Capstone Project

Network Events - Prediction of outages

Rakesh Auplish – DSI+ - May 2018

Business Problem

- A telecom company is on a journey to enhance customer experience and putting customer first.
- Top priority items are network uptime and network quality of service.
- The aspiration is to be proactive in network maintenance.
- A statistical model for predicting network outages and their severity is required to facilitate proactive maintenance.

Executive Summary

Objective

- Prediction of 3 class outcome of an incident on the telephone network. Develop a machine learning model to perform automatic predictions.
  - Possible outcomes – 0: No disruption, 1: Momentary glitch, 2: Service disruption

Features / Metrics

- 5 features used for prediction
  - Location, Logfeature and Volume, Event Type, Resource, Incident Severity

Methodology

- 5 classification estimators used to get accuracy scores for objective evaluation
- Derived best hyperparameters for each estimator using *sklearn.model_selection.RandomisedSearchCV*
- 4 modelling scenarios used to assess impact of other factors
  - Use hyperparameters for optimization, Use default parameters, Exclude feature
- 20 accuracy scores compared

Assumptions

- Data definitions and business meaning of data assumed
  - No metadata / data dictionary provided
- Relationships between data entities assumed from profiling data

Risks

- Assumptions about business meaning could be wrong
- An incorrect inference of relationship between data entities could have a profound impact on prediction accuracy

Accuracy Results

Baseline Accuracy 64.82%

- GradientBoost Classifier is the best estimator in all scenarios
  - 76.48% accuracy for predictions with hyperparameters
  - 76.16% accuracy for 10 iterations with default parameters
- The model performed very well on a single run
- Suitable for real-time predictions on events as they are reported

Recommendation

- **Recommend GradientBoost Classifier for implementation**

Data Dictionary

Entity	Logs		
Description	Periodic information provided by equipment sensors connected to the network relating to state of network		
Attributes	Id	integer	Unique Identifier for a log
	Location	object	Id for the location of the equipment (no description)
	Fault Severity	object	Resultant severity
			3 classes *(0 - No disruption, 1 - Momentary glitch, 2 - Total disruption)*
Comments	Two data sets have been provided		
	train		*dataset for training the models, contains fault_severity*
	test		*dataset for testing the models*

Entity	Events		
Description	Type of event reported by equipment sensors. Multiple event types can be associated to a Log		
Attributes	Id	integer	Unique Identifier for a log
	Event_type	object	Id for event type(no description)
			Multiple Classes - values like 11, 15, 20, 7

Comment One data set has been provided

Entity Resources

Description Type of resource providing information to a log. Multiple resources can be associated to a Log

Attributes

Id	integer	Unique Identifier for a log
resource_type	object	Id for a resource (no description)
		Multiple classes - v*alues: 1,2,3,4,5,6,7,8,9,10*

Comment One data set has been provided

Entity Severity Type

Description Type of severity for the log. One severity associated to a Log

Attributes

Id	integer	Unique Identifier for a log
severity_type	object	Id for the severity (no description)
		Multiple classes - values: 1, 2, 3, 4, 5

Comment One data set has been provided

Entity Log feature

Description Feature(s) associated with a log. Attribute volume gives the intensity of the feature. More than one log feature associated to a Log

Attributes

Id	integer	Unique Identifier for a log
log_feature	object	Id for the log feature (no description)
		Multiple classes - values like 68, 172, 56, 193
volume	integer	Intensity of the log feature
		(Values range from 1 to 1310)

Comment One data set has been provided

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sys
import time
```

Load Data

Define function to convert values in column from string to integer. The function is called as parameter "converters" in pd.read_csv .

This paramter requires a dictionary in format {'column name':function name}. For each row that is loaded, <function name> is called

for the corresponding <column name> and the column value is passed to the function.

All csv files have column values in the format <column name>n . Eg: event_type 1, event_type 2 ... event_type n

The function splits the string by space into two elements, eg: [0] event_type and [1] 1 and converts element [1] to int

In [3]:
```
#
def str_num(string):
    return(int(string.split(' ')[1]))
#
event = pd.read_csv('./data/event_type.csv', converters={'event_type':str_num})
logfeature = pd.read_csv('./data/log_feature.csv', converters={'log_feature':str_
resource = pd.read_csv('./data/resource_type.csv', converters={'resource_type':st
severity = pd.read_csv('./data/severity_type.csv', converters={'severity_type':st
train = pd.read_csv('./data/train.csv', converters={'location':str_num})
test = pd.read_csv('./data/test.csv', converters={'location':str_num})
```

DISCOVER DATA

Profile Data

In [4]:
```
###############################################################################
# Check sizes of data frames
###############################################################################
print('Dataframe event - number of rows columns', event.shape)
print('Dataframe logfeature - number of rows columns', logfeature.shape)
print('Dataframe resource - number of rows columns', resource.shape)
print('Dataframe severity - number of rows columns', severity.shape)
print('Dataframe train - number of rows columns', train.shape)
print('Dataframe test - number of rows columns', test.shape)
```

