Aim

1. Clean and Featurize data

In [1]:
```
import html
import os
import re
import string
import unicodedata
from datetime import datetime

import category_encoders as ce
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from nltk.corpus import stopwords
from pandarallel import pandarallel

# from gensim.corpora import Dictionary
# from gensim.models import TfidfModel
# from gensim.utils import simple_preprocess
# from sklearn.feature_extraction.text import TfidfVectorizer
from tqdm import tqdm
```

In [2]:
```
DATA_ROOT = "../data"

# os.makedirs(f"{DATA_ROOT}/train/features")
```

In [3]:
```
df_train = pd.read_pickle(f"{DATA_ROOT}/train/raw/data.pkl")
df_test = pd.read_pickle(f"{DATA_ROOT}/test/raw/data.pkl")
```

In [4]:
```
df_train.head(2)
```

Out[4]:

	ID	Source	TMC	Start_Time	Distance(mi)	Description	Side	City	C
0	A-2478859	Bing	0.0	2016-02-08 00:37:08	3.23	Between Sawmill Rd/Exit 20 and OH-315/Olentang...	R	Dublin	F
1	A-1	MapQuest	201.0	2016-02-08 05:46:00	0.01	Right lane blocked due to accident on I-70 Eas...	R	Dayton	Montg

2 rows × 38 columns

In [5]:
```
df_test.head(2)
```

Out[5]:

	ID	Source	TMC	Start_Time	Distance(mi)	Description	Side	City	County
0	A-3017746	Bing	0.0	2020-01-01 00:01:00	0.0	At Hampshire Rd/Exit 41 - Accident.	R	Westlake Village	Ventura
1	A-3017745	Bing	0.0	2020-01-01 00:02:00	0.0	At Sheep Creek Rd - Accident.	L	Phelan	San Bernardino

2 rows × 38 columns

In [6]:
```
df_train.columns
```

Out[6]:
```
Index(['ID', 'Source', 'TMC', 'Start_Time', 'Distance(mi)', 'Description',
       'Side', 'City', 'County', 'State', 'Zipcode', 'Timezone',
       'Airport_Code', 'Temperature(F)', 'Humidity(%)', 'Pressure(in)',
       'Visibility(mi)', 'Wind_Direction', 'Wind_Speed(mph)',
       'Weather_Condition', 'Amenity', 'Bump', 'Crossing', 'Give_Way',
       'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station', 'Stop',
       'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop', 'Sunrise_Sunse
t',
       'Civil_Twilight', 'Nautical_Twilight', 'Astronomical_Twilight',
       'Severity'],
      dtype='object')
```

Cleaning : `Wind_Direction`

In [7]:
```
df_train["Wind_Direction"].unique()
```

Out[7]:
```
array(['SW', 'Calm', 'SSW', 'WSW', 'WNW', 'NW', 'West', 'NNW', 'NNE',
       'South', 'W', 'North', 'Variable', 'SSE', 'SE', 'ESE', 'none',
       'East', 'NE', 'ENE', 'E', 'CALM', 'S', 'VAR', 'N'], dtype=object)
```

1. convert to upper case
2. replace VAR with variable

In [8]:
```
def clean_wind_direction(df):
    df["Wind_Direction"] = df["Wind_Direction"].str.upper()
    df["Wind_Direction"] = df["Wind_Direction"].apply(
        lambda x: "variable" if x == "VAR" else x
    )

    return df


df_train = clean_wind_direction(df_train)
df_test = clean_wind_direction(df_test)
```

In [9]:
```
print(sorted(df_train["Wind_Direction"].unique()))
```

```
['CALM', 'E', 'EAST', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NONE', 'NORT
H', 'NW', 'S', 'SE', 'SOUTH', 'SSE', 'SSW', 'SW', 'VARIABLE', 'W', 'WEST',
'WNW', 'WSW', 'variable']
```

In [10]:
```
print(sorted(df_test["Wind_Direction"].unique()))
```

```
['CALM', 'E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NONE', 'NW', 'S', 'S
E', 'SSE', 'SSW', 'SW', 'W', 'WNW', 'WSW', 'variable']
```

Cleaning : `Description`

In [18]:

```python
def rm_numbers(x):
    x = re.sub(r"[0-9]+", "", x)
    return x


def rm_html(x):
    x = html.unescape(x)
    x = BeautifulSoup(x).get_text()
    return x


def rm_url(x):
    x = re.sub("http*\S+", " ", x)
    return x


def rm_multiple_dots(x):
    x = re.sub(r"\.+", ". ", x)
    x = re.sub("\|+", ". ", x)
    return x


def rm_unicode(x):
    x = unicodedata.normalize("NFKD", x)
    return x


def rm_punctuation(x):
    x = re.sub("[%s]" % re.escape(string.punctuation.replace(".", "")), " ",
    return x


def rm_spaces(x):
    x = re.sub(" +", " ", x)
    return x


def rm_word(x, word):
    x = x.replace(word, "")
    return x


def clean_text(input_string):
    ss = input_string
    ss = rm_html(ss)
    ss = rm_url(ss)
    ss = rm_punctuation(ss)
    ss = rm_multiple_dots(ss)
    ss = rm_unicode(ss)
    ss = rm_spaces(ss)
    # ss = rm_numbers(ss)
    ss = rm_word(ss, "\n")
    ss = rm_word(ss, "\t")
    ss = ss.strip()

    return ss
```

