A Study on the Correlation of Weather Station Data and the Occurence of Wet Avalanches

Contribution

Mahad:

> Mahad handled all of the practical analysis done on the data. Mahad also performed most of the data organization and machine learning done on the data.

Grace:

> Grace handled all of the linear and logistic regression performed on the data. Grace did assist in organizing and cleaning up the data for her own practice; however, the code used for these tasks in this notebook is Mahad's. Grace organized and prepared the notebooks for submission. Grace wrote the abstract, introduction, and other pieces of information throughout the notebook. Lastly, she helped to research details about wet avalanches as well as different methods for analyzing the data.

Daniela:

> Daniela communicated with sponsors, professor, and team mates. She set up a meeting with the sponsors and kept track of all emails and data. Daniela also recorded all information and next steps during presentations. She helped with researching Pycaret and defaults along with researching methods that showed the most accuracy.Worked on the testing

> and training portion. Looked at if there were changes between 60:40, 70:30, and 80:20.

Required Files:

The following xlsx files contain the data needed to run the notebook and are available in the CAIC drive folder. They are read in under 'Data Import and Pre-Processing'

- All Wet Avalanches 2014-Present.xlsx
- OriginalWeatherData.xlsx

It is important to note that all of the Machine Learning Code underneath the sub section 'Prediction Model' takes at least a minute, likely more, to run for each box of code

Abstract

This report organizes and provides analysis for data provided by the Colorado Avalanche Information Center, regarding wet avalanches. Our goal was to clean and subsequently analyze the wet avalanche data recorded by CAIC as well as the corresponding weather station data in order to find any correlation between the various data points. Overall, this notebook provides a template for organizing, analyzing and predicting wet avalanches; however, the predictions could be more precise with more locations. In the future, we hope that this information can be used to study the causes of wet avalanches in order to predict when they occur.

Introduction

The Colorado Avalanche Information Center is a program within the Colorado Department of Natural Resources that is committed to educating the public on avalanche safety as well as forecasting avalanche conditions. Currently, the CAIC employs a system that is titled '[The Avalanche Problem]', which uses four characteristics (avalanche character or type, location, likelihood, and size) to determine the avalanche hazard rating throughout the mountains each day.

Our research focuses on two of these avalanche characters, Wet Slab and Wet Loose, which we have combined to the single term 'wet avalanches' for the sake of the study. There is little research surrounding wet avalanches and their causes, so our goal was to take the weather station data and wet avalanche data recorded by the CAIC and study their correlation. We hope that the research that we have done can be used as a first-step in identifying where and why wet avalanches form.

At the start of our research, we relied primarily on graphics in order to visualize the status of various weather varibles alongside the occurence of wet avalanches. Time series and distribution plots were used for this portion of the research. Linear regression was used to help identify how each weather variable directly correlates to avalanche occurence, and machine learning tools were implemented to start generating predictive modeling options for future use.

Methods

There are two key pieces of data that we are using in this project, wet avalanche data and weather station data. The wet avalanche data was given to us by the CAIC team in the form of a 664 KB .csv file. We then turned it into a .xlsx file (All Wet Avalanches 2014-Present.xlsx); however, the .csv form of the file is still available in the CAIC drive folder. The SNOTEL weather station data was downloaded from the National Resources Conservation Center website. There are 13 SNOTEL stations that were

used in this project; so, the data from each station were combined into one .xlsx file (OriginalWeatherData.xlsx) and is 6 MB. The wet avalance data is both categorical and numerical. A description of each column and its units is listed below:

Columns and units for 'All Wet Avalanches 2014-Present.xlsx':

1. id = id
2. obs_id = Observation id
3. avi_hw_op_bc = Was the avalanche in a highway, within an operation (ski area) or backcountry
4. avi_hw_zone_id = If highway what is the pass id (-1 = not highway)
5. avi_path = Avalanche path name if known. These are mostly highway avalanche paths where the name is known.
6. avi_op_name = name of operation if within an operating boundary
7. avi_loc = general area of avalanche from a drop down list
8. avi_bc_zone_id = if avalanche is a backcountry avalanche which CAIC zone is it in
9. avi_mark = Location within a backcountry zone if known
10. avi_number = number of avalanches reported at that place and time
11. avi_type = type of avalanche (WL = wet loose, WS = wet slab)
12. avi_aspect = Compas aspect if known
13. avi_elev = elevation compared to treeline can be (>TL = above treeline, TL = at treeline, <TL = below treeline)
14. avi_rsize = avalanche size relative to the largest avalanche that is possible from a slide path
15. avi_dsize = avalanche destructive size (1 = relatively harmless, 2 large enough to kill or injure a person to 5 largest avalanche known to man historic)

16. avi_prim_trig = avalanche trigger (N = natural, anything starting with A* = artificially triggered could be by a skier or could be by explosives)
17. avi_sec_trig = additional information describing the primary trigger
18. avi_comments = text comments about the avalanche
19. avi_date = date
20. avi_date_known = is the date known, unknown or estimated
21. avi_time_known = is the time known, unknown or estimated
22. avi_area = another descriptor of location
23. avi_angle_avg = average slope angle of the avalanche path
24. avi_angle_max = maximum slope angle on that particular slope
25. avi_elevation = elevation if known usually in feet
26. avi_elevation_units = usually feet, but can be meters as well
27. avi_surface = were did the avalanche release within the snowpack
28. avi_weak_layer = what was the weak layer that the avalanche released on if known
29. avi_grain_type = what was the grain type of the weak layer if known
30. avi_crown_avg = average crown height of the avalanche. average depth of the avalanche
31. avi_crown_max = maximum crown height of the avalanche. maximum depth of the avalanche
32. avi_crown_units = units for crown depth in = inches, cm = centimeters
33. avi_width_avg = average width of the avalanche
34. avi_width_max = maximum width of the avalanche
35. avi_width_units = units for avalanche width 36. usually ft = feet or m = meters
36. avi_vertical_avg = what is the average vertical fall of the avalanche

37. avi_vertical_max = what is the maximum vertical fall of the avalanche
38. avi_vertical_units = units for vertical fall
39. avi_terminus = where did the avalanche stop (terminus) TP = Top of path, BP = Bottom path, MP = Middle of Path
40. avi_road_status = if an avalanche hit a roadway was the roadway open or closed
41. avi_road_depth = what was the depth of avalanche debris if the avalanche hit the roadway
42. avi_road_length = what was the width of the avalanche if the avalanche hit the roadway
43. avi_road_units = units for avalanche width and epth on roadway

It is important to note that not all of the data from 'All Wet Avalanches 2014-Present.xlsx' will be used. This is discussed more in the section titled Data Import and Pre-Processing of the notebook.

The weather station data is just numerical. A description of each column and it's units is listed below:

Columns and units for 'OriginalWeatherData.xlsx':

1. Date = date
2. Snow Water Equivalent (in) Start of Day Values = inches
3. Precipitation Accumulation (in) Start of Day Values = inches
4. Air Temperature Maximum (degF) = degrees farenheit
5. Air Temperature Minimum (degF) = degrees farenheit
6. Air Temperature Average (degF) = degrees farenheit
7. Precipitation Increment (in) = inches

Data Import and Pre-Processing

This section contains the code as well as descriptions for how the wet avalanche data and weather station data was read-in, cleaned, organized and combined for us to analyze.

Some of the import statements, as well as this next box of code are not for use in this section but in future code.

*The code below may take longer than a minute to run

```
pip install pycaret -q
```

```
     |████████████████████████████████| 256k
     |████████████████████████████████| 66.3
     |████████████████████████████████| 1.6M
     |████████████████████████████████| 102k
     |████████████████████████████████| 174k
     |████████████████████████████████| 266k
     |████████████████████████████████| 13.9
     |████████████████████████████████| 276k
     |████████████████████████████████| 6.8M
     |████████████████████████████████| 157.
     |████████████████████████████████| 1.8M
     |████████████████████████████████| 2.1M
     |████████████████████████████████| 71kB
     |████████████████████████████████| 71kB
     |████████████████████████████████| 3.1M
     |████████████████████████████████| 604k
     |████████████████████████████████| 153k
     |████████████████████████████████| 337k
     |████████████████████████████████| 81kB
     |████████████████████████████████| 163k
     |████████████████████████████████| 61kB
     |████████████████████████████████| 1.1M
     |████████████████████████████████| 296k
     |████████████████████████████████| 204k
     |████████████████████████████████| 92kB
     |████████████████████████████████| 133k
     |████████████████████████████████| 2.6M
     |████████████████████████████████| 71kB
     |████████████████████████████████| 81kB
     |████████████████████████████████| 51kB
  Building wheel for pyLDAvis (setup.py) ...
  Building wheel for pyod (setup.py) ... don
  Building wheel for combo (setup.py) ... do
  Building wheel for suod (setup.py) ... don
  Building wheel for htmlmin (setup.py) ...
```



```
  Building wheel for databricks-cli (setup.p
  Building wheel for prometheus-flask-export
  Building wheel for alembic (setup.py) ...
```

```
#Import python libraries

import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from google.colab import drive
from google.colab import files
import io
import os
from pycaret.classification import *
import statsmodels.api as sm
from statsmodels.formula.api import ols
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt


%matplotlib inline


filepath='/content/drive'
drive.mount(filepath)
filepath+='/My Drive/'
filepath+='Colab Notebooks/Math Clinic/2020fa/' # comment this line out if you're not
filepath+='CAIC/' #comment this line out if you're not Gracie or Prof.
print(*os.listdir(filepath),sep='\n')
```

```
    Drive already mounted at /content/drive; to
    UltimateJupiterNotebookCAIC-1_AF.ipynb
    Loveland Wet Loose CU Den (1).csv
    Red Mountain Pass Wet Loose CU Den.csv
    Red Mountain Pass Wet Slab  CU Den.csv
    Loveland Wet Slab CU Den.csv
    All Wet Avalanches 2014-Present.csv
    Loveland Wet Loose CU Den (1).gsheet
    Definitions.gdoc
    Meeting Agenda Notes.gdoc
    All Wet Avalanches 2014-Present.xlsx
    Metadata for CU denver.txt
    All_Colorado_Weather_Stations.csv
    Wet Avalanche Info.gdoc
    testCAIC.ipynb
```

```
Metadata for CU denver.gdoc
All Wet Avalanches 2014-Present.gsheet
Weather station data
All_Colorado_Weather_Stations.gsheet
lizardhead.txt
Copy of CAIC Rough Code.ipynb
PracticeWithOverlay.ipynb
OriginalWeatherData.xlsx
Time Dependent Correlation.gdoc
CAICdataCount.ipynb
CAIC Rough Code.ipynb
Untitled0.ipynb
Meeting  Notes.gdoc
v2 Rough Code.ipynb
AntepenultimateJupiterNotebookCAIC-1_AF.ipyn
AntepenultimateJupiterNotebookCAIC.ipynb
PenultimateJupiterNotebookCAIC_AF.ipynb
PenultimateJupiterNotebookCAIC.ipynb
Default.docx
UltimateJupiterNotebookCAIC.ipynb
```

```
# Import avalanches dataset

DataAvalanches = pd.read_excel(filepath+'All Wet Avalanches 2014-Present.xlsx')

# Import Weather data for each location

DataWeather = pd.read_excel(filepath+'OriginalWeatherData.xlsx' ,sheet_name=None)
```

```
# Display part of the dataset

DataAvalanches
```