```
Dataframe event - number of rows columns (31170, 2)
Dataframe logfeature - number of rows columns (58671, 3)
Dataframe resource - number of rows columns (21076, 2)
Dataframe severity - number of rows columns (18552, 2)
Dataframe train - number of rows columns (7381, 3)
Dataframe test - number of rows columns (11171, 2)
```

In [5]:
```
###############################################################################
# Check Number of Unique IDs in each dataframe.
###############################################################################
print('Dataframe event - unique IDs', event.id.nunique())
print('Dataframe logfeature - unique IDs', logfeature.id.nunique())
print('Dataframe resource - unique IDs', resource.id.nunique())
print('Dataframe severity - unique IDs', severity.id.nunique())
print('Dataframe train - unique IDs', train.id.nunique())
print('Dataframe test - unique IDs', test.id.nunique())
```

```
Dataframe event - unique IDs 18552
Dataframe logfeature - unique IDs 18552
Dataframe resource - unique IDs 18552
Dataframe severity - unique IDs 18552
Dataframe train - unique IDs 7381
Dataframe test - unique IDs 11171
```

In [7]:
```
###############################################################################
# Total number of rows and unique Log IDs in train + test is 18552
# Column fault_severity is the Target Variable containing the predicted severity
# Number of Log IDs is 18552 in all other datasets
###############################################################################
```

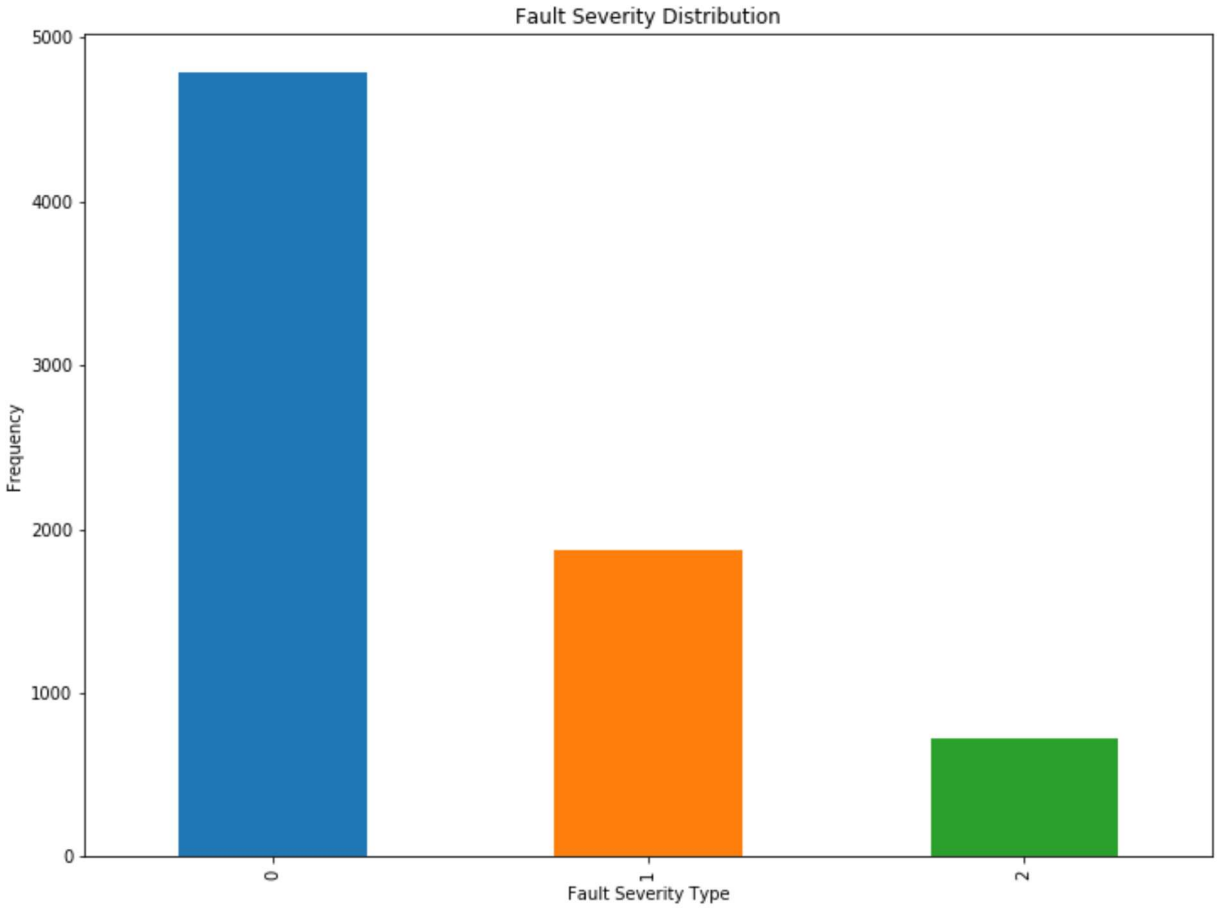
In [6]:
```
###############################################################################
# Check if IDs and Location are mutually exclusive between train and test
###############################################################################
trainID = set(train.id)
trainloc = set(train.location)
testID = set(test.id)
testloc = set(test.location)
IDu = trainID & testID
LOCu = trainloc & testloc
print('Number of Repeat IDs', len(IDu))
print('Number of Repeat Locations', len(LOCu))
```

