# REPORT: Problem 2

# Missing Data Analysis

Problem Statement: To analyze the EDGAR Log File Data Set [https://www.sec.gov/data/edgar-log-filedata-set.html]. The page lists the meta data for the datasets and you are expected to develop a pipeline which does the following. Given a year, your program should get data for the first day of the month for every month in the year and process the file for handling missing data.

1) First, Initialize the log file. Use the following code to do that.

```
root = logging.getLogger()
root.setLevel(logging.DEBUG)
fH = logging.FileHandler('problem2_log.log')
fH.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
fH.setFormatter(formatter)
root.addHandler(fH)
```

2) The program takes the parameter from the user. Here the year of the data would be needed, and it must be ranged from 2003 to 2017. If an invalid year is provided, the program will exit.

```
print ("Please input the S3 Access Key")
accessKey = input()
logging.info("Access Key = %s" % accessKey)
```

```
print ('Please input the S3 Secret Access Key')
secretAccessKey = input()
logging.info("Secret Access Key = %s" % secretAccessKey)
```

```
print ("Please input your location")
location = input()
if location not in ['APNortheast', 'APSoutheast', 'APSoutheast2', 'EU', 'EUCentral1', 'SAEast', 'USWest', 'USWest2']:
    location = 'Default'
logging.info("Location = %s" % location)
```

```
year_range = range(2003, 2018)

print ('Please input the Year')
year = input()
if int(year) not in year_range:
    logging.error("Invalid year. Please enter a valid year between 2003 and 2017.")
    exit()
logging.info("Year = %s", year)
```

3) Validate the AWS account

```
AWS_ACCESS_KEY_ID = accessKey
AWS_SECRET_ACCESS_KEY = secretAccessKey

try:
    s3_connection = boto.connect_s3(AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY)
    region = s3_connection.get_all_regions()
    print ('S3 connection Successful!')

except:
    logging.info("Oops! The credentials are invalid")
    exit()
```

4) Cleaning up the required directory through the following code.

```
zip_dir = year + '_zips'
unzipped_dir = year + '_unzipped'
try:
    if not os.path.exists(zip_dir):
        os.makedirs(zip_dir, mode=0o777)
    else:
        shutil.rmtree(os.path.join(os.path.dirname(__file__), zip_dir), ignore_errors=False)
        os.makedirs(zip_dir, mode=0o777)

    if not os.path.exists(unzipped_dir):
        os.makedirs(unzipped_dir, mode=0o777)
    else:
        shutil.rmtree(os.path.join(os.path.dirname(__file__), unzipped_dir), ignore_errors=False)
        os.makedirs(unzipped_dir, mode=0o777)
    logging.info('Directories cleanup completed.')
except Exception as e:
    logging.error(str(e))
    exit()
```

5) Generate the URL to download the zip file. The URL is formed with domain and key-value pairs in month quarter dictionary

```
domain = "http://www.sec.gov/dera/data/Public-EDGAR-log-file-data/"
urls = []
year_range = range(2003, 2017)
month_quarter = {'Qtr1': ['01', '02', '03'], 'Qtr2': ['04', '05', '06'],
                 'Qtr3': ['07', '08', '09'], 'Qtr4': ['10', '11', '12']}

for key, value in month_quarter.items():
    for v in value:
        url = domain + str(year) + '/' + str(key) + '/' + 'log' + str(year) + str(v) + '01.zip'
        logging.info('url to download zip %s', url)
        urls.append(url)
```

6) Download the zip file and unzip it. If the file does not exist, the program will log a warning.

```
try:
    for i in range(0, 12):
        month_zip_dir = zip_dir + '/' + str(i) + '.zip'
        month_unzipped_dir = unzipped_dir + '/' + str(i)
        r = requests.get(urls[i],allow_redirects=True)
        open(month_zip_dir, 'wb').write(r.content)
        if os.path.getsize(month_zip_dir) <= 4515: #catching empty file
            os.remove(month_zip_dir)
            logging.warning('Log file %s is empty.', i)
        else:
            logging.info('Log file %s successfully downloaded', i)
            try:
                zip_ref = zipfile.ZipFile(month_zip_dir, 'r')
                for file in zip_ref.namelist():
                    if file.endswith('.csv'):
                        zip_ref.extract(file, unzipped_dir)
                        zip_ref.close()
                        logging.info('Log file %s was successfully unzipped', i)
            except Exception as e:
                logging.error(str(e))
                exit()
except Exception as e:  # Catching file not found
    logging.warning('Log for month %s not found!', i)
    exit()
```

7) Read all unzipped csv files and put them in a Data Frame structure.

```
file_lists = glob.glob(unzipped_dir + "/*.csv")

all_csv_df_dict = {period: pd.read_csv(period) for period in file_lists}
logging.info('All the csv read into individual dataframes')
```

8) Detect the anomalies and clean the data set. By doing so, the program counts the null values, checks if there is any incorrect value instead of 0 and 1 in idx, norefer and noagent fields.

