datascience_dash_project

August 19, 2020

1 *Tree Growth Prediction Project*

Author

Ahmet Ozdemir

Dataset

Dataset contains tree observations from four areas of one national forest district. This dataset includes information on tree type, shadow coverage, distance to nearby landmarks, soil type, and local topography.

Goal

The goal of the project is to build a model that predicts what types of trees grow in an area.

Target Variable

Cover type: identifies the tree types.

Project Steps

1.1 *Exploratory data analysis, data preparation and transformation*

[394]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go

# ignore the warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# for printing more rows and columns than the pandas' default behavior
pd.options.display.max_rows = 100
pd.options.display.max_columns = 100

# for printing desired digit in floats
pd.options.display.float_format = '{:,.3f}'.format
# max column width
pd.options.display.max_colwidth = 100

# font definitions
font_title = {'family': 'times new roman', 'color': 'darkred',
              'weight': 'bold', 'size': 14}
font_axis  = {'family': 'times new roman', 'color': 'darkred',
              'weight': 'bold', 'size': 14}

#custom color codes
my_colors = ['#CB4335','#3498DB', '#138D75',
             '#95a5a6', '#9b59b6', '#73C6B6',
             '#C39BD3', '#E59866', '#16A085']
my_palette = sns.color_palette(my_colors)
binary_colors = ['#F0C3BA','#5E77DE']
binary_palette = sns.color_palette(binary_colors)
```

1.1.1 *SQL Import and Export*

- MySQL server is used from the command line and commands are presented in the markdown.
- For retrieving the column names as desired input for the database `awk` and `sed` tools are used.

Creating the database

- Connect as root.

```
mysql -u root -p
```

- Create database, user and grant priviliges to that user.

```
mysql> CREATE DATABASE tree;
```

```
mysql> CREATE USER 'ahmet'@'localhost' IDENTIFIED WITH mysql_native_password BY
'Project1234!';
```

```
mysql> GRANT ALL ON tree.* TO 'ahmet'@'localhost';
```

- Set `local_infile` variable to `true` in order to import from `.csv` file.

```
set global local_infile=true;
```

```
exit
```

- Connect as new user with local_infile variable.

```
mysql --local_infile=1 -u ahmet -p
```

- Switch database.

```
mysql> use tree;
```

- From another shell prepare the table column names(lowercase) and column types(add space and INTEGER keyword)

```
head -1 data/covtype.csv | awk '{print tolower($0)}' | sed 's/,/ INTEGER, /g'
```

- Create table from sql shell

```
mysql> CREATE TABLE covtype (elevation INTEGER, aspect INTEGER, slope INTEGER,
horizontal_distance_to_hydrology INTEGER, vertical_distance_to_hydrology INTEGER,
horizontal_distance_to_roadways INTEGER, hillshade_9am INTEGER, hillshade_noon
INTEGER, hillshade_3pm INTEGER, horizontal_distance_to_fire_points INTEGER,
wilderness_area1 INTEGER, wilderness_area2 INTEGER, wilderness_area3 INTEGER,
wilderness_area4 INTEGER, soil_type1 INTEGER, soil_type2 INTEGER, soil_type3
INTEGER, soil_type4 INTEGER, soil_type5 INTEGER, soil_type6 INTEGER, soil_type7
INTEGER, soil_type8 INTEGER, soil_type9 INTEGER, soil_type10 INTEGER, soil_type11
INTEGER, soil_type12 INTEGER, soil_type13 INTEGER, soil_type14 INTEGER,
soil_type15 INTEGER, soil_type16 INTEGER, soil_type17 INTEGER, soil_type18
INTEGER, soil_type19 INTEGER, soil_type20 INTEGER, soil_type21 INTEGER,
soil_type22 INTEGER, soil_type23 INTEGER, soil_type24 INTEGER, soil_type25
INTEGER, soil_type26 INTEGER, soil_type27 INTEGER, soil_type28 INTEGER,
soil_type29 INTEGER, soil_type30 INTEGER, soil_type31 INTEGER, soil_type32
INTEGER, soil_type33 INTEGER, soil_type34 INTEGER, soil_type35 INTEGER,
soil_type36 INTEGER, soil_type37 INTEGER, soil_type38 INTEGER, soil_type39
INTEGER, soil_type40 INTEGER, cover_type INTEGER);
```

- From another shell prepare the table column names

```
head -1 data/covtype.csv | awk '{print tolower($0)}' | sed 's/,/, /g'
```

- Load local data from `.csv`

```
LOAD DATA LOCAL INFILE 'data/covtype.csv' INTO TABLE covtype FIELDS TERMINATED
BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n' IGNORE 1 LINES (elevation,
aspect, slope, horizontal_distance_to_hydrology, vertical_distance_to_hydrology,
horizontal_distance_to_roadways, hillshade_9am, hillshade_noon, hillshade_3pm,
horizontal_distance_to_fire_points, wilderness_area1, wilderness_area2,
wilderness_area3, wilderness_area4, soil_type1, soil_type2, soil_type3,
soil_type4, soil_type5, soil_type6, soil_type7, soil_type8, soil_type9,
soil_type10, soil_type11, soil_type12, soil_type13, soil_type14, soil_type15,
soil_type16, soil_type17, soil_type18, soil_type19, soil_type20, soil_type21,
soil_type22, soil_type23, soil_type24, soil_type25, soil_type26, soil_type27,
soil_type28, soil_type29, soil_type30, soil_type31, soil_type32, soil_type33,
soil_type34, soil_type35, soil_type36, soil_type37, soil_type38, soil_type39,
soil_type40, cover_type);
```

Reading SQL data

[38]:
```
from sqlalchemy import create_engine

# engine definition; user, password and db info provided
engine = create_engine('mysql+mysqlconnector://ahmet:Project1234!@localhost/
 ↪tree')

# read sql table
forest = pd.read_sql_table('covtype', engine)
```

1.1.2 *Basic data information*

- Data is completely numeric.

[51]:
```
forest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 581012 entries, 0 to 581011
Data columns (total 55 columns):
 #   Column                              Non-Null Count   Dtype
---  ------                              --------------   -----
 0   elevation                           581012 non-null  int64
 1   aspect                              581012 non-null  int64
 2   slope                               581012 non-null  int64
 3   horizontal_distance_to_hydrology    581012 non-null  int64
 4   vertical_distance_to_hydrology      581012 non-null  int64
 5   horizontal_distance_to_roadways     581012 non-null  int64
 6   hillshade_9am                       581012 non-null  int64
 7   hillshade_noon                      581012 non-null  int64
 8   hillshade_3pm                       581012 non-null  int64
 9   horizontal_distance_to_fire_points  581012 non-null  int64
 10  wilderness_area1                    581012 non-null  int64
 11  wilderness_area2                    581012 non-null  int64
 12  wilderness_area3                    581012 non-null  int64
 13  wilderness_area4                    581012 non-null  int64
 14  soil_type1                          581012 non-null  int64
 15  soil_type2                          581012 non-null  int64
 16  soil_type3                          581012 non-null  int64
 17  soil_type4                          581012 non-null  int64
 18  soil_type5                          581012 non-null  int64
 19  soil_type6                          581012 non-null  int64
 20  soil_type7                          581012 non-null  int64
 21  soil_type8                          581012 non-null  int64
 22  soil_type9                          581012 non-null  int64
 23  soil_type10                         581012 non-null  int64
 24  soil_type11                         581012 non-null  int64
 25  soil_type12                         581012 non-null  int64
 26  soil_type13                         581012 non-null  int64
```

```
 27  soil_type14                 581012 non-null  int64
 28  soil_type15                 581012 non-null  int64
 29  soil_type16                 581012 non-null  int64
 30  soil_type17                 581012 non-null  int64
 31  soil_type18                 581012 non-null  int64
 32  soil_type19                 581012 non-null  int64
 33  soil_type20                 581012 non-null  int64
 34  soil_type21                 581012 non-null  int64
 35  soil_type22                 581012 non-null  int64
 36  soil_type23                 581012 non-null  int64
 37  soil_type24                 581012 non-null  int64
 38  soil_type25                 581012 non-null  int64
 39  soil_type26                 581012 non-null  int64
 40  soil_type27                 581012 non-null  int64
 41  soil_type28                 581012 non-null  int64
 42  soil_type29                 581012 non-null  int64
 43  soil_type30                 581012 non-null  int64
 44  soil_type31                 581012 non-null  int64
 45  soil_type32                 581012 non-null  int64
 46  soil_type33                 581012 non-null  int64
 47  soil_type34                 581012 non-null  int64
 48  soil_type35                 581012 non-null  int64
 49  soil_type36                 581012 non-null  int64
 50  soil_type37                 581012 non-null  int64
 51  soil_type38                 581012 non-null  int64
 52  soil_type39                 581012 non-null  int64
 53  soil_type40                 581012 non-null  int64
 54  cover_type                  581012 non-null  int64
dtypes: int64(55)
memory usage: 243.8 MB
```

[40]:
```
forest.head()
```

[40]:

	elevation	aspect	slope	horizontal_distance_to_hydrology	vertical_distance_to_hydrology	horizontal_distance_to_roadways
0	2596	51	3	258	0	510
1	2590	56	2	212	-6	390
2	2804	139	9	268	65	3180
3	2785	155	18	242	118	3090
4	2595	45	2	153	-1	391

```
   hillshade_9am  hillshade_noon  hillshade_3pm  \
0            221             232            148
1            220             235            151
2            234             238            135
3            238             238            122
4            220             234            150

   horizontal_distance_to_fire_points  wilderness_area1  wilderness_area2  \
0                                6279                 1                 0
1                                6225                 1                 0
2                                6121                 1                 0
3                                6211                 1                 0
4                                6172                 1                 0

   wilderness_area3  wilderness_area4  soil_type1  soil_type2  soil_type3  \
0                 0                 0           0           0           0
1                 0                 0           0           0           0
2                 0                 0           0           0           0
3                 0                 0           0           0           0
4                 0                 0           0           0           0

   soil_type4  soil_type5  soil_type6  soil_type7  soil_type8  soil_type9  \
0           0           0           0           0           0           0
1           0           0           0           0           0           0
2           0           0           0           0           0           0
3           0           0           0           0           0           0
4           0           0           0           0           0           0

   soil_type10  soil_type11  soil_type12  soil_type13  soil_type14  \
0            0            0            0            0            0
1            0            0            0            0            0
2            0            0            1            0            0
3            0            0            0            0            0
4            0            0            0            0            0

   soil_type15  soil_type16  soil_type17  soil_type18  soil_type19  \
0            0            0            0            0            0
1            0            0            0            0            0
2            0            0            0            0            0
3            0            0            0            0            0
4            0            0            0            0            0

   soil_type20  soil_type21  soil_type22  soil_type23  soil_type24  \
0            0            0            0            0            0
1            0            0            0            0            0
2            0            0            0            0            0
3            0            0            0            0            0
```

```
4              0              0              0              0              0

   soil_type25  soil_type26  soil_type27  soil_type28  soil_type29  \
0            0            0            0            0            1
1            0            0            0            0            1
2            0            0            0            0            0
3            0            0            0            0            0
4            0            0            0            0            1

   soil_type30  soil_type31  soil_type32  soil_type33  soil_type34  \
0            0            0            0            0            0
1            0            0            0            0            0
2            0            0            0            0            0
3            1            0            0            0            0
4            0            0            0            0            0

   soil_type35  soil_type36  soil_type37  soil_type38  soil_type39  \
0            0            0            0            0            0
1            0            0            0            0            0
2            0            0            0            0            0
3            0            0            0            0            0
4            0            0            0            0            0

   soil_type40  cover_type
0            0           5
1            0           5
2            0           2
3            0           2
4            0           5
```

1.1.3 *Detection of variable types*

- There is 10 continuos and 45 categoric variables which are all dummy encoded except `cover_type`.

[73]:
```
# identify the number of unique values and variable type(categoric, continuous)
# and print the df
def unique_vals(df):
    type_of_vars = {}
    cols_df = pd.DataFrame([])
    for col in df.columns:
        type_of_vars['col_name'] = col
        # if number of unique vals greater than 20, assume that it is␣
↪continuous, else categoric
        type_of_vars['type'] = 'continuous' if df[col].nunique()>20 else␣
↪'categoric'
```

