

Machine Learning

Introduction

In the dynamic landscape of machine learning and forecasting, applying regression techniques is a pivotal and versatile tool. This reading material is a gateway into the intricate world where statistical methodologies meet advanced computational algorithms to uncover patterns, predict trends, and make informed decisions. Here, we continue with regression analysis and delve into its applications within the expansive domain of machine learning, where predictive modeling and classification problems take center stage. Additionally, we explore how regression methodologies are crucial in forecasting, offering invaluable insights into future trends, behaviors, and outcomes.

In recent times, machine learning (ML) and data science fields have witnessed a remarkable surge in popularity. This trend is expected to continue, with substantial exponential growth anticipated in the coming years. Machine learning, a discipline within artificial intelligence (AI) and computer science, has garnered significant attention. It revolves around using data and sophisticated algorithms to emulate the learning processes observed in humans, progressively refining its accuracy over time. Murphy (2012) defines ML as the “set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty”. Machine learning is broadly categorized into two main types: Supervised and Unsupervised. A dependent (or target) variable is present in supervised learning, and problems include Classification and Regression.

In contrast, Unsupervised learning involves algorithms learning autonomously without supervision or a dependent variable. In this mode, algorithms uncover hidden patterns and relationships within the provided data, and problems include dimensionality reduction and clustering. We focus on supervised learning and discuss using multiple regression and logistic regression in machine learning.

Machine Learning

Machine learning algorithms undergo training through statistical methods to facilitate classifications or predictions. Multiple Linear Regression is employed for making predictions, while Logistic Regression is specifically designed for classifications. In both scenarios covered, the machine learning algorithm performance assessment relies on the train-test split technique. This method involves dividing the dataset into two subsets. The initial subset, the training dataset, is utilized for model fitting, while the second subset is the test dataset. The purpose is to evaluate the machine learning model's performance on new data, specifically the test dataset—data not utilized during the

model training phase. The train-test procedure is particularly suitable when dealing with a sufficiently large dataset. Within this process, a decision needs to be made regarding the size of both the train and test sets, typically expressed as a percentage ranging from 0 to 1. For instance, a training set size of 0.70 (70 percent) implies that the remaining 0.30 (30 percent) is allocated to the test set. Commonly employed split percentages include the first scenario with Train: 80%, Test: 20%, the second with Train: 67%, Test: 33%, and the third with Train: 50%, Test: 50%. Now that we have comprehended the train-test split model evaluation methodology, let us explore how to implement it using Python.

The scikit-learn library implements the train-test split procedure via the `train_test_split()` function. The function takes a loaded dataset as input and returns the dataset split into two subsets.

Some of the steps in solving machine learning algorithms are:

1. Identify the significant variables in the model.
2. Split the dataset into training and test sets.
3. Fit the regression model to the training set.
4. Fitting the regression model developed in step 3 to the testing set.
5. The last step is checking the performance of the model. This is done by comparing the predicted and actual values in the testing set.

Multiple Regression in Machine Learning

In multiple linear regression, the dependent (Y) is continuous, but the independent variable may be of continuous or categorical form. Each independent variable must model the linear relationship with the dependent variable. The major applications of Multiple Linear Regression are assessing the effectiveness of independent variables on prediction and predicting the impact of changes in independent variables. Assumptions for Multiple Linear Regression are, first, a linear relationship should exist between the Independent and dependent variables; second, the regression residuals must be normally distributed; and third, there should be little or no multicollinearity between the independent variable. For implementing MLR for machine learning, we have taken the example discussed in Chapter 9, Example 6.

The following steps are followed to implement MLR for machine learning. The steps are explained with by solving example 1. In example 1, the data file is AREx1,

Table 1: Python Functions used in Example 1

Steps	Description and python functions
Identify the significant variables in the model.	There are 8 independent variables in this problem first we run MLR on whole dataset taking CarPrice as dependent variable In the output, we find that three variables are not significant they are 'Region_North', 'Region_South', 'Region_West'. They are removed for further analysis. For step 2, the following X and Y variables are taken. <pre>X_cols = ['Incometax', 'SECClass_A2', 'SECClass_B1', 'SECClass_B2', 'SECClass_C'] X1 = ex1new[X_cols] Y1=ex1new['CarPrice']</pre>
Split the dataset into training and test set.	sklearn library is used split X1 and Y1, in x train, y train, x test and y test. The data is divided into a 70/30 ratio. <pre>from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X1, Y1, test_size = 0.3)</pre> Original Data Set (1000, 5) (1000,) Train Data Set (700, 5) (700,) Test Data Set (300, 5) (300,)
Fit the regression model to the training set.	<pre>X_train = sm1.add_constant(X_train) modelf = sm1.OLS(y_train,X_train).fit() print(modelf.summary())</pre>
Fitting the MLR model develop ed in step 2 to the testing set.	<pre>X_test = sm1.add_constant(X_test) testdf = pd.DataFrame({ "actual_Y_Test": y_test, "predict_test": modelf.predict(X_test) })</pre>
The last step is checking the performance of the model. This	Calculating mean square error, r square and root mean square error from the data.

