# Lasso, Ridge, and ElasticNet Regression

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Vishvash C Batch ID:** 23012024

**Topic: Lasso, Ridge, and ElasticNet Regression**

**Guidelines:**

**1. An assignment submission is considered complete only when the correct and executable code(s) and documentation explaining the method and results are submitted. Failing to submit either of those will be considered an invalid submission and not a correct submission.**

**2. Ensure that you submit your assignments correctly and in full. Resubmission is not allowed.**

**3. Post the submission you can evaluate your work by referring to the keys provided. (will be available only post the submission).**

**Hints:**

1. **Business Problem**
   1. **What is the business objective?**
   2. **Are there any constraints?**
2. **Work on each feature of the dataset to create a data dictionary as displayed in the below image:**

**Make a table as shown above and provide information about the features such as its data type and its relevance to the model building. And if not relevant, provide reasons and a description of the feature.**

1. **Data Pre-processing**

**3.1 Data Cleaning, Feature Engineering, etc.**

**3.2 Outlier Treatment.**

1. **Exploratory Data Analysis (EDA):**
   1. **Summary.**
   2. **Univariate analysis.**
   3. **Bivariate analysis.**
2. **Model Building**
   1. **Build the model on the scaled data (try multiple options).**
   2. **Perform Lasso, Ridge, and Elastic Net Regressions.**
   3. **Train and test the model and compare RMSE values. Tabulate R-Squared and RMSE values for different models in the documentation and provide an explanation.**
   4. **Briefly explain the model output in the documentation.**
3. **Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided?**

**Problem Statements:**

1. An analytics company has been tasked with the crucial job of finding out what factors affect a startup company and if it will be profitable or not. For this, they have collected some historical data and would like to apply multilinear regression to derive brief insights into their data. Predict profit, given different attributes for various startup companies.

Table

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Name of Feature | Description | Type | Relevance |
| R&D Spend | Amount spent on R&D | Quantitative | Relevant |
| Administration | Amount spent on administration | Quantitative | Relevant |
| Marketing Spend | Amount spent on marketing | Quantitative | Relevant |
| State | State where the business operates | Nominal | Relevant |
| Profit | Profit generated | Quantitative | Relevant |

**Code:**

## Problem Statement

'''

An analytics company has been tasked with the crucial job of finding out what factors affect a startup company and if it will be profitable or not. For this, they have collected some historical data and would like to apply multilinear regression to derive brief insights into their data. Predict profit, given different attributes for various startup companies.

\*\*CRISP-ML(Q) process model describes six phases:\*\*

- Business and Data Understanding

- Data Preparation (Data Engineering)

- Model Building (Machine Learning)

- Model Evaluation and Tunning

- Deployment

- Monitoring and Maintenance

\*\*Objective(s):\*\* Maximize the profits

\*\*Constraints:\*\* Minimize the spends

\*\*Success Criteria\*\*

- \*\*Business Success Criteria\*\*: Improve the profits from anywhere between 10% to 20%

- \*\*ML Success Criteria\*\*: RMSE should be less than 0.15

- \*\*Economic Success Criteria\*\*: Business should see increase in sales by atleast 20%

'''

# Load the Data and perform EDA and Data Preprocessing

# Importing necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# pip install sidetable

# pip install watchdog

# pip install --upgrade watchdog

import sidetable

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import OneHotEncoder

from feature\_engine.outliers import Winsorizer

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from statsmodels.tools.tools import add\_constant

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

# import statsmodels.formula.api as smf

import statsmodels.api as sm

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

import joblib

import pickle

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import KFold

from sklearn.model\_selection import GridSearchCV

# Recursive feature elimination

from sklearn.feature\_selection import RFE

from sqlalchemy import create\_engine, text

engine = create\_engine("mysql+pymysql://{user}:{pw}@localhost/{db}"

.format(user = "root",# user

pw = "1234", # passwrd

db = "profit\_db")) #database

# Load the offline data into Database to simulate client conditions

profit = pd.read\_csv(r"C:/Users/Lenovo/Downloads/Study material/Data Science/Multiple\_Linear Regression/Assignments/Multiple Linear Regression/50\_Startups.csv")

profit.to\_sql('profit', con = engine, if\_exists = 'replace', chunksize = 1000, index= False)