```
        type_of_vars['n_unique'] = df[col].nunique()
        type_of_vars['unique'] = df[col].unique()
        cols_df = pd.concat([cols_df,pd.
↪DataFrame([type_of_vars])],ignore_index=True)
    # sort and display
    #display(cols_df.sort_values(by='n_unique'))
    return cols_df
```

[183]:
```
df = unique_vals(forest)
# to inspect the unique values more thoroughly
pd.set_option('display.max_colwidth', 200)

display(df)
```

	col_name	type	n_unique	\
0	elevation	continuous	1978	
1	aspect	continuous	361	
2	slope	continuous	67	
3	horizontal_distance_to_hydrology	continuous	551	
4	vertical_distance_to_hydrology	continuous	700	
5	horizontal_distance_to_roadways	continuous	5785	
6	hillshade_9am	continuous	207	
7	hillshade_noon	continuous	185	
8	hillshade_3pm	continuous	255	
9	horizontal_distance_to_fire_points	continuous	5827	
10	wilderness_area1	categoric	2	
11	wilderness_area2	categoric	2	
12	wilderness_area3	categoric	2	
13	wilderness_area4	categoric	2	
14	soil_type1	categoric	2	
15	soil_type2	categoric	2	
16	soil_type3	categoric	2	
17	soil_type4	categoric	2	
18	soil_type5	categoric	2	
19	soil_type6	categoric	2	
20	soil_type7	categoric	2	
21	soil_type8	categoric	2	
22	soil_type9	categoric	2	
23	soil_type10	categoric	2	
24	soil_type11	categoric	2	
25	soil_type12	categoric	2	
26	soil_type13	categoric	2	
27	soil_type14	categoric	2	
28	soil_type15	categoric	2	
29	soil_type16	categoric	2	
30	soil_type17	categoric	2	
31	soil_type18	categoric	2	

32	soil_type19	categoric	2
33	soil_type20	categoric	2
34	soil_type21	categoric	2
35	soil_type22	categoric	2
36	soil_type23	categoric	2
37	soil_type24	categoric	2
38	soil_type25	categoric	2
39	soil_type26	categoric	2
40	soil_type27	categoric	2
41	soil_type28	categoric	2
42	soil_type29	categoric	2
43	soil_type30	categoric	2
44	soil_type31	categoric	2
45	soil_type32	categoric	2
46	soil_type33	categoric	2
47	soil_type34	categoric	2
48	soil_type35	categoric	2
49	soil_type36	categoric	2
50	soil_type37	categoric	2
51	soil_type38	categoric	2
52	soil_type39	categoric	2
53	soil_type40	categoric	2
54	cover_type	categoric	7

0	[2596, 2590, 2804, 2785, 2595, 2579, 2606, 2605, 2617, 2612, 2886, 2742, 2609, 2503, 2495,
1	[51, 56, 139, 155, 45, 132, 49, 59, 201, 151, 134, 214, 157, 259, 72, 0, 38, 71, 209, 114,
2	[3, 2, 9, 18, 6, 7, 4, 10, 11, 22, 1, 5, 17, 23, 8, 16, 24, 30, 19, 28, 12, 27, 25, 14, 13
3	[258, 212, 268, 242, 153, 300, 270, 234, 240, 247, 180, 371, 150, 67, 42, 120, 85, 95, 60,
4	[0, -6, 65, 118, -1, -15, 5, 7, 56, 11, 51, 26, 69, 46, 4, 2, 6, 10, 8, 30, 82, 14, -4, 23
5	[510, 390, 3180, 3090, 391, 67, 633, 573, 666, 636, 735, 5253, 3215, 771, 674, 752, 607, 5
6	[221, 220, 234, 238, 230, 222, 223, 228, 218, 248, 213, 224, 216, 214, 206, 252, 225, 215,
7	[232, 235, 238, 234, 237, 225, 230, 221, 219, 243, 240, 224, 247, 239, 227, 228, 223, 253,
8	[148, 151, 135, 122, 150, 140, 138, 144, 133, 124, 161, 136, 92, 170, 137, 156, 126, 179,
9	[6279, 6225, 6121, 6211, 6172, 6031, 6256, 6228, 6244, 6230, 6222, 4051, 6091, 5600, 5576,
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	

[182]:
```
categoric = [col for col in forest.columns if forest[col].nunique()<20]
continuous = [col for col in forest.columns if forest[col].nunique()>20]

print('\nContinuous Features:\n\n',continuous)

print('\nCategorical Features:\n\n',categoric)
```

```
Continuous Features:

 ['elevation', 'aspect', 'slope', 'horizontal_distance_to_hydrology',
'vertical_distance_to_hydrology', 'horizontal_distance_to_roadways',
'hillshade_9am', 'hillshade_noon', 'hillshade_3pm',
'horizontal_distance_to_fire_points']
```

```
Categorical Features:

 ['wilderness_area1', 'wilderness_area2', 'wilderness_area3',
'wilderness_area4', 'soil_type1', 'soil_type2', 'soil_type3', 'soil_type4',
'soil_type5', 'soil_type6', 'soil_type7', 'soil_type8', 'soil_type9',
'soil_type10', 'soil_type11', 'soil_type12', 'soil_type13', 'soil_type14',
'soil_type15', 'soil_type16', 'soil_type17', 'soil_type18', 'soil_type19',
'soil_type20', 'soil_type21', 'soil_type22', 'soil_type23', 'soil_type24',
'soil_type25', 'soil_type26', 'soil_type27', 'soil_type28', 'soil_type29',
'soil_type30', 'soil_type31', 'soil_type32', 'soil_type33', 'soil_type34',
'soil_type35', 'soil_type36', 'soil_type37', 'soil_type38', 'soil_type39',
'soil_type40', 'cover_type']
```

1.1.4 *Missing Values*

- There is no missing values in data.

[50]:
```
forest.isnull().mean()
```

[50]:
```
elevation                           0.000
aspect                              0.000
slope                               0.000
horizontal_distance_to_hydrology    0.000
vertical_distance_to_hydrology      0.000
horizontal_distance_to_roadways     0.000
hillshade_9am                       0.000
hillshade_noon                      0.000
hillshade_3pm                       0.000
horizontal_distance_to_fire_points  0.000
wilderness_area1                    0.000
wilderness_area2                    0.000
wilderness_area3                    0.000
wilderness_area4                    0.000
soil_type1                          0.000
soil_type2                          0.000
soil_type3                          0.000
soil_type4                          0.000
soil_type5                          0.000
soil_type6                          0.000
soil_type7                          0.000
soil_type8                          0.000
soil_type9                          0.000
soil_type10                         0.000
soil_type11                         0.000
soil_type12                         0.000
soil_type13                         0.000
```

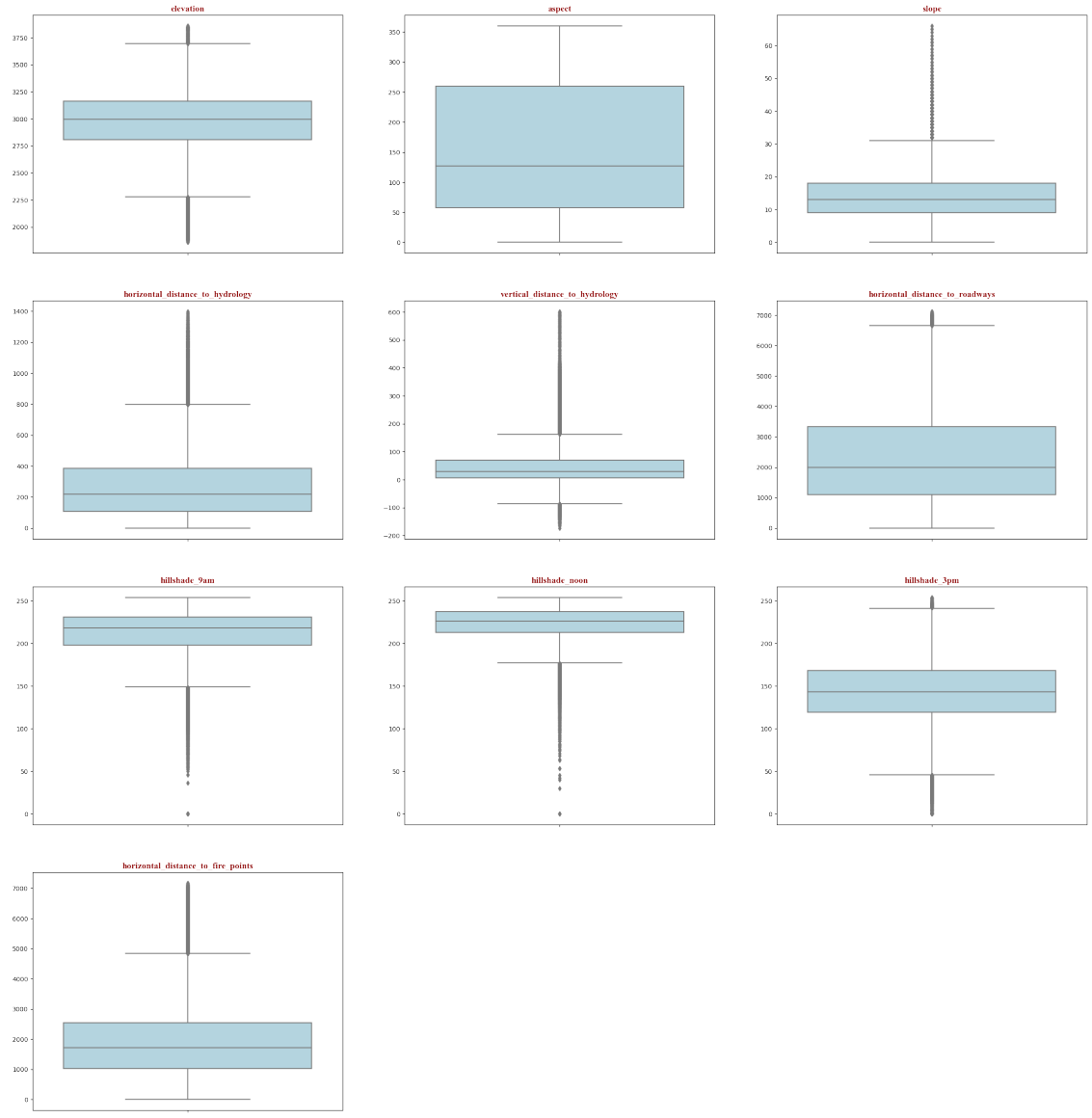
```
soil_type14                         0.000
soil_type15                         0.000
soil_type16                         0.000
soil_type17                         0.000
soil_type18                         0.000
soil_type19                         0.000
soil_type20                         0.000
soil_type21                         0.000
soil_type22                         0.000
soil_type23                         0.000
soil_type24                         0.000
soil_type25                         0.000
soil_type26                         0.000
soil_type27                         0.000
soil_type28                         0.000
soil_type29                         0.000
soil_type30                         0.000
soil_type31                         0.000
soil_type32                         0.000
soil_type33                         0.000
soil_type34                         0.000
soil_type35                         0.000
soil_type36                         0.000
soil_type37                         0.000
soil_type38                         0.000
soil_type39                         0.000
soil_type40                         0.000
cover_type                          0.000
dtype: float64
```

1.1.5 *Outliers*

- There are many outliers and they are not removed.

[364]:
```
# identify the number of columns
column_n = 3
# calculate the number fo rows
row_n = np.ceil(len(continuous)/column_n)
# calculate the figure size
plt.figure(figsize=(10 * column_n,8 * row_n ),dpi=100)

for i, column in enumerate(continuous):
    plt.subplot(row_n,column_n,i+1)
    ax = sns.boxplot(y=forest[column], color='lightblue')
    ax.set(ylabel='')
    plt.title(column, fontdict=font_title)
```

