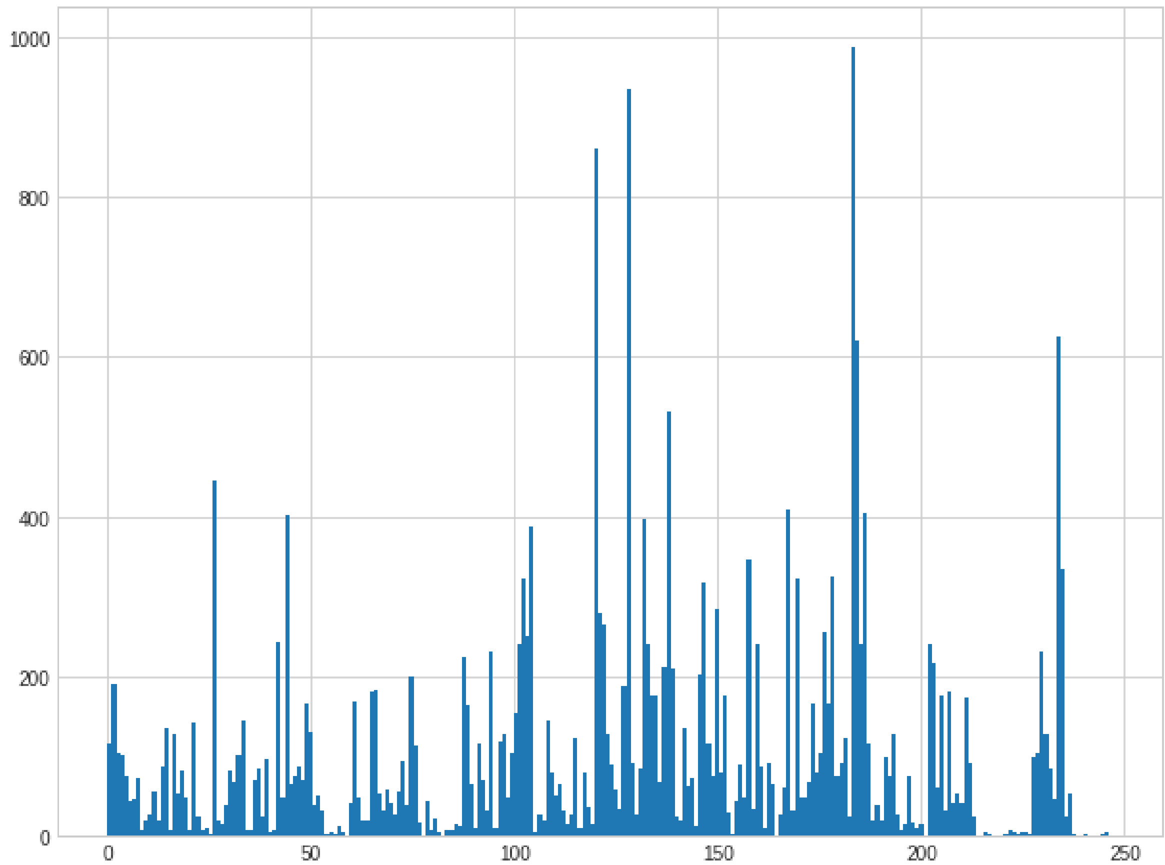
In [18]:

```
pd.value_counts(df['serviceId']).plot.bar(figsize=(18, 8))
plt.title('Service Id histogram')
plt.xlabel('Serive Id')
plt.ylabel('Frequency')
df['serviceId'].value_counts()
plt.show()
```



In [19]:

```
# plot of traget values
df['serviceId'] = df['serviceId'].astype(float)
fig=plt.figure(figsize=(8,6))
his=fig.add_axes([0,0,1,1])
plt.hist(df['serviceId'], bins = 250)
plt.show()
```



Challenge:

It seems we have an imbalanced classification problem. As it is clear in above plot, some classes have much more number of data than others. In other words, the distribution of examples across the classes is not equal.

Proposed solution

We over-sampling data to have more data points for miniority classes.