#### Read the Table (data) from MySQL database

sql = 'SELECT \* FROM profit'

# sql2="show tables"

# tables = pd.read\_sql\_query(sql2, engine)

dataset = pd.read\_sql\_query(text(sql), engine.connect())

# dataset = pd.read\_csv(r"C:/Data/profitwithState.csv")

#### Descriptive Statistics and Data Distribution

dataset.describe()

# Missing values check

dataset.isnull().any()

dataset.info()

# Seperating input and output variables

X = pd.DataFrame(dataset.iloc[:, 1:5])

y = pd.DataFrame(dataset.iloc[:, 0])

# Checking for unique values

X["State"].unique()

X["State"].value\_counts()

# Build a frequency table using sidetable library

X.stb.freq(["State"])

# Segregating Non-Numeric features

categorical\_features = X.select\_dtypes(include = ['object']).columns

print(categorical\_features)

# Segregating Numeric features

numeric\_features = X.select\_dtypes(exclude = ['object']).columns

print(numeric\_features)

## Missing values Analysis

# Define pipeline for missing data if any

num\_pipeline = Pipeline(steps = [('impute', SimpleImputer(strategy = 'mean'))])

preprocessor = ColumnTransformer(transformers = [('num', num\_pipeline, numeric\_features)])

# Fit the imputation pipeline to input features

imputation = preprocessor.fit(X)

# Save the pipeline

joblib.dump(imputation, 'meanimpute')

# Transformed data

cleandata = pd.DataFrame(imputation.transform(X), columns = numeric\_features)

cleandata

## Outlier Analysis

# Multiple boxplots in a single visualization.

# Columns with larger scales affect other columns.

# Below code ensures each column gets its own y-axis.

# pandas plot() function with parameters kind = 'box' and subplots = True

X.plot(kind = 'box', subplots = True, sharey = False, figsize = (25, 18))

'''sharey True or 'all': x- or y-axis will be shared among all subplots.

False or 'none': each subplot x- or y-axis will be independent.'''

# Increase spacing between subplots

plt.subplots\_adjust(wspace = 0.75)

# ws is the width of the padding between subplots, as a fraction of the average Axes width.

plt.show()

# Winsorization for outlier treatment

winsor = Winsorizer(capping\_method = 'iqr', # choose IQR rule boundaries or gaussian for mean and std

tail = 'both', # cap left, right or both tails

fold = 1.5,

variables = list(cleandata.columns))

clean = winsor.fit(cleandata)

# Save winsorizer model

joblib.dump(clean, 'winsor')

cleandata1 = pd.DataFrame(clean.transform(cleandata), columns = numeric\_features)

# Boxplot

cleandata1.plot(kind = 'box', subplots = True, sharey = False, figsize = (25, 18))

plt.subplots\_adjust(wspace = 0.75)

plt.show()

# Scaling

## Scaling with MinMaxScaler

scale\_pipeline = Pipeline([('scale', MinMaxScaler())])

scale\_columntransfer = ColumnTransformer([('scale', scale\_pipeline, numeric\_features)])

# Skips the transformations for remaining columns

scale = scale\_columntransfer.fit(cleandata1)

# Save Minmax scaler pipeline model

joblib.dump(scale, 'minmax')

scaled\_data = pd.DataFrame(scale.transform(cleandata1), columns = numeric\_features)

scaled\_data.describe()

## Encoding

# Categorical features

encoding\_pipeline = Pipeline([('onehot', OneHotEncoder())])

preprocess\_pipeline = ColumnTransformer([('categorical', encoding\_pipeline, categorical\_features)])

clean = preprocess\_pipeline.fit(X) # Works with categorical features only

# Save the encoding model

joblib.dump(clean, 'encoding')

encode\_data = pd.DataFrame(clean.transform(X))