```
Number of Repeat IDs 0
Number of Repeat Locations 842
```

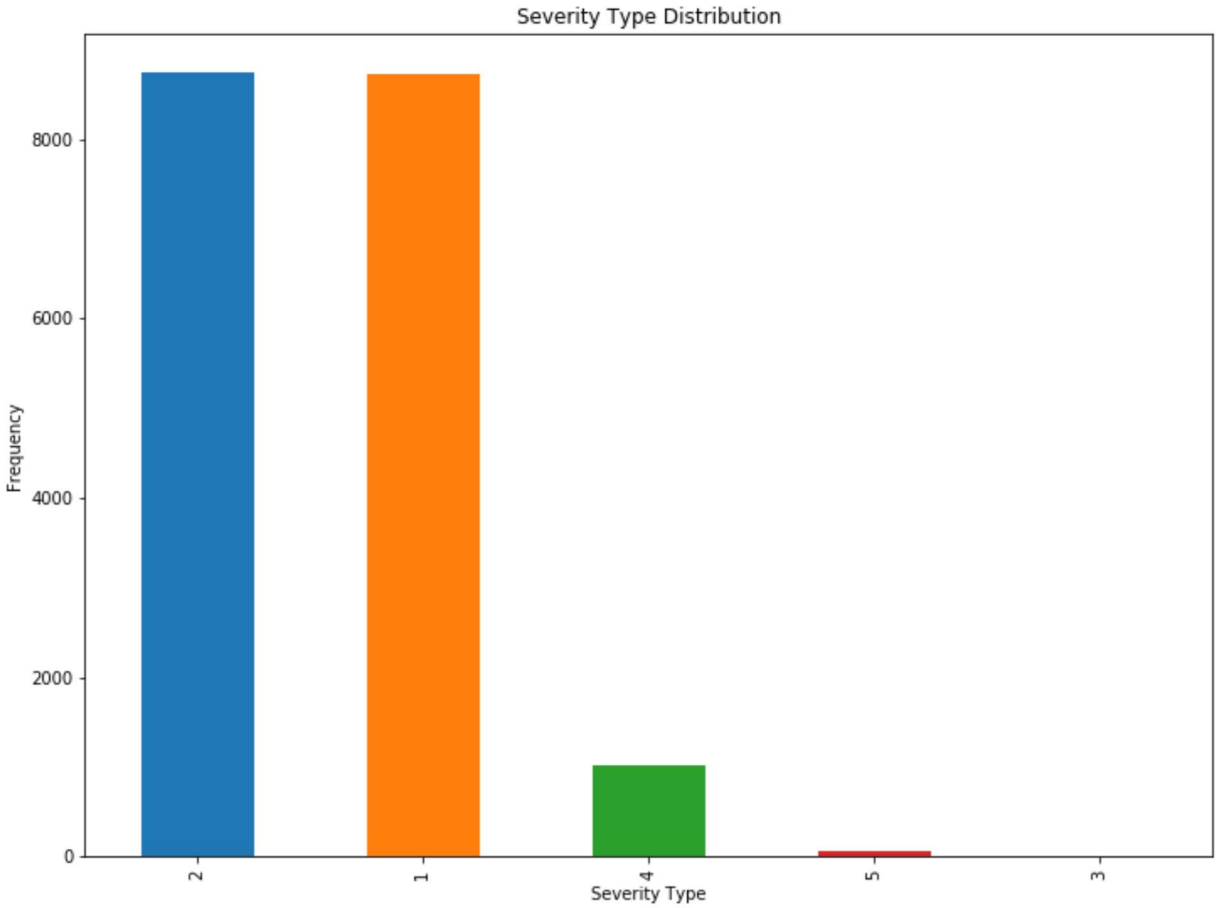
In [7]:
```
###############################################################################
# Combine train and test dataframes to analyse location
###############################################################################
train['df'] = 'train'
test['df'] = 'test'
test['fault_severity'] = ''
combine = pd.concat([train,test])
```

Visualise Data Distributions

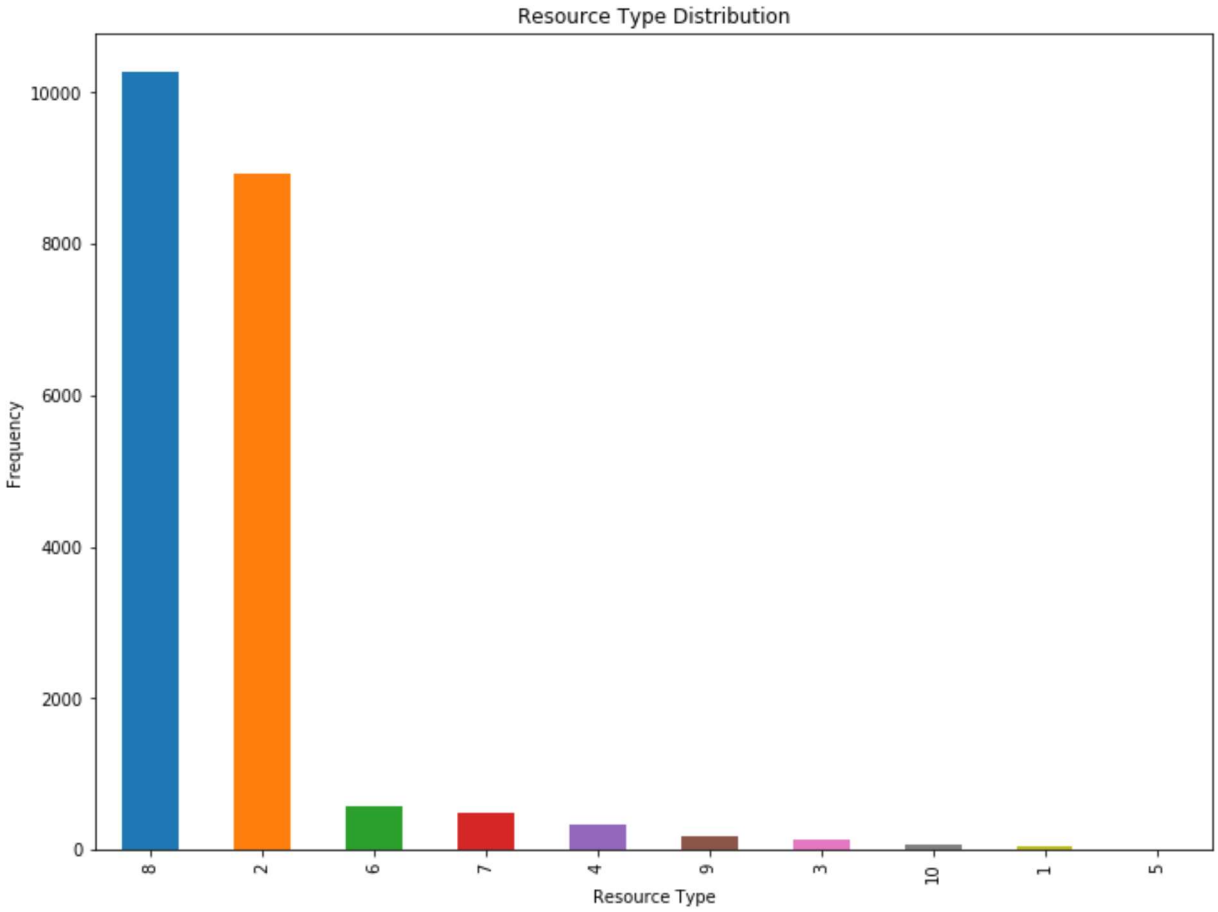
In [26]:
```
fig, ax = plt.subplots(figsize=(12,9))
tr = train.fault_severity.value_counts().plot(kind='bar', title='Fault Severity D
plt.ylabel('Frequency')
plt.xlabel('Fault Severity Type')
plt.show()
```



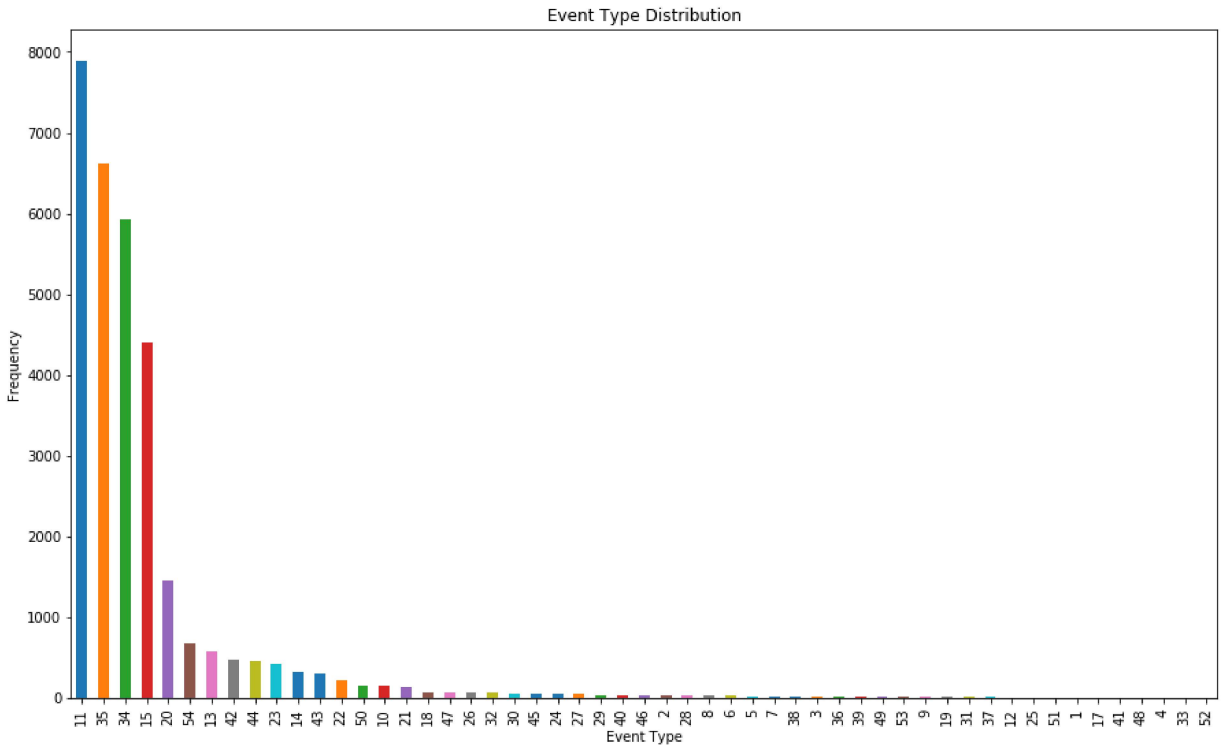
In [27]:
```
