CZ1115 LAB NOTES

<http://www.unit-conversion.info/texttools/replace-text/>

TAB = AUTOCOMPLETE

APPEND '?' to command for documentation

======================================================

# Basic

import pandas as pd

csvData = pd.read\_csv('train.csv', header = 'infer'/None)

>> if train.csv in a folder, use 'data/train.csv'

csvData.head(n)

csvData.tail(n)

n = integer

.sort\_values('colName', ascending = True)

columns = data.columns #select column headers

## SELECT/DROP COLUMNS

csvData.select\_dtypes(include = 'int64')

csvData.drop( ['colName'] )

>> generates a new object. csvData2 = ... to save the generated result.

>> e.g. csvDataTop20 = csvData.head(20)

csvData[csvData['colName'] == False]

# Strongest Pokemons in each Generation -- the Top 10

generation = 1

pkmndata\_clean[pkmndata\_clean['GENERATION'] == generation].sort\_values('TOTAL', ascending=False).head(10)

csvData.iloc[7]

>> extract a single ROW (data of object 7)

>> returns a Series

colName.isna() / colName.notna() / colName.isnull()

>> filter rows where column x is/is not NaN

## CREATE NEW COLUMN

csvData['colXExists'] = csvData.colX.notna()

csvData.colXExists = csvData.colXExists.astype('category')

>> Categories for colXExists is either True/False due to notna()

csvData.dtypes #verify

## FIND MAX

colName.max(): value of max

maxEntry = csvData.iloc[csvData.colName.argmax()]: entry of max

for var in csvData:

print(var, ': ', maxEntry[var])

## COUNT

len(csvData['colName'].unique())

>> number of unique names in the column

csvData['colName'].value\_counts()

>> total number of entries in the column

======================================================

# Statistics

csvData.describe()

Q1 = csvData.quantile(0.25)

Q3 = csvData.quantile(0.75)

IQR = Q3-Q1

((csvData < (Q1 - 1.5 \* IQR)) | (csvData > (Q3 + 1.5 \* IQR))).sum()

======================================================

# Clean Data

Refer M2 ExploratoryAnalysis second half

# Compute the Average TOTAL across every pair of TYPEs

total\_means = pkmndata\_clean.groupby(['TYPE\_1', 'TYPE\_2']).mean().loc[:, 'TOTAL']

# Strongest Pokemons in each Pair of Types -- the Top 10

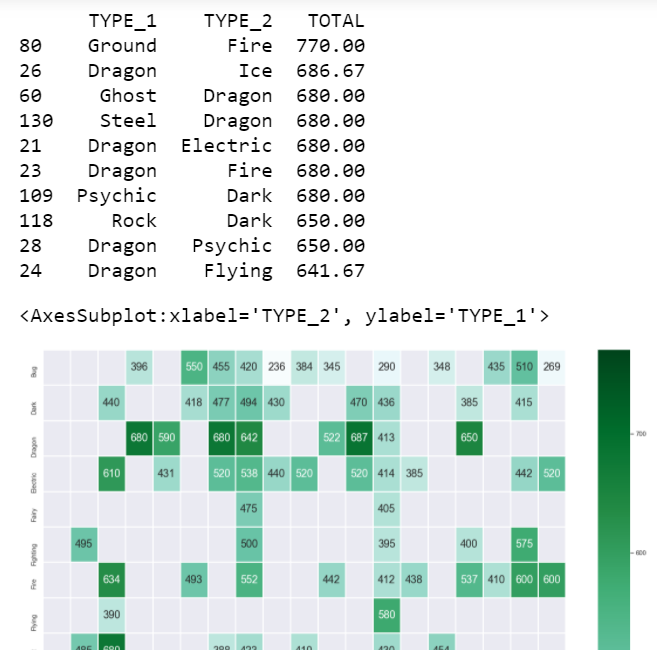
print(total\_means.reset\_index().sort\_values('TOTAL', ascending=False).head(10).round(2))

# Heatmap of Average TOTAL across every pair of TYPEs

f = plt.figure(figsize=(20, 20))

sb.heatmap(total\_means.unstack(), linewidths = 1,

annot = True, fmt = '.0f', annot\_kws = {'size': 18}, cmap = 'BuGn')



======================================================

# Visualisation

## Categorical vs numeric: boxplot

## Numeric vs numeric: scatterplot

Year and month: numeric or categorical?

Year built: categorical

Year sold: categorical

Year sold - year built = age: numeric

OverallQual: numeric or categorical?

