Early Data Exploration

Joshua Elms

This notebook handles some basic parsing and processing for the data.

Description of Each Cell

1. Necessary imports for the notebook, might require installing these modules if you are attempting to run locally.
2. Handles parsing raw data sources and writing out two CSVs
   - Only needs to be run the first time using this program; after that, you can skip this cell and use the one below
   - Requires you to fill in the paths to each of your input and output sources (relative and absolute paths are both fine)
   - This could take anywhere from 30 seconds to 8-9 minutes, depending on your system
   - The cleaning used here (removing NaNs) is KNN Imputation . In essence, it finds the closest points to any missing values (calculated by looking at all the values that aren't missing) and fills in the missing based on its nearest neighbor's values.
   - Other methods for this could involve calculating the mean for the entire field and using that, or performing linear regression using the most highly correlated field to the one missing.
3. Using the csv we just wrote, we can load the new dataframe into memory
4. Print a description of summary statistics for each field in the dataframe
5. Generate the correlation matrix for the data - it is exactly the same as the one from your excel file, which is a good confirmation that the data cleaning didn't make the new data unrepresentative of the old
6. This is the start of an example of how I found a list of the most highly correlated variables. We define a function that generates all possible combinations of the input list, and pass into it our list of field names. The time complexity for the next cell is exponential because however many columns we pass to the function here, we will have 2^n combinations to check the correlations of (I ran the complete version of this on BigRed3 to calculate the best ones shown below).

7. Iterate through the list of combinations and perform multilinear regression on each of them, then use the regression results to calculate the correlation between those variables and hailstone size. Every time a new combination is better than the last, it will be printed out for viewing.

Best Combinations Found on BR3

I excluded a few ranges of variables for the BR3 run; first, columns 1-22 included almost identical versions of 23-33, so I decided not to go for the diminishing returns on accuracy there and just took 1-22 out. Second, of all the columns from 37-54, I only kept 41-50 because the rest were composites that I was attempting to beat. In total, I only used the ranges 21-37 and 41-50, giving 25 variables and 2^25 = 33554432 combinations to check with BR3.

Of these, it returned about 45 new best combinations, and I have included the 3 big jumps in correlation coefficient below.

- Variables SB LI, SB b3km, Shear 0-6 km, and Eff Inflow yield a correlation of 0.2214997567203551
- Variables SB CIN, SB LCL, SB LFC, SB LI, SB hght0c, SB b3km, Shear 0-6 km, Eff Inflow, and SRH 0-3 km yield a correlation of 0.22440901507528213
- Variables SB CIN, SB LCL, SB LFC, SB EL, SB LI, SB hght0c, SB cap, SB b3km, SB brn, Shear 0-1 km, Shear 0-6 km, Eff Inflow, ebwd[0], SRH 0-1 km, SRH 0-3 km, and Eff SRH yield a correlation of 0.22614826662814497

Just from these, we can see it is relatively simple to beat SHIP with only SB Li, SB b3km, Shear 0-6km, and Eff Inflow which have a correlation of ~0.2215 with hailstone size, compared to SHIP's 0.2037.

In [ ]:

```
#1 All modules necessary to run this notebook

import pandas as pd
from itertools import combinations, chain
from sklearn.linear_model import LinearRegression
from sklearn.impute import KNNImputer
from numpy import corrcoef
```

In [ ]:

```
#2 Parses data sources into usable dataframe, could take a while


raw_data_path = "" # fill in with path to raw data of 53 parameters for each ob
raw_sizes_path = "" # fill in w/ path to sizes of hail
tidy_data_path = "" # fill in path for where you want the tidy (but incomplete,
col_names_path = "" # fill w/ path to parameter names file
clean_data_path = "" # fill w/ path to file you want to store clean and complet


with open(raw_data_path, "r") as f1:
    with open(raw_sizes_path, "r") as f2:
        with open(tidy_data_path, "w") as f3:
```

```
            with open(col_names_path, "r") as f4:
                ### Write Field Names ###
                col_names = [line.strip().rstrip("\n") for line in f4.readlines
                col_names.append("Hailstone Size")
                f3.write(",".join(col_names) + "\n")

                ### Read every 53 items, then read size and write all 54 into a
                pos = 2
                line = f1.readline().strip().rstrip("\n")
                entry_lst = [line]

                while line:
                    line = f1.readline().strip().rstrip("\n")
                    entry_lst.append(line)
                    pos += 1

                    if pos == 54:
                        pos = 1
                        entry_lst.append(f2.readline().strip().rstrip("\n"))
                        f3.write(",".join(entry_lst) + "\n")
                        entry_lst = []


# Read tidy csv and clean it up with KNN Imputation, relabel and write out
df = pd.read_csv(tidy_data_path)
df_knn = KNNImputer().fit_transform(df)
df_knn_actual = pd.DataFrame(df_knn)
df_knn_actual.columns = df.columns

df_knn_actual.to_csv(clean_data_path)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [19], in <cell line: 11>()
      7 col_names_path = "" # fill w/ path to parameter names file
      8 clean_data_path = "" # fill w/ path to file you want to store clean an
d complete data in
---> 11 with open(raw_data_path, "r") as f1:
     12     with open(raw_sizes_path, "r") as f2:
     13         with open(tidy_data_path, "w") as f3:

FileNotFoundError: [Errno 2] No such file or directory: ''
```

