code

December 10, 2017

0.0.1 spark session and libraries

```
In [1]: from pyspark import SparkConf
        from pyspark.sql import SparkSession
        from pyspark.sql import Row
        from pyspark.sql import functions
        from pyspark.sql import DataFrameStatFunctions
        from pyspark.sql.functions import col, udf
        from IPython.display import display
        from datetime import datetime
        from pyspark.sql.types import FloatType, BooleanType, IntegerType, StringTy
        from time import mktime
        from datetime import datetime
        import pandas as pd
        import scipy
        from scipy.stats.stats import pearsonr

        spark_conf = SparkConf() \
            .setAll([
                 ['spark.serializer','org.apache.spark.serializer.KryoSerializer'],
                 ['spark.rdd.compress','true'],
            ])

        spark = SparkSession \
            .builder \
            .appName("pager") \
            .config(conf=spark_conf) \
            .enableHiveSupport() \
            .getOrCreate()

        hdfs_path = 'hdfs://ip-172-31-52-225.ec2.internal'
```

1 ETL

1.0.1 preprocessing: skip if processed file is in hadoop

save the processed file as parquet since it is a columnar format we can perfrom groupby operations faster.

```
In [9]: # read 311_service_requests data from hdfs
        df = spark.read.csv(hdfs_path+ '/pager/311_service_requests.csv', header=Tr
                            inferSchema=True, ignoreLeadingWhiteSpace=None, ignoreTr

        # rename columns and convert to lower case
        for col, dtype in df.dtypes:
            new_col = col.replace(" ", "")
            df = df.withColumnRenamed(col, new_col)
            if dtype == 'string':
                df = df.withColumn(new_col, functions.lower(df[new_col]))

        # UDF fuctions "utils"
        get_month_func = udf(lambda x: datetime.strptime(x, '%m/%d/%Y %I:%M:%S %p')
        get_year_func = udf(lambda x: datetime.strptime(x, '%m/%d/%Y %I:%M:%S %p').
        time_difference_func = udf(lambda x, y: (datetime.strptime(x, '%m/%d/%Y %I:
                                                 datetime.strptime(y, '%m/%d/%Y %I:
                                   FloatType())
        is_school_func = udf(lambda x: False if x == "unspecified" else True, Boole

        # drop NA in dates
        df = df.dropna(subset=['CreatedDate', 'ClosedDate'])

        # create new columns
        df = df.withColumn('Month', get_month_func(df['CreatedDate']))
        df = df.withColumn('Year', get_year_func(df['CreatedDate']))
        df = df.withColumn('TimeTaken', time_difference_func(df['ClosedDate'], df['
        df = df.withColumn('SchoolZone', is_school_func(df['SchoolName']))
        df = df.withColumn('Incidentzip', df['Incidentzip'].substr(1, 6))

        # rename columns
        df = df.withColumnRenamed('YCoordinate(StatePlane)', 'YCoordinateStatePlane
        df = df.withColumnRenamed('XCoordinate(StatePlane)', 'XCoordinateStatePlane

        # filters
        df = df.filter(df['TimeTaken'] > 0)

        df.write.parquet(hdfs_path + '/pager/parquet/311_data/', mode='overwrite')
```

1.1 Use the parquet format from now on.

Always start here if data parquet format on hdfs

```
In [7]: df = spark.read.parquet(hdfs_path + "/pager/parquet/311_data")
```

2 Exploratory Analysis

Group based on the each of the selected column for selected column + Time Taken column pair
For each grouping use aggregate for count and mean

Convert the grouped data into pandas dataframes and write into excel sheets

Perfrom exploratory analysis means of grouped data by joining all the pandas dataframes formed

