Customer Spend Prediction - Using Simple Linear Regression

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

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 [41]:

```
# 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

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

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

Load the data into a dataframe

In [2]:

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

In [3]:

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

Out[3]:

	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 Simple Linear Regression we just need two variables which are:

1. SHOP_WEEKDAY
2. SPEND

In [7]:

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

Out[7]:

	SHOP_WEEKDAY	SPEND
0	5	103
1	4	28
2	5	84
3	4	221
4	5	334

Using Ordinary Least Squares Method (OLS)

There are two kinds of variables in a alinear regression model:

1. The input or predictor variable commonly refered to as **X**
2. The output is the variable that we want to predict commonly refered to as **Y**

$$\boxed{Y_e = \alpha + \beta X}$$

where Y_e is the estimated or predicted value of Y based on our linear equation.

The objective of the Ordinary Least Square Method is to find the values of α and β in the **y =** β**x +** α linear regression equation that minimise the sum of the squared difference between Y and Y_e.

$$\beta = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$\alpha = \bar{Y} - \beta * \bar{X}$$

where $\bar{X}$ is the mean of X values and $\bar{Y}$ is the mean of Y values.

β as simply Cov(X, Y) / Var(X)

If we are able to determine the optimum values of these two parameters, then we will have the line of best fit that we can use to predict the values of Y, given the value of X.

In [8]:

```
X = supermarket_till_transactions_df["SHOP_WEEKDAY"]
y = supermarket_till_transactions_df["SPEND"]

# calculate the mean of X and y
xmean = np.mean(X)
ymean = np.mean(y)

# Calculate the terms needed for the numator and denominator of beta
supermarket_till_transactions_df['xycov'] = (supermarket_till_transactions_df['SHOP_WEEKDAY
supermarket_till_transactions_df['xvar'] = (supermarket_till_transactions_df['SHOP_WEEKDAY'

# Calculate beta and alpha
beta = supermarket_till_transactions_df['xycov'].sum() / supermarket_till_transactions_df['
alpha = ymean - (beta * xmean)
```

In [9]:

```
# View the alpha and beta values
print(f'alpha = {alpha}')
print(f'beta = {beta}')
```

```
alpha = 330.09788218544224
beta = -30.045025805303435
```

Great, we now have an estimate for alpha and beta! Our model can be written as Y_e = 330.098 + -30.045 X, and we can start making predictions:

In [10]:

```
ypred = alpha + beta * X
```

In [11]:

```
# View the predictions
ypred
```

Out[11]:

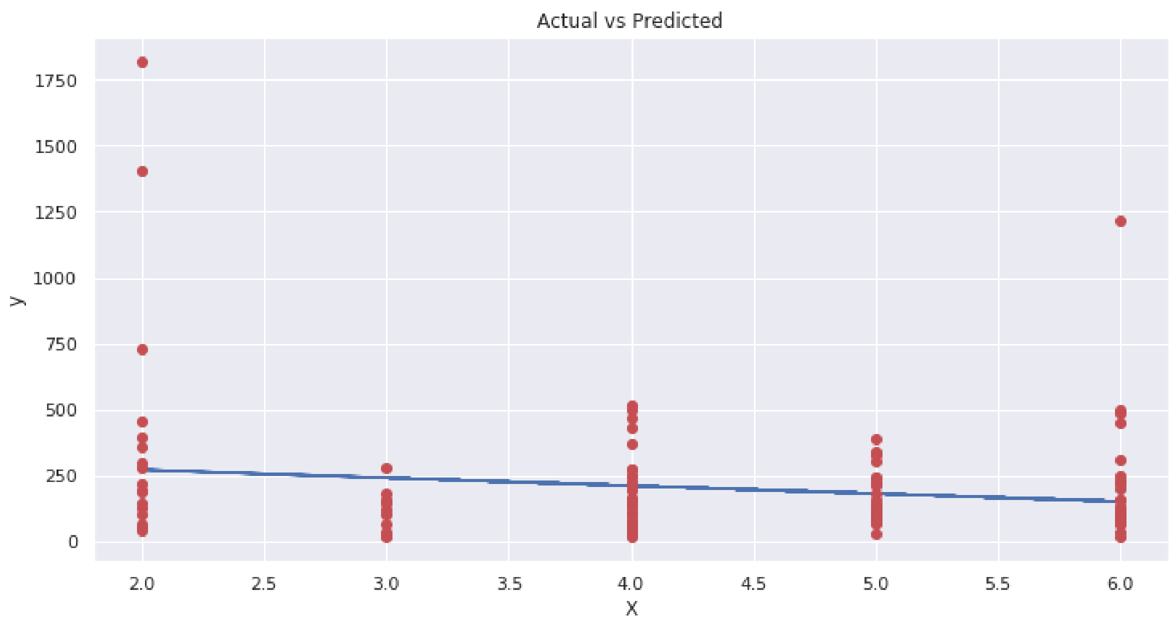
```
0      179.872753
1      209.917779
2      179.872753
3      209.917779
4      179.872753
          ...
114    270.007831
115    149.827727
116    149.827727
117    149.827727
118    149.827727
Name: SHOP_WEEKDAY, Length: 119, dtype: float64
```

Let’s plot our prediction ypred against the actual values of y, to get a better visual understanding of our model.

In [12]:

```
# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(X, ypred)     # regression line
plt.plot(X, y, 'ro')   # scatter plot showing actual data
plt.title('Actual vs Predicted')
plt.xlabel('X')
plt.ylabel('y')

plt.show()
```



The blue line is our line of best fit i.e. Y_e = 330.098 + -30.045 X

We can see from this graph that there is a negative linear relationship between X and y. Using our model, we can predict y from any values of X!

For example, if we had a value X = 7, we can predict that: (According to the data description 7 represents Saturday)

Y_e = 330.098 + -30.045 (7) = **119.783**

According to this it means that customer spend reduces from Monday to Saturday

Using statsmodels

In [18]:

```
# Initialise and fit linear regression model using `statsmodels`
stats_model = smf.ols('SPEND ~ SHOP_WEEKDAY', data=supermarket_till_transactions_df)
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 [36]:

```
stats_model.params
```

Out[36]:

```
Intercept       330.097882
SHOP_WEEKDAY    -30.045026
dtype: float64
```

From the results above:

1. β0 = 330.097882 - This is the y intercept when x is zero
2. β1 = -30.045026 - This is the regression coefficient that measures a unit change in SPEND when SHOP_WEEKDAY changes

The negative value on the regression co-efficient for SHOP_WEEKDAY means that SHOP_WEEKDAY has a negative impact to the SPEND.

In [35]:

```
stats_model.summary()
```

Out[35]:

OLS Regression Results

Dep. Variable:	SPEND	**R-squared:**	0.023
Model:	OLS	**Adj. R-squared:**	0.015
Method:	Least Squares	**F-statistic:**	2.797
Date:	Mon, 06 Jan 2020	**Prob (F-statistic):**	0.0971
Time:	07:28:52	**Log-Likelihood:**	-823.42
No. Observations:	119	**AIC:**	1651.
Df Residuals:	117	**BIC:**	1656.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	330.0979	79.239	4.166	0.000	173.169	487.027
SHOP_WEEKDAY	-30.0450	17.965	-1.672	0.097	-65.625	5.535

Omnibus:	125.423	**Durbin-Watson:**	2.042
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1847.385
Skew:	3.748	**Prob(JB):**	0.00
Kurtosis:	20.787	**Cond. No.**	16.2

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 [38]:

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

Out[38]:

```
0.023346566175179162
```

This means that 2.335% of the SPEND can be explained by SHOP_WEEKDAY

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 [40]:

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

Out[40]:

```
0.014999100928881318
```

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 [28]:

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

Out[28]:

	0	1
Intercept	173.168683	487.027081
SHOP_WEEKDAY	-65.624681	5.534630

Hypothesis Testing and P-Values

In [39]:

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

Out[39]:

```
Intercept       0.000060
SHOP_WEEKDAY    0.097122
dtype: float64
```

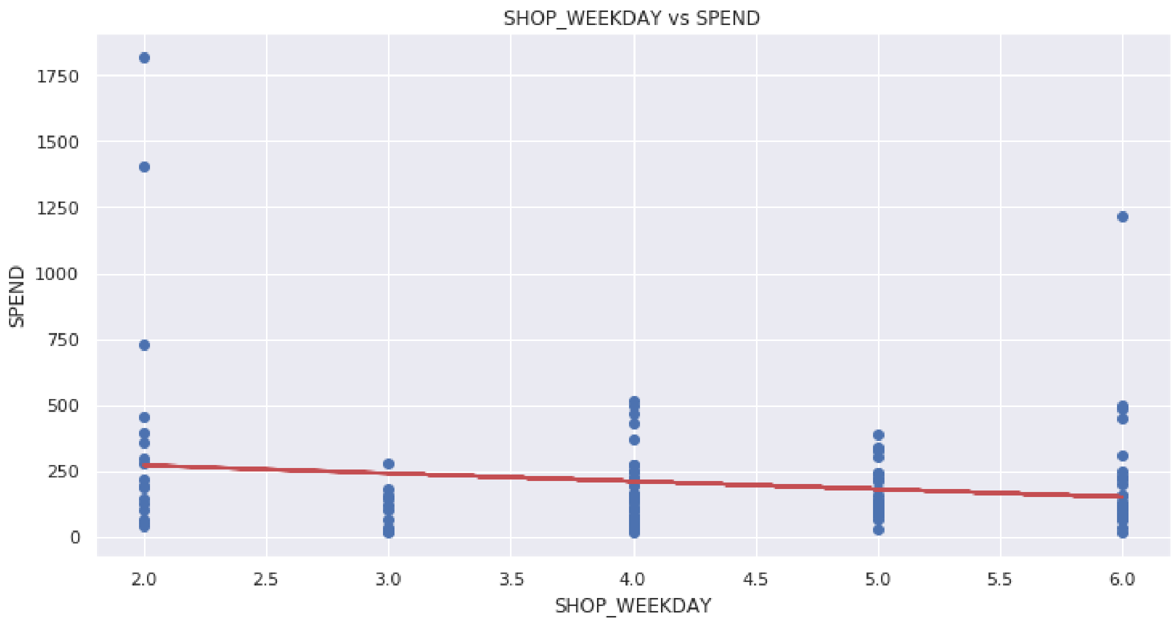
Now that we've fit a simple regression model, we can try to predict the values of spend based on the equation we just derived using the .predict method.

In [20]:

```
# Predict values
spend_pred = stats_model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(supermarket_till_transactions_df['SHOP_WEEKDAY'], supermarket_till_transactions_df
plt.plot(supermarket_till_transactions_df['SHOP_WEEKDAY'], spend_pred, 'r', linewidth=2)
plt.xlabel('SHOP_WEEKDAY')
plt.ylabel('SPEND')
plt.title('SHOP_WEEKDAY vs SPEND')

plt.show()
```



With this model, we can predict spend from given day of the week. For example, if we want to predict the spend for sunday, we can predict that spend will increase to 300.05 shillings:

In [43]:

```
new_X = 1
ypred = stats_model.predict({"SHOP_WEEKDAY": new_X})
```

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 [42]:

```
# calc rmse
rmse = rmse(y, ypred)
rmse
```

Out[42]:

```
244.81829310949846
```