Carried out by:
YASSER El KARKOURI

ID : **S450915**

Module: Predictive Modelling

Optimizing Sales and Performance

Module supervisor teacher:
Dr Stuart Barnes

20 Octobre 2023

Table of Contents

Tables of Figure

Introduction

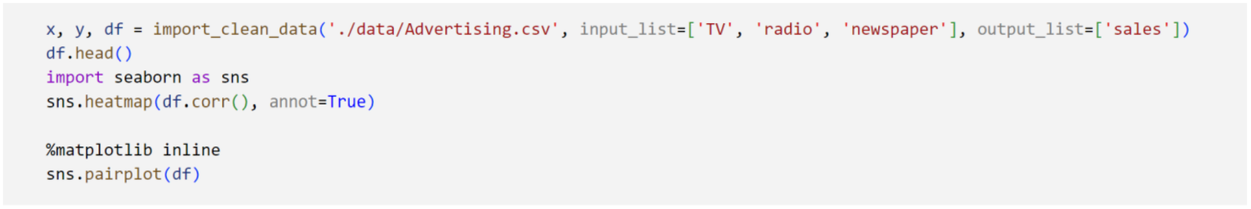
Simple linear regression is a technique for analysing bivariate data to provide insights on the linear relationship between the two variables and predict the value of the dependent variable using the other known independent variable.[1] This is achieved by drawing the line of best fit along the known values. When there are more independent variables given in the multivariate data, **multiple linear regression** technique is used. Since the technique includes multiple known independent variables, it greatly reduces the error during the prediction of the values for the dependent variable. To enhance the accuracy of the multiple linear regression model, **gradient descent** method is implemented. Gradient descent method is an optimisation algorithm used to train machine learning models and neural networks.[2]

The objectives of this project are to: (1) develop a Python program and implement gradient descent algorithm on the multi linear regression model, (2) test multiple iterations learning rates and ratio test/ train data, (3) evaluate the goodness-of-fit through r-squared test, and (4) validate the model using Python library features (Scikit-learn). Also, this project provides an opportunity for the students to apply the fundamentals of data analysis such as manipulation, visualisation and statistical process skills that will be useful for future major projects.

Context & Methodology

Description of the Datasets

Two different datasets are used in this project. Advertising.csv consists of 4 variables with 200 observations while Auto.csv dataset consists of 9 variables (only 2 variables will be used) with 397 observations. To understand the datasets better, data analysis is conducted using ‘Correlation Heatmap’ and ‘Data Correlation’ features to illustrate the strength of the relationships between the variables. In this case, Advertising dataset will be used. See **Figure 1** for the source code.



```
x, y, df = import_clean_data('./data/Advertising.csv', input_list=['TV', 'radio', 'newspaper'], output_list=['sales'])
df.head()
import seaborn as sns
sns.heatmap(df.corr(), annot=True)

%matplotlib inline
sns.pairplot(df)
```

Figure 1: Source Codes for Correlation Heatmap and Data Correlations

In **Figure 2**, the ‘Correlation heatmap’ has shown that TV-Sales (0.78), Radio-Sales (0.58) and Radio-Newspaper (0.35) are the top three highest in strength.

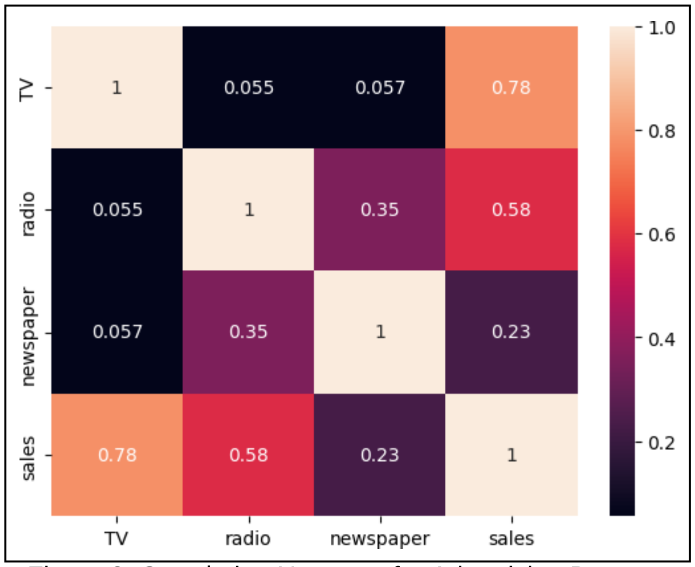


Figure 2: Correlation Heatmap for Advertising Dataset

In **Figure 3**, it is observed that TV-Sales and Radio-Sales are positively correlated since the values increase in the same direction. Other variables demonstrated neutral correlation as the values are randomly scattered and show no sign of relationship.

