import pandas as pd

data = pd.read\_csv('data2.csv')

data.head()

print(data)

data["K"].plot(kind="hist")

data["E"].plot(kind="hist")

import matplotlib.pyplot as plt

from sklearn import datasets

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

from sklearn.linear\_model import LinearRegression

# # Generate scatter plot of independent vs Dependent variable

plt.style.use('ggplot')

fig = plt.figure(figsize = (18, 18))

for index, feature\_name in enumerate(data.columns.values):

ax = fig.add\_subplot(3, 3, index + 1)

ax.scatter(data[feature\_name], data['K'])

ax.set\_ylabel('K')

ax.set\_xlabel(feature\_name)

plt.show()

import numpy as np

# deli = np.where(data['E']<240)

# data.drop(deli[0], inplace = True)

# print(data)

# deli = np.where(data['K']>40000)

# data.drop(deli[0], inplace = True)

# print(data)

data = data.loc[~((data['E'] < 240) | (data['K'] > 40000))]

data["E"].plot(kind="hist")

data["K"].plot(kind="hist")

data.plot(x="n", y="K", kind="scatter")

data[data["n"] == 3]["K"].plot(kind="hist")

data[data["n"] == 3].plot(x="T", y="K", kind="scatter")

import matplotlib.pyplot as plt

from sklearn import datasets

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

from sklearn.linear\_model import LinearRegression

# # Generate scatter plot of independent vs Dependent variable

plt.style.use('ggplot')

fig = plt.figure(figsize = (18, 18))

for index, feature\_name in enumerate(data.columns.values):

ax = fig.add\_subplot(3, 3, index + 1)

ax.scatter(data[feature\_name], data['K'])

ax.set\_ylabel('K')

ax.set\_xlabel(feature\_name)

plt.show()

X = data[['n','T']]

y = data['K']

print(X)

print(y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

X, y,

test\_size = 0.25)

print("Train data shape of X = % s and Y = % s : "%(

x\_train.shape, y\_train.shape))

print("Test data shape of X = % s and Y = % s : "%(

x\_test.shape, y\_test.shape))

# Apply multiple Linear Regression Model

lreg = LinearRegression()

lreg.fit(x\_train, y\_train)

# Generate Prediction on test set

lreg\_y\_pred = lreg.predict(x\_test)

# calculating Mean Squared Error (mse)

mean\_squared\_error = np.mean((lreg\_y\_pred - y\_test)\*\*2)

print("Mean squared Error on test set : ", mean\_squared\_error)

print(mean\_squared\_error)

# Putting together the coefficient and their corresponding variable names

lreg\_coefficient = pd.DataFrame()

lreg\_coefficient["Columns"] = x\_train.columns

lreg\_coefficient['Coefficient Estimate'] = pd.Series(lreg.coef\_)

print(lreg\_coefficient)

# plotting the coefficient score

fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',

'tab:green', 'tab:red', 'tab:purple', 'tab:brown',

'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan',

'tab:orange', 'tab:green', 'tab:blue', 'tab:olive']

ax.bar(lreg\_coefficient["Columns"],

lreg\_coefficient['Coefficient Estimate'],

color = color)

ax.spines['bottom'].set\_position('zero')

plt.style.use('ggplot')

plt.show()

pred = lreg.predict(x\_test)

k = np.mean((pred - y\_test)\*\*2)

# print(pred)

print(k)

lreg.score(x\_test,y\_test)

x\_plot = plt.scatter(pred, (pred - y\_test), c='b')

plt.hlines(y=0, xmin= -1000, xmax=5000)

plt.title('Residual plot')

predictors = x\_train.columns

coef = pd.Series(lreg.coef\_,predictors).sort\_values()

coef.plot(kind='bar', title='Modal Coefficients')

from sklearn.linear\_model import Ridge

## training the model

ridgeReg = Ridge(alpha=0.09, normalize=True)

ridgeReg.fit(x\_train,y\_train)

pred = ridgeReg.predict(x\_test)

mse = np.mean((pred - y\_test)\*\*2)