Check for variance in the means

(i) Large variance/stddev implies that the feature is important driver because it means that in each feature the category are diverse and have extreme varying effects

```
In [13]: grouped_counts_dict = {}
         pandas_df_dict = {}

         # removing too detail oriented columns or columns that are copy of some ot
         select_columns = list(set(df.columns) -
                               set(['Latitude', 'Longitude', 'Location', 'XCoordina
                                    'ClosedDate', 'CreatedDate', 'DueDate', 'Resolut
                                    'TimeTaken', 'ParkBorough', 'AgencyName', 'Uniqu
         # print(select_columns)
         # Create a Pandas Excel writer using XlsxWriter as the engine.
         writer = pd.ExcelWriter('./pager_all_col_analysis.xlsx', engine='xlsxwrite

         for col in select_columns:
             group = [col] + ['TimeTaken']
             grouped = df.select(group).groupBy(col)
             grouped_counts_dict[col] = {}
             grouped_counts_dict[col]['counts'] = grouped.count()
             grouped_counts_dict[col]['mean_time_taken'] = grouped.mean('TimeTaken'

             pandas_df_count = grouped_counts_dict[col]['counts'].toPandas()
             pandas_df_mean = grouped_counts_dict[col]['mean_time_taken'].toPandas

             pandas_df = pandas_df_count.join(pandas_df_mean.set_index(col), on=col
             pandas_df_dict[col] = pandas_df[['avg(TimeTaken)']].describe()

             pandas_df.to_excel(writer, sheet_name=col)

         exploratory_analysis = pd.concat([pandas_df_dict[col] for col in select_co
         exploratory_analysis.columns = ['TimeTaken'] + select_columns
         exploratory_analysis.to_excel(writer, sheet_name='exploratory_analysis')

         display(exploratory_analysis)
```

	TimeTaken	CrossStreet2	FacilityType	ParkFacilityName	CrossStreet1	\
0	count	26116.000000	6.000000	4186.000000	25768.000000	
1	mean	365.658151	1054.828844	818.797291	376.971932	
2	std	877.238402	1330.409651	1549.255497	933.795507	
3	min	0.006389	4.483830	0.099167	0.003333	
4	25%	65.795363	149.095611	118.831109	68.875782	
5	50%	184.565391	772.227826	271.531251	187.949180	

```
6       75%    399.636323   1137.524710        850.106070    412.771342
7       max  37206.214844   3579.358369      32159.117188  41332.250000

   ComplaintType    StreetName  TaxiPickUpLocation  SchoolState  \
0     279.000000  31380.000000            8.000000     2.000000
1     666.591178    476.336761          851.979023   383.900277
2    1557.538318   1227.822275          215.543938    40.821188
3       0.011983      0.000278          350.880788   355.035338
4      45.155536     67.981020          857.112164   369.467808
5     203.892079    264.445251          915.669316   383.900277
6     575.287516    466.363801          966.817305   398.332747
7   17367.164875  40425.359375         1019.611859   412.765216

   IntersectionStreet1      ...             Year    SchoolName       Agency  \
0         21002.000000      ...         8.000000   3713.000000    22.000000
1           345.495133      ...       362.505193    883.848733   502.816256
2          1063.045161      ...       100.895896   1617.103482   611.327217
3             0.000278      ...       167.944542      0.099167     0.014705
4            43.081249      ...       331.418470    130.923849    94.940681
5           142.102847      ...       386.125926    299.774564   254.354355
6           317.563921      ...       426.852057    958.029668   591.492468
7         35128.300781      ...       483.215222  32159.117188  2099.378765

   BridgeHighwayName  SchoolAddress  SchoolRegion  AddressType  \
0          89.000000    3056.000000     12.000000     6.000000
1         318.197072     772.859935   1244.499186   373.093974
2         554.418336    1249.000958    475.560591   168.564998
3          10.849722       0.099167    246.876731   210.045546
4         103.146715     134.058203   1221.010513   249.373963
5         180.570535     297.172283   1362.218380   324.680185
6         283.423990     885.079060   1539.157409   464.460444
7        4352.266968   12776.428711   1809.082021   643.832178

   FerryTerminalName  SchoolPhoneNumber    SchoolCode
0        5251.000000         395.000000   1688.000000
1         323.293855         310.632592   1557.043044
2         688.868380         399.286093   2086.798106
3           0.075000           0.099167      0.190278
4          44.030277         150.131510    323.303968
5         120.492775         219.734943    977.235737
6         282.840836         330.113720   2064.995386
7        7250.304688        4516.438477  32159.117188

[8 rows x 43 columns]
```

```
In [30]: # Correlation between Time Taken and Time Due
         df.filter(df['DueDate'].isNotNull()).withColumn(
```

```
            'TimeDue', time_difference_func(df['DueDate'], df['CreatedDate'])).cor
```

```
Out[30]: 0.030723661434716595
```

```
In [ ]:
```