	id	obs_id	avi_hw_op_bc	avi_hw_zo
0	151182	60877	bc	
1	53786	23807	bc	
2	66321	35489	bc	
3	66493	36044	bc	
4	75843	40808	bc	
...	...	...	...	
1494	132467	56615	hw	
1495	149990	60613	hw	

```
# Check dataset structure information

DataAvalanches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1499 entries, 0 to 1498
Data columns (total 62 columns):
 #   Column               Non-Null Count  Dt
---  ------               --------------  --
 0   id                   1499 non-null   in
 1   obs_id               1499 non-null   in
 2   avi_hw_op_bc         1499 non-null   ob
 3   avi_hw_zone_id       1499 non-null   in
 4   avi_path             335 non-null    ob
```

```
 5   avi_op_name          1499 non-null   ob
 6   avi_loc              1 non-null      ob
 7   avi_bc_zone_id       1499 non-null   in
 8   avi_mark             847 non-null    ob
 9   avi_number           1499 non-null   in
 10  avi_type             1499 non-null   ob
 11  avi_aspect           1461 non-null   ob
 12  avi_elev             1468 non-null   ob
 13  avi_rsize            1334 non-null   ob
 14  avi_dsize            1460 non-null   ob
 15  avi_prim_trig        1452 non-null   ob
 16  avi_sec_trig         216 non-null    ob
 17  avi_comments         584 non-null    ob
 18  avi_date             1499 non-null   ob
 19  avi_date_known       1499 non-null   ob
 20  avi_time_known       1499 non-null   ob
 21  avi_area             1021 non-null   ob
 22  avi_angle_avg        163 non-null    ob
 23  avi_angle_max        139 non-null    ob
 24  avi_elevation        431 non-null    ob
 25  avi_elevation_units  1496 non-null   ob
 26  avi_surface          455 non-null    ob
 27  avi_weak_layer       170 non-null    ob
 28  avi_grain_type       165 non-null    ob
 29  avi_crown_avg        156 non-null    ob
 30  avi_crown_max        106 non-null    fl
 31  avi_crown_units      1495 non-null   ob
 32  avi_width_avg        247 non-null    ob
 33  avi_width_max        225 non-null    fl
 34  avi_width_units      1496 non-null   ob
 35  avi_vertical_avg     326 non-null    ob
 36  avi_vertical_max     257 non-null    fl
 37  avi_vertical_units   1497 non-null   ob
 38  avi_terminus         195 non-null    ob
 39  avi_road_status      273 non-null    ob
 40  avi_road_depth       150 non-null    ob
 41  avi_road_length      148 non-null    fl
 42  avi_road_units       1495 non-null   ob
 43  avi_lat              1496 non-null   ob
 44  avi_lon              1496 non-null   fl
 45  id.1                 1497 non-null   ob
 46  obs_id.1             1497 non-null   fl
 47  avi_descr            835 non-null    ob
 48  id.2                 1493 non-null   ob
 49  zone_id              1495 non-null   ob
 50  lat                  1493 non-null   ob
 51  lon                  1494 non-null   ob
 52  utm_zone             1494 non-null   ob
 53  utm_e                1493 non-null   fl
```

```
# Adjust variables types

DataAvalanches['avi_date'] = pd.to_datetime(DataAvalanches['avi_date'], errors='coerce

# Create categorical variable to sign avalanche

DataAvalanches['Avalanche'] = 'Yes'
```



```
DataAvalanches['Avalanche'] = 'Yes'

# Check result

DataAvalanches.head()
```

	id	obs_id	avi_hw_op_bc	avi_hw_zone_
0	151182	60877	bc	
1	53786	23807	bc	
2	66321	35489	bc	
3	66493	36044	bc	
4	75843	40808	bc	

```
DataAvalanches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1499 entries, 0 to 1498
Data columns (total 63 columns):
 #   Column               Non-Null Count  Dt
---  ------               --------------  --
 0   id                   1499 non-null   in
 1   obs_id               1499 non-null   in
 2   avi_hw_op_bc         1499 non-null   ob
 3   avi_hw_zone_id       1499 non-null   in
 4   avi_path             335 non-null    ob
 5   avi_op_name          1499 non-null   ob
 6   avi_loc              1 non-null      ob
 7   avi_bc_zone_id       1499 non-null   in
 8   avi_mark             847 non-null    ob
```

```
 9   avi_number            1499 non-null   in
 10  avi_type              1499 non-null   ob
 11  avi_aspect            1461 non-null   ob
 12  avi_elev              1468 non-null   ob
 13  avi_rsize             1334 non-null   ob
 14  avi_dsize             1460 non-null   ob
 15  avi_prim_trig         1452 non-null   ob
 16  avi_sec_trig          216 non-null    ob
 17  avi_comments          584 non-null    ob
 18  avi_date              1498 non-null   da
 19  avi_date_known        1499 non-null   ob
 20  avi_time_known        1499 non-null   ob
 21  avi_area              1021 non-null   ob
 22  avi_angle_avg         163 non-null    ob
 23  avi_angle_max         139 non-null    ob
 24  avi_elevation         431 non-null    ob
 25  avi_elevation_units   1496 non-null   ob
 26  avi_surface           455 non-null    ob
 27  avi_weak_layer        170 non-null    ob
 28  avi_grain_type        165 non-null    ob
 29  avi_crown_avg         156 non-null    ob
 30  avi_crown_max         106 non-null    fl
 31  avi_crown_units       1495 non-null   ob
 32  avi_width_avg         247 non-null    ob
 33  avi_width_max         225 non-null    fl
 34  avi_width_units       1496 non-null   ob
 35  avi_vertical_avg      326 non-null    ob
 36  avi_vertical_max      257 non-null    fl
 37  avi_vertical_units    1497 non-null   ob
 38  avi_terminus          195 non-null    ob
 39  avi_road_status       273 non-null    ob
 40  avi_road_depth        150 non-null    ob
 41  avi_road_length       148 non-null    fl
 42  avi_road_units        1495 non-null   ob
 43  avi_lat               1496 non-null   ob
 44  avi_lon               1496 non-null   fl
 45  id.1                  1497 non-null   ob
 46  obs_id.1              1497 non-null   fl
 47  avi_descr             835 non-null    ob
 48  id.2                  1493 non-null   ob
 49  zone_id               1495 non-null   ob
 50  lat                   1493 non-null   ob
 51  lon                   1494 non-null   ob
 52  utm_zone              1494 non-null   ob
 53  utm e                 1493 non-null   fl
```

With that part done, it is time to move for the pre-processing of the weather dataset. The first step is to combine the data from all the different tabs into a single dataset.

```
# Combine weather data from different stations into a single dataframe

CombinedDataWeather = pd.concat(DataWeather, keys=DataWeather.keys())
```



```
CombinedDataWeather.reset_index(inplace=True)

CombinedDataWeather.drop('level_1', axis=1, inplace=True)

CombinedDataWeather.rename(columns={'level_0':'Location'}, inplace=True)

CombinedDataWeather
```

	Location	Date	Snow Water Equivalent (in) Start of Day Values	Precipit: Accumul: (in) Sta: Day V:
0	Berthoud Pass	1978-10-01	0.0	
1	Berthoud Pass	1978-10-02	0.0	
2	Berthoud Pass	1978-10-03	0.0	
3	Berthoud Pass	1978-10-04	0.0	
4	Berthoud Pass	1978-10-05	0.0	
...	...	...	...	
174871	Wolf Creek pass Snotel	2020-10-03	0.0	
174872	Wolf Creek pass Snotel	2020-10-04	0.0	
174873	Wolf Creek pass Snotel	2020-10-05	0.0	
174874	Wolf Creek pass Snotel	2020-10-06	0.0	
174875	Wolf Creek pass Snotel	2020-10-07	0.0	

174876 rows × 8 columns

We can see now that all the stations have their data properly labeled into a unique dataset.

```
# Get information about the dataframe

CombinedDataWeather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 174876 entries, 0 to 174875
Data columns (total 8 columns):
 #   Column
---  ------
 0   Location
 1   Date
 2   Snow Water Equivalent (in) Start of Day
 3   Precipitation Accumulation (in) Start o
 4   Air Temperature Maximum (degF)
 5   Air Temperature Minimum (degF)
 6   Air Temperature Average (degF)
 7   Precipitation Increment (in)
dtypes: float64(6), object(2)
memory usage: 10.7+ MB
```

As it was done with the avalanches data, we also need to change the date variable to datetime format.

```
# Adjust variables types

CombinedDataWeather['Date'] = pd.to_datetime(CombinedDataWeather['Date'])

CombinedDataWeather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 174876 entries, 0 to 174875
Data columns (total 8 columns):
 #   Column
---  ------
 0   Location
 1   Date
 2   Snow Water Equivalent (in) Start of Day
 3   Precipitation Accumulation (in) Start o
 4   Air Temperature Maximum (degF)
 5   Air Temperature Minimum (degF)
 6   Air Temperature Average (degF)
 7   Precipitation Increment (in)
dtypes: datetime64[ns](1), float64(6), objec
memory usage: 10.7+ MB
```

```
CombinedDataWeather
```


	Location	Date	Snow Water Equivalent (in) Start of Day Values	Precipit Accumul (in) Sta Day V
0	Berthoud Pass	1978-10-01	0.0	
1	Berthoud Pass	1978-10-02	0.0	
2	Berthoud Pass	1978-10-03	0.0	
3	Berthoud Pass	1978-10-04	0.0	
4	Berthoud Pass	1978-10-05	0.0	
...	...	...	...	
174871	Wolf Creek pass Snotel	2020-10-03	0.0	
174872	Wolf Creek pass Snotel	2020-10-04	0.0	
174873	Wolf Creek pass Snotel	2020-10-05	0.0	
174874	Wolf Creek pass Snotel	2020-10-06	0.0	
174875	Wolf Creek pass Snotel	2020-10-07	0.0	

174876 rows × 8 columns

In this notebook we will average the weather data by date.

```
AveragedDataWeather = CombinedDataWeather.groupby(['Date']).mean().reset_index()

AveragedDataWeather
```

	Date	Snow Water Equivalent (in) Start of Day Values	Precipitation Accumulation (in) Start of Day Values	Temp
0	1978-10-01	0.000000	0.000000	
1	1978-10-02	0.000000	0.000000	
2	1978-10-03	0.000000	0.000000	
3	1978-10-04	0.000000	0.000000	
4	1978-10-05	0.000000	0.100000	
...	...	...	...	
15343	2020-10-03	0.023077	0.008333	5
15344	2020-10-04	0.030769	0.007692	6
15345	2020-10-05	0.030769	0.016667	6
15346	2020-10-06	0.107692	0.008333	6
15347	2020-10-07	0.092308	0.018182	

15348 rows × 7 columns

The grouping was successfully made so now we will merge that information in the avalanches dataset by using Date as the primary key to connect the datasets.

```
# Merge Avalanches DataFrame with Weather DataFrame

# FullData = pd.merge(CombinedDataWeather, DataAvalanches[['avi_mark', 'avi_date', 'Av

FullData = pd.merge(DataAvalanches[['avi_mark', 'avi_date', 'Avalanche', 'avi_number',

FullData.sort_values(by='Date', inplace=True)

FullData
```


	avi_mark	avi_date	Avalanche	avi_
1499	NaN	NaT	NaN	
1500	NaN	NaT	NaN	
1501	NaN	NaT	NaN	
1502	NaN	NaT	NaN	
1503	NaN	NaT	NaN	
...	...	...	...	
16369	NaN	NaT	NaN	
16370	NaN	NaT	NaN	
985	Hoosier Pass	NaT	Yes	
1001	Independence Pass-East side	1970-01-01	Yes	
1002	NaN	1970-01-01	Yes	

16371 rows × 12 columns

We can see that the new dataset appended the avalanches columns that we selected into the dataset of weather measurements. Since all occurrences of avalanches are identified with an `Yes` in the column `Avalanche` we can fill the rows with missing data with a `No`, to identify that there was not an avalanche in that date/location. Also we fill missing values with zero in the column of avalanche numbers, for the same reason.

```
# Fill null values of avalanche numbers and categorical

FullData['Avalanche'].fillna('No', inplace=True)
FullData['avi_number'].fillna(0, inplace=True)
```

```
# Check information about merged dataset

FullData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16371 entries, 1499 to 1002
Data columns (total 12 columns):
 #   Column
---  ------
 0   avi_mark
 1   avi_date
 2   Avalanche
 3   avi_number
 4   avi_type
 5   Date
 6   Snow Water Equivalent (in) Start of Day
 7   Precipitation Accumulation (in) Start o
 8   Air Temperature Maximum (degF)
 9   Air Temperature Minimum (degF)
 10  Air Temperature Average (degF)
 11  Precipitation Increment (in)
dtypes: datetime64[ns](2), float64(7), objec
memory usage: 1.6+ MB
```

In the table above we can see that 1499 avalanches matched the existing data on weather for the locations that were provided.