In [ ]:

```
#3 Load dataframe into memory, display head and tail

path = "/Users/joshuaelms/Desktop/github_repos/CSCI-B365/Meteorology_Modeling_P

df = pd.read_csv(path, index_col=0)
df.index += 1
df
```

Out[ ]:

	ML CAPE	ML CIN	ML LCL	ML LFC	ML EL	ML LI	ML hg
1	565.886137	-2.456216	591.712340	760.740300	10016.261419	-2.475117	3057.72
2	93.557330	-61.118000	818.659297	1485.730600	4147.988929	1.094013	2878.87
3	416.713894	-0.701233	682.113493	751.489413	7419.731564	-2.174859	3043.08
4	1110.622796	-12.420499	536.926037	989.547800	11364.753475	-4.154931	3532.14
5	1107.162497	-12.514324	536.912773	1008.662600	11386.082876	-4.102513	3583.43
...	...	...	...	...	...	...	
29098	0.000000	0.000000	2343.569759	1107.279086	2343.569759	12.494209	3528.14
29099	0.000000	0.000000	2326.323051	1107.279086	2326.323051	14.986276	3497.38
29100	0.000000	0.000000	2690.384769	2630.432840	2690.384769	14.638317	3482.93
29101	0.000000	0.000000	2807.261593	2441.488395	2807.261593	16.123160	3451.46
29102	0.000000	0.000000	2902.643807	2796.044274	2902.643807	16.707049	3439.64

29102 rows × 54 columns

In [ ]:

```
#4 Prints summary statistics for df, will be way too long to display nicely

df.describe()
```

Out[ ]:

	ML CAPE	ML CIN	ML LCL	ML LFC	ML EL	ML LI
count	29102.000000	29102.000000	29102.000000	29102.000000	29102.000000	29102.000000
mean	1141.143080	-81.896813	1301.060693	2314.606640	9690.948151	-3.514117
std	989.637687	140.670608	552.490866	1128.501556	3429.726589	3.998366
min	0.000000	-3509.328383	383.101334	386.799418	396.859210	-15.297868
25%	305.835266	-112.762023	893.117781	1478.504345	8644.974500	-6.035576
50%	958.885288	-40.506631	1191.115297	2261.775381	10711.671530	-3.935588
75%	1748.335956	-4.875115	1618.797002	2970.830400	11972.982532	-1.644113
max	6212.355825	0.000000	4839.833885	11801.322738	15505.246353	28.964380

8 rows × 54 columns

In [ ]:

```
#5 Generate the correlation matrix for dataframe, show most strongly correlatec

df_corr = df.corr()

df_corr["Hailstone Size"].abs().sort_values(ascending=False)
```

Out[ ]:

```
Hailstone Size    1.000000
ship              0.203719
crav              0.179000
lrat              0.161617
sweat             0.138882
MU LI             0.137036
SB LI             0.128339
ML LI             0.123399
SB CAPE           0.119698
MU CAPE           0.115961
DCAPE             0.113458
Shear 0-6 km      0.109000
ML CAPE           0.107440
ML EL             0.092197
SB EL             0.090327
stp_fixed         0.087747
MU EL             0.072912
tei               0.072430
SB LCL            0.071378
SB hght0c         0.070903
MU hght0c         0.070903
ML hght0c         0.070903
stp_mixed         0.068026
scp               0.063640
ML LFC            0.056274
ebwd[0]           0.051901
pwat              0.049410
Eff SRH           0.047059
mlmr              0.035647
SB LFC            0.035416
ML LCL            0.032367
ML b3km           0.029507
Shear 0-1 km      0.027687
mu_tlcl           0.027546
MU cap            0.027209
ML CIN            0.026495
MU brn            0.017402
ML cap            0.017300
Eff Inflow        0.017007
ML brn            0.016907
SRH 0-1 km        0.016673
SB brn            0.016409
ml_tlcl           0.016282
SB CIN            0.015348
sb_tlcl           0.012252
MU b3km           0.008658
MU LFC            0.007641
SRH 0-3 km        0.007508
kinx              0.007002
SB b3km           0.005629
t500              0.004464
MU LCL            0.003182
MU CIN            0.001754
SB cap            0.000352
Name: Hailstone Size, dtype: float64
```