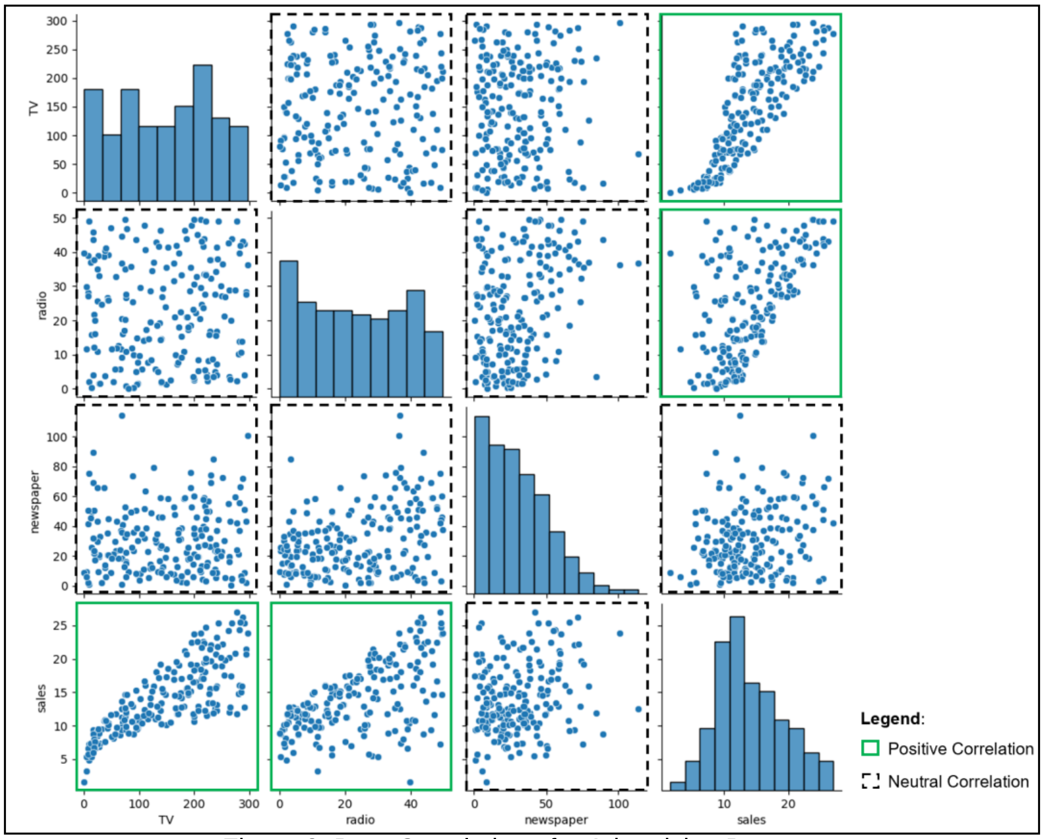


Figure 3: Data Correlations for Advertising Dataset

Multiple Regression Method

To compute the prediction, multiple linear regression technique is adopted as the given datasets (Advertising and Auto) consist of more than one variable. The mathematical equations to calculate the multiple linear regression model are as shown below. With the given datasets equation (1) can be estimated. To determine the effectiveness of the regression model to the datasets, vector $J(\theta)$ in equation (2) has to be evaluated how close it fits to the data. This results in defining a cost function defined by the mean square error between the prediction and the outcome.[3] Thus, the goal is to optimise θ*'s* by minimising the cost function to maximise the fitting of the model using gradient descent method.

$$(1): \quad y(x_i^1, \ldots, x_i^r) = \theta_0 + \theta_1 x_i^1 + \cdots + \theta_r x_i^r$$

$$(2): \quad J(\theta_0, \ldots, \theta_r) = \frac{1}{n} \sum_{i=1}^{n} (y(x_i^1, \ldots, x_i^r) - y_i)^2$$

i= number of observations (ranges from 1 to n)
j= number of features (ranges from 1 to n)

Gradient Descent Method

This method is to minimise a differentiable function, which, in this case, represents the cost function. It does so by utilising the partial derivative known as the gradient. In each iteration of this method, each parameter, denoted as theta (θ), is updated by subtracting the partial derivative of the cost function with respect to the corresponding θ. This subtraction is further multiplied by the learning rate, denoted as η. See equation (3). The learning rate signifies how quickly the algorithm converges towards the minimum values of the thetas, thus influencing the precision of the approach.

Additionally, each partial derivative of the cost function with respect to θ is computed as twice the sum of the product of the discrepancy between the predicted value and the actual value, and the respective feature value. This is performed for each sample and then divided by the total number of observations. Consequently, if, during each iteration, the cost function tends to approach a value very close to zero, it suggests that the model fits the data well. Typically, the iterative gradient descent algorithm is continued until the cost function converges, indicating that further iterations do not significantly change the model's performance. See equation (4).

$$(3): \quad \theta_j = \theta_j - \eta \frac{\partial J(\theta_0, \ldots, \theta_r)}{\partial \theta_j}$$

$$(4): \quad \frac{\partial J(\theta_0, \ldots, \theta_r)}{\partial \theta_j} = \frac{2}{n} \sum_{i=1}^{n} (y(x_i^1, \ldots, x_i^r) - y_i) . x_i^j$$

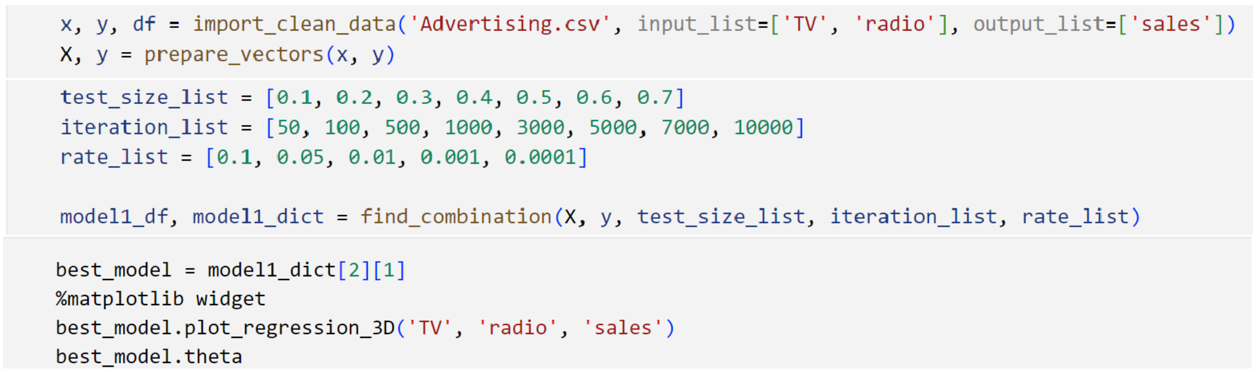
Validation of the Model

After training the regression model, it is tested using test data to check if the model predicts well with other (unseen) data. To evaluate the effectiveness of this model, the model is first tested using a test dataset (unseen data) to check if the generated model predicts well. Next, it is then assessed and compared using Scikit-learn function as part of Python's library. The function will generate regression models based on the same datasets used.

