Customer Spend Prediction - Using Multiple Linear Regression

by by A4Ayub Data Science Labs (http://www.a4ayub.me/ (http://www.a4ayub.me/))

Class Problem Statement

Build a model to predict customer spend based on day of week, shop hour and quantity of items purchased

Please take note that the illustrations in this notebook is NOT for results/accuracy but for explaining the various concepts

Data Description

This data is proprietory and cannot be shared to anyone who is NOT attending A4Ayub Data Science Labs.!

Each row in the dataset corresponds to one unique product in a basket (e.g. if there are three occurences of the same product in that basket, it will have one row for the product in that basket, with quantity equal to three)

The file has the below structure:

Column Name	Description	Type	Sample Values
shop_week	Identifies the week of the basket	Char	Format is YYYYWW where the first 4 characters identify the fiscal year and the other two characters identify the specific week within the year (e.g. 200735). Being the fiscal year, the first week doesn't start in January. (See time.csv file for start/end dates of each week)
shop_date	Date when shopping has been made. Date is specified in the yyyymmdd format	Char	20060413, 20060412
shop_weekday	Identifies the day of the week	Num	1=Sunday, 2=Monday, …, 7=Saturday
shop_hour	Hour slot of the shopping	Num	0=00:00-00:59, 1=01:00-01:59, …23=23:00-23:59
Quantity	Number of items of the same product bought in this basket	Num	Integer number
spend	Spend associated to the items bought	Num	Number with two decimal digits
prod_code	Product Code	Char	PRD0900001, PRD0900003
prod_code_10	Product Hierarchy Level 10 Code	Char	CL00072, CL00144
prod_code_20	Product Hierarchy Level 20 Code	Char	DEP00021, DEP00051
prod_code_30	Product Hierarchy Level 30 Code	Char	G00007, G00015
prod_code_40	Product Hierarchy Level 40 Code	Char	D00002, D00003

Column Name	Description	Type	Sample Values
cust_code	Customer Code	Char	CUST0000001624, CUST0000001912
cust_price_sensitivity	Customer's Price Sensitivity	Char	LA=Less Affluent, MM=Mid Market, UM=Up Market, XX=unclassified
cust_lifestage	Customer's Lifestage	Char	YA=Young Adults, OA=Older Adults, YF=Young Families, OF=Older Families, PE=Pensioners, OT=Other, XX=unclassified
basket_id	Basket ID. All items in a basket share the same basket_id value.	Num	994100100000020, 994100100000344
basket_size	Basket size	Char	L=Large, M=Medium, S=Small
basket_price_sensitivity	Basket price sensitivity	Char	LA=Less Affluent, MM=Mid Market, UM=Up Market, XX=unclassified
basket_type	Basket type	Char	Small Shop, Top Up, Full Shop, XX
basket_dominant_mission	Shopping dominant mission	Char	Fresh, Grocery, Mixed, Non Food, XX
store_code	Store Code	Char	STORE00001, STORE00002
store_format	Format of the Store	Char	LS, MS, SS, XLS
store_region	Region the store belongs to	Char	E02, W01, E01, N03

Workbench

Importing the required libraries

In [39]:

```
# Import the numpy and pandas package
import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# Import the warnings
import warnings

# Import statsmodels
import statsmodels.formula.api as smf

# Import RMSE
from statsmodels.tools.eval_measures import rmse

# Import Linear Regression from scikit-learn
from sklearn.linear_model import LinearRegression

# Import Polynomial Features
from sklearn.preprocessing import PolynomialFeatures

# Import Metrics
from sklearn.metrics import mean_squared_error, r2_score

# configuration settings
%matplotlib inline
sns.set(color_codes=True)
warnings.filterwarnings('ignore') ## Surpress the warnings
```

Load the data into a dataframe

In [40]:

```
# load the data into a dataframe called supermarket_till_transactions_df
supermarket_till_transactions_df = pd.read_csv("../data/beginner/supermarket_till_transacti
```

In [41]:

```
# view the top five records
supermarket_till_transactions_df.head(5)
```

Out[41]:

	SHOP_WEEK	SHOP_DATE	SHOP_WEEKDAY	SHOP_HOUR	QUANTITY	SPEND	PROD_COD
0	200607	20060413	5	20	1	103	PRD090009
1	200607	20060412	4	19	1	28	PRD090035
2	200607	20060413	5	20	3	84	PRD090055
3	200607	20060412	4	19	1	221	PRD090164
4	200607	20060413	5	20	1	334	PRD090206

5 rows × 22 columns

In order to illustrate Multiple Linear Regression we just need two variables which are:

1. SHOP_WEEKDAY
2. SHOP_HOUR
3. QUANTITY
4. SPEND

In [42]:

```
supermarket_till_transactions_df = supermarket_till_transactions_df[["SHOP_WEEKDAY","SHOP_H
supermarket_till_transactions_df.head(5)
```

Out[42]:

	SHOP_WEEKDAY	SHOP_HOUR	QUANTITY	SPEND
0	5	20	1	103
1	4	19	1	28
2	5	20	3	84
3	4	19	1	221
4	5	20	1	334

Using statsmodel

Simple linear regression can easily be extended to include multiple features. This is called multiple linear regression:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

Each x represents a different feature, and each feature has its own coefficient. In this case:

$$y = \beta_0 + \beta_1 \times SHOPWEEKDAY + \beta_2 \times SHOPHOUR + \beta_3 \times QUANTITY$$

Let's use Statsmodels to estimate these coefficients:

In [43]:

```
# Initialise and fit linear regression model using `statsmodels`
stats_model = smf.ols('SPEND ~ SHOP_WEEKDAY + SHOP_HOUR + QUANTITY', data=supermarket_till_
stats_model = stats_model.fit()
```

We no longer have to calculate alpha and beta ourselves as this method does it automatically for us! Calling model.params will show us the model's parameters:

In [44]:

```
stats_model.params
```

Out[44]:

```
Intercept       366.801757
SHOP_WEEKDAY    -18.779187
SHOP_HOUR       -12.306012
QUANTITY         53.434573
dtype: float64
```

From the results above:

1. β0 = 366.8018 - This is the y intercept when x is zero
2. β1 = -18.779187 - This is the regression coefficient that measures a unit change in SPEND when SHOP_WEEKDAY changes
3. β2 = -12.306012 - This is the regression coefficient that measures a unit change in SPEND when SHOP_HOUR changes
4. β3 = 53.434573 - This is the regression coefficient that measures a unit change in SPEND when QUANTITY changes

In [45]:

```
# print a summary of the fitted model
stats_model.summary()
```

Out[45]:

OLS Regression Results

Dep. Variable:	SPEND	**R-squared:**	0.120
Model:	OLS	**Adj. R-squared:**	0.097
Method:	Least Squares	**F-statistic:**	5.240
Date:	Tue, 07 Jan 2020	**Prob (F-statistic):**	0.00200
Time:	07:54:34	**Log-Likelihood:**	-817.20
No. Observations:	119	**AIC:**	1642.
Df Residuals:	115	**BIC:**	1654.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	366.8018	93.598	3.919	0.000	181.402	552.202
SHOP_WEEKDAY	-18.7792	18.221	-1.031	0.305	-54.872	17.313
SHOP_HOUR	-12.3060	5.712	-2.154	0.033	-23.621	-0.991
QUANTITY	53.4346	17.801	3.002	0.003	18.175	88.695

Omnibus:	119.230	**Durbin-Watson:**	2.114
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1503.081
Skew:	3.549	**Prob(JB):**	0.00
Kurtosis:	18.898	**Cond. No.**	64.1

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

R Squared

The Coefficient of determination, R-Squared – This is used to measure how much of the variation in the outcome can be explained by the variation in the independent variables. R-Squared always increases as more predictors are added to the MLR model even though the predictors may not be related to the outcome variable.

R2 by itself can't thus be used to identify which predictors should be included in a model and which should be excluded. R2 can only be between 0 and 1, where 0 indicates that the outcome cannot be predicted by any of the independent variables and 1 indicates that the outcome can be predicted without error from the independent variables.

In [46]:

```
# print the R-squared value for the model
stats_model.rsquared
```

Out[46]:

```
0.12024717016691944
```

This means that 12.025% of the SPEND can be explained by SHOP_WEEKDAY, SHOP_HOUR and QUANTITY

Adjusted R-Squared

When we add more predictor variables into the equation, R-Squared will always increase making R-Squared not accurate as the number of predictor variables increases.

Adjusted R-Squared, accounts for the increase of the predictor variables.

Because of the nature of the equation, the adjusted R-Squared should always be lower or equal to the R-Squared

In [47]:

```
# print the Adjusted R-squared value for the model
stats_model.rsquared_adj
```

Out[47]:

```
0.0972970963451868
```

Perform the prediction

In [48]:

```
new_shop_weekday = 1
new_shop_hour = 18
new_quantity = 2
stats_model_ypred = stats_model.predict({"SHOP_WEEKDAY": new_shop_weekday,"SHOP_HOUR": new_
```

RMSE

The root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample and population values) predicted by a model and the values actually observed

The smaller the value the better

In [49]:

```
# calc rmse
stats_model_rmse = rmse(y, stats_model_ypred)
stats_model_rmse
```

Out[49]:

```
249.5712033427086
```

Confidence in the model

A confidence interval gives an estimated range of values which is likely to include an unknown population parameter, the estimated range being calculated from a given set of sample data.

A confidence interval is how much uncertainty there is with any particular statistic. Confidence intervals are often used with a margin of error. It tells you how confident you can be that the results reflect what you would expect to find if it were possible to study the entire population.

In [50]:

```
# print the confidence intervals for the model coefficients
stats_model.conf_int()
```

Out[50]:

	0	1
Intercept	181.401596	552.201919
SHOP_WEEKDAY	-54.871733	17.313360
SHOP_HOUR	-23.621078	-0.990946
QUANTITY	18.174604	88.694541

Hypothesis Testing and P-Values

p-values tell you how statistically significant the variable is. Removing variables with high p-values can cause your accuracy/R squared to increase, and even the p-values of the other variables to increase as well — and that's a good sign.

In [51]:

```
# print the p-values for the model coefficients
stats_model.pvalues
```

Out[51]:

```
Intercept       0.000152
SHOP_WEEKDAY    0.304878
SHOP_HOUR       0.033304
QUANTITY        0.003293
dtype: float64
```

Using scikit-learn

In [52]:

```
# Build linear regression model using SHOP_WEEKDAY,SHOP_HOUR and QUANTITY as predictors
# Split data into predictors X and output Y
predictors = ['SHOP_WEEKDAY', 'SHOP_HOUR', 'QUANTITY']
X = supermarket_till_transactions_df[predictors]
y = supermarket_till_transactions_df['SPEND']

# Initialise and fit model
lm = LinearRegression()
scikit_model = lm.fit(X, y)
```

In [53]:

```
print(f'alpha = {scikit_model.intercept_}')
print(f'betas = {scikit_model.coef_}')
```

```
alpha = 366.80175720345426
betas = [-18.77918675 -12.30601205  53.43457262]
```

Therefore, our model can be written as:

SPEND = 366.802 + (-18.779*SHOP_WEEKDAY)* + *(-12.306*SHOP_HOUR) + (53.435*QUANTITY)

We can predict values by simply using .predict():

In [54]:

```
new_X = [[1, 18, 2]] # Sunday 6pm buying 2 items
scikit_learn_ypred = scikit_model.predict(new_X)
```

Calculate the RMSE when scikit learn is used

In [55]:

```
# calc rmse
scikit_learn_rmse = rmse(y, scikit_learn_ypred)
scikit_learn_rmse
```

Out[55]:

```
249.5712033427086
```

This means that anyone buying 2 items at 6pm on sunday is most likely to spend 233.38 Shillings.