Numeric: statistics work

Category: levels are different

'Ordinal' category

import numpy as np

import pandas as pd

import seaborn as sb

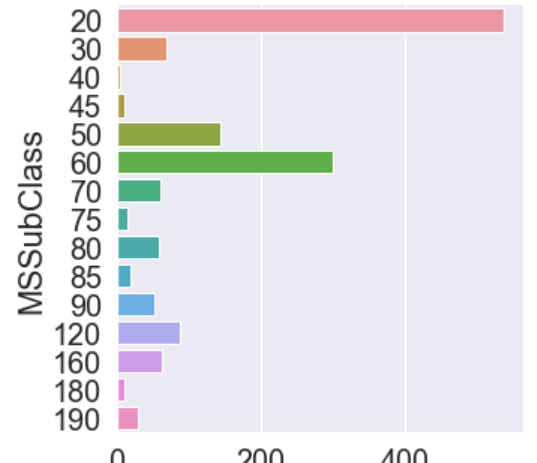
import matplotlib.pyplot as plt

# UNIVARIATE

hp = pd.DataFrame(csvData['HP'])

## ONE CATEGORICAL VARIABLE

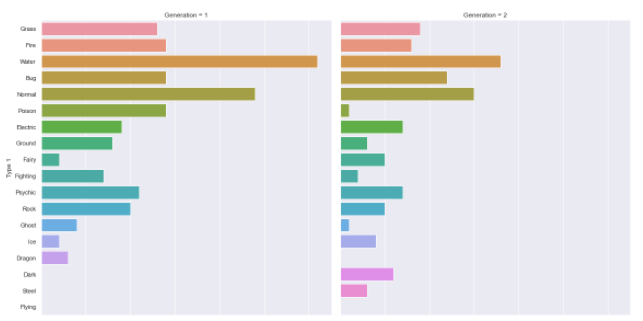
sb.catplot(y = 'colName', data= csvData, kind = 'count', order = csvData['colName'].value\_counts().index, height = 8)



for var in csvData:

sb.catplot(y = var,....)

sb.catplot(y = 'colName1', data = csvData, col = 'colName2', kind = 'count', col\_wrap = 2, height = 8)



## ONE CONTINUOUS VARIABLE

# Set up matplotlib figure with three subplots: 1x3 o o o

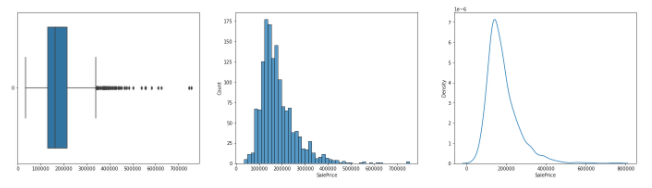
f, pltplt.subplots(1, 3, figsize=(24, 6))

# Plot the basic uni-variate figures for HP

sb.boxplot(data = hp, orient = 'h', ax = axes[0])

sb.histplot(data = hp, ax = axes[1])

sb.kdeplot(data = hp, ax = axes[2])



sb.histplot(data = hp, ax=axes[3], kde = True)

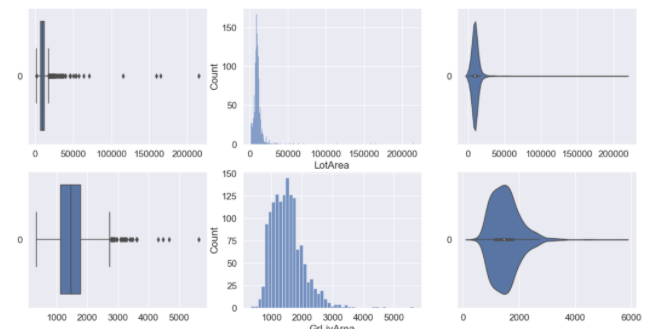
sb.violinplot(data = hp, orient = 'h')

numDF = pd.DataFrame(csvData[['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']])

for var in numDF:

....numDF[var]....ax=axes[count,0]

