CSCI - 6409 - Process of Data Science - Summer 2022

</center>

Assignment 1

</center>

Benny Daniel Tharigopala - B00899629

Karthik Kannan Nanthakumar - B00891387

Import Files to Google Colab

In [1]:

```
# from google.colab import files
# uploaded = files.upload()
# #alternative  method to read csvs - read CSVs from google drive
# # df=pd.read_csv('gdrive/My Drive/data.csv')
```

Imports

In [2]:

```
import pandas as pd
import io
```

In [3]:

```
# dataset_2017 = pd.read_csv(io.BytesIO(uploaded['modis_2017_Australia.csv']))
# dataset_2018 = pd.read_csv(io.BytesIO(uploaded['modis_2018_Australia.csv']))
# dataset_2019 = pd.read_csv(io.BytesIO(uploaded['modis_2019_Australia.csv']))
# dataset_2020 = pd.read_csv(io.BytesIO(uploaded['modis_2020_Australia.csv']))
dataset_2017 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/modis_2017_Australia.csv')
dataset_2018 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/modis_2018_Australia.csv')
dataset_2019 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/modis_2019_Australia.csv')
dataset_2020 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/modis_2020_Australia.csv')
```

Shape of each dataframe

Shape of each dataframe

In [4]:

```
print(dataset_2017.shape)
print(dataset_2018.shape)
print(dataset_2019.shape)
print(dataset_2020.shape)
#We have found out that all the CSVs have the same number of columns
```

```
(273312, 15)
(307359, 15)
(310484, 15)
(155524, 15)
```

In [5]:

```
dataset_2017.describe()
```

Out[5]:

	latitude	longitude	brightness	scan	track	acq_time	confidence	version	bright_t31	frp	type
count	273312.000000	273312.000000	273312.000000	273312.000000	273312.000000	273312.000000	273312.000000	2.733120e+05	273312.000000	273312.000000	273312.000000
mean	-21.207506	132.701744	332.737582	1.714042	1.242864	579.442191	71.355517	6.200000e+00	303.583344	73.042409	0.008770
std	6.541458	9.331279	22.613024	0.895719	0.267355	499.312085	22.233986	2.664540e-15	10.356276	164.081231	0.135229
min	-43.431600	113.668600	300.000000	1.000000	1.000000	0.000000	0.000000	6.200000e+00	266.300000	0.000000	0.000000
25%	-24.437200	125.969575	317.800000	1.100000	1.000000	211.000000	57.000000	6.200000e+00	296.200000	14.600000	0.000000
50%	-19.763050	131.301900	328.700000	1.300000	1.100000	446.000000	74.000000	6.200000e+00	303.400000	29.100000	0.000000
75%	-16.387775	138.819325	341.600000	2.000000	1.400000	554.000000	90.000000	6.200000e+00	310.200000	66.400000	0.000000
max	-10.121900	153.585000	506.200000	4.800000	2.000000	2359.000000	100.000000	6.200000e+00	400.100000	7401.100000	3.000000

In [6]:

```
dataset_2018.describe()
```

Out[6]:

	latitude	longitude	brightness	scan	track	acq_time	confidence	version	bright_t31	frp	type
count	307359.000000	307359.000000	307359.000000	307359.000000	307359.000000	307359.000000	307359.000000	307359.00	307359.000000	307359.000000	307359.000000
mean	-20.045215	133.088962	332.360836	1.695274	1.238009	587.148702	70.732199	6.03	303.800152	66.583566	0.016082
std	6.471534	9.479791	21.268470	0.871393	0.262551	505.639317	22.367220	0.00	9.761293	146.255953	0.180227
min	-43.490000	113.129400	300.000000	1.000000	1.000000	0.000000	0.000000	6.03	265.700000	0.000000	0.000000

	latitude	longitude	brightness	scan	track	acq_time	confidence	version	bright_t31	frp	type
25%	-24.129250	125.925400	318.500000	1.100000	1.000000	212.000000	56.000000	6.03	297.000000	14.600000	0.000000
50%	-17.980000	131.551500	328.800000	1.300000	1.100000	442.000000	74.000000	6.03	303.800000	28.500000	0.000000
75%	-15.013300	141.785950	341.100000	2.000000	1.400000	557.000000	89.000000	6.03	310.100000	62.700000	0.000000
max	-9.246300	153.583600	506.300000	4.800000	2.000000	2359.000000	100.000000	6.03	400.100000	7395.400000	3.000000

In [7]:

```
dataset_2019.describe()
```

Out[7]:

	latitude	longitude	brightness	scan	track	acq_time	confidence	version	bright_t31	frp	type
count	310484.000000	310484.000000	310484.000000	310484.000000	310484.000000	310484.000000	310484.000000	310484.00	310484.000000	310484.000000	310484.000000
mean	-23.996230	137.901682	334.137700	1.633547	1.218591	694.263801	72.027029	6.03	302.832124	76.205352	0.013627
std	8.511506	11.479861	24.739488	0.825830	0.250785	575.643137	23.863859	0.00	11.671343	187.749687	0.168098
min	-43.407900	113.458100	300.000000	1.000000	1.000000	0.000000	0.000000	6.03	265.700000	0.000000	0.000000
25%	-31.340300	128.466900	317.800000	1.100000	1.000000	323.000000	56.000000	6.03	294.700000	15.400000	0.000000
50%	-25.467200	136.767100	329.600000	1.300000	1.100000	450.000000	76.000000	6.03	302.000000	30.400000	0.000000
75%	-15.324000	150.059100	343.500000	1.900000	1.300000	1254.000000	93.000000	6.03	309.900000	67.600000	0.000000
max	-9.386900	153.591900	507.000000	4.800000	2.000000	2359.000000	100.000000	6.03	400.100000	7454.500000	3.000000

dataset_2020.describe()

All the datasets have the same number of columns and the columns are identical as well. Let us now proceed to merge the dataset for 4 years into a single dataframe.

In [8]:

```
dataset_merged = pd.concat([dataset_2017,dataset_2018,dataset_2019,dataset_2020])
print("Shape of the Merged Dataframe: ", dataset_merged.shape)
print("Number of Instances: ", len(dataset_merged.index))
```

```
Shape of the Merged Dataframe:  (1046679, 15)
Number of Instances:  1046679
```

Data quality report:

The data quality report consists of: Tabular report for continuous features Tabular report for categorical features Data visualizations of values in each feature [Data Quality Report - Brightspace Tutorial]

Quality Report - Brightspace Tutorial

In [9]:

```
import warnings

pd.set_option("display.max_rows", None)
pd.set_option("display.max_columns", None)
pd.set_option('display.float_format', '{:.2f}'.format)
#Referred from Tutorial 2 of CSCI 6409 - [https://dal.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=fe8e7287-82c2-42bc-85ac-ae9
40127b726]
```

What are the features in the Australian Bushfire Dataset ?

In [10]:

```
dataset_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1046679 entries, 0 to 155523
Data columns (total 15 columns):
 #   Column      Non-Null Count    Dtype  
---  ------      --------------    -----  
 0   latitude    1046679 non-null  float64
 1   longitude   1046679 non-null  float64
 2   brightness  1046679 non-null  float64
 3   scan        1046679 non-null  float64
 4   track       1046679 non-null  float64
 5   acq_date    1046679 non-null  object 
 6   acq_time    1046679 non-null  int64  
 7   satellite   1046679 non-null  object 
 8   instrument  1046679 non-null  object 
 9   confidence  1046679 non-null  int64  
 10  version     1046679 non-null  float64
 11  bright_t31  1046679 non-null  float64
 12  frp         1046679 non-null  float64
 13  daynight    1046679 non-null  object 
 14  type        1046679 non-null  int64  
dtypes: float64(8), int64(3), object(4)
memory usage: 127.8+ MB
```

We can observe that all 15 columns of the dataset do not have any Null-records within them, since the total number of rows is 1046679 and the number of non-null records in each column is also 1046679.

Now let's peek at the first few rows of our data frame

In [11]:

```
dataset_merged.loc[0]
```

Out[11]:

	latitude	longitude	brightness	scan	track	acq_date	acq_time	satellite	instrument	confidence	version	bright_t31	frp	daynight	type
0	-23.91	147.30	320.10	1.70	1.30	2017-01-01	47	Terra	MODIS	53	6.20	296.60	17.60	D	0
0	-15.59	143.69	319.30	2.70	1.60	2018-01-01	15	Terra	MODIS	65	6.03	291.40	38.80	D	0
0	-13.99	127.39	324.20	2.60	1.50	2019-01-01	122	Terra	MODIS	39	6.03	294.80	31.00	D	0
0	-13.21	143.15	337.20	1.00	1.00	2020-01-01	50	Terra	MODIS	87	6.03	298.00	27.90	D	0

In [12]:

```
dataset_merged.head (5)
```

Out[12]:

	latitude	longitude	brightness	scan	track	acq_date	acq_time	satellite	instrument	confidence	version	bright_t31	frp	daynight	type
0	-23.91	147.30	320.10	1.70	1.30	2017-01-01	47	Terra	MODIS	53	6.20	296.60	17.60	D	0
1	-23.69	150.10	314.30	2.70	1.60	2017-01-01	47	Terra	MODIS	22	6.20	289.30	30.00	D	0
2	-23.59	150.17	315.80	2.70	1.60	2017-01-01	47	Terra	MODIS	33	6.20	291.70	35.40	D	0
3	-22.41	148.85	316.70	2.10	1.40	2017-01-01	47	Terra	MODIS	26	6.20	295.30	25.80	D	0
4	-20.59	147.64	320.70	1.60	1.30	2017-01-01	47	Terra	MODIS	34	6.20	299.60	19.00	D	0

Fit the best possible datatypes for columns of type object.

In [13]:

```
dataset_merged = dataset_merged.convert_dtypes()
dataset_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1046679 entries, 0 to 155523
Data columns (total 15 columns):
 #   Column      Non-Null Count    Dtype  
---  ------      --------------    -----  
 0   latitude    1046679 non-null  Float64
 1   longitude   1046679 non-null  Float64
 2   brightness  1046679 non-null  Float64
 3   scan        1046679 non-null  Float64
```

```
 3   scan        1046679 non-null  Float64
 4   track       1046679 non-null  Float64
 5   acq_date    1046679 non-null  string
 6   acq_time    1046679 non-null  Int64
 7   satellite   1046679 non-null  string
 8   instrument  1046679 non-null  string
 9   confidence  1046679 non-null  Int64
 10  version     1046679 non-null  Float64
 11  bright_t31  1046679 non-null  Float64
 12  frp         1046679 non-null  Float64
 13  daynight    1046679 non-null  string
 14  type        1046679 non-null  Int64
dtypes: Float64(8), Int64(3), string(4)
memory usage: 171.0 MB
```

In [14]:

```
dataset_merged.columns
```

Out[14]:

```
Index(['latitude', 'longitude', 'brightness', 'scan', 'track', 'acq_date',
       'acq_time', 'satellite', 'instrument', 'confidence', 'version',
       'bright_t31', 'frp', 'daynight', 'type'],
      dtype='object')
```

In [15]:

```
dataset_merged.describe(include=['number'])
```

Out[15]:

	latitude	longitude	brightness	scan	track	acq_time	confidence	version	bright_t31	frp	type
count	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00
mean	-21.96	135.25	332.88	1.66	1.23	622.02	71.06	6.07	303.23	70.50	0.01
std	7.79	10.52	23.11	0.85	0.26	532.04	22.92	0.07	10.73	169.47	0.17
min	-43.50	113.13	300.00	1.00	1.00	0.00	0.00	6.03	265.70	-29.90	0.00
25%	-28.77	126.80	317.90	1.10	1.00	225.00	56.00	6.03	295.90	14.60	0.00
50%	-19.92	133.14	328.70	1.30	1.10	444.00	74.00	6.03	303.00	28.60	0.00
75%	-15.30	144.96	341.60	1.90	1.40	629.00	90.00	6.20	309.80	63.50	0.00
max	-9.25	153.59	507.00	4.80	2.00	2359.00	100.00	6.20	400.10	11164.10	3.00

In [16]:

```
dataset_merged.describe(exclude=['number'])
```

Out[16]:

	acq_date	satellite	instrument	daynight
count	1046679	1046679	1046679	1046679
unique	1461	2	1	2
top	2020-01-04	Aqua	MODIS	D
freq	7351	600236	1046679	806375

Features in the Dataset:

Continuous Features:

Latitude, Longitude, Brightness, Scan, Track, Confidence, bright_t31, frp, type, acq_time, version

Categorical Features:

acq_date, Satellite, instrument, daynight

QUESTION 1.1 & 1.2

Data Quality Report for Continuous Features:

Code referrered from [CSCI 6409 - Tutorial 2]

In [17]:

```
def build_continuous_features_report(data_df):

    """Build tabular report for continuous features"""

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Min": lambda df: df.min(),
        "1st Qrt.": lambda df: df.quantile(0.25),
        "Mean": lambda df: df.mean(),
        "Median": lambda df: df.median(),
        "3rd Qrt": lambda df: df.quantile(0.75),
        "Max": lambda df: df.max(),
```

```
        "Std. Dev.": lambda df: df.std(),
    }

    contin_feat_names = data_df.select_dtypes("number").columns
    continuous_data_df = data_df[contin_feat_names]

    report_df = pd.DataFrame(index=contin_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
build_continuous_features_report(dataset_merged)
```

Out[17]:

	Count	Miss %	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt	Max	Std. Dev.
latitude	1046679	0.00	259382	-43.50	-28.77	-21.96	-19.92	-15.30	-9.25	7.79
longitude	1046679	0.00	325532	113.13	126.80	135.25	133.14	144.96	153.59	10.52
brightness	1046679	0.00	2050	300.00	317.90	332.88	328.70	341.60	507.00	23.11
scan	1046679	0.00	39	1.00	1.10	1.66	1.30	1.90	4.80	0.85
track	1046679	0.00	11	1.00	1.00	1.23	1.10	1.40	2.00	0.26
acq_time	1046679	0.00	855	0.00	225.00	622.02	444.00	629.00	2359.00	532.04
confidence	1046679	0.00	101	0.00	56.00	71.06	74.00	90.00	100.00	22.92
version	1046679	0.00	2	6.03	6.03	6.07	6.03	6.20	6.20	0.07
bright_t31	1046679	0.00	972	265.70	295.90	303.23	303.00	309.80	400.10	10.73
frp	1046679	0.00	13993	-29.90	14.60	70.50	28.60	63.50	11164.10	169.47
type	1046679	0.00	3	0.00	0.00	0.01	0.00	0.00	3.00	0.17

Data Quality Report for Categorical Features:

Code refererred from [CSCI 6409 - Tutorial 2]

In [18]:

```
def build_categorical_features_report(data_df):

    """Build tabular report for categorical features"""
```

```python
    def _mode(df):
        return df.apply(lambda ft: ft.mode().to_list()).T

    def _mode_freq(df):
        return df.apply(lambda ft: ft.value_counts()[ft.mode()].sum())

    def _second_mode(df):
        return df.apply(lambda ft: ft[~ft.isin(ft.mode())].mode().to_list())

    def _second_mode_freq(df):
        return df.apply(
            lambda ft: ft[~ft.isin(ft.mode())]
            .value_counts()[ft[~ft.isin(ft.mode())].mode()]
            .sum()
        )

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Mode": _mode,
        "Mode Freq": _mode_freq,
        "Mode %": lambda df: _mode_freq(df) / len(df) * 100,
        "2nd Mode": _second_mode,
        "2nd Mode Freq": _second_mode_freq,
        "2nd Mode %": lambda df: _second_mode_freq(df) / len(df) * 100,
    }

    cat_feat_names = data_df.select_dtypes(exclude="number").columns
    continuous_data_df = data_df[cat_feat_names]

    report_df = pd.DataFrame(index=cat_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
```

In [19]:

```python
build_categorical_features_report(dataset_merged)
```

Out[19]:

	Count	Miss %	Card.	Mode	Mode Freq	Mode %	2nd Mode	2nd Mode Freq	2nd Mode %

	Count	Miss %	Card.	Mode	Mode Freq	Mode %	2nd Mode	2nd Mode Freq	2nd Mode %
acq_date	1046679	0.00	1461	2020-01-04	7351	0.70	[2019-12-30]	6925	0.66
satellite	1046679	0.00	2	Aqua	600236	57.35	[Terra]	446443	42.65
instrument	1046679	0.00	1	MODIS	1046679	100.00	[]	0	0.00
daynight	1046679	0.00	2	D	806375	77.04	[N]	240304	22.96

Visualization of Continuous Features

In [20]:

```
from matplotlib import pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [12, 8]
plt.rcParams["font.size"] = 15
```

In [21]:

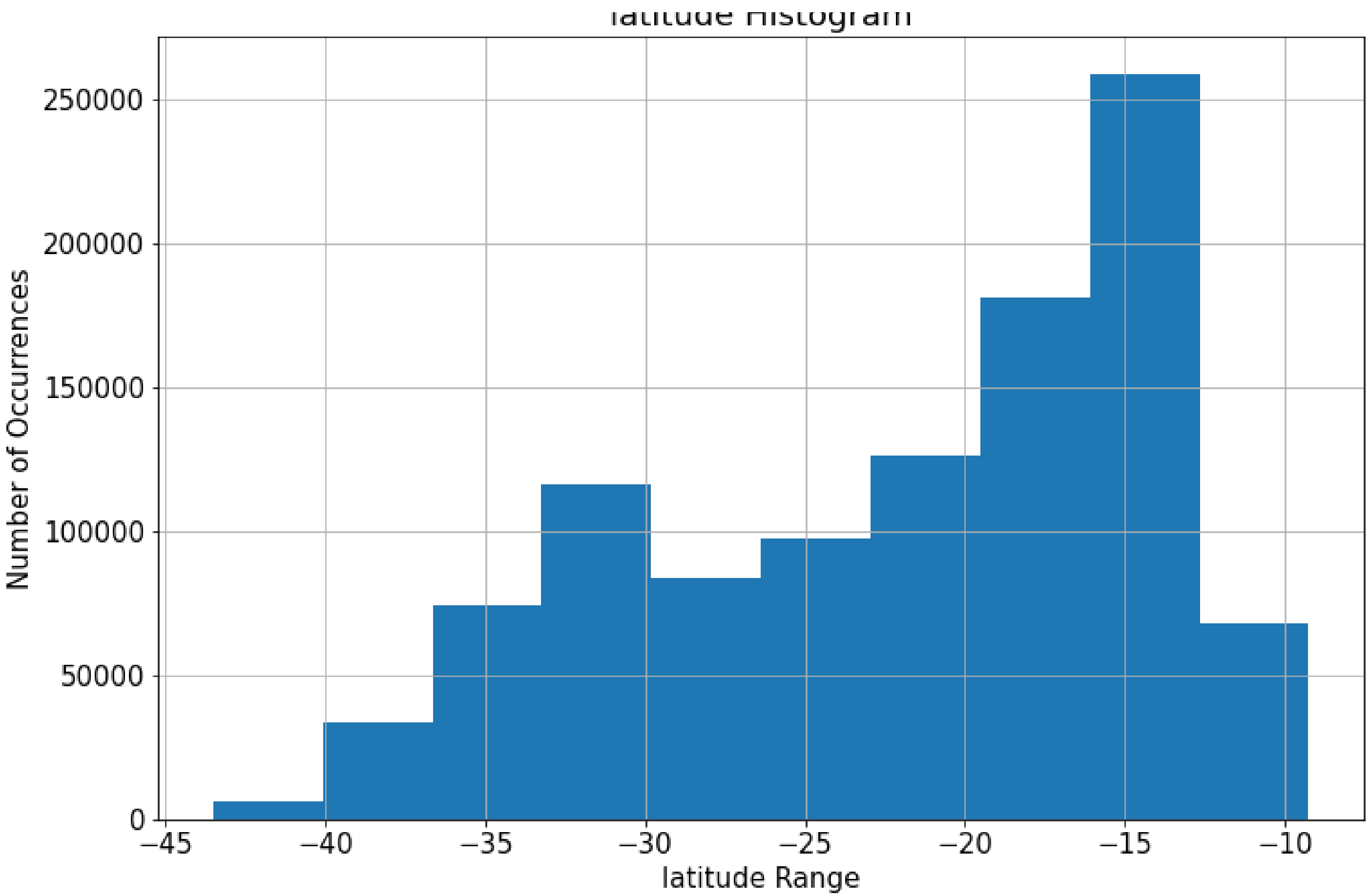
```
dataset_merged.describe(include=['number'])
```