In [ ]:

```python
#6 Generate the power set (without the empty set) for any given iterator
def power_set(iterable):
    pset = chain.from_iterable(combinations(iterable, r) for r in range(len(ite
    return list(list(combo) for combo in pset if len(combo) > 0)
```

```python
# This section is really just an example; the version I wrote to check all comb
# That is a very large set of combinations, so I ran it on BigRed3 for hours to
num_cols = 3
field_pset = power_set(df.columns[:num_cols]) # exponential time complexity; 2^
field_pset
```

Out[ ]:

```
[['ML CAPE'],
 ['ML CIN'],
 ['ML LCL'],
 ['ML CAPE', 'ML CIN'],
 ['ML CAPE', 'ML LCL'],
 ['ML CIN', 'ML LCL'],
 ['ML CAPE', 'ML CIN', 'ML LCL']]
```

In [ ]:

```python
#7 loop over every combination of powerset, prints the new best combination eve
max = 0
for combination in field_pset:
    arr = df[combination].to_numpy()
    target = df["Hailstone Size"].to_numpy()
    obj = LinearRegression().fit(X=arr, y=target)
    coefficients = obj.coef_
    linear_combination = (df[combination]*coefficients).sum(axis=1)
    correlation = corrcoef(linear_combination, target)

    corr = correlation[0][1]

    if corr > max:
        max = corr
        variable_str = f'{", ".join(combination[:-1])}, and {combination[-1]}'
        print(f"Variable{'s' if len(combination) > 1 else ''} {variable_str} yi
```

```
Variable ML CAPE yields a correlation of 0.10744035377599044
Variable ML LI yields a correlation of 0.12339936047286572
Variables ML CAPE and ML LFC yield a correlation of 0.13397420322247117
Variables ML CAPE and ML b3km yield a correlation of 0.1413263618247371
Variables ML LI and ML b3km yield a correlation of 0.14939146315491544
Variables ML CAPE, ML LI, and ML b3km yield a correlation of 0.154325594128616
6
Variables ML CAPE, ML CIN, ML LI, and ML b3km yield a correlation of 0.1566452
1453989333
Variables ML CAPE, ML LI, ML cap, and ML b3km yield a correlation of 0.1608582
833480194
Variables ML CAPE, ML CIN, ML LI, ML cap, and ML b3km yield a correlation of
0.16556220762448348
Variables ML CAPE, ML CIN, ML LCL, ML LI, ML cap, and ML b3km yield a correlat
ion of 0.16707554039400685
Variables ML CAPE, ML CIN, ML LCL, ML LFC, ML LI, ML cap, and ML b3km yield a
correlation of 0.16740602037646263
Variables ML CAPE, ML CIN, ML LCL, ML EL, ML LI, ML cap, and ML b3km yield a c
orrelation of 0.16755966365650488
Variables ML CAPE, ML CIN, ML LCL, ML LI, ML cap, ML b3km, and ML brn yield a
correlation of 0.16810054739256006
Variables ML CAPE, ML CIN, ML LCL, ML LFC, ML LI, ML cap, ML b3km, and ML brn
yield a correlation of 0.1684263071780544
Variables ML CAPE, ML CIN, ML LCL, ML EL, ML LI, ML cap, ML b3km, and ML brn y
ield a correlation of 0.16855792958203014
Variables ML CAPE, ML CIN, ML LCL, ML LFC, ML EL, ML LI, ML cap, ML b3km, and
ML brn yield a correlation of 0.1690631748181805
Variables ML CAPE, ML CIN, ML LCL, ML LFC, ML EL, ML LI, ML hght0c, ML cap, ML
b3km, and ML brn yield a correlation of 0.1690632142248824
```