fig, ax = plt.subplots(figsize=(12,9))
se = severity.severity_type.value_counts().plot(kind='bar', title='Severity Type
plt.ylabel('Frequency')
plt.xlabel('Severity Type')
plt.show()
```



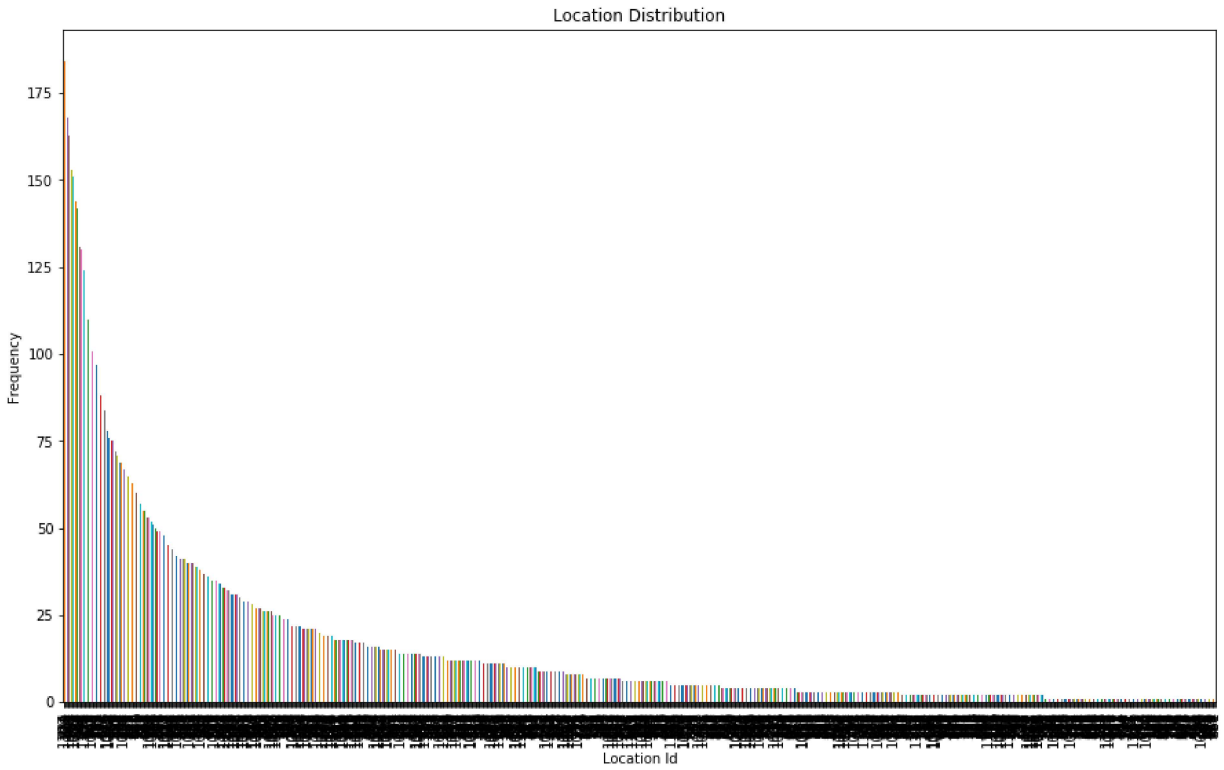
In [28]:
```
fig, ax = plt.subplots(figsize=(12,9))
re = resource.resource_type.value_counts().plot(kind='bar', title='Resource Type
plt.ylabel('Frequency')
plt.xlabel('Resource Type')
plt.show()
```



In [29]:
```
fig, ax = plt.subplots(figsize=(15,9))
ev = event.event_type.value_counts().plot(kind='bar', title='Event Type Distribut
plt.ylabel('Frequency')
plt.xlabel('Event Type')
plt.show()
```



In [30]:
```
fig, ax = plt.subplots(figsize=(15,9))
lts = combine.location.value_counts().plot(kind='bar', title='Location Distributi
plt.ylabel('Frequency')