Results & Analysis

For the development of the Python program, Object Orientated Programming (OOP) approach is adopted to define a multi linear regression Class Model. The model is defined with the input and output vectors, learning rate, number of iterations and test size. The dataset is split into training and testing. After the object is created, the attributes for the theta are initialised using np.zeros(). The grad_descent (self) is developed and ready to compute the descent with the previously provided attributes. For each iteration, cost is computed with the updated theta values.

From the Advertising dataset, two variables/ predictors, 'TV' and 'Radio' are used to predict the sales. The parameters used for machine learning are shown in **Figure 4** using 'find_combination' function to compute the different parameters combinations. R-squared (R^2) and Mean Square Error (MSE) are the indicators to evaluate the performance of the regression models in machine learning.[4]



```
x, y, df = import_clean_data('Advertising.csv', input_list=['TV', 'radio'], output_list=['sales'])
X, y = prepare_vectors(x, y)
```

```
test_size_list = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]
iteration_list = [50, 100, 500, 1000, 3000, 5000, 7000, 10000]
rate_list = [0.1, 0.05, 0.01, 0.001, 0.0001]

model1_df, model1_dict = find_combination(X, y, test_size_list, iteration_list, rate_list)
```

```
best_model = model1_dict[2][1]
%matplotlib widget
best_model.plot_regression_3D('TV', 'radio', 'sales')
best_model.theta
```

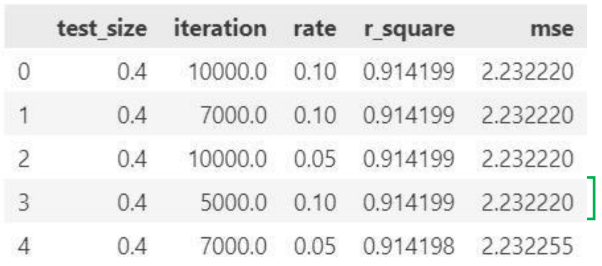
Figure 4: Source Code for Machine Learning Parameters (TV & Radio)

Figure 5 illustrates the machine learning results of Model 1 (TV & Radio) and Model 2 (TV, Radio & Newspaper). Model 1 was generated using the given equation (5) while Model 2 was generated based on equation (6). Model 2 was computed using the best parameters obtained from Model 1. The values for R^2 and MSE are relatively similar. The program has derived the optimum parameters at 5000 iterations as indicated in row 2 and row 3 (green box). This is because the as iterations approached towards 5000 times, it yields negligible differences despite using more resources. See **Figure 6**.

(5): $Sales = \theta_0 + \theta_1 * TV + \theta_2 * radio$

(6): $Sales = \theta_0 + \theta_1 * TV + \theta_2 * radio + \theta_3 * newspaper$

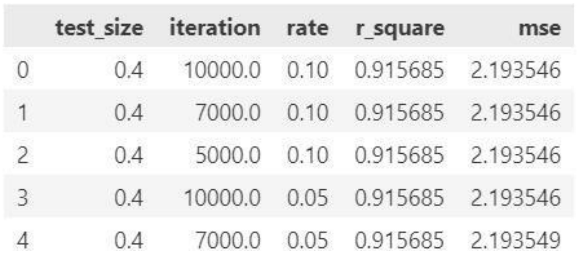
Model 1 - TV & Radio



	test_size	iteration	rate	r_square	mse
0	0.4	10000.0	0.10	0.914199	2.232220
1	0.4	7000.0	0.10	0.914199	2.232220
2	0.4	10000.0	0.05	0.914199	2.232220
3	0.4	5000.0	0.10	0.914199	2.232220
4	0.4	7000.0	0.05	0.914198	2.232255

Mse: 3.6984684000727555
R_square: 0.8685642978945274

Model 2 - TV, Radio & Newspaper



	test_size	iteration	rate	r_square	mse
0	0.4	10000.0	0.10	0.915685	2.193546
1	0.4	7000.0	0.10	0.915685	2.193546
2	0.4	5000.0	0.10	0.915685	2.193546
3	0.4	10000.0	0.05	0.915685	2.193546
4	0.4	7000.0	0.05	0.915685	2.193549

Mse: 3.862023458501355
R_square: 0.8627518989195783

Figure 5: Results of Model 1 & 2 Parameters Combinations

To enhance visualisation of the regression line and the cost function, ‘matplotlib widget’ feature is used to plot the three-dimensional regression graph as shown in **Figure 6**. It is observed that TV and Radio are positively correlated to the Sales. As both the TV and Radio budgets increase, the values of sales also increase almost linearly.

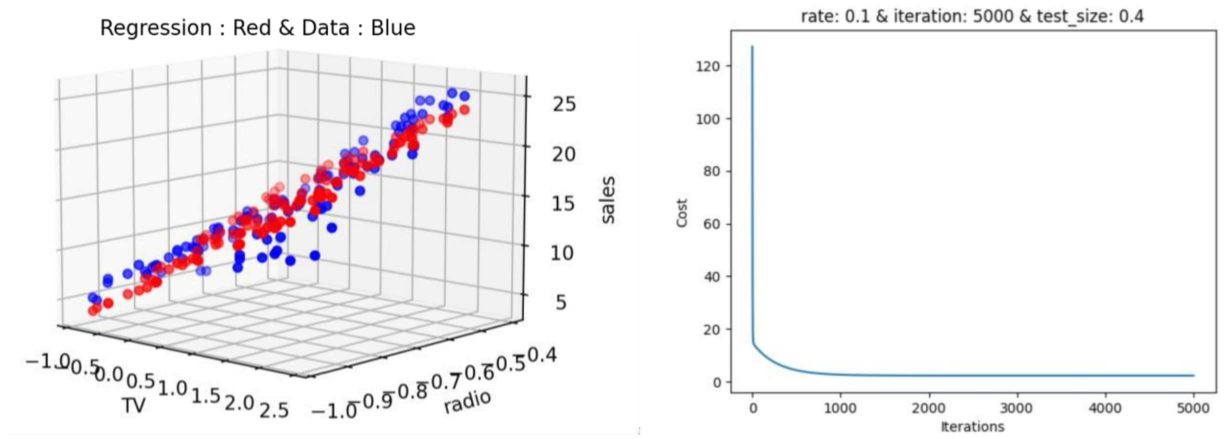


Figure 6: Prediction of TV & Radio to Sales

Figure 7 illustrates the prediction of Newspaper, Radio, and TV using various colours to represent the sales. For TV, the sales generally increase as the budget increases. For Radio and newspaper, it is observed that there is no relationship between sales and budget.