```
plt.show()
```



1.1.6 *Variable Analysis*

Univariate Analysis

Statistical info

[81]:
```
forest.describe()
```

[81]:

	elevation	aspect	slope	horizontal_distance_to_hydrology	\
count	581,012.000	581,012.000	581,012.000	581,012.000	
mean	2,959.365	155.657	14.104	269.428	
std	279.985	111.914	7.488	212.549	
min	1,859.000	0.000	0.000	0.000	
25%	2,809.000	58.000	9.000	108.000	
50%	2,996.000	127.000	13.000	218.000	
75%	3,163.000	260.000	18.000	384.000	
max	3,858.000	360.000	66.000	1,397.000	

	vertical_distance_to_hydrology	horizontal_distance_to_roadways	\
count	581,012.000	581,012.000	
mean	46.419	2,350.147	
std	58.295	1,559.255	
min	-173.000	0.000	
25%	7.000	1,106.000	
50%	30.000	1,997.000	
75%	69.000	3,328.000	
max	601.000	7,117.000	

	hillshade_9am	hillshade_noon	hillshade_3pm	\
count	581,012.000	581,012.000	581,012.000	
mean	212.146	223.319	142.528	
std	26.770	19.769	38.275	
min	0.000	0.000	0.000	
25%	198.000	213.000	119.000	
50%	218.000	226.000	143.000	
75%	231.000	237.000	168.000	
max	254.000	254.000	254.000	

	horizontal_distance_to_fire_points	wilderness_area1	wilderness_area2	\
count	581,012.000	581,012.000	581,012.000	
mean	1,980.291	0.449	0.051	
std	1,324.195	0.497	0.221	
min	0.000	0.000	0.000	
25%	1,024.000	0.000	0.000	
50%	1,710.000	0.000	0.000	
75%	2,550.000	1.000	0.000	
max	7,173.000	1.000	1.000	

	wilderness_area3	wilderness_area4	soil_type1	soil_type2	soil_type3	\
count	581,012.000	581,012.000	581,012.000	581,012.000	581,012.000	
mean	0.436	0.064	0.005	0.013	0.008	
std	0.496	0.244	0.072	0.113	0.091	
min	0.000	0.000	0.000	0.000	0.000	
25%	0.000	0.000	0.000	0.000	0.000	
50%	0.000	0.000	0.000	0.000	0.000	

```
75%              1.000             0.000       0.000       0.000       0.000
max              1.000             1.000       1.000       1.000       1.000

       soil_type4  soil_type5  soil_type6  soil_type7  soil_type8  soil_type9  \
count 581,012.000 581,012.000 581,012.000 581,012.000 581,012.000 581,012.000
mean         0.021       0.003       0.011       0.000       0.000       0.002
std          0.144       0.052       0.106       0.013       0.018       0.044
min          0.000       0.000       0.000       0.000       0.000       0.000
25%          0.000       0.000       0.000       0.000       0.000       0.000
50%          0.000       0.000       0.000       0.000       0.000       0.000
75%          0.000       0.000       0.000       0.000       0.000       0.000
max          1.000       1.000       1.000       1.000       1.000       1.000

       soil_type10  soil_type11  soil_type12  soil_type13  soil_type14  \
count  581,012.000  581,012.000  581,012.000  581,012.000  581,012.000
mean         0.056        0.021        0.052        0.030        0.001
std          0.230        0.145        0.221        0.171        0.032
min          0.000        0.000        0.000        0.000        0.000
25%          0.000        0.000        0.000        0.000        0.000
50%          0.000        0.000        0.000        0.000        0.000
75%          0.000        0.000        0.000        0.000        0.000
max          1.000        1.000        1.000        1.000        1.000

       soil_type15  soil_type16  soil_type17  soil_type18  soil_type19  \
count  581,012.000  581,012.000  581,012.000  581,012.000  581,012.000
mean         0.000        0.005        0.006        0.003        0.007
std          0.002        0.070        0.077        0.057        0.083
min          0.000        0.000        0.000        0.000        0.000
25%          0.000        0.000        0.000        0.000        0.000
50%          0.000        0.000        0.000        0.000        0.000
75%          0.000        0.000        0.000        0.000        0.000
max          1.000        1.000        1.000        1.000        1.000

       soil_type20  soil_type21  soil_type22  soil_type23  soil_type24  \
count  581,012.000  581,012.000  581,012.000  581,012.000  581,012.000
mean         0.016        0.001        0.057        0.099        0.037
std          0.125        0.038        0.233        0.299        0.188
min          0.000        0.000        0.000        0.000        0.000
25%          0.000        0.000        0.000        0.000        0.000
50%          0.000        0.000        0.000        0.000        0.000
75%          0.000        0.000        0.000        0.000        0.000
max          1.000        1.000        1.000        1.000        1.000

       soil_type25  soil_type26  soil_type27  soil_type28  soil_type29  \
count  581,012.000  581,012.000  581,012.000  581,012.000  581,012.000
mean         0.001        0.004        0.002        0.002        0.198
std          0.029        0.067        0.043        0.040        0.399
```

```
min          0.000        0.000        0.000        0.000        0.000
25%          0.000        0.000        0.000        0.000        0.000
50%          0.000        0.000        0.000        0.000        0.000
75%          0.000        0.000        0.000        0.000        0.000
max          1.000        1.000        1.000        1.000        1.000

       soil_type30  soil_type31  soil_type32  soil_type33  soil_type34  \
count  581,012.000  581,012.000  581,012.000  581,012.000  581,012.000
mean         0.052        0.044        0.090        0.078        0.003
std          0.222        0.205        0.287        0.268        0.053
min          0.000        0.000        0.000        0.000        0.000
25%          0.000        0.000        0.000        0.000        0.000
50%          0.000        0.000        0.000        0.000        0.000
75%          0.000        0.000        0.000        0.000        0.000
max          1.000        1.000        1.000        1.000        1.000

       soil_type35  soil_type36  soil_type37  soil_type38  soil_type39  \
count  581,012.000  581,012.000  581,012.000  581,012.000  581,012.000
mean         0.003        0.000        0.001        0.027        0.024
std          0.057        0.014        0.023        0.162        0.152
min          0.000        0.000        0.000        0.000        0.000
25%          0.000        0.000        0.000        0.000        0.000
50%          0.000        0.000        0.000        0.000        0.000
75%          0.000        0.000        0.000        0.000        0.000
max          1.000        1.000        1.000        1.000        1.000

       soil_type40  cover_type
count  581,012.000 581,012.000
mean         0.015       2.051
std          0.122       1.397
min          0.000       1.000
25%          0.000       1.000
50%          0.000       2.000
75%          0.000       2.000
max          1.000       7.000
```

[95]:
```
continuous
```

[95]:
```
['elevation',
 'aspect',
 'slope',
 'horizontal_distance_to_hydrology',
 'vertical_distance_to_hydrology',
 'horizontal_distance_to_roadways',
 'hillshade_9am',
 'hillshade_noon',
 'hillshade_3pm',
```