In [20]:
```
pandarallel.initialize(verbose=True,)
df_train["Description"] = df_train["Description"].parallel_apply(
    lambda x: clean_text(x)
)
```

In [26]:
```
df_train["Description"].sample(5).values
```

Out[26]:
```
array(['Accident on US 101 Oregon Coast Hwy near Cedar St.',
       'Accident on Bingle Rd at Houston Rosslyn Rd.',
       'Accident on Six Forks Rd at Lead Mine Rd.',
       'Right hand shoulder blocked due to accident on I 210 Eastbound befor
e Exit 19 CA 2.',
       'Between VA 619 Exit 150 and VA 234 Exit 152 Accident.'],
      dtype=object)
```

In [24]:
```
pandarallel.initialize(verbose=True,)
df_test["Description"] = df_test["Description"].parallel_apply(lambda x: cle
```

In [27]:
```
df_test["Description"].sample(5).values
```

Out[27]:
```
array(['At Old Hiway Accident.',
       'Lane blocked due to accident on I 385 Northbound near Exit 34 Butler
Rd.',
       'Accident on MN 36 Westbound at CR 35 Hadley Ave.',
       'At Southwood Plantation Rd Accident.',
       'Right lane blocked due to accident on Sam Houston Tlwy Eastbound at
I 45 Gulf Fwy Exit 32.'],
      dtype=object)
```

Featurizing : `Description`

1. Get K keywords
2. create a binary vector of dimension K
3. if presence of word_x mark that dimension 1

Why this feature?

1. this will allow us to capture important words describing an accident
2. these words inturn might help in identifying severity

- YAKE github
- Reference - Key word extractor

YAKE is a lightweight, unsupervised automatic keyword extraction method that relies on statistical text features extracted from individual documents to identify the most relevant keywords in the text.

In [13]:
```
import yake

kw_extractor = yake.KeywordExtractor()
text = ".".join(
    df_train["Description"].sample(n=100_000).tolist()
)  # extracting from a sample as compute intensive process
language = "en"
max_ngram_size = 1
deduplication_threshold = 0.1
numOfKeywords = 1000
```

```python
custom_kw_extractor = yake.KeywordExtractor(
    lan=language,
    n=max_ngram_size,
    dedupLim=deduplication_threshold,
    top=numOfKeywords,
    features=None,
    stopwords=stopwords.words("english"),
)
keywords = custom_kw_extractor.extract_keywords(text)
```

In [28]:
```python
# The lower the score, the more relevant the keyword is.
keywords
```

Out[28]:
```
[('Accident', 6.169566081677733e-09),
 ('Northbound', 1.3069501235051463e-07),
 ('Hwy', 2.738028950102289e-06),
 ('ramp', 4.8401705088886265e-06),
 ('slow', 8.777147725501163e-06),
 ('Trl', 0.00014915096836162105),
 ('Mopac', 0.001457774067552258),
 ('Okeechobee', 0.0015743939528227214),
 ('Brookshire', 0.0016599371751449834),
 ('Huntington', 0.0029626324177345 76),
 ('NYS', 0.005783750908517111),
 ('Fuqua', 0.009709526119156922),
 ('Middlefield', 0.05507099513618 1314),
 ('JFK', 0.07611004069260154),
 ('Cedarhurst', 0.1031640040545067),
 ('57-56', 0.12423963791402771),
 ('PGBT.', 0.3325493080742511),
 ('Rhinecliff', 0.6395068253572433),
 ('Chavaneaux', 0.6552093622826601),
 ('Gibsonburg', 0.677368534171853)]
```

In [30]:
```python
pd.to_pickle(
    keywords, f"{DATA_ROOT}/train/keywords.pkl",
)
```

In [34]:
```python
# using top 15 words
keywords_list = [i[0] for i in keywords[:15]]
```

In [35]:
```python
keywords_list
```

Out[35]:
```
['Accident',
 'Northbound',
 'Hwy',
 'ramp',
 'slow',
 'Trl',
 'Mopac',
 'Okeechobee',
 'Brookshire',
 'Huntington',
 'NYS',
 'Fuqua',
 'Middlefield',
 'JFK',
 'Cedarhurst']
```