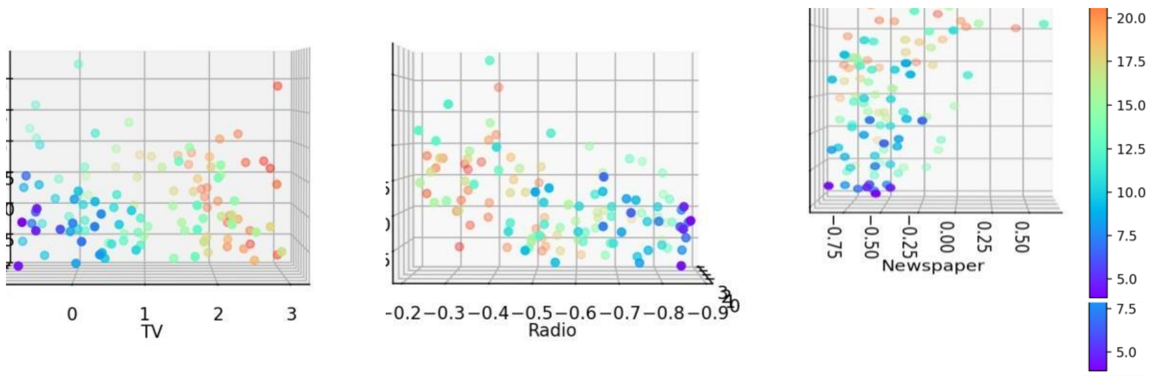
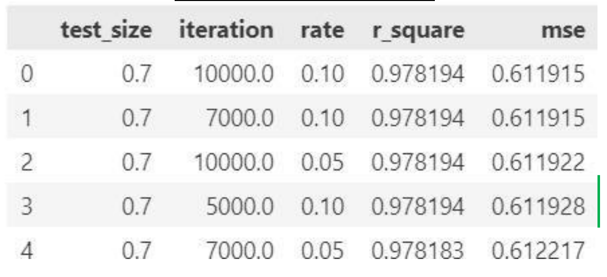


Figure 7: Scatter Plot of Training Data (TV, Radio & Newspaper)

To understand if spending money on Radio advertising increases the effectiveness of TV advertising, equation (7) is used to compute the results. **Figure 8** illustrates the machine learning results of Model 3 (TV & Radio). It is observed that the optimum parameters at 5000 iterations yield negligible differences despite using more resources. See row 2 (in green box).

$$(7): \qquad Sales = \theta_0 + \theta_1 * TV + \theta_2 * radio + \theta_3 * (radio * TV)$$

Model 3 – TV & Radio



	test_size	iteration	rate	r_square	mse
0	0.7	10000.0	0.10	0.978194	0.611915
1	0.7	7000.0	0.10	0.978194	0.611915
2	0.7	10000.0	0.05	0.978194	0.611922
3	0.7	5000.0	0.10	0.978194	0.611928
4	0.7	7000.0	0.05	0.978183	0.612217

Mse: 1.015088013948224
R_square: 0.9616153473720442

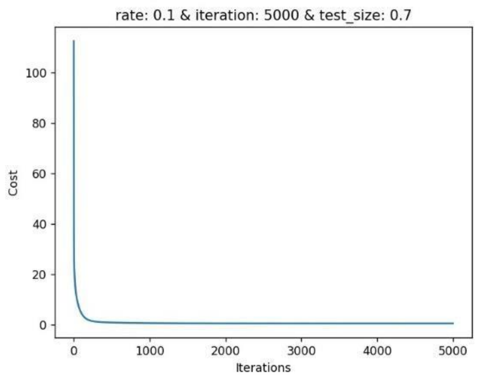
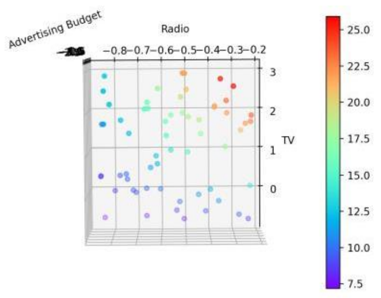


Figure 8: Results of Model 3 Parameters Combinations

Figure 9 illustrates the prediction of Radio advertising to effectiveness of TV advertising. It is observed that as more money is spent on Radio advertising, TV advertising increases.