In [20]:

```
from imblearn.over_sampling import RandomOverSampler
# define dataset
X = df[['service_based_form', 'target_date','tradeClassificationType','title']].val
y = df['serviceId']

oversample = RandomOverSampler(sampling_strategy='minority')
# fit and apply the transform
X_over, y_over = oversample.fit_resample(X, y)
```

In [21]:

```
df_over = pd.DataFrame(data=np.column_stack((X_over,y_over)),columns=['service_base
df_over.head(3)
```

Out[21]:

	service_based_form	target_date	tradeClassificationType	title	serviceId
0	1	4	1	Badsanierung 39291 Lostau	138
1	1	162	3	4 m¬≤ Privatfl√§che pflastern, Unterbau erstel...	26
2	0	162	2	Dusche nachr√ºsten, Heizk√∂rper tauschen, Wass...	139

7. **Cleaning textual features**

Cleaning text (nlp) features

- **Remove all irrelevant characters such as any non alphanumeric characters**
- **Tokenize your text by separating it into individual words**
- **Remove stop words- stopwords are those german words which do not add much meaning to a sentence.They are very commonly used words and we do not required those words. So we can remove those stopwords**

In [22]:

```
# Remove irrelevant characters

df_over['title'] = df_over['title'].astype('str')
#df_over['description'] = df_over['description'].astype('str')
remove_characters = ["->", "≤", "¬" ,"¥" ,"  ","º", "√","§", "¬≤", "∂", "(" ,')', '
for chr in remove_characters:

    #df_over.description = df_over.description.str.replace(chr, '', regex=True)
    df_over.title = df_over.title.str.replace(chr, '', regex=True)

# check the data
df_over.head(3)
#df_test.shape
```

Out[22]:

	service_based_form	target_date	tradeClassificationType	title	serviceId
0	1	4	1	Badsanierung Lostau	138
1	1	162	3	m Privatflche pflastern Unterbau erstellen Ma...	26
2	0	162	2	Dusche nachrsten Heizkrper tauschen WasserAbwa...	139

In [23]:

```
# Tokenize the text
tokenizer=nltk.tokenize.RegexpTokenizer(r'\w+')

#df_over['description'] = df_over['description'].apply(lambda x:tokenizer.tokenize(
df_over['title'] = df_over['title'].apply(lambda x:tokenizer.tokenize(x))

# check the data
df_over.head(3)
#df_test.shape
```

Out[23]:

	service_based_form	target_date	tradeClassificationType	title	serviceId
0	1	4	1	[Badsanierung, Lostau]	138
1	1	162	3	[m, Privatflche, pflastern, Unterbau, erstelle...	26
2	0	162	2	[Dusche, nachrsten, Heizkrper, tauschen, Wasse...	139

In [24]:

```
len(stopwords.words('german'))
```

Out[24]:

232

In [25]:

```
#Remove stop words
def remove_stopwords(text):
    words = [w for w in text if w not in stopwords.words('german')]
    return words
df_over['title'] = df_over['title'].apply(lambda x : remove_stopwords(x))
#df['description'] = df['description'].apply(lambda x : remove_stopwords(x))

# check the data
df_over.head(3)
#df_test.shape
```

Out[25]:

	service_based_form	target_date	tradeClassificationType	title	serviceId
0	1	4	1	[Badsanierung, Lostau]	138
1	1	162	3	[m, Privatflche, pflastern, Unterbau, erstelle...	26
2	0	162	2	[Dusche, nachrsten, Heizkrper, tauschen, Wasse...	139

In [26]:

```
def combine_text(list_of_text):
    '''Takes a list of text and combines them into one large chunk of text.'''
    combined_text = ' '.join(list_of_text)
    return combined_text

df_over['title'] = df_over['title'].apply(lambda x : combine_text(x))
#df_over['description'] = df_over['description'].apply(lambda x : combine_text(x))


# check the data
df_over.head(3)
#df_test.shape
```

Out[26]:

	service_based_form	target_date	tradeClassificationType	title	serviceId
0	1	4	1	Badsanierung Lostau	138
1	1	162	3	m Privatflche pflastern Unterbau erstellen Mat...	26
2	0	162	2	Dusche nachrsten Heizkrper tauschen WasserAbwa...	139

Choose the model

Here we have a mixture of numerical and text features. We would like to use the information in both of them. In other words, we have a Multi-Data-Type Classification including both numerical and textual information. We suggest two models for implementing the classification:

Multi-Date Type classification with Keras

Creating a deep learning model in Keras that is capable of accepting multiple inputs, concatenating the two outputs and then performing classification using the aggregated input.