# print(mse)

ridgeReg.score(x\_test,y\_test)

lreg\_coefficient = pd.DataFrame()

lreg\_coefficient["Columns"] = x\_train.columns

lreg\_coefficient['Coefficient Estimate'] = pd.Series(ridgeReg.coef\_)

print(lreg\_coefficient)

x\_plot = plt.scatter(pred, (pred - y\_test), c='b')

plt.hlines(y=0, xmin= -1000, xmax=5000)

plt.title('Residual plot')

predictors = x\_train.columns

coef = pd.Series(ridgeReg.coef\_,predictors).sort\_values()

coef.plot(kind='bar', title='Modal Coefficients')

from sklearn.linear\_model import Lasso

lassoReg = Lasso(alpha=0.05, normalize=True)

lassoReg.fit(x\_train,y\_train)

pred = lassoReg.predict(x\_test)

# calculating mse

mse = np.mean((pred - y\_test)\*\*2)

lassoReg.score(x\_test,y\_test)

lreg\_coefficient = pd.DataFrame()

lreg\_coefficient["Columns"] = x\_train.columns

lreg\_coefficient['Coefficient Estimate'] = pd.Series(lassoReg.coef\_)

print(lreg\_coefficient)

x\_plot = plt.scatter(pred, (pred - y\_test), c='b')

plt.hlines(y=0, xmin= -1000, xmax=5000)

plt.title('Residual plot')

predictors = x\_train.columns

coef = pd.Series(lassoReg.coef\_,predictors).sort\_values()

coef.plot(kind='bar', title='Modal Coefficients')

from sklearn.linear\_model import ElasticNet

ENreg = ElasticNet(alpha=1, l1\_ratio=0.3, normalize=False)

ENreg.fit(x\_train,y\_train)

pred = ENreg.predict(x\_test)

#calculating mse

mse = np.mean((pred - y\_test)\*\*2)

ENreg.score(x\_test,y\_test)

lreg\_coefficient = pd.DataFrame()

lreg\_coefficient["Columns"] = x\_train.columns

lreg\_coefficient['Coefficient Estimate'] = pd.Series(ENreg.coef\_)

print(lreg\_coefficient)

x\_plot = plt.scatter(pred, (pred - y\_test), c='b')

plt.hlines(y=0, xmin= -1000, xmax=5000)

plt.title('Residual plot')

predictors = x\_train.columns

coef = pd.Series(ENreg.coef\_,predictors).sort\_values()

coef.plot(kind='bar', title='Modal Coefficients')

mlr = LinearRegression()

mlr.fit(x\_train, y\_train)

#Prediction of test set

pred\_cv= mlr.predict(x\_test)

#Predicted values

mse = np.mean((pred\_cv - y\_test)\*\*2)

mlr.score(x\_test,y\_test)

lreg\_coefficient = pd.DataFrame()

lreg\_coefficient["Columns"] = x\_train.columns

lreg\_coefficient['Coefficient Estimate'] = pd.Series(mlr.coef\_)

print(lreg\_coefficient)

x\_plot = plt.scatter(pred, (pred - y\_test), c='b')

plt.hlines(y=0, xmin= -1000, xmax=5000)

plt.title('Residual plot')

predictors = x\_train.columns

coef = pd.Series(mlr.coef\_,predictors).sort\_values()

coef.plot(kind='bar', title='Modal Coefficients')

n = 8

T = 1023

features = np.array([[n,T]])

prediction = ENreg.predict(features)

print("Prediction value of K for n : {} and T : {} using Elastic Net Regression : {}".format(n,T,prediction))

prediction = lassoReg.predict(features)

print("Prediction value of K for n : {} and T : {} using Lasso regression : {}".format(n,T,prediction))

prediction = ridgeReg.predict(features)

print("Prediction value of K for n : {} and T : {} using Ridge Regression : {}".format(n,T,prediction))

prediction = lreg.predict(features)

print("Prediction value of K for n : {} and T : {} using Linear Regression : {}".format(n,T,prediction))

prediction = mlr.predict(features)

print("Prediction value of K for n : {} and T : {} using Multiple Linear Regression : {}".format(n,T,prediction))