... # equivalent to hp

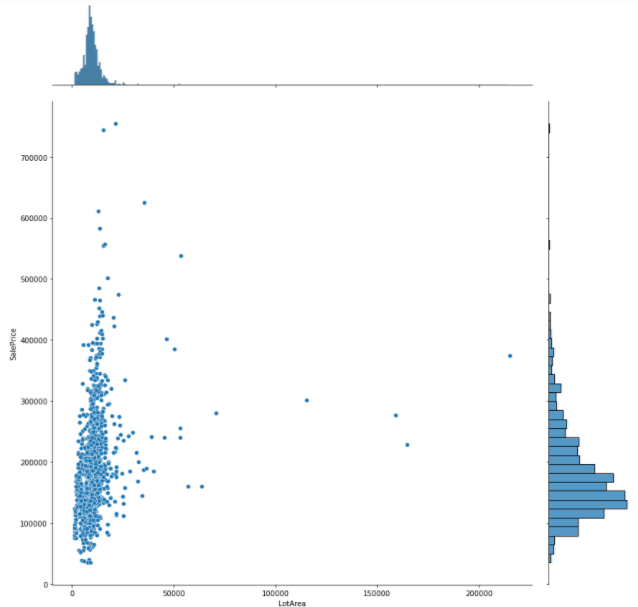


# BIVARIATE

jointData = pd.concat([csvData1, csvData2], axis = 1).reindex(csvData1.index)

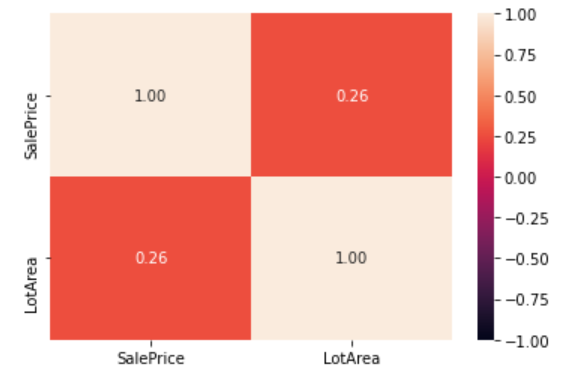
## TWO CONTINUOUS VARIABLES

sb.jointplot(data = jointData, x='csvData2', y='csvData1', height = 12)



jointData.corr()

sb.heatmap(jointData.corr(), vmin = -1, vmax = 1, annot = True, fmt = '.2f', annot\_kws = {"size":18), cmap = "RdBu")

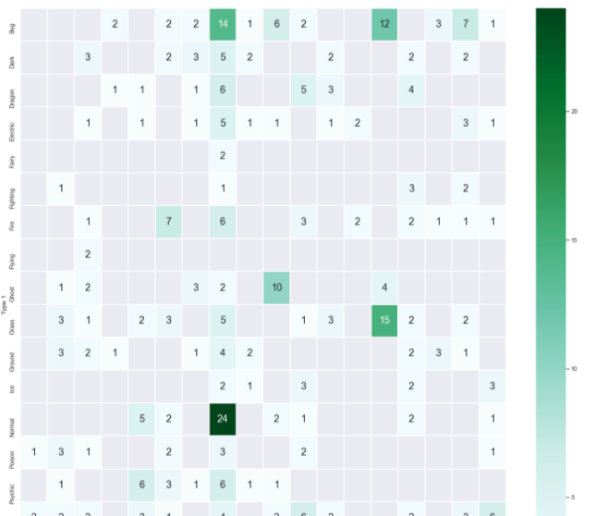


## TWO CATEGORICAL VARIABLES

dualtype\_gen1 = dualtype\_data[dualtype\_data['Generation'] == 1]

sb.heatmap(dualtype\_gen1.groupby(['Type 1', 'Type 2']).size().unstack(),

linewidths = 1, annot = True, annot\_kws = {'size': 14}, cmap = 'BuGn', ax = axes[0,0])



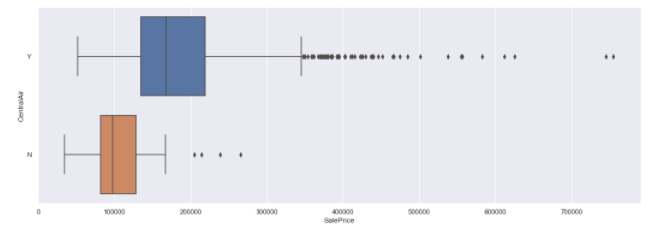
## ONE CONTINUOUS + ONE CATEGORICAL VARIABLE

for var in houseCatData:

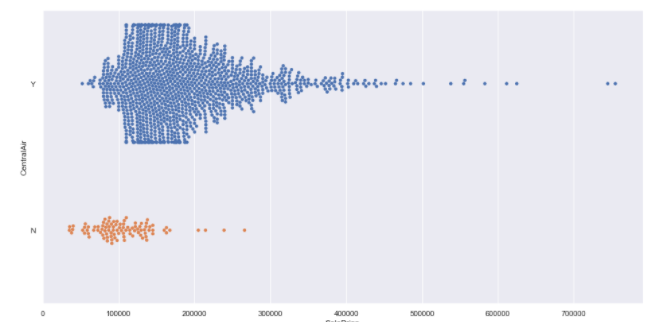
sb.boxplot(y = houseData['SalePrice'], x = houseCatData[var], ax = axes[lol])

lol +=1

sb.boxplot(x='SalePrice', y='categorical', data=jointDF, orient = 'h')