In [45]:
```python
fuzz.partial_ratio("hello world 2", "hello world")  # demo of partial_ratio
```

Out[45]:
```
100
```

In [57]:
```
def get_kw_vec(x, kw_list):
    vec = [fuzz.partial_ratio(i.lower(), x.lower()) for i in kw_list]
    vec = np.array(vec)
    vec = np.where(vec > 60, 1, 0).tolist()
    return vec
```

In [58]:
```
pandarallel.initialize(verbose=True,)
df_train["kw_vec"] = df_train["Description"].parallel_apply(
    lambda x: get_kw_vec(x, keywords_list)
)
```

In [59]:
```
pandarallel.initialize(verbose=True,)
df_test["kw_vec"] = df_test["Description"].parallel_apply(
    lambda x: get_kw_vec(x, keywords_list)
)
```

In [62]:
```
df_train["kw_vec"].sample(5).head()
```

Out[62]:
```
2428374    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
209306     [1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]
849060     [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
70600      [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1845646    [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Name: kw_vec, dtype: object
```

Featurizing : `Zipcode`

Reference encoding zipcode

Why breakdown zip code?

- zip codes are hierarchical in nature
- using first N digits of zipcode will give us some understanding of the region
- using next N digits will give understanding of next sub regions
- we also reduce the number of unique zipcodes this way
- this will enable model to learn better

In [64]:
```
df_train["zip_02"] = df_train["Zipcode"].str[:2]
df_test["zip_02"] = df_test["Zipcode"].str[:2]
```

In [65]:
```
df_train["zip_25"] = df_train["Zipcode"].str[2:5]
df_test["zip_25"] = df_test["Zipcode"].str[2:5]
```

Zip codes are of varying length that could be one of the features

`zip_len` could be one of the features

In [66]:
```
df_train["zip_len"] = df_train["Zipcode"].apply(len)
df_test["zip_len"] = df_test["Zipcode"].apply(len)
```

We can observe some compound zip codes like

1. 43068-3402
2. 93401-8325
3. 60607-3612

`is_compound` could be one of the boolean variables

In [67]:
```
df_train["zip_is_compound"] = df_train["Zipcode"].apply(lambda x: "-" in x)
df_test["zip_is_compound"] = df_test["Zipcode"].apply(lambda x: "-" in x)
```

df_train = pd.read_pickle(f"{DATA_ROOT}/train/featurized/data.pkl") df_test = pd.read_pickle(f"{DATA_ROOT}/test/featurized/data.pkl")

Handling categorical features

Scikit learn's package dtrees does not handle categorical values

- we need to encode categories into some kind of encoding, in order to train the model.
- reference

In [4]:
```
df_train.dtypes
```

Out[4]:
```
ID                        object
Source                    object
TMC                      float64
Distance(mi)             float64
Side                      object
City                      object
County                    object
State                     object
Timezone                  object
Airport_Code              object
Temperature(F)           float64
Humidity(%)              float64
Pressure(in)             float64
Visibility(mi)           float64
Wind_Direction            object
Wind_Speed(mph)          float64
Weather_Condition         object
Amenity                     bool
Bump                        bool
Crossing                    bool
Give_Way                    bool
Junction                    bool
No_Exit                     bool
Railway                     bool
Roundabout                  bool
Station                     bool
Stop                        bool
Traffic_Calming             bool
Traffic_Signal              bool
Turning_Loop                bool
Sunrise_Sunset            object
Civil_Twilight            object
Nautical_Twilight         object
Astronomical_Twilight     object
kw_vec                    object
zip_02                    object
zip_25                    object
zip_len                    int64
zip_is_compound             bool
Severity                   int64
dtype: object
```

Category transformer

In [5]:
```
# making list of cateogrical features that need encoding
# features that are as dtype string will need encoding

# categorical_feature -> encoding method
categorical_features = {
    "Source": "base_2",  # 3 unique values
    "Side": "base_2",  # 2 unique values
    "City": "base_4",  # 11895 unique values
    "County": "base_4",  # 1713 unique values
    "State": "base_2",  # 49 unique values
    "Timezone": "base_2",  # 4 unique values
    "Airport_Code": "base_4",  # 2001 unique values
    "Wind_Direction": "base_4",  # 24 unique values (after cleaning)
    "Weather_Condition": "base_4",  # 127 unique values
    "Sunrise_Sunset": "base_2",  # 2 unique values
    "Civil_Twilight": "base_2",  # 2 unique values
    "Nautical_Twilight": "base_2",  # 2 unique value
    "Astronomical_Twilight": "base_2",  # 2 unique value
    #
    # engineered features
    "zip_02": "ordinal",
    "zip_25": "ordinal",
}
```