plt.xlabel('Location Id')
plt.show()
```

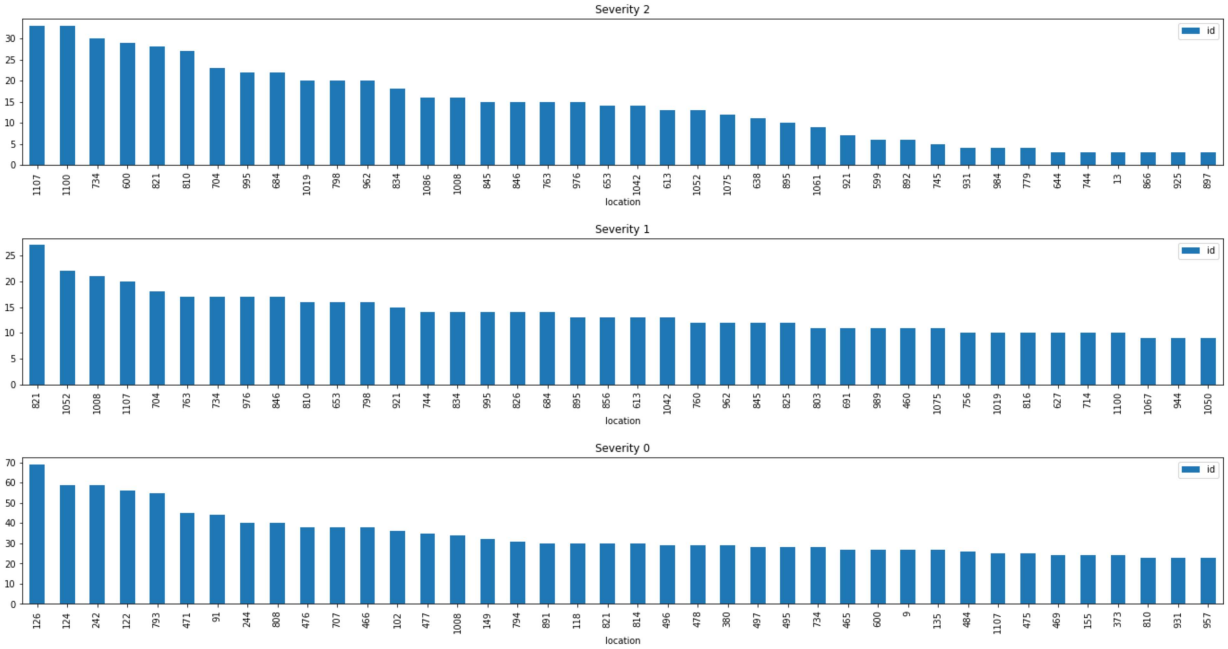


In [31]:
```
###############################################################################
# Profile frequency of locations and show the top 30. A count of 100 is the best
###############################################################################
combine.location.value_counts().head(30)
```

Out[31]:
```
1107    184
734     184
821     183
126     177
810     168
704     163
1008    162
653     161
684     153
793     151
1019    148
1052    146
124     144
834     142
600     136
122     134
962     131
763     130
242     129
798     124
995     124
1100    123
244     121
976     114
613     110
102     109
846     109
1042    106
845     101
808     100
Name: location, dtype: int64
```

In [41]:
```
###############################################################################
# Plot location and fault_severity
###############################################################################
fig, ax = plt.subplots(nrows=3, ncols=1,figsize=(24,12))
fig.subplots_adjust(hspace=.5)
train[['id','location']][train.fault_severity==2].groupby(['location']).count().s
.plot(kind = 'bar',ax=ax[0],  title = 'Severity 2')
train[['id','location']][train.fault_severity==1].groupby(['location']).count().s
.plot(kind = 'bar',ax=ax[1],  title = 'Severity 1')
train[['id','location']][train.fault_severity==0].groupby(['location']).count().s
.plot(kind = 'bar',ax=ax[2],  title = 'Severity 0')
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x2b2e227b668>