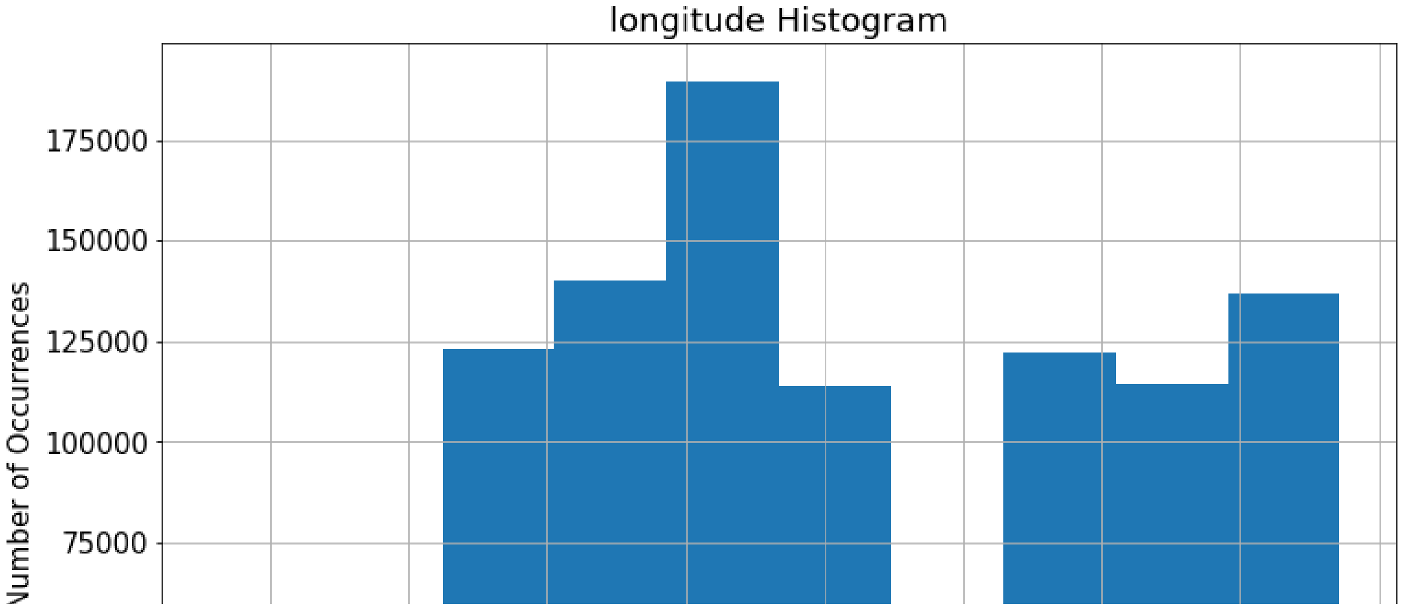
Out[21]:

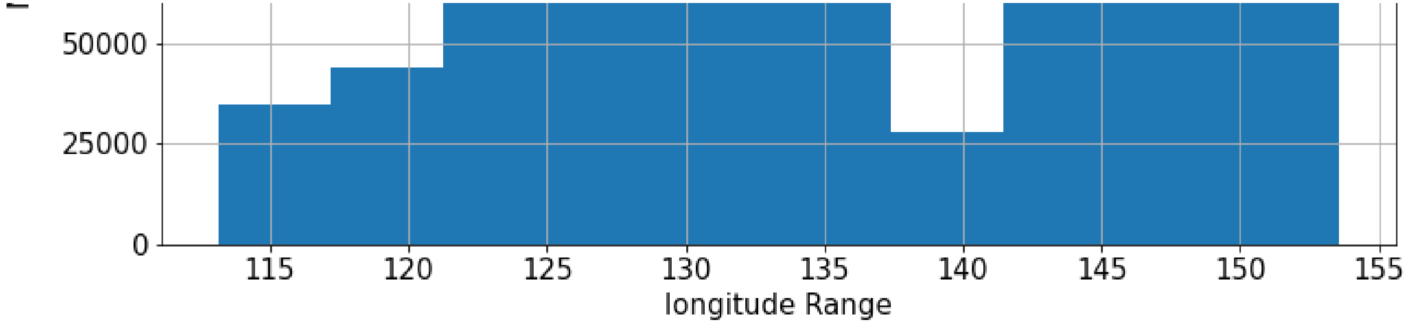
	latitude	longitude	brightness	scan	track	acq_time	confidence	version	bright_t31	frp	type
count	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00	1046679.00
mean	-21.96	135.25	332.88	1.66	1.23	622.02	71.06	6.07	303.23	70.50	0.01
std	7.79	10.52	23.11	0.85	0.26	532.04	22.92	0.07	10.73	169.47	0.17
min	-43.50	113.13	300.00	1.00	1.00	0.00	0.00	6.03	265.70	-29.90	0.00
25%	-28.77	126.80	317.90	1.10	1.00	225.00	56.00	6.03	295.90	14.60	0.00
50%	-19.92	133.14	328.70	1.30	1.10	444.00	74.00	6.03	303.00	28.60	0.00
75%	-15.30	144.96	341.60	1.90	1.40	629.00	90.00	6.20	309.80	63.50	0.00
max	-9.25	153.59	507.00	4.80	2.00	2359.00	100.00	6.20	400.10	11164.10	3.00

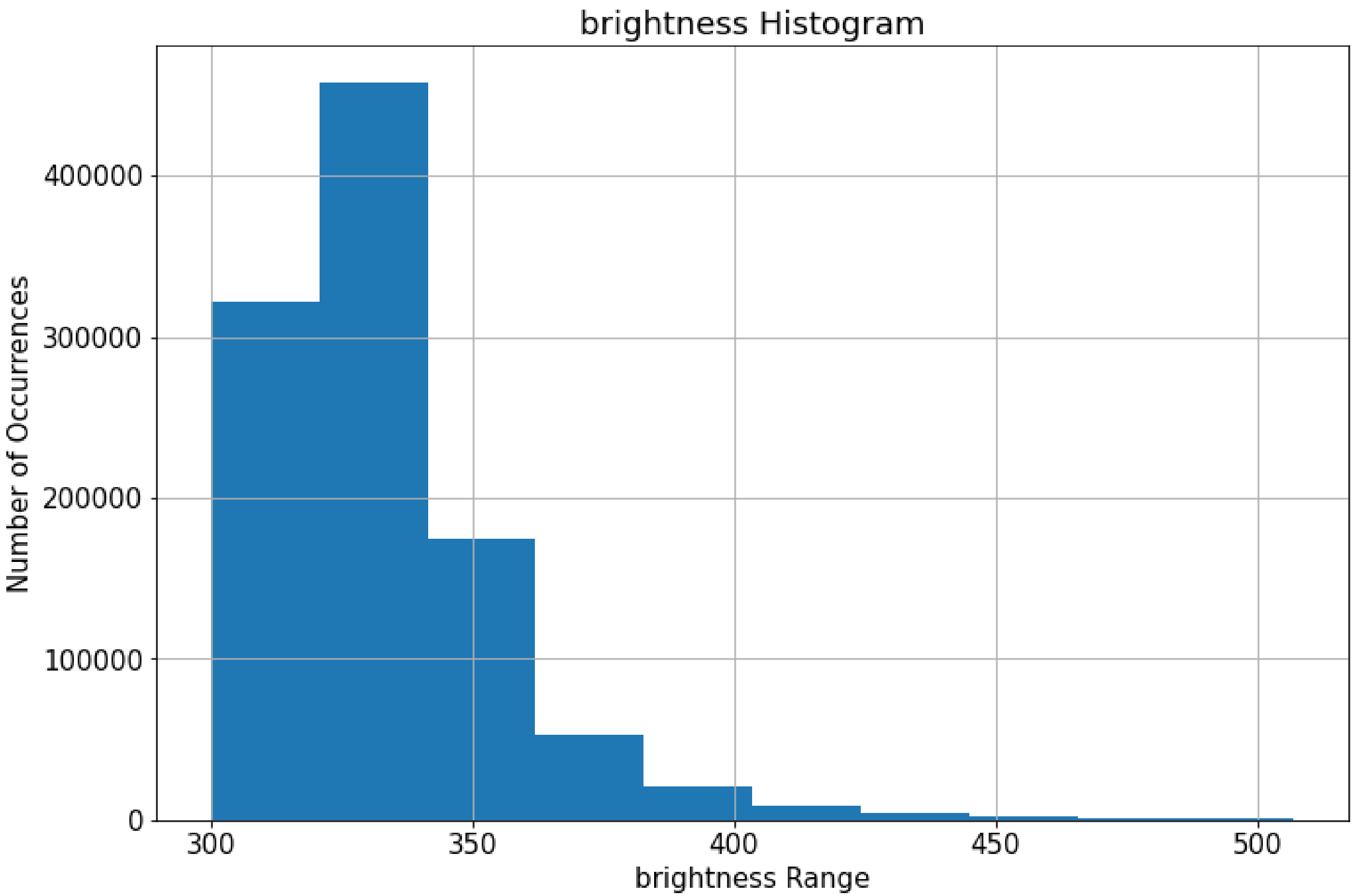
In [22]:

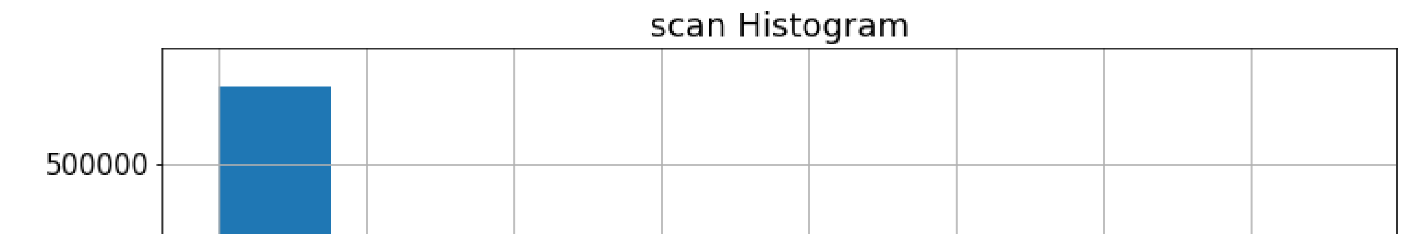
```
continuous_features = dataset_merged.describe(include=['number']).columns
for col in continuous_features:
  if(col!='type' and col!='version'):
    dataset_merged.hist(column=['{}'.format(col)])
    plt.xlabel('{} Range'.format(col))
    plt.ylabel('Number of Occurrences')
    plt.title('{} Histogram'.format(col))
```

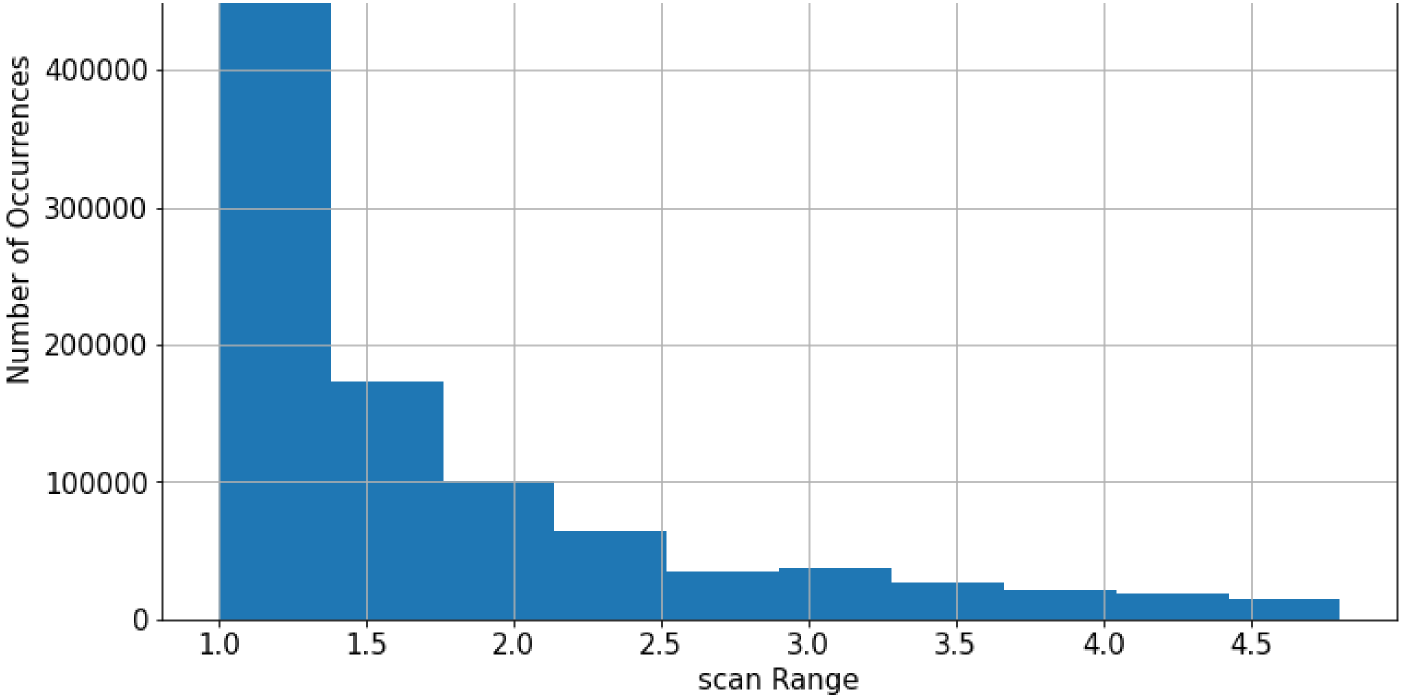

latitude Histogram
Number of Occurrences
250000
200000
150000
100000
50000
0
−45
−40
−35
−30
−25
−20
−15
−10
latitude Range

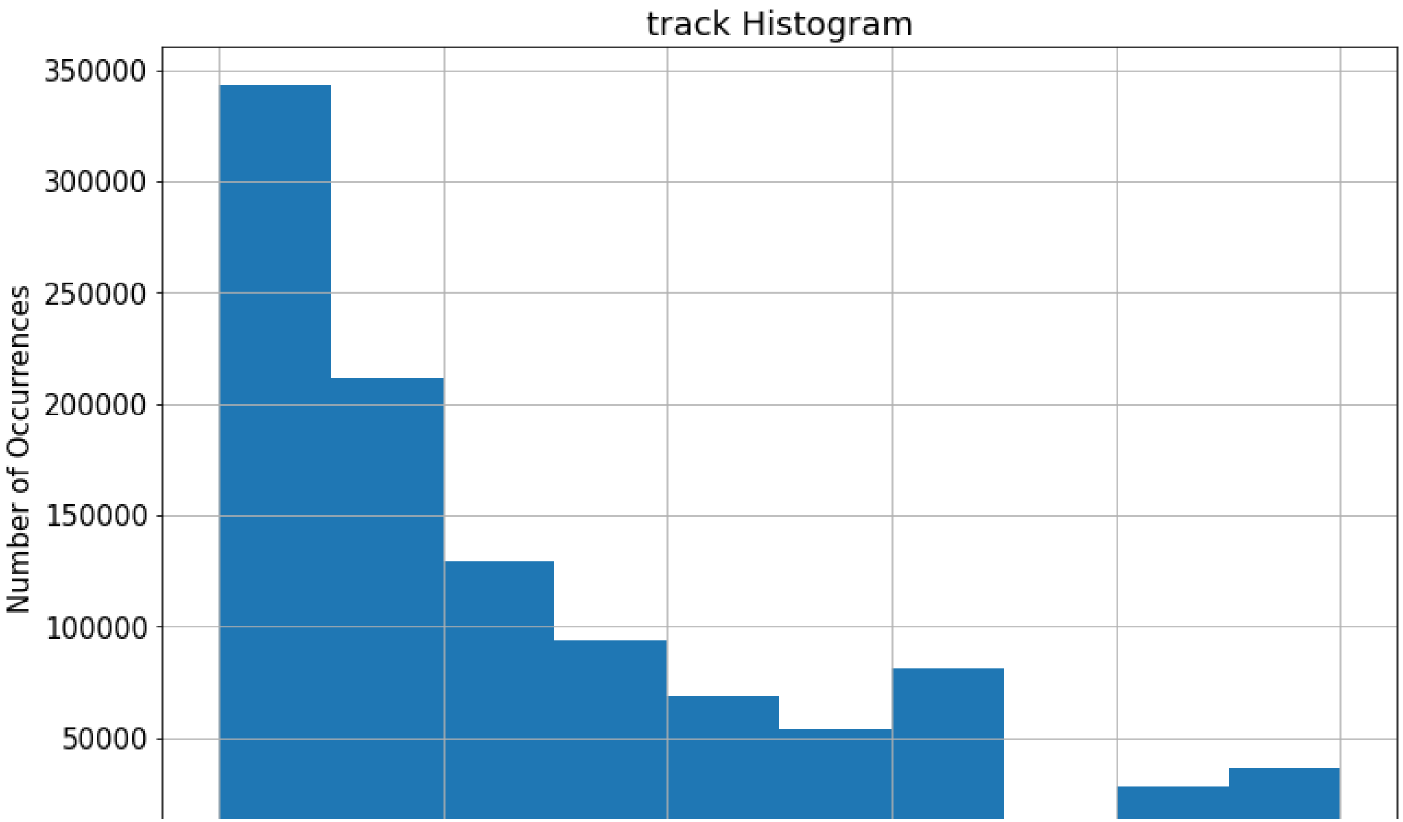

longitude Histogram
Number of Occurrences
175000
150000
125000
100000
75000

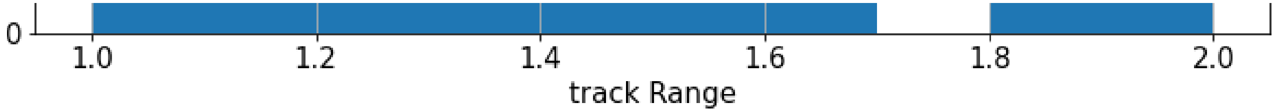

50000
25000
0
115
120
125
130
135
140
145
150
155
longitude Range

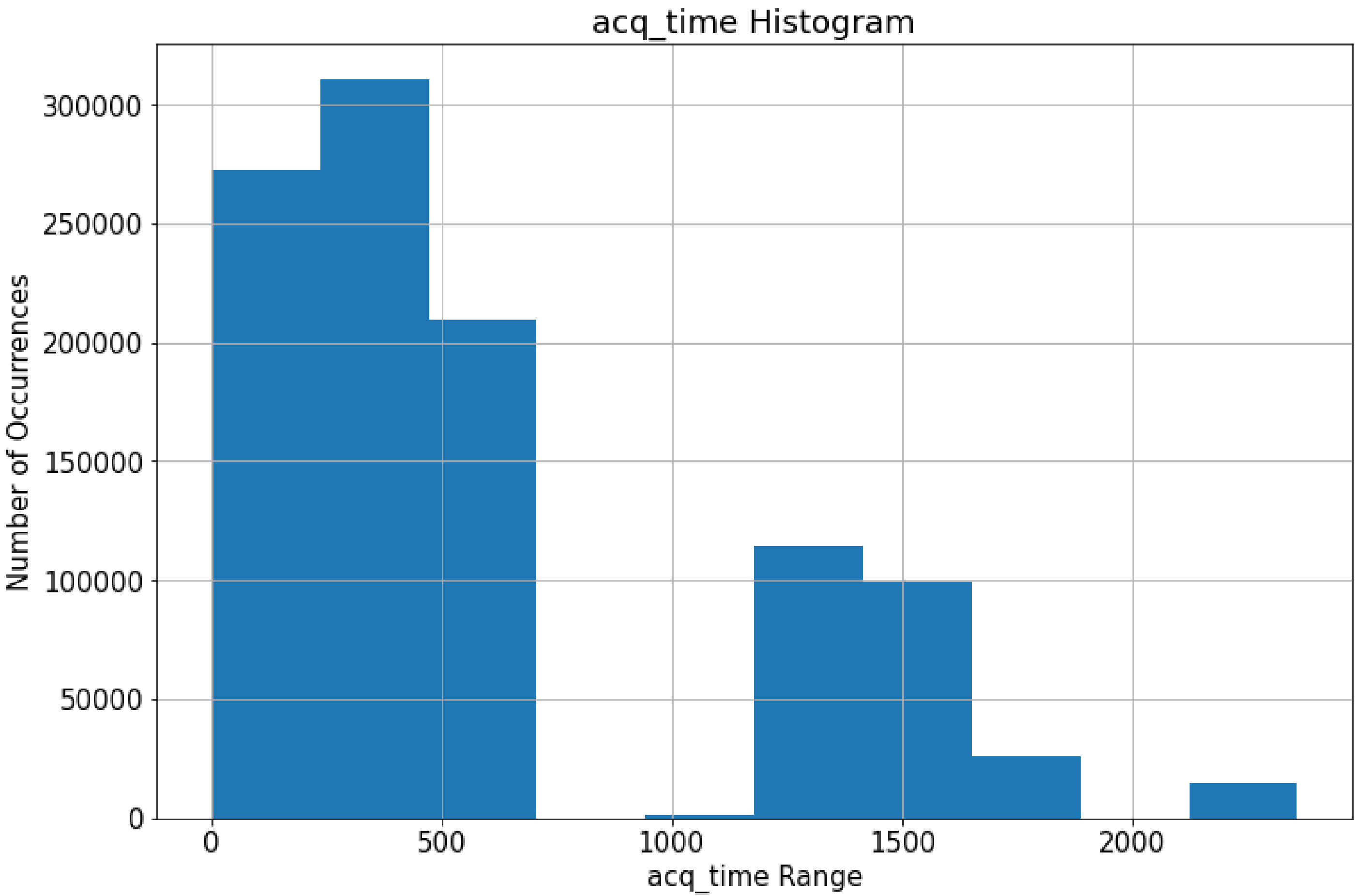

brightness Histogram
Number of Occurrences
400000
300000
200000
100000
0
300
350
400
450
500
brightness Range