Top View



Side View

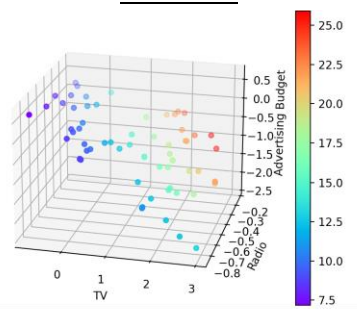
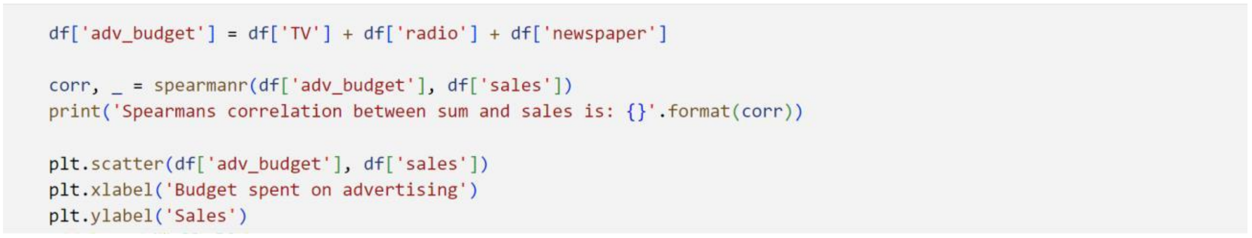


Figure 9: Graph of TV and Radio against Advertising Sales

To determine how strong the relationship between the advertising budget and sales is, ‘Spearmans Correlation’ function can be used (this function can assess monotonic relationship unlike ‘Pearson’s Correlation’ function which can only measure linear relationship).[5] This is achieved by computing the total advertising budgets spent against sales. See **Figure 10** for source code and **Figure 11** for the visual representation. **Figure 11** depicts that as more budget is spent on advertising, the sales increase almost linearly. The correlation between these two variables is ~0.88 (1 = identical), confirmed that they have strong relationships between each other.

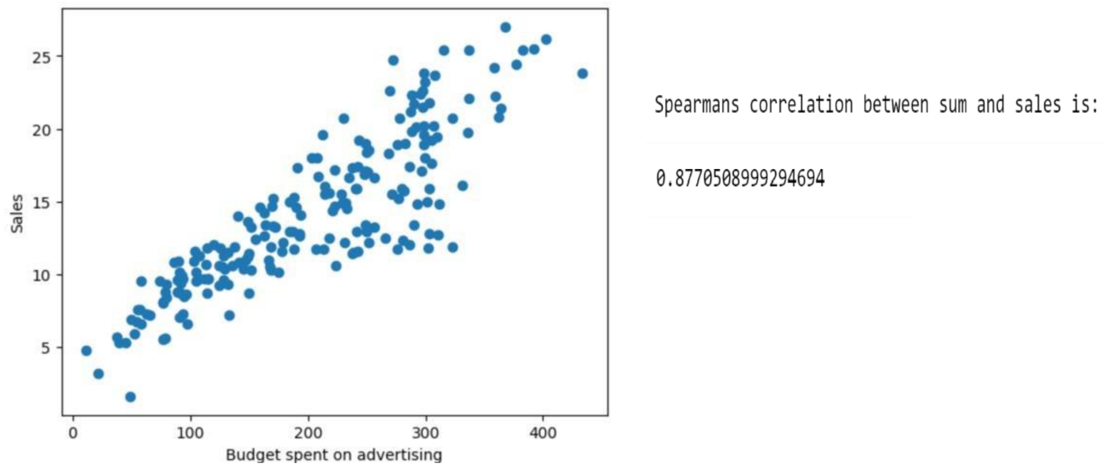


```
df['adv_budget'] = df['TV'] + df['radio'] + df['newspaper']

corr, _ = spearmanr(df['adv_budget'], df['sales'])
print('Spearmans correlation between sum and sales is: {}'.format(corr))

plt.scatter(df['adv_budget'], df['sales'])
plt.xlabel('Budget spent on advertising')
plt.ylabel('Sales')
```

Figure 10: Source Code for Spearmans Correlation between Adveritsing Budget and Sales



```
Spearmans correlation between sum and sales is:

0.8770508999294694
```

Figure 11: Graph of Budget Spent on Advertising against Sales

To validate the effectiveness of the regression model (manually constructed), Scikit-learn (SKlearn) as part of Python’s library feature, will be used to generate the required models. The models consist of linear, quadratic and degree 5, will be generated using the same dataset file (Auto.csv), mpg as the target and horsepower as the predictor**. Figures 12**, **13** and **14** depict the various models for comparisons as shown below. In general, the data points (blue dots) on the graph may not be identical as selection of data for training and testing are at random, thus the differences.

In **Figure 12**, it is observed that the regression models created manually and by SKlearn are largely identical with a negligible difference in R^2 value of 0.000042 (~0.0042%). As the models are designed to fit linearly, they are unable to accommodate small clusters of data points (green box). This illustrates a scenario where a linear line fits a parabolic pattern of the data points using degree one. The manual constructed model is assessed to be better than the Sklearn model for linear regression.

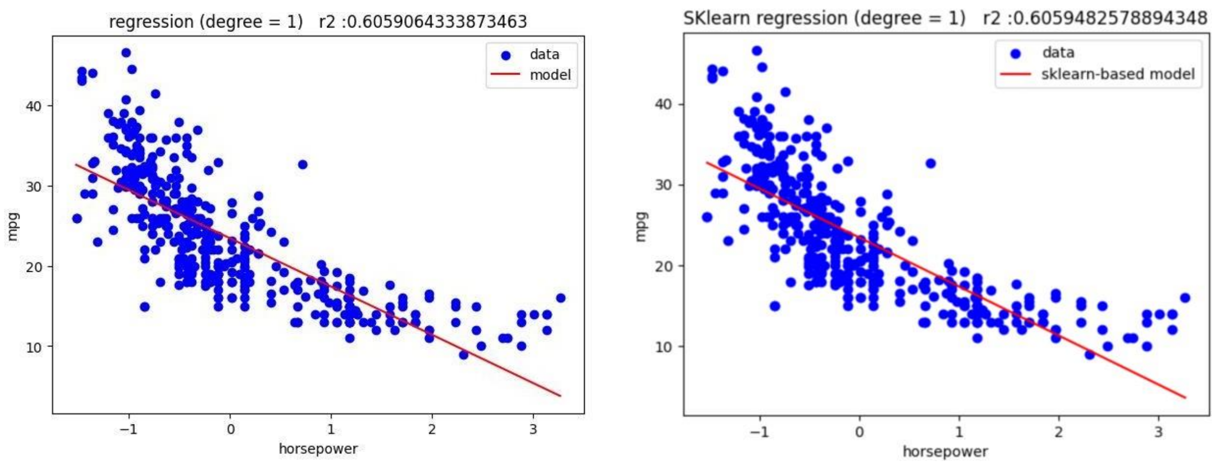


Figure 12: Comparison of Regression Models (Degree 1)

In **Figure 13**, both R^2 values improved by 0.08 (~8%) as compared to **Figure 12**. The quadratic models are now able to fit more of the smaller cluster data points. SK learn model is better than the manual model.