Load data into SQLLite3 database for analysis

In [12]:
```
# Establish database connection
import sqlite3
db = './Database/Testra.sqlite'
conn = sqlite3.connect(db)
c = conn.cursor()
```

In [13]:
```
# Load data from DataFrames
train.to_sql(name = 'train', con = conn, if_exists = 'replace', index = False)
test.to_sql(name = 'test', con = conn, if_exists = 'replace', index = False)
severity.to_sql(name = 'severity', con = conn, if_exists = 'replace', index = Fal
resource.to_sql(name = 'resource', con = conn, if_exists = 'replace', index = Fal
event.to_sql(name = 'event', con = conn, if_exists = 'replace', index = False)
logfeature.to_sql(name = 'logfeature', con = conn, if_exists = 'replace', index =
```

Data Engineering

Resource Type

- Bin classes in resource type. Class 8 and Class 2 have high frequencies. Club all others into OTH
- Binarise the classes
- Dataframe has got multiple rows for some IDs. To get 1 row for each ID we need to groupby ID. Since the resource_type values are mutually exclusive for each ID
  and they have been binarised doing a sum on the binarised columns will not affect the values

Event Type

- Bin classes in event type. Classes 11,35,34,15,20 have high frequencies. Club all others into OTH
- Binarise the classes
- Dataframe has got multiple rows for some IDs. To get 1 row for each ID we need to groupby ID. Since the resource_type values are mutually exclusive for each ID
  and they have been binarised doing a sum on the binarised columns will not affect the values

Locations

- Bin Locations - if count of logs < 100 then OTH
- Binarise locations

Logfeatures

- Logfeature has one more rows for each Log. Each feature has an associated Volume with it. Transpose the rows into columns and preserve the volume

Severity Type

- Binarise Severity Type classes

In [5]:
```
# Bin classes in resource type. Class 8 and Class 2 have high frequencies. Club a
def map_res(r):
    if r == 8:
        x='RT8'
    elif r == 2:
        x='RT2'
    else:
        x='OTH'
    return x
resource['resource_type'] = resource['resource_type'].map(map_res)
```

In [6]:
```
# Binarise the resource_type classes.
resd = pd.get_dummies(resource,columns=['resource_type'])
# Dataframe has got multiple rows for some IDs. To get 1 row for each ID we need
# are mutually exclusive for each ID and they have been binarised doing a sum on
resource_id = resd.groupby('id').sum().reset_index()
```

In [7]:
```
# Bin classes in event type. Classes 11,35,34,15,20 have high frequencies. Club a
def map_event(e):
    if e in [11,35,34,15,20]:
        x = 'ET' + str(e)
    else:
        x = 'OTH'
    return x
event['event_type'] = event['event_type'].map(map_event)
```

In [8]:
```
# Binarise the event_type classes
eved = pd.get_dummies(event,columns=['event_type'])
# Dataframe may have multiple rows for some IDs. To get 1 row for each ID we need
# are mutually exclusive for each ID and they have been binarised doing a sum on
event_id = eved.groupby('id').sum().reset_index()
```

In [9]:
```
# Bin Locations - if number of logs < 100 then OTH
# Count of locations in a dictionary
d = dict(combine.location.value_counts())
def map_loc(l):
    if d[l] >= 100:
        x = str(l)
    else:
        x = 'OTH'
    return x
combine['location'] = combine.location.map(map_loc)
```

In [10]:
```
# Binarise locations
combine_id = pd.get_dummies(combine,columns=['location'])
```

In [20]:
```
# TRY THIS OPTION IF THE MODELS DO NOT RUN WITH THE LARGER NUMBER OF COLUMNS
# Get sum of volumes and mean of volumes for each ID from logfeature and create a
# x= logfeature.groupby(['id']).agg({'volume':'sum'}).reset_index()
# y = logfeature.groupby(['id']).agg({'volume':'mean'}).reset_index()
# x.rename(columns={'volume':'volsum'},inplace=True)
# y.rename(columns={'volume':'volavg'},inplace=True)
# logfeature_id = pd.merge(x,y, on='id')
```

In [11]:
```
# LOGFEATURE HAS REPEATING ROWS FOR IDs AND HAS VOLUME ASSOCIATED WITH EACH LOGFE
# INTO COLUMNS AND PRESERVES THE VOLUME VALUES AS WELL
lfdict = {}                                          # Initialise a dictionar
for row in logfeature.itertuples():                  # for each row in logfea
    if row.id not in lfdict:                         # create an entry for ea
        lfdict[row.id] = {}                          # Initialise a sub-dicti
    if row.log_feature not in lfdict[row.id]:        # Create an entry for ea
        lfdict[row.id][row.log_feature] = row.volume # Update the volume
len(lfdict)
########################
# Create list of column names for dataframe. There are 386 unique values for log_
cols = ['id']
for i in range(1,387):
    cols.append('log_feature' + str(i))
########################
# Create two dimensional array for storing values from dictionary and initalise w
lfarray = np.zeros((18552,387))
rownum = 0                          # Initialise row number. Each row is for an ID
for id,fdict in lfdict.items():  # Outer dictionary - get id and inner dictionary
    lfarray[rownum,0] = id       # ID in the first column
    for lf,vol in fdict.items():   # Get log feature and volume from inner dictio
        lfarray[rownum,lf] = vol # Store volume in the "column" for the feature
    rownum += 1                  # Increment rownum for the next ID
len(lfarray)
########################
# Create datafrome from the array
logfeature_id = pd.DataFrame(data=lfarray, columns=cols,dtype=np.int)
```

In [12]:
```
# Binarise severity_type
severity_id = pd.get_dummies(severity,columns=['severity_type'])
```

MERGE DATA FOR MODELLING

In [13]:
```
# Check shapes of separate dataframes
print('DF combine', combine_id.shape)
print('DF severity', severity_id.shape)
print('DF resource', resource_id.shape)
print('DF logfeature', logfeature_id.shape)
```

```
DF combine (18552, 35)
DF severity (18552, 6)
DF resource (18552, 4)
DF logfeature (18552, 387)
```

In [14]:
```
# Option 1 all features
dfmerged = pd.merge(combine_id,event_id,on='id')
dfmerged = pd.merge(dfmerged,severity_id,on='id')
dfmerged = pd.merge(dfmerged,resource_id,on='id')
dfmerged = pd.merge(dfmerged,logfeature_id,on='id')
```