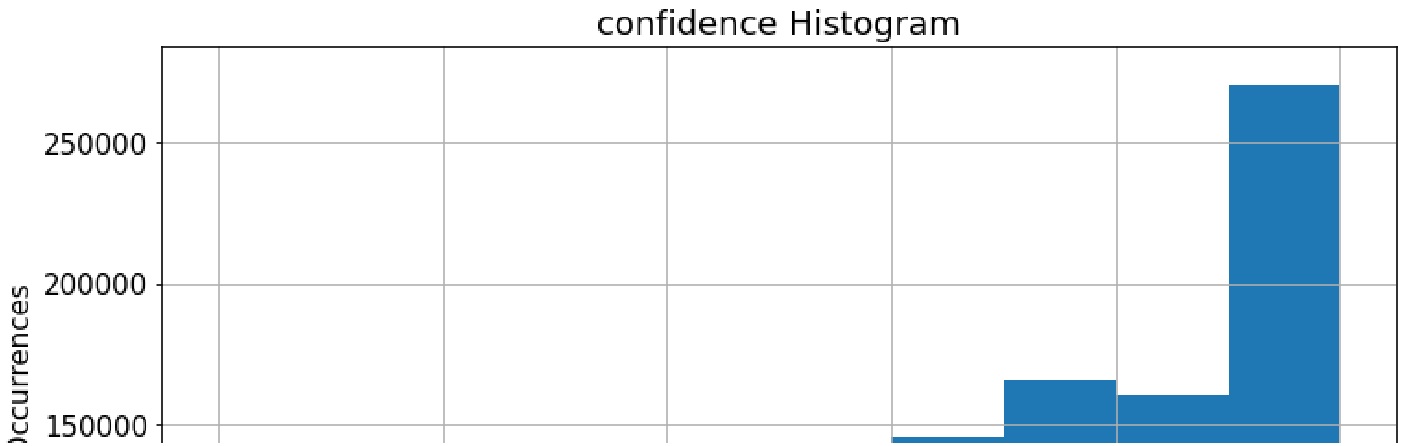

scan Histogram
500000

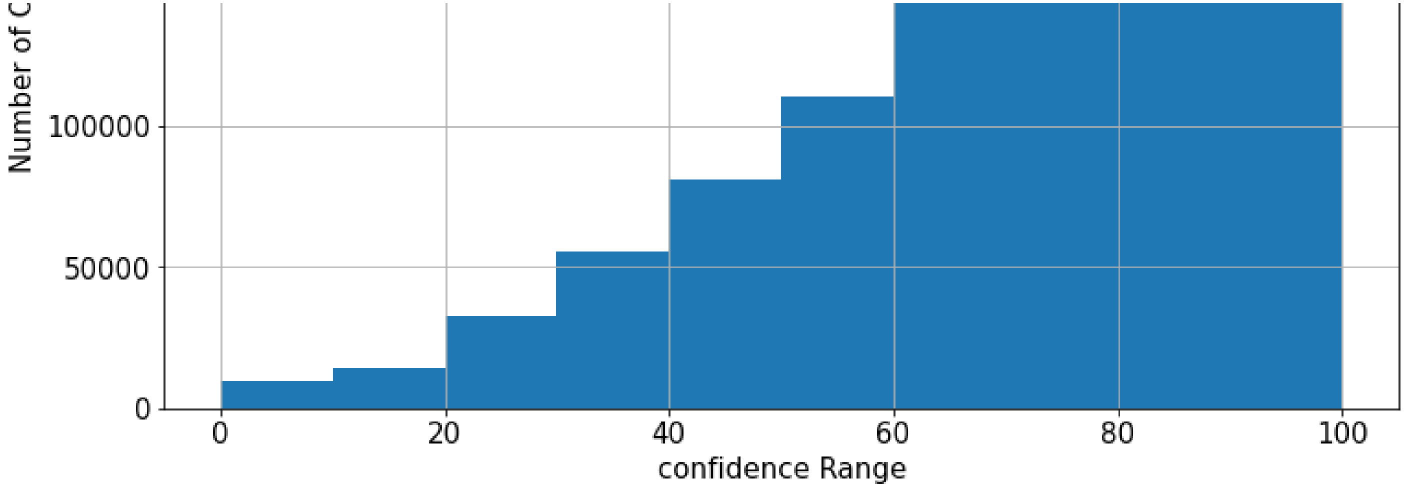

Number of Occurrences
400000
300000
200000
100000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
scan Range

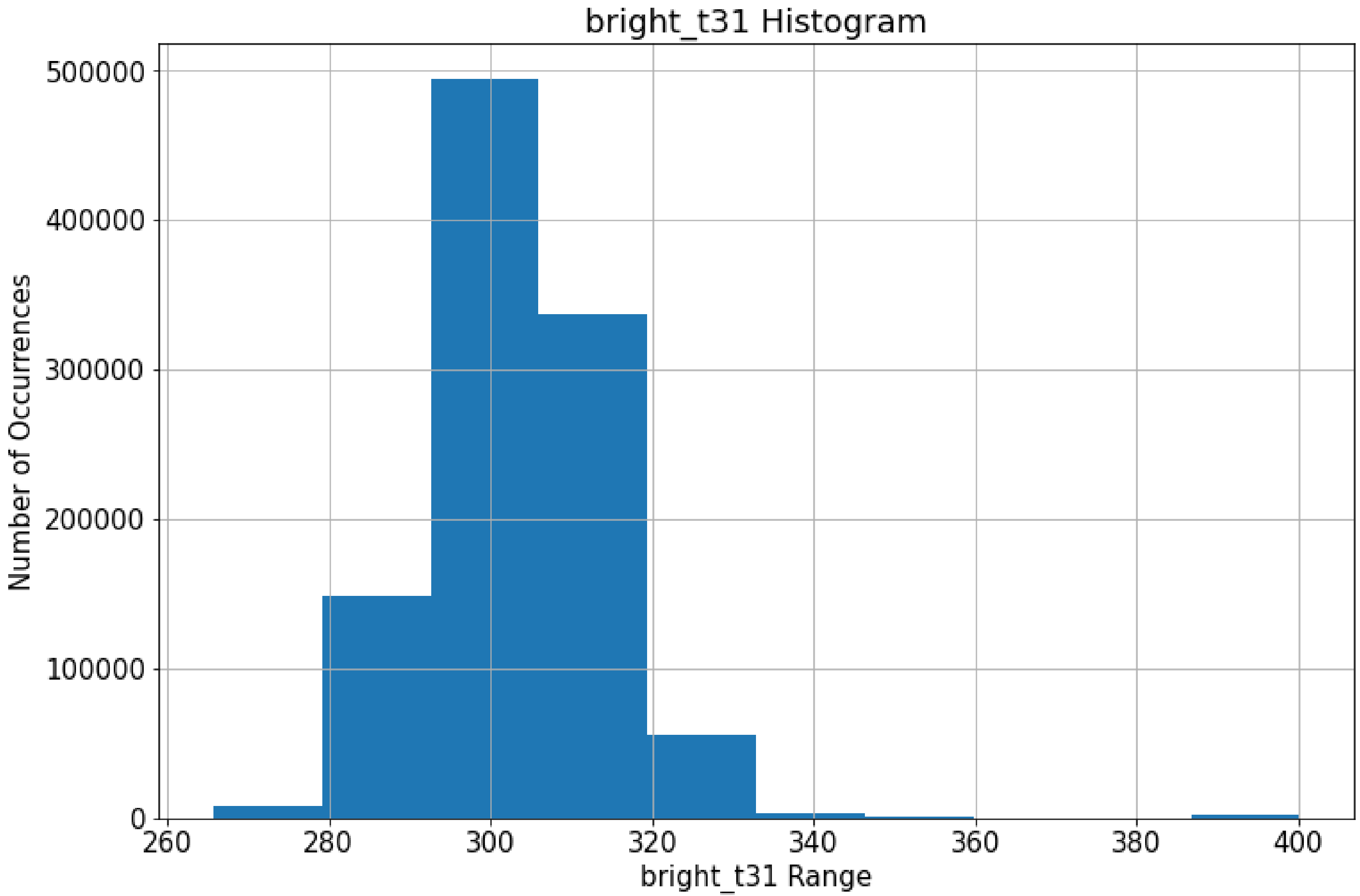

track Histogram
Number of Occurrences
350000
300000
250000
200000
150000
100000
50000


0
1.0
1.2
1.4
1.6
1.8
2.0
track Range

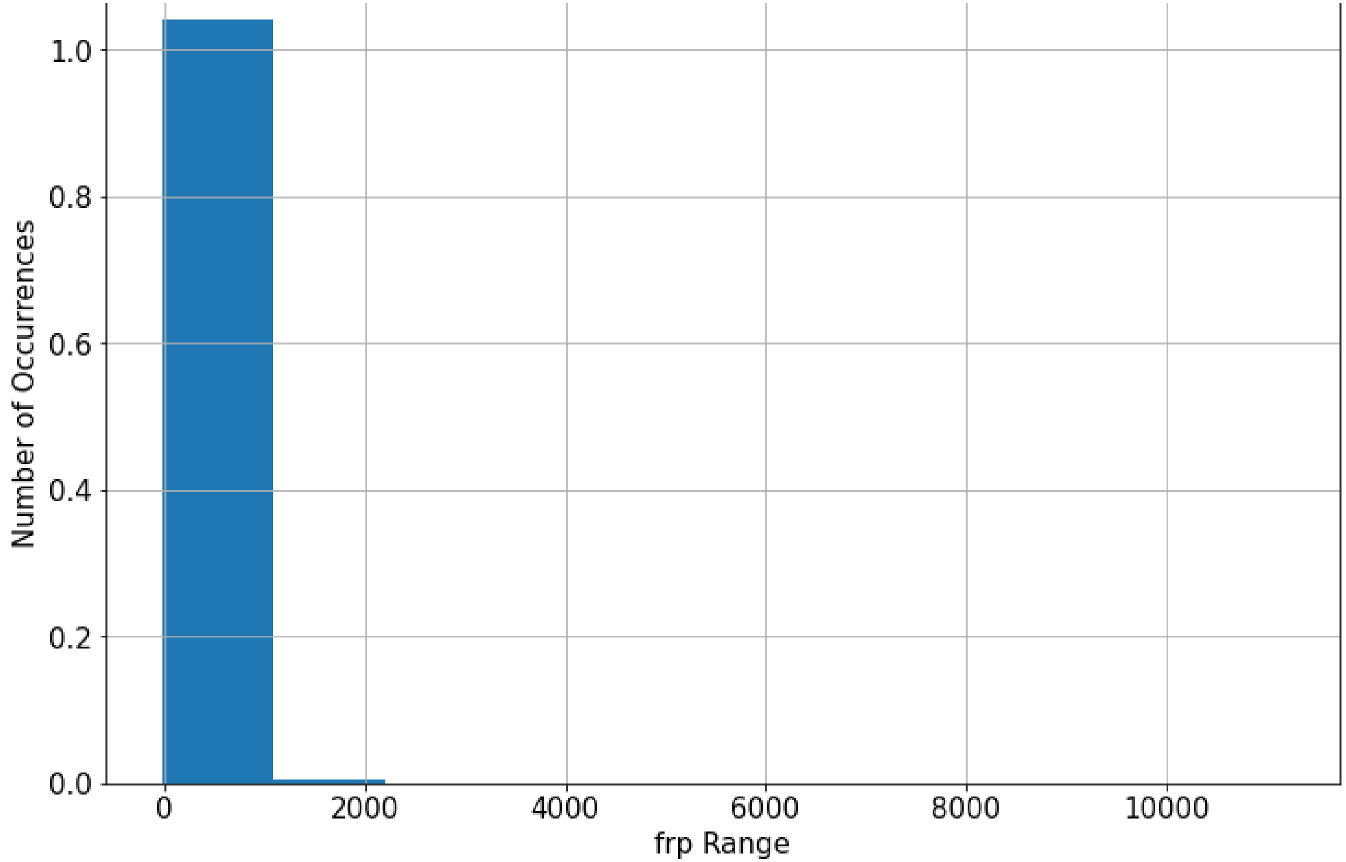

acq_time Histogram
Number of Occurrences
300000
250000
200000
150000
100000
50000
0
0
500
1000
1500
2000
acq_time Range


confidence Histogram
250000
200000
150000


Number of O
100000
50000
0
0
20
40
60
80
100
confidence Range


bright_t31 Histogram
500000
400000
300000
200000
100000
0
Number of Occurrences
260
280
300
320
340
360
380
400
bright_t31 Range


frp Histogram
1e6



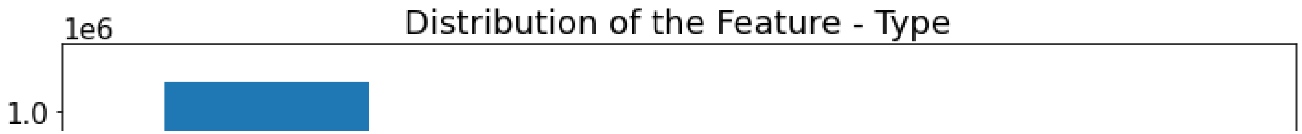
For Continuous Features with Cardinality < 10 we are going to use Bar plots to determine the frequency of values

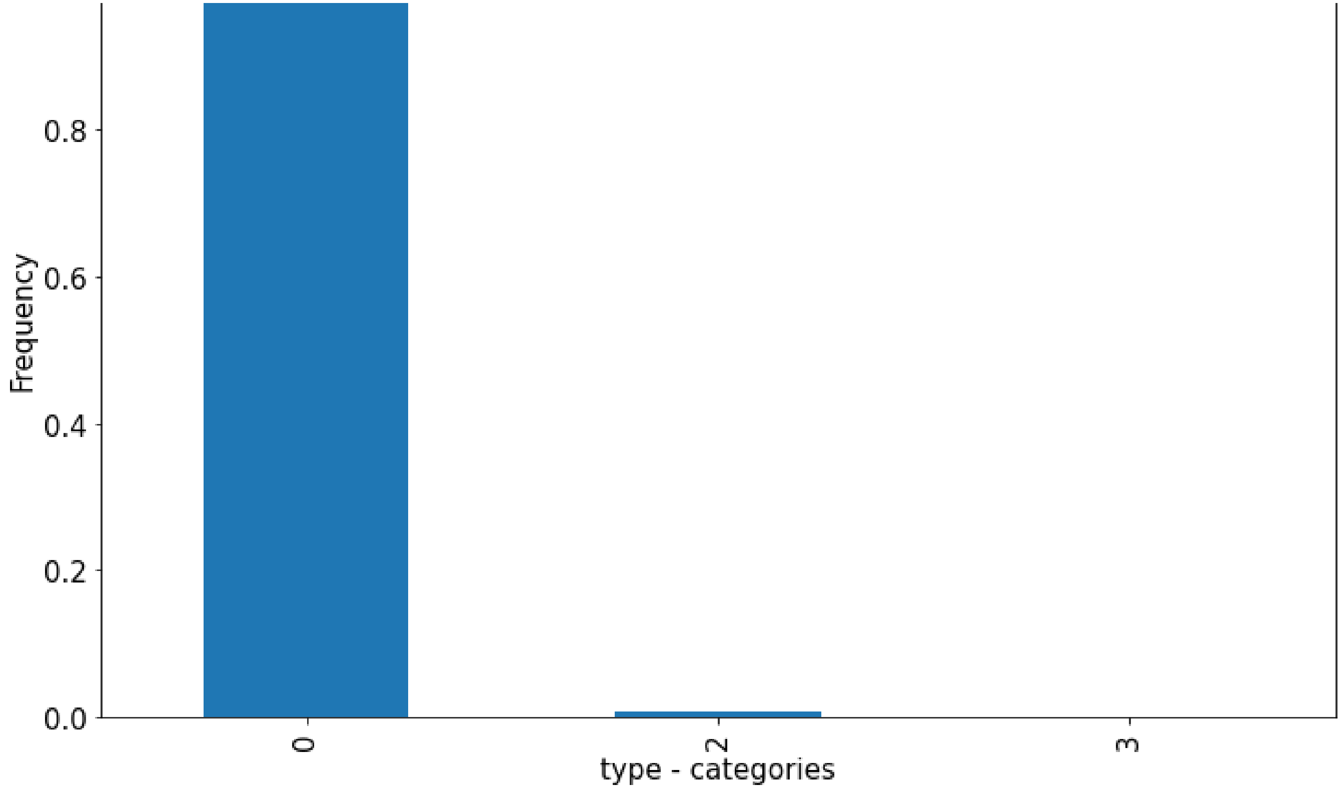
In [23]:

```
dataset_merged['type'].value_counts().plot.bar()
plt.xlabel('type - categories')
plt.ylabel('Frequency')
plt.title('Distribution of the Feature - Type')
#Please note that the Y-axis is in exponential format where 1e6 refers to 10^6 or 1 million
```

Out[23]:

```
Text(0.5, 1.0, 'Distribution of the Feature - Type')
```



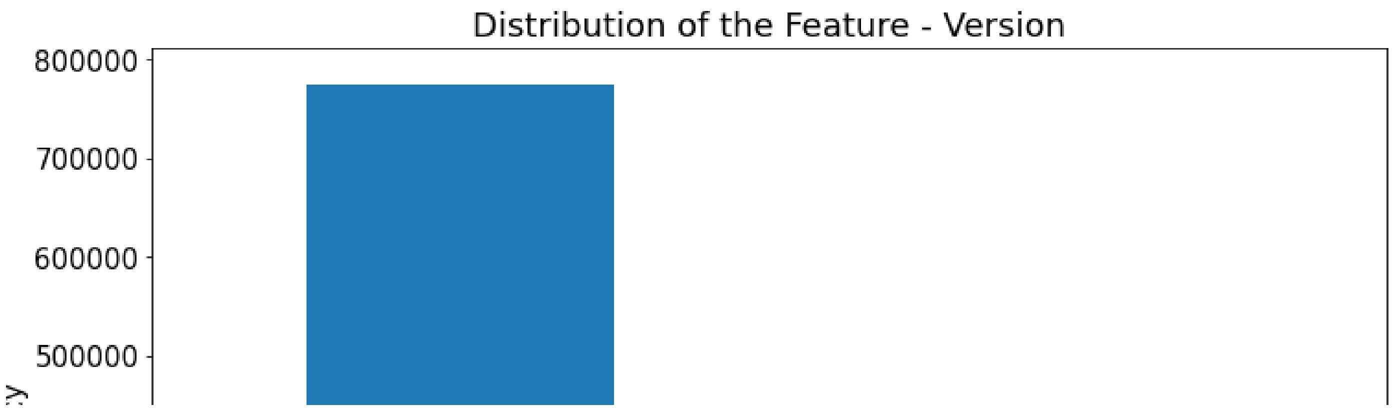


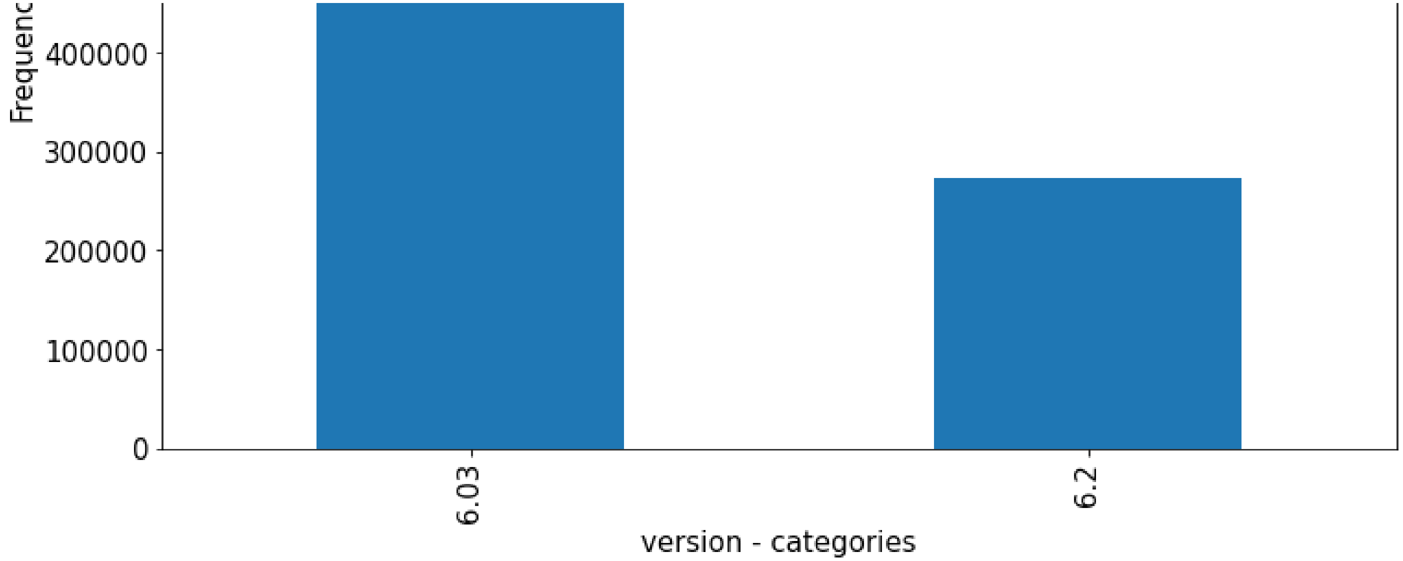
In [24]:

```
dataset_merged['version'].value_counts().plot.bar()
plt.xlabel('version - categories')
plt.ylabel('Frequency')
plt.title('Distribution of the Feature - Version')
```

Out[24]:

```
Text(0.5, 1.0, 'Distribution of the Feature - Version')
```





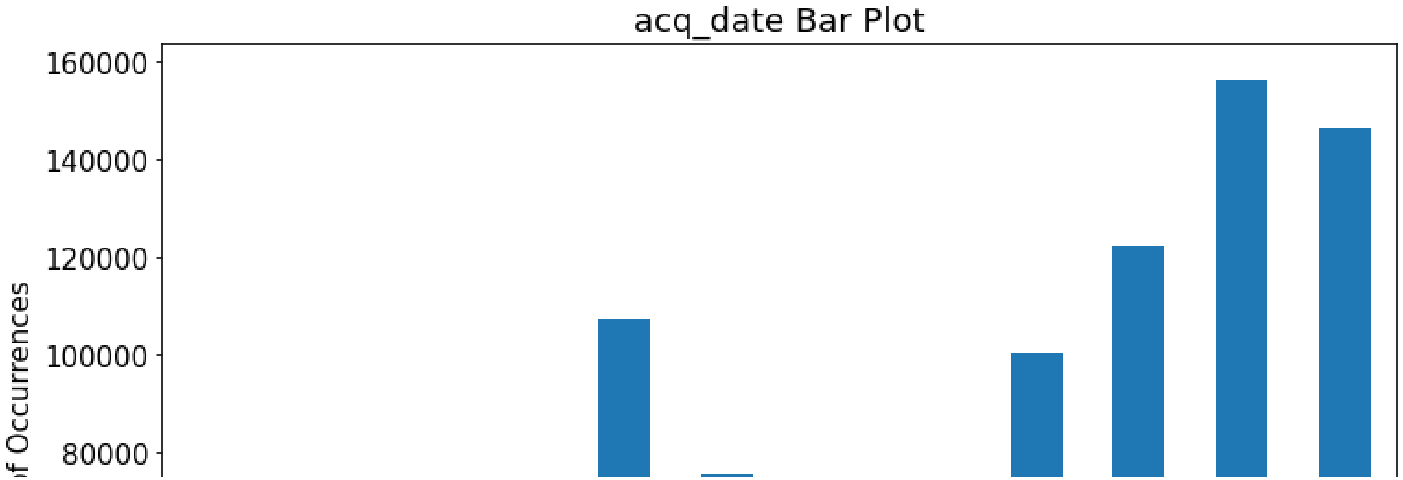
Visualization of Categorical Features

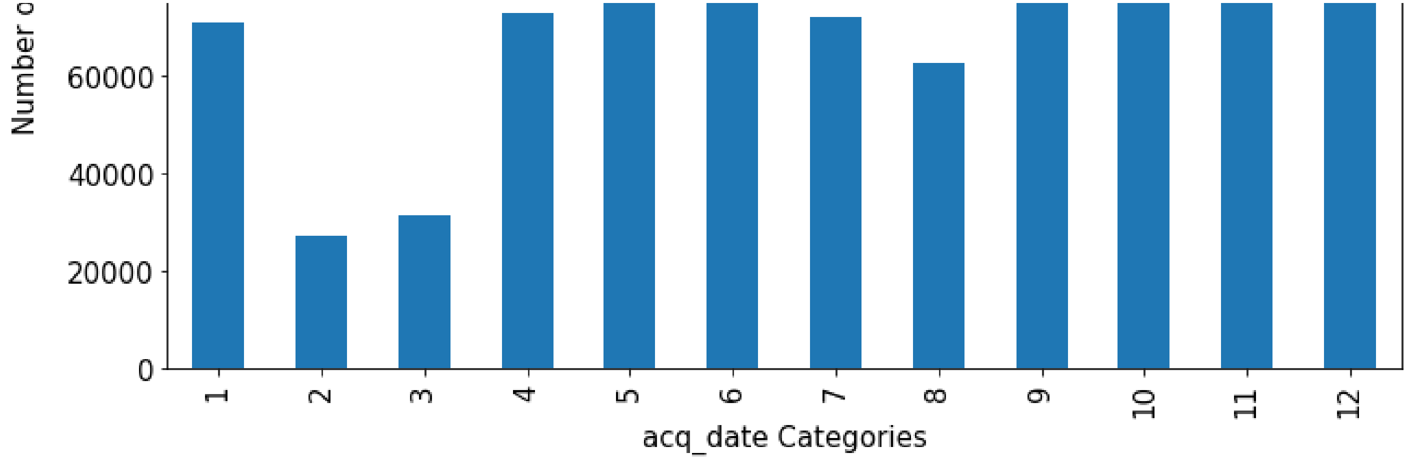
In [25]:

```
dataset_merged["acq_date"] =  pd.to_datetime(dataset_merged["acq_date"], format="%Y/%m/%d")
dataset_merged["acq_date"].groupby(dataset_merged["acq_date"].dt.month).count().plot(kind="bar")
plt.xlabel('acq_date Categories')
plt.ylabel('Number of Occurrences')
plt.title('acq_date Bar Plot')
```

Out[25]:

```
Text(0.5, 1.0, 'acq_date Bar Plot')
```



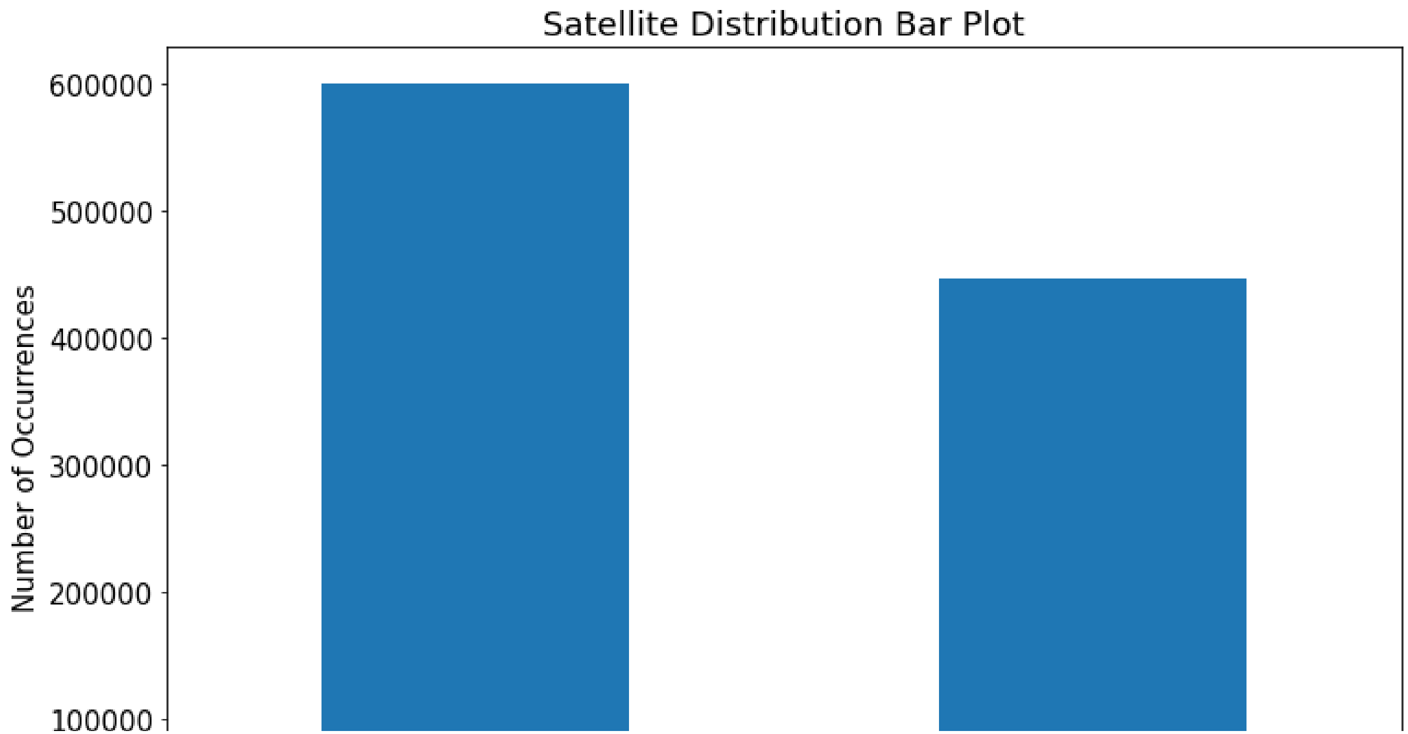


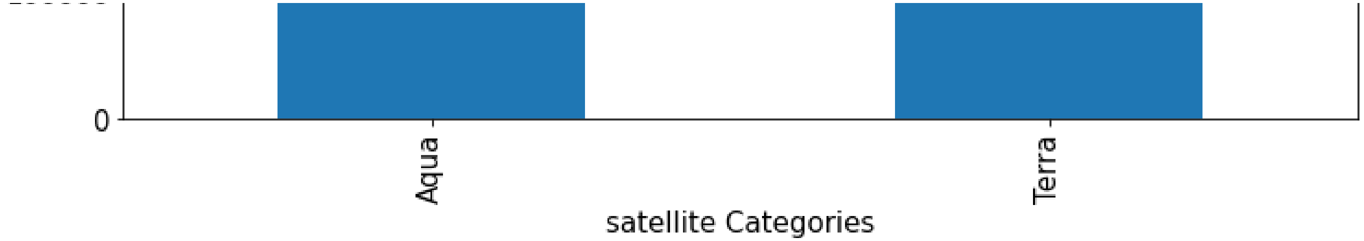
In [26]:

```
dataset_merged['satellite'].value_counts().plot.bar();
plt.xlabel('satellite Categories')
plt.ylabel('Number of Occurrences')
plt.title('Satellite Distribution Bar Plot')
```

Out[26]:

```
Text(0.5, 1.0, 'Satellite Distribution Bar Plot')
```



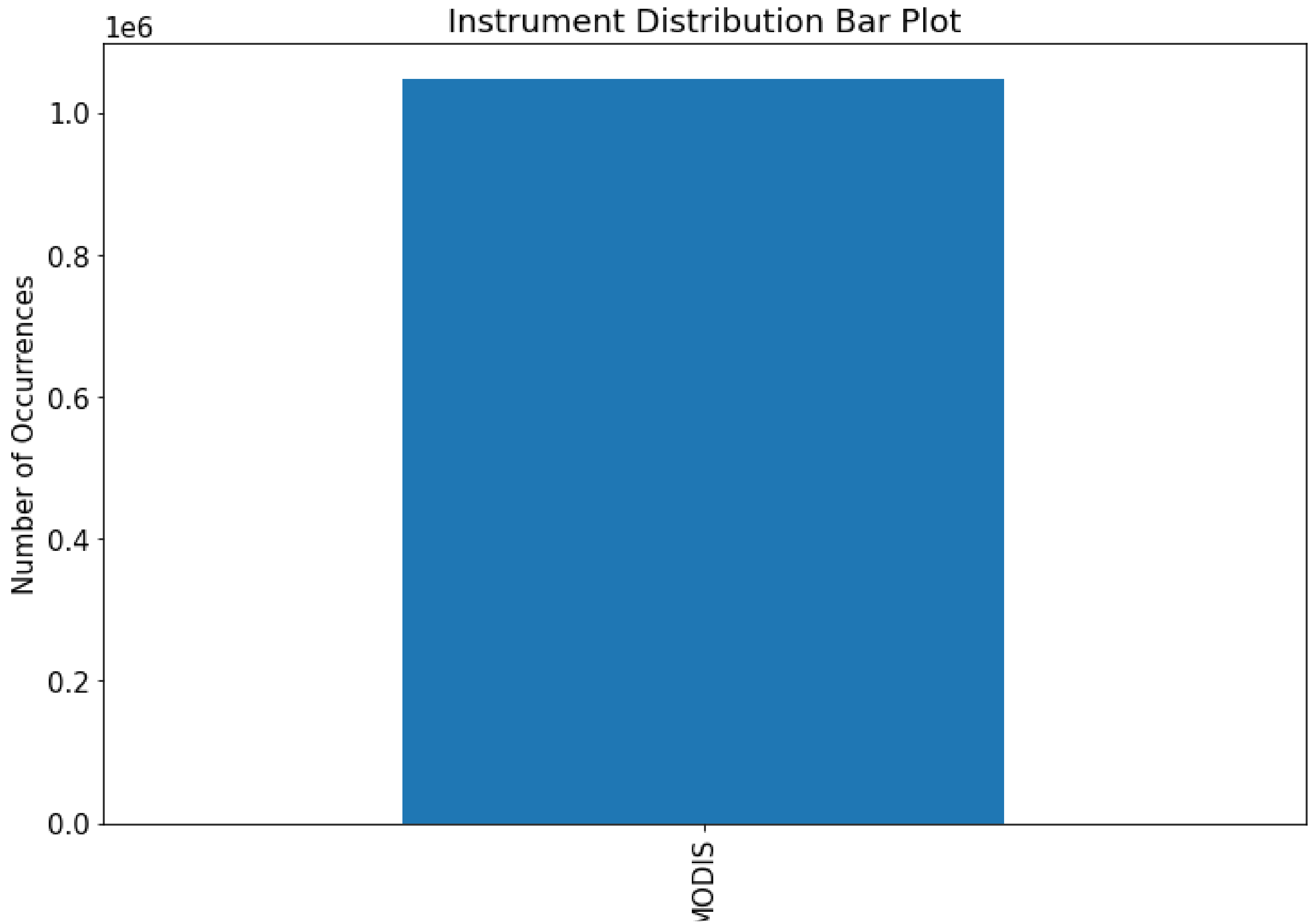


In [27]:

```
dataset_merged['instrument'].value_counts().plot.bar();
plt.xlabel('instrument Categories')
plt.ylabel('Number of Occurrences')
plt.title('Instrument Distribution Bar Plot')
```

Out[27]:

```
Text(0.5, 1.0, 'Instrument Distribution Bar Plot')
```



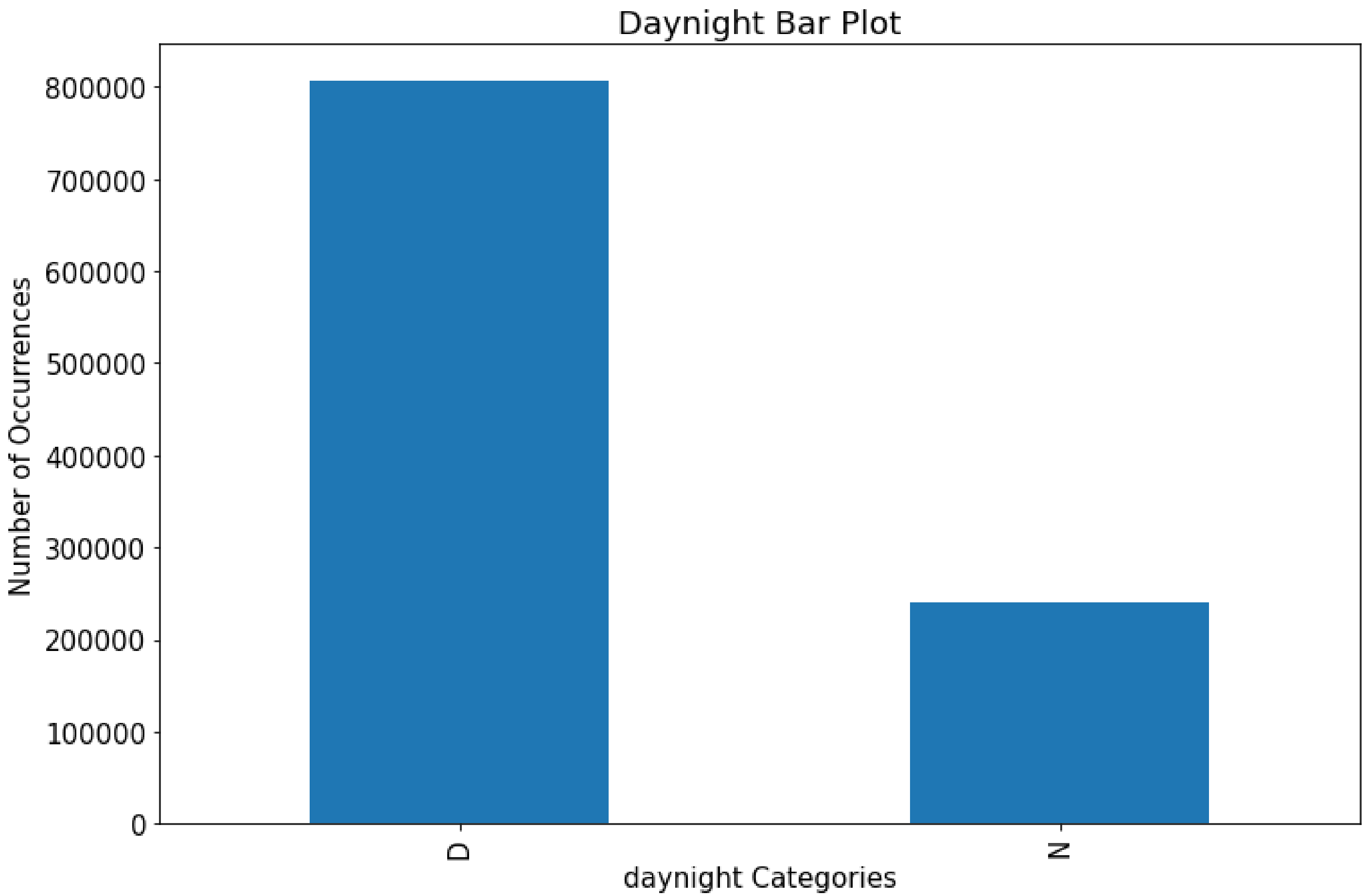
instrument Categories

In [28]:

```
dataset_merged['daynight'].value_counts().plot.bar();
plt.xlabel('daynight Categories')
plt.ylabel('Number of Occurrences')
plt.title('Daynight Bar Plot')
```

Out[28]:

```
Text(0.5, 1.0, 'Daynight Bar Plot')
```



Data Quality Issues & Data Quality Plan

Missing values:

Missing values:

By referring to cells: [Data Quality Report -1]() and [Data Quality Report 2]() (Continuous and Categorical) we can notice that the percentage of missing values across all features is zero. Therefore there we are not required to handle any missing data or values.

In [29]:

```
dataset_merged.isna().sum().sum()
```

Out[29]:

```
0
```

Irregular Cardinality

Continuous Features

The following features will be dropped since they are of little significance and their cardinality is too low for a typical continuous feature.

Scan and Track refer to the acutal pixel size. Version refers to the source of the data. It is of two types: Standard Processing & Near Real-Time.

Feature and Cardinality

Scan - 39

Track - 11

Version - 2

In [30]:

```
dataset_merged.drop('scan', axis=1, inplace=True)
dataset_merged.drop('track', axis=1, inplace=True)
dataset_merged.drop('version', axis=1, inplace=True)
```

Categorical features

The feature - acq_date (Acquired Date) has a cardinality of 1461 which is understandable since the dates span across 4 years ((365*4)+1).

We dealt with this by first transforming dates to week numbers

Example: 2019-12-25 will be resolved to week number 52.

In [31]:

```
dataset_merged['acq_week']=dataset_merged['acq_date'].dt.week
dataset_merged.head(5)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Series.dt.weekofyear and Series.dt.week have been d
eprecated.  Please use Series.dt.isocalendar().week instead.
  """Entry point for launching an IPython kernel.
```

Out[31]:

	latitude	longitude	brightness	acq_date	acq_time	satellite	instrument	confidence	bright_t31	frp	daynight	type	acq_week
0	-23.91	147.30	320.10	2017-01-01	47	Terra	MODIS	53	296.60	17.60	D	0	52
1	-23.69	150.10	314.30	2017-01-01	47	Terra	MODIS	22	289.30	30.00	D	0	52
2	-23.59	150.17	315.80	2017-01-01	47	Terra	MODIS	33	291.70	35.40	D	0	52
3	-22.41	148.85	316.70	2017-01-01	47	Terra	MODIS	26	295.30	25.80	D	0	52
4	-20.59	147.64	320.70	2017-01-01	47	Terra	MODIS	34	299.60	19.00	D	0	52

Now, we have a column - 'acq_week' with a cardinality of 53. We could have aggregated the dates to months too but since the assignment deals with dates, we presumed that it would be ideal to retain the granularity of the dates as close to the original granularity (days).

In [32]:

```
build_continuous_features_report(dataset_merged)
```

Out[32]:

	Count	Miss %	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt	Max	Std. Dev.
latitude	1046679	0.00	259382	-43.50	-28.77	-21.96	-19.92	-15.30	-9.25	7.79
longitude	1046679	0.00	325532	113.13	126.80	135.25	133.14	144.96	153.59	10.52
brightness	1046679	0.00	2050	300.00	317.90	332.88	328.70	341.60	507.00	23.11
acq_time	1046679	0.00	855	0.00	225.00	622.02	444.00	629.00	2359.00	532.04
confidence	1046679	0.00	101	0.00	56.00	71.06	74.00	90.00	100.00	22.92
bright_t31	1046679	0.00	972	265.70	295.90	303.23	303.00	309.80	400.10	10.73
frp	1046679	0.00	13993	-29.90	14.60	70.50	28.60	63.50	11164.10	169.47
type	1046679	0.00	3	0.00	0.00	0.01	0.00	0.00	3.00	0.17
acq_week	1046679	0.00	53	1.00	19.00	31.40	35.00	45.00	53.00	15.03

Variables and Correlation

In order to determine how features in the dataset are related to each other we need to calculate the correlation of the features/columns.