is done by comparing the predicted and actual values in the testing set.	<pre> mean_squared_error(testdf.actual_Y_Test, testdf.predict_test) r2_score(testdf.actual_Y_Test, testdf.predict_test) np.sqrt(metrics.mean_squared_error(testdf.actual_Y_Test, testdf.predict_test)) </pre>
--	---

We have calculated the three scores for the training dataset and test dataset. The score shows that our model is 93% accurate with the training dataset and 94% accurate with the test dataset.

Example 1: Data file ARex1.csv as data on 1000 families; data is of their socioeconomic class, region, income tax paid, and the price of the latest car purchased by them. It is assumed that socioeconomic class, region, and income tax paid impact the price of latest car purchased by them. Use Python to answer the following questions, mentioning all the steps and codes in word file.

- Determine the significant independent variables using regression at the 0.05 level of significance—state multiple regression equations with car price as the dependent variable.
- Split the data into training and testing sets. Use 70% for training and 30% for testing.
- Fit the regression model on the training set.
- Predict the values in the training set based on the coefficients developed in the training data set.
- Calculate mean squared error and coefficient of determination, and set root mean square error between predicted and actual values in the train and test data sets.

Hints and Interpretation:

For part a, the multiple regression equation is

$\text{CarPrice} \sim \text{Incometax} + \text{SECClass_A2} + \text{SECClass_B1} + \text{SECClass_B2} + \text{SECClass_C}$

Since P-value lower than 0.05 level of significance, all the independent variables are significant. It can be seen from Python output line 9.

For part b, both the training and test set are using Python line of code 7.

For part c, the regression model is built using Python line of code 9. See the output line of code 9.

For part d, the model prediction on the training set in Python Input and output line of code 10. In the training set, the performance are as follows:

Mean squared error: 4845868352.56
 Coefficient of determination: 0.93
 Set Root Mean Square Error: 69612.27

For part e, we use the test set to make the prediction in Python Input and output line of code 12. In the training set, the performance are as follows:

Mean squared error: 4170506767.52
 Coefficient of determination: 0.94
 Root Mean Square Error: 64579.46

Table 2: Python Code of Example 1

In [1]	import pandas as pd import matplotlib.pyplot as plt import numpy as np import seaborn as sns from scipy import stats from pandas import plotting from sklearn.linear_model import LinearRegression from sklearn.metrics import mean_squared_error, r2_score from sklearn import metrics																														
In [2]	ex1=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/BusStatUP/Datafiles/ARex1.csv') ex1.columns = ex1.columns.str.replace(' ', '') ex1.head()																														
Out [2]	<table><tr><th>FamilyNo</th><th>SECClass</th><th>CarPrice</th><th>Incometax</th><th>Region</th></tr><tr><td>0 1</td><td>A1</td><td>1480990</td><td>1184792.00</td><td>West</td></tr><tr><td>1 2</td><td>A1</td><td>1437384</td><td>1149907.20</td><td>South</td></tr><tr><td>2 3</td><td>A1</td><td>1484581</td><td>1128281.56</td><td>South</td></tr><tr><td>3 4</td><td>A2</td><td>1426385</td><td>1126844.15</td><td>East</td></tr><tr><td>4 5</td><td>A2</td><td>1477912</td><td>1108434.00</td><td>South</td></tr></table>	FamilyNo	SECClass	CarPrice	Incometax	Region	0 1	A1	1480990	1184792.00	West	1 2	A1	1437384	1149907.20	South	2 3	A1	1484581	1128281.56	South	3 4	A2	1426385	1126844.15	East	4 5	A2	1477912	1108434.00	South
FamilyNo	SECClass	CarPrice	Incometax	Region																											
0 1	A1	1480990	1184792.00	West																											
1 2	A1	1437384	1149907.20	South																											
2 3	A1	1484581	1128281.56	South																											
3 4	A2	1426385	1126844.15	East																											
4 5	A2	1477912	1108434.00	South																											
In [3]	X_parameters = ex1.columns X_parameters																														
Out [3]	Index(['FamilyNo', 'SECClass', 'CarPrice', 'Incometax', 'Region'], dtype='object')																														
In [4]	categorical_variables = ['SECClass', 'Region'] ex1new= pd.get_dummies(ex1[X_parameters], columns = categorical_variables, drop_first = True) ex1new.head()																														
Out	FamilyNoCarPriceIncometax SECClass_A2 SECClass_B1 SECClass_B2 SECClass_CRegion_NorthRegion_SouthRegion_West																														