2.1 Do the top features change over time

Group based on Year/Month and the column
Aggregate on the TimeTaken column (mean, counts)
convert the result into pandas dataframe and write to xlsx file
use the excel file to check if the mean is changing with year/month (time)

```
In [33]: top_features = ['Agency', 'ComplaintType', 'Descriptor', 'LocationType', '
```

2.1.1 Yearly

```
In [34]: # Create a Pandas Excel writer using XlsxWriter as the engine.
         writer = pd.ExcelWriter('./pager_yearly_analysis.xlsx', engine='xlsxwriter

         for col in top_features:
             group = [col] + ['TimeTaken', 'Year']
             grouped = df.select(group).groupBy(['Year'] + [col])
             grouped_counts_dict[col] = {}
             grouped_counts_dict[col]['counts'] = grouped.count()
             grouped_counts_dict[col]['mean_time_taken'] = grouped.mean('TimeTaken'

             pandas_df_count = grouped_counts_dict[col]['counts'].toPandas()
             pandas_df_mean = grouped_counts_dict[col]['mean_time_taken'].toPandas

             pandas_df = pandas_df_count.join(pandas_df_mean.set_index(['Year'] +
                                              on=['Year'] + [col]).sort_values(by=c
             pandas_df.to_excel(writer, sheet_name=col)
```

2.1.2 Monthly

```
In [35]: # Create a Pandas Excel writer using XlsxWriter as the engine.
         writer = pd.ExcelWriter('./pager_monthly_analysis.xlsx', engine='xlsxwrite

         for col in top_features:
             group = [col] + ['TimeTaken', 'Month']
             grouped = df.select(group).groupBy(['Month'] + [col])
             grouped_counts_dict[col] = {}
             grouped_counts_dict[col]['counts'] = grouped.count()
             grouped_counts_dict[col]['mean_time_taken'] = grouped.mean('TimeTaken'

             pandas_df_count = grouped_counts_dict[col]['counts'].toPandas()
             pandas_df_mean = grouped_counts_dict[col]['mean_time_taken'].toPandas
```

```
            pandas_df = pandas_df_count.join(pandas_df_mean.set_index(['Month'] +
                                                    on=['Month'] + [col]).sort_values(by=
            pandas_df.to_excel(writer, sheet_name=col)
```

In [ ]:

2.2 Are there rats?

```
In [52]: df = spark.read.parquet(hdfs_path + "/pager/parquet/311_data")
         df = df[['ComplaintType', 'Descriptor', 'LocationType', 'Borough', 'Incide
         df = df.filter(df['Borough'].isin(['bronx', 'manhattan', 'brooklyn', 'quee

         incident_count_df = df.groupBy('Borough').agg(functions.count('Incidentzip

         descriptor_list = df.select('Descriptor').distinct().toPandas()['Descripto
```

removing unwanted descriptors not related to rats in restuarnts with low inspection grades

```
In [53]: select_descriptor_list = list(set(descriptor_list) - set(['underage - lice
                                                                  'after hours - l
                                                                 'loud talking', '
                                                                 'loud music/party

In [56]: rat_encounters_df = df.filter(df['Descriptor'].like('rat sighting')).group
             'Borough').agg(functions.count('Incidentzip').alias('rat_encounters'))

         restaurant_count_df = df.filter(df['LocationType'].like('%restaurant%') &
                                         ).groupBy('Borough').agg(functions.count('I

         rats_df = rat_encounters_df.join(restaurant_count_df.set_index('Borough'),
             incident_count_df.set_index('Borough'), on='Borough')

         display(rats_df)
         print("Correlation between rat_encounters and restaurant_count")
         display(rats_df.corr())
         print("Correleation b/w rat_encounters and restaurant_count \n after normal
         pearsonr(rats_df['rat_encounters']/rats_df['incident_count'], rats_df['res
```

	Borough	rat_encounters	restaurant_count	incident_count
0	bronx	14063	4982	2526713
1	manhattan	17816	19992	2853110
2	brooklyn	27153	12595	4252203
3	queens	12183	10297	3164081
4	staten island	4051	2038	685958

```
Correlation between rat_encounters and restaurant_count
```

	rat_encounters	restaurant_count	incident_count
rat_encounters	1.000000	0.639739	0.918782
restaurant_count	0.639739	1.000000	0.623292
incident_count	0.918782	0.623292	1.000000

Corrleation b/w rat_encounters and restaurant_count
after normalizing their counts with Incident_count

Out[56]: (0.27148626993565828, 0.65862739214395583)

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: