CSCI - 6409 - The Process of Data Science - Summer 2022

Assignment 1

Harshit Lakhani

B00887087

Aditya Mahale

B00867619

[1]:
```
!pip install tqdm
```

```
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages (4.64.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv class="ansi-yellow-fg">
```

[2]:
```
# Library imports
import pandas as pd
import warnings
import numpy as np

# Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt
from xgboost import plot_tree

# Geographical visualization library
import folium
from folium.plugins import HeatMap
from folium.plugins import HeatMapWithTime
from folium import plugins

# Modelling and preprocessing library
from sklearn import preprocessing
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xg

from sklearn import set_config
from sklearn.model_selection import train_test_split

# Evaluation metrics
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import r2_score as R2

from tqdm import tqdm
import datetime

import scipy
```

[3]:
```
# Prerequisites
# Ignoring Warnings
warnings.filterwarnings('ignore')
set_config(print_changed_only=False)
# Set figure size to 20 X 10
plt.figure(figsize=(20, 10))

# Set SNS plots figure size to 15 X 15
sns.set(rc={'figure.figsize': (15, 15)})

# Set options to avoid truncation when displaying a dataframe
pd.set_option("display.max_rows", None)
pd.set_option("display.max_columns", None)

# Set floating point numbers to be displayed with 2 decimal places
pd.set_option('display.float_format', '{:.5f}'.format)

# Setting plot font parameters
font_label = {'family': 'serif','color':  'darkred','weight': 'normal','size': 16,}
font_title = {'family': 'serif','color':  'darkred','weight': 'bold','size': 22,}
```

```
<Figure size 1440x720 with 0 Axes>
```

1. Data Exploration and preprocessing

There are four different files in this dataset divided by years. In the upcoming code blocks, we will combine the files in a data frame for data exploration and

There are four different files in this dataset divided by years. In the upcoming code blocks, we will combine the files in a data frame for data exploration and preprocessing.

[4]:
```
# Loading the file datasets into four dataframes
df_2017 = pd.read_csv("../input/australia-and-investigative-special-wildfires-data/INVESTIGATION SPECIAL COORDINATES/AUSTRALIA/modis_2017_Australia.csv")
df_2018 = pd.read_csv("../input/australia-and-investigative-special-wildfires-data/INVESTIGATION SPECIAL COORDINATES/AUSTRALIA/modis_2018_Australia.csv")
df_2019 = pd.read_csv("../input/australia-and-investigative-special-wildfires-data/INVESTIGATION SPECIAL COORDINATES/AUSTRALIA/modis_2019_Australia.csv")
df_2020 = pd.read_csv("../input/australia-and-investigative-special-wildfires-data/INVESTIGATION SPECIAL COORDINATES/AUSTRALIA/modis_2020_Australia.csv")

# Combining all the dataframes in a single dataframe
df = pd.concat([df_2017, df_2018, df_2019, df_2020])

# Resetting the index after combining the data frames
df.reset_index(drop=True, inplace=True)
```

1.1 Data Quality Report

1.1.a. Generate data quality reports for the continuous and the categorical features of the data set

[5]:
```
# Summarizing the data types and the number of records available in each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1046679 entries, 0 to 1046678
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: 119.8+ MB
```

- There are 15 columns and none of them have any missing values.
- Some of the features are of type object. It cannot be used in preprocessing data and model training. So, we will convert those types to the relevant data types using the in built convert_dtypes method.

[6]:
```
# Displaying summary after converting data types
df = df.convert_dtypes()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1046679 entries, 0 to 1046678
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  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: 130.8 MB
```

[7]:
```
# Displaying the first row of the data to better understand the data instance
df.loc[0]
```

[7]:
```
latitude       -23.90830
longitude      147.29660
brightness     320.10000
scan             1.70000
track            1.30000
acq_date      2017-01-01
acq_time              47
satellite          Terra
instrument         MODIS
confidence            53
version          6.20000
bright_t31     296.60000
frp             17.60000
daynight               D
type                   0
Name: 0, dtype: object
```

[8]:
```
# Custom function to create the continuous feature report
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
```

[9]:
```
# Preliminary statistics for the continuous features
build_continuous_features_report(df)
```

[9]:

	Count	Miss %	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt	Max	Std. Dev.
latitude	1046679	0.00000	259382	-43.50060	-28.77290	-21.95891	-19.92190	-15.30320	-9.24630	7.78563
longitude	1046679	0.00000	325532	113.12940	126.80485	135.25296	133.14480	144.96105	153.59190	10.51687
brightness	1046679	0.00000	2050	300.00000	317.90000	332.88192	328.70000	341.60000	507.00000	23.10974
scan	1046679	0.00000	39	1.00000	1.10000	1.66498	1.30000	1.90000	4.80000	0.85488
track	1046679	0.00000	11	1.00000	1.00000	1.22804	1.10000	1.40000	2.00000	0.25816
acq_time	1046679	0.00000	855	0.00000	225.00000	622.02228	444.00000	629.00000	2359.00000	532.03785
confidence	1046679	0.00000	101	0.00000	56.00000	71.06028	74.00000	90.00000	100.00000	22.92438
version	1046679	0.00000	2	6.03000	6.03000	6.07439	6.03000	6.20000	6.20000	0.07467
bright_t31	1046679	0.00000	972	265.70000	295.90000	303.23415	303.00000	309.80000	400.10000	10.72635
frp	1046679	0.00000	13993	-29.90000	14.60000	70.50157	28.60000	63.50000	11164.10000	169.46749
type	1046679	0.00000	3	0.00000	0.00000	0.01463	0.00000	0.00000	3.00000	0.17369

Initial Observations

- We can see that in the numerical data columns, there are no missing values
- We can say that the columns type, version, track are categorical features as because the cardinality is less than 10

[10]:
```
# Custom function to create the categorical feature report
def build_categorical_features_report(data_df):

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

    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
```

[11]:
```
# Preliminary statistics for the categorical features
build_categorical_features_report(df)
```

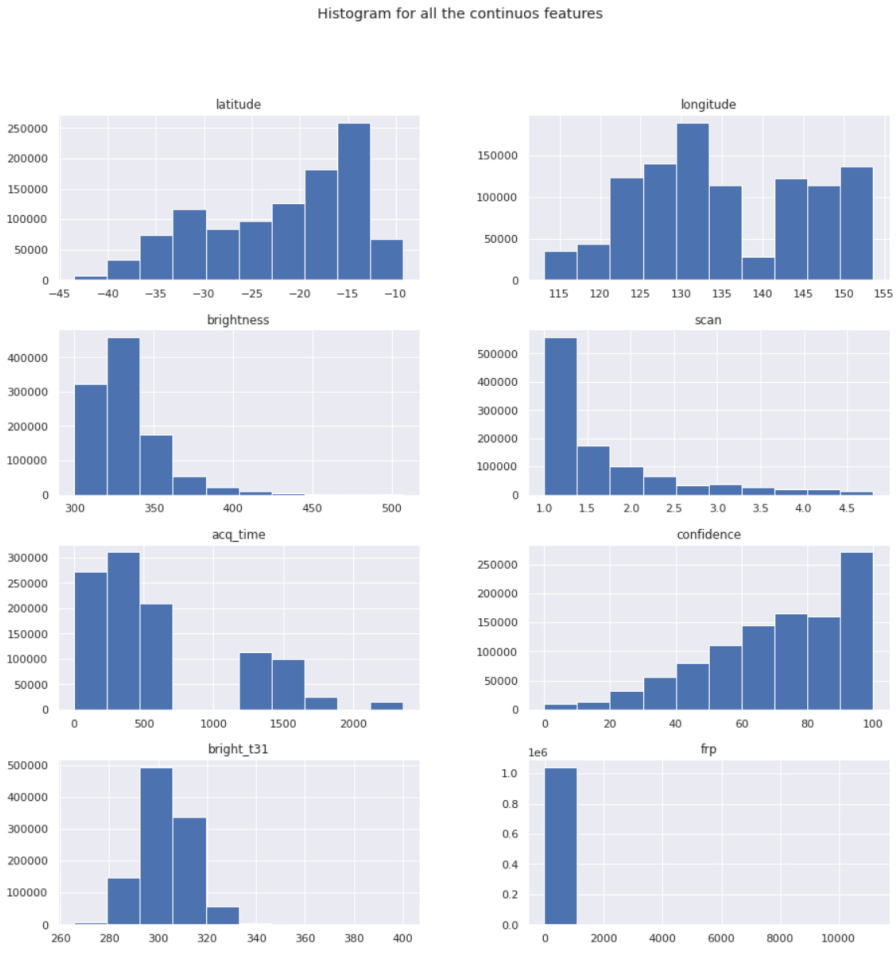
[11]:

	Count	Miss %	Card.	Mode	Mode Freq	Mode %	2nd Mode	2nd Mode Freq	2nd Mode %
acq_date	1046679	0.00000	1461	2020-01-04	7351	0.70232	[2019-12-30]	6925	0.66162
satellite	1046679	0.00000	2	Aqua	600236	57.34671	[Terra]	446443	42.65329
instrument	1046679	0.00000	1	MODIS	1046679	100.00000	[]	0	0.00000
daynight	1046679	0.00000	2	D	806375	77.04129	[N]	240304	22.95871

Initial Observations

- Instrument is a redundant feature as the cardinality is 1

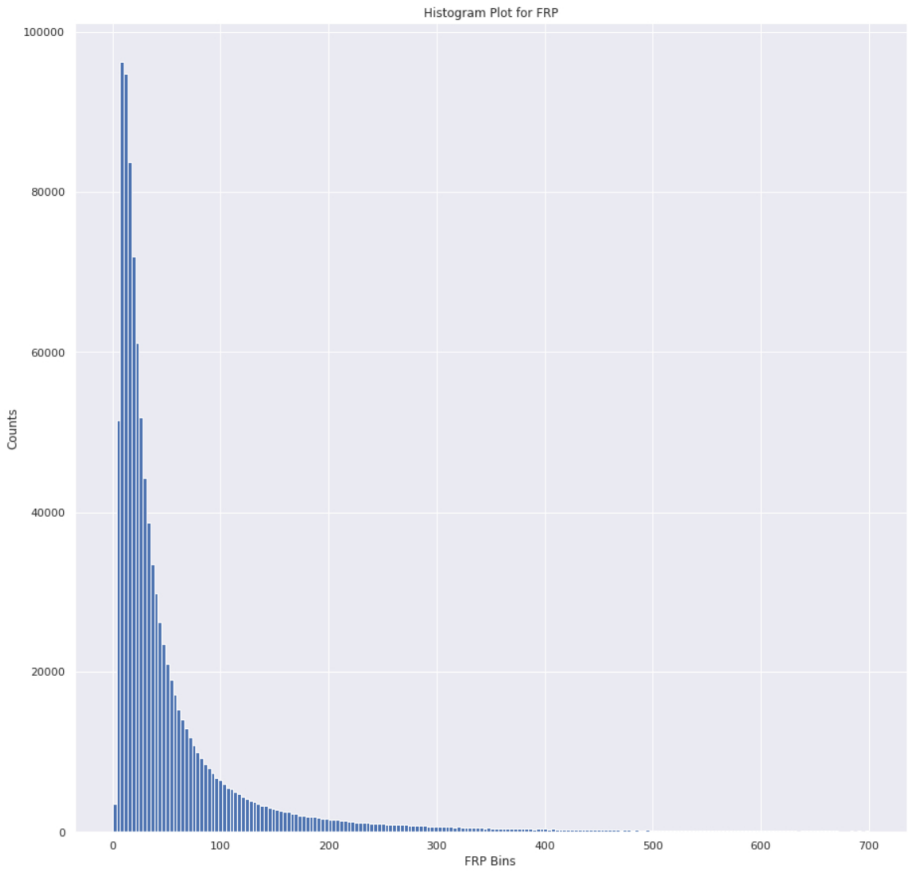
[12]:
```
# Visualization

# Histograms of continuous features
# Choosing only numerical columns and columns where cardinality is greater than or equal to ten

df.hist(column=['latitude', 'longitude', 'brightness', 'scan', 'acq_time',  'confidence', 'bright_t31', 'frp'], layout=(4, 2));
plt.suptitle("Histogram for all the continuos features")
plt.show()
```



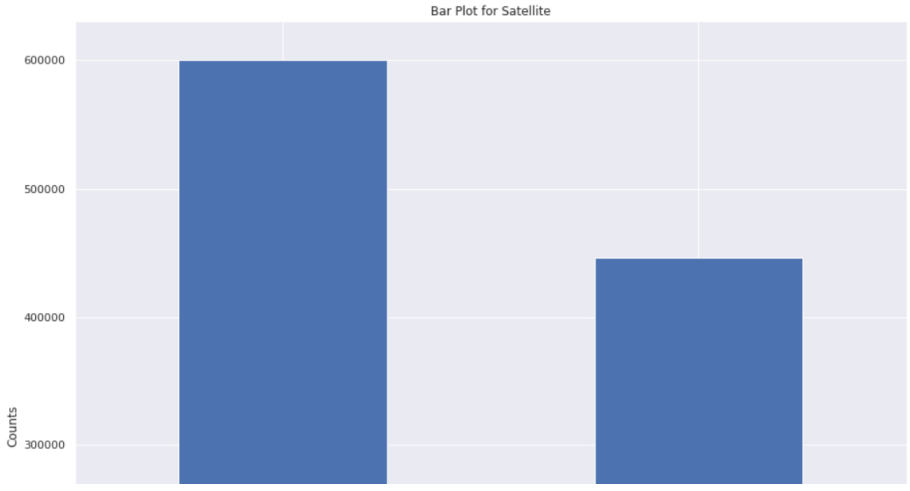
Histogram Observations

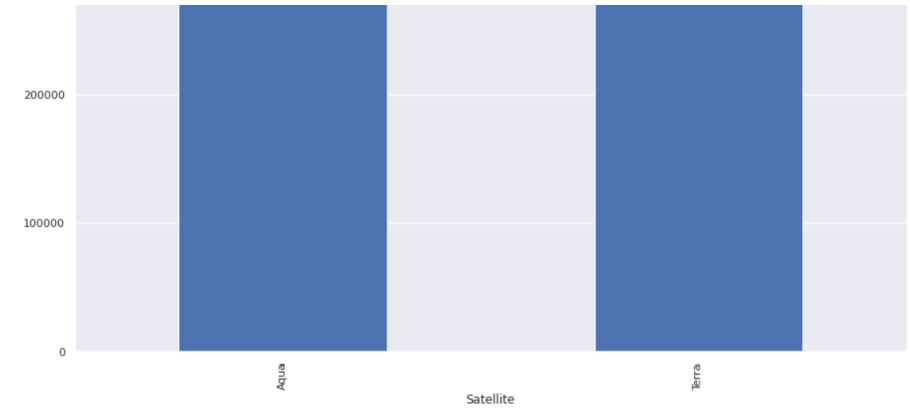
- It can be seen that the brightness feature distribution is unimodal(skewed right)
- The scan feature has an exponential distribution
- The confidence feature has a skewed distribution
- The FRP feature has an exponential distribution

[13]:
```
# Better visualization of the FRP histogram
df.hist(column=['frp'], bins=200, range=(0, 700))
plt.xlabel('FRP Bins')
plt.ylabel('Counts')
plt.title('Histogram Plot for FRP')
plt.show()
```



[14]:
```
# Categorical Features Visualizations

# Satellite feature bar plot
df['satellite'].value_counts().plot.bar();
plt.xlabel('Satellite')
plt.ylabel('Counts')
plt.title('Bar Plot for Satellite')
plt.show()
```

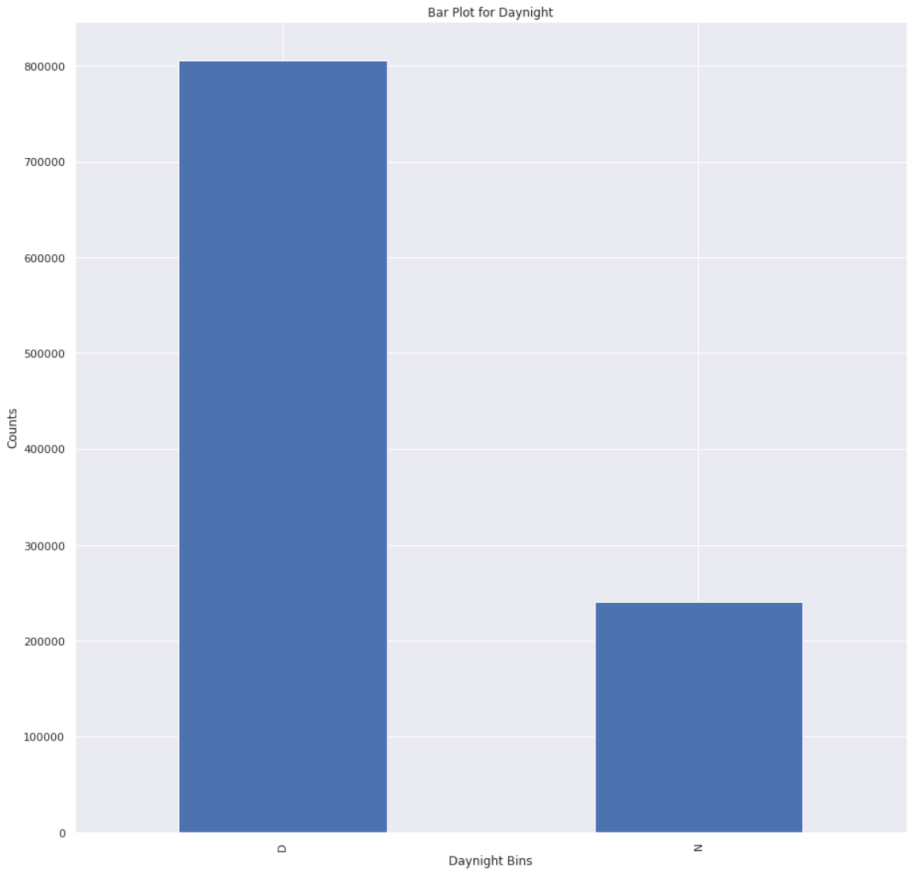




Satellite Bar Plot Observation

- The satellite categories are almost equally distributed

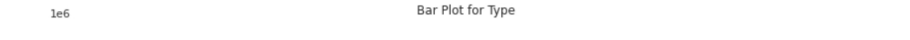
[15]:
```
# Daynight feature bar plot
df['daynight'].value_counts().plot.bar();
plt.xlabel('Daynight Bins')
plt.ylabel('Counts')
plt.title('Bar Plot for Daynight')
plt.show()
```

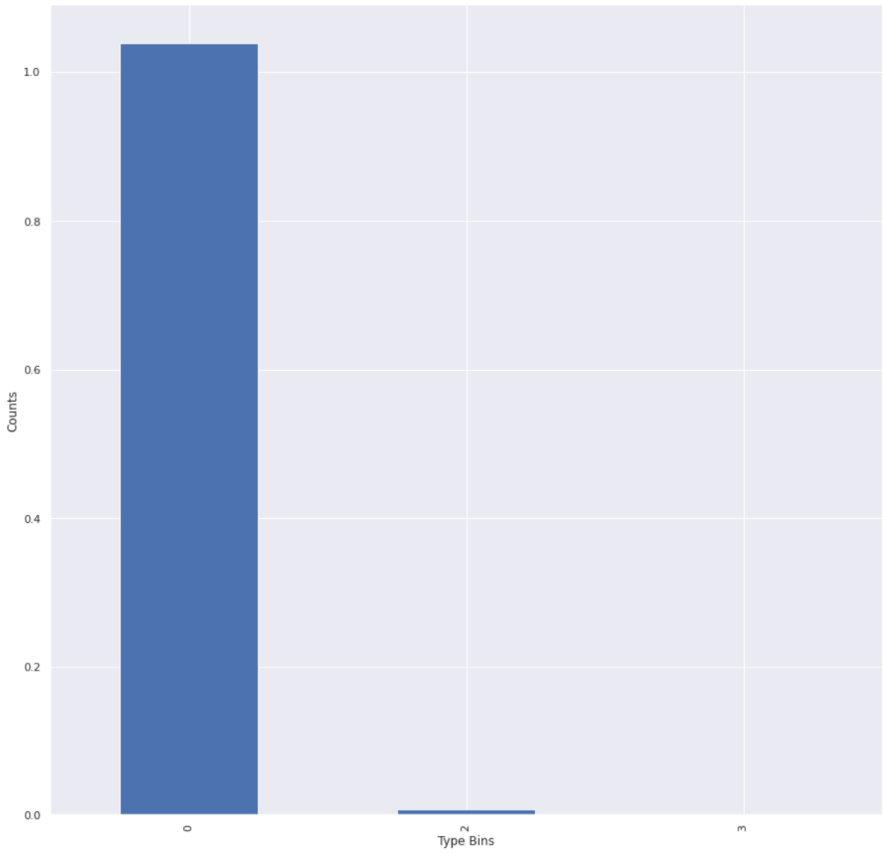


Daynight Bar Plot Observation

- The daynight feature has an unequal distribution

[16]:
```
# Type feature bar plot
df['type'].value_counts().plot.bar()
plt.xlabel('Type Bins')
plt.ylabel('Counts')
plt.title('Bar Plot for Type')
plt.show()
```



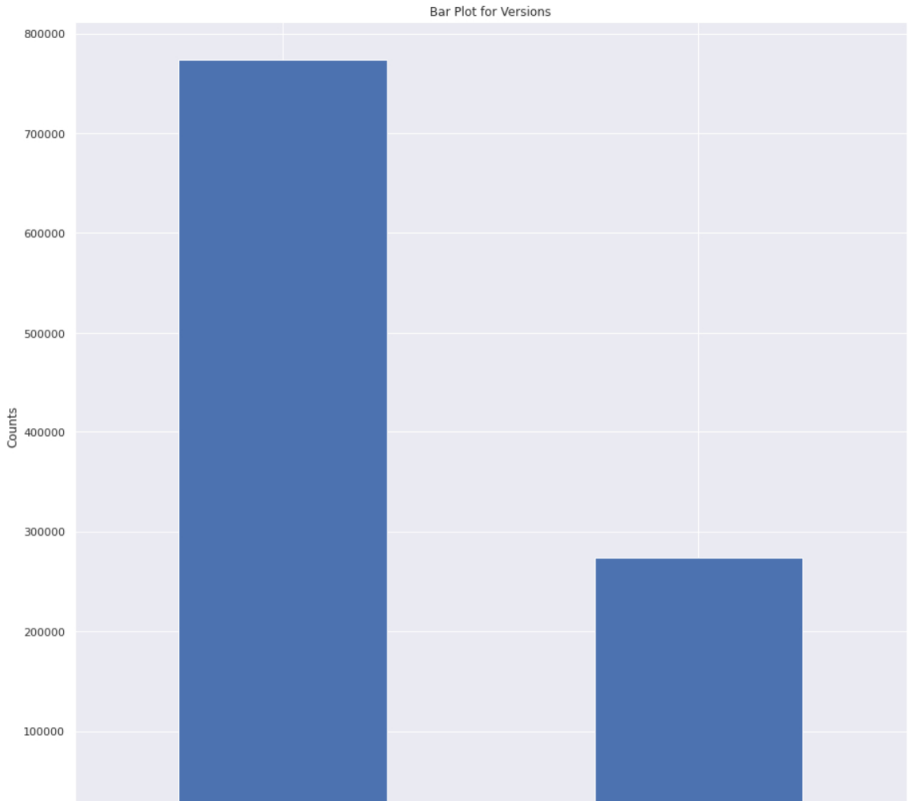


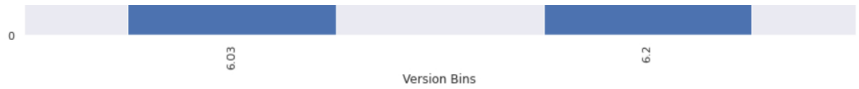
Type Bar Plot Observation

- The type feature has highly skewed data. Almost all data instances are of type 0

[17]:

```
# Version feature bar plot
df['version'].value_counts().plot.bar();
plt.xlabel('Version Bins')
plt.ylabel('Counts')
plt.title('Bar Plot for Versions')
plt.show()
```

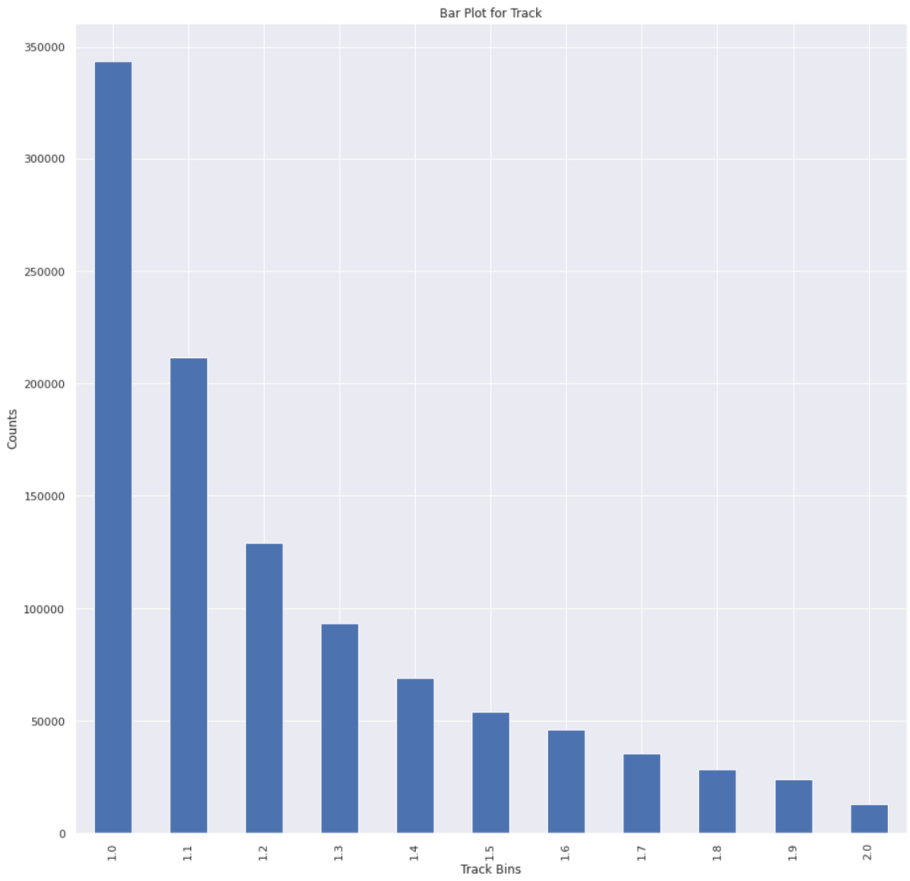




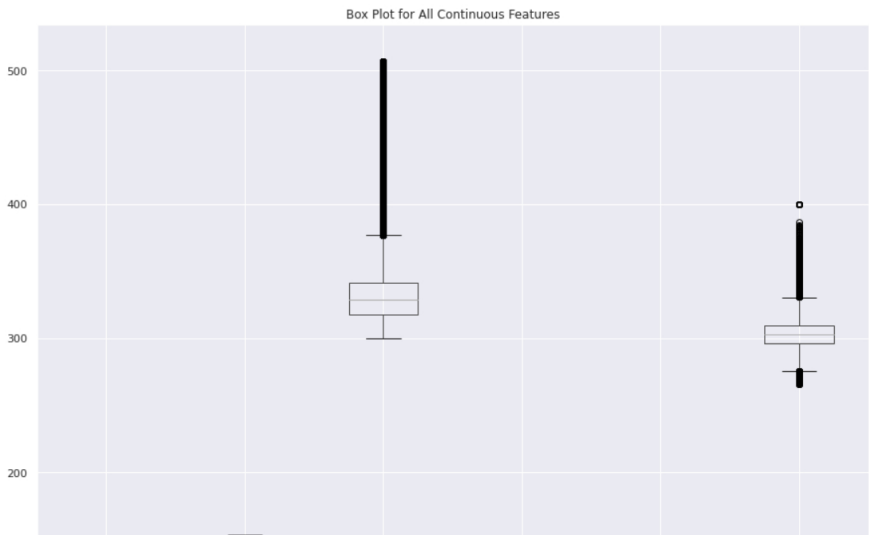
Version Bar Plot Observation

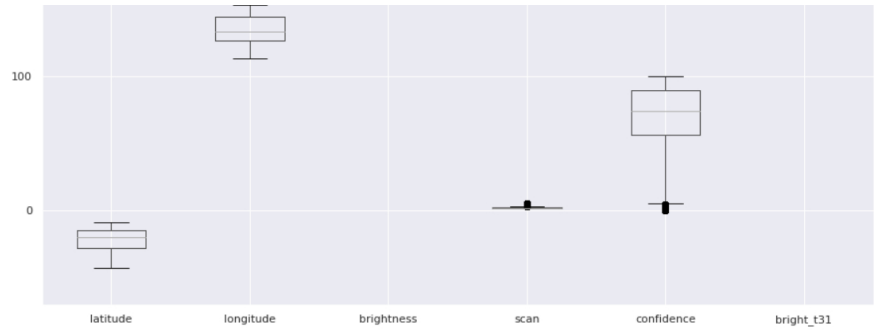
- The version feature has an unequal distribution

[18]:
```
# Track feature bar plot
df['track'].value_counts().plot.bar();
plt.xlabel('Track Bins')
plt.ylabel('Counts')
plt.title('Bar Plot for Track')
plt.show()
```

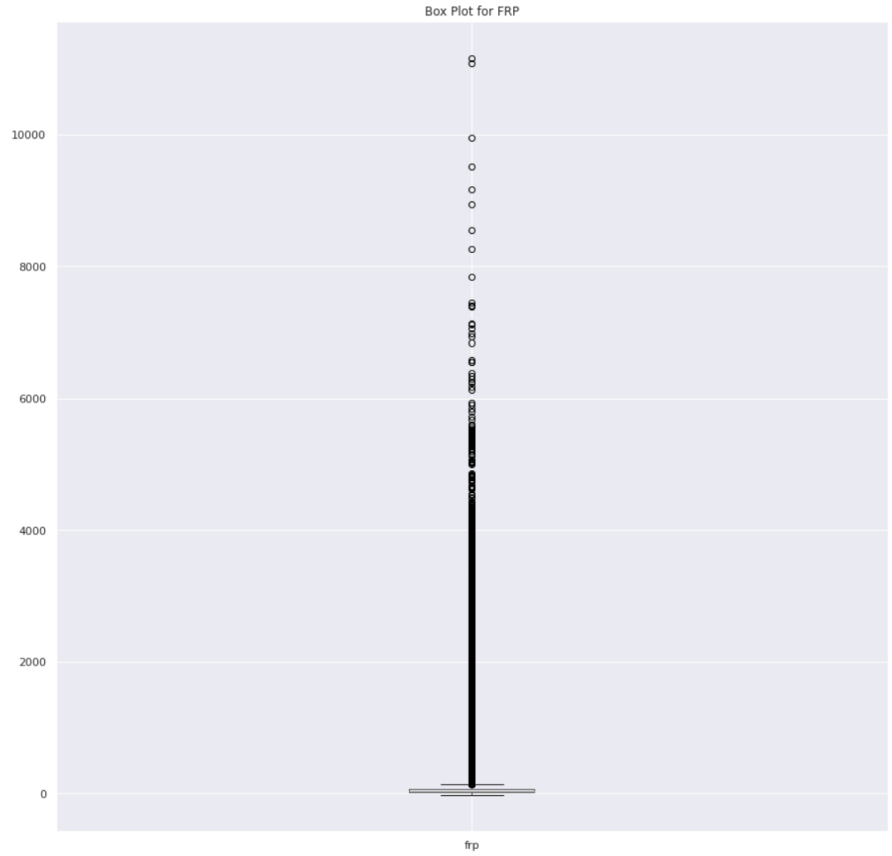



```
df.boxplot(['latitude', 'longitude', 'brightness', 'scan', 'confidence', 'bright_t31'])
plt.title('Box Plot for All Continuous Features')
plt.show()
```






```
df.boxplot('frp')
plt.title('Box Plot for FRP')
plt.show()
```

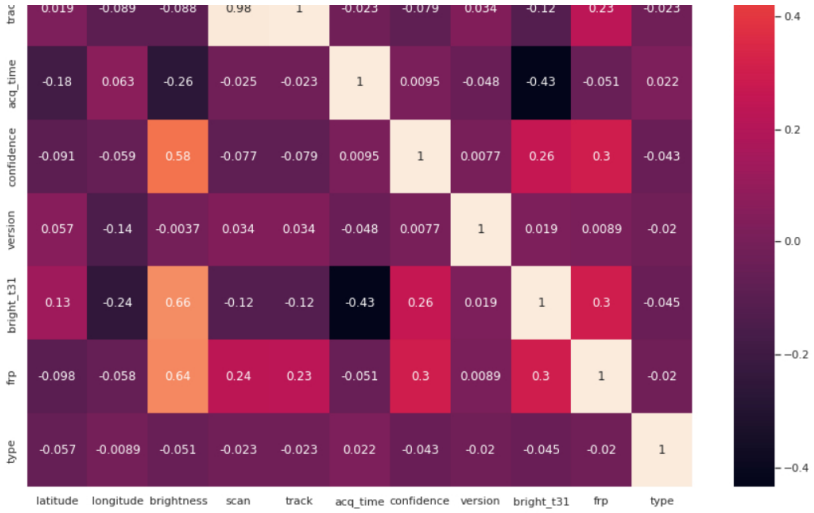


Box Plot Observations

The box plot shows that for the features such as brightness, brightness_t31 and FRP, the data is highly skewed and there are many outliers

1.1.b Heatmap Correlation

[21]:
```
sns.heatmap(df.corr(), annot=True);
```



Heatmap Initial Observations

1. Brightness and FRP are correlated with the confidence feature.
2. Brightness and FRP are also correlated with each other.
3. Although these correlations are not strong, the heatmap shows that there is some relations between the entities.

1.1.c Data Quality Issues

1. There are no missing values in any of the features in the dataset. So, no action is required here.
2. The instrument feature has only 1 unique value. Hence, it is a redundant feature.
3. Type, version, and track are categorical features instead of continuous features.
4. By looking at the box plots and the histograms, we can see that FRP, brightness, and brightness_t31 are highly skewed features.
5. Categorical features are in numerical order. For example, the type has values like 0, 1, and 2. This categorical feature should be one hot encoded to give equal weightage to each class. Value has more weight compared to value 0. So, that will create a biased prediction.

1.1.c Data Quality Plan

1. We are dropping the instrument column because it has a cardinality of 1. It will not add any new information while training the model.
2. We are splitting the date column into two different columns of month and year because we think that there could a relation between bushfires with particular months of a year.
3. Sampling: As the data is very huge, we want to pick samples from the data. We want to do stratified sampling based on the month and year. We are doing this to ensure that we get data from the monthly duration with equal weightage. We have 48 months in total and to have 150k records, we have to take 3200 samples from each month for every year. For the months which have insufficient records, we are oversampling the data till it becomes 3200.
4. Models cannot work with a string value. To train the model, we will convert the string values into numerical values by using one-hot encoding.

1.2 Preprocessing data according to the data quality plan

Dropping the instrument column because of cardinality 1

[22]:
```
df.drop(['instrument'], axis = 1, inplace = True)
```

Changing data type of acq_date from string to Datetime

[23]:
```
df["acq_date"] = pd.to_datetime(df["acq_date"], format="%Y-%m-%d").dt.date
```

Add new columns for year and month

[24]:
```
df["year"] = pd.DatetimeIndex(df["acq_date"]).year
df["month"] = pd.DatetimeIndex(df["acq_date"]).month
```

Sampling Data instances in each year

[25]:
```
df["year"].value_counts()
```

[25]:
```
2019    310484
2018    307359
2017    273312
2020    155524
Name: year, dtype: int64
```

Minimum data is for 1st month of 2017. We have to upsample the data to atleast 3200 for each month. 153600 = 32008 * 12 (months) * 4 (years)

[26]:
```
df.groupby(["year", "month"]).size()
```

[26]:
```
year  month
2017  1         2093
      2         3144
      3         9403
      4        18032
      5        32605
      6        23850
      7        13860
      8        22009
      9        45291
      10       29201
      11       42931
      12       30893
2018  1         8875
      2         3504
      3         8205
      4        33950
      5        41823
      6        20493
      7        19514
      8        18172
      9        27553
      10       52975
      11       44252
      12       28043
2019  1        23078
      2        13956
      3         9670
      4        13220
      5        17085
      6        18235
      7        21983
      8        16254
      9        19757
      10       28696
      11       47138
      12       81412
2020  1        36994
      2         6826
      3         4046
      4         7718
      5        15859
      6        13112
      7        16993
      8         6215
      9         8035
      10       11659
      11       21927
      12        6140
dtype: int64
```

Group by (year + month) and randomly sample out 3200 records at max from each group

[27]:
```
stratified_year_month_data = df.groupby(['month','year'], group_keys=False).apply(lambda x: x.sample(min(len(x), 3200)))
```

[28]:
```
stratified_data_group = stratified_year_month_data.groupby(["year", "month"])
```

Checking if any group has less than 3200 records. In that case, it's randomly sampling the remaining records from that particular group

[29]:
```
samples = [stratified_year_month_data]
for index, group in stratified_data_group:
    samples.append(group.sample(3200-len(group)))


sampled_df = pd.concat(samples)
```

Counts after sampling

[30]:
```
sampled_df.groupby(["year", "month"]).size()
```

[30]:
```
year  month
2017  1        3200
      2        3200
      3        3200
      4        3200
      5        3200
      6        3200
      7        3200
      8        3200
      9        3200
      10       3200
      11       3200
      12       3200
2018  1        3200
      2        3200
      3        3200
      4        3200
      5        3200
      6        3200
      7        3200
      8        3200
      9        3200
      10       3200
      11       3200
      12       3200
2019  1        3200
      2        3200
      3        3200
      4        3200
      5        3200
      6        3200
      7        3200
      8        3200
      9        3200
      10       3200
      11       3200
      12       3200
2020  1        3200
      2        3200
      3        3200
      4        3200
      5        3200
      6        3200
      7        3200
      8        3200
      9        3200
      10       3200
      11       3200
      12       3200
dtype: int64
```

One hot encoding for the type feature

[31]:
```
sampled_df = pd.get_dummies(sampled_df, columns=['type'])
```

Label encoding the daynight column(Because it is a categorical feature)

[32]:
```
label_encoder = preprocessing.LabelEncoder()

sampled_df['daynight'] = label_encoder.fit_transform(sampled_df['daynight'])

sampled_df.head()
```

[32]:

	latitude	longitude	brightness	scan	track	acq_date	acq_time	satellite	confidence	version	bright_t31	frp	daynight	year	month	type_0	type_2	type_3
1317	-23.25740	114.96310	310.90000	1.70000	1.30000	2017-01-22	1802	Aqua	81	6.20000	294.30000	20.60000	1	2017	1	1	0	0
1304	-23.12350	142.15920	317.50000	2.50000	1.50000	2017-01-22	350	Aqua	34	6.20000	291.10000	34.30000	0	2017	1	1	0	0
1694	-24.00420	146.77910	309.30000	1.10000	1.00000	2017-01-26	1250	Terra	71	6.20000	295.20000	8.00000	1	2017	1	1	0	0
702	-32.51980	149.21740	351.00000	1.00000	1.00000	2017-01-13	353	Aqua	93	6.20000	317.20000	38.90000	0	2017	1	1	0	0
1735	-21.59450	147.47750	341.00000	1.20000	1.10000	2017-01-27	408	Aqua	82	6.20000	320.30000	14.20000	0	2017	1	1	0	0

Label encoding satellite and version columns

[33]:
```
sampled_df['satellite'] = label_encoder.fit_transform(sampled_df['satellite'])
sampled_df['version'] = label_encoder.fit_transform(sampled_df['version'])
sampled_df.head()
```

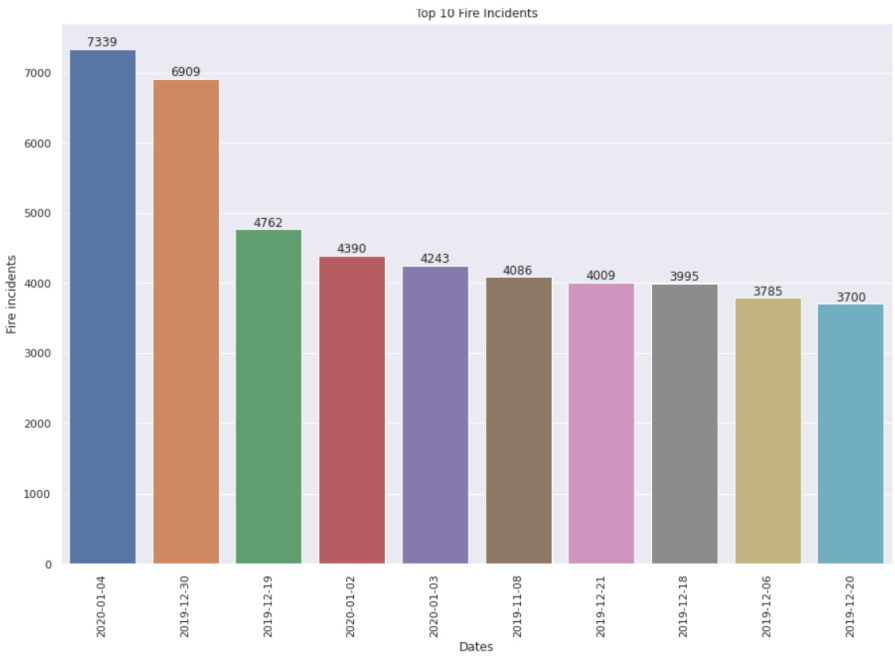
[33]:

	latitude	longitude	brightness	scan	track	acq_date	acq_time	satellite	confidence	version	bright_t31	frp	daynight	year	month	type_0	type_2	type_3
1317	-23.25740	114.96310	310.90000	1.70000	1.30000	2017-01-22	1802	0	81	1	294.30000	20.60000	1	2017	1	1	0	0
1304	-23.12350	142.15920	317.50000	2.50000	1.50000	2017-01-22	350	0	34	1	291.10000	34.30000	0	2017	1	1	0	0
1694	-24.00420	146.77910	309.30000	1.10000	1.00000	2017-01-26	1250	1	71	1	295.20000	8.00000	1	2017	1	1	0	0
702	-32.51980	149.21740	351.00000	1.00000	1.00000	2017-01-13	353	0	93	1	317.20000	38.90000	0	2017	1	1	0	0
1735	-21.59450	147.47750	341.00000	1.20000	1.10000	2017-01-27	408	0	82	1	320.30000	14.20000	0	2017	1	1	0	0

[34]:
```
sampled_df.drop(['acq_date'], axis = 1, inplace = True)
```

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

[35]:
```
# inspiration for bar chart
# https://stackoverflow.com/questions/43214978/seaborn-barplot-displaying-values/68323374#68323374

# Picking only type 0 = vegetation fire
bushfire_data = df.loc[df["type"] == 0]

# Aggregating incidents by the acquisition date
x = bushfire_data["acq_date"].value_counts()


bushfire_counts = x[:10]


plt.figure(figsize=(15,10))
sns.set(font_scale = 1)
ax = sns.barplot(x="Dates", y="Fire incidents", data=pd.DataFrame(list(zip(x.index[0:10], bushfire_counts)), columns=["Dates", "Fire incidents"]))
ax.bar_label(ax.containers[0])
plt.xticks(rotation=90)
plt.title('Top 10 Fire Incidents')
plt.show()
```



Here we have shown the top 10 number of days on which the satelite had recorded the highest number of incidents. On 4th of January 2020 satellite has recorded 7339 number of incidents.

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.

- Based on the heatmap correlation plot, we can see that the confidence feature is correlated with the FRP(0.58).
- Also, brightness and brightness_t31 are loosely correlated. So, we think that there should a relationship between these features.
- Hence, the confidence of an incident should be dependent upon these features(brightness, FRP, and brightness_t31).

2. Spatio-temporal data

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.

[36]:
```
# Filter dataset for the year 2017 only for the Aqua satellite¶
df_aqua_2017 = df_2017[df_2017.satellite == 'Aqua']
```

```
# Choose only latitute longitude and FRP columns
df_aqua_2017 = df_aqua_2017[['latitude', 'longitude', 'frp']]

# Group by the coordinate and calculate mean value for the FRP
df_aqua_2017.groupby(['latitude', 'longitude']).mean()
df_aqua_2017.head()
```

[36]:

	latitude	longitude	frp
35	-31.43490	151.87030	40.20000
36	-29.25750	152.99950	21.20000
37	-29.25600	153.01190	16.40000
38	-27.32900	149.99160	59.90000
39	-27.31560	150.00730	59.80000

Aggregation Method

There are plenty of ways to interpret this information. For example, the max value will show the maximum recorded FRP for a particular coordinate. However, we choose to go with the "MEAN" of the FRP to analyze the average FRP during the selected period (which is 2017 in our case). The mean aggregation will give us an idea of the overall picture of the most affected region throughout the year [1].

[37]:

```
# Re-scale FRP column for satisfying folium requirement
a = df_aqua_2017["frp"]
norm_a = (a - a.min())/(a.max() - a.min())
df_aqua_2017["frp_norm"] = norm_a
df_aqua_2017.drop('frp', axis=1, inplace=True)
df_aqua_2017.head()
```

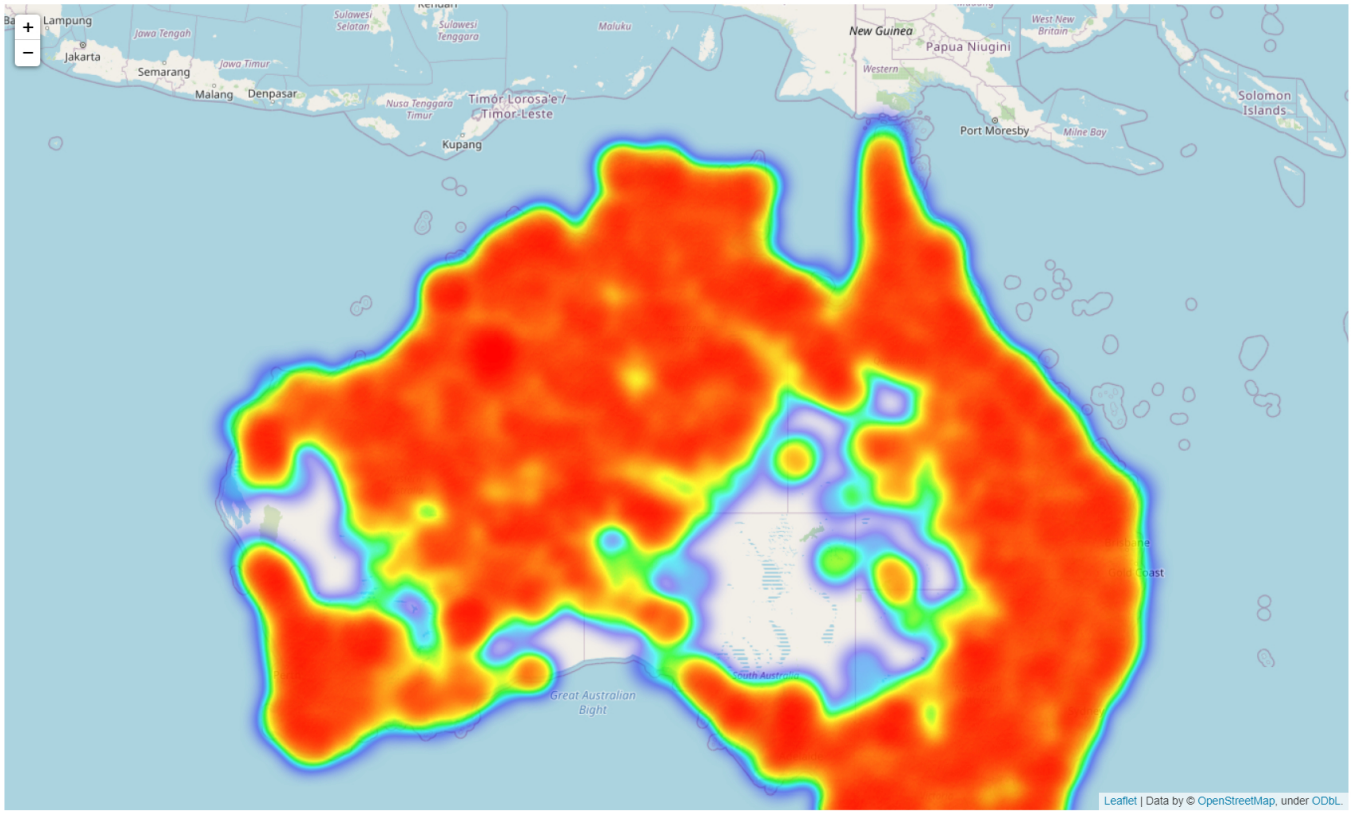
[37]:

	latitude	longitude	frp_norm
35	-31.43490	151.87030	0.00543
36	-29.25750	152.99950	0.00286
37	-29.25600	153.01190	0.00222
38	-27.32900	149.99160	0.00809
39	-27.31560	150.00730	0.00808

[38]:

```
# Plotting geographical heatmap using folium
mapObj = folium.Map(location=[df_aqua_2017["latitude"].mean(), df_aqua_2017["longitude"].mean()], zoom_start=5)

HeatMap(df_aqua_2017).add_to(mapObj)
mapObj
```

[38]:



To visualize the map, please run the notebook

To see the detailed map, zoom in on the map

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

Get all coordinates for the given latitude

[39]:
```
df[df["latitude"] == -35.6]
```

[39]:

	latitude	longitude	brightness	scan	track	acq_date	acq_time	satellite	confidence	version	bright_t31	frp	daynight	type	year	month
812748	-35.60000	150.18980	365.10000	1.30000	1.10000	2019-12-02	2349	Terra	100	6.03000	304.00000	137.60000	D	0	2019	12
815719	-35.60000	150.17990	355.00000	1.80000	1.30000	2019-12-04	32	Terra	87	6.03000	287.80000	165.70000	D	0	2019	12
818540	-35.60000	150.18920	340.80000	1.00000	1.00000	2019-12-05	352	Aqua	72	6.03000	291.40000	35.20000	D	0	2019	12
873383	-35.60000	150.01710	320.50000	2.30000	1.50000	2019-12-27	38	Terra	24	6.03000	296.20000	41.10000	D	0	2019	12
893743	-35.60000	148.19210	318.80000	1.00000	1.00000	2020-01-01	1259	Terra	29	6.03000	289.70000	15.00000	N	0	2020	1
924101	-35.60000	148.97720	333.60000	1.00000	1.00000	2020-01-27	1513	Aqua	100	6.03000	293.70000	31.20000	N	0	2020	1

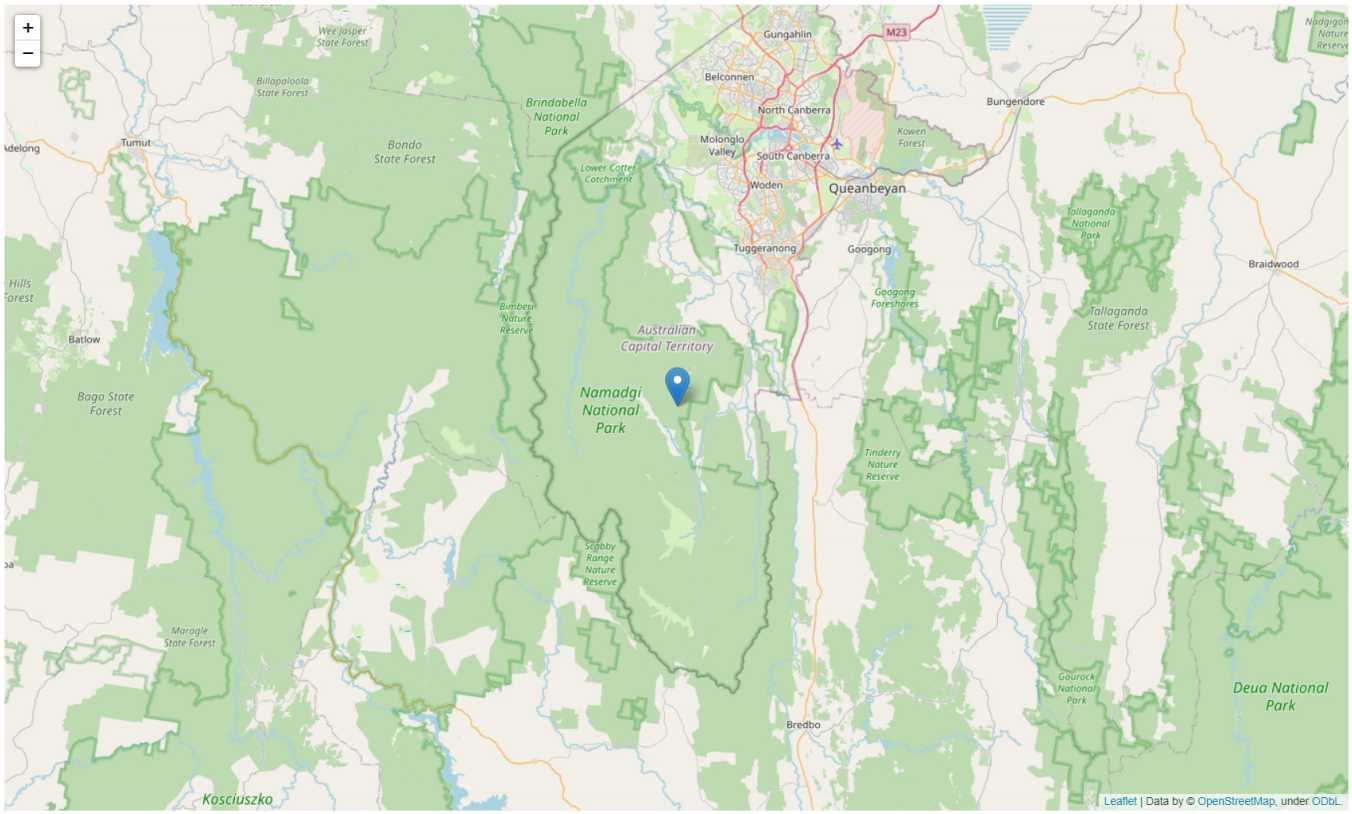
Find the closest location for the given coordinate difference between longitude value 149.12 and all the other values available

[40]:
```
df[df["latitude"] == -35.6]["longitude"].apply(lambda x: abs(x - 149.12))
```

[40]:
```
812748   1.06980
815719   1.05990
818540   1.06920
873383   0.89710
893743   0.92790
924101   0.14280
Name: longitude, dtype: float64
```

[41]:
```
# Mark the activity using folium
from folium.features import DivIcon

mapObj = folium.Map(location=[-35.6, 148.97], zoom_start=10)

folium.Marker([-35.6, 148.97]).add_to(mapObj)
mapObj
```

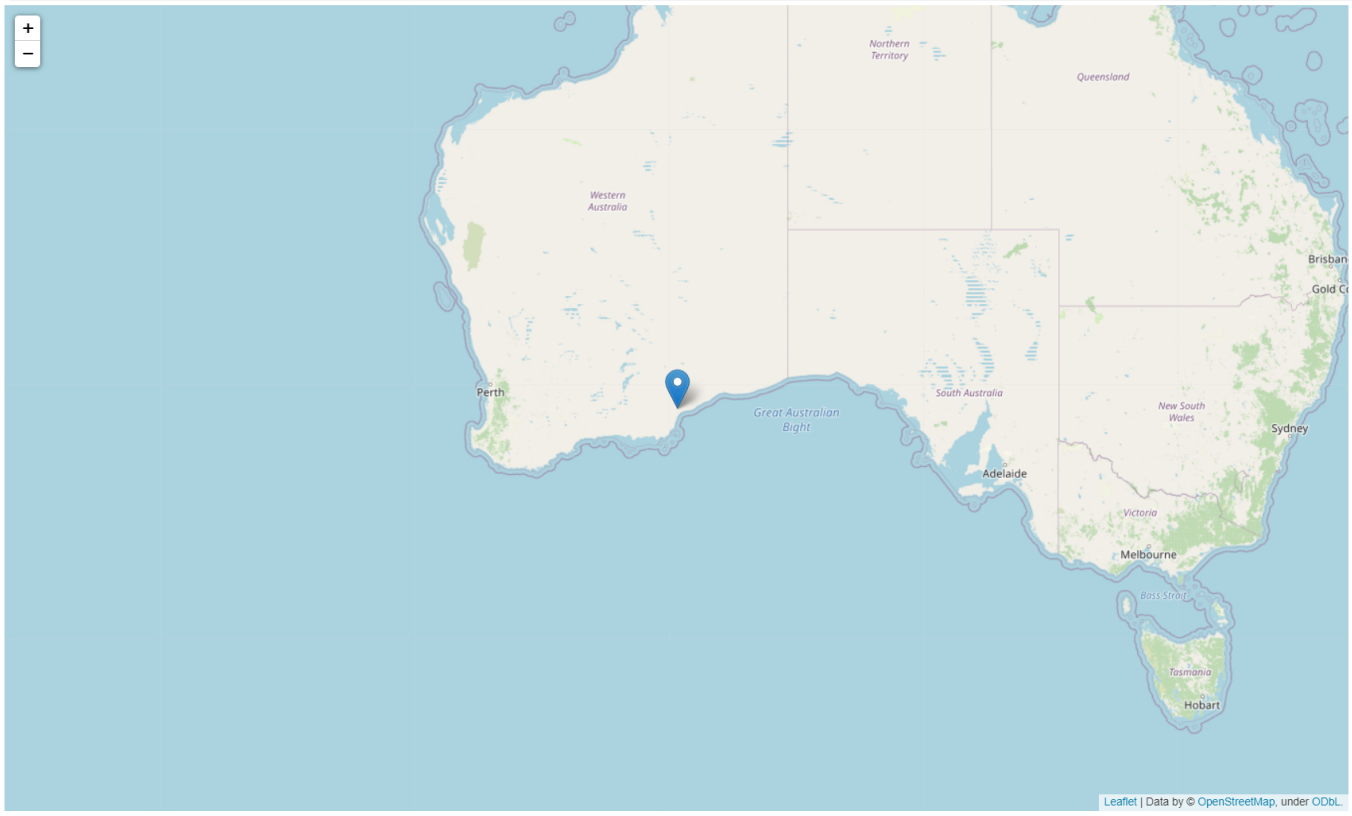
[41]:



To visualize the map, please run the notebook

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

[42]:
```
# Get the max radiation for the given date
location = df.loc[df["frp"] == df[df["acq_date"] == datetime.datetime(2020, 1, 8).date()]["frp"].max()]

# Mark it on the map using folium
mapObj = folium.Map(location=[location["latitude"], location["longitude"]], zoom_start=5)

folium.Marker([location["latitude"], location["longitude"]]).add_child(folium.Popup('Highest Recorded FRP for 01-08-2020')).add_to(mapObj)
mapObj
```

[42]:


To visualize the map, please run the notebook

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

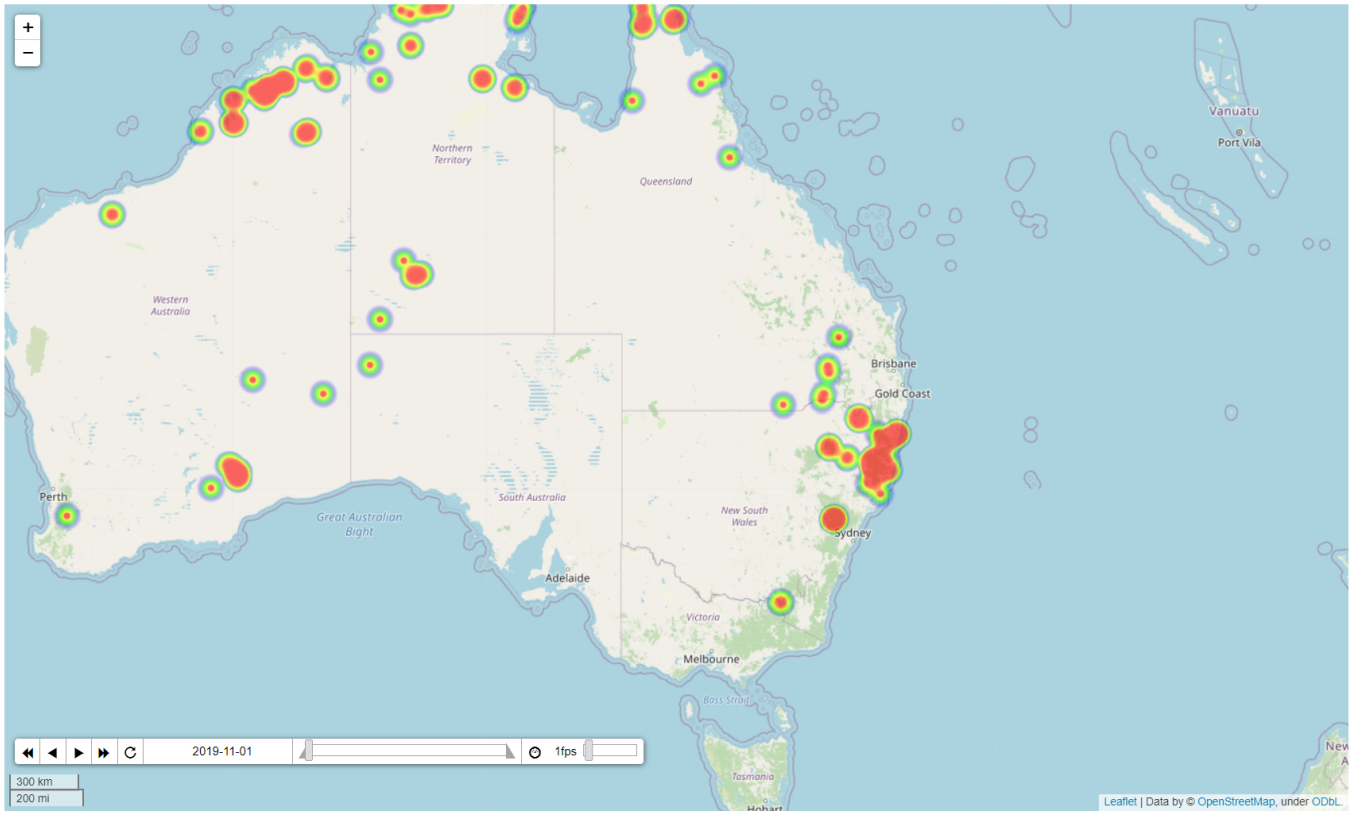
Data is filtered between 1st November 2019 to 31st january 2020. It's only the type = 0 incidents [1].

[43]:
```
mask = ((df["acq_date"] >= datetime.datetime(2019, 11, 1).date()) & (df["acq_date"] <= datetime.datetime(2020, 1, 31).date()))
nav_jan_data = df.loc[mask]
nav_jan_data = nav_jan_data[nav_jan_data["type"] == 0]
```

[44]:
```
map = folium.Map(location=[nav_jan_data["latitude"].mean(), nav_jan_data["longitude"].mean()], zoom_start=5, control_scale=True)
```

[45]:
```
# The data is grouped by date

lat_long_list = []
group_index = []
for index, group in nav_jan_data.groupby("acq_date"):
    temp = []
    group_index.append(str(index))
    for lat, long, frp in zip(group["latitude"],group["longitude"], group["frp"]):
        temp.append([lat, long, frp])
    lat_long_list.append(temp)
```

[45]:
```
# HeatMap with time

timeslider = plugins.HeatMapWithTime(lat_long_list, index=group_index).add_to(map)
map
```

[46]:


To visualize the map, please run the notebook

3. Build a model for spatial prediction of wildfire

3.1. In between 'Brightness temperature I-5' and 'Fire Radiative Power' choose either as the target feature.

We are choosing brightness as our target variable because it is correlated with confidence and FRP. The confidence variable shows the probability of bush fire in that particular region. And as brightness is a continuous variable, we want to do a regression task on this data. We are using brightness because it is a good indicator of a bush fire.

3.2. Explain what the task you're solving is (e.g., supervised x unsupervised, classification x regression x clustering or similarity matching x, etc)

Since we already know the target variable, this is a supervised learning problem. Both the given target variables (FRP and brightness) are continuous. Hence, a regression model is the most suitable model to predict the target variable. We are performing regression to find out the brightness for indicating whether that region is going to be affected by a bush fire.

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

[47]:
```
# Separate input features and the selected target variable
# Output variable
Y = sampled_df['brightness']
# Input variable
X = sampled_df.drop(['brightness'], axis=1)
```

[48]:
```
# Use select K best method to get the best features to predict the output variable
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression


selector = SelectKBest(mutual_info_regression, k=4)
selector.fit_transform(X, Y)
# Get columns to keep and create new dataframe with those only
cols = selector.get_support(indices=True)
```

```
X_new = X.iloc[:,cols]
X_new.head()
```

[48]:

	acq_time	confidence	bright_t31	frp
1317	1802	81	294.30000	20.60000
1304	350	34	291.10000	34.30000
1694	1250	71	295.20000	8.00000
702	353	93	317.20000	38.90000
1735	408	82	320.30000	14.20000

[49]:
```
X_new = X_new.astype('float64')
```

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

There are plenty of evaluation metrics available for regression tasks. For example,

1. Mean absolute error(MAE)

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

We take absolute difference between actual value and predicted value and take a mean by summing all N values and later on dividing it by N. This metrics is very basic and it shows a direct representation of the total error.

2. Mean Squared error (MSE)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Instead of taking a absolute difference into consideration, we do square of the difference and add all the N differences and divide that sum with N. This way the larger differences (Wrong Predictions) will have more contribution in the total MSE. But one of the disadvantages of MSE is it's affected by scaling [a]. The MSE is different when you do scale your big feature (e.g house price in 100s of thounds) between 0 to 1.

3. R Square

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i(y_i - \hat{y}_i)^2}{\sum_i(y_i - \bar{y})^2}$$

R Square is calculated by the sum of squared of prediction error divided by the total sum of the square which replaces the calculated prediction with mean. R Square measures how much variability in dependent variable can be explained by the model. Also, the R Square value lies between 0 to 1, so we can show it in terms of percentage of variablity explained by the model [8].

We would like to choose R square as our evaluation metrics for this regression task. It gives a good idea about the variability explained (in percentage) by the trained model. It's not affected by feature scaling as well [5].

5. Build a baseline model

KNN algorithm is used in classification and regression problems. This algorithm works on the "feature similarity" to predict the new datapoint's value. In other words, it assigns value based on how closely it resembles the points in the training set. Here we have data with geo-temporal features like latitude, longitude, acquisition date, etc. Therefore, we have hypothesized that using the KNN regressor might be a good baseline model choice for the data [3].

[50]:
```
# here we are split the data into training (80%) and testing (20%) sets.
X_new = X_new.astype("float64")
train_X, test_X, train_y, test_y = train_test_split(X_new, Y, test_size = 0.2, random_state = 123)
```

[51]:
```
# KNN Regressor
# Initially we have taken n_neighbors = 5
neigh = KNeighborsRegressor(n_neighbors=5)
neigh.fit(train_X, train_y)
```

[51]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

[52]:
```
# Predict the model
```

```
pred = neigh.predict(test_X)
# MAE, RMSE, R2 Computation
mae = MAE(test_y, pred)
rmse = np.sqrt(MSE(test_y, pred))
rsquare = R2(test_y, pred)
print("MAE : % f" %(mae))
print("RMSE : % f" %(rmse))
print("R-Square : % f" %(rsquare))
```

```
MAE :  4.283283
RMSE :  7.183898
R-Square :  0.895536
```

With an arbitary selection of k=5 our model is getting 89% R square value, which means the trained model is able to explain 89% of the variance in the testing data. But in order to find an optimal value of k we will run a loop over it and check the R Square.

What are you doing to avoid overfitting?

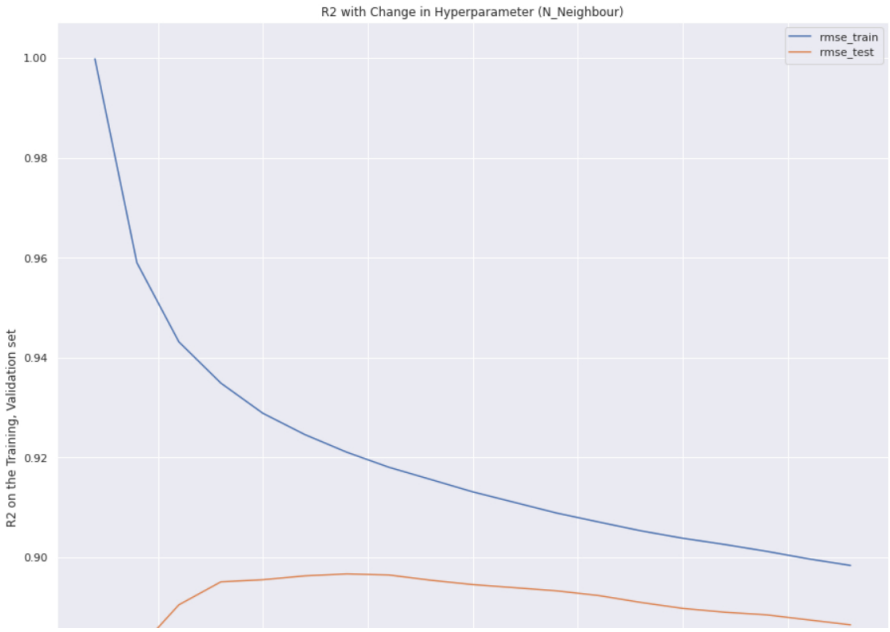
In KNN based approach, the important hyperparameter is the value of k. If we use a small k (Example 1 or 2), the model will overfit to its nearest similar data points. At the same time, if we use the big value of k, it will under-fit the model. Here, we have changed the value of k over a range of 20 with increment of 1. We have tried to find an optimal value of k for which the model is giving least error on validation (here it's the testing data) dataset.

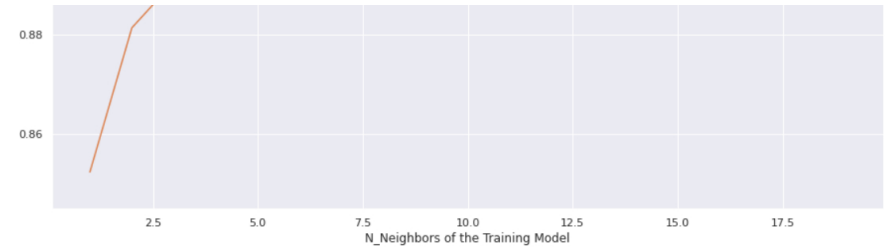
[53]:
```
# model training function for KNN regressor
def knn_train_model(n):
    model = KNeighborsRegressor(n_neighbors=n)
    model.fit(train_X, train_y)
    return model

# prediction function
def knn_prediction(model, X):
    # Predict the model
    pred = model.predict(X)
    return pred

r2_train = []
r2_test = []

# Training on KNeighborsRegressor over range(1,20) n_neighbors
# Testing the model prediction on training and testing data
for i in tqdm(range(1, 20)):
    model = knn_train_model(i)
    train_pred = knn_prediction(model, train_X)
    test_pred = knn_prediction(model, test_X)
    r2_train.append(R2(train_y, train_pred))
    r2_test.append(R2(test_y, test_pred))
```

```
100%|██████████| 19/19 [00:51<00:00,  2.69s/it]
```

Visualization of the training process

[54]:
```
plt.plot(range(1, 20), r2_train, label='rmse_train')
plt.plot(range(1, 20), r2_test, label='rmse_test')
plt.legend(loc='upper right')
plt.xlabel("N_Neighbors of the Training Model")
plt.ylabel("R2 on the Training, Validation set")
plt.title("R2 with Change in Hyperparameter (N_Neighbour)")
plt.show()
```





We can see that with change in the value of n_neighbors hyperparameter the evaluation metrics (R2) is decreasing on trianing data, but on the validation data the R2 increased till n_neighbours = 6 and then after it started to decrease again. This graph shows that the n_neighbours = 6 is the best hyperparameter value for our dataset. Using values above 6 will overfit the model.

[55]:
```
knn_final = KNeighborsRegressor(n_neighbors=6)
knn_final.fit(train_X, train_y)

# Predict the model
knn_final_prediction = neigh.predict(test_X)

knn_rsquare = R2(test_y, knn_final_prediction)

print("R-Square for final KNN regressor model: % f" %(knn_rsquare))
```

```
R-Square for final KNN regressor model:  0.895536
```

6. Build a candidate final model (you are encouraged to experiment with more that one model but only include and discuss your “best” model)

We all know that gradient boosting is one of the most powerful algorithms for classification and regression problems except the neural networks. It works on the hypothesis that the next best possible model combined with previous models will give the least prediction error. XGBoost is also an ensemble learning-based gradient boosting algorithm that works on the same philosophy. Here we are hypothesizing that it will perform well with the geo-temporal data [4].

[56]:
```
# XGB
# Initialy we have choose random hyperparameter values for the XGB regressor
# n_estimators is the max number of weak learners
# max_depth is the Xgb tree depth / level
# eta is the learning rate
xgb_r = xg.XGBRegressor(n_estimators = 35, seed = 123, max_depth=10, eta=0.25)
```

[57]:
```
# model training
eval_set=[(test_X, test_y)]
xgb_r.fit(train_X, train_y, eval_set=eval_set)
```

```
[0]    validation_0-rmse:248.66270
[1]    validation_0-rmse:186.56861
[2]    validation_0-rmse:140.00112
[3]    validation_0-rmse:105.08831
[4]    validation_0-rmse:78.91233
[5]    validation_0-rmse:59.31306
[6]    validation_0-rmse:44.65244
[7]    validation_0-rmse:33.70121
[8]    validation_0-rmse:25.56064
[9]    validation_0-rmse:19.54361
[10]   validation_0-rmse:15.12732
[11]   validation_0-rmse:11.93968
[12]   validation_0-rmse:9.69411
[13]   validation_0-rmse:8.16350
[14]   validation_0-rmse:7.15813
[15]   validation_0-rmse:6.51862
[16]   validation_0-rmse:6.12346
[17]   validation_0-rmse:5.89494
[18]   validation_0-rmse:5.75745
[19]   validation_0-rmse:5.67456
[20]   validation_0-rmse:5.62993
[21]   validation_0-rmse:5.60376
[22]   validation_0-rmse:5.58033
[23]   validation_0-rmse:5.57458
[24]   validation_0-rmse:5.56186
[25]   validation_0-rmse:5.55037
[26]   validation_0-rmse:5.54749
[27]   validation_0-rmse:5.54642
[28]   validation_0-rmse:5.54609
[29]   validation_0-rmse:5.54604
[30]   validation_0-rmse:5.53752
[31]   validation_0-rmse:5.53446
[32]   validation_0-rmse:5.52603
[33]   validation_0-rmse:5.52222
[34]   validation_0-rmse:5.52235
```

[57]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False, eta=0.25,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.25, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=10, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=35, n_jobs=0,
             num_parallel_tree=1, objective='reg:squarederror',
             predictor='auto', random_state=123, ...)
```

Note: Here model evaluation is measured by RMSE.

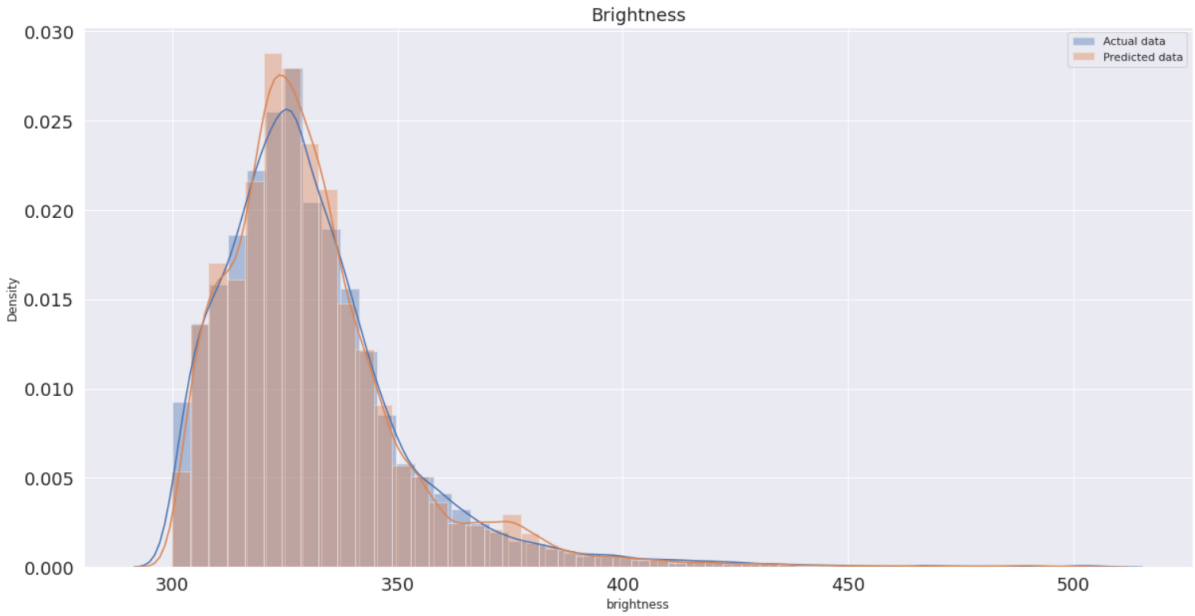
[58]:
```
pred = xgb_r.predict(test_X)
# MAE, RMSE, RSquare Computation
mae = MAE(test_y, pred)
rmse = np.sqrt(MSE(test_y, pred))
r2 = R2(test_y, pred)
print("MAE : % f" %(mae))
print("RMSE: % f" %(rmse))
print("R2 : % f" %(r2))
```

```
MAE :  3.572359
RMSE:  5.522347
R2 :  0.938271
```

With an arbitary selection of n_estimators=35, max_depth=10 and learning_rate (eta) is 0.25. Our model is getting 93% R square value, which means the trained model is able to explain 93% of the variance in the target variable. But in order to find an optimal value of n_estimators we will run a loop over it and check the R Square.

Visualization the distribution of actual and predicted data

```
sns.set(rc ={'figure.figsize':(20,10)})
sns.distplot(test_y, label="Actual data")
sns.distplot(pd.Series(pred), label="Predicted data")
plt.legend()
plt.title('Brightness', fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.show()
```



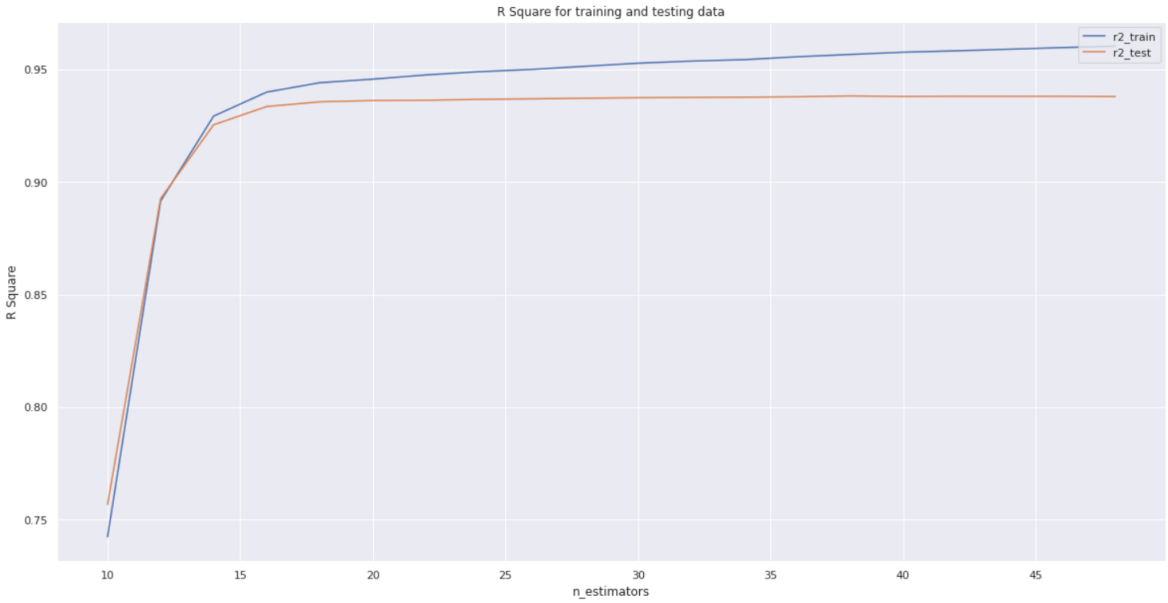
Here, we have tried to plot the last trained estimator after training is completed. Both the distributions are very similar.

[60]:
```
# Training on XGB Regressor
def xgb_train_model(n):
    model = xg.XGBRegressor(objective ='reg:linear',n_estimators = n, seed = 123, max_depth=10, verbosity = 0)
    model.fit(train_X, train_y)
    return model

# Prediction model
def xgb_prediction(model, X):
    pred = model.predict(X)
    return pred


r2_train = []
r2_test = []

for i in tqdm(range(10, 50, 2)):
    model = xgb_train_model(i)
    train_pred = xgb_prediction(model, train_X)
    test_pred = xgb_prediction(model, test_X)
    r2_train.append(R2(train_y, train_pred))
    r2_test.append(R2(test_y, test_pred))
```

100%|██████████| 20/20 [00:41<00:00, 2.08s/it]

Visualization of the training process

[61]:
```
plt.plot(range(10, 50, 2), r2_train, label='r2_train')
plt.plot(range(10, 50, 2), r2_test, label='r2_test')
plt.legend(loc='upper right')
plt.xlabel("n_estimators")
plt.ylabel("R Square")
plt.title("R Square for training and testing data")
plt.show()
```



We can observe that after around n_estimators = 20 our models validation/testing score is not increasing. It shows that the model might be overfitting on the training data. We can choose n_estimators = 20 as the best hyperparameter for our model.

[62]:
```
xgb_model = xg.XGBRegressor(objective ='reg:linear',n_estimators = 20, seed = 123, max_depth=10, verbosity = 0)
xgb_r.fit(train_X, train_y)

xgb_final_prediction = xgb_r.predict(test_X)

xgb_rsquare = R2(test_y, xgb_final_prediction)

print("R-Square for final XGB regressor model: % f" %(xgb_rsquare))
```

```
R-Square for final XGB regressor model:  0.938271
```

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

Here, our null hypothesis will be there is no statistical differece between both model (KNN Regressor, XGB Regressor) predictions.

[63]:
```
scipy.stats.ttest_ind(knn_final_prediction, xgb_final_prediction)
```

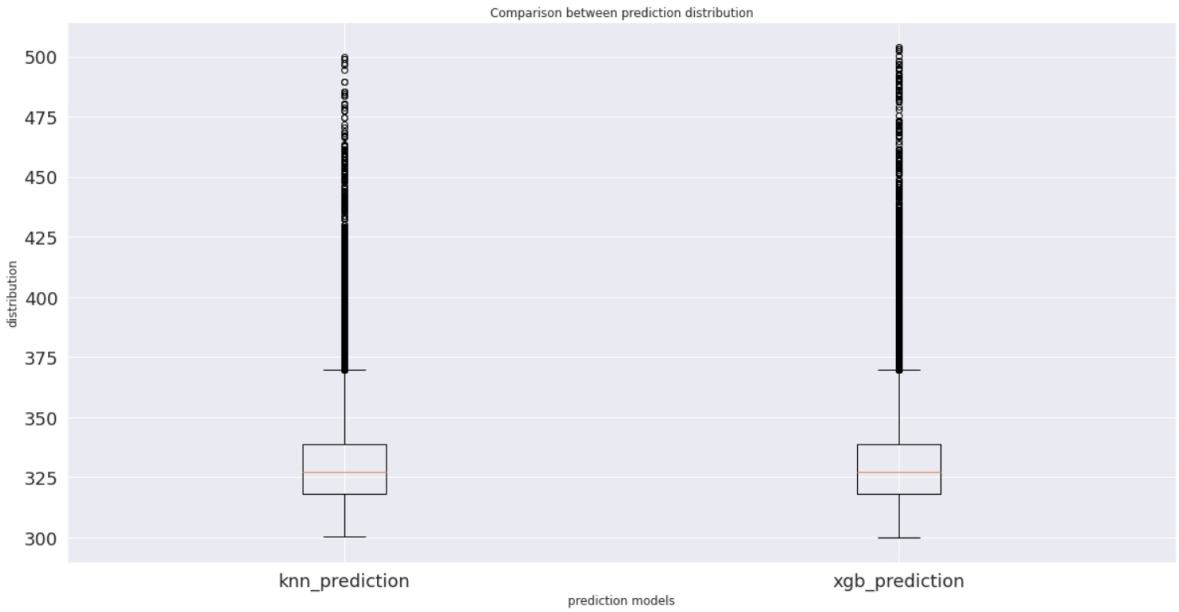
[63]:
```
Ttest_indResult(statistic=-0.434002632419718, pvalue=0.6642880534495363)
```

Here the higher pvalue indicates that the statistical test between two distributions is insignificant and we can accept the null hypothesis.

[64]:
```
final_predictions = pd.DataFrame({'knn_prediction': knn_final_prediction, 'xgb_prediction': xgb_final_prediction})
final_predictions.reset_index(drop=True, inplace=True)
```

[65]:
```
fig1, ax1 = plt.subplots()
ax1.set_title('Basic Plot')
ax1.boxplot(final_predictions)
plt.xticks([1, 2], ['knn_prediction','xgb_prediction'])
plt.xlabel("prediction models")
plt.ylabel("distribution")
```

```
plt.ylabel("distribution")
plt.title("Comparison between prediction distribution")
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.show()
```



We can see very similar prediction distribution for both the final models for KNN Regressor and XGB Regressor. As our final R square errors for both the models were 0.89 and 0.93 and difference is not so big, respectively we were expecting the prediction distribution to be similar.

References

[1] https://python-visualization.github.io/folium/
[2] https://stackoverflow.com/questions/43214978/seaborn-barplot-displaying-values/68323374
[3] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
[4] https://www.datatechnotes.com/2019/06/regression-example-with-xgbregressor-in.html
[5] https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b
[6] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_regression.html
[8] https://vitalflux.com/mean-square-error-r-squared-which-one-to-use/