In this model, we created two submodeles. The first submodel will accept textual input from "title" features (to avoid complexity, we do not consider other textual features) in the form of text data. This submodel will consist of an input shape layer, an embedding layer, and an LSTM layer of 300 neurons. The second submodel will accept input in the form of meta information from the numerical columns. The second submodel also consist of three layers. An input layer and two dense layers.

The output from the LSTM layers of the first submodel and the output from the dense layer of the second submodel will be concatenated together and will be used as concatenated input to another dense layer with 10 neurons. Finally, the output dense layer will have 250 neuorns corresponding to each serviceId.

In [56]:

```
X = df_over.drop(['serviceId'], axis=1)

y = df_over['serviceId']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_st
```

In [57]:

```
X1_train = []
sentences = list(X_train["title"])
for sen in sentences:
    X1_train.append(sen)

X1_test = []
sentences = list(X_test["title"])
for sen in sentences:
    X1_test.append(sen)
```

In [58]:

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X1_train)

X1_train = tokenizer.texts_to_sequences(X1_train)
X1_test = tokenizer.texts_to_sequences(X1_test)


vocab_size = len(tokenizer.word_index) + 1

maxlen = 200

X1_train = pad_sequences(X1_train, padding='post', maxlen=maxlen)
X1_test = pad_sequences(X1_test, padding='post', maxlen=maxlen)
```

In [30]:

```
embeddings_dictionary = dict()

glove_file = open('glove.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary[word] = vector_dimensions

glove_file.close()

embedding_matrix = zeros((vocab_size, 300))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

In [59]:

```
X2_train = X_train[[ 'service_based_form', 'target_date','tradeClassificationType']
X2_test = X_test[['service_based_form', 'target_date','tradeClassificationType']].v
```

In [60]:

```
X2_train = X2_train.astype(int)
X2_test = X2_train.astype(int)
y_train = y_train.astype(int)
y_test = y_test.astype(int)
```

Feature scaling

As our numerical features have different scales, we need ot standardize them into the fixed range.

In [33]:

```
scaler = StandardScaler()
X2_train = scaler.fit_transform(X2_train)
X2_test = scaler.fit_transform(X2_test)
```

In [34]:

```
input_1 = Input(shape=(maxlen,))


input_2 = Input(shape=(3,))
```

In [35]:

```
embedding_layer = Embedding(vocab_size, 300, weights=[embedding_matrix], trainable=

LSTM_Layer_1 = LSTM(300)(embedding_layer)


dense_layer_1 = Dense(300, activation='relu')(input_2)
dense_layer_2 = Dense(300, activation='relu')(dense_layer_1)

concat_layer = Concatenate()([LSTM_Layer_1, dense_layer_2])
dense_layer_3 = Dense(300, activation='relu')(concat_layer)
output = Dense(250, activation='softmax')(dense_layer_3)
model = Model(inputs=[input_1, input_2], outputs=output)


model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam',  metric
```

In [64]:

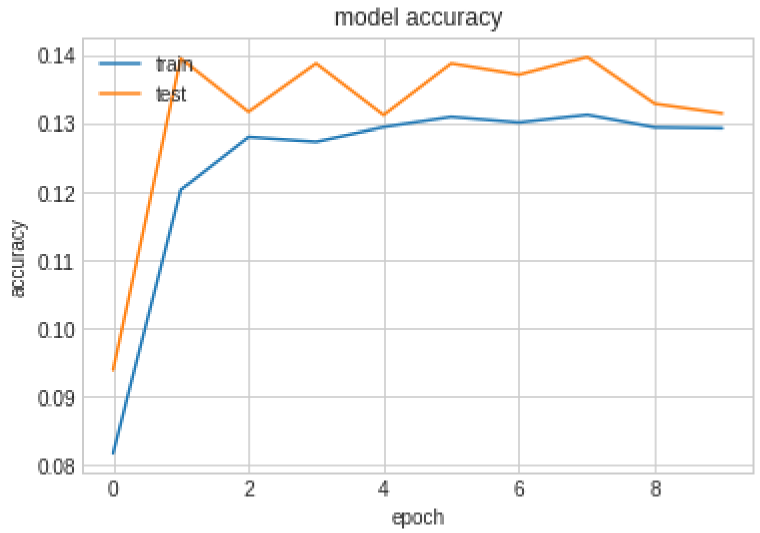
```
history = model.fit(x=[X1_train, X2_train], y=y_train, batch_size=128, epochs=10, v
```