There are three avalanches entries that didn't have a date assigned for them, so we will delete those entries to avoid issues in analyzing the data.

```
FullData.dropna(subset=['Date'], inplace=True)

FullData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16368 entries, 1499 to 16370
Data columns (total 12 columns):
 #   Column
---  ------
 0   avi_mark
 1   avi_date
 2   Avalanche
 3   avi_number
 4   avi_type
 5   Date
 6   Snow Water Equivalent (in) Start of Day
 7   Precipitation Accumulation (in) Start o
 8   Air Temperature Maximum (degF)
 9   Air Temperature Minimum (degF)
 10  Air Temperature Average (degF)
```



```
 11  Precipitation Increment (in)
dtypes: datetime64[ns](2), float64(7), objec
memory usage: 1.6+ MB
```

Since the dataset is very unbalanced we can remove all data before 2014, since we don't have information about avalanches prior to that.

```
FullData = FullData[FullData['Date'].dt.year >= 2014]

FullData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3492 entries, 14375 to 16370
Data columns (total 12 columns):
 #   Column
---  ------
 0   avi_mark
 1   avi_date
 2   Avalanche
 3   avi_number
 4   avi_type
 5   Date
 6   Snow Water Equivalent (in) Start of Day
 7   Precipitation Accumulation (in) Start o
 8   Air Temperature Maximum (degF)
 9   Air Temperature Minimum (degF)
 10  Air Temperature Average (degF)
 11  Precipitation Increment (in)
dtypes: datetime64[ns](2), float64(7), objec
memory usage: 354.7+ KB
```

With that the pre-processing section is finished and there is a unique dataframe which we can analyze to check if the existing weather data shows correlation with avalanche occurrences.

```
FullData.head()
```

	avi_mark	avi_date	Avalanche	avi_nur
14375	NaN	NaT	No	
14376	NaN	NaT	No	

Methods for Analyzing Data

Practical Analysis

14379	NaN	NaT	No

For Practical Analysis, we look into Time Series Plots, Distribution Plots, and Distribution Plots by Avalanche Type to identify a relationship between avalanche occurrences and the weather. We look at descriptive statistics information about the data, compared time with different weather variables, and weather variables with the avalanche count. Then we did something similar, but with types of avalanches.

Linear Regression

Linear regression was something that we attempted to use in order to determine if there was a direct correlation between each weather variable and the occurence of avalanches. Linear regression requires a set of independent variables, x, and a dependent variable, y, to determine if there is a linear relationship between them. If there is a linear relationship, it can be used to predict the future occurence of the dependent variable. In this scenario, the variable 'avi_number' is the dependent variable that we are attempting to predict, and the weather variables, 'Snow Water Equivalent (in) Start of Day Values', 'Precipitation Accumulation (in) Start of Day Values', 'Air Temperature Average (degF)', 'Air

Temperature Maximum (degF)', 'Air Temperature Minimum (degF)', and 'Precipitation Increment' were each used as the various independent variables.

We used the NumPy and skicit-learn packages in order to implement linear regression. We manipulated the 'avi_number' data in multiple ways to search for an R2 score that would be significant enough for us to pursue linear regression, and based off our findings, decided it was not an appropriate method for calculating correlation.

Logistic Regression

Logistic regression was a model that we wanted to look into further because it utilizes avalanche count as a binary variable (0 for no avalanche/1 for an avalanche occurence). This is ideal because we are trying to predict the occurence of an avalanche under different weather variables.

We used the scikit-learn package to implement logistic regression. The variable 'AvCount' was created as our binary dependent variable because we are trying to predict it. The highest accuracy model was generated when we used all of the available weather variables, 'Snow Water Equivalent (in) Start of Day Values', 'Precipitation Accumulation (in) Start of Day Values', 'Air Temperature Average (degF)', 'Air Temperature Maximum (degF)', 'Air Temperature Minimum (degF)', and 'Precipitation Increment'. More information about the logistic regression model that we implemented is available here.

This method of logistic regression differs from the logistic regression model in the machine learning portion of our results. That model is generated using PyCaret which a machine learning library and doesn't require as much code to run and concludes the most important predictors without intervention from the coder.

Machine Learning

We used PyCaret which is an open-source, low-code machine learning library in Python. It allows us to check the accuracy of multiple methods at ones. The methods included are both categorical and regression. From there we took the columns Avalanche, Snow Water Equivalent, Precipitation Accumilation, Air Temperature Maximum, Ait Temperature Minimum, Air Temperature Average, and Precipitation Increment. Then we set up our target to be Avalanche where No became 0 and Yes became 1. It then shows you defaults like the [fold number], which is the original sample is randomly partitioned into k equal size subsamples for training and testing. Its default for testing and training is 70:30. Once that step is done, you compare models and it shows you the accuracy from highest to lowest. We then took three models from different accuracies, but still high, they were Cat Boost, extreme Gradient Boost, and Logistic Regression. We then tuned each model, evaluated it using a [Confusion Matrix], and included a portion where it plots a Feature Importance Plot and a heat map.

Results and Discussion

Practical Analysis

In the practical analysis we will use statistical and graphical techniques to see if we can identify a relationship between avalanche occurrences and the weather.

First we start checking descriptive statistics information about the data, to see if there is anything strange and get a sense of the distribution of the data.

```
# Descriptive Statistics of numerical variables
```



```
FullData.describe().transpose()
```

	count	mean	std	
avi_number	3492.0	0.995132	2.836968	0.0(
Snow Water Equivalent (in) Start of Day Values	3492.0	10.647652	8.890274	0.0(
Precipitation Accumulation (in) Start of Day Values	3492.0	19.511723	9.630722	0.0(
Air Temperature				

Despite the wide temperature distribution existing in the data, there are no signs of anomalies in the measurement that should be removed from the dataset. Let's also check some information about the categorical variables below.

```
# Descriptive Statistics of categorical variables

FullData.describe(include='O').transpose()
```

	count	unique	top	freq
avi_mark	845	94	-1	124
Avalanche	3492	2	No	1996
avi_type	1496	2	WL	1009

We can see that the Wet Loose avalanches represent more than two thirds of all avalanches between 2014 and today. Also the dataset seems well balanced between occurences of avalanches.

Let's check some time series plots about the weather and avalanches.

Time Series Plots

```
# Group data by date to improve plot quality

GroupedData = FullData.groupby('Date').agg({'Snow Water Equivalent (in) Start of Day V
                                             'Precipitation Accumulation (in) Start of
                                             'Air Temperature Average (degF)':'mean',
                                             'Air Temperature Maximum (degF)':'mean',
                                             'Air Temperature Minimum (degF)':'mean',
                                             'Precipitation Increment (in)':'mean',
                                             'avi_number':'sum'}).reset_index()

GroupedData.head()
```

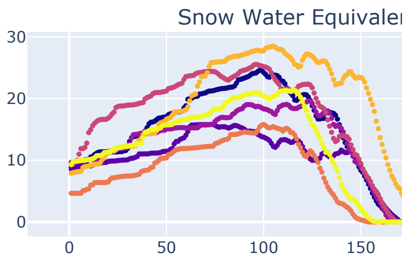
	Date	Snow Water Equivalent (in) Start of Day Values	Precipitation Accumulation (in) Start of Day Values	Temperat Aver (de
0	2014-01-01	8.923077	9.615385	19.461
1	2014-01-02	9.146154	9.823077	21.615
2	2014-01-03	9.176923	9.869231	25.692
3	2014-01-04	9.238462	9.892308	12.307
4	2014-01-05	9.500000	10.053846	0.384

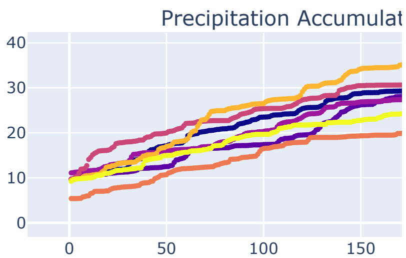
```
d=GroupedData['Date']
GroupedData.insert(1,'Year',d.dt.year)
GroupedData.insert(2,'Day of Year',d.dt.dayofyear)
del d
GroupedData.head()
```

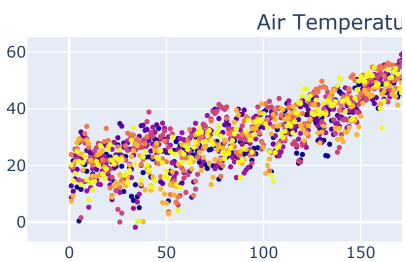

Day	Snow Water Equivalent	Precipitati

```
# Create time series subplots
xKey='Date'        # for all years end-to-end along axis (not tested)
xKey='Day of Year' # for all years superposed
yKeys=('Snow Water Equivalent (in) Start of Day Values',
       'Precipitation Accumulation (in) Start of Day Values',
       'Air Temperature Average (degF)',
       'Air Temperature Minimum (degF)',
       'Air Temperature Maximum (degF)',
       'Precipitation Increment (in)')
fig = make_subplots(rows=6,
                    cols=1,
                    subplot_titles=yKeys)
for row in range(1,len(yKeys)+1) :
  fig.add_trace(
      go.Scatter(x=GroupedData[xKey],
                 y=GroupedData[yKeys[row - 1]],
                 marker=dict(color=GroupedData['Year'],
                             showscale=True,
                             size=4),
                 mode='markers'),
      row=row,
      col=1)
fig.update_layout(height=1800,
                  width=800,
                  showlegend=False,
                  title_text="Weather Data by " + xKey)
fig.show()
```