t	0	1	1480990	1184792.00	0	0	0	0	0	0	1
[4]	1	2	1437384	1149907.20	0	0	0	0	0	1	0
	2	3	1484581	1128281.56	0	0	0	0	0	1	0
	3	4	1426385	1126844.15	1	0	0	0	0	0	0
	4	5	1477912	1108434.00	1	0	0	0	0	1	0
In	model = sm.ols('CarPrice~Incometax+SECClass_A2+SECClass_B1+SECClass_B2+SECClass_C+Region_North+Region_South+Region_West', data=ex1new).fit() print(model.summary())										
Out											
[5]											
In	X_cols = ['Incometax', 'SECClass_A2', 'SECClass_B1', 'SECClass_B2', 'SECClass_C'] X1 = ex1new[X_cols] Y1=ex1new['CarPrice']										
[6]											
In	from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X1, Y1, test_size = 0.3)										
[7]											
In	print("Original Data Set") print (X1.shape, Y1.shape) print("\nTrain Data Set") print (X_train.shape, y_train.shape) print("\nTest Data Set") print (X_test.shape, y_test.shape)										
[8]											
Out	Original Data Set (1000, 5) (1000,) Train Data Set (700, 5) (700,) Test Data Set (300, 5) (300,)										
[8]											
In	X_train = sm1.add_constant(X_train) modelf = sm1.OLS(y_train,X_train).fit() print(modelf.summary())										
[9]											
Out	OLS Regression Results =====										
[9]	=====										
	Dep. Variable: CarPrice R-squared: 0.938										
	Model: OLS Adj. R-squared: 0.937										
	Method: Least Squares F-statistic: 2097.										
	Date: Wed, 15 Jun 2022 Prob (F-statistic): 0.00										

	Time: 11:38:22 Log-Likelihood: -8775.2 No. Observations: 700 AIC: 1.756e+04 Df Residuals: 694 BIC: 1.759e+04 Df Model: 5 Covariance Type: nonrobust ===== ===== coef std err t P> t [0.025 0.975] ----- const 1.748e+05 2.95e+04 5.931 0.000 1.17e+05 2.33e+05 Incometax 1.1874 0.035 34.103 0.000 1.119 1.256 SECClass_A2 -4.508e+04 1.19e+04 -3.785 0.000 -6.85e+04 -2.17e+04 SECClass_B1 -4.662e+04 1.51e+04 -3.086 0.002 -7.63e+04 -1.7e+04 SECClass_B2 -4.622e+04 1.25e+04 -3.696 0.000 -7.08e+04 -2.17e+04 SECClass_C -7.893e+04 1.92e+04 -4.104 0.000 -1.17e+05 -4.12e+04 ===== ===== Omnibus: 28.740 Durbin-Watson: 1.936 Prob(Omnibus): 0.000 Jarque-Bera (JB): 31.355 Skew: 0.516 Prob(JB): 1.55e-07 Kurtosis: 3.102 Cond. No. 9.48e+06 ===== =====
In [10]]	traindf = pd.DataFrame({ "actual_Y_Train": y_train, "predict_train": modelf.predict(X_train) }) traindf.head(5)
Ou t [10]]	actual_Y_Train predict_train 668 760149 6.881792e+05 782 584007 6.203793e+05 625 647837 7.133898e+05 759 652979 6.399137e+05 193 1042721 1.068804e+06
In [11]]	X_test = sm1.add_constant(X_test) testdf = pd.DataFrame({ "actual_Y_Test": y_test, "predict_test": modelf.predict(X_test) }) testdf.head(5)
Ou t [11]]	actual_Y_Train predict_train 668 760149 6.881792e+05 782 584007 6.203793e+05 625 647837 7.133898e+05 759 652979 6.399137e+05

	193 1042721 1.068804e+06
In [12]]	<pre>X_test = sm1.add_constant(X_test) testdf = pd.DataFrame({ "actual_Y_Test": y_test, "predict_test": modelf.predict(X_test) }) testdf.head(5)</pre>
Out [12]]	<pre>actual_Y_Test predict_test 956 479334 4.499638e+05 846 519886 5.422797e+05 962 485809 4.431396e+05 395 987934 8.397116e+05 202 1096822 1.062915e+06</pre>
In [13]]	<pre>print("Train Data Set") print("Mean squared error: %.2f" % mean_squared_error(traindf.actual_Y_Train, traindf.predict_train)) print("Coefficient of determination: %.2f" % r2_score(traindf.actual_Y_Train, traindf.predict_train)) print("Set Root Mean Square Error: %.2f"% np.sqrt(metrics.mean_squared_error(traindf.actual_Y_Train, traindf.predict_train))))</pre>
Out [13]]	<pre>Train Data Set Mean squared error: 4845868352.56 Coefficient of determination: 0.93 Set Root Mean Square Error: 69612.27</pre>
In [14]]	<pre>print("Test Data Set") print("Mean squared error: %.2f" % mean_squared_error(testdf.actual_Y_Test, testdf.predict_test)) print("Coefficient of determination: %.2f" % r2_score(testdf.actual_Y_Test, testdf.predict_test)) print("Root Mean Square Error: %.2f" % np.sqrt(metrics.mean_squared_error(testdf.actual_Y_Test, testdf.predict_test)))</pre>
Out [14]]	<pre>Test Data Set Mean squared error: 4170506767.52 Coefficient of determination: 0.94 Root Mean Square Error: 64579.46</pre>

Logistic Regression in Machine Learning

Logistic regression is a statistical method used in machine learning to analyze and classify data. It is used to predict the probability of a binary response based on one or more

predictor variables. Logistic regression is popular in various fields, including finance, healthcare, marketing, and social sciences. The algorithm uses a logistic function to model the relationship between the independent and dependent variables. The logistic function produces an S-shaped curve representing a binary response's probability. Logistic regression is a simple yet powerful approach that is easy to implement and interpret. It is widely used in machine learning applications due to its high accuracy and simplicity.

Machine learning using multiple regression is an effective approach for analyzing and predicting complex relationships between multiple independent variables and a dependent variable. By leveraging the power of multiple regression algorithms, we can uncover valuable insights and make accurate predictions. This technique allows us to model the impact of various factors on the target variable, enabling us to understand the underlying patterns and dynamics in the data. Machine learning models built on multiple regression can be used in a wide range of applications, such as predicting sales, forecasting stock prices, or analyzing customer behavior. Overall, machine learning using multiple regression empowers us to harness the power of data and make data-driven decisions.

Logistic regression is similar to linear regression, which was discussed in the previous section. Linear regression is used for prediction, whereas Logistic regression is used for solving classification problems. Logistic regression predicts the output of a categorical dependent variable in terms of probability between 0 and 1. Logistic regression assumes that the dependent variable must be categorical in nature, and independent variables should not have multicollinearity. Logistic regression is a machine learning algorithm that uses the supervised learning technique. It is used to predict the categorical dependent variable using a given set of independent variables. Logistic regression has the ability to provide probabilities and classify new data using continuous and discrete datasets. Logistic regression can be used to classify the observations using data and can easily determine the most relevant variables used for the classification.

Some of the steps in solving machine learning algorithms are:

1. Identify the significant variables in the model.
2. Split the dataset into training and test sets.
3. Fitting Logistic Regression to the Training Set
4. Predicting the test result
5. Test accuracy of the result (Creation of Confusion matrix)
6. Visualizing the test set result.

Performance Measures in Classification Problems

Accuracy, precision, recall, and F1 score are commonly used metrics to evaluate the performance of classification models. But before discussing these metrics, we will need to understand the confusion matrix. A confusion matrix is a powerful tool in machine learning, illustrating the performance of a classification model. It tabulates true positive, true negative, false positive, and false negative results, providing insights into the model's accuracy, precision, recall, and F1 score.

1. True Positive means the model predicted positive, and it is true. This means in example 2, the model predicted that a person has a house, and person has.
2. True Negative means the model predicted negative, and it is true. This means that in example 2, the model predicted that the person does not have a house and that he does not have one.
3. False Positive means model predicted positive, and it is false. In the context of example 2, the model predicted that a person has a house but does not.
4. False Negative means model predicted negative, and it is false. In the context of example 2, the model predicted person does not have a house but actually has one.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

5. The total number of correct predictions is the summation of True Positive and True Negative.
6. Accuracy measures the proportion of correctly classified instances from all instances in the dataset. It is calculated as the ratio of correct predictions to the total number of predictions made.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$$

7. Precision measures the proportion of true positive predictions among all positive predictions made by the model. It reflects the model's ability to avoid false positives.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

8. Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions among all actual positive instances in the dataset. It reflects the model's ability to identify all relevant instances.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

9. The F1 score is the harmonic mean of precision and recall. It balances precision and recall, giving equal weight to both metrics. It is particularly useful when there is an imbalance between classes in the dataset.

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Table 3: Python Functions used in Example 2

Steps	Description and Python Functions
Identify the significant variables in the model.	logit = sm.logit('OwnHouse~CarPrice+Incometax+SECClass_A2+SECClass_B1+SECClass_B2+SECClass_C+Region_North+Region_South+Region_West', data=ex2new).fit() two variables are found significant Incometax and SECClass_B2'
Split the dataset into training and test set.	from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 42)
Fitting Logistic Regression to the Training set	modelf = sm.Logit(y_train,X_train).fit() print(modelf.summary())
Predicting the test result	Testdf data frame is created which has actual Y values in test dataset and predicted probability of test X values using coefficients of train data set. testdf = pd.DataFrame({ "actual": y_test, "predicted_prob": modelf.predict(X_test) }) predicted column is added in testdf, if probability is more than

	0.5 value in predicted column is 1 else 0. <pre>testdf['predicted'] = testdf.predicted_prob.map(lambda x: 1 if x > 0.5 else 0)</pre>
Test accuracy of the result (Creation of Confusion matrix)	<pre>cm=metrics.confusion_matrix(testdf.actual, testdf.predicted) cm</pre> <pre>array([[128, 25], [31, 116]], dtype=int64)</pre>
Extracting true positive, true negative, false positive, and false negative results from confusion matrix	<pre>cf=metrics.confusion_matrix(testdf.actual, testdf.predicted).ravel() tn, fp, fn, tp = cf print('True Positive', tp) print('True Negative', tn) print('False Positive', fp) print('False Negative', fn)</pre>
Visualizing the test set result in matrix.	We can find the accuracy of the predicted result by interpreting the confusion matrix. By above output, we can interpret that 128+116= 244 (Correct Output) and 25+31= 56 (Incorrect Output).
Calculating accuracy	<pre>metrics.accuracy_score(testdf.actual, testdf.predicted)</pre>
Calculating precision	<pre>metrics.precision_score(testdf.actual, testdf.predicted)</pre>
Calculating recall	<pre>recall= tp/(tp+fn) print(recall)</pre>
accuracy measures	Classification_report() method in sklearn.metrics gives a detailed report of precision recall and F1 Score. Actual and predicted column of testdfdataframeare used. <pre>print(metrics.classification_report(testdf.actual, testdf.predicted))</pre>

Example 2: Data file ARex2.csv has data of 1000 families, data is of their Socioeconomic class, region, income tax paid, price of latest car purchased by them and whether they own a house or not. It is assumed that Socio economic class, region, income tax and car

price paid impacts whether they own a house or not. Use Python to answer following questions, mention all the steps and codes in word file.

- Build a logistics regression model to predict the probability of a family owning the house. Determine the significant dependent variables using logistics regression at the 0.05 significance level. State logistics regression equation with car price as dependent variable.
- Split the data into training and testing set. Use 70% for training and 30% for testing. Fit the logistics regression model on the training set. Predict the values in the training set based on the coefficients in developed in the training data set. Build the confusion matrix based on the prediction in the test data set. Calculate True Positive, True Negative, False Positive and False Negative.
- Calculate accuracy measures such as Sensitivity, Recall, Precision, and F1 score.

Hints and Interpretation:

For part a, the regression model is
 $\text{OwnHouse} \sim \text{CarPrice} + \text{Incometax} + \text{SECClass_A2} + \text{SECClass_B1} + \text{SECClass_B2} + \text{SECClass_C} + \text{Region_North} + \text{Region_South} + \text{Region_West}$

In the above model, only income tax and SECClass_B2 are found to be significant since P-value is found to be lower than 0.05 significance level. The above can be seen from Python input and output line 6.

For part b and c, both the training and test set are using Python line of code 10. The model is built using the Python line of code 11. The model is tested on the test set in Python line of code 14. The confusion matrix is created using the prediction output of test set which can be seen in Python line of code 19. In the test data frame there are 300 families; when we predict in the test data set using the parameters of the model fitted on the train data set, the findings are:

S No	Performance Measures	Explanation
1	True Positive is 116.	This means the logit model predicted that a person has a house and a hose in the test data set.
2	True Negative is 128	This means logit model predicted that person does not have a house and he actually does not have a house.
3	False Positive is 25	This means model predicted person has a house but in actual he does not have.
4	False Negative is 31	This means model predicted person does not have a house but in actual he has a house.

5	accuracy_score 0.8133	High accuracy indicates that the model is making correct predictions overall.
6	precision_score 0.8227	Precision focuses on the quality of the positive predictions.High precision indicates that when the model predicts a positive result, it is usually correct.
7	recall 0.789	Recall focuses on the model's ability to capture all positive instances.High recall indicates that the model is able to find most of the positive instances.
8	F1 Score 0.82	The F1 score provides a balance between precision and recall.It's useful when you want to consider both false positives and false negatives.F1 score reaches its best value at 1 and worst at 0.

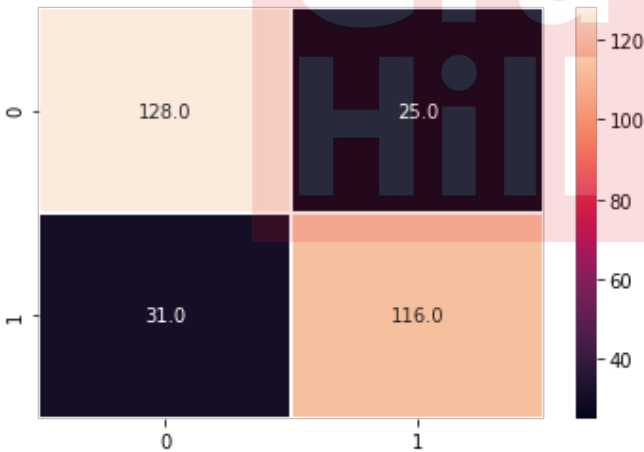
Table 4: Python Code of Example 2

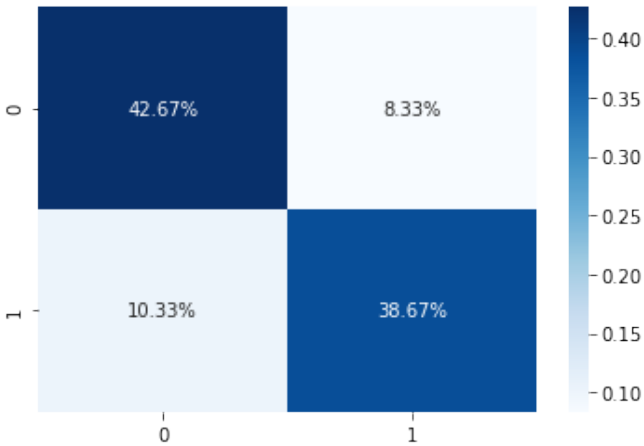
In [1]	<pre>import pandas as pd import numpy as np import seaborn as sns import statsmodels.api as sm1 import statsmodels.formula.api as sm import statsmodels.stats.anova as cx from scipy import stats import math from sklearn import metrics</pre>					
In [2]	<pre>ex2=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/BusStatUP/Datafiles/ARex2.csv') ex2.columns = reg6.columns.str.replace(' ', '') ex2.head()</pre>					
Out [2]	FamilyNo	SECClass	CarPrice	Incometax	Region	OwnHouse
	0	1	A1	1480990	1184792.00	West 1
	1	2	A1	1437384	1149907.20	South 1
	2	3	A1	1484581	1128281.56	South 1
	3	4	A2	1426385	1126844.15	East 1
	4	5	A2	1477912	1108434.00	South 1
In [3]	<pre>variables = ex2.columns variables</pre>					
In [4]	<pre>categorical_variables = ['SECClass', 'Region'] ex2new = pd.get_dummies(ex2[variables], columns = categorical_variables, drop_first = True) ex2new.columns</pre>					

Out [5]	Index(['FamilyNo', 'CarPrice', 'Incometax', 'OwnHouse', 'SECClass_A2', 'SECClass_B1', 'SECClass_B2', 'SECClass_C', 'Region_North', 'Region_South', 'Region_West'], dtype='object')	
In [6]	logi = sm.logit('OwnHouse~CarPrice+Incometax+SECClass_A2+ SECClass_B1+ SECClass_B2+SECClass_C+Region_North+Region_South+ Region_West', data=ex2new).fit() print(logi.summary())	
Out [6]	Optimization terminated successfully. Current function value: 0.508993 Iterations 7 Logit Regression Results =====	Dep. Variable: OwnHouse No. Observations: 1000 Model: Logit Df Residuals: 990 Method: MLE Df Model: 9 Date: Sun, 12 Jun 2022 Pseudo R-squ.: 0.2655 Time: 11:33:01 Log-Likelihood: -508.99 converged: True LL-Null: -692.99 Covariance Type: nonrobust LLR p-value: 9.176e-74 =====
	coef std err z P> z [0.025 0.975]	
	Intercept -3.9880 1.177 -3.387 0.001 -6.296 -1.680	
	CarPrice -2.128e-07 1.16e-06 -0.184 0.854 -2.48e-06 2.06e-06	
	Incometax 7.461e-06 2.13e-06 3.507 0.000 3.29e-06 1.16e-05	
	SECClass_A2 0.5174 0.390 1.327 0.185 -0.247 1.282	
	SECClass_B1 -0.6531 0.532 -1.229 0.219 -1.695 0.389	
	SECClass_B2 1.0089 0.394 2.558 0.011 0.236 1.782	
	SECClass_C 0.4602 0.702 0.656 0.512 -0.915 1.835	
	Region_North -0.3506 0.221 -1.585 0.113 -0.784 0.083	
	Region_South -0.3177 0.221 -1.439 0.150 -0.750 0.115	
	Region_West -0.0492 0.218 -0.225 0.822 -0.477 0.378	
	=====	
In [7]	## fitting Logit Model on the significant variables logi = sm.logit('OwnHouse~Incometax+SECClass_B2',data=ex2new).fit() print(logi.summary())	
Out	Optimization terminated successfully. Current function value: 0.529215	

[7]	<pre> Iterations 6 Logit Regression Results ===== ===== Dep. Variable: OwnHouse No. Observations: 1000 Model: Logit Df Residuals: 997 Method: MLE Df Model: 2 Date: Sun, 12 Jun 2022 Pseudo R-squ.: 0.2363 Time: 11:33:02 Log-Likelihood: -529.21 converged: True LL-Null: -692.99 Covariance Type: nonrobust LLR p-value: 7.505e-72 ===== ===== coef std err z P> z [0.025 0.975] ----- Intercept -4.0680 0.293 -13.875 0.000 -4.643 -3.493 Incometax 7.166e-06 5.21e-07 13.756 0.000 6.14e-06 8.19e-06 SECClass_B2 0.8833 0.303 2.913 0.004 0.289 1.478 </pre>
In	<code>X=reg6new[['Incometax','SECClass_B2']]</code>
[8]	<code>Y=reg6new.OwnHouse</code>
In	<code>import statsmodels.api as sm</code>
[9]	<code>X = sm.add_constant(X)</code>
In	<code>from sklearn.model_selection import train_test_split</code>
[10]	<code>X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 42)</code>
In	<code>modelf = sm.Logit(y_train,X_train).fit()</code>
[11]	<code>print(modelf.summary())</code>
Out	<pre> Optimization terminated successfully. Current function value: 0.536176 Iterations 6 Logit Regression Results ===== ===== Dep. Variable: OwnHouse No. Observations: 700 Model: Logit Df Residuals: 697 Method: MLE Df Model: 2 Date: Sun, 12 Jun 2022 Pseudo R-squ.: 0.2258 Time: 11:33:02 Log-Likelihood: -375.32 converged: True LL-Null: -484.79 </pre>

	Covariance Type: nonrobust LLR p-value: 2.874e-48 ===== ===== coef std err z P> z [0.025 0.975] ----- const -3.9606 0.347 -11.409 0.000 -4.641 -3.280 Incometax 7.078e-06 6.19e-07 11.425 0.000 5.86e-06 8.29e-06 SECClass_B2 0.8417 0.389 2.166 0.030 0.080 1.603 ===== =====																								
In [12]	y_pred_traindf = pd.DataFrame({ "actual": y_train, "predicted_prob": modelf.predict(X_train) }) y_pred_traindf.head(5)																								
Out [13]	<table><tr><th></th><th>actual</th><th>predicted_prob</th></tr><tr><td>541</td><td>0</td><td>0.439594</td></tr><tr><td>440</td><td>0</td><td>0.527724</td></tr><tr><td>482</td><td>1</td><td>0.486123</td></tr><tr><td>422</td><td>1</td><td>0.538930</td></tr><tr><td>778</td><td>0</td><td>0.268602</td></tr></table>		actual	predicted_prob	541	0	0.439594	440	0	0.527724	482	1	0.486123	422	1	0.538930	778	0	0.268602						
	actual	predicted_prob																							
541	0	0.439594																							
440	0	0.527724																							
482	1	0.486123																							
422	1	0.538930																							
778	0	0.268602																							
In [13]	testdf = pd.DataFrame({ "actual": y_test, "predicted_prob": modelf.predict(X_test) }) testdf.head(5)																								
Out [13]	<table><tr><th></th><th>actual</th><th>predicted_prob</th></tr><tr><td>521</td><td>1</td><td>0.454848</td></tr><tr><td>737</td><td>0</td><td>0.303029</td></tr><tr><td>740</td><td>0</td><td>0.302173</td></tr><tr><td>660</td><td>0</td><td>0.353688</td></tr><tr><td>411</td><td>1</td><td>0.543494</td></tr></table>		actual	predicted_prob	521	1	0.454848	737	0	0.303029	740	0	0.302173	660	0	0.353688	411	1	0.543494						
	actual	predicted_prob																							
521	1	0.454848																							
737	0	0.303029																							
740	0	0.302173																							
660	0	0.353688																							
411	1	0.543494																							
In [14]	testdf['predicted'] = testdf.predicted_prob.map(lambda x: 1 if x > 0.5 else 0) testdf.head()																								
Out [14]	<table><tr><th></th><th>actual</th><th>predicted_prob</th><th>predicted</th></tr><tr><td>521</td><td>1</td><td>0.454848</td><td>0</td></tr><tr><td>737</td><td>0</td><td>0.303029</td><td>0</td></tr><tr><td>740</td><td>0</td><td>0.302173</td><td>0</td></tr><tr><td>660</td><td>0</td><td>0.353688</td><td>0</td></tr><tr><td>411</td><td>1</td><td>0.543494</td><td>1</td></tr></table>		actual	predicted_prob	predicted	521	1	0.454848	0	737	0	0.303029	0	740	0	0.302173	0	660	0	0.353688	0	411	1	0.543494	1
	actual	predicted_prob	predicted																						
521	1	0.454848	0																						
737	0	0.303029	0																						
740	0	0.302173	0																						
660	0	0.353688	0																						
411	1	0.543494	1																						
In [15]	cm=metrics.confusion_matrix(testdf.actual, testdf.predicted) cm																								

Out [15]	<pre>array([[128, 25], [31, 116]], dtype=int64)</pre>
In [16]	<pre>ps=metrics.precision_score(testdf.actual, testdf.predicted) r= tp/(tp+fn) as1=metrics.accuracy_score(testdf.actual, testdf.predicted) print('accuracy_score', as1) print('precision_score', ps) print('recall', r)</pre>
Out [16]	<pre>accuracy_score 0.8133333333333334 precision_score 0.8226950354609929 recall 0.7891156462585034</pre>
	<pre>ps=metrics.precision_score(testdf.actual, testdf.predicted) print('precision_score', ps)</pre>
	<pre>precision_score 0.8226950354609929</pre>
In [17]	<pre>sns.heatmap(cm, annot=True,linewidths=.5, fmt= '.1f')</pre>
Out [17]	 Confusion matrix heatmap showing counts for classes 0 and 1. The matrix is a 2x2 grid. The top-left cell (0,0) is light orange with value 128.0. The top-right cell (0,1) is dark purple with value 25.0. The bottom-left cell (1,0) is dark purple with value 31.0. The bottom-right cell (1,1) is light orange with value 116.0. A color bar on the right indicates values from 40 to 120.
In [18]	<pre>sns.heatmap(cm/np.sum(cm), annot=True, fmt='.2%', cmap='Blues')</pre>

Out [18]	
In [19]	<pre>cf=metrics.confusion_matrix(testdf.actual, testdf.predicted).ravel() tn, fp, fn, tp = cf print('True Positive', tp) print('True Negative', tn) print('False Positive', fp) print('False Negative', fn)</pre>
Out [19]	<pre>True Positive 116 True Negative 128 False Positive 25 False Negative 31</pre>
In [20]	<pre>print(metrics.classification_report(testdf.actual, testdf.predicted))</pre>
Out [20]	<pre>precision recall f1-score support 0 0.81 0.84 0.82 153 1 0.82 0.79 0.81 147 accuracy 0.81 300 macro avg 0.81 0.81 0.81 300 weighted avg 0.81 0.81 0.81 300</pre>

Summary

Machine learning involves creating algorithms that learn from data to make predictions. Multiple regression extends simple linear regression to predict a dependent variable using multiple independent variables, providing a comprehensive analysis of their relationships. Logistic regression is used for binary classification, predicting the probability of an outcome belonging to one of two categories. It outputs a probability score that can be thresholded for classification. Performance measures for classification include accuracy (proportion of correct predictions), precision (true positives among predicted positives),

recall (true positives among actual positives), F1 score, and ROC-AUC. These metrics help in evaluating the effectiveness of a classification model and comparing different models.