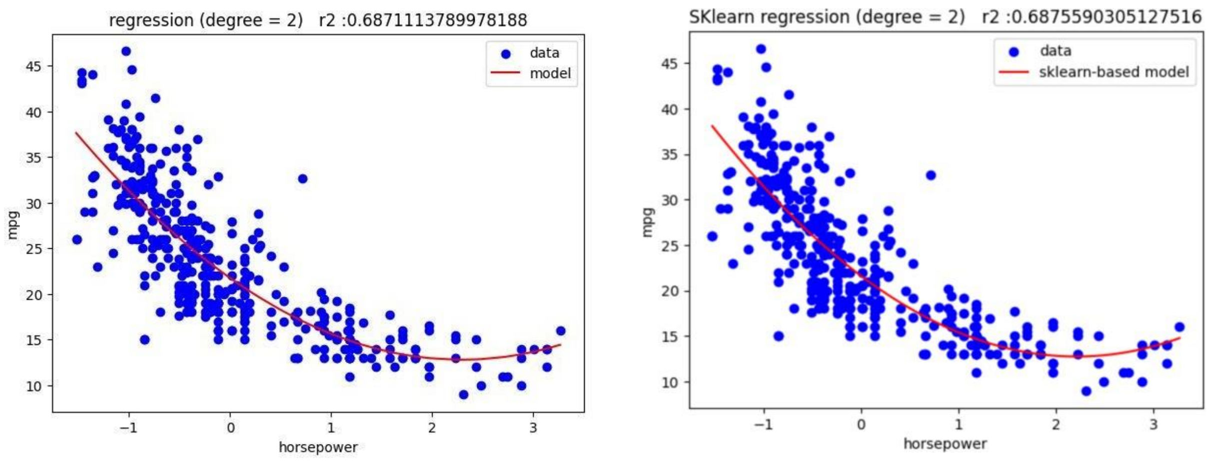
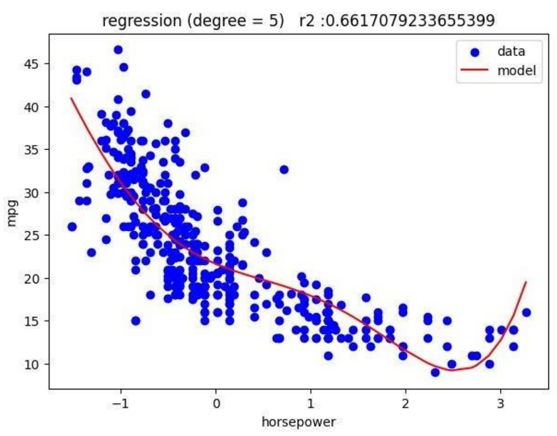


Figure 13: Comparison of Regression Models (Degree 2)

In **Figure 14**, it is observed that the R^2 values for manual and SKlearn model regression increases by 0.08 (~8%). Having more degree to fit more data points, the models have achieved higher R^2 values.



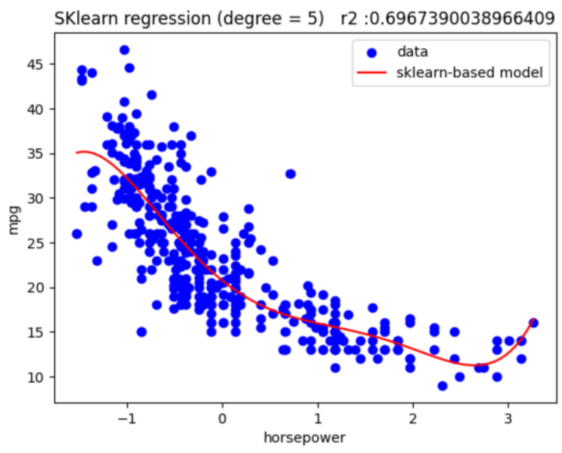


Figure 14: Comparison of Regression Models (Degree 5)

Conclusion

About the Project

The objectives of this project have been met. The regression models have been developed, trained, tested, and validated using SKlearn function. The manual constructed regressions models are largely comparable to the SKlearn models. In this scenario, using the Sklearn models will be more resource efficient than manual model since the value of R^2 does not deviate too much.

As the degree of polynomial regression increases, it does not necessarily generate better results. Thus, it is important to identify the best parameters (miniMax) minimum resources to generate the best outcome.

R-squared is one indicator to illustrate if a regression model fits the data points. For example, the indicator can show if the model is underfitting or overfitting based on the data points. However, the evaluation can be improved if other indicators like MSE are used.

Overall Reflection

Regression analysis is a very useful tool to study dependent and independent variables. It allows the user to describe, predict and estimate the relationships and draw realistic conclusions from the dataset. A few key considerations to note while conducting regression analysis, the type and number of dependent/ independent variables, and size of the dataset. Some examples of regressions are logistic, ridge, Bayesian, Poisson etc. Using an inappropriate type will result in inaccurate conclusions. As the number of dependent/ independent variables increases, it may result in overfitting of the regression model. When the dataset is huge, it enhances machine learning to learn better as compared to a smaller dataset which limits the training.

Future Works

Although the objectives have been achieved, there are some recommended future works to further improve the accuracy of the regression models. Firstly, adding limits to the gradient descent method such as bound constraints to limit the values of model parameters (coefficient) within desired bounds. Lastly, exploring the gradient descent method with adaptative learning rate to converge quickly to the optimised theta (θ), thus reduces the time complexity of the algorithms.

References

[1] [3] Pardoe, I. (2012). *Applied Regression Modeling* (2 ed.). New Jersey: John Wiley. doi:10.1002/9781118345054

[2] Saad Hikmat, H., & Adnan Mohsin, A. (2021). Comparison of Optimization techniques based on Gradient Descent Algorithm: A REVIEW. *PalArch's Journal of Archaeology of Egypt / Egyptology, 18*(4), 2715–2721. Retrieved Oct 2023, from https://archives.palarch.nl/index.php/jae/article/view/6705/6490

[4] Minitab. (2013, May 30). *Minitab*. (Minitab) Retrieved Oct 2023, from Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit?: https://blog.minitab.com/en/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit#:~:text=In%20general%2C%20the%20higher%20the,the%20model%20fits%20your%20data.

[5] Brownlee, J. (2020, Aug 20). *Machine Learning Mastery*. (Guiding Tech Media) Retrieved Oct 2023, from How to Calculate Correlation Between Variables in Python: https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/#:~:text=Positive%20Correlation%3A%20both%20variables%20change,variables%20change%20in%20opposite%20directions.

Appendix

A: Python Source Code (Model Class & Utils.py)

B: Contributions of Members

Appendix A (Python Source Code)

Model Class

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns                          #not used (instructions commented)
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split      #not used (instructions commented)
from sklearn.linear_model import LinearRegression

class Model:
    """ A model is defined with its dataset, learning rate, number of iterations and test size. """
    def__init_(self, X, y, learning_rate, iterations, test_size):
        self.X = X                  #input data
        self.y = y                  #output data
        self.X_train = None         #will be set after data is splitted into train and test set
        self.X_test = None          #will be set after data is splitted into train and test set
        self.y_train = None         #will be set after data is splitted into train and test set
        self.y_test = None          #will be set after data is splitted into train and test set
        self.X_origin = X           #store the original X before transformation
        self.X_origin_test = None
        self.X_origin_train = None
        self.m = len(y)
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.test_size = test_size #proportion of the test set
        #initialise theta to 0
        self.theta = np.zeros((self.X.shape[1],1))       #np.random.randn((X.shape[1]+1),1)  to set theta to random values
        self.learning_cost = []     #list that will contain the cost for each iteration of the gradient descent
        self.r_square = 0           #coefficient r^2
        self.mse = 0                #mean square error

    def model(self):
        return self.X_train.dot(self.theta)

    def split_data(self):
        """Splits the data into train and test sets and adds the bias (Column of 1s))"""
        #split data into train and test
        self.X_train = self.X[:int(self.m*(1-self.test_size))]
        self.X_test = self.X[int(self.m*(1-self.test_size)):]
        self.y_train = self.y[:int(self.m*(1-self.test_size))]
        self.y_test = self.y[int(self.m*(1-self.test_size)):]
        self.X_origin_test = self.X_origin[int(self.m*(1-self.test_size)):]
        self.X_origin_train = self.X_origin[:int(self.m*(1-self.test_size))]

        #add bias
        self.X_train = np.c_[self.X_train, np.ones(self.X_train.shape[0])]
        self.X_test = np.c_[self.X_test, np.ones(self.X_test.shape[0])]
        self.X_origin_test = np.c_[self.X_origin_test, np.ones(self.X_origin_test.shape[0])]
        self.X_origin_train = np.c_[self.X_origin_train, np.ones(self.X_origin_train.shape[0])]
        #update m
        self.m = len(self.y_train)

        #update theta
        self.theta = np.random.randn((self.X_train.shape[1]),1)
        #self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(self.X, self.y, test_size=self.test_size,
random_state=42)
        #self.m = len(self.y_train)
        #self.theta = np.random.randn((self.X_train.shape[1]),1)
        return self.X_train, self.X_test, self.y_train, self.y_test

    def cost_computing(self):
        """Computes the cost function for a given theta at a given iteration"""
        return 1/(self.m) * np.sum((self.model() - self.y_train)**2)

    def gradient(self):
        """Computes the gradient of the cost function for a given theta at a given iteration"""
        return 1/self.m * self.X_train.T.dot(self.model() - self.y_train)

    def grad_descent(self):
        """Runs the gradient descent and returns the theta and the cost for each iteration"""
        self.learning_cost = np.zeros(self.iterations)
        for i in range(self.iterations):
            self.theta = self.theta - self.learning_rate * self.gradient()
            self.learning_cost[i] = self.cost_computing()
        return self.theta, self.learning_cost

    def compute_r_square(self):
        """Returns the coefficient of determination R^2 of the prediction."""
        #self.r_square = r2_score(self.y_train, self.model())
        self.r_square = 1 - np.sum((self.y_train - self.model())**2) / np.sum((self.y_train - np.mean(self.y_train))**2)
        return self.r_square

    def compute_mean_square_error(self):
        """Returns the mean squared error of the prediction."""
        #self.mse = mean_squared_error(self.y_train, self.model())
        self.mse = np.sum((self.y_train - self.model())**2) / self.m
        return self.mse
```