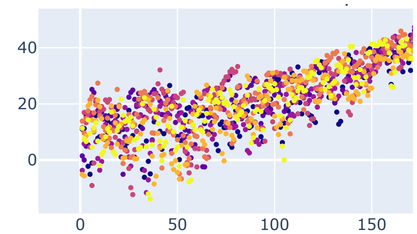
Weather Data by Day of Year

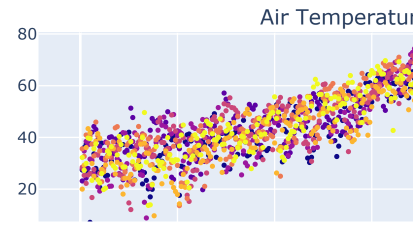






Air Temperatu

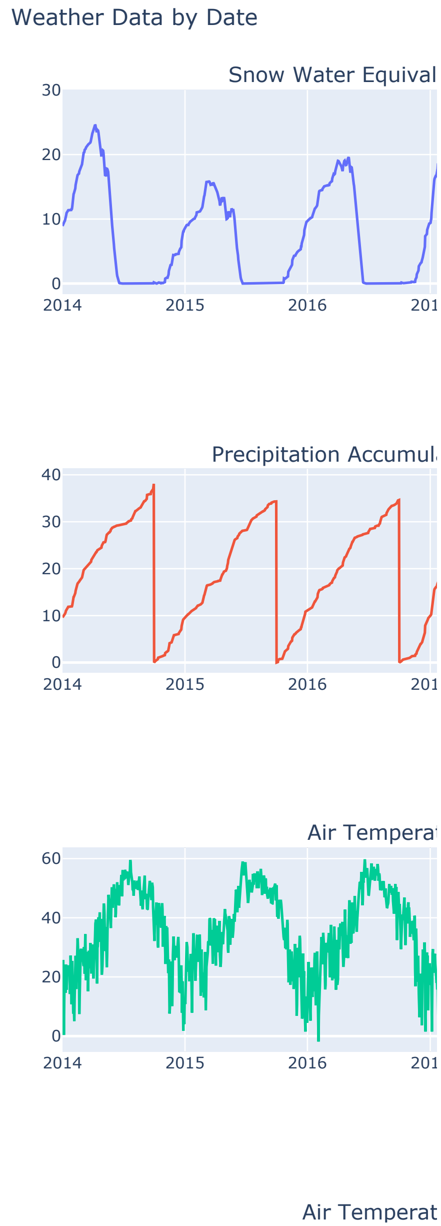


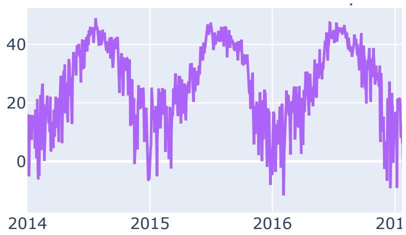


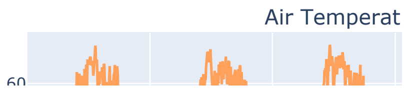

```
# Create time series subplots

fig = make_subplots(rows=6,
                    cols=1,
                    subplot_titles=('Snow Water Equivalent (in) Start of Day Values',
                                    'Precipitation Accumulation (in) Start of Day Valu
                                    'Air Temperature Average (degF)',
                                    'Air Temperature Minimum (degF)',
                                    'Air Temperature Maximum (degF)',
                                    'Precipitation Increment (in)'))

fig.add_trace(
    go.Scatter(x=GroupedData['Date'],
               y=GroupedData['Snow Water Equivalent (in) Start of Day Values']),
               row=1,
               col=1)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'],
               y=GroupedData['Precipitation Accumulation (in) Start of Day Values']),
               row=2,
               col=1)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'],
               y=GroupedData['Air Temperature Average (degF)']),
               row=3,
               col=1)
```

```
fig.add_trace(
    go.Scatter(x=GroupedData['Date'],
               y=GroupedData['Air Temperature Minimum (degF)']),
               row=4,
               col=1)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'],
               y=GroupedData['Air Temperature Maximum (degF)']),
               row=5,
               col=1)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'],
               y=GroupedData['Precipitation Increment (in)']),
               row=6,
               col=1)

fig.update_layout(height=1800,
                  width=800,
                  showlegend=False,
                  title_text="Weather Data by Date")
fig.show()
```

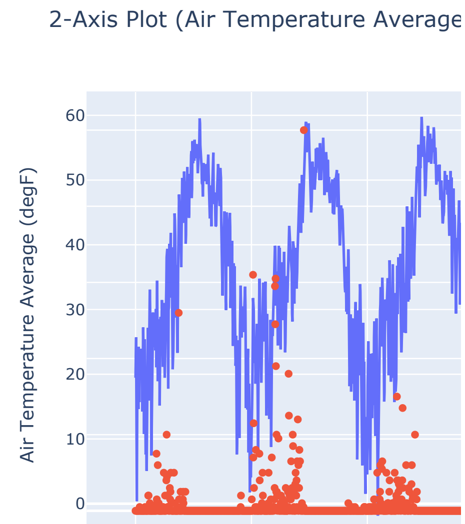

1/11/2021 UltimateJupiterNotebookCAIC-1_AF.ipynb - Colaboratory








```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Air Temperature Average (degF)'],
    secondary_y=False,
)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'], mode='markers', nar
    secondary_y=True,
)

# Add figure title
fig.update_layout(
    title_text="2-Axis Plot (Air Temperature Average)"
)

# Set x-axis title
fig.update_xaxes(title_text="Date")

# Set y-axes titles
fig.update_yaxes(title_text="Air Temperature Average (degF)", secondary_y=False)
fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True)

fig.show()
```

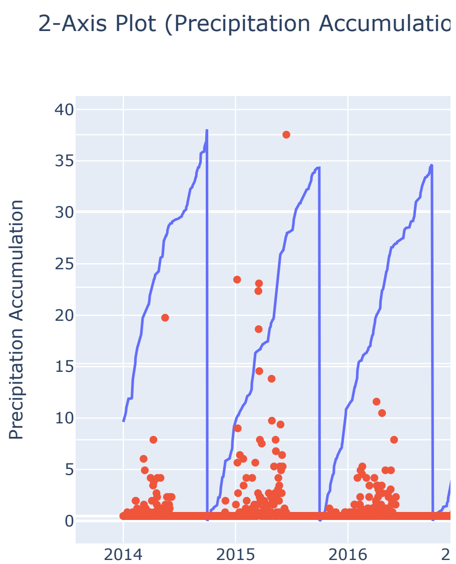



By looking at the time series data we can identify a pattern. It seems that most avalanches happen when temperatures are rising. This makes a lot of sense because it is when there is most snow in the mountains, and with the increase in temperature this snow starts to melt. Once it is all melted there is not enough snow to generate an avalanche.

By having those initial insights we can look more in details to each one of the weather variables measured, and see if it is correlated with avalanche occurrences. To do that we can create a distribution plot and split the data from the days when there was an avalanche and when there wasn't, to compare the distributions.

```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Precipitation Accumulation (in) s
    secondary_y=False,
)
```

```
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'],mode= 'markers', nan
    secondary_y=True,
)

# Add figure title
fig.update_layout(
    title_text="2-Axis Plot (Precipitation Accumulation)"
)

# Set x-axis title
fig.update_xaxes(title_text="Date")

# Set y-axes titles
fig.update_yaxes(title_text="Precipitation Accumulation", secondary_y=False)
fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True)

fig.show()
```




```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])
```



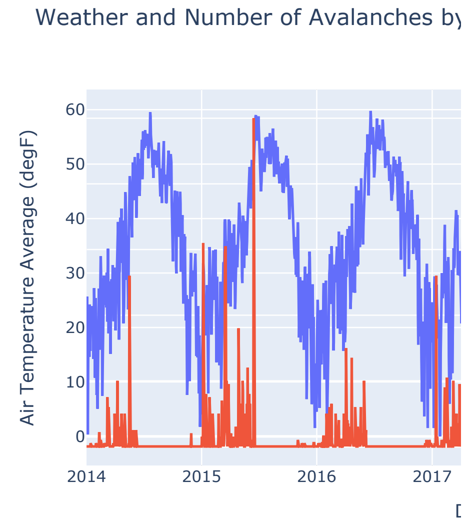
```
# Add traces
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Snow Water Equivalent (in) Start
    secondary_y=False,
)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'],mode= 'markers', nan
    secondary_y=True,
)

# Add figure title
fig.update_layout(
    title_text="2-Axis Plot (Snow Water Equivalent (in))"
)

# Set x-axis title
fig.update_xaxes(title_text="Date")

# Set y-axes titles
fig.update_yaxes(title_text="Snow Water Equivalent (in)", secondary_y=False)
fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True)

fig.show()
```

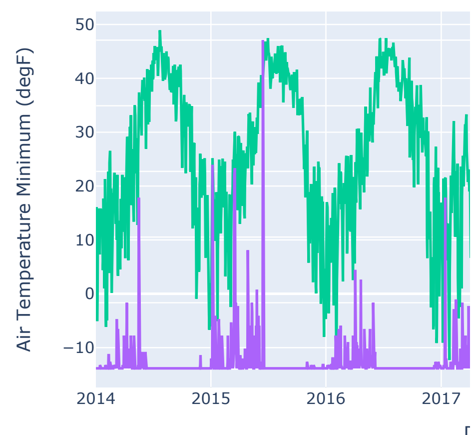
```
# Create subplots of dual axis plots

# Create figure with secondary y-axis
fig = make_subplots(rows=6,
                    cols=1,
                    specs=[[{"secondary_y": True}],
                           [{"secondary_y": True}],
                           [{"secondary_y": True}],
                           [{"secondary_y": True}],
                           [{"secondary_y": True}],
                           [{"secondary_y": True}]])

# Add traces for plot (1,1)
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Air Temperature Average (degF)'],
    secondary_y=False,
    row=1,
    col=1
)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'], name="Number of Ava
    secondary_y=True,
    row=1,
    col=1
)

# Add traces for plot (2,1)
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Air Temperature Minimum (degF)'],
    secondary_y=False,
    row=2,
    col=1,
)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'], name="Number of Ava
    secondary_y=True,
    row=2,
    col=1
)

# Add traces for plot (3,1)
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Air Temperature Maximum (degF)'],
    secondary_y=False,
    row=3,
    col=1
)
```

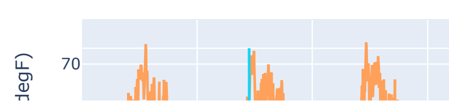


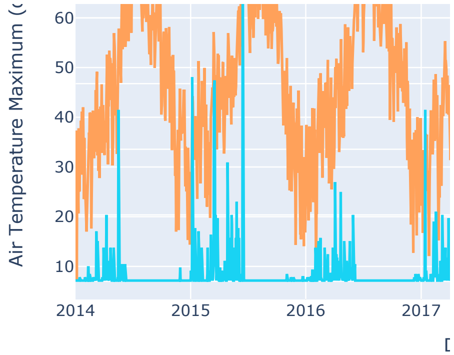
```
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'], name="Number of Ava
    secondary_y=True,
    row=3,
    col=1
)

# Add traces for plot (4,1)
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Precipitation Increment (in)'], r
    secondary_y=False,
    row=4,
    col=1
)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'], name="Number of Ava
    secondary_y=True,
    row=4,
    col=1
)

# Add traces for plot (5,1)
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Precipitation Accumulation (in) S
    secondary_y=False,
    row=5,
    col=1
)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'], name="Number of Ava
    secondary_y=True,
    row=5,
    col=1
)

# Add traces for plot (6,1)
fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['Snow Water Equivalent (in) Start
    secondary_y=False,
    row=6,
    col=1
)

fig.add_trace(
    go.Scatter(x=GroupedData['Date'], y=GroupedData['avi_number'], name="Number of Ava
    secondary_y=True,
    row=6,
    col=1
)
```

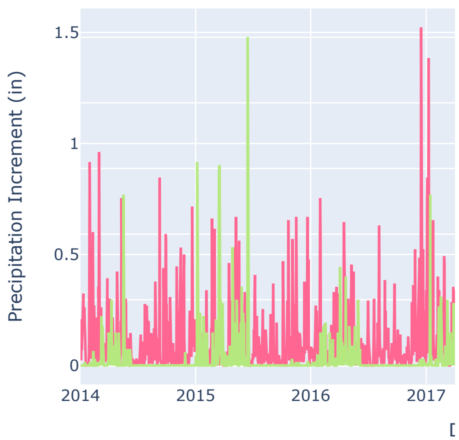
```
    ,

    # Add figure title
    fig.update_layout(height=2500,
                      width=800,
                      showlegend=False,
                      title_text="Weather and Number of Avalanches by Date")

    # Set x-axis title
    fig.update_xaxes(title_text="Date")

    # Set y-axes titles
    fig.update_yaxes(title_text="Air Temperature Average (degF)", secondary_y=False, row=1
    fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True, row=1, col=1)

    fig.update_yaxes(title_text="Air Temperature Minimum (degF)", secondary_y=False, row=2
    fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True, row=2, col=1)

    fig.update_yaxes(title_text="Air Temperature Maximum (degF)", secondary_y=False, row=3
    fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True, row=3, col=1)

    fig.update_yaxes(title_text="Precipitation Increment (in)", secondary_y=False, row=4,
    fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True, row=4, col=1)

    fig.update_yaxes(title_text='Precipitation Accumulation (in) Start of Day Values', sec
    fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True, row=5, col=1)

    fig.update_yaxes(title_text='Snow Water Equivalent (in) Start of Day Values', seconda
    fig.update_yaxes(title_text="Number of Avalanches", secondary_y=True, row=6, col=1)

    fig.show()
```