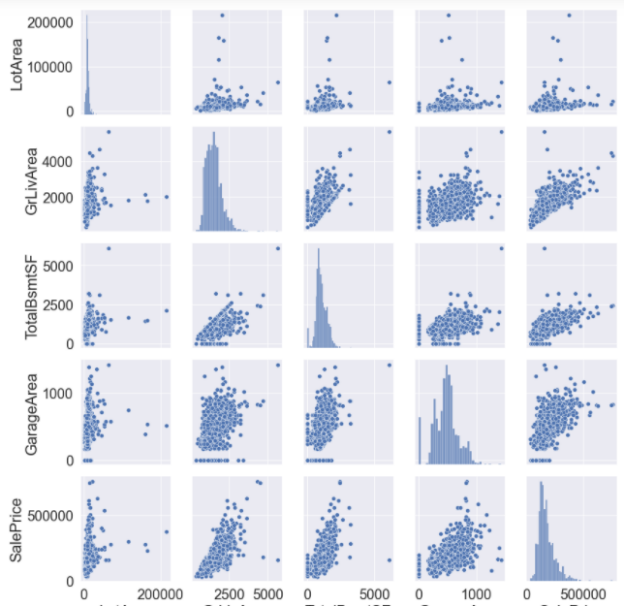


sb.swarmplot(x='SalePrice', y='categorical', data=jointDF, orient='h')



# MULTIVARIATE

sb.pairplot(data = numDF)



======================================================

# LINEAR REGRESSION

# Import essential models and functions from sklearn

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

response = pd.DataFrame(csvData['YY'])

predictor = pd.DataFrame(csvData['XX'])

# Split the Dataset into Train and Test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(predictor, response, test\_size = 0.25)

# Check the sample sizes

print('Train Set :', y\_train.shape, X\_train.shape)

print('Test Set :', y\_test.shape, X\_test.shape)

# Linear regression using train data

linreg = LinearRegression()

linreg.fit(X\_train, y\_train)

#Plot using math

# Coefficients of the Linear Regression line

print('Intercept of Regression \t: b = ', linreg.intercept\_)

print('Coefficients of Regression \t: a = ', linreg.coef\_)

print()

# Formula for the Regression line

regline\_x = X\_train

regline\_y = linreg.intercept\_ + linreg.coef\_ \* X\_train

# Plot the Linear Regression line

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_train, y\_train)

plt.plot(regline\_x, regline\_y, 'r-', linewidth = 3)

plt.show()



# Predict response values using predictor values for both train and test

y\_train\_pred = linreg.predict(X\_train)

y\_test\_pred = linreg.predict(X\_test)

#Plot the predictions (train)

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_train, y\_train)

plt.scatter(X\_train, y\_train\_pred, color = 'r')

plt.show()



#Plot the predictions (test)

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_test, y\_test, color = 'green')

plt.scatter(X\_test, y\_test\_pred, color = 'red')

plt.show()



# Check the Goodness of Fit (on Train Data)

print('Goodness of Fit of Model \tTrain Dataset')

print('Explained Variance (R^2) \t:', linreg.score(X\_train, y\_train))

print('Mean Squared Error (MSE) \t:', mean\_squared\_error(y\_train, y\_train\_pred))

import numpy as np

print('Root Mean Squared Error (RMSE) \t: , np.sqrt(mean\_sqaured\_error(y\_train, y\_train\_pred))')

print()

# Check the Goodness of Fit (on Test Data)

print('Goodness of Fit of Model \tTest Dataset')

print('Explained Variance (R^2) \t:', linreg.score(X\_test, y\_test))

print('Mean Squared Error (MSE) \t:', mean\_squared\_error(y\_test, y\_test\_pred))

print()

# Plot the Predictions vs the True values

f, axes = plt.subplots(1, 2, figsize=(24, 12))

axes[0].scatter(y\_train, y\_train\_pred, color = 'blue')

axes[0].plot(y\_train, y\_train, 'w-', linewidth = 1)

axes[0].set\_xlabel('True values of the Response Variable (Train)')

axes[0].set\_ylabel('Predicted values of the Response Variable (Train)')

axes[1].scatter(y\_test, y\_test\_pred, color = 'green')

axes[1].plot(y\_test, y\_test, 'w-', linewidth = 1)

axes[1].set\_xlabel('True values of the Response Variable (Test)')

axes[1].set\_ylabel('Predicted values of the Response Variable (Test)')

plt.show()