In [33]:

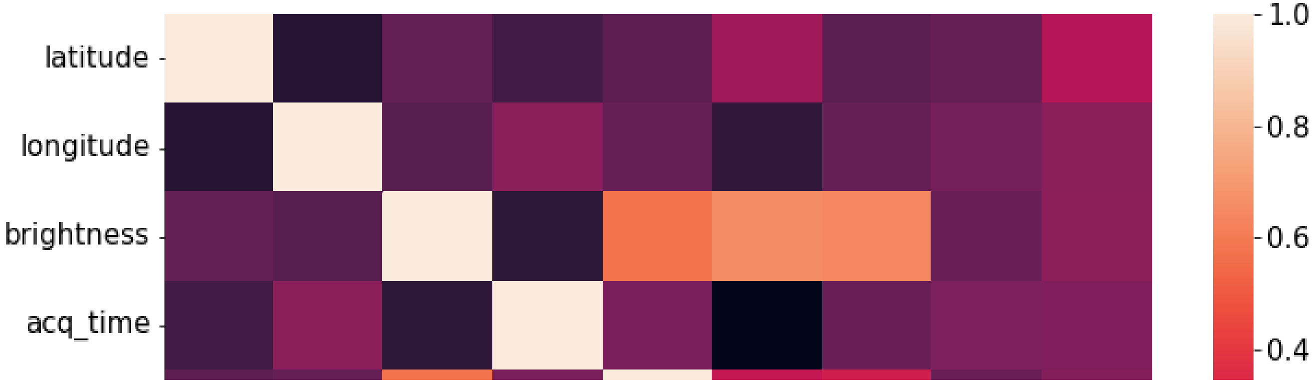
```
dataset_merged.corr(method='pearson')
```

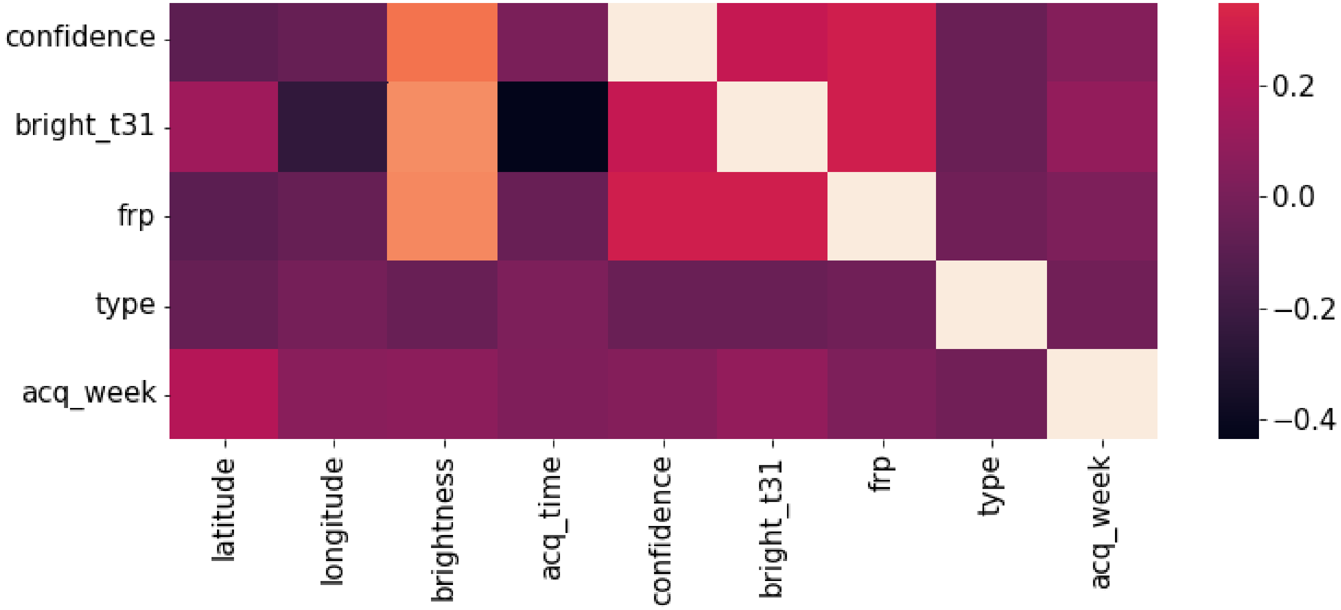
Out[33]:

	latitude	longitude	brightness	acq_time	confidence	bright_t31	frp	type	acq_week
latitude	1.00	-0.29	-0.06	-0.18	-0.09	0.13	-0.10	-0.06	0.20
longitude	-0.29	1.00	-0.10	0.06	-0.06	-0.24	-0.06	-0.01	0.06
brightness	-0.06	-0.10	1.00	-0.26	0.58	0.66	0.64	-0.05	0.07
acq_time	-0.18	0.06	-0.26	1.00	0.01	-0.43	-0.05	0.02	0.03
confidence	-0.09	-0.06	0.58	0.01	1.00	0.26	0.30	-0.04	0.04
bright_t31	0.13	-0.24	0.66	-0.43	0.26	1.00	0.30	-0.05	0.09
frp	-0.10	-0.06	0.64	-0.05	0.30	0.30	1.00	-0.02	0.02
type	-0.06	-0.01	-0.05	0.02	-0.04	-0.05	-0.02	1.00	-0.02
acq_week	0.20	0.06	0.07	0.03	0.04	0.09	0.02	-0.02	1.00

In [34]:

```
dataset_merged.style.background_gradient(cmap='Greens')
import matplotlib.pyplot as plt
import seaborn as sb
dataplot=sb.heatmap(dataset_merged.corr())
plt.show()
```





Question 1.3

Q 1.3-A:

What are the dates on which bushfires present the high number of incidents?

Answer:

In [35]:

```
dataframe_3_a = dataset_merged[dataset_merged['type']==0].drop(['latitude','longitude','acq_time','satellite','instrument','daynight','acq_week'], axis=1)
dataframe_3_a.head(10)
# dataframe_3_a.groupby(['acq_date'])['acq_date'].count().sort_values()
```

Out[35]:

	brightness	acq_date	confidence	bright_t31	frp	type
0	320.10	2017-01-01	53	296.60	17.60	0
1	314.30	2017-01-01	22	289.30	30.00	0
2	315.80	2017-01-01	33	291.70	35.40	0
3	316.70	2017-01-01	26	295.30	25.80	0

	brightness	acq_date	confidence	bright_t31	frp	type
4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
5	320.50	2017-01-01	36	299.70	22.90	0
6	314.60	2017-01-01	51	294.50	28.60	0
7	320.30	2017-01-01	28	300.20	25.70	0
8	324.00	2017-01-01	65	300.00	40.40	0
9	327.80	2017-01-01	40	304.00	34.90	0

In [36]:

```
dataframe_3_a.groupby(['acq_date'])['acq_date'].count().sum()
```

Out[36]:

```
1039215
```

In [37]:

```
dataframe_3_a.nunique()
```

Out[37]:

```
brightness     2050
acq_date       1461
confidence      101
bright_t31      972
frp           13981
type              1
dtype: int64
```

In order to determine the dates where the number of incidents are "high" we have taken the 8oth percentile as the lower limit. Therefore dates which have number of incidents GREATER THAN the value of the 99th percentile are treated as those with "HIGH" number of bushfire incidents. Please note that we have considered only instances of Bushfires, that is, type = 0.

In [38]:

```
incidents = dataframe_3_a.groupby(['acq_date'])['brightness'].count().to_frame()
threshold = incidents.brightness.quantile(0.99)
print(threshold)
# total_incidents = dataframe_3_a.groupby(['acq_date'])['acq_date'].count().sum()
# total_dates = 1461
# average_incidents = total_incidents / total_dates
```

```
3304.6000000000013
```

In [39]:

```
# dataframe_3_a[dataframe_3_a.groupby(['acq_date'])['acq_date'].count()>average_incidents]
dates = incidents[incidents['brightness']>threshold]
dates.sort_values("brightness", axis = 0, ascending = True,inplace = True)
dates
```

```
/usr/local/lib/python3.7/dist-packages/pandas/util/_decorators.py:311: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  return func(*args, **kwargs)
```

Out[39]:

acq_date	brightness
2018-10-11	3313
2019-11-12	3335
2019-12-31	3430
2019-12-05	3483
2020-01-01	3672
2019-12-20	3700
2019-12-06	3785
2019-12-18	3995
2019-12-21	4009
2019-11-08	4086
2020-01-03	4243
2020-01-02	4390
2019-12-19	4762
2019-12-30	6909
2020-01-04	7339

Q 1.3-B:

Based on the data quality report, which attributes do you think are useful to predict the confidence of an incident? Explain why you think that the selected attribute is important.

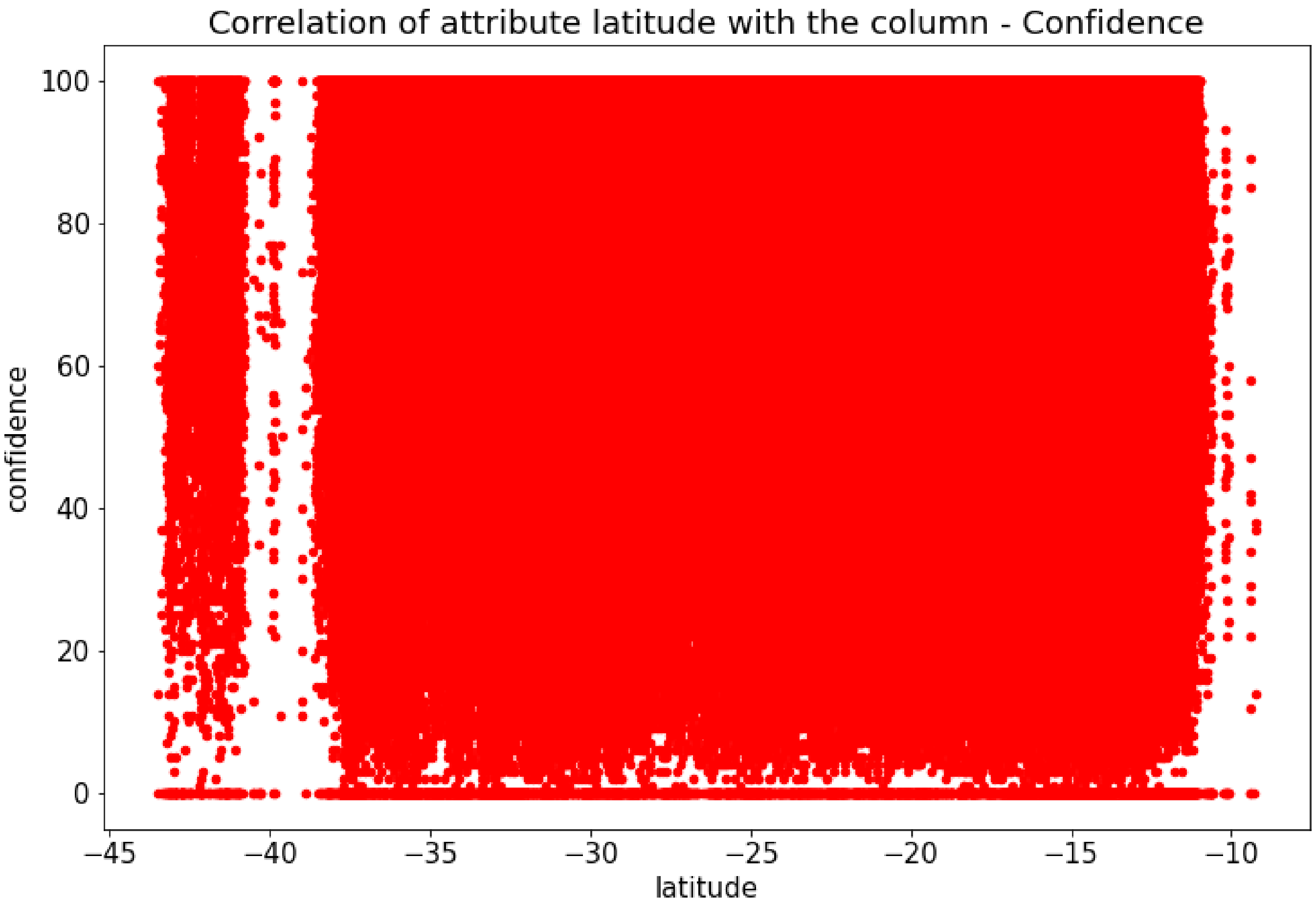
Answer: Based on the data quality report and the correlation heatmap, in my opinion, the attributes **Brightness** and **FRP** are useful to predict the confidence of an

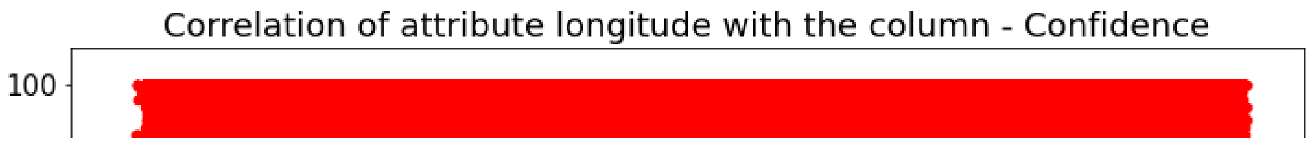
incident. This is because these attributes have relatively higher coo and elation with Confidence than the other attributes, with values of 0.58 and 0.30 respectively.

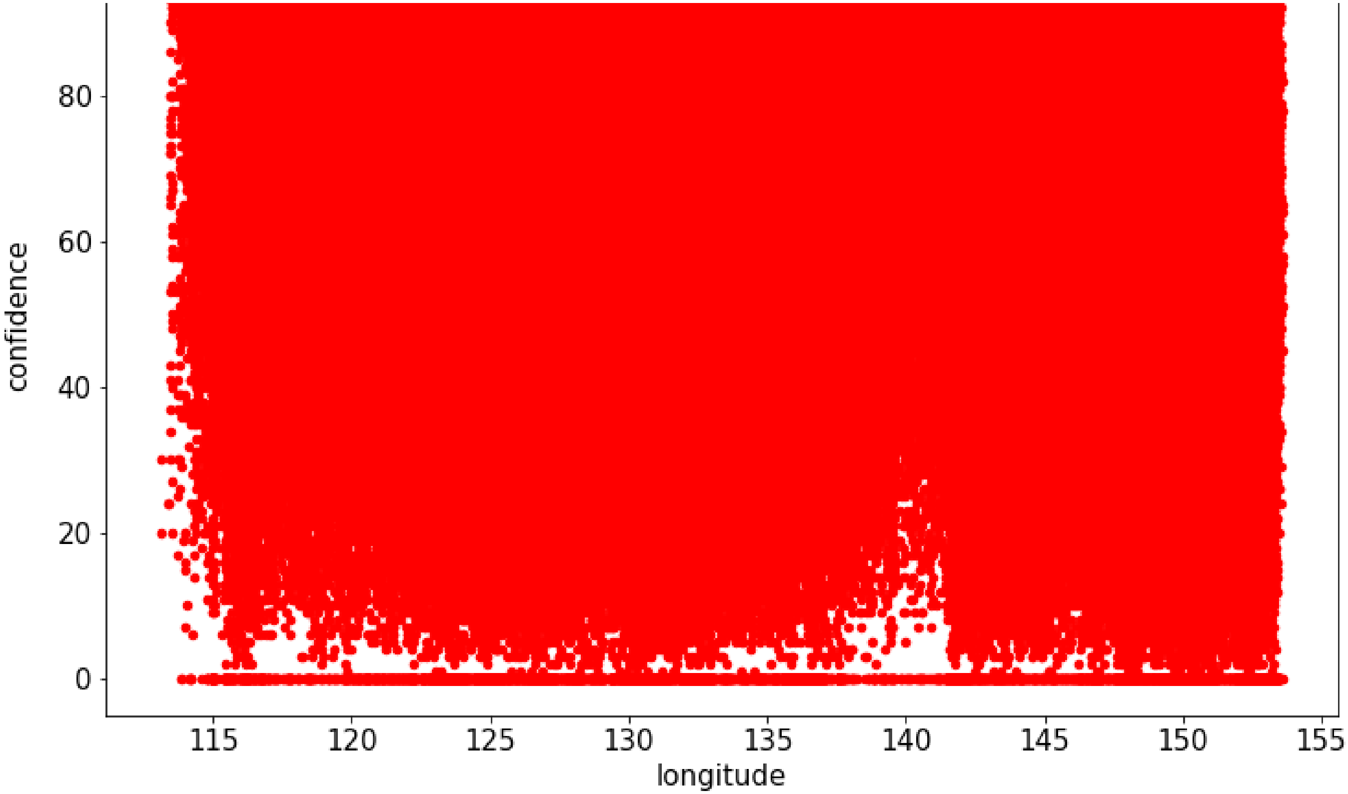
In addition, by referring to the Scatter Plots below (Cell 48) we can observe that Brightness and FRP are more correlated to confidence than other attributes, since we know that the closer the data points come to forming a straight line when plotted, the higher the correlation between the two variables, or the stronger the relationship between the variables. [Interpreting Scatter Plots]

In [40]:

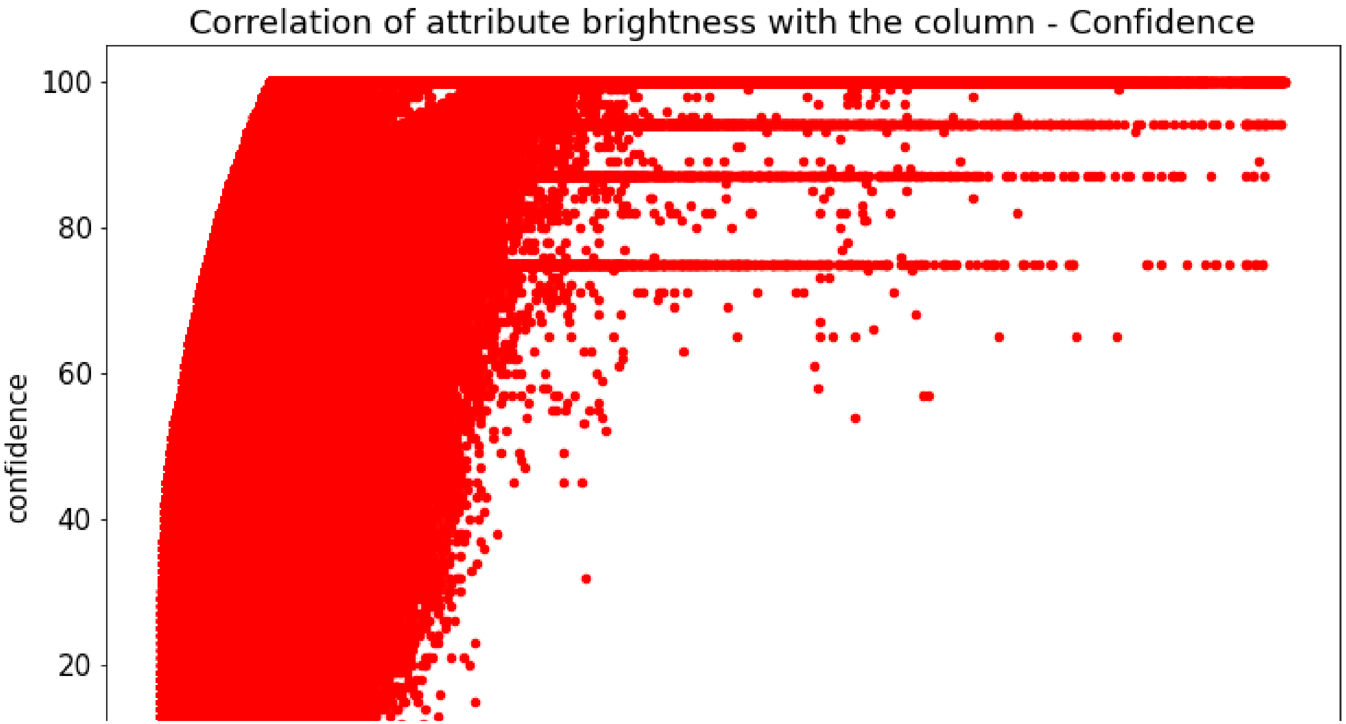
```
for col in dataset_merged.describe(include=['number']).columns:
  if(col!='confidence'):
    # print("Correlation of attribute {} with the column - Confidence".format(col))
    dataset_merged.plot.scatter(x='{}'.format(col),y='confidence',c='red')
    plt.title("Correlation of attribute {} with the column - Confidence".format(col))
```



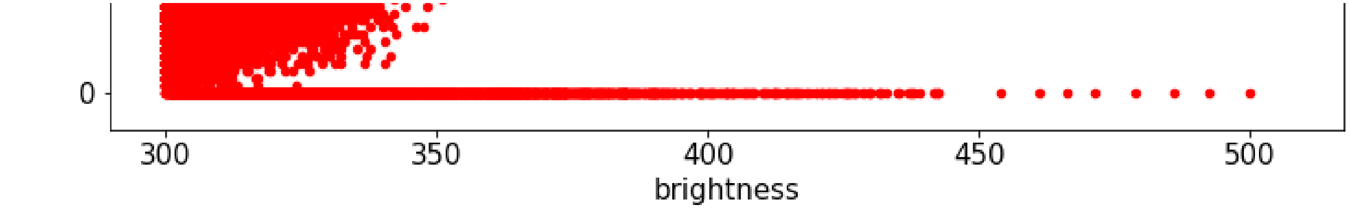




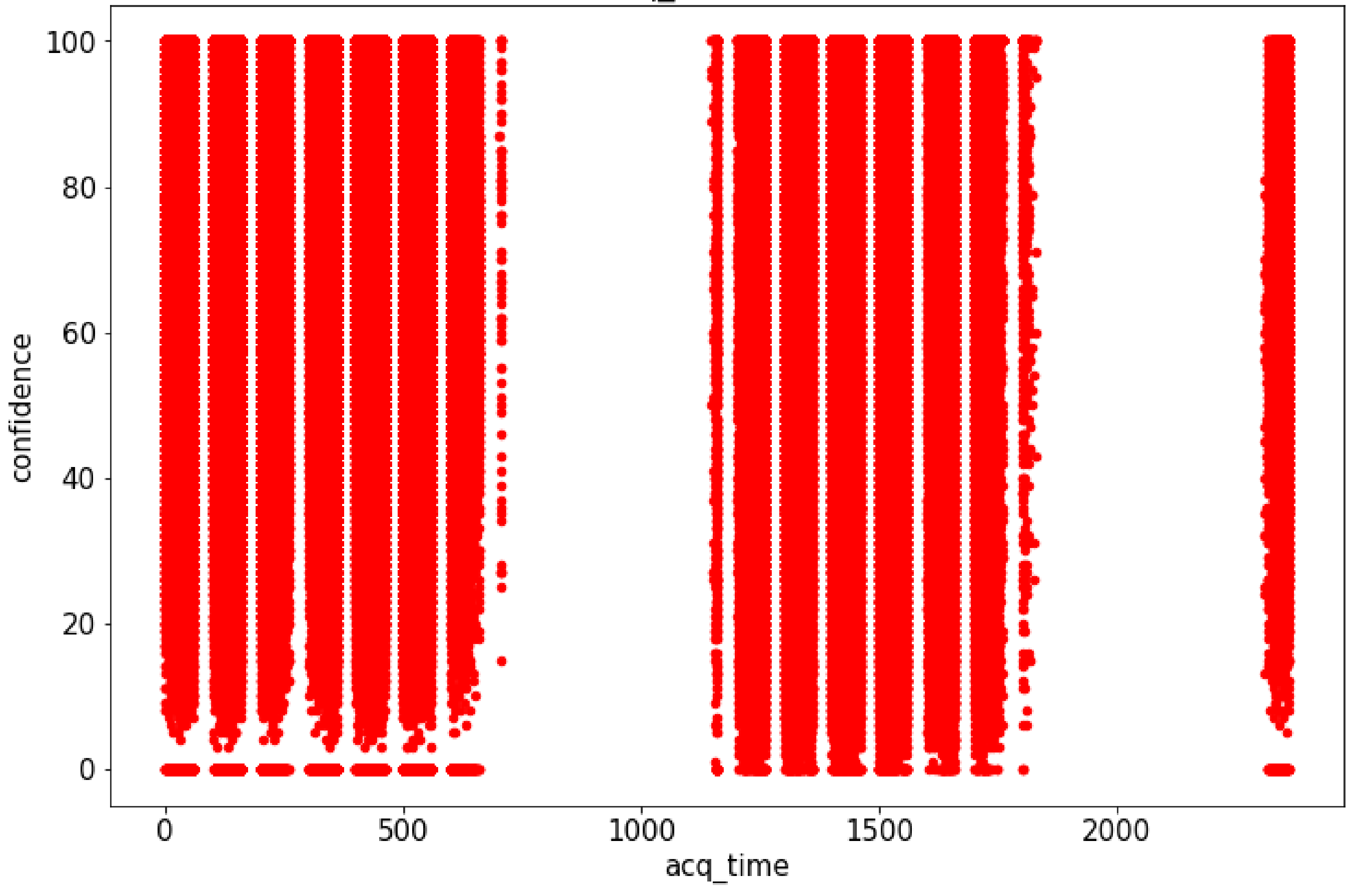
80
60
40
20
0
confidence
115
120
125
130
135
140
145
150
155
longitude



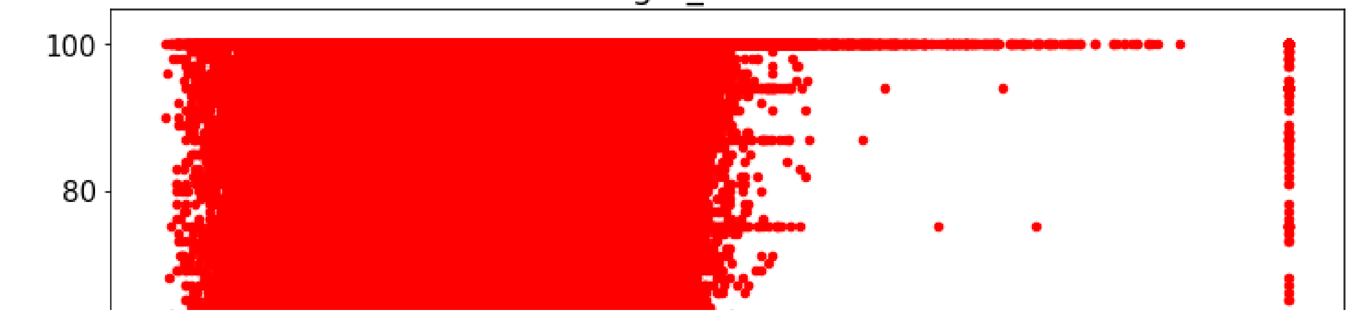
Correlation of attribute brightness with the column - Confidence
100
80
60
40
20
confidence

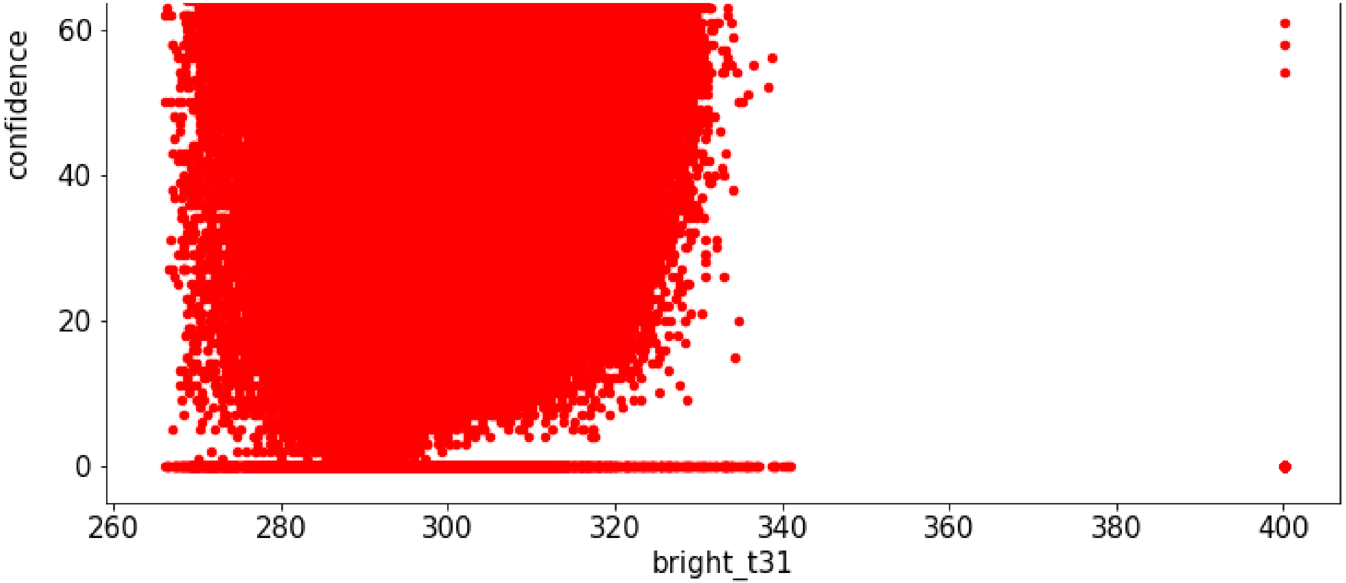


Correlation of attribute acq_time with the column - Confidence

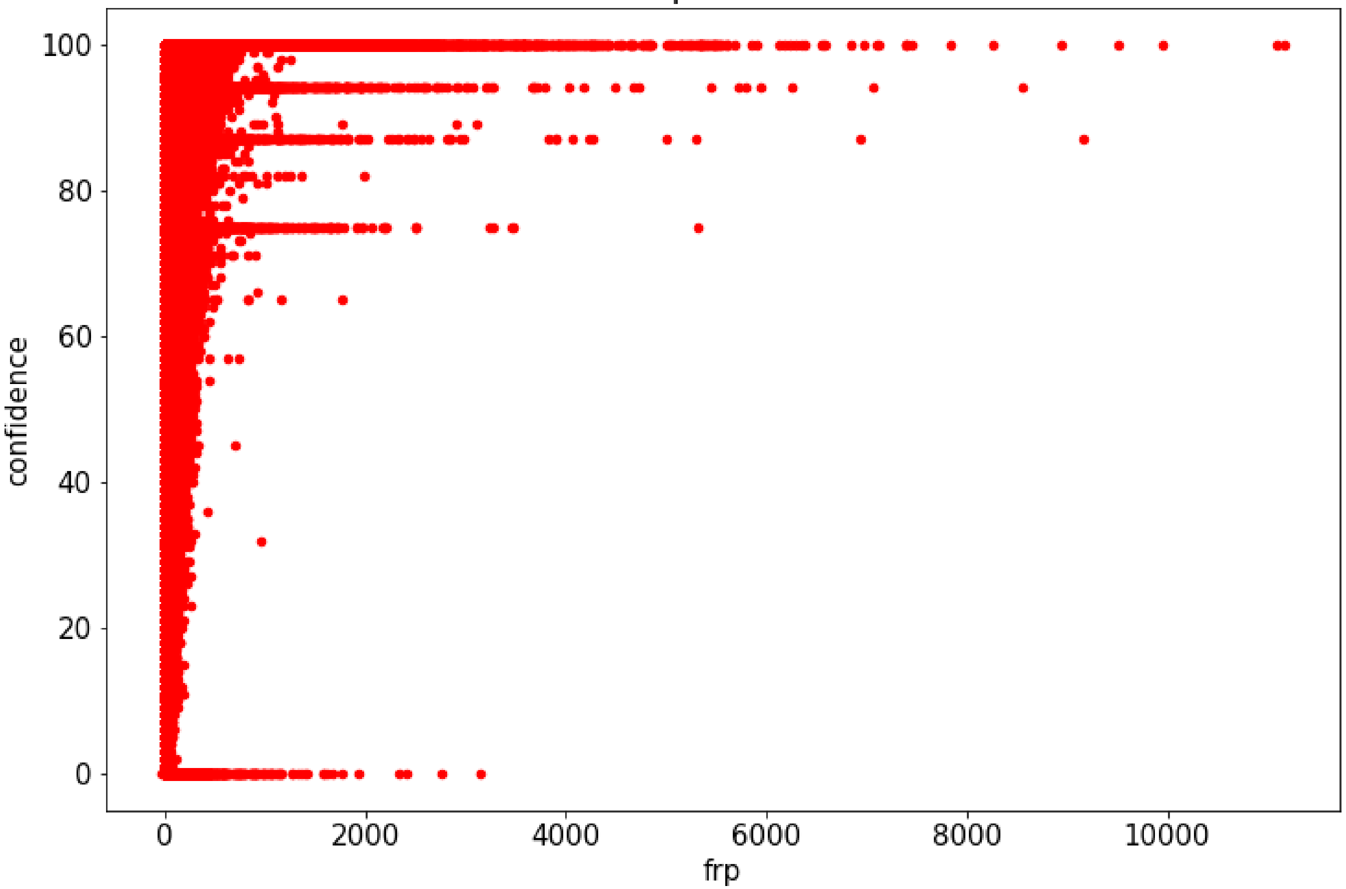


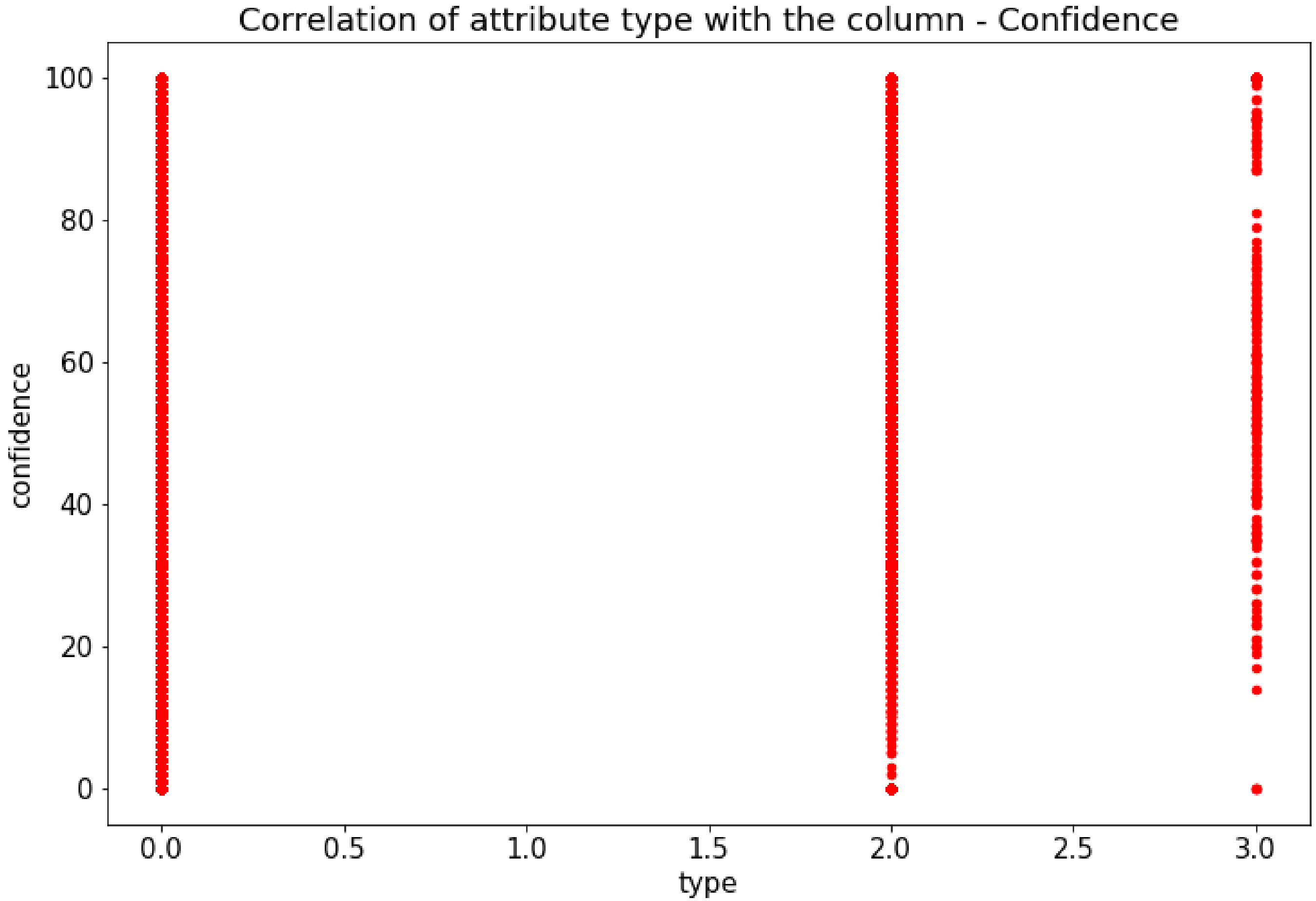
Correlation of attribute bright_t31 with the column - Confidence



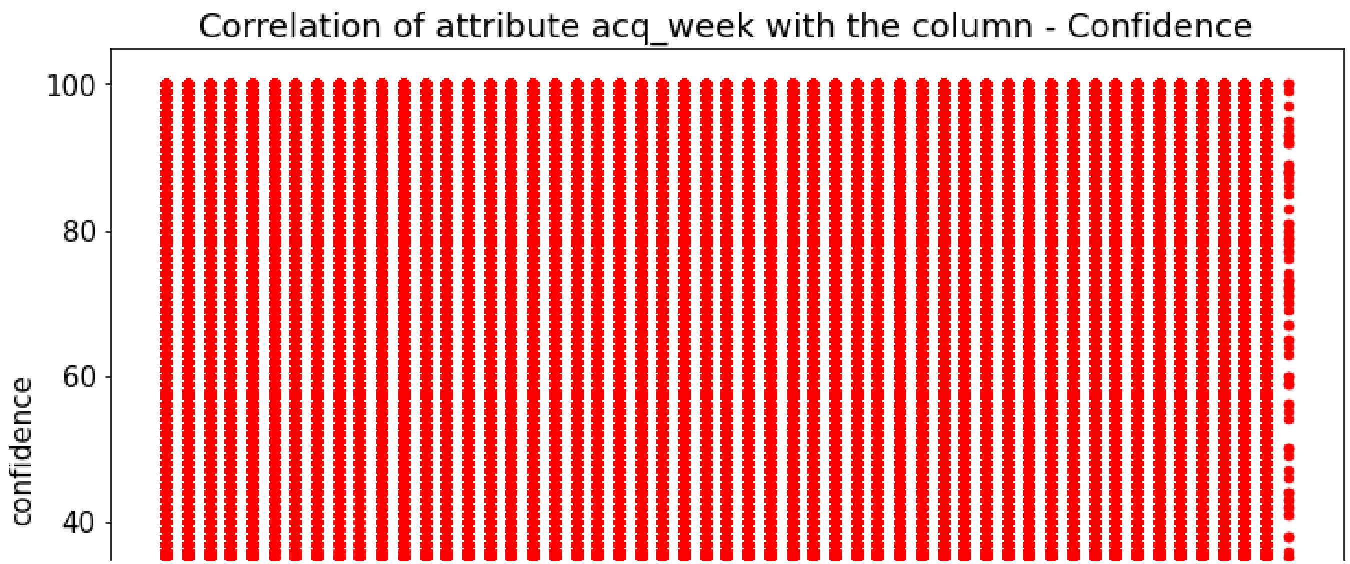


Correlation of attribute frp with the column - Confidence

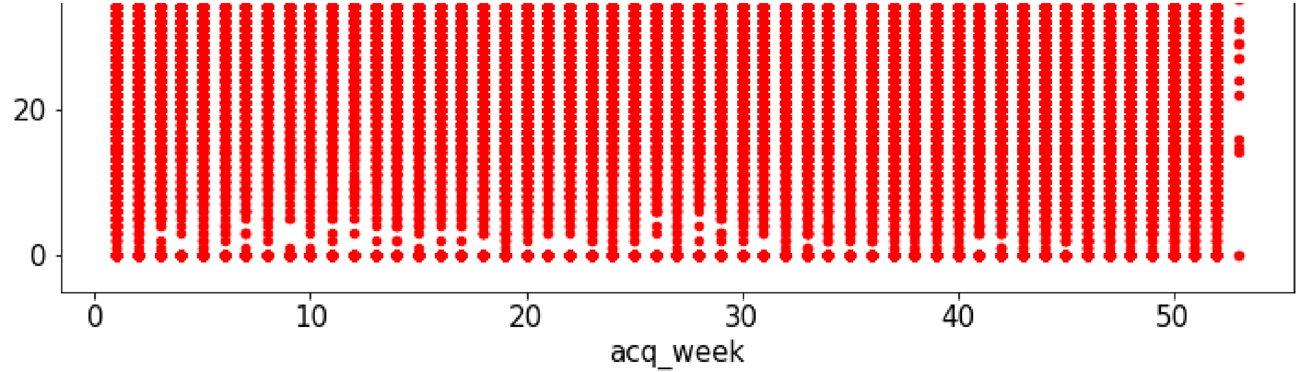




Correlation of attribute type with the column - Confidence
confidence
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
type



Correlation of attribute acq_week with the column - Confidence
confidence
100
80
60
40



Question 2

For visualizing the map (of Australia) we have used folium. Reference - [Folium - Official Documentation](#)

In [41]:

```
import folium
from folium import plugins
```

Question 2.1

Plot a geographical heat map of FRP for the Aqua area. Use data from the year 2017 and any aggregation method (e.g. mean, summation, maximum, or something else) of your choice. Justify your choice of aggregation method.

Description of Solution:

1. **Obtain data from the year 2017.**
2. **Filter data only for rows where the satellite is 'Aqua'**
3. **Aggregate the FRP Values by taking the mean of all dates across all regions. That is aggregation along dates.**
4. **Plot the Heatmap for the Auqa area.**

Initially, we considered using the maximum FRP values instead of the mean value. However, we brainstormed and came to a conclusion that it's better to visualize the typical (average) intensity/power of the fire rathern than the maximum since the maximum can be a rare occurence and will not reflect a normal value for a normal day. Visualizing the mean FRP values will give the viewers an idea of how intense the fire will be on a normal/average day.

In [42]:

```
dataframe_2017 = dataset_merged[(dataset_merged['acq_date'].dt.year==2017) & (dataset_merged['satellite']=='Aqua')]
dataframe_2017['acq_date'].dt.year.unique() #Dataset for the year 2017
```

Out[42]:

```
array([2017])
```

In [43]:

```
dataframe_2017['satellite'].unique()
```

Out[43]:

```
<StringArray>
['Aqua']
Length: 1, dtype: string
```

In [44]:

```
dataframe_2017.head(10)
```

Out[44]:

	latitude	longitude	brightness	acq_date	acq_time	satellite	instrument	confidence	bright_t31	frp	daynight	type	acq_week
35	-31.43	151.87	329.80	2017-01-01	328	Aqua	MODIS	83	290.90	40.20	D	0	52
36	-29.26	153.00	333.50	2017-01-01	329	Aqua	MODIS	63	309.10	21.20	D	0	52
37	-29.26	153.01	330.50	2017-01-01	329	Aqua	MODIS	44	307.50	16.40	D	0	52
38	-27.33	149.99	345.40	2017-01-01	330	Aqua	MODIS	88	309.90	59.90	D	0	52
39	-27.32	150.01	345.70	2017-01-01	330	Aqua	MODIS	88	311.40	59.80	D	0	52
40	-24.67	151.62	322.10	2017-01-01	330	Aqua	MODIS	43	296.90	15.80	D	0	52
41	-27.33	150.01	343.60	2017-01-01	330	Aqua	MODIS	86	311.20	51.20	D	0	52
42	-27.32	149.99	341.00	2017-01-01	330	Aqua	MODIS	83	311.00	42.50	D	0	52
43	-25.73	152.05	330.10	2017-01-01	330	Aqua	MODIS	56	305.90	21.00	D	0	52
44	-26.27	151.08	336.70	2017-01-01	330	Aqua	MODIS	80	309.40	26.60	D	0	52

In [45]:

```
dataframe_2017_frp=dataframe_2017[['latitude','longitude','acq_date','frp']]
dataframe_2017_frp.head(10)
```

Out[45]:

	latitude	longitude	acq_date	frp
35	-31.43	151.87	2017-01-01	40.20
36	-29.26	153.00	2017-01-01	21.20
37	-29.26	153.01	2017-01-01	16.40

	latitude	longitude	acq_date	frp
38	-27.33	149.99	2017-01-01	59.90
39	-27.32	150.01	2017-01-01	59.80
40	-24.67	151.62	2017-01-01	15.80
41	-27.33	150.01	2017-01-01	51.20
42	-27.32	149.99	2017-01-01	42.50
43	-25.73	152.05	2017-01-01	21.00
44	-26.27	151.08	2017-01-01	26.60

In [46]:

```
dataframe_2017_frp = dataframe_2017_frp.groupby(['latitude','longitude'])['frp'].mean().reset_index()
dataframe_2017_frp.head(10)
```

Out[46]:

	latitude	longitude	frp
0	-43.43	146.87	74.00
1	-43.31	146.84	111.20
2	-43.31	146.89	470.00
3	-43.31	146.95	25.30
4	-43.31	146.88	2262.90
5	-43.30	146.97	56.60
6	-43.30	146.83	71.60
7	-43.30	146.89	169.60
8	-43.30	146.82	48.90
9	-43.29	146.95	37.00

In [47]:

```
dataframe_2017_frp.shape
```

Out[47]:

```
(157213, 3)
```

In [48]:

```
from folium.plugins import HeatMap
#Create a Map
```

```
map = folium.Map(location=[-35.6,149.12], control_scale=True, zoom_start=5,prefer_canvas=False)

heat_data = [[row['latitude'],row['longitude'],row['frp']] for index, row in dataframe_2017_frp.iterrows()]

# Plot it on the map
HeatMap(heat_data,radius=2.5,blur=5).add_to(map)

map

#Reference - Folium Official Documentation [ https://python-visualization.github.io/folium/quickstart.html#Markers ]
```

Output hidden; open in https://colab.research.google.com to view.