In [10]:
```
def category_transformer(
    df_train: pd.DataFrame, df_test: pd.DataFrame, categorical_features: dic
):
    mapping_dict = dict()

    for i in categorical_features:
        method = categorical_features[i]

        # get values which will be encoded
        index_values = df_train[i].drop_duplicates().values.tolist()

        if "base" in method:
            print(f"""encoding {i} with {method}""")
            baseN = int(method.split("_")[1])
            enc = ce.binary.BaseNEncoder(base=2)
            # get values for train
            t_train = enc.fit_transform(df_train[i])
            df_train[i] = t_train.values.tolist()

            # get values for test
            t_test = enc.transform(df_test[i])
            df_test[i] = t_test.values.tolist()

        if method == "ordinal":
            print(f"""encoding {i} as {method}""")
            enc = ce.ordinal.OrdinalEncoder()
            # get values for train
            t_train = enc.fit_transform(df_train[i])
            df_train[i] = t_train.values.tolist()

            # get values for test
            t_test = enc.transform(df_test[i])
            df_test[i] = t_test.values.tolist()

        # store params
        mapping_dict[i] = {
            "enc_model_params": enc.get_params(),
```

```
            "enc_model_values": index_values,
        }

    return df_train, df_test, mapping_dict


df_train, df_test, mapping_dict = category_transformer(
    df_train, df_test, categorical_features
)
```

```
encoding Source with base_2
encoding Side with base_2
encoding City with base_4
encoding County with base_4
encoding State with base_2
encoding Timezone with base_2
encoding Airport_Code with base_4
encoding Wind_Direction with base_4
encoding Weather_Condition with base_4
encoding Sunrise_Sunset with base_2
encoding Civil_Twilight with base_2
encoding Nautical_Twilight with base_2
encoding Astronomical_Twilight with base_2
encoding zip_02 as ordinal
encoding zip_25 as ordinal
```

Mapping dictionary

In [13]:
```
# sample of how categories are encoded
# -1 denote representation of unknown value
# -2 denotes representation of missing value
mapping_dict["Source"]
```

Out[13]:
```
{'enc_model_params': {'base': 2,
  'cols': ['Source'],
  'drop_invariant': False,
  'handle_missing': 'value',
  'handle_unknown': 'value',
  'mapping': [{'col': 'Source',
    'mapping':      Source_0  Source_1
     1          0         1
     2          1         0
     3          1         1
    -1          0         0
    -2          0         0}],
  'return_df': True,
  'verbose': 0},
 'enc_model_values': ['Bing', 'MapQuest', 'MapQuest-Bing']}
```

In [14]:
```
# saving this as it will help in interpretation of results
pd.to_pickle(mapping_dict, f"{DATA_ROOT}/train/mapping_dict.pkl")
```

In [15]:
```
boolean_features = {
    "Amenity",
    "Bump",
    "Crossing",
    "Give_Way",
    "Junction",
    "No_Exit",
    "Railway",
    "Roundabout",
    "Station",
    "Stop",
```

```
    "Traffic_Calming",
    "Traffic_Signal",
    "Turning_Loop",
    # engineered features
    "zip_is_compound",
}
```

In [ ]:
```
len(
    set(df_train["Wind_Direction"].unique()).intersection(
        set(df_test["Wind_Direction"].unique())
    )
)
```

Out[ ]:
```
19
```

In [16]:
```
final_feature_list = [
    "ID",  # will be removing this before preparing modelling data
    "Source",
    "TMC",
    # "Start_Time", -> removing this as data is now sorted and split
    "Distance(mi)",
    # "Description", -> extracted features & removing this
    "Side",
    "City",
    "County",
    "State",
    # "Zipcode", -> extracted features & removing this
    "Timezone",
    "Airport_Code",
    "Temperature(F)",
    "Humidity(%)",
    "Pressure(in)",
    "Visibility(mi)",
    "Wind_Direction",
    "Wind_Speed(mph)",
    "Weather_Condition",
    "Amenity",
    "Bump",
    "Crossing",
    "Give_Way",
    "Junction",
    "No_Exit",
    "Railway",
    "Roundabout",
    "Station",
    "Stop",
    "Traffic_Calming",
    "Traffic_Signal",
    "Turning_Loop",
    "Sunrise_Sunset",
    "Civil_Twilight",
    "Nautical_Twilight",
    "Astronomical_Twilight",
    # engineered features
    "kw_vec",
    "zip_02",
    "zip_25",
    "zip_len",
    "zip_is_compound",
    # to predict
    "Severity",
]
```

In [17]:
```
os.makedirs(f"{DATA_ROOT}/train/featurized/", exist_ok=True)
os.makedirs(f"{DATA_ROOT}/test/featurized/", exist_ok=True)
```

In [18]:
```
df_train[final_feature_list].to_pickle(f"{DATA_ROOT}/train/featurized/data.p
df_test[final_feature_list].to_pickle(f"{DATA_ROOT}/test/featurized/data.pkl
```