# To get feature names for Categorical columns after Onehotencoding

encode\_data.columns = clean.get\_feature\_names\_out(input\_features = X.columns)

encode\_data.info()

clean\_data = pd.concat([scaled\_data, encode\_data], axis = 1)

# concatenated data will have new sequential index

clean\_data.info()

####################

# Multivariate Analysis

sns.pairplot(dataset) # original data

# Correlation Analysis on Original Data

orig\_df\_cor = dataset.corr(numeric\_only=True)

orig\_df\_cor

# Heatmap

dataplot = sns.heatmap(orig\_df\_cor, annot = True, cmap = "YlGnBu")

# Heatmap enhanced

# Generate a mask to show values on only the bottom triangle

# Upper triangle of an array.

mask = np.triu(np.ones\_like(orig\_df\_cor, dtype = bool))

sns.heatmap(orig\_df\_cor, annot = True, mask = mask, vmin = -1, vmax = 1)

plt.title('Correlation Coefficient Of Predictors')

plt.show()

# Library to call OLS model

# import statsmodels.api as sm

# Build a vanilla model on full dataset

# By default, statsmodels fits a line passing through the origin, i.e. it

# doesn't fit an intercept. Hence, you need to use the command 'add\_constant'

# so that it also fits an intercept

P = add\_constant(clean\_data)

basemodel = sm.OLS(y, P).fit()

basemodel.summary()

# p-values of coefficients found to be insignificant due to colinearity

# Identify the variale with highest colinearity using Variance Inflation factor (VIF)

# Variance Inflation Factor (VIF)

# Assumption: VIF > 10 = colinearity

# VIF on clean Data

vif = pd.Series([variance\_inflation\_factor(P.values, i) for i in range(P.shape[1])], index = P.columns)

vif

# inf = infinity

# Tune the model by verifying for influential observations

sm.graphics.influence\_plot(basemodel)

clean\_data1\_new = clean\_data.drop(clean\_data.index[[45, 46, 48, 49]])

y\_new = y.drop(y.index[[45, 46, 48, 49]])

# clean\_data1\_new.drop("const", axis=1, inplace=True)

# Build model on dataset

basemode3 = sm.OLS(y\_new, clean\_data1\_new).fit()

basemode3.summary()

# Regularization Techniques: LASSO, RIDGE and ElasticNet Regression

################

### LASSO MODEL ###

from sklearn.linear\_model import Lasso

#help(Lasso)

lasso = Lasso(alpha = 0.13)

lasso.fit(clean\_data, y)

# Coefficient values for all independent variables#

lasso.intercept\_

lasso.coef\_

plt.bar(height = pd.Series(lasso.coef\_), x = pd.Series(clean\_data.columns))

# Create a function called lasso,

pred\_lasso = lasso.predict(clean\_data)

# Adjusted r-square

s1 = lasso.score(clean\_data, y.Profit)

s1

# RMSE

np.sqrt(np.mean((pred\_lasso - np.array(y['Profit']))\*\*2))

### RIDGE REGRESSION ###

from sklearn.linear\_model import Ridge

help(Ridge)

rm = Ridge(alpha = 0.13)

rm.fit(clean\_data, y)

# Coefficients values for all the independent vairbales

rm.intercept\_

rm.coef\_

result = rm.coef\_.flatten()

result

plt.bar(height = pd.Series(result), x = pd.Series(clean\_data.columns))

rm.alpha

pred\_rm = rm.predict(clean\_data)

# Adjusted r-square

s2 = rm.score(clean\_data, y.Profit)

s2

# RMSE

np.sqrt(np.mean((pred\_rm - np.array(y['Profit']))\*\*2))

### ELASTIC NET REGRESSION ###

from sklearn.linear\_model import ElasticNet

help(ElasticNet)

enet = ElasticNet(alpha = 0.13)

enet.fit(clean\_data, y.Profit)

# Coefficients values for all the independent vairbales

enet.intercept\_

enet.coef\_

plt.bar(height = pd.Series(enet.coef\_), x = pd.Series(clean\_data.columns))

pred\_enet = enet.predict(clean\_data)