```
 'horizontal_distance_to_fire_points']
```

Visualisation of continuous variables

- Used melt-facetgrid-map-distplot
- Elevation and hillshade_3pm varibales are similar to a normal distribution but other variables are not.

[365]:
```
# melt method melts the df and reconstruct all columns in value_vars list
# as a single column with corresponding values
# each variable in value vars are concatenated along y axis
f= pd.melt(forest, value_vars=continuous)
```

[369]:
```
f.head(5)
```

[369]:

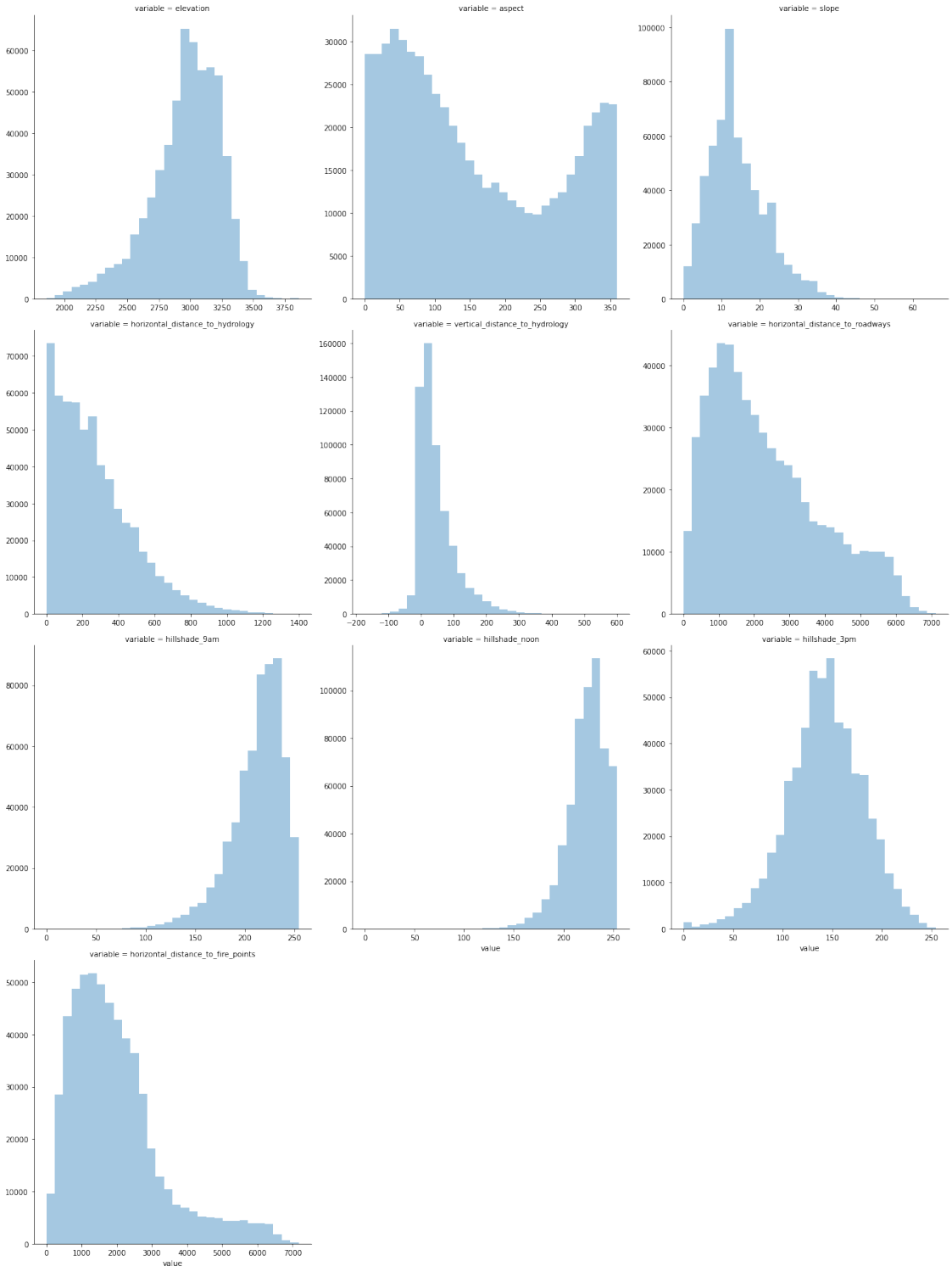
	variable	value
0	elevation	2596
1	elevation	2590
2	elevation	2804
3	elevation	2785
4	elevation	2595

[370]:
```
f.tail(5)
```

[370]:

	variable	value
5810115	horizontal_distance_to_fire_points	837
5810116	horizontal_distance_to_fire_points	845
5810117	horizontal_distance_to_fire_points	854
5810118	horizontal_distance_to_fire_points	864
5810119	horizontal_distance_to_fire_points	875

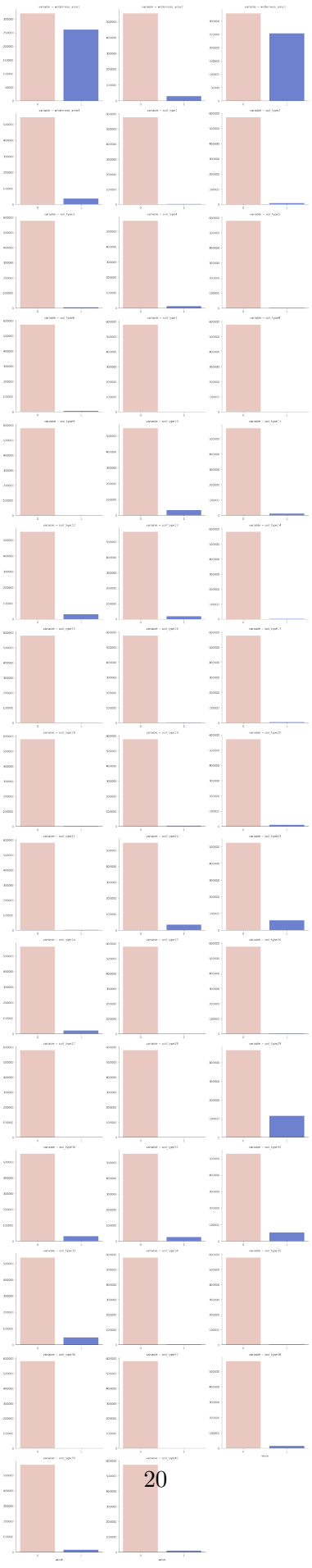
[367]:
```
# facetgrid allows to plot multiple graphs in a single figure with identified␣
 ↪column wrapping, height etc.
g = sns.FacetGrid(data=f, col='variable', col_wrap=3, sharex=False,␣
 ↪sharey=False, height=6)
# map method plots the desired graph identified in the first argument,
# other arguments are parameters for the first plotting method
g = g.map(sns.distplot, 'value', kde=False, bins=30)
```



Visualisation of categorical variables

- Used melt-facetgrid-map-countplot.
- Cover types 1 and 2 are dominant types in the forest. It may be needed to resample the data for overcome the unbalancy. (***See Developing the model***)
- It is obvious that some of soil types are negligible. (***See Feature Engineering***)

[395]:
```
f = pd.melt(forest, value_vars=categoric)
g = sns.FacetGrid(f, col='variable', col_wrap=3, sharex=False, sharey=False,␣
 ↪height=5)
g = g.map(sns.countplot,'value', palette=binary_palette)
```

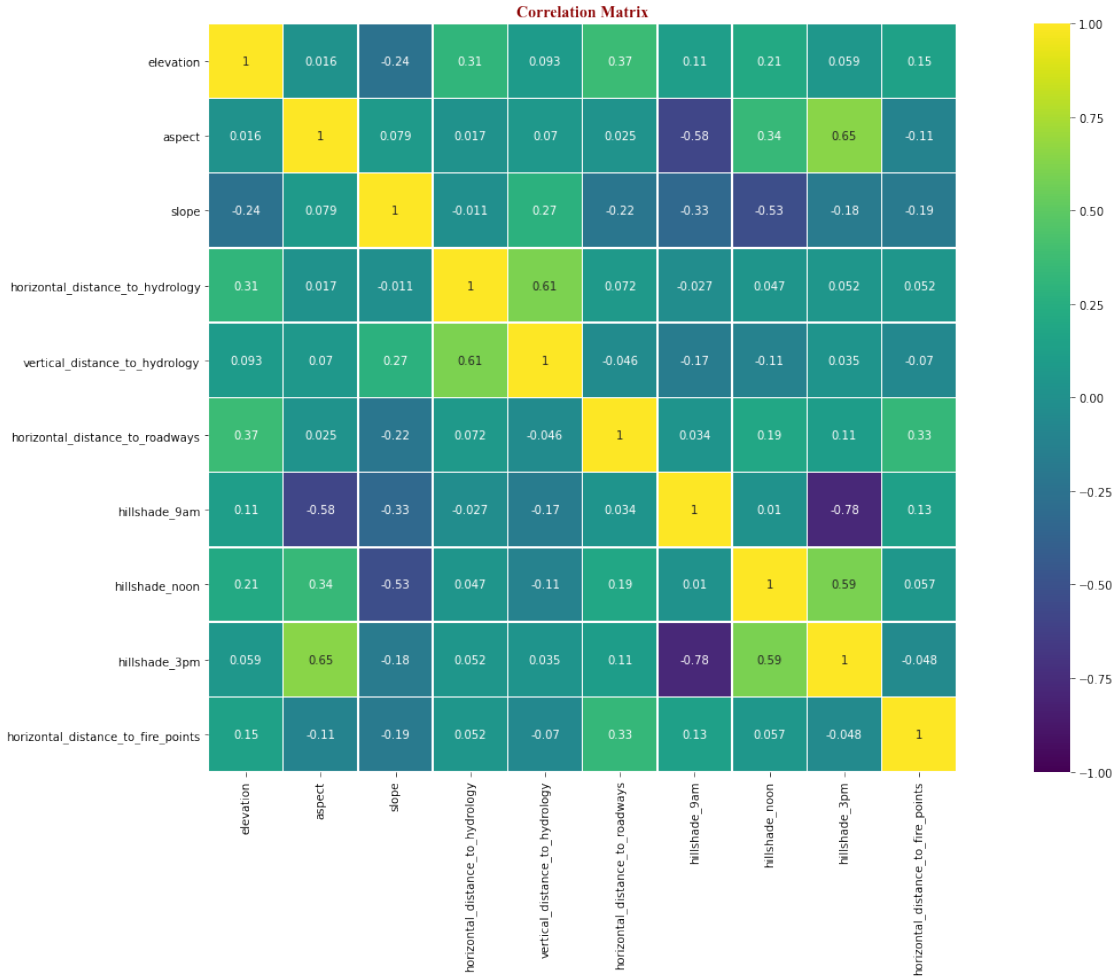


Multivariate analysis

Correlation

- Inspect the correlation of continuous variables with heatmap.
- hillshade_3pm and hillshade_9am are highly negative correlation, one of them can be droppped. (***See Feature Engineering***)

[97]:
```
# continuous variables are correlated
corr_matrix = forest[continuous].corr()

# plot the matrix with heatmap
plt.figure(figsize=(24,12))
sns.heatmap(corr_matrix,square=True, annot=True, linewidths=.5, vmin=-1,␣
↪vmax=1, cmap='viridis')
plt.title("Correlation Matrix", fontdict=font_title)

plt.show()
```



[167]:
```
# detect the corrolated variables
corr_tuples=[]
for col in corr_matrix.columns:
    if corr_matrix.loc[(corr_matrix[col]>0.5)&(corr_matrix[col]<1)].shape[0] >␣
↪0:
        c = corr_matrix.loc[(corr_matrix[col]>0.5)&(corr_matrix[col]<1)].
↪index[0]
        if {col,c} not in corr_tuples:
            corr_tuples.append({col,c})
    if corr_matrix.loc[(corr_matrix[col]<-0.5)].shape[0] > 0:
        c = corr_matrix.loc[corr_matrix[col]<-0.5].index[0]
        if {col,c} not in corr_tuples:
            corr_tuples.append({col,c})
```

[184]:
```
list(corr_tuples)
```

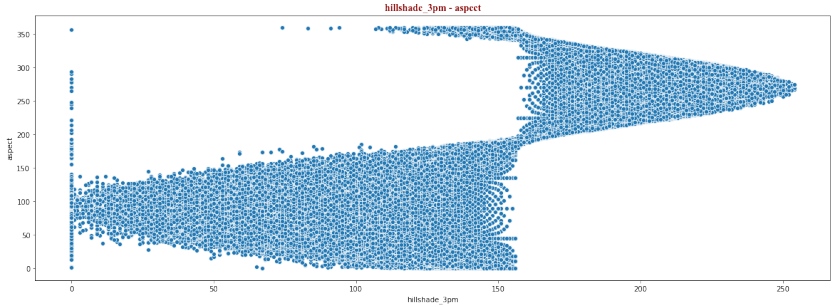
[184]:
```
[{'aspect', 'hillshade_3pm'},
 {'aspect', 'hillshade_9am'},
 {'hillshade_noon', 'slope'},
 {'horizontal_distance_to_hydrology', 'vertical_distance_to_hydrology'},
 {'hillshade_3pm', 'hillshade_noon'},
 {'hillshade_3pm', 'hillshade_9am'}]
```

Analysis of Mostly Correlated Continuous Variables

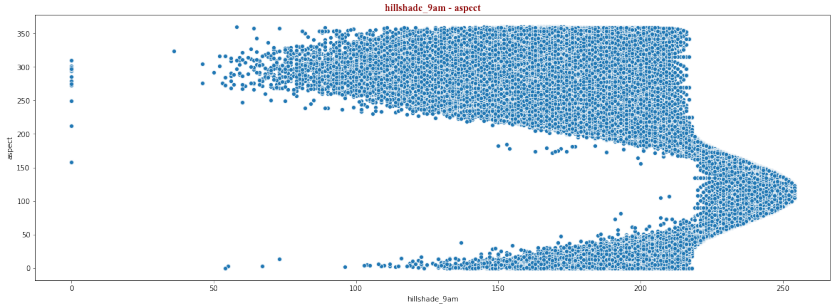
- Anlaysis of mostly correlated and reverse correlated variables with scatterplot.
- Points in the plots go from lower left corner to upper right corner like a diagonal in strongly positively correlated variables like horizontal_distance_to_hydrology - vertical_distance_to_hydrology.
- horizontal_distance_to_hydrology and vertical_distance_to_hydrology can be merged as euclidien distance to hydrology.(***See Feature Engineering***)

[177]:
```
# identify the number of columns
column_n = 1
# calculate the number fo rows
row_n = np.ceil(len(corr_tuples)/column_n)
# calculate the figure size
plt.figure(figsize=(20 * column_n, 8 * row_n ))

for i, column in enumerate(corr_tuples):
    plt.subplot(row_n,column_n,i+1)
    ax = sns.scatterplot(data=forest, x=list(corr_tuples[i])[0],␣
 ↪y=list(corr_tuples[i])[1])
    #ax.set(ylabel='')
    plt.title(str(list(corr_tuples[i])[0])+' - '+str(list(corr_tuples[i])[1]),␣
 ↪fontdict=font_title)