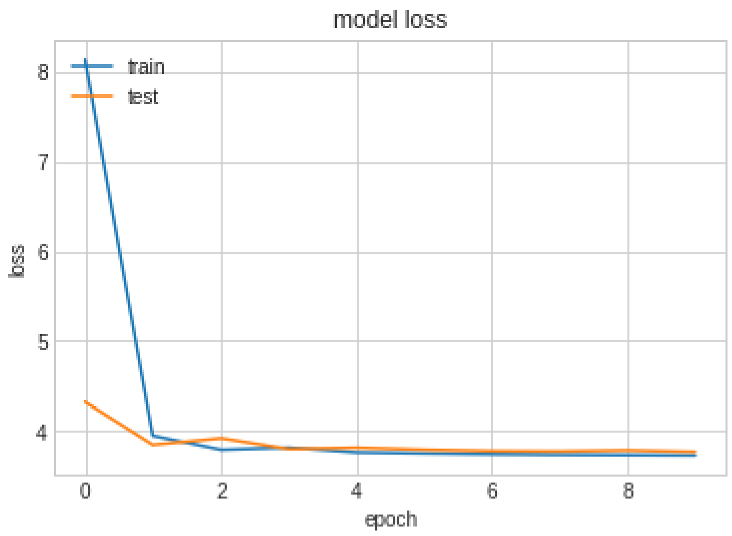
```
Epoch 1/10
133/133 [==============================] - 399s 3s/step - loss: 8.1389
- acc: 0.0818 - val_loss: 4.3249 - val_acc: 0.0939
Epoch 2/10
133/133 [==============================] - 412s 3s/step - loss: 3.9492
- acc: 0.1204 - val_loss: 3.8482 - val_acc: 0.1396
Epoch 3/10
133/133 [==============================] - 417s 3s/step - loss: 3.7931
- acc: 0.1281 - val_loss: 3.9206 - val_acc: 0.1318
Epoch 4/10
133/133 [==============================] - 407s 3s/step - loss: 3.8169
- acc: 0.1274 - val_loss: 3.8008 - val_acc: 0.1389
Epoch 5/10
133/133 [==============================] - 412s 3s/step - loss: 3.7630
- acc: 0.1295 - val_loss: 3.8154 - val_acc: 0.1313
Epoch 6/10
133/133 [==============================] - 406s 3s/step - loss: 3.7527
- acc: 0.1310 - val_loss: 3.7936 - val_acc: 0.1389
Epoch 7/10
133/133 [==============================] - 405s 3s/step - loss: 3.7439
- acc: 0.1302 - val_loss: 3.7783 - val_acc: 0.1372
Epoch 8/10
133/133 [==============================] - 406s 3s/step - loss: 3.7378
- acc: 0.1313 - val_loss: 3.7754 - val_acc: 0.1398
Epoch 9/10
133/133 [==============================] - 399s 3s/step - loss: 3.7340
- acc: 0.1295 - val_loss: 3.7860 - val_acc: 0.1330
Epoch 10/10
133/133 [==============================] - 405s 3s/step - loss: 3.7307
- acc: 0.1294 - val_loss: 3.7698 - val_acc: 0.1315
```

Make the prediction

In [66]:

```
import matplotlib.pyplot as plt

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```





Summarising Results

We do not get the good accuracy in prediction results. There maybe some reasons for it:

- For huge complexity, we had to run just for 10 iterations. It is possible that accuracy will be increased as the number of iterations increase.
- We just consider one textual feature in generating our model, still there is a probability that with making more complex our network with having other textual features, we can increase the accuracy.
- As we have 250 different classes, and we do not have a balanced dataset, make increasing the number of sample data points can help to a better accuracy