# Adjusted r-square

s3 = enet.score(clean\_data, y.Profit)

s3

# RMSE

np.sqrt(np.mean((pred\_enet - np.array(y.Profit))\*\*2))

####################

# Lasso Regression

# from sklearn.model\_selection import GridSearchCV

parameters = {'alpha': [1e-10, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.13, 0.2, 1, 5, 10, 20]}

lasso = Lasso()

lasso\_reg = GridSearchCV(lasso, parameters, scoring = 'r2', cv = 5)

lasso\_reg.fit(clean\_data, y.Profit)

lasso\_reg.best\_params\_

lasso\_reg.best\_score\_

lasso\_pred = lasso\_reg.predict(clean\_data)

# Adjusted r-square#

s4 = lasso\_reg.score(clean\_data, y.Profit)

s4

# RMSE

np.sqrt(np.mean((lasso\_pred - np.array(y.Profit))\*\*2))

# Ridge Regression

# from sklearn.model\_selection import GridSearchCV

# from sklearn.linear\_model import Ridge

ridge = Ridge()

# parameters = {'alpha': [1e-10, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.13, 0.2, 1, 5, 10, 20]}

ridge\_reg = GridSearchCV(ridge, parameters, scoring = 'r2', cv = 5)

ridge\_reg.fit(clean\_data, y.Profit)

ridge\_reg.best\_params\_

ridge\_reg.best\_score\_

ridge\_pred = ridge\_reg.predict(clean\_data)

# Adjusted r-square#

s5 = ridge\_reg.score(clean\_data, y.Profit)

s5

# RMSE

np.sqrt(np.mean((ridge\_pred - np.array(y.Profit))\*\*2))

# ElasticNet Regression

# from sklearn.model\_selection import GridSearchCV

# from sklearn.linear\_model import ElasticNet

enet = ElasticNet()

# parameters = {'alpha': [1e-10, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.13, 0.2, 1, 5, 10, 20]}

enet\_reg = GridSearchCV(enet, parameters, scoring = 'r2', cv = 5)

enet\_reg.fit(clean\_data, y.Profit)

enet\_reg.best\_params\_

enet\_reg.best\_score\_

enet\_pred = enet\_reg.predict(clean\_data)

# Adjusted r-square

s6 = enet\_reg.score(clean\_data, y.Profit)

s6

# RMSE

np.sqrt(np.mean((enet\_pred - np.array(y.Profit))\*\*2))

scores\_all = pd.DataFrame({'models':['Lasso', 'Ridge', 'Elasticnet', 'Grid\_lasso', 'Grid\_ridge', 'Grid\_elasticnet'], 'Scores':[s1, s2, s3, s4, s5, s6]})

scores\_all

# Save the Best score model

finalgrid = ridge\_reg.best\_estimator\_

finalgrid

# Save the best model

pickle.dump(finalgrid, open('grid\_best.pkl', 'wb'))

##########

# Prediction

model1 = pickle.load(open('grid\_best.pkl','rb'))

impute = joblib.load('meanimpute')

winsor = joblib.load('winsor')

minmax = joblib.load('minmax')

encoding = joblib.load('encoding')

data = pd.read\_excel(r"C:/Users/Lenovo/Downloads/Study material/Data Science/Lasso, Ridge and ElasticNet Regression/Assignments/Lasso and Ridge Reg/profit\_test.xlsx")

clean = pd.DataFrame(impute.transform(data), columns = data.select\_dtypes(exclude = ['object']).columns)

clean1 = winsor.transform(clean)

clean2 = pd.DataFrame(minmax.transform(clean1))

clean3 = pd.DataFrame(encoding.transform(data))

clean\_data = pd.concat([clean2, clean3], axis = 1, ignore\_index = True)

clean\_data1 = clean\_data.drop(3, axis = 1)

prediction = pd.DataFrame(model1.predict(clean\_data), columns = ['Predict\_Profit'])

final = pd.concat([prediction,data], axis = 1)

final

**Output:**

State count percent cumulative\_count cumulative\_percent

0 New York 17 34.0 17 34.0

1 California 17 34.0 34 68.0

2 Florida 16 32.0 50 100.0

scaled\_data.describe()