plt.show()
```

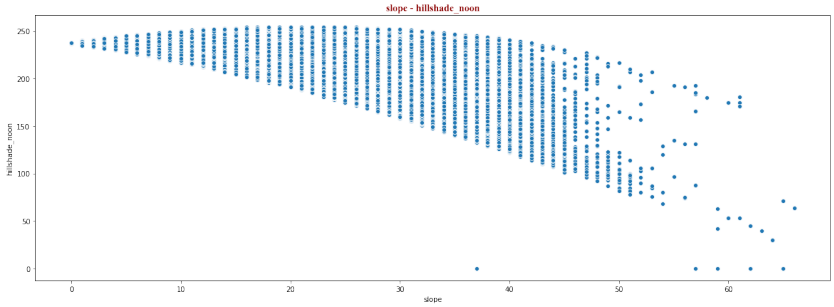
hillshadc_3pm - aspect



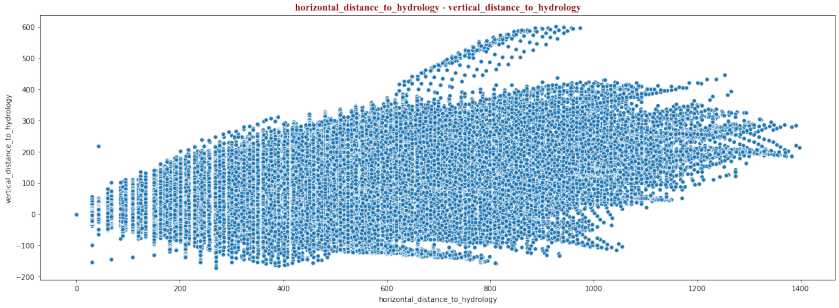
hillshadc_9am - aspect



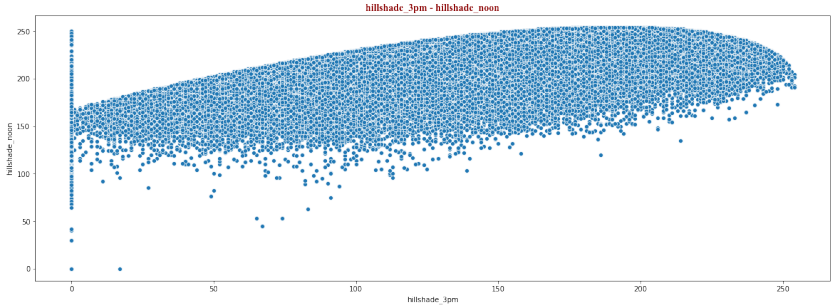
slope - hillshadc_noon



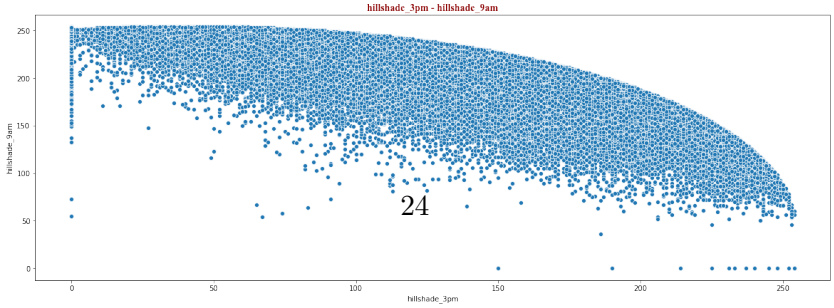
horizontal_distance_to_hydrology - vertical_distance_to_hydrology



hillshadc_3pm - hillshadc_noon



hillshadc_3pm - hillshadc_9am



Multivariate Analysis of Categoric Variables

- Plotted the countplots with the target variable cover_type
- There are much more trees in wilderness area 1 and 3 than wilderness area 2 and 4.
- More types of trees grow in Wilderness area 3 than 1.
- As seen in figure some of the soil types are very rare and negligible.(***See Feature Engineering***)

[192]:
```
categoric.remove('cover_type')
```

[396]:
```
f = pd.melt(forest, id_vars='cover_type', value_vars=categoric)
```

[397]:
```
def countplot(value, hue, **kwargs):
    sns.countplot(x=value, hue=hue, palette=my_palette)

g = sns.FacetGrid(f, col="variable",  col_wrap=3, sharex=False, sharey=False,␣
 ↪height=5, aspect=1.2)
g = g.map(countplot, "value", "cover_type")
g = g.add_legend(title='cover_type', loc='top')
```


variable = wilderness_area1
variable = wilderness_area2
variable = wilderness_area3
variable = wilderness_area4
variable = soil_type1
variable = soil_type2
variable = soil_type3
variable = soil_type4
variable = soil_type5
variable = soil_type6
variable = soil_type7
variable = soil_type8
variable = soil_type9
variable = soil_type10
variable = soil_type11
variable = soil_type12
variable = soil_type13
variable = soil_type14
variable = soil_type15
variable = soil_type16
variable = soil_type17
variable = soil_type18
variable = soil_type19
variable = soil_type20
variable = soil_type21
variable = soil_type22
variable = soil_type23
variable = soil_type24
variable = soil_type25
variable = soil_type26
variable = soil_type27
variable = soil_type28
variable = soil_type29
variable = soil_type30
variable = soil_type31
variable = soil_type32
variable = soil_type33
variable = soil_type34
variable = soil_type35
variable = soil_type36
variable = soil_type37
variable = soil_type38
variable = soil_type39
variable = soil_type40
count_Cover_Type
value

Soil type analysis

- Calculated the total tree counts of soil types and plotted with plotly.
- Plotly provides information about data when hovering over the points. Thus, more fertile soil types can be detected easily.
- Soil types 29,23,32 and 33 are most productive soil types.
- When zoomed in the bottom of the graph, the least productive soil types can be seen.

[295]:
```
# list compeherension for soil type names
soil_types = ['soil_type'+str(i) for i in range(1,41)]

# calculate the sum of soil types
soil_type_sums = {}
for soil in soil_types:
    soil_type_sums[soil]=forest[soil].sum()
# create soil type df
soil_df = pd.DataFrame([soil_type_sums],index=['sum']).T
```

[313]:
```
soil_df.head(2)
```

[313]:
```
              sum
soil_type1   3031
soil_type2   7525
```

- Inspect the most productive soil types.

[316]:
```
soil_df.sort_values(by='sum').tail(10)
```

[316]:
```
                sum
soil_type24   21278
soil_type31   25666
soil_type12   29971
soil_type30   30170
soil_type10   32634
soil_type22   33373
soil_type33   45154
soil_type32   52519
soil_type23   57752
soil_type29  115247
```

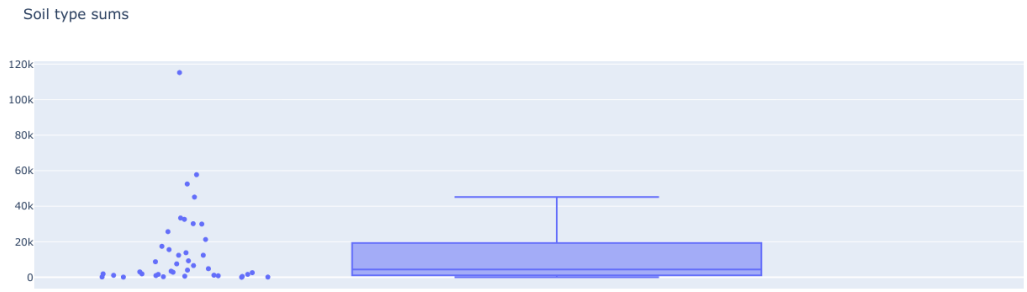
- Inspect the least productive soil types.

[317]:
```
soil_df.sort_values(by='sum').head(15)
```

[317]:

	sum
soil_type15	3
soil_type7	105
soil_type36	119
soil_type8	179
soil_type37	298
soil_type25	474
soil_type14	599
soil_type21	838
soil_type28	946
soil_type27	1086
soil_type9	1147
soil_type5	1597
soil_type34	1611
soil_type35	1891
soil_type18	1899

[362]:
```
data = [go.Box(y=soil_df['sum'], name='', boxpoints='all', jitter=0.5,␣
 ↪pointpos=-1.8, text=soil_df.index)]
layout = go.Layout(title='Soil type sums',
                   hovermode='closest')
fig=go.Figure(data=data, layout=layout)
fig.show()
```



Wilderness area analysis

- Calculated the total tree counts of wilderness areas and plotted with plotly.
- Wilderness area 1 and 3 houses almost all of the trees as seen in figure.

[318]:
```
areas = ['wilderness_area'+str(i) for i in range(1,5)]
```

[319]:
```
# calculate the sum of soil types
area_sums = {}
```

```
for area in areas:
    area_sums[area]=forest[area].sum()
# create soil type df
area_df = pd.DataFrame([area_sums],index=['sum']).T
```

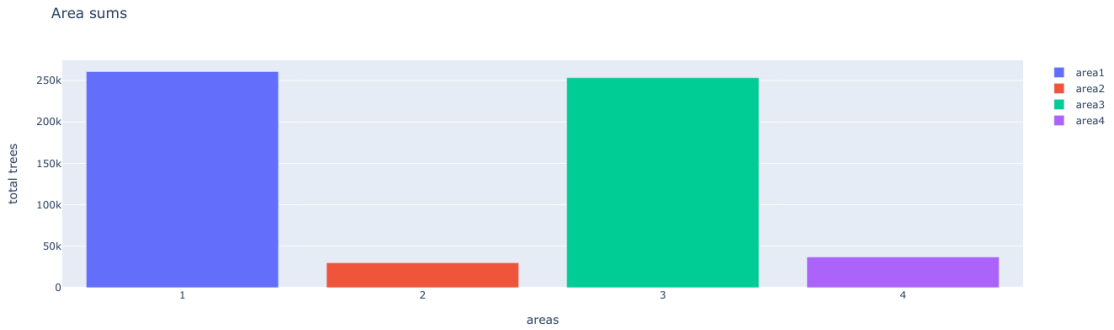
[321]:
```
area_df
```

[321]:
```
                      sum
wilderness_area1  260796
wilderness_area2   29884
wilderness_area3  253364
wilderness_area4   36968
```

[348]:
```
area_df.loc['wilderness_area1'].name
```

[348]:
```
'wilderness_area1'
```

[361]:
```
# used list compherension for creating the traces
data = [go.Bar(x=[i+1],
               y=area_df.loc[area],
               name=area_df.loc[area].name.split('_')[1]) for i, area in␣
 ↪enumerate(areas)]
layout = go.Layout(title='Area sums',
                   yaxis={'title':'total trees'},
                   xaxis={'title':'areas', 'tick0':1, 'dtick':1})
fig=go.Figure(data=data, layout=layout)
fig.show()
```

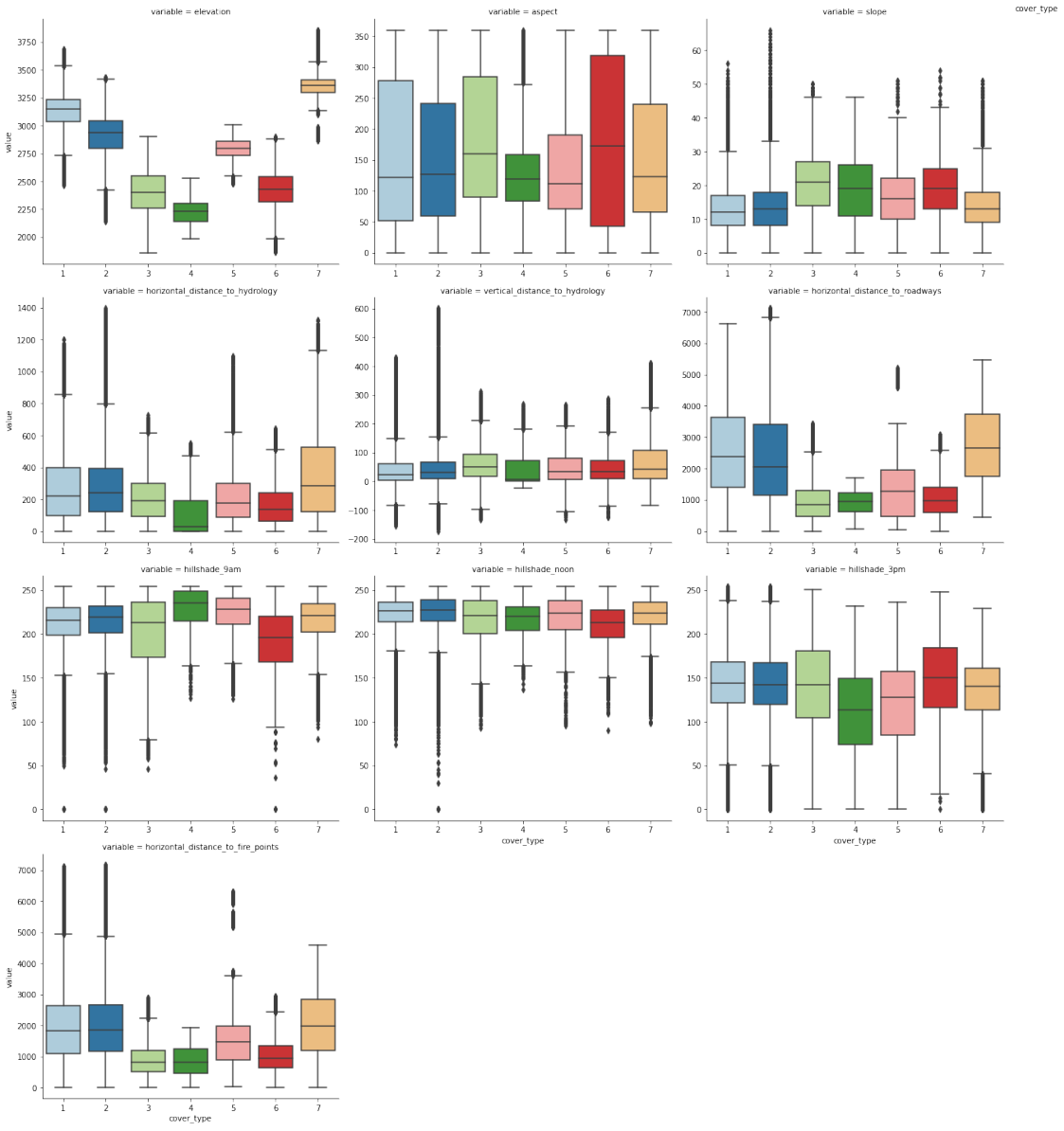


Multivariate Analysis of Continuous Variables

- Plotted the box with the target variable cover_type.
- Elevation has a variance between different types of trees, the others has very little variances.

[388]:
```
f = pd.melt(forest, id_vars='cover_type', value_vars=continuous)
```

[389]:
```
g = sns.FacetGrid(f, col="variable",  col_wrap=3, sharex=False, sharey=False,␣
↪height=5, aspect=1.2)
g = g.map(sns.boxplot, "cover_type", "value", palette=sns.
↪color_palette("Paired"))
g = g.add_legend(title='cover_type', loc='top')
```



Feature engineering - Extraction and Selection

- Select and extract features using MySQL database and SQL queries.

Feature Extraction SQL

- Create a new column by calculating the euclidean distance to hydrology with horizontal_distance_to_hydrology and vertical_distance_to_hydrology with SQL.
- Connect to DB.