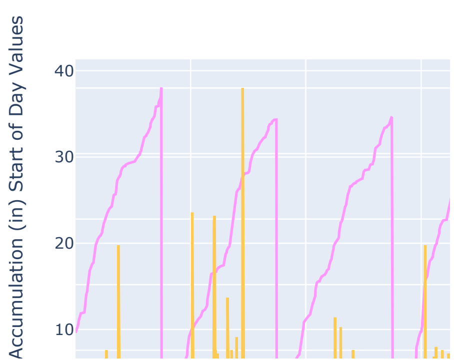



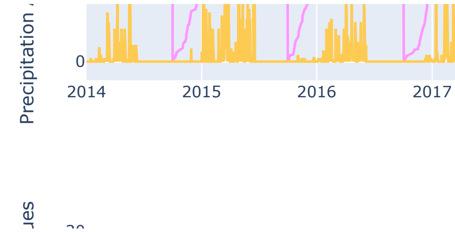






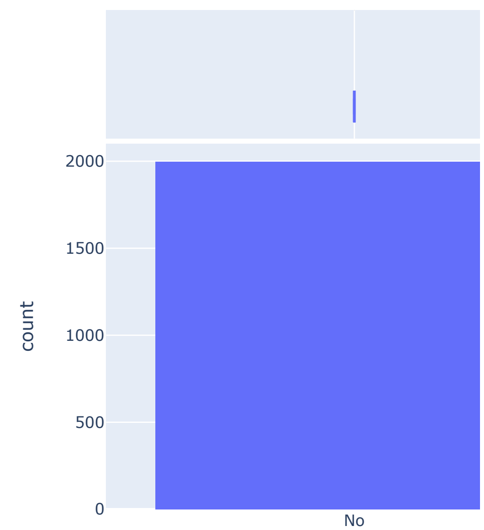






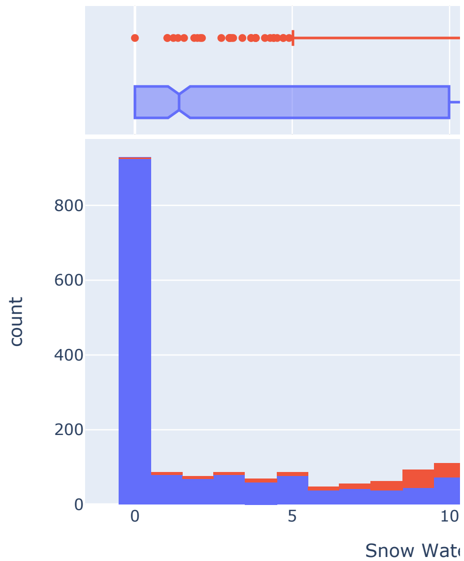
Distribution Plots

```
# Create histogram / distribution visualization for the different variables

px.histogram(FullData,
             x='Avalanche',
             color='Avalanche',
             marginal='box')
```




```
# Create histogram / distribution visualization for the different variables
```

```
px.histogram(FullData,
             x='Snow Water Equivalent (in) Start of Day Values',
             color='Avalanche',
             marginal='box')
```



We can clearly see that there is a significant difference between the results of days when there was an avalanche and of then there was none and that it should be detected by a prediction model.

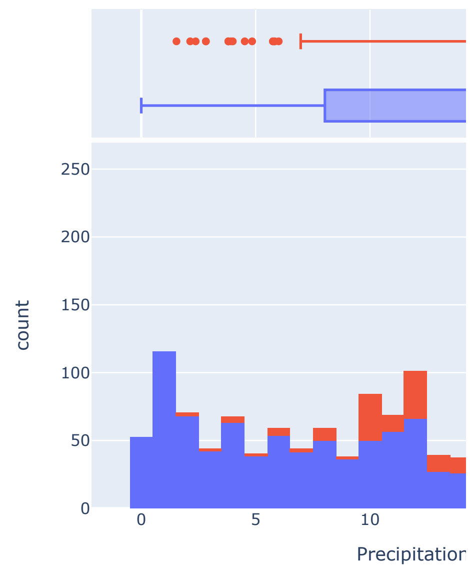
Now adding some critical thinking on those results, it is important to remember that the data that is being used is an average of all locations, and not the measurement in the specific location where the avalanche happened. So it is better to read the results as the relations between weather data and any incidence of avalanche.

Also we saw that spring months are definitely the ones with most avalanche (probably because the snow is

melting). If there is a relation between the season and the snow water equivalent results, it is possible that the results that we are seeing are not directly correlated, but are correlated by a third variable. But in this specific case there is a natural theorical explanation that when there is more snow, there is a higher instability in the mountains that can lead to avalanches.

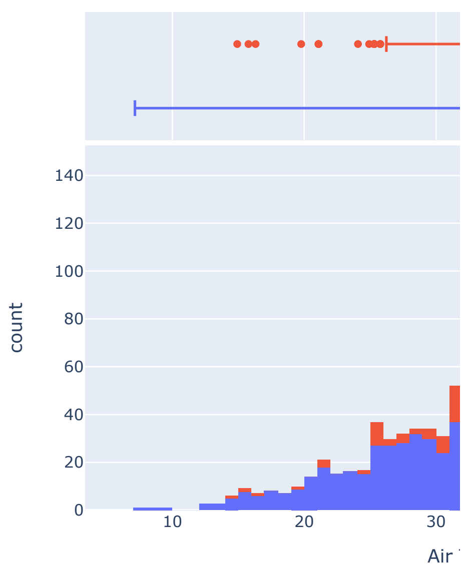
More details on Snow Water Equivalent: [Link]

```
# Create histogram / distribution visualization for the different variables

px.histogram(FullData,
             x='Precipitation Accumulation (in) Start of Day Values',
             color='Avalanche',
             marginal='box')
```

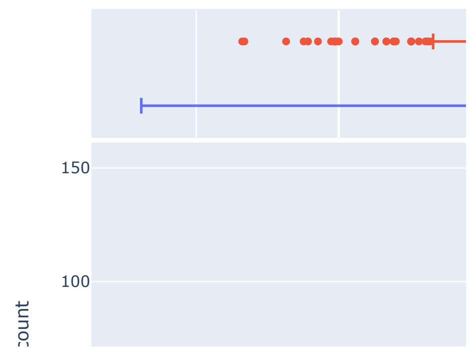



```
# Create histogram / distribution visualization for the different variables

px.histogram(FullData,
```

```
             x='Air Temperature Maximum (degF)',
             color='Avalanche',
             marginal='box')
```

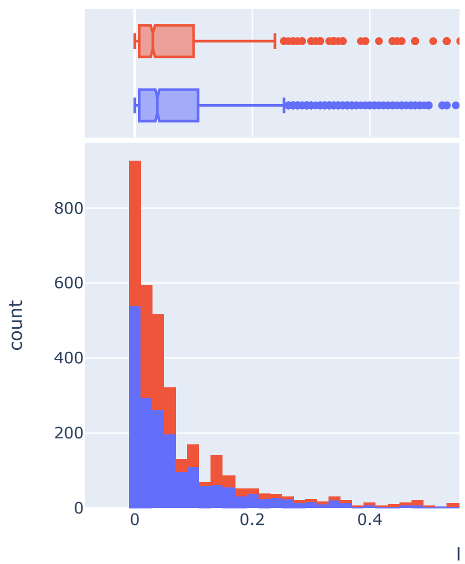



```
# Create histogram / distribution visualization for the different variables

px.histogram(FullData,
             x='Air Temperature Minimum (degF)',
             color='Avalanche',
             marginal='box')
```


```
# Create histogram / distribution visualization for the different variables

px.histogram(FullData,
             x='Air Temperature Average (degF)',
             color='Avalanche',
             marginal='box')
```

```
# Create histogram / distribution visualization for the different variables

px.histogram(FullData,
             x='Precipitation Increment (in)',
             color='Avalanche',
             marginal='box')
```




```
# Create histogram / distribution visualization for the different variables


#variable = 'Air Temperature Average (degF)' #@param ['Location', 'Date', 'Snow Water

#px.histogram(FullData,
 #           color='Avalanche',
  #           marginal='box')
```

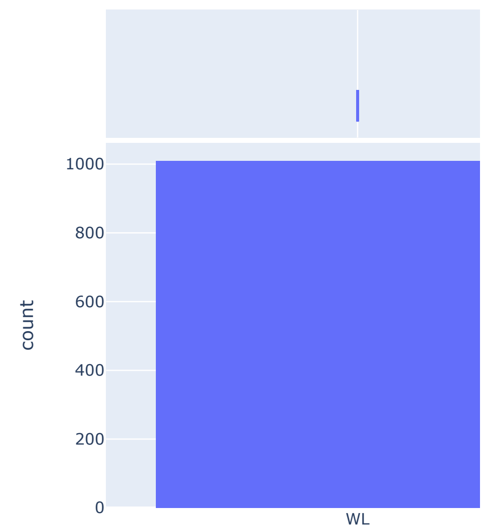
We can also look for patterns related to the different types of avalanches, Wet Slab and Wet Loose, using the

same method.

Distribution Plots by Avalanche Type

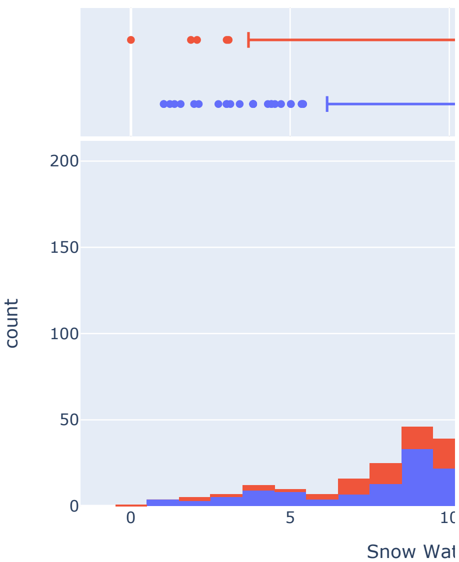
```
# Filter Data to check results in which there was an avalanche

AvalanchesOnly = FullData[FullData['Avalanche'] == 'Yes']
```

```
# Create histogram / distribution visualization for the different variables

px.histogram(AvalanchesOnly,
             x='avi_type',
             color='avi_type',
             marginal='box')
```




```
# Create histogram / distribution visualization for the different variables

px.histogram(AvalanchesOnly,
             x='Snow Water Equivalent (in) Start of Day Values',
```

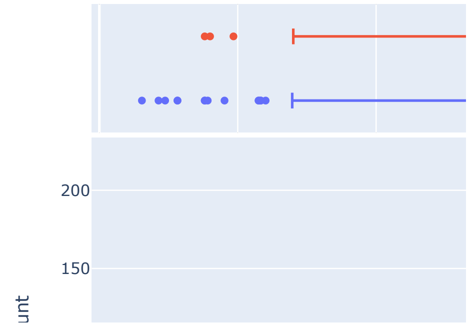
```
             color='avi_type',
             marginal='box')
```



Aimé Fournier
Dec 12, 2020
Resolve

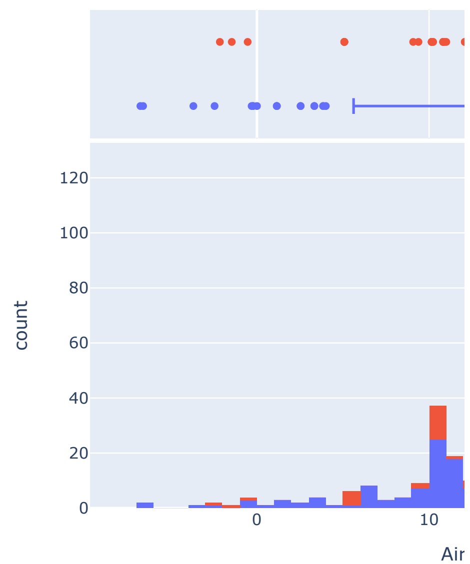
", numeric_features=['Year']" because categorical year messes things up

```
# Create histogram / distribution visualization for the different variables

px.histogram(AvalanchesOnly,
             x='Precipitation Accumulation (in) Start of Day Values',
             color='avi_type',
             marginal='box')
```