Q2.2: Mark the “Fire activity” (lat, long = -35.6,149.12) on the city map.

The "Fire Activity" with latitude and longitude (-35.6,149.12) has been marked on the map.

In [49]:

```
#Create a Map
map = folium.Map(location=[0,0], control_scale=True, zoom_start=3,prefer_canvas=False)

#Add a Marker with attributes - location and icon to the map
folium.Marker(
location=[-35.6,149.12],
tooltip= "Fire Activity: Latitude, Longitude = -35.6,149.12 ",
icon=folium.Icon(color='red',icon='info-sign')
).add_to(map)

map
```

Out[49]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Q 2.3 Mark the regions with the highest recorded fire radiation in a day for measurements where "acq_date = 2020-01-08

Solution:

The solution is as follows:

1. Create a new dataframe from the primary dataframe by dropping unnecessary columns. 2. From the dataframe that we have, create a new dataframe by filtering the column 'acq_date' to obtain activities corresponding to the date **2020-01-08**. 3. Sort the frp values in descending order and take the first value. 4. Plot the activity on the map using the latitude and longitude obtained from the previous step.

In [50]:

```
dataframe_highest_frp=dataset_merged[['latitude','longitude','acq_date','frp']]
dataframe_highest_frp.head(10)
```

Out[50]:

	latitude	longitude	acq_date	frp
0	-23.91	147.30	2017-01-01	17.60
1	-23.69	150.10	2017-01-01	30.00
2	-23.59	150.17	2017-01-01	35.40
3	-22.41	148.85	2017-01-01	25.80
4	-20.59	147.64	2017-01-01	19.00
5	-21.13	148.19	2017-01-01	22.90
6	-25.07	149.75	2017-01-01	28.60
7	-25.61	148.93	2017-01-01	25.70
8	-25.61	148.92	2017-01-01	40.40
9	-28.63	149.74	2017-01-01	34.90

In [51]:

```
dataframe_highest_frp = dataframe_highest_frp[dataframe_highest_frp['acq_date']=='2020-01-08']
dataframe_highest_frp.head(10)
```

Out[51]:

	latitude	longitude	acq_date	frp
22772	-12.73	142.47	2020-01-08	12.80
22773	-14.61	143.94	2020-01-08	63.60
22774	-14.62	143.95	2020-01-08	53.50
22775	-14.63	143.96	2020-01-08	45.00
22776	-14.63	143.97	2020-01-08	28.70
22777	-14.69	143.98	2020-01-08	41.90
22778	-14.69	143.89	2020-01-08	80.20
22779	-14.69	143.90	2020-01-08	102.70
22780	-14.71	143.96	2020-01-08	16.00
22781	-14.86	143.10	2020-01-08	12.80

In [52]:

```
result = dataframe_highest_frp.sort_values(by='frp',ascending=False).head(1)
result
```

Out[52]:

	latitude	longitude	acq_date	frp
23118	-32.89	124.10	2020-01-08	11164.10

In [53]:

```
result.iloc[0]['latitude']
```

Out[53]:

```
-32.8923
```

In [54]:

```
map = folium.Map(location=[-26.115986,135.044605], control_scale=True, zoom_start=2) # Start with Map at the center of Australia, which was found using [ https://www.findlatitudeandlongitude.com/l/Centre+Point+australia/1505494/ ]

folium.Marker(
location=[result.iloc[0]['latitude'], result.iloc[0]['longitude']],
tooltip= "date: "+ str(result.iloc[0]['acq_date']) + "<br/> frp: " + str(result.iloc[0]['frp']) + "<br/> Latitude: " + str(result.iloc[0]['latitude']) + "<br/> Longitude: " + str(result.iloc[0]['longitude']),
icon=folium.Icon(color='blue',icon='cloud')
).add_to(map)

map
```

Out[54]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Q 2.4 Find a visualization to plot the progress of the fire activity across all points from Nov 1, 2019, to Jan 31, 2020.

Solution:

The solution is as follows:

1. **Create a new dataframe from the primary dataframe by dropping unnecessary columns.**
2. **From the dataframe that we have, create a new dataframe by filtering the column 'acq_date' to obtain activities corresponding to the date range [2019-11-01 to 2020-01-31].**
3. **Now, we need to add the markers for the summation of FRPs across each day grouped by the coordinates. Therefore we have created a nested List where each inner list corresponds to a list of pair of coordinates and the aggregate FRP (sum).**
4. **Add the list to the map object and render the map.**

Since we have sorted the dates by ascending order the map will visualize the FRP values in chronological order as well!

In [55]:

```
!pip uninstall folium -y
```

```
Found existing installation: folium 0.12.1.post1
Uninstalling folium-0.12.1.post1:
  Successfully uninstalled folium-0.12.1.post1
```

In [56]:

```
!pip install folium
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting folium
  Using cached folium-0.12.1.post1-py2.py3-none-any.whl (95 kB)
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.7/dist-packages (from folium) (2.11.3)
Requirement already satisfied: branca>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from folium) (0.5.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from folium) (2.23.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from folium) (1.21.6)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2>=2.9->folium) (2.0.1)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (2022.6.15)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->f
olium) (1.24.3)
Installing collected packages: folium
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the s
ource of the following dependency conflicts.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.12.1.post1 which is incompatible.
Successfully installed folium-0.12.1.post1
```

HeatMap with Time Function referred from HeatMap with Time

Please note that the HeatMap with time functionality is available only in later versions of folium (0.4 and beyond). Therefore we have to install the latest version of folium and restart the runtime, in order to view the complete functionality (Start & Stop button).

In [57]:

```
from folium.plugins import HeatMapWithTime

dataframe_fire_activity=dataset_merged[['latitude','longitude','acq_date','frp']]

dataframe_fire_activity = dataframe_fire_activity[(dataframe_fire_activity['acq_date']>='2019-11-01') & (dataframe_fire_activity[
'acq_date']<='2020-01-31')]

data = []
data = [dataframe_fire_activity[dataframe_fire_activity.acq_date==date].groupby(['latitude', 'longitude'])['frp'].sum().reset_ind
ex().values.tolist() for date in dataframe_fire_activity.acq_date.sort_values().unique()]
# print(data[0])

map = folium.Map(location=[-26.115986,135.044605], zoom_start=3,tiles="stamentoner")

hm = HeatMapWithTime(data)
hm.add_to(map)
map
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Q3. Build a model for spatial prediction of wildfire

Q3.1 In between ‘Brightness temperature I-5’ and ‘Fire Radiative Power’ choose either as the target feature.

Answer:

For this task of building models for spatial prediction of wildfires, we have chosen 'Brightness Temperature' alias Brightness as the Target Feature

Sampling

Our approach to sampling involves Systematic Sampling. This involves selection of instances at regular intervals from the population/dataset to form a derived dataset of the desired size. We have started at a point in the dataset and selected elements at a regular, fixed interval. In statistical terminology, we essentially selected every k-th element in the dataset.

We chose Systematic Sampling since our merged dataset consists of data from 4 years and we wanted our derived dataset to represent each year in an equivalent manner.

Our merged dataset contains 1046679 instances as shown below. We desire a sample size of around 600,000 instances. Therefore the interval is calculated as follows:

1046679/600000 = 1.744465

In [58]:

```
1046679/600000
```

Out[58]:

```
1.744465
```

In [59]:

```
dataset_merged.shape
```

Out[59]:

```
(1046679, 13)
```

We can observe that our sampled dataset has 601,540 instances

In [60]:

```
import numpy as np
```

```
indexes = np.arange(0, len(dataset_merged), step=1.74)
sampled_df = dataset_merged.iloc[indexes]
sampled_df.shape
```

Out[60]:

```
(601540, 13)
```

Q3.2 Explain what the task you’re solving

Answer:

The task specified in the assignment handout is to builda model for the spatial precition of Wildfire in Australia. Initially, this seems to point towards a Classification task, since prediction of Wildfires involves two discrete class labels - 'Yes'/'No'. However, the traget variable in this task (Brightness) is a continuous quantity. Thus we are predicting the Brightness value in the "unseen" or test data. Therefore, the task we are solving is relevant to Regression. We can then use the predicted value to determine the probability of a wildfire in a specific location and timeframe.

To obtain, as good an accuracy as posible, we have decided to transofrm acuired_dates to week numbers in a year. Typically, the number of weeks in a year range from 52-53. We could have chosen month too, but the granularity of month is much larger when compared to the granularity of Date.

Suppose, we desire to answer the the following question, after building our model:

"What are the chances of a Wildfire occurrence on 2023-December-15 at coordinates [-22.52,147.69] ? "

Now if we had built our model with the attribute - "Month" instead of "Week" we believe that we cannot answer the aforementioned question accurately, since, we cannot predict the occurence of a Wildfire at the "Date / Day" level, rather we can do so only at the "Month" level. Therefore, in order to keep the granularity of the model as close to the dataset as possible, we have transformed dates to the "Week Level".

Q3.3 Use a feature selection method to select the features to build a model.

Answer:

We have performed Feature selection to to reduce the number of input variables to both, reduce the computational cost of modeling and to improve the performance of the model. Feature iselection is important since, a large number of variables can slow the development and training of models and require a large amount of system memory.

The Feature selection process was performed using Pearson’s Correlation Coefficient via the *f_regression() function* in sklearn. Pearson's correlation coefficient is a measure of linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations. Thus it is essentially a normalized measurement of the covariance, such that the result always has a value between −1 and 1. [f_regression - Reference]

We had dropped the columns - "scan", "track" and "version" from the original dataset, since these columns have very few unique values and represent little variation of data in the dataset. Therefore, we determined those columns to be irrelevant to the problem at hand. Please refer to the following cell to view the 3 columns being dropped: [Dropping the columns - scan, track and version]

First let's transform all attributes to numeric representations

First let's transform all attributes to numeric representations.

In [61]:

```
sampled_df.head(10)
```

Out[61]:

	latitude	longitude	brightness	acq_date	acq_time	satellite	instrument	confidence	bright_t31	frp	daynight	type	acq_week
0	-23.91	147.30	320.10	2017-01-01	47	Terra	MODIS	53	296.60	17.60	D	0	52
1	-23.69	150.10	314.30	2017-01-01	47	Terra	MODIS	22	289.30	30.00	D	0	52
3	-22.41	148.85	316.70	2017-01-01	47	Terra	MODIS	26	295.30	25.80	D	0	52
5	-21.13	148.19	320.50	2017-01-01	47	Terra	MODIS	36	299.70	22.90	D	0	52
6	-25.07	149.75	314.60	2017-01-01	48	Terra	MODIS	51	294.50	28.60	D	0	52
8	-25.61	148.92	324.00	2017-01-01	48	Terra	MODIS	65	300.00	40.40	D	0	52
10	-28.63	149.74	329.70	2017-01-01	49	Terra	MODIS	58	304.10	46.20	D	0	52
12	-22.53	118.52	338.20	2017-01-01	226	Terra	MODIS	40	319.70	15.80	D	0	52
13	-22.50	118.53	353.40	2017-01-01	226	Terra	MODIS	95	323.40	44.90	D	0	52
15	-22.51	118.51	377.40	2017-01-01	226	Terra	MODIS	100	322.30	122.70	D	0	52

Satellite

In [62]:

```
# df['brightness_temperature'] = df['brightness_temperature'].map({'low':0, 'High':1, 'Extreme':2})
sampled_df.satellite.unique()
```

Out[62]:

```
<StringArray>
['Terra', 'Aqua']
Length: 2, dtype: string
```

In [63]:

```
sampled_df['satellite'] = sampled_df['satellite'].map({'Aqua':0, 'Terra':1})
sampled_df.satellite.unique()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
```

```
s-a-copy
  """Entry point for launching an IPython kernel.
```

Out[63]:

```
array([1, 0])
```

Instrument

In [64]:

```
sampled_df.instrument.unique()
```

Out[64]:

```
<StringArray>
['MODIS']
Length: 1, dtype: string
```

The column "Instrument" has only one value. Therefore it influences no variance in the dataset. Hence we will drop it!

In [65]:

```
sampled_df.drop(['instrument'], axis=1,inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4913: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  errors=errors,
```

In [66]:

```
sampled_df.head(10)
```

Out[66]:

	latitude	longitude	brightness	acq_date	acq_time	satellite	confidence	bright_t31	frp	daynight	type	acq_week
0	-23.91	147.30	320.10	2017-01-01	47	1	53	296.60	17.60	D	0	52
1	-23.69	150.10	314.30	2017-01-01	47	1	22	289.30	30.00	D	0	52
3	-22.41	148.85	316.70	2017-01-01	47	1	26	295.30	25.80	D	0	52
5	-21.13	148.19	320.50	2017-01-01	47	1	36	299.70	22.90	D	0	52
6	-25.07	149.75	314.60	2017-01-01	48	1	51	294.50	28.60	D	0	52
8	-25.61	148.92	324.00	2017-01-01	48	1	65	300.00	40.40	D	0	52

	latitude	longitude	brightness	acq_date	acq_time	satellite	confidence	bright_t31	frp	daynight	type	acq_week
10	-28.63	149.74	329.70	2017-01-01	49	1	58	304.10	46.20	D	0	52
12	-22.53	118.52	338.20	2017-01-01	226	1	40	319.70	15.80	D	0	52
13	-22.50	118.53	353.40	2017-01-01	226	1	95	323.40	44.90	D	0	52
15	-22.51	118.51	377.40	2017-01-01	226	1	100	322.30	122.70	D	0	52

Daynight

In [67]:

```
sampled_df.daynight.unique()
```

Out[67]:

```
<StringArray>
['D', 'N']
Length: 2, dtype: string
```

In [68]:

```
sampled_df['daynight'] = sampled_df['daynight'].map({'D':0, 'N':1})
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  """Entry point for launching an IPython kernel.
```

In [69]:

```
sampled_df.head(10)
```

Out[69]:

	latitude	longitude	brightness	acq_date	acq_time	satellite	confidence	bright_t31	frp	daynight	type	acq_week
0	-23.91	147.30	320.10	2017-01-01	47	1	53	296.60	17.60	0	0	52
1	-23.69	150.10	314.30	2017-01-01	47	1	22	289.30	30.00	0	0	52
3	-22.41	148.85	316.70	2017-01-01	47	1	26	295.30	25.80	0	0	52
5	-21.13	148.19	320.50	2017-01-01	47	1	36	299.70	22.90	0	0	52
6	-25.07	149.75	314.60	2017-01-01	48	1	51	294.50	28.60	0	0	52
8	-25.61	148.92	324.00	2017-01-01	48	1	65	300.00	40.40	0	0	52

10	latitude	longitude	brightness	acq_date	acq_time	satellite	confidence	bright_t31	frp	daynight	type	acq_week
12	-22.53	118.52	338.20	2017-01-01	226	1	40	319.70	15.80	0	0	52
13	-22.50	118.53	353.40	2017-01-01	226	1	95	323.40	44.90	0	0	52
15	-22.51	118.51	377.40	2017-01-01	226	1	100	322.30	122.70	0	0	52

Normalizing Columns

Min-Max Normalization

In [70]:

```
y=sampled_df.brightness.astype(float)
X=sampled_df.drop(['brightness','acq_date'],axis=1).astype(float)
print(type(X))
```

```
<class 'pandas.core.frame.DataFrame'>
```

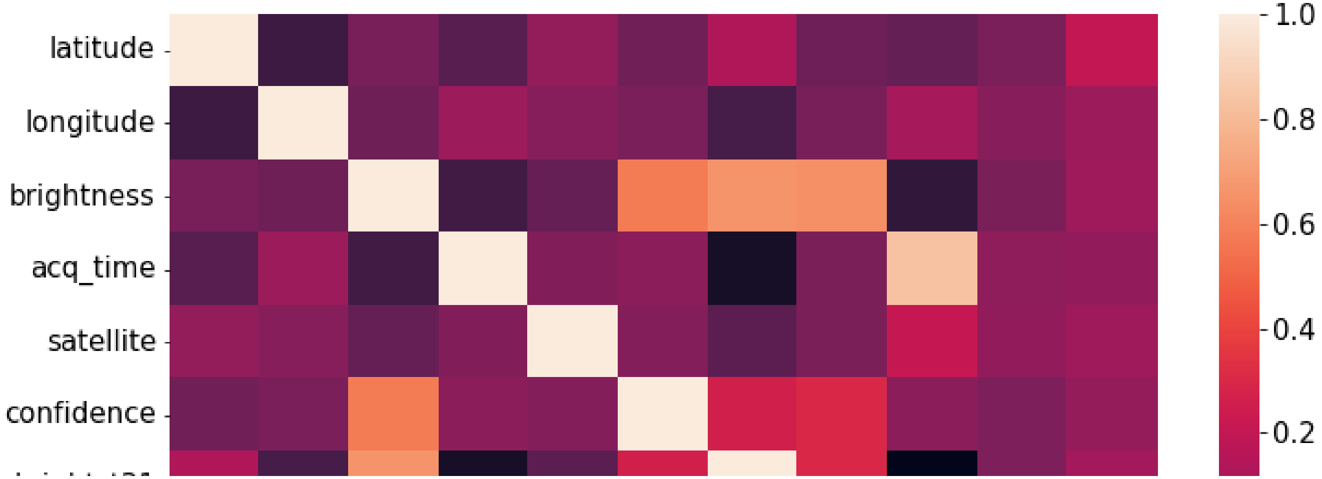
In [71]:

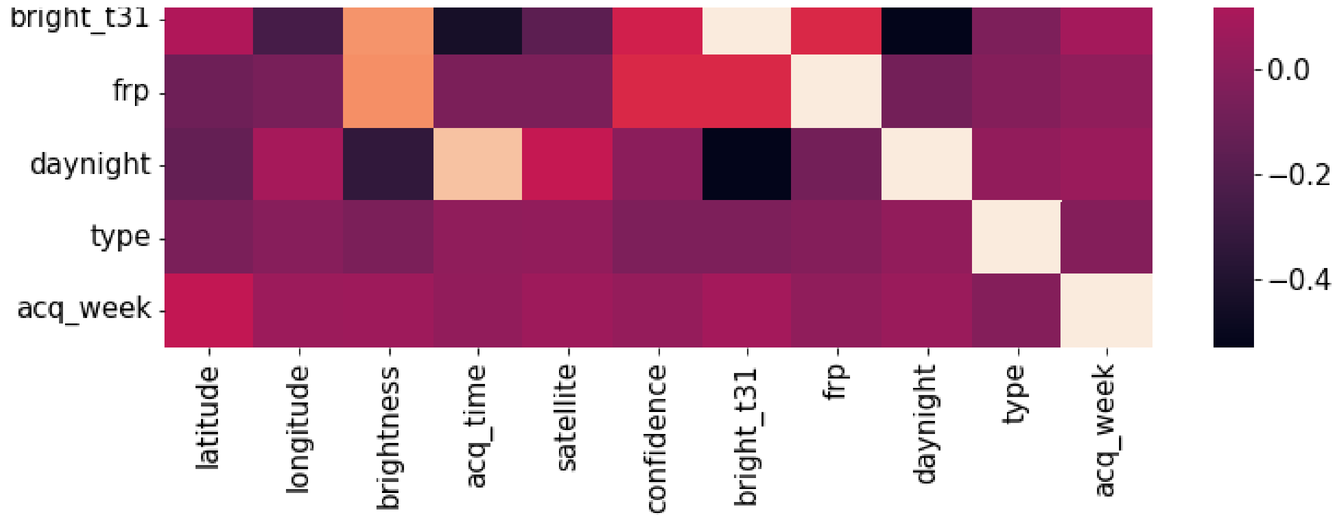
```
#Perform Min-Max Normalization to Normalize Data
X = (X-X.min()) / (X.max()-X.min())
print(type(X))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [72]:

```
dataplot_sampled=sb.heatmap(sampled_df.corr())
plt.show()
```





In [73]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression

'''
X = Input Feature Dataframe
y = Target Feature Dataframe
'''
# define feature selection