```
mysql -u ahmet -p
```

- Create another duplicate table for transforming data.

```
mysql> CREATE TABLE covtype_tf AS SELECT * FROM covtype;
```

- Calculate euclidean distance and write the result to a new table.

```
mysql> CREATE TABLE euclidean SELECT SQRT((POW(horizontal_distance_to_hydrology,2)+
POW(vertical_distance_to_hydrology,2))) AS euclidean_distance_to_hydrology FROM
covtype_tf;
```

- Add incremental unique ids to the two table for joining purpose.

```
mysql> ALTER TABLE covtype_tf ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT,
ADD INDEX (id);          mysql> ALTER TABLE euclidean ADD id INT UNSIGNED NOT NULL
AUTO_INCREMENT, ADD INDEX (id);
```

- Create a table from the join result of two tables.

```
mysql> CREATE TABLE covtype_euclidean AS SELECT elevation, aspect,
slope, horizontal_distance_to_hydrology, vertical_distance_to_hydrology,
horizontal_distance_to_roadways, hillshade_9am, hillshade_noon, hillshade_3pm,
horizontal_distance_to_fire_points, wilderness_area1, wilderness_area2,
wilderness_area3, wilderness_area4, soil_type1, soil_type2, soil_type3,
soil_type4, soil_type5, soil_type6, soil_type7, soil_type8, soil_type9,
soil_type10, soil_type11, soil_type12, soil_type13, soil_type14, soil_type15,
soil_type16, soil_type17, soil_type18, soil_type19, soil_type20, soil_type21,
soil_type22, soil_type23, soil_type24, soil_type25, soil_type26, soil_type27,
soil_type28, soil_type29, soil_type30, soil_type31, soil_type32, soil_type33,
soil_type34, soil_type35, soil_type36, soil_type37, soil_type38, soil_type39,
soil_type40, euclidean_distance_to_hydrology, cover_type  FROM covtype_tf INNER
JOIN euclidean ON covtype_tf.id = euclidean.id;
```

Pandas

[458]:
```
# calculate euclidean distance with pandas and numpy
forest['euclidean_distance_to_hydrology'] = np.
↪sqrt(forest['horizontal_distance_to_hydrology'].pow(2)
                                                    ␣
↪+forest['vertical_distance_to_hydrology'].pow(2))
```

[460]:
```
forest.head(1)
```

[460]:
```
   elevation  aspect  slope  horizontal_distance_to_hydrology  \
0       2596      51      3                               258
```

```
   vertical_distance_to_hydrology  horizontal_distance_to_roadways  \
0                               0                              510

   hillshade_9am  hillshade_noon  hillshade_3pm  \
0            221             232            148

   horizontal_distance_to_fire_points  wilderness_area1  wilderness_area2  \
0                                6279                 1                 0

   wilderness_area3  wilderness_area4  soil_type1  soil_type2  soil_type3  \
0                 0                 0           0           0           0

   soil_type4  soil_type5  soil_type6  soil_type7  soil_type8  soil_type9  \
0           0           0           0           0           0           0

   soil_type10  soil_type11  soil_type12  soil_type13  soil_type14  \
0            0            0            0            0            0

   soil_type15  soil_type16  soil_type17  soil_type18  soil_type19  \
0            0            0            0            0            0

   soil_type20  soil_type21  soil_type22  soil_type23  soil_type24  \
0            0            0            0            0            0

   soil_type25  soil_type26  soil_type27  soil_type28  soil_type29  \
0            0            0            0            0            1

   soil_type30  soil_type31  soil_type32  soil_type33  soil_type34  \
0            0            0            0            0            0

   soil_type35  soil_type36  soil_type37  soil_type38  soil_type39  \
0            0            0            0            0            0

   soil_type40  cover_type  euclidean_distance_to_hydrology
0            0           5                         258.000
```

Feature Selection

- Drop hillshade_3pm which is highly correlated with hillshade_9am
- Drop redundant horizontal_distance_to_hydrology and vertical_distance_to_hydrology
- Drop soil_type columns with less than 1500 trees.

SQL

```
mysql> ALTER TABLE covtype_euclidean     -> DROP COLUMN hillshade_3pm,
-> DROP COLUMN horizontal_distance_to_hydrology,     -> DROP COLUMN
vertical_distance_to_hydrology,     -> DROP COLUMN soil_type7,     -> DROP COLUMN
```

```
soil_type8,     -> DROP COLUMN soil_type9,     -> DROP COLUMN soil_type14,     ->
DROP COLUMN soil_type15,     -> DROP COLUMN soil_type21,     -> DROP COLUMN
soil_type25,     -> DROP COLUMN soil_type27,     -> DROP COLUMN soil_type28,
-> DROP COLUMN soil_type36,     -> DROP COLUMN soil_type37;
```

Pandas

[461]:
```
# list compeherension for soil type names
soil_types = ['soil_type'+str(i) for i in range(1,41)]

# create soil type list to be removed
cols_remove = ['hillshade_3pm', 'horizontal_distance_to_hydrology',␣
↪'vertical_distance_to_hydrology']
for soil in soil_types:
    if (forest[soil].sum()<1500):
        cols_remove.append(soil)
cols_remove
```

[461]:
```
['hillshade_3pm',
 'horizontal_distance_to_hydrology',
 'vertical_distance_to_hydrology',
 'soil_type7',
 'soil_type8',
 'soil_type9',
 'soil_type14',
 'soil_type15',
 'soil_type21',
 'soil_type25',
 'soil_type27',
 'soil_type28',
 'soil_type36',
 'soil_type37']
```

Reading SQL data after data transformation

[38]:
```
from sqlalchemy import create_engine

# engine definition; user, password and db info provided
engine = create_engine('mysql+mysqlconnector://ahmet:Project1234!@localhost/
↪tree')

# read sql table fater transformation
forest = pd.read_sql_table('covtype_euclidean', engine)
```

1.2 *Developing a predictive model*

[258]:
```
# classifiers
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import xgboost as xgb
# pipeline meta model
from sklearn.pipeline import Pipeline
# split train data
from sklearn.model_selection import train_test_split
# imbalancy elimination
from imblearn.over_sampling import ADASYN
# metrics
from sklearn.metrics import mean_squared_error, classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score
# cross validation
from sklearn.model_selection import cross_validate, cross_val_score
# hyperparameter tuning
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

1.2.1 *Split Train and Test Data*

[462]:
```
# target variable
y = forest['cover_type']

# explaining variables (cover_type has dropped from categoric variable list)
# for pandas version
# X = forest.drop(cols_remove+['cover_type'], axis=1)
# for sql version
X = forest.drop(['cover_type'], axis=1)

# split the train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

1.2.2 *KNN Model*

[469]:
```
%%time
# to measure the duration of cell execution
# create a basic knn model
knn_cl = KNeighborsClassifier(n_neighbors=5, weights='uniform')
# train the model
knn_cl.fit(X_train, y_train)
```

```
CPU times: user 4.99 s, sys: 48.1 ms, total: 5.04 s
Wall time: 5.02 s
```

[469]:
```
KNeighborsClassifier()
```

[470]:
```
# accuracy score of the model
accuracy = knn_cl.score(X_test, y_test)
# predict
y_pred = knn_cl.predict(X_test)
# f1 score, weighted for unbalanced data
f1 = f1_score(y_test,y_pred,average='weighted')
```

[471]:
```
print('Accuracy score {:.4f}'.format(accuracy))
print('F1 score       {:.4f}'.format(f1))
```

```
Accuracy score 0.9661
F1 score       0.9661
```

1.2.3 *Logistic Regression Model*

[472]:
```
%%time
# to measure the duration of cell execution
# create a basic model
log_cl = LogisticRegression()
# train the model
log_cl.fit(X_train, y_train)
```

```
CPU times: user 2min 3s, sys: 1min 3s, total: 3min 6s
Wall time: 45 s
```

[472]:
```
LogisticRegression()
```

[473]:
```
# accuracy score of the model
accuracy = log_cl.score(X_test, y_test)
# predict
y_pred = log_cl.predict(X_test)
# f1 score, weighted for unbalanced data
f1 = f1_score(y_test,y_pred,average='weighted')
```

[474]:
```
print('Accuracy score {:.4f}'.format(accuracy))
print('F1 score       {:.4f}'.format(f1))
```

```
Accuracy score 0.6081
F1 score       0.5684
```

1.2.4 *Decision Tree Model*

[475]:
```
%%time
# to measure the duration of cell execution
# create a basic decision tree model
dt_cl = DecisionTreeClassifier()
# train the model
dt_cl.fit(X_train, y_train)
```