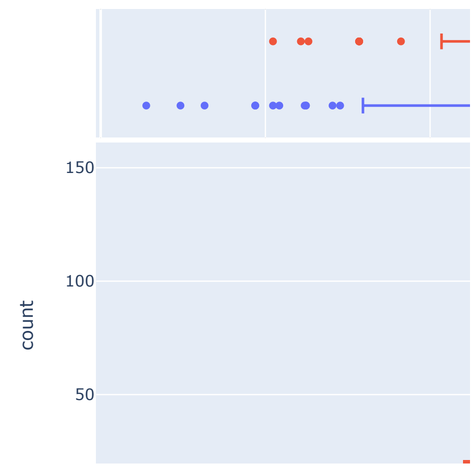




```
# Create histogram / distribution visualization for the different variables

px.histogram(AvalanchesOnly,
             x='Air Temperature Maximum (degF)',
             color='avi_type',
             marginal='box')
```

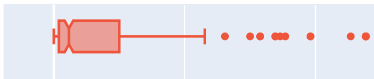
```
# Create histogram / distribution visualization for the different variables

px.histogram(AvalanchesOnly,
             x='Air Temperature Minimum (degF)',
             color='avi_type',
             marginal='box')
```




```
# Create histogram / distribution visualization for the different variables

px.histogram(AvalanchesOnly,
             x='Air Temperature Average (degF)',
             color='avi_type',
             marginal='box')
```

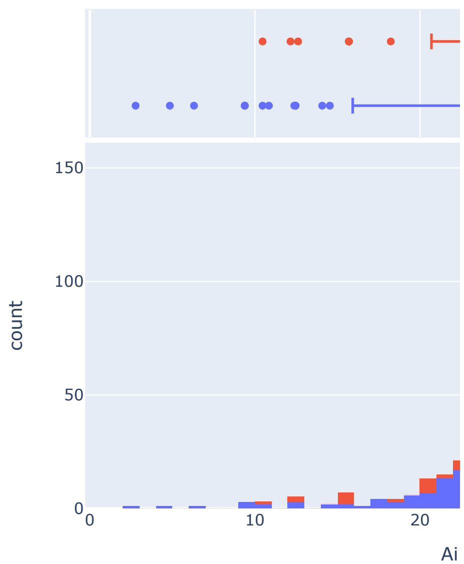




```
# Create histogram / distribution visualization for the different variables

px.histogram(AvalanchesOnly,
             x='Precipitation Increment (in)',
             color='avi_type',
             marginal='box')
```



variable: Air Tem

```
# Create histogram / distribution visualization for the c

variable = 'Air Temperature Average (degF)' #@param ['Loc

fig = px.histogram(AvalanchesOnly,
             x=variable,
             color='avi_type',
             marginal='box')

fig.show()
```



Here there weren't many big differences in the measured variables' distribution. The one specific observation is that it seems that Wet Slab avalanches has a slightly

higher median for the Minimum, Maximum and Average temperatures than the Wet Loose ones. It is possible to test that difference in the medians or means to understand if that is statistically significant.

Linear Regression

The following code shows our implementation of linear regression and why it did not work. The 'avi_number' data was our biggest obstacle because of how sparse the data was. Multiple attempts were made in order to account for this sparseness; however, none of them generated on R^2 score that we felt was satisfactory enough to continue working on linear regression.

```
#Creating a copy of the data without null values
NoNullGroupedData = GroupedData.dropna()
```

This is the first attempt to test for a correlation between the weather variable, Snow Water Equivalent, and avalanche count. This attempt uses the data as is, with no adjustments made to avalanche count.

```
#creating our x and y variables
x1 = NoNullGroupedData.iloc[:,[3]].values.reshape(-1, 1) #Snow Water Equivalent
y1 = NoNullGroupedData.iloc[:,[9]].values.reshape(-1, 1) #avi_number

# Model initialization
regression_model_one = LinearRegression()
# Fit the data(train the model)
regression_model_one.fit(x1, y1)
#Calculate and print R2 score
r_sq_one = regression_model_one.score(x1,y1)
print('R2 score: ', r_sq_one)
```

```
R2 score:  0.07665051806996348
```

Ideally, the R2 score should be as close to 1 as possible. The low R2 score shows that the data, in this form, does not capture the relationship between Snow Water Equivalence and the occurence of avalanches

This is the second attempt to test for a correlation between the weather variable, Snow Water Equivalent, and avalanche count. In this attempt, we made avalanche count a binary variable (0 for no/1 for yes).

```
#making new column for avalanche data as a binary count
NoNullGroupedData.loc[NoNullGroupedData['avi_number'] == 0, 'AvCount'] = 0
NoNullGroupedData.loc[NoNullGroupedData['avi_number'] >= 1, 'AvCount'] = 1
```

```
#creating our x and y variables
x2 = NoNullGroupedData.iloc[:,[3]].values.reshape(-1, 1) #Snow Water Equivalent
y2 = NoNullGroupedData.iloc[:,[10]].values.reshape(-1, 1) #AvCount

# Model initialization
regression_model_two = LinearRegression()
# Fit the data(train the model)
regression_model_two.fit(x2, y2)
#Calculate and print R2 score
r_sq_two = regression_model_two.score(x2,y2)
print('R2 score: ', r_sq_two)
```

```
R2 score:  0.2755908525407361
```

The R2 score generated is helpful in understanding there is a relationship between 'Snow Water Equivalent' and the occurence of an avalalanche. However, it is not significant enough for us to pursue it further.

This is the third attempt to test for a correlation between the weather variable, Snow Water Equivalent, and avalanche count. In this attempt, we removed all of the months after May because the majority of avalanches occur at this time. We also could have included the months November and December; however, the smaller sample worked just as well.

```
#Only including data between January and May
NoNullGroupedData = NoNullGroupedData[NoNullGroupedData['Date'].dt.month <= 5]
```



```
#creating our x and y variables
x3 = NoNullGroupedData.iloc[:,[3]].values.reshape(-1, 1) #Snow Water Equivalent
y3 = NoNullGroupedData.iloc[:,[9]].values.reshape(-1, 1) #avi_number

# Model initialization
regression_model_three = LinearRegression()
# Fit the data(train the model)
regression_model_three.fit(x3, y3)
#Calculate and print R2 score
r_sq_three = regression_model_three.score(x3,y3)
print('R2 score: ', r_sq_three)
```

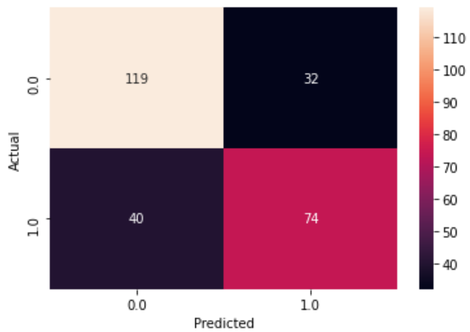
```
R2 score:  0.020505479899889267
```

The low R2 score shows that the data, in this form, does not capture the relationship between Snow Water Equivalence and the occurence of avalanches

Logistic Regression

```
#split dataset in features and target variable
X = NoNullGroupedData[['Day of Year',
                       'Snow Water Equivalent (in) Start of Day Values',
                       'Precipitation Accumulation (in) Start of Day Values',
                       'Precipitation Increment (in)',
                       'Air Temperature Average (degF)',
                       'Air Temperature Maximum (degF)',
                       'Air Temperature Minimum (degF)']]
y = NoNullGroupedData['AvCount']

#splitting data into a testing set and training set
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)

#creating an instance of the logistic regression model and fitting it with data
logistic_regression= LogisticRegression()
logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)

#confusion matrix used to visualize the predictions
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predict
sn.heatmap(confusion_matrix, annot=True, fmt='g')

#displaying the confusion matrix and accuracy of the model
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
plt.show()
```

```
Accuracy:  0.7283018867924528
```



In this confusion matrix, the 119 and 74 are actual predictions, and 40 and 32 are incorrect predictions. The model has as accuracy of approximately .73.

Prediction Model

We were unable to get this code to run.

```
ModelDataset = FullData.copy()
ModelDataset2 = FullData.copy()
d=ModelDataset['Date']
ModelDataset.insert(1,'Year',d.dt.year)
ModelDataset.insert(2,'Day of Year',d.dt.dayofyear)
ModelDataset.drop(['Date','avi_mark','avi_date', 'avi_number', 'avi_type'], axis=1, ir
ModelDataset2.drop(['avi_mark','avi_date', 'avi_type'], axis=1, inplace=True)
ModelDataset2.insert(1,'Year',d.dt.year)
ModelDataset2.insert(2,'Day of Year',d.dt.dayofyear)
del d

ModelDataset.head()
```


	Year	Day of	Avalanche	Snow Water Equivalent (in) Start	Pre Ac
14376	2014	2	No	9.146154	

Once our dataset is ready, we can define what our target variable below. For the Confusion Matrix Charts 0 is equal to no avalanches and 1 is equal to yes avalanches.

```
Model = setup(ModelDataset, target='Avalanche', numeric_features=['Year'])
```

	Description	Value
0	session_id	1506
1	Target	Avalanche
2	Target Type	Binary
3	Label Encoded	No: 0, Yes: 1
4	Original Data	(3492, 9)
5	Missing Values	True
6	Numeric Features	8
7	Categorical Features	0
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(2444, 8)
12	Transformed Test Set	(1048, 8)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	a3ee
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	Numeric Imputer	mean
25	Iterative Imputation Numeric Model	None
26	Categorical Imputer	constant
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent

29	Normalize	False
30	Normalize Method	None
31	Transformation	False
32	Transformation Method	None

```
compare_models()
```

	Model	Accuracy	AUC	Recall
rf	Random Forest Classifier	0.9055	0.9719	0.9078
lightgbm	Light Gradient Boosting Machine	0.9055	0.9684	0.9050
et	Extra Trees Classifier	0.9043	0.9736	0.8956
xgboost	Extreme Gradient Boosting	0.9006	0.9662	0.9012
catboost	CatBoost Classifier	0.8985	0.9671	0.9097
gbc	Gradient Boosting Classifier	0.8891	0.9614	0.9012
dt	Decision Tree Classifier	0.8846	0.8863	0.8994
knn	K Neighbors Classifier	0.8834	0.9415	0.8834
ada	Ada Boost Classifier	0.8756	0.9487	0.8843
nb	Naive Bayes	0.8613	0.9120	0.9135

55	Interaction Threshold	None

```
XGB = create_model(estimator='xgboost')
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.9020	0.9665	0.9065	0.8739	0.8899
1	0.9265	0.9778	0.9159	0.9159	0.9159
2	0.9020	0.9634	0.8879	0.8879	0.8879
3	0.8857	0.9620	0.8868	0.8545	0.8704
4	0.8811	0.9620	0.9057	0.8348	0.8688
5	0.8770	0.9530	0.9151	0.8220	0.8661
6	0.9098	0.9649	0.8585	0.9286	0.8922
7	0.9016	0.9645	0.9151	0.8661	0.8899
8	0.9139	0.9674	0.8962	0.9048	0.9005

Extreme Gradient Boosting Model

What is it and where does it come from? https://towardsdatascience.com/xgboost-theory-and-practice-fb8912930ad6