Out[54]:

R&D\_Spend Administration Marketing\_Spend

count 50.000000 50.000000 50.000000

mean 0.445854 0.533345 0.447292

std 0.277608 0.213286 0.259208

min 0.000000 0.000000 0.000000

25% 0.241527 0.399260 0.274066

50% 0.441799 0.543661 0.450876

75% 0.614474 0.712221 0.634759

max 1.000000 1.000000 1.000000

encode\_data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 50 entries, 0 to 49

Data columns (total 3 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 categorical\_\_State\_California 50 non-null float64

1 categorical\_\_State\_Florida 50 non-null float64

2 categorical\_\_State\_New York 50 non-null float64

dtypes: float64(3)

memory usage: 1.3 KB

clean\_data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 50 entries, 0 to 49

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 R&D\_Spend 50 non-null float64

1 Administration 50 non-null float64

2 Marketing\_Spend 50 non-null float64

3 categorical\_\_State\_California 50 non-null float64

4 categorical\_\_State\_Florida 50 non-null float64

5 categorical\_\_State\_New York 50 non-null float64

dtypes: float64(6)

memory usage: 2.5 KB

basemodel.summary()

Out[87]:

<class 'statsmodels.iolib.summary.Summary'>

"""

OLS Regression Results

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

Dep. Variable: Profit R-squared: 0.951

Model: OLS Adj. R-squared: 0.945

Method: Least Squares F-statistic: 169.9

Date: Fri, 05 Apr 2024 Prob (F-statistic): 1.34e-27

Time: 19:05:51 Log-Likelihood: -525.38

No. Observations: 50 AIC: 1063.

Df Residuals: 44 BIC: 1074.

Df Model: 5

Covariance Type: nonrobust

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

coef std err t P>|t| [0.025 0.975]

-------------------------------------------------------------------------------------------------

const 3.659e+04 3354.540 10.909 0.000 2.98e+04 4.34e+04

R&D\_Spend 1.333e+05 7673.355 17.369 0.000 1.18e+05 1.49e+05

Administration -3547.3528 6861.263 -0.517 0.608 -1.74e+04 1.03e+04

Marketing\_Spend 1.273e+04 8087.397 1.574 0.123 -3570.409 2.9e+04

categorical\_\_State\_California 1.215e+04 2102.952 5.776 0.000 7907.672 1.64e+04

categorical\_\_State\_Florida 1.234e+04 2388.628 5.168 0.000 7530.719 1.72e+04

categorical\_\_State\_New York 1.21e+04 2158.898 5.607 0.000 7753.034 1.65e+04

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

Omnibus: 14.782 Durbin-Watson: 1.283

Prob(Omnibus): 0.001 Jarque-Bera (JB): 21.266

Skew: -0.948 Prob(JB): 2.41e-05

Kurtosis: 5.572 Cond. No. 1.42e+16

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

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 5.12e-31. This might indicate that there are

strong multicollinearity problems or that the design matrix is singular.

Out[725]:

models Scores

0 Lasso 0.950752

1 Ridge 0.949068

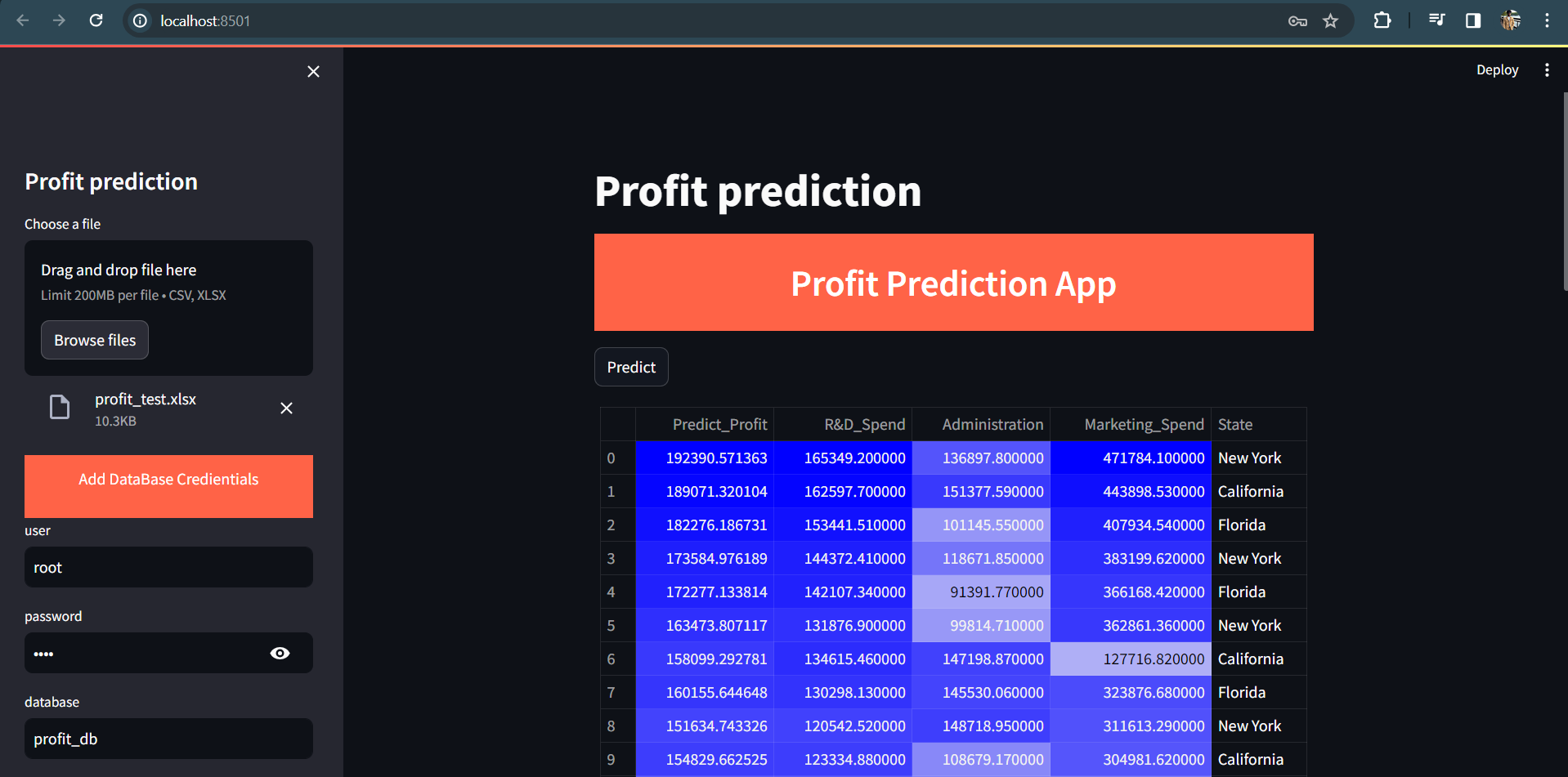
2 Elasticnet 0.803042

3 Grid\_lasso 0.950740

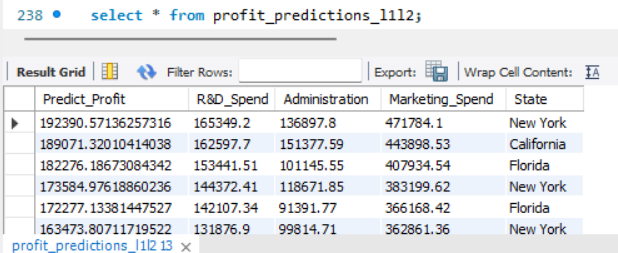
4 Grid\_ridge 0.950752

5 Grid\_elasticnet 0.950752

**Deployment of Prediction of Business Profit using Lasso, Ridge, Elastic Net Regression using Streamlit:**



**Storage of Predicted data in MySQL for continuous monitoring:**

****