In [25]:
```
dfmerged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18552 entries, 0 to 18551
Columns: 435 entries, df to log_feature386
dtypes: int32(386), int64(1), object(2), uint8(46)
memory usage: 28.7+ MB
```

In [15]:
```
# Option 2 all features excluding logfeature
dfmergesml = pd.merge(combine_id,event_id,on='id')
dfmergesml = pd.merge(dfmergesml,severity_id,on='id')
dfmergesml = pd.merge(dfmergesml,resource_id,on='id')
```

In [20]:
```
dfmergesml.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18552 entries, 0 to 18551
Data columns (total 49 columns):
df                   18552 non-null object
fault_severity       18552 non-null object
id                   18552 non-null int64
location_1008        18552 non-null uint8
location_1019        18552 non-null uint8
location_102         18552 non-null uint8
location_1042        18552 non-null uint8
location_1052        18552 non-null uint8
location_1100        18552 non-null uint8
location_1107        18552 non-null uint8
location_122         18552 non-null uint8
location_124         18552 non-null uint8
location_126         18552 non-null uint8
location_242         18552 non-null uint8
location_244         18552 non-null uint8
location_471         18552 non-null uint8
location_600         18552 non-null uint8
location_613         18552 non-null uint8
location_653         18552 non-null uint8
location_684         18552 non-null uint8
location_704         18552 non-null uint8
location_734         18552 non-null uint8
location_763         18552 non-null uint8
location_793         18552 non-null uint8
location_798         18552 non-null uint8
location_808         18552 non-null uint8
location_810         18552 non-null uint8
location_821         18552 non-null uint8
location_834         18552 non-null uint8
location_845         18552 non-null uint8
location_846         18552 non-null uint8
location_962         18552 non-null uint8
location_976         18552 non-null uint8
location_995         18552 non-null uint8
location_OTH         18552 non-null uint8
event_type_ET11      18552 non-null uint8
event_type_ET15      18552 non-null uint8
event_type_ET20      18552 non-null uint8
event_type_ET34      18552 non-null uint8
event_type_ET35      18552 non-null uint8
event_type_OTH       18552 non-null uint8
severity_type_1      18552 non-null uint8
severity_type_2      18552 non-null uint8
severity_type_3      18552 non-null uint8
severity_type_4      18552 non-null uint8
severity_type_5      18552 non-null uint8
resource_type_OTH    18552 non-null uint8
resource_type_RT2    18552 non-null uint8
resource_type_RT8    18552 non-null uint8
dtypes: int64(1), object(2), uint8(46)
memory usage: 1.4+ MB
```

In [16]:
```
# Split back into train and test datasets with new classes for location
train_id = dfmerged[dfmerged['df'] == 'train']
test_id = dfmerged[dfmerged['df'] == 'test']
train_id.drop(['df'], axis=1,inplace=True)
test_id.drop(['df'],axis=1,inplace=True)
```

```
C:\Users\Vinita Auplish\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  after removing the cwd from sys.path.
C:\Users\Vinita Auplish\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  """
```

In [21]:
```
# Split back into train and test datasets with new classes for location
train_sml = dfmergesml[dfmergesml['df'] == 'train']
test_sml = dfmergesml[dfmergesml['df'] == 'test']
train_sml.drop(['df'], axis=1,inplace=True)
test_sml.drop(['df'],axis=1,inplace=True)
```

```
C:\Users\Vinita Auplish\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  after removing the cwd from sys.path.
C:\Users\Vinita Auplish\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  """
```

In [18]:
```
train_sml.head()
```

Out[18]:

	fault_severity	id	location_1008	location_1019	location_102	location_1042	location_1052	l
0	1	14121	0	0	0	0	0	
1	0	9320	0	0	0	0	0	
2	1	14394	0	0	0	0	0	
3	1	8218	0	0	0	0	0	
4	0	14804	0	0	0	0	0	

5 rows × 434 columns

In [22]:
```
# Check dataframes - full and small
print('Dataframe train - number of rows columns', train_id.shape)
print('Dataframe test - number of rows columns', test_id.shape)
print('Dataframe train - number of rows columns', train_sml.shape)
print('Dataframe test - number of rows columns', test_sml.shape)
```

```
Dataframe train - number of rows columns (7381, 434)
Dataframe test - number of rows columns (11171, 434)
Dataframe train - number of rows columns (7381, 48)
Dataframe test - number of rows columns (11171, 48)
```

In [23]:
```
# Write full dataset to CSV files
dfmerged.to_csv('./Data/dfmerged.csv', index=False)
train_id.to_csv('./Data/train_id.csv',index=False)
test_id.to_csv('./Data/test_id.csv',index=False)
```

In [25]:
```
# Write small dataset to CSV files
dfmergesml.to_csv('./Data/dfmergesml.csv', index=False)
train_sml.to_csv('./Data/train_sml.csv',index=False)
test_sml.to_csv('./Data/test_sml.csv',index=False)
```

In [ ]:
```

```