```
TunedModel = tune_model(XGB)
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.9020	0.9614	0.9346	0.8547	0.8929
1	0.9143	0.9755	0.9346	0.8772	0.9050
2	0.8776	0.9633	0.9065	0.8291	0.8661
3	0.8735	0.9598	0.9245	0.8099	0.8634
4	0.8852	0.9645	0.9434	0.8197	0.8772
5	0.8811	0.9573	0.9434	0.8130	0.8734
6	0.9180	0.9628	0.8962	0.9135	0.9048
7	0.9016	0.9716	0.9434	0.8475	0.8929
8	0.9016	0.9673	0.9151	0.8661	0.8899
9	0.8934	0.9813	0.9434	0.8333	0.8850
Mean	0.8948	0.9665	0.9285	0.8464	0.8850
SD	0.0145	0.0071	0.0164	0.0309	0.0140

```
evaluate_model(XGB)
```

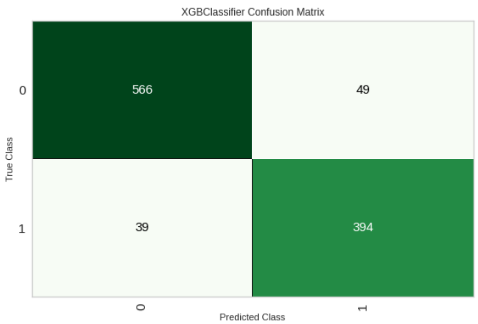

Plot Type: Hyperparameters

AUC
Confusion Matrix
Threshold
Precision Recall
Prediction Error
Class Report
Feature Selection
Learning Curve
Manifold Learning
Calibration Curve
Validation Curve
Dimensions
Feature Importance
Feature Importance…
Decision Boundary
Lift Chart
Gain Chart
Decision Tree

	Parameters
objective	binary:logistic
use_label_encoder	True
base_score	0.5
booster	gbtree
colsample_bylevel	1
colsample_bynode	1
colsample_bytree	1
gamma	0
gpu_id	-1
importance_type	gain
interaction_constraints	
learning_rate	0.300000012
max_delta_step	0

max_delta_step	0
max_depth	6

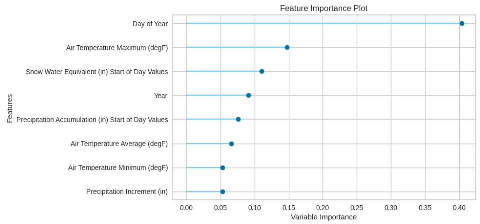
```
plot_model(XGB, 'confusion_matrix')
```



The confusion matrix shows us all the data points that were tested to verify the quality of the model. From the tested data points:

- 545 points were correctly classified as days without avalanches
- 47 days had an avalanche predicted but none happened
- In 405 days avalanches were predicted and they really happened
- In 51 days there were unpredicted avalanches

```
plot_model(XGB, 'feature')
```

Logistic Regression

```
LR = create_model(estimator='lr')
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.8816	0.9417	0.8785	0.8545	0.8664
1	0.8816	0.9568	0.8318	0.8900	0.8599
2	0.8571	0.9240	0.8785	0.8103	0.8430
3	0.8204	0.8981	0.8208	0.7768	0.7982
4	0.8115	0.9133	0.8208	0.7632	0.7909
5	0.8402	0.8956	0.8208	0.8131	0.8169
6	0.8279	0.9332	0.7642	0.8265	0.7941
7	0.8689	0.9309	0.8491	0.8491	0.8491
8	0.8689	0.9258	0.8962	0.8190	0.8559
9	0.8975	0.9578	0.9057	0.8649	0.8848
Mean	0.8556	0.9277	0.8466	0.8267	0.8359
SD	0.0277	0.0203	0.0412	0.0371	0.0317

```
TunedModel2 = tune_model(LR)
```

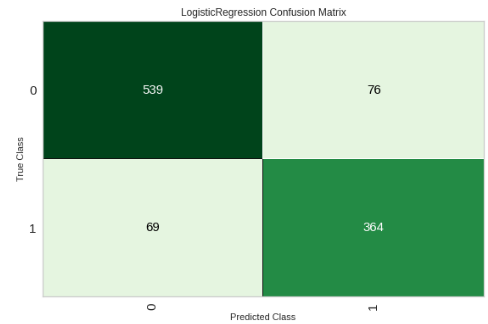
	Accuracy	AUC	Recall	Prec.	F1
0	0.8980	0.9412	0.8972	0.8727	0.8848
1	0.8816	0.9570	0.8318	0.8900	0.8599
2	0.8408	0.9212	0.8505	0.7982	0.8235
3	0.8286	0.9015	0.8302	0.7857	0.8073
4	0.8115	0.9134	0.8208	0.7632	0.7909
5	0.8402	0.8957	0.8208	0.8131	0.8169
6	0.8279	0.9345	0.7642	0.8265	0.7941
7	0.8689	0.9297	0.8585	0.8426	0.8505
8	0.8730	0.9246	0.8962	0.8261	0.8597
9	0.9016	0.9577	0.9057	0.8727	0.8889

```
evaluate_model(LR)
```


Plot Type: Hyperparameters

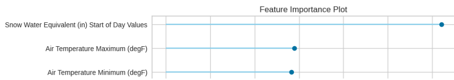
AUC

```
plot_model(LR, 'confusion_matrix')
```



Gain Chart

```
plot_model(LR, 'feature')
```



CatBoost


```
catboost = create_model(estimator='catboost')
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.8776	0.9657	0.8692	0.8532	0.8611
1	0.9347	0.9828	0.9439	0.9099	0.9266
2	0.8857	0.9619	0.9159	0.8376	0.8750
3	0.8776	0.9611	0.9057	0.8276	0.8649
4	0.8893	0.9559	0.9245	0.8376	0.8789
5	0.8811	0.9526	0.9057	0.8348	0.8688
6	0.9098	0.9719	0.8774	0.9118	0.8942
7	0.9016	0.9700	0.9151	0.8661	0.8899
8	0.9180	0.9669	0.9434	0.8772	0.9091
9	0.9098	0.9827	0.8962	0.8962	0.8962
Mean	0.8985	0.9671	0.9097	0.8652	0.8865
SD	0.0184	0.0096	0.0234	0.0305	0.0198

```
TunedModel3 = tune_model(catboost)
```


	Accuracy	AUC	Recall	Prec.	F1
0	0.8857	0.9616	0.8785	0.8624	0.8704
1	0.9306	0.9827	0.9346	0.9091	0.9217

```
evaluate_model(catboost)
```

Plot Type: Hyperparameters

AUC

Confusion Matrix

Threshold

Precision Recall

Prediction Error

Class Report

Feature Selection

Learning Curve

Manifold Learning

Calibration Curve

Validation Curve

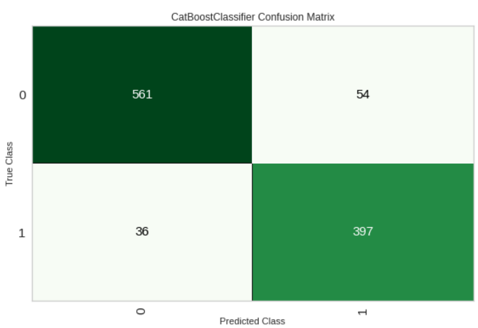
Dimensions

Feature Importance

Feature Importance…

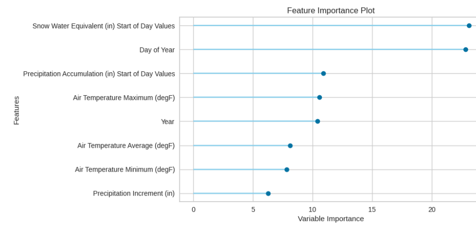
Decision Boundary

```
plot_model(catboost, 'confusion_matrix')
```

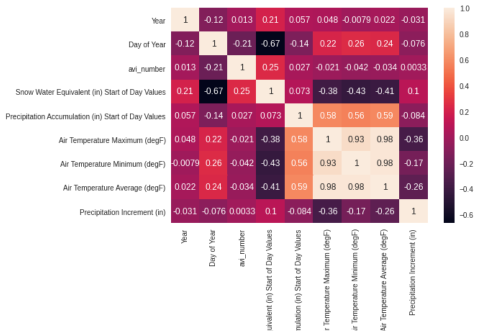


bayesian_matrix_reg 0.1000000014901

```
plot_model(catboost, 'feature')
```


```
import seaborn as sns

import matplotlib.pyplot as plt
correlation_mat = ModelDataset2.corr()

sns.heatmap(correlation_mat, annot = True)

plt.show()
```



Testing and Training

In this section, the data was tested and trained with a 60:40 and 80:20. Previously the data was tested and trained with a 70:30. The reason this is done is to see if the accuracy of the models changes. We look into the same three models as earlier. Included is a link showing all charts together so that it is easier to look through them. It shows all of the same models with similiar accuracies even if the testing and training ratio changes. The three models are Logistic Regression, CatBoost, and Extreme Gradient Boost. All models and testing and training has showed that Snow Water Equivalent is important. https://drive.google.com/file/d/1njKn81QJfTAFwHFyHKL_ljy8a6pCZoV4/view?usp=sharing

```
Model = setup(ModelDataset, target='Avalanche',train_size = 0.6,numeric_features=['Yea
compare_models()
```


	Model	Accuracy	AUC	Recall
lightgbm	Light Gradient Boosting Machine	0.9141	0.9696	0.9037
et	Extra Trees Classifier	0.9126	0.9750	0.8992
rf	Random Forest Classifier	0.9122	0.9731	0.9117
xgboost	Extreme Gradient Boosting	0.9055	0.9670	0.9026
catboost	CatBoost Classifier	0.9050	0.9661	0.9049
gbc	Gradient Boosting Classifier	0.8969	0.9609	0.8991

```
XGB = create_model(estimator='xgboost')
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.9048	0.9647	0.9213	0.8632	0.8913
1	0.8952	0.9516	0.8764	0.8764	0.8764
2	0.8952	0.9564	0.9213	0.8454	0.8817
3	0.9095	0.9816	0.9318	0.8632	0.8962
4	0.9333	0.9728	0.9318	0.9111	0.9213
5	0.8708	0.9590	0.8409	0.8506	0.8457
6	0.9043	0.9682	0.9318	0.8542	0.8913
7	0.9330	0.9865	0.9318	0.9111	0.9213
8	0.9043	0.9750	0.8523	0.9146	0.8824
9	0.9043	0.9546	0.8864	0.8864	0.8864
Mean	0.9055	0.9670	0.9026	0.8776	0.8894
SD	0.0172	0.0112	0.0338	0.0254	0.0207

```
TunedModel = tune_model(XGB)
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.8952	0.9536	0.8989	0.8602	0.8791
1	0.8714	0.9490	0.7865	0.8974	0.8383
2	0.9000	0.9580	0.8876	0.8778	0.8827
3	0.9000	0.9757	0.8864	0.8764	0.8814
4	0.9238	0.9690	0.9205	0.9000	0.9101
5	0.9043	0.9626	0.8636	0.9048	0.8837
6	0.8947	0.9571	0.8977	0.8587	0.8778
7	0.9234	0.9787	0.9091	0.9091	0.9091
8	0.9187	0.9730	0.8409	0.9610	0.8970
9	0.8900	0.9528	0.8295	0.9012	0.8639
Mean	0.9022	0.9629	0.8721	0.8947	0.8823

```
evaluate_model(XGB)
```