```
CPU times: user 7.77 s, sys: 12.1 ms, total: 7.78 s
Wall time: 7.79 s
```

[475]:
```
DecisionTreeClassifier()
```

[476]:
```
# accuracy score of the model
accuracy = dt_cl.score(X_test, y_test)
# predict
y_pred = dt_cl.predict(X_test)
# f1 score, weighted for unbalanced data
f1 = f1_score(y_test,y_pred,average='weighted')
```

[477]:
```
print('Accuracy score {:.4f}'.format(accuracy))
print('F1 score       {:.4f}'.format(f1))
```

```
Accuracy score 0.9384
F1 score       0.9383
```

1.2.5 *Random Forest Model*

[479]:
```
%%time
# to measure the duration of cell execution
# create a basic model
rf_cl = RandomForestClassifier()
# train the model
rf_cl.fit(X_train, y_train)
```

```
CPU times: user 1min 58s, sys: 427 ms, total: 1min 59s
Wall time: 1min 59s
```

[479]:
```
RandomForestClassifier()
```

[480]:
```
# accuracy score of the model
accuracy = rf_cl.score(X_test, y_test)
# predict
y_pred = rf_cl.predict(X_test)
# f1 score, weighted for unbalanced data
f1 = f1_score(y_test,y_pred,average='weighted')
```

[481]:
```
print('Accuracy score {:.4f}'.format(accuracy))
print('F1 score       {:.4f}'.format(f1))
```

```
Accuracy score 0.9566
F1 score       0.9565
```

1.2.6 *XGBoost Model*

[485]:
```
%%time
# to measure the duration of cell execution
# xgboost classification model with objective paramater reg:logistic
xgb_cl = XGBClassifier(objective ='reg:logistic', max_depth=6)
# train the model
xgb_cl.fit(X_train, y_train)
```

```
CPU times: user 14min 52s, sys: 208 ms, total: 14min 52s
Wall time: 3min 44s
```

[485]:
```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method=None, validate_parameters=False, verbosity=None)
```

[486]:
```
# accuracy score of the model
accuracy = xgb_cl.score(X_test, y_test)
# predict
y_pred = xgb_cl.predict(X_test)
# f1 score, weighted for unbalanced data
f1 = f1_score(y_test,y_pred,average='weighted')
```

[487]:
```
print('Accuracy score {:.4f}'.format(accuracy))
print('F1 score       {:.4f}'.format(f1))
```

```
Accuracy score 0.8703
F1 score       0.8697
```

1.2.7 *LightGBM Model*

[482]:
```
%%time
# to measure the duration of cell execution
# lightgbm classification model with default parameters
lgb_cl = LGBMClassifier(max_depth=-1, n_jobs=4)
# train the model
lgb_cl.fit(X_train, y_train)
```

```
CPU times: user 38.4 s, sys: 160 ms, total: 38.6 s
Wall time: 9.98 s
```

[482]:
```
LGBMClassifier(n_jobs=4)
```

[483]:
```
# accuracy score of the model
accuracy = lgb_cl.score(X_test, y_test)
# predict
y_pred = lgb_cl.predict(X_test)
# f1 score, weighted for unbalanced data
f1 = f1_score(y_test,y_pred,average='weighted')
```

[484]:
```
print('Accuracy score {:.4f}'.format(accuracy))
print('F1 score       {:.4f}'.format(f1))
```

```
Accuracy score 0.8326
F1 score       0.8316
```

1.2.8 *Model Selection with Pipeline*

- Pipeline class is like a meta model and can be used to test our model performances to choose the best model.
- The most commonly used metric for imbalanced datasets in the multiclass setting is the multiclass version of the f-score:
  - `f1_weighted` computes the mean of the per-class f-scores, weighted by their support.
- In the cross validation section Logistic regression model removed because of performance results.

[496]:
```
%%time
# to measure the duration of cell execution

# pipeline definition
# knn for the baseline model
pipe = Pipeline([('classifier', KNeighborsClassifier())])

# define the models with corresponding parameters
```

```
param_grid = [{'classifier': [KNeighborsClassifier()],
               'classifier__n_neighbors': [4,5],                # default 5
               'classifier__weights': ['uniform', 'distance']}, # default␣
 ↪uniform
              {'classifier': [DecisionTreeClassifier()],
               'classifier__max_depth': [None, 4]},     # default None
              {'classifier': [RandomForestClassifier()],
               'classifier__max_depth': [None, 4]},     # default None
              {'classifier': [XGBClassifier()],
               'classifier__reg_alpha': [0, 10.0],      # default 0
               'classifier__reg_lambda': [0, 10.0],     # default 0
               'classifier__max_depth': [5, 6]},        # default 6
              {'classifier': [LGBMClassifier()],
               'classifier__reg_alpha': [0, 10.0],  # default 0
               'classifier__reg_lambda': [0, 10.0], # default 0
               'classifier__max_depth': [-1, 5]}    # default -1(no limit)
             ]
# define gridsearch object
grid_cl = GridSearchCV(estimator=pipe,
                       param_grid=param_grid,
                       scoring='f1_weighted',
                       cv=5, n_jobs=4, return_train_score=True)
# train the model
grid_cl.fit(X_train, y_train)

# results
print("Best params: \n{}\n".format(grid_cl.best_params_))
print("Best cross-validation score: \n{:.2f}\n".format(grid_cl.best_score_))
print("Test-set score: \n{:.2f}\n".format(grid_cl.score(X_test, y_test)))
```

```
Best params:
{'classifier': KNeighborsClassifier(n_neighbors=4, weights='distance'),
'classifier__n_neighbors': 4, 'classifier__weights': 'distance'}

Best cross-validation score:
0.96

Test-set score:
0.97

CPU times: user 15.3 s, sys: 1.32 s, total: 16.6 s
Wall time: 2h 19min 17s
```

[497]:
```
# save the results to the df
grid_cl_df = pd.DataFrame(grid_cl.cv_results_)
grid_cl_df.head(1)
```

[497]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0          6.471         0.656           10.275           0.631

                                        param_classifier  \
0  KNeighborsClassifier(n_neighbors=4, weights='distance')

  param_classifier__n_neighbors param_classifier__weights  \
0                             4                   uniform

  param_classifier__max_depth param_classifier__reg_alpha  \
0                         NaN                         NaN

  param_classifier__reg_lambda  \
0                          NaN

                params  \
0  {'classifier': KNeighborsClassifier(n_neighbors=4, weights='distance'),
'classifier__n_neighbors…

   split0_test_score  split1_test_score  split2_test_score  split3_test_score  \
0              0.958              0.957              0.958              0.958

   split4_test_score  mean_test_score  std_test_score  rank_test_score  \
0              0.956            0.957           0.001                4

   split0_train_score  split1_train_score  split2_train_score  \
0               0.978               0.978               0.978

   split3_train_score  split4_train_score  mean_train_score  std_train_score
0               0.978               0.978             0.978            0.000
```

[498]:
```
# sort
grid_cl_df = grid_cl_df.sort_values(by='mean_test_score', ascending=False)
# show the best ones
grid_cl_df.loc[:,['param_classifier','mean_train_score', 'mean_test_score']].
 ↪head(5)
```

[498]:
```
                                        param_classifier  mean_train_score  \
1  KNeighborsClassifier(n_neighbors=4, weights='distance')             1.000
3  KNeighborsClassifier(n_neighbors=4, weights='distance')             1.000
2  KNeighborsClassifier(n_neighbors=4, weights='distance')             0.980
0  KNeighborsClassifier(n_neighbors=4, weights='distance')             0.978
6                              RandomForestClassifier()                1.000

   mean_test_score
1            0.964
3            0.962
```

```
2            0.960
0            0.957
6            0.951
```

[501]:
```
# create another df for best results of each classifiers
best_of_classifiers = pd.DataFrame([])
for classifier in grid_cl_df.param_classifier.unique():
    best_of_classifiers = pd.
 ↪concat([best_of_classifiers,grid_cl_df[grid_cl_df['param_classifier']==classifier].
 ↪head(1)])
```

[502]:
```
best_of_classifiers
```

[502]:
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
1           4.982         0.113            6.663           0.301
6         121.491         9.038            5.133           0.176
4           7.432         0.074            0.101           0.003
14        804.587         8.629            5.543           0.170
16         31.410         0.806            3.484           0.209

       param_classifier  \
1
KNeighborsClassifier(n_neighbors=4, weights='distance')
6
RandomForestClassifier()
4
DecisionTreeClassifier()
14  XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,\n
colsample_by…
16
LGBMClassifier()

   param_classifier__n_neighbors param_classifier__weights  \
1                              4                  distance
6                            NaN                       NaN
4                            NaN                       NaN
14                           NaN                       NaN
16                           NaN                       NaN

   param_classifier__max_depth param_classifier__reg_alpha  \
1                          NaN                         NaN
6                         None                         NaN
4                         None                         NaN
14                           6                      10.000
16                          -1                           0

   param_classifier__reg_lambda  \
```

```
1                          NaN
6                          NaN
4                          NaN
14                           0
16                           0

                params  \
1   {'classifier': KNeighborsClassifier(n_neighbors=4, weights='distance'),
'classifier__n_neighbors…
6                             {'classifier': RandomForestClassifier(),
'classifier__max_depth': None}
4                             {'classifier': DecisionTreeClassifier(),
'classifier__max_depth': None}
14  {'classifier': XGBClassifier(base_score=None, booster=None,
colsample_bylevel=None,
          …
16  {'classifier': LGBMClassifier(), 'classifier__max_depth': -1,
'classifier__reg_alpha': 0, 'class…

    split0_test_score  split1_test_score  split2_test_score  \
1               0.964              0.963              0.964
6               0.951              0.951              0.951
4               0.931              0.933              0.932
14              0.871              0.870              0.867
16              0.831              0.829              0.829

    split3_test_score  split4_test_score  mean_test_score  std_test_score  \
1               0.964              0.963            0.964           0.001
6               0.952              0.951            0.951           0.001
4               0.931              0.931            0.932           0.001
14              0.874              0.873            0.871           0.002
16              0.832              0.833            0.831           0.001

    rank_test_score  split0_train_score  split1_train_score  \
1                 1               1.000               1.000
6                 5               1.000               1.000
4                 6               1.000               1.000
14                7               0.882               0.882
16               15               0.837               0.837

    split2_train_score  split3_train_score  split4_train_score  \
1                1.000               1.000               1.000
6                1.000               1.000               1.000
4                1.000               1.000               1.000
14               0.881               0.883               0.884
16               0.837               0.837               0.838
```