```
def compute_regression(self):
    """Split the dataset, run the gradient descent and compute the metrics"""
    self.split_data()
    self.grad_descent()
    #compute r_square and mse
    self.compute_r_square()
    self.compute_mean_square_error()

def test_model(self):
    """Use the test set to evaluate the model"""
    predictions = self.X_test.dot(self.theta)
    # Calculate metrics
    mse = np.sum((self.y_test - predictions)**2) / len(self.y_test)
    r_square = 1 - np.sum((self.y_test - predictions)**2) / np.sum((self.y_test - np.mean(self.y_test))**2)
    return mse, r_square, predictions

def sklearn_regression(self):
    reg = LinearRegression().fit(self.X_train, self.y_train) #fit the model
    predictions = reg.predict(self.X_test) #predict
    #theta
    theta = reg.coef_
    #The mean squared error
    mse = mean_squared_error(self.y_test, predictions)
    #r2
    r2 = r2_score(self.y_test, predictions)
    return theta, mse, r2

def transform(self, degree):
    """Transform the input data into polynomial features"""
    self.X_origin = self.X          #save the original X before transformation
    m = self.X.shape[0]             #Get the number of rows in self.X
    X_transform = np.ones((m, 1)) #initialize with ones for the bias term
    x_power_rechaped = 0
    for j in range(1, degree + 1):
        x_power = np.power(self.X, j)
        x_power_rechaped = x_power.reshape(-1, 1)
        X_transform = np.append(X_transform, x_power_rechaped, axis=1)  #append the transformed features
    self.X = X_transform
    return X_transform

def plot_cost(self):
    """Plot the cost for each iteration of the gradient descent"""
    plt.plot(range(self.iterations), self.learning_cost)
    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.title('rate: {} & iteration: {} & test_size: {}'.format(self.learning_rate, self.iterations, self.test_size))
```

```
        plt.show()

    def plot_regression_3D(self, xlabel, ylabel, zlabel):
        """Plot the regression line with the data in 3D"""
        #%matplotlib widget
        fig = plt.figure()
        ax = fig.add_subplot(111,projection='3d')
        ax.scatter(self.X_train[:,0], self.X_train[:,1], self.y_train, c='b')
        ax.scatter(self.X_train[:,0], self.X_train[:,1], self.model(), c='r')
        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.set_zlabel(zlabel)
        ax.set_title('Regression : Red & Data : Blue')

    def plot_regression_2D(self, xlabel, ylabel, title):
        """Plot the regression line with the data in 2D"""
        predictions = self.model()

        #Sort the values of x before line plot (to get a nice line)
        sorted_idx = self.X_origin_train[:,0].flatten().argsort()
        x_test_sorted = self.X_origin_train[:,0][sorted_idx]
        pred_sorted = predictions[sorted_idx]

        # Create a single plot
        plt.figure(figsize=(7, 5))
        plt.scatter(self.X_origin_train[:,0], self.y_train[:,0], color='blue', label='data')
        plt.plot(x_test_sorted, pred_sorted, color='red', label='model')
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.title(title+' r2 :' + str(self.r_square))
        plt.legend()
        plt.show()

    def show_data(self):
        """Prints the metrics and the theta"""
        print('theta: ', self.theta)
        print('last cost: ', self.learning_cost[-1])
        print('r_square: ', self.r_square)
        print('mse: ', self.mse)

    def get_r_square(self):
        """Returns the coefficient of determination R^2 of the prediction."""
        return self.r_square

if __name__ == '__main__':
    print("Nothing to do here, it's just a Class ;)")
```

Utils.py

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

from  Class.ModelClass import * # Importing the Model class from ModelClass.py

def import_clean_data(path,input_list, output_list):
    """Import the data from the csv file and clean it, then randomise the order of the rows."""
    df = pd.read_csv(path)
    #remove all columns that are not in the input list and the output list
    df = df[input_list + output_list].apply(pd.to_numeric, errors='coerce')
    df = df.dropna()
    #drop all cells with char
    df.dropna(subset=input_list + output_list, inplace=True)
    df.reset_index(drop=True, inplace=True)
    #randomise the order of the rows
    #df = df.sample(frac=1).reset_index(drop=True)
    x = df[input_list].values
    y = df[output_list].values
    return x, y, df

def prepare_vectors(x, y):
    """Takes the input and output vectors and returns the normalised input vector and the output vector reshaped."""
    #normalise
    X = (x - np.mean(x)) / np.std(x)
    y = y.reshape(y.shape[0],1)
    return X, y

def find_combination(X, y, test_size_list, iteration_list, rate_list ):
    """Returns a dataframe with the metrics for each combination of test_size, iteration and rate."""
    model_dict = {}
    model_df = pd.DataFrame(columns=['test_size', 'iteration', 'rate', 'r_square', 'mse'])  #create empty dataframe

    #for each combination of test_size, iteration and rate, compute the model and add it to the model_dict
    for test_size in test_size_list:
        for iteration in iteration_list:
            for rate in rate_list:
                model = Model(X, y, rate, iteration, test_size)
                model.compute_regression()
                model_dict[(test_size, iteration, rate)] = model
    #sort model_dict by r_square, if r_square is the same, sort by iteration Descending
    model_dict = sorted(model_dict.items(), key=lambda x: x[1].get_r_square(), reverse=True)

    #convert dict to dataframe
    for i in range(len(model_dict)):
        model_df.loc[i] = [model_dict[i][0][0], model_dict[i][0][1], model_dict[i][0][2], model_dict[i][1].get_r_square(),
model_dict[i][1].mse]

    return model_df, model_dict

def sk_compute_plot(X, y, degree, xlabel, ylabel, title):
    """Compute the polynomial regression using sci kit learn and plot the graph"""
    sk_poly = PolynomialFeatures(degree)
    X = X.reshape(-1, 1)  # Reshape X to a 2D array
    X_poly = sk_poly.fit_transform(X)

    sk_model = LinearRegression()
    sk_model.fit(X_poly, y)
    r_squared_sklearn = r2_score(y, sk_model.predict(X_poly))

    #generate points for plotting the regression line using sci kit learn
    x = np.linspace(X.min(), X.max(), 100)
    x_poly = sk_poly.transform(x.reshape(-1, 1))

    #plot the graph
    plt.scatter(X, y, color='blue', label='data')
    plt.plot(x, sk_model.predict(x_poly), color='red', label='sklearn-based model')
    plt.title(title + '  r2 :' + str(r_squared_sklearn))
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.legend()
    plt.show()

if __name__ == '__main__':
```