## Manual prediction testing

csvData\_selected = csvData[csvData['Name'].isin(['Charizard', 'Snorlax', 'Vivillon'])]

predictors = ['HP', 'Attack', 'Defense']

X\_pred = pd.DataFrame(csvData\_selected[predictors])

y\_pred = linreg.predict(X\_pred)

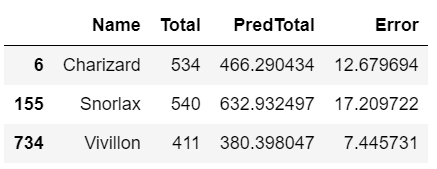
y\_pred = pd.DataFrame(y\_pred, columns = ['PredTotal'], index = csvData\_selected.index)

pkmndata\_acc = pd.concat([csvData\_selected[['Name', 'Total']], y\_pred], axis = 1)

y\_errs = 100 \* abs(pkmndata\_acc['Total'] - pkmndata\_acc['PredTotal']) / pkmndata\_acc['Total']

y\_errs = pd.DataFrame(y\_errs, columns = ['Error'], index = csvData\_selected.index)

pkmndata\_acc = pd.concat([pkmndata\_acc, y\_errs], axis = 1)



======================================================

# Classification Tree

# Import Decision Tree Classifier model from Scikit-Learn

from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree Classifier object

# you can change the max\_depth as you wish

dectree = DecisionTreeClassifier(max\_depth = 2)

# Import the required function from sklearn

from sklearn.model\_selection import train\_test\_split

# Extract Response and Predictors

y = pd.DataFrame(csvData[response])

X = pd.DataFrame(csvData[predictor])

# Split the Dataset into random Train and Test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 360)

# Check the sample sizes

print('Train Set :', X\_train.shape, y\_train.shape)

print('Test Set :', X\_test.shape, y\_test.shape)

dectree.fit(X\_train, y\_train)

# Plot the trained Decision Tree

from sklearn.tree import plot\_tree

f = plt.figure(figsize=(12,12))

plot\_tree(dectree, filled=True, rounded=True,

feature\_names=X\_train.columns,

class\_names=['N','Y'])

# Import the required metric from sklearn

from sklearn.metrics import confusion\_matrix

# Predict the Response corresponding to Predictors

y\_train\_pred = dectree.predict(X\_train)

# Plot the two-way Confusion Matrix

sb.heatmap(confusion\_matrix(y\_train, y\_train\_pred),

annot = True, fmt='.0f', annot\_kws={'size': 18})

# Print the Classification Accuracy

print('Train Data')

print('Accuracy :\t', dectree.score(X\_train, y\_train))

print()

# Print the Accuracy Measures from the Confusion Matrix

cmTrain = confusion\_matrix(y\_train, y\_train\_pred)

tpTrain = cmTrain[1][1] # True Positives : Y (1) predicted Y (1)

fpTrain = cmTrain[0][1] # False Positives : N (0) predicted Y (1)

tnTrain = cmTrain[0][0] # True Negatives : N (0) predicted N (0)

fnTrain = cmTrain[1][0] # False Negatives : Y (1) predicted N (0)

print('TPR Train :\t', (tpTrain/(tpTrain + fnTrain)))

print('TNR Train :\t', (tnTrain/(tnTrain + fpTrain)))

print()

print('FPR Train :\t', (fpTrain/(tnTrain + fpTrain)))

print('FNR Train :\t', (fnTrain/(tpTrain + fnTrain)))

#DO ALL OF THE ABOVE BUT FOR TEST DATA

======================================================

# Types

DataFrame is a class

E.g. csvData[['columnName']]

Series is a Python-like list

E.g. csvData.columnName

csvData['columnName']

int64 is a number datatype in numpy

======================================================

## Anaconda Prompt (Must run as administrator!)

(base)

* Anaconda environment called 'base'.
* This is the default environment. Environments are used for different development contexts (e.g. web – Django, Flask, data etc)

> conda

* Package manager.

+ '--v': check version number

+ 'list': list all packages currently installed

+ 'update conda': update conda. Replace 'conda' with '--all' to update

+ 'install plotly': installs plotly app. Replace 'plotly' with appName

> jupyter notebook

* launches jupyter notebook, same as clicking the icon