Handle the missing data. The program will delete the rows which have missing values in CIK, accession number, ip, date and time columns. For the columns of idx, browser, code, find, extension, zone, the program will fill the missing values with the most used data of that column; For the columns of norefer, noagent, replace missing values with 1; For the column of crawler, replace the missing values with 0; For the column of size, replace the missing value with average value.

For each month log file, compute the summary matrix

```
try:
    for k, v in all_csv_df_dict.items():
        st = all_csv_df_dict[k]
        for key, value in st.items():
            key_drop = {'cik', 'accession', 'ip', 'date', 'time'}
            key_max = {'idx', 'browser', 'code', 'find', 'extention', 'zone'}
            df = pd.DataFrame(st[key])
            null_count = df.isnull().sum()
            logging.info("count of null in  %s is %s" % (key, null_count))
            most_used_value = pd.DataFrame(df.groupby(key).size().rename('cnt')).idxmax()[0]
            if key == "idx":
                incorrect_idx = (~df.isin([0.0, 1.0])).sum()
                logging.info("count of incorrect idx is %s" % incorrect_idx)
                st[key] = st[key].fillna(most_used_value)
                logging.info("fill the null value in column %s with the most used value" % key)
            elif key == "norefer":
                incorrect_norefer = (~df.isin([0.0, 1.0])).sum()
                logging.info("count of incorrect norefer is %s" % incorrect_norefer)
                st[key] = st[key].fillna('1')
                logging.info("fill the null value in column %s with 1" % key)
            elif key == "noagent":
                incorrect_noagent = (~df.isin([0.0, 1.0])).sum()
                logging.info("count of incorrect noagent is %s" % incorrect_noagent)
                st[key] = st[key].fillna('1')
                logging.info( "fill the null value in column %s with 1" % key)
            elif key in key_drop:
                st[key] = st.dropna(subset=[key])
                logging.info("the null in %s is dropped" % key)
            elif key in key_max:
                st[key] = st[key].fillna(most_used_value)
                logging.info("fill the null value in column %s with the most used value" % key)
            elif key == "crawler":
                st[key] = st[key].fillna('0')
                logging.info("fill the null value in column %s with 0" % key)
```

```
            elif key == "size":
                st[key] = st[key].fillna(st[key].mean(axis=0))
                logging.info("fill the null value in column %s with the average value" % key)
except Exception as e:
    logging.error(str(e))
    exit()
```

9) Combine all the individual Data Frames into one Data Frame and export a csv file.

10) Zip the csv file

```
try:
    dfs = pd.concat(all_csv_df_dict)
    dfs.to_csv('Edgar_log_data.csv')
    logging.info('All dataframes of csvs are combined and exported as csv: master_csv.csv.')
except Exception as e:
    logging.error(str(e))
    exit()
```

```
def zipdir(path, ziph):
    ziph.write(os.path.join('Edgar_log_data.csv'))
    ziph.write(os.path.join('problem2_log.log'))
```

```
zipf = zipfile.ZipFile('Problem2.zip', 'w', zipfile.ZIP_DEFLATED)
zipdir('/', zipf)
zipf.close()
logging.info("csv and log files successfully zipped!")
```

11) Create a new Amazon S3 bucket and upload all files.

```
try:
    zipfile = 'Problem2.zip'
    ts = time.time()
    st = datetime.datetime.fromtimestamp(ts)
    bucket_name = AWS_ACCESS_KEY_ID.lower() + str(st).replace(" ", "").replace("-", "").replace(":", "").replace(".", "")
    conn = boto.connect_s3(AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY)
    bucket = conn.create_bucket(bucket_name, location=location)
    print ('bucket created')
    print ("Uploading %s to Amazon S3 bucket %s" % (zipfile, bucket_name))

    k = Key(bucket)
    k.key = 'Problem2'
    k.set_contents_from_filename(zipfile)
    print("Zip File successfully uploaded to S3")
except:
    logging.info("AWS credentials are invalid!")
    exit()
```