Plot Type: Hyperparameters

AUC

Confusion Matrix

Threshold

Precision Recall

Prediction Error

Class Report

Feature Selection

Learning Curve

Manifold Learning

Calibration Curve

Validation Curve

Dimensions

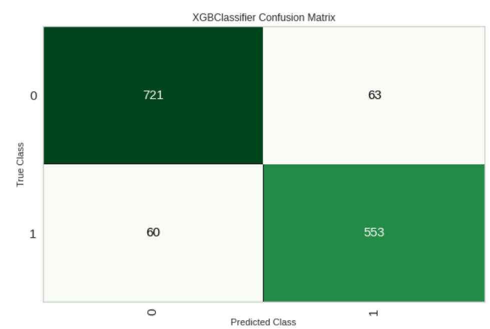
Feature Importance

Feature Importance…

Decision Boundary

Lift Chart

```
plot_model(XGB, 'confusion_matrix')
```




```
LR = create_model(estimator='lr')
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.8429	0.9009	0.8539	0.7917	0.8216
1	0.8524	0.9150	0.8202	0.8295	0.8249
2	0.8667	0.9168	0.8876	0.8144	0.8495
3	0.8952	0.9571	0.8636	0.8837	0.8736
4	0.8762	0.9304	0.8523	0.8523	0.8523
5	0.8756	0.9215	0.8636	0.8444	0.8539
6	0.8517	0.9149	0.8636	0.8000	0.8306
7	0.8708	0.9486	0.8523	0.8427	0.8475
8	0.8278	0.9193	0.7841	0.8023	0.7931
9	0.8325	0.8959	0.7386	0.8442	0.7879
Mean	0.8592	0.9221	0.8380	0.8305	0.8335
SD	0.0203	0.0181	0.0427	0.0271	0.0260

```
TunedModel = tune_model(LR)
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.8429	0.9008	0.8539	0.7917	0.8216
1	0.8524	0.9150	0.8202	0.8295	0.8249
2	0.8667	0.9173	0.8876	0.8144	0.8495
3	0.8952	0.9571	0.8636	0.8837	0.8736
4	0.8762	0.9297	0.8523	0.8523	0.8523
5	0.8756	0.9215	0.8636	0.8444	0.8539
6	0.8517	0.9150	0.8636	0.8000	0.8306
7	0.8708	0.9486	0.8523	0.8427	0.8475
8	0.8278	0.9193	0.7841	0.8023	0.7931
9	0.8325	0.8959	0.7386	0.8442	0.7879
Mean	0.8592	0.9220	0.8380	0.8305	0.8335
SD	0.0203	0.0180	0.0427	0.0271	0.0260

```
evaluate_model(LR)
```


Plot Type: Hyperparameters

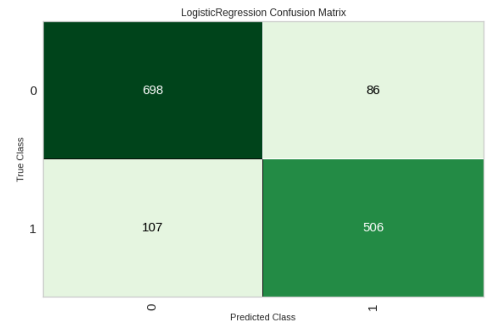
AUC

Confusion Matrix

Threshold

Precision Recall

```
plot_model(LR, 'confusion_matrix')
```




```
catboost = create_model(estimator='catboost')
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.9048	0.9651	0.9213	0.8632	0.8913
1	0.8857	0.9552	0.8539	0.8736	0.8636

```
TunedModel = tune_model(catboost)
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.8905	0.9650	0.8989	0.8511	0.8743
1	0.8952	0.9589	0.8764	0.8764	0.8764
2	0.8905	0.9589	0.9213	0.8367	0.8770
3	0.9190	0.9762	0.9205	0.8901	0.9050
4	0.9238	0.9762	0.9318	0.8913	0.9111
5	0.9139	0.9687	0.8864	0.9070	0.8966
6	0.9043	0.9666	0.9318	0.8542	0.8913
7	0.9187	0.9850	0.9432	0.8737	0.9071
8	0.9282	0.9770	0.8864	0.9398	0.9123
9	0.8947	0.9524	0.8636	0.8837	0.8736
Mean	0.9079	0.9685	0.9060	0.8804	0.8925
SD	0.0138	0.0096	0.0258	0.0282	0.0152

```
evaluate_model(catboost)
```


Plot Type: Hyperparameters

AUC
Confusion Matrix
Threshold
Precision Recall
Prediction Error
Class Report
Feature Selection
Learning Curve
Manifold Learning
Calibration Curve
Validation Curve
Dimensions
Feature Importance
Feature Importance…
Decision Boundary
Lift Chart
Gain Chart
Decision Tree

	Paramet
nan_mode	
eval_metric	Lo
iterations	
sampling_frequency	Per
leaf_estimation_method	Ne
grow_policy	Symmetric
penalties_coefficient	
boosting_type	I
model_shrink_mode	Con
feature_border_type	GreedyLog
bayesian_matrix_reg	0.1000000014901
l2_leaf_reg	

```
plot_model(catboost, 'confusion_matrix')
```

Processing:

```
---------------------------------------------------------------------------
TypeError
Traceback (most recent call last)
<ipython-input-111-0ebd7cfd32de> in
<module>()
----> 1 plot_model(catboost,
'confusion_matrix')
```

10 frames

```
<__array_function__ internals> in
union1d(*args, **kwargs)

<__array_function__ internals> in
unique(*args, **kwargs)

/usr/local/lib/python3.6/dist-
packages/numpy/lib/arraysetops.py in
_unique1d(ar, return_index, return_inverse,
return_counts)
    309         aux = ar[perm]
    310     else:
--> 311         ar.sort()
    312         aux = ar
    313     mask = np.empty(aux.shape,
dtype=np.bool_)

TypeError: '<' not supported between
instances of 'int' and 'str'
```

SEARCH STACK OVERFLOW


```
Model = setup(ModelDataset, target='Avalanche',train_size = 0.8,numeric_features=['Yea
compare_models()
```


	Model	Accuracy	AUC	Recall
et	Extra Trees Classifier	0.9166	0.9767	0.9019
rf	Random Forest Classifier	0.9144	0.9741	0.9120
xgboost	Extreme Gradient Boosting	0.9141	0.9700	0.9154
lightgbm	Light Gradient Boosting Machine	0.9137	0.9702	0.9137
catboost	CatBoost Classifier	0.9123	0.9698	0.9129
gbc	Gradient Boosting Classifier	0.8994	0.9651	0.9070
dt	Decision Tree Classifier	0.8919	0.8928	0.8994
knn	K Neighbors	0.8869	0.9443	0.8869

```
XGB = create_model(estimator='xgboost')
```

```
TunedModel = tune_model(XGB)
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.8964	0.9739	0.9333	0.8421	0.8854
1	0.9107	0.9662	0.9000	0.8926	0.8963
2	0.9214	0.9786	0.9417	0.8828	0.9113
3	0.8925	0.9683	0.9328	0.8346	0.8810
4	0.8889	0.9487	0.9076	0.8438	0.8745
5	0.9104	0.9715	0.9412	0.8615	0.8996
6	0.8961	0.9636	0.9244	0.8462	0.8835
7	0.8889	0.9673	0.9160	0.8385	0.8755
8	0.8925	0.9709	0.9244	0.8397	0.8800
9	0.9176	0.9877	0.9832	0.8478	0.9105
Mean	0.9015	0.9697	0.9304	0.8529	0.8898
SD	0.0116	0.0096	0.0218	0.0188	0.0130

```
evaluate_model(XGB)
```


Plot Type: Hyperparameters

AUC
Confusion Matrix
Threshold
Precision Recall
Prediction Error
Class Report
Feature Selection
Learning Curve
Manifold Learning
Calibration Curve
Validation Curve
Dimensions
Feature Importance
Feature Importance…
Decision Boundary
Lift Chart
Gain Chart
Decision Tree

Parameters

```
plot_model(XGB, 'confusion_matrix')
```


```
LR = create_model(estimator='lr')
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.8679	0.9317	0.8250	0.8609	0.8426
1	0.8679	0.9238	0.8333	0.8547	0.8439
2	0.8607	0.9489	0.8500	0.8293	0.8395
3	0.8674	0.9223	0.8655	0.8306	0.8477
4	0.8244	0.8991	0.7899	0.7966	0.7932
5	0.8746	0.9199	0.8403	0.8621	0.8511
6	0.8530	0.9431	0.8403	0.8197	0.8299
7	0.8602	0.9367	0.8739	0.8125	0.8421
8	0.8100	0.9015	0.7899	0.7705	0.7801
9	0.8602	0.9254	0.8739	0.8125	0.8421
Mean	0.8546	0.9252	0.8382	0.8249	0.8312
SD	0.0198	0.0153	0.0288	0.0278	0.0231

```
TunedModel = tune_model(LR)
```


Accuracy	AUC	Recall	Prec.	F1

```
evaluate_model(LR)
```

Plot Type: Hyperparameters

AUC

Confusion Matrix

Threshold

Precision Recall

Prediction Error

Class Report

Feature Selection

Learning Curve

Manifold Learning

Calibration Curve

Validation Curve

Dimensions

Feature Importance

Feature Importance…

Decision Boundary

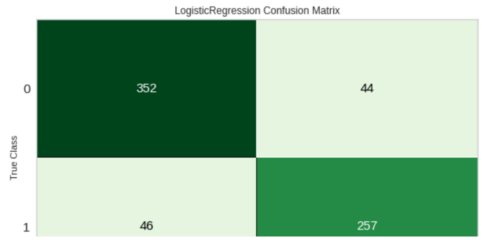
Lift Chart

Gain Chart

Decision Tree

	Parameters
C	1.0

```
plot_model(LR, 'confusion_matrix')
```




```
catboost = create_model(estimator='catboost')
```

	Accuracy	AUC	Recall	Prec.	F1
0	0.9000	0.9713	0.8750	0.8898	0.8824
1	0.9143	0.9665	0.8750	0.9211	0.8974
2	0.9393	0.9794	0.9333	0.9256	0.9295
3	0.9068	0.9659	0.9244	0.8661	0.8943
4	0.8889	0.9469	0.8992	0.8492	0.8735
5	0.9104	0.9707	0.9160	0.8790	0.8971
6	0.8889	0.9683	0.8908	0.8548	0.8724
7	0.9176	0.9683	0.9076	0.9000	0.9038
8	0.9211	0.9723	0.9412	0.8819	0.9106
9	0.9355	0.9881	0.9664	0.8915	0.9274
Mean	0.9123	0.9698	0.9129	0.8859	0.8988
SD	0.0163	0.0100	0.0279	0.0241	0.0189

```
TunedModel = tune_model(catboost)
```


	Accuracy	AUC	Recall	Prec.	F1
0	0.9143	0.9732	0.9167	0.8871	0.9016
1	0.9143	0.9698	0.8917	0.9068	0.8992
2	0.9393	0.9786	0.9333	0.9256	0.9295
3	0.9140	0.9696	0.9244	0.8800	0.9016

```
evaluate_model(catboost)
```

Plot Type: Hyperparameters

AUC

Confusion Matrix

Threshold

Precision Recall

Prediction Error

Class Report

Feature Selection

Learning Curve

Manifold Learning

Calibration Curve

Validation Curve

Dimensions

Feature Importance

Feature Importance…

Decision Boundary

Lift Chart

Gain Chart

```
plot_model(catboost, 'confusion_matrix')
```


Conclusions

In our study we analyzed data from avalanches and measured weather variables for specific locations and dates to identify whether it was possible to predict an avalanche using the existing measured data.

In this specific notebook the average of weather data from all locations was used in the analysis and prediction, so instead of focusing on particular data to understand avalanches in specific locations the study is more focused in a macro-weather environment that seems to be prone to avalanche incidence.

By looking at the distributions of weather data from the days with and without avalanches we can see that Snow Water Equivalent is very important and this was confirmed when looking at feature importance in the final model. Also we were able to identify that the majority of avalanches happen in spring, probably because that is when snow is melting and mountains might be more unstable.

The prediction model created is well capable to predict whether an avalanche might happen in one of the locations considering the overall weather data. However this model might not be good to predict the occurence of an avalanche in a specific location, as it uses that average of weather information. To get results for specific locations it would be recommended to collect weather data for as many locations as possible, and then cross those datasets to create the model.