```
    mean_train_score  std_train_score
1              1.000            0.000
6              1.000            0.000
4              1.000            0.000
14             0.882            0.001
16             0.837            0.001
```

[503]:
```
# for plots naming must be done
def get_names(model_df):
    name_list = []
    for i in model_df.index:
        classifier = model_df.loc[i,'param_classifier']
        name = classifier.__class__.__name__
        if (name == 'SVC'):
            name = name + '_' + classifier.kernel
        elif (name == 'LogisticRegression'):
            name = name.split('ression')[0]
            name = name + '_' + classifier.solver
        else:
            name = name.split('assifier')[0]

        name_list.append(name)

    return name_list

print(get_names(best_of_classifiers))
```

```
['KNeighborsCl', 'RandomForestCl', 'DecisionTreeCl', 'XGBCl', 'LGBMCl']
```

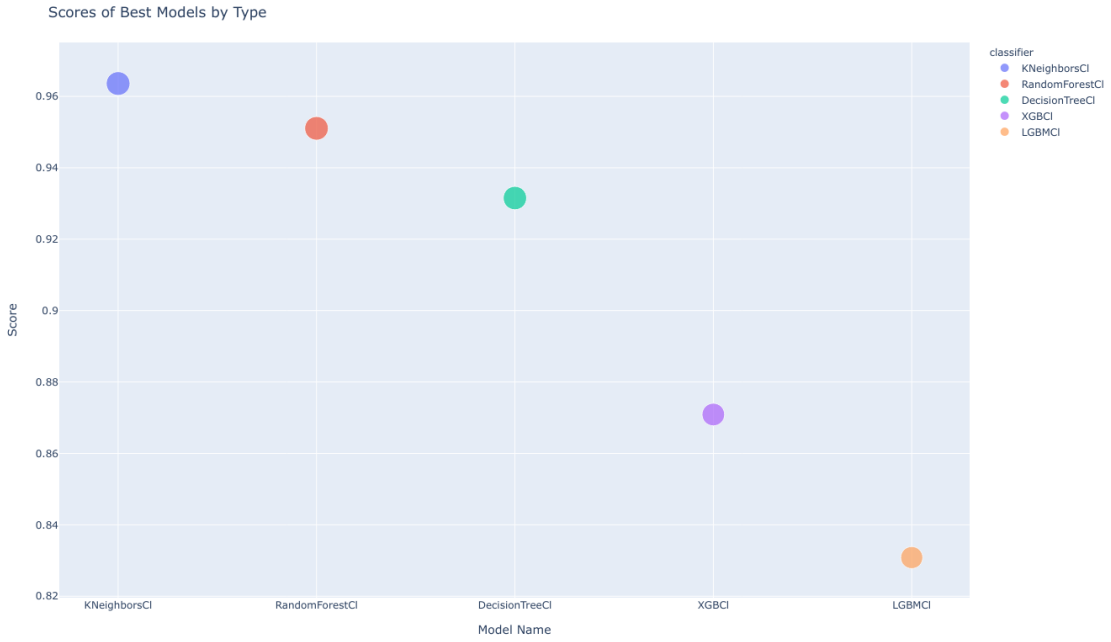
[504]:
```
# identify the classifier name
best_of_classifiers['classifier'] = get_names(best_of_classifiers)
```

[505]:
```
# inspect the score value of each classifier
best_of_classifiers.loc[:,['classifier','mean_train_score', 'mean_test_score']]
```

[505]:
```
        classifier  mean_train_score  mean_test_score
1     KNeighborsCl             1.000            0.964
6   RandomForestCl             1.000            0.951
4   DecisionTreeCl             1.000            0.932
14           XGBCl             0.882            0.871
16          LGBMCl             0.837            0.831
```

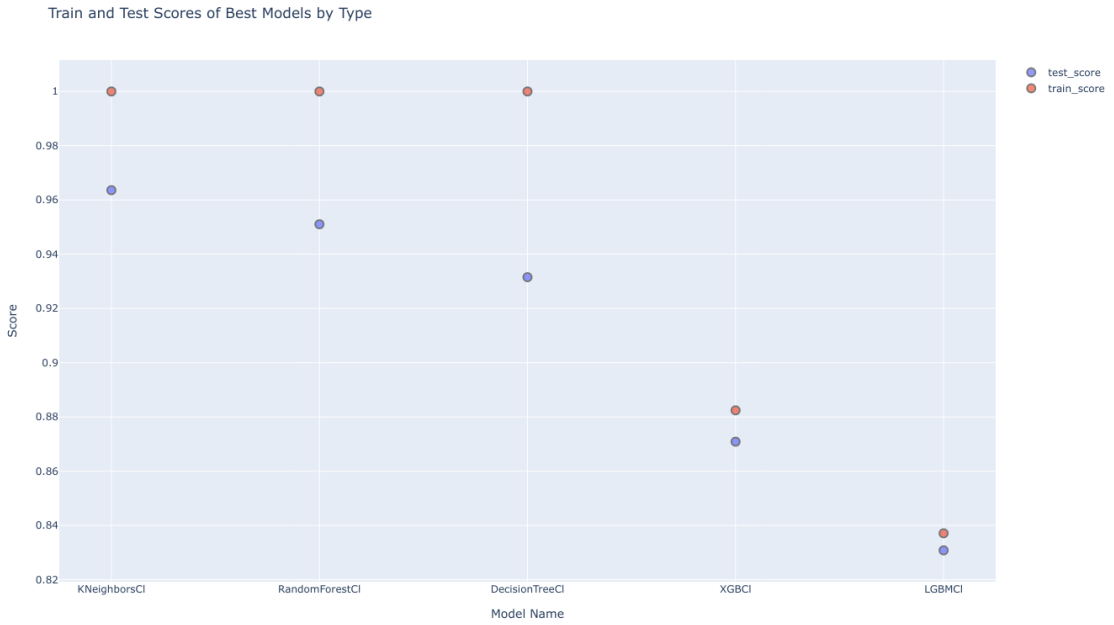
1.3 *Presentation and visualization of results*

[506]:
```
fig = px.scatter(best_of_classifiers, x="classifier", y="mean_test_score",␣
 ↪color="classifier",
                 size='mean_test_score', hover_data=['mean_test_score'])
# change the layout properties
fig.update_layout(title='Scores of Best Models by Type', legend_orientation="v",
                  xaxis_title='Model Name',
                  yaxis_title='Score', width=1300, height=800, )
fig.show()
```



[508]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(x=best_of_classifiers['classifier'],
                         y=best_of_classifiers['mean_test_score'],
                         name='test_score',
                         mode='markers', opacity=0.7))
fig.add_trace(go.Scatter(x=best_of_classifiers['classifier'],
                         y=best_of_classifiers['mean_train_score'],
                         name='train_score',
                         mode='markers', opacity=0.7))
# define marker size
fig.update_traces(mode='markers', marker_size=10, marker_line_width=2)
# change the layout properties
fig.update_layout(title='Train and Test Scores of Best Models by Type',␣
 ↪legend_orientation="v",
```

```
                    xaxis_title='Model Name',
                    yaxis_title='Score', width=1300, height=800, )
fig.show()
```



1.3.1 *Results with default parameters and without cross validation*

- Performance results show that the best model is K-NN algorithm with 96%, the second one is Random Forest with 95%, the worst model is Logistic Regression with 56%.
- Fastest model is the K-NN with 5 seconds before Decision Tree with 7 seconds and slowest one is XGBoost with 5 minutes.

1.3.2 *Results with hyperparameter search and cross validation*

- Performance results show that the best model is K-NN algorithm with 96%, the second best one is Random Forest with 95%.
- K-NN and Decision Tree models are the most fastest ones with 5 and 7 seconds fit time respectively. XGBoost fit time increased to the 800 secs with the hyperparameter search.

1.3.3 *Plotly with Dash*

- Dash allows us to serve dynamic plotly graphs over the web. JupyterDash extension is used to serve Dash apps inline in the Jupyter notebook.

[512]:
```
# identify the classifier name
grid_cl_df['classifier'] = get_names(grid_cl_df)
```

[515]:
```
grid_cl_df['classifier'].unique()
```

[515]:
```
array(['KNeighborsCl', 'RandomForestCl', 'DecisionTreeCl', 'XGBCl',
       'LGBMCl'], dtype=object)
```

[513]:
```
grid_cl_df.head(1)
```

[513]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
1          4.982         0.113            6.663           0.301

                                        param_classifier  \
1  KNeighborsClassifier(n_neighbors=4, weights='distance')

  param_classifier__n_neighbors param_classifier__weights  \
1                             4                 distance

  param_classifier__max_depth param_classifier__reg_alpha  \
1                         NaN                         NaN

  param_classifier__reg_lambda  \
1                          NaN

                params  \
1  {'classifier': KNeighborsClassifier(n_neighbors=4, weights='distance'),
'classifier__n_neighbors…

   split0_test_score  split1_test_score  split2_test_score  split3_test_score  \
1              0.964              0.963              0.964              0.964

   split4_test_score  mean_test_score  std_test_score  rank_test_score  \
1              0.963            0.964           0.001                1

   split0_train_score  split1_train_score  split2_train_score  \
1               1.000               1.000               1.000

   split3_train_score  split4_train_score  mean_train_score  std_train_score  \
1               1.000               1.000             1.000            0.000

     classifier
1  KNeighborsCl
```

[548]:
```
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html
```

```
from dash.dependencies import Input, Output
import json
```

- Above dash app serves a dropdown for each classifier to plot the corresponding model results. When hovering over the data points model parameters can be seen dynamically under the graph.

[613]:
```
# load Data
df = grid_cl_df
# build App
app = JupyterDash(__name__)
app.layout = html.Div([
    html.H1("Model Results", style={'text-align': 'center'}),
    html.Div(
    html.Label([
        "Select the model and hover over the data to get the parameters",
        dcc.Dropdown(
            id='model-dropdown', clearable=False,
            value='KNeighborsCl', options=[
                {'label': c, 'value': c}
                for c in grid_cl_df['classifier'].unique()
            ])
    ])),
    html.Div(dcc.Graph(id='model-graph')),
    html.Div(html.H3(id='hover-data',style={'font-family':'verdana'}),␣
 ↪style={'text-align': 'center'})

])

# define callback to update parameter text
@app.callback(Output('hover-data','children'),
              [Input("model-dropdown", "value"),
               Input('model-graph','hoverData')])
def callback_text(model, hoverData):
    try:
        train_score = hoverData['points'][0]['x']
        test_score = hoverData['points'][0]['y']
        fit_time = hoverData['points'][0]['text']
        #return json.dumps(hoverData, indent=2)
        if (model=='KNeighborsCl'):
            neighbors = df.loc[(df['classifier']==model)&
                               (df['mean_test_score']==test_score)].
 ↪iloc[0]['param_classifier__n_neighbors']
            weight = df.loc[(df['classifier']==model)&
                               (df['mean_test_score']==test_score)].
 ↪iloc[0]['param_classifier__weights']
            return 'neighbors: ' + str(neighbors) + ',\t weight: ' + str(weight)
```

```
        if (model=='DecisionTreeCl' or model=='RandomForestCl'):
            max_depth = df.loc[(df['classifier']==model)&
                               (df['mean_test_score']==test_score)].
↪iloc[0]['param_classifier__max_depth']
            return 'max depth: ' + str(max_depth)
        if (model=='XGBCl' or model=='LGBMCl'):
            alpha = df.loc[(df['classifier']==model)&
                               (df['mean_test_score']==test_score)].
↪iloc[0]['param_classifier__reg_alpha']
            lambda_ = df.loc[(df['classifier']==model)&
                               (df['mean_test_score']==test_score)].
↪iloc[0]['param_classifier__reg_lambda']
            max_depth = df.loc[(df['classifier']==model)&
                               (df['mean_test_score']==test_score)].
↪iloc[0]['param_classifier__max_depth']
            return 'alpha: ' + str(alpha) + ',\t lambda: ' + str(lambda_) +␣
↪',\t max depth: ' + str(max_depth)
    except:
        pass


# define callback to update graph
@app.callback(Output('model-graph', 'figure'),
              [Input("model-dropdown", "value")])
def update_figure(model):

    data = [go.Scatter(x=df[df['classifier']==model]['mean_train_score'],
                       y=df[df['classifier']==model]['mean_test_score'],
                       mode='markers', marker=dict(size=15,
                                                   opacity=0.8,
                                                  ␣
↪color=df[df['classifier']==model]['mean_fit_time'],
                                                   colorscale='Oranges',
                                                   showscale=True,
                                                  ␣
↪colorbar=dict(title="mean_fit_time (secs)"),
                                                   line={'width':2, 'color':
↪'black'}),
                       text=df[df['classifier']==model]['mean_fit_time']
                      )]
    layout = go.Layout(title= model + ' Test and Train Scores with Fit Time',
                       hovermode='closest',
                       xaxis={'title': 'Mean train score'},
                       yaxis={'title': 'Mean test score'})
    fig = {'data':data, 'layout':layout}
```

```
    return fig

# Run app and display result inline in the notebook
app.run_server(mode='inline')
```

```
<IPython.lib.display.IFrame at 0x7fb9559e2b90>
```