# apply feature selection
# fs = SelectKBest(f_regression, k=6)
# k_best = fs.fit_transform(X,y)

selector = SelectKBest(f_regression, k=9)
selector.fit(X, y)
# Get columns to keep and create new dataframe with the K best columns
cols = selector.get_support(indices=True)
X_new = X.iloc[:,cols]

print(X_new.shape)
print(X_new.columns)
print(y.shape)
```

```
(601540, 9)
Index(['latitude', 'longitude', 'acq_time', 'satellite', 'confidence',
       'bright_t31', 'frp', 'daynight', 'acq_week'],
      dtype='object')
(601540,)
```

Q3.4 Select one or more evaluation metrics. Justify your choice.

Answer: We chose 3 metrics for evaluating our models [Evaluation Metrics]:

1. **R Square**
2. **Mean Square Error(MSE)**
3. **Mean Absolute Error(MAE)**

R Square measures how much variability in dependent variable can be explained by the model. R Square value is between 0 to 1 and a bigger value indicates a better fit between prediction and actual value. R Square is a good measure to determine how well the model fits the dependent variables. However, it does not take into consideration of overfitting problem.

MSE is calculated by the sum of square of prediction error which is real output minus predicted output and then divide by the number of data points. It gives an absolute number on how much the predicted results deviate from the actual number.

Mean Absolute Error(MAE) is similar to Mean Square Error(MSE).However, instead of the sum of square of error in MSE, MAE is taking the sum of the absolute value of error.

Compare to MSE or RMSE, MAE is a more direct representation of sum of error terms. MSE gives larger penalization to big prediction error by square it while MAE treats all errors the same.

Q3.5 Build a baseline model (Linear Regression)

In [74]:

```
from sklearn.model_selection import train_test_split
input_features = X
target_features = y
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.8)
x_train.shape,x_test.shape,y_train.shape, y_test.shape
```

Out[74]:

```
((120308, 10), (481232, 10), (120308,), (481232,))
```

In [75]:

```
from sklearn.linear_model import LinearRegression
# creating an object of LinearRegression class
LR = LinearRegression()
# fitting the training data
history = LR.fit(x_train,y_train)
```

In [76]:

```
y_prediction =  LR.predict(x_test)
# y_prediction
```

In [77]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
r2 = r2_score(y_test, y_prediction)
mse = mean_squared_error(y_test,y_prediction,squared=False)
mae = mean_absolute_error(y_test, y_prediction)
print('r2 Score: ',r2)
print('mean_sqrd_error: ',mse)
print('mean absolute error: ',mae)
```

```
r2 Score:  0.7624725231415908
mean_sqrd_error:  11.26612386407195
mean absolute error:  7.743893524158036
```

Learning Curve

In [78]:

```
from sklearn.model_selection import learning_curve
train_sizes = [1, 100, 500, 2000, 5000, 20000, 50000, 100000, 150000, 200000, 250000, 300000,]

train_sizes, train_scores, validation_scores = learning_curve(
estimator = LinearRegression(),
X = input_features,
y = target_features, train_sizes = train_sizes, cv = 2,
scoring = 'neg_mean_squared_error')
```

In [79]:

```
train_scores_mean = -train_scores.mean(axis = 1)
validation_scores_mean = -validation_scores.mean(axis = 1) #Changed the sign of the mean validation scores
print('Mean training scores\n\n', pd.Series(train_scores_mean, index = train_sizes))
print('\n', '-' * 20) # separator
print('\nMean validation scores\n\n',pd.Series(validation_scores_mean, index = train_sizes))
```

```
Mean training scores

 1         -0.00
100        36.06
500        54.01
2000       85.31
5000       89.42
20000      85.82
50000      98.67
100000    103.35
150000    111.93
```

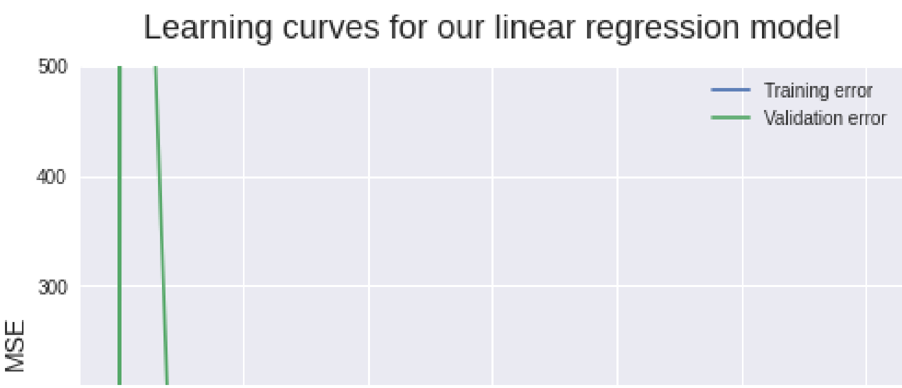
```
150000   111.95
200000   114.21
250000   128.73
300000   123.65
dtype: float64

 --------------------

Mean validation scores

 1                          692.71
100       1956804901826556160.00
500                         174.02
2000                       2722.28
5000                       1124.19
20000                       183.12
50000                       184.53
100000                      138.18
150000                      137.43
200000                      136.91
250000                      140.18
300000                      139.56
dtype: float64
```

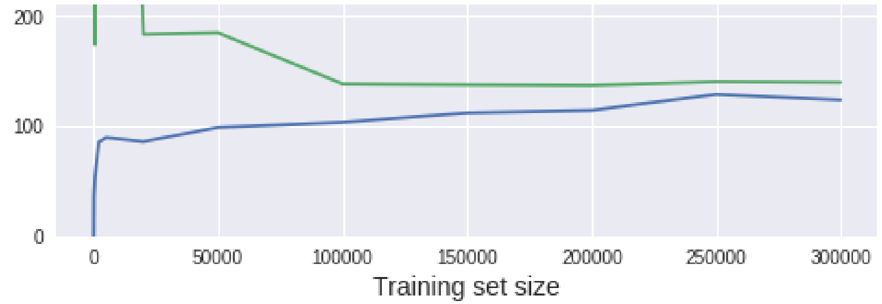
In [80]:

```
plt.style.use('seaborn')
plt.plot(train_sizes, train_scores_mean, label = 'Training error')
plt.plot(train_sizes, validation_scores_mean, label = 'Validation error')
plt.ylabel('MSE', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title('Learning curves for our linear regression model', fontsize = 18, y = 1.03)
plt.legend()
plt.ylim(0,500)
```

Out[80]:

```
(0.0, 500.0)
```





We think that this is the appropriate visualization for visualizing the learning process of our model, since,we can determine if the machine model benefits from adding more training data and whether the estimator suffers more from a variance error or a bias error.

How do you make sure not to overfit?

We have attemmpted to minimize overfitting as much as possible by removing irrelevant attributes using the f_regression score function in the "SelectKBest" class of Scikit learn.

In addition, we have held back data by splitting it into equivalent training and testing sets.

Q3.6 Build a candidate final model - K-Nearest Neighbors Algorithm(KNN)

[Reference - KNN Regression]

In [81]:

```
from sklearn.model_selection import train_test_split
input_features = X
target_features = y
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.6)
x_train.shape,x_test.shape,y_train.shape, y_test.shape
```

Out[81]:

```
((240616, 10), (360924, 10), (240616,), (360924,))
```

In [82]:

```
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.8)


from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled = scaler.fit_transform(x_train)
```

```
x_train = pd.DataFrame(x_train_scaled)

x_test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test_scaled)
```

In [83]:

```
from sklearn import neighbors
rmse_val = []
r2_val = []
mae_val = []
for K in range(20):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)

    model.fit(x_train, y_train)
    pred=model.predict(x_test)
    # error = sqrt(mean_squared_error(y_test,pred))
    # rmse_val.append(error)
    rmse_error = mean_squared_error(y_test,pred,squared=False)
    rmse_val.append(rmse_error)

    r2_error = r2_score(y_test,pred)
    r2_val.append(r2_error)

    mae_error = mean_absolute_error(y_test,pred)
    mae_val.append(mae_error)
    print('RMSE value for k= ' , K , 'is:', rmse_error)
    print('R2 value for k= ' , K , 'is:', r2_error)
    print('MAE value for k= ' , K , 'is:', mae_error)
```

```
RMSE value for k=  1 is: 9.455116578828831
R2 value for k=  1 is: 0.8329762031940333
MAE value for k=  1 is: 5.499101057286297
RMSE value for k=  2 is: 8.745050548084198
R2 value for k=  2 is: 0.8571207294643514
MAE value for k=  2 is: 5.014954845064335
RMSE value for k=  3 is: 8.541494251762618
R2 value for k=  3 is: 0.8636948446995669
MAE value for k=  3 is: 4.853271602885927
RMSE value for k=  4 is: 8.484457013452992
R2 value for k=  4 is: 0.8655091670522972
MAE value for k=  4 is: 4.784160186770622
RMSE value for k=  5 is: 8.471840959620538
R2 value for k=  5 is: 0.8659088348425319
MAE value for k=  5 is: 4.751165300728132
RMSE value for k=  6 is: 8.484606217522145
R2 value for k=  6 is: 0.8655044368130023
MAE value for k=  6 is: 4.736679120812139
RMSE value for k=  7 is: 8.509787418830356
```

```
R2 value for k=  7 is: 0.8647049216687962
MAE value for k=  7 is: 4.733412425963837
RMSE value for k=  8 is: 8.543288973044792
R2 value for k=  8 is: 0.8636375583615027
MAE value for k=  8 is: 4.735860717907372
RMSE value for k=  9 is: 8.580466792790446
R2 value for k=  9 is: 0.862448159500227
MAE value for k=  9 is: 4.742259173676452
RMSE value for k=  10 is: 8.61994537093577
R2 value for k=  10 is: 0.8611795004188547
MAE value for k=  10 is: 4.74991050054859
RMSE value for k=  11 is: 8.65850260117436
R2 value for k=  11 is: 0.8599348278038492
MAE value for k=  11 is: 4.758203282030668
RMSE value for k=  12 is: 8.689091059512386
R2 value for k=  12 is: 0.8589434449441031
MAE value for k=  12 is: 4.766460009586507
RMSE value for k=  13 is: 8.727945356845002
R2 value for k=  13 is: 0.8576791223612915
MAE value for k=  13 is: 4.77766330840586
RMSE value for k=  14 is: 8.760365985172614
R2 value for k=  14 is: 0.8566198347824038
MAE value for k=  14 is: 4.7886415640332665
RMSE value for k=  15 is: 8.790285546277428
R2 value for k=  15 is: 0.8556387804998975
MAE value for k=  15 is: 4.79819921922621
RMSE value for k=  16 is: 8.81987520278236
R2 value for k=  16 is: 0.8546652539053474
MAE value for k=  16 is: 4.808215631961301
RMSE value for k=  17 is: 8.853300246628377
R2 value for k=  17 is: 0.8535616044569612
MAE value for k=  17 is: 4.81984497143606
RMSE value for k=  18 is: 8.886181018799174
R2 value for k=  18 is: 0.8524718528952129
MAE value for k=  18 is: 4.831966459697886
RMSE value for k=  19 is: 8.91791982803178
R2 value for k=  19 is: 0.8514161170907146
MAE value for k=  19 is: 4.8436152143708355
RMSE value for k=  20 is: 8.949860855836967
R2 value for k=  20 is: 0.8503498547368515
MAE value for k=  20 is: 4.855534066728727
```

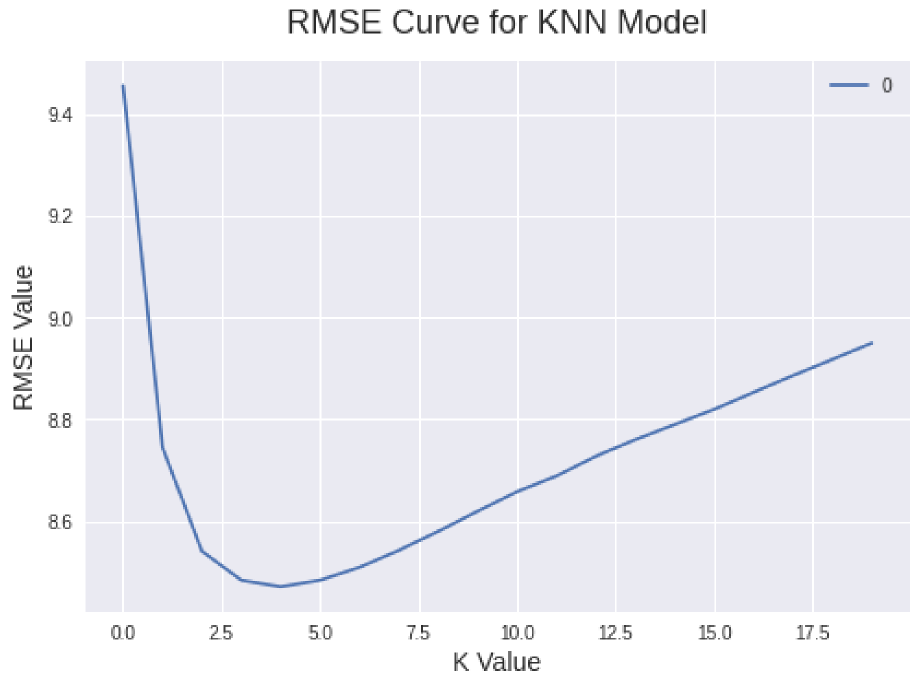
In [84]:

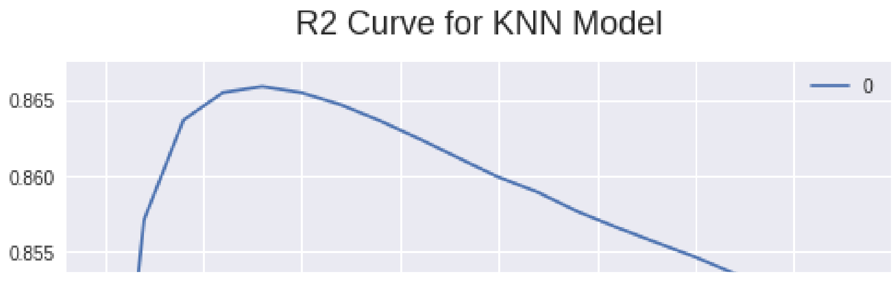
```
from matplotlib.pyplot import figure
rmse_curve = pd.DataFrame(rmse_val)
rmse_curve.plot()
plt.ylabel('RMSE Value', fontsize = 14)
plt.xlabel('K Value', fontsize = 14)
plt.title('RMSE Curve for KNN Model', fontsize = 18, y = 1.03)
```

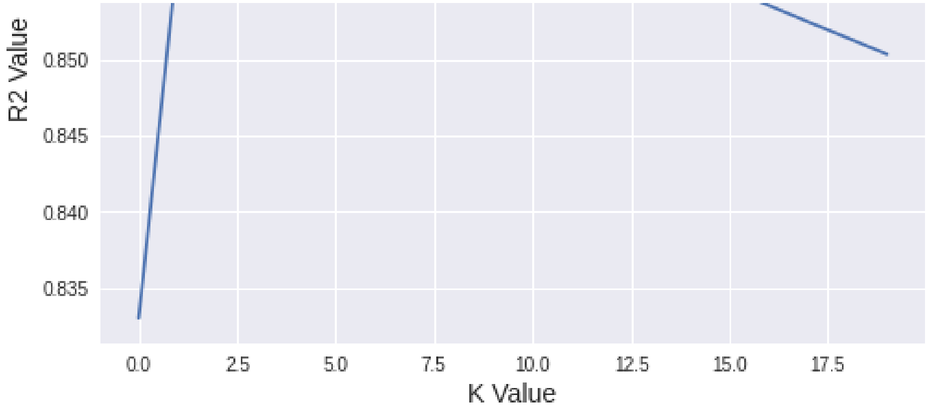
```
from matplotlib.pyplot import figure
r2_curve = pd.DataFrame(r2_val)
r2_curve.plot()
plt.ylabel('R2 Value', fontsize = 14)
plt.xlabel('K Value', fontsize = 14)
plt.title('R2 Curve for KNN Model', fontsize = 18, y = 1.03)

from matplotlib.pyplot import figure
mae_curve = pd.DataFrame(mae_val)
mae_curve.plot()
plt.ylabel('MAE Value', fontsize = 14)
plt.xlabel('K Value', fontsize = 14)
plt.title('MAE Curve for KNN Model', fontsize = 18, y = 1.03)
```

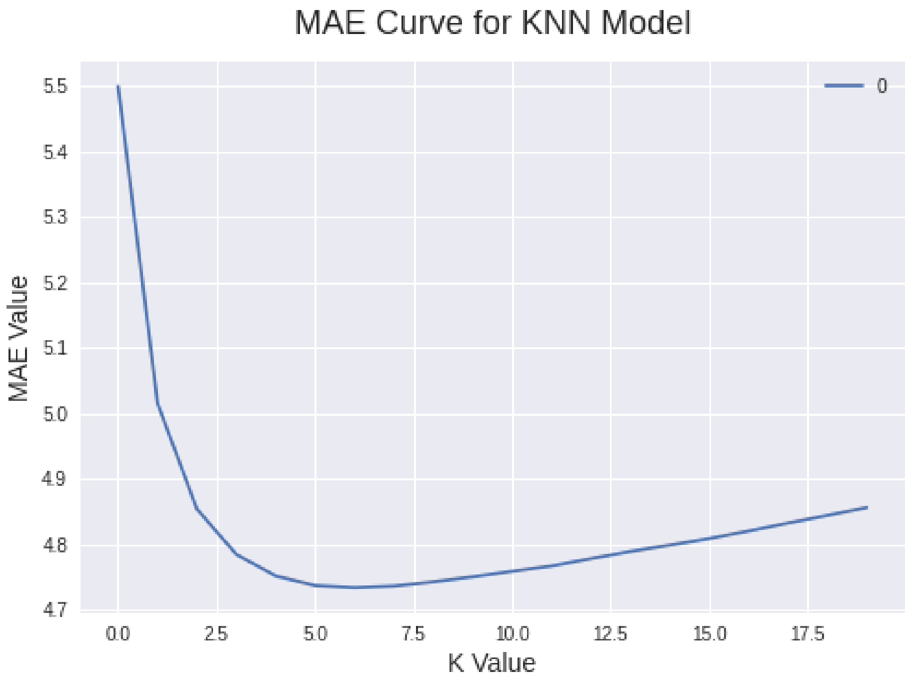
Out[84]:

```
Text(0.5, 1.03, 'MAE Curve for KNN Model')
```









Hyperparameter Tuning

To find the best value of K, we can utilize the GridSearch algorithm

[**Reference - GridSearch with KNN**]

In [85]:

```
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
```

```
knn = neighbors.KNeighborsRegressor()

model = GridSearchCV(knn, params, cv=5)
model.fit(x_train,y_train)
model.best_params_
```

Out[85]:

```
{'n_neighbors': 5}
```

Q3.7 Compare the two models with a statistical significance test. Use a box plot to visualize your comparison

[Reference - Compare Machine Learning Algorithms Consistently](#)

In [ ]:

```
# Compare Algorithms
import pandas
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn import preprocessing
from sklearn import utils
# load dataset
lab = preprocessing.LabelEncoder()
target = lab.fit_transform(target_features)

input = input_features

seed = 7
models = []
models.append(('LR', LogisticRegression()))
models.append(('KNN', KNeighborsRegressor()))
results = []
names = []
scoring = 'accuracy'
for name, model in models:
 kfold = model_selection.KFold(n_splits=5, random_state=None)
 cv_results = model_selection.cross_val_score(model, input, target, cv=kfold, scoring=scoring)
 results.append(cv_results)
 names.append(name)
 msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
 print(msg)
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
```

```
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

Experimental Model - XGboost

In [ ]:

```
import xgboost as xg
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE

train_X, test_X, train_y, test_y = train_test_split(X,y,
     test_size = 0.3, random_state = 123)

train_dmatrix = xg.DMatrix(data = train_X, label = train_y)
test_dmatrix = xg.DMatrix(data = test_X, label = test_y)

param = {"booster":"gblinear", "objective":"reg:linear"}

xgb_r = xg.train(params = param, dtrain = train_dmatrix, num_boost_round = 10)
pred = xgb_r.predict(test_dmatrix)

r2 = r2_score(test_y, pred)
mse = mean_squared_error(test_y, pred,squared=False)
mae = mean_absolute_error(test_y, pred)
print('r2 Score: ',r2)
print('root mean_sqrd_error: ',mse)
print('mean absolute error: ',mae)
```

References:

1. https://dal.brightspace.com/d2l/le/content/221741/viewContent/3023670/View
2. https://dal.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=fe8e7287-82c2-42bc-85ac-ae940127b726 https://dal.brightspace.com/d2l/le/content/221741/viewContent/3023670/View
3. https://dal.brightspace.com/d2l/le/content/221741/viewContent/3023670/View
4. https://www.texasgateway.org/resource/interpreting-scatterplots
5. https://python-visualization.github.io/folium/
6. https://python-visualization.github.io/folium/quickstart.html#Markers
7. https://www.findlatitudeandlongitude.com/l/Centre+Point+australia/1505494/
8. https://python-visualization.github.io/folium/plugins.html?highlight=time#folium.plugins.HeatMapWithTime
9. https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
10. https://colab.research.google.com/drive/1LI-rVN6hhqTh8EgeNgYwJWnwuX93xt7T#scrollTo=q3K79i1Romwd&line=2&uniqifier=1
11. https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/

11. https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/
12. https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/
13. https://machinelearningknowledge.ai/knn-classifier-in-sklearn-using-gridsearchcv-with-example/
14. https://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn/