Linear Regression

Peer-graded Assignment

==

Project Description

When you’re scrolling through your favorite e-commerce site or app, you might find you spend more time on the site than you originally expected. E-commerce giants know the longer that you’re on the site or the app, the more likely you are to spend more and more on their products. But how can they maximize their profits by targeting the right customer? The answer lies in the data.

Imagine you’ve been hired as a data science contractor by an e-commerce company, who is trying to decide whether to focus their efforts on their mobile app experience or their website. To provide the company with expert advice, you can run a linear regression model to provide you with a data-driven answer to the problem of which customers to target.

==

Import Libraries

In [1]:
```
import numpy as np
from numpy import count_nonzero, median, mean
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random



import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.formula.api import ols
# Import variance_inflation_factor from statsmodels
from statsmodels.stats.outliers_influence import variance_inflation_factor
# Import Tukey's HSD function
from statsmodels.stats.multicomp import pairwise_tukeyhsd

import datetime
from datetime import datetime, timedelta, date

# import shap
# import eli5
```

```
# from IPython.display import display

#import os
#import zipfile
import scipy
from scipy import stats
from scipy.stats.mstats import normaltest # D'Agostino K^2 Test
from scipy.stats import boxcox
from collections import Counter

import sklearn
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, OneHo

from sklearn.model_selection import cross_val_score, train_test_split, cross_valida
from sklearn.model_selection import KFold, cross_val_predict, RandomizedSearchCV, S

from sklearn.metrics import accuracy_score, auc, classification_report, confusion_m
from sklearn.metrics import plot_confusion_matrix, plot_roc_curve, precision_score,
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.feature_selection import f_regression, f_classif, chi2, RFE, RFECV
from sklearn.feature_selection import mutual_info_regression, mutual_info_classif
from sklearn.feature_selection import VarianceThreshold, GenericUnivariateSelect
from sklearn.feature_selection import SelectFromModel, SelectKBest, SelectPercentil

from sklearn.inspection import permutation_importance

from sklearn.linear_model import ElasticNet, Lasso, LinearRegression, LogisticRegre


import feature_engine

from feature_engine.selection import (DropConstantFeatures, DropDuplicateFeatures,
                                      DropCorrelatedFeatures, SmartCorrelatedSelect
from feature_engine.selection import SelectBySingleFeaturePerformance, SelectByShuf
from feature_engine.selection import RecursiveFeatureAddition

%matplotlib inline
#sets the default autosave frequency in seconds
%autosave 60
sns.set_style('dark')
sns.set(font_scale=1.2)

plt.rc('axes', titlesize=9)
plt.rc('axes', labelsize=14)
plt.rc('xtick', labelsize=12)
plt.rc('ytick', labelsize=12)

import warnings
warnings.filterwarnings('ignore')

# This module lets us save our models once we fit them.
# import pickle

pd.set_option('display.max_columns',None)
#pd.set_option('display.max_rows', None)
```

```
pd.set_option('display.width', 1000)
pd.set_option('display.float_format','{:.2f}'.format)

random.seed(0)
np.random.seed(0)
np.set_printoptions(suppress=True)
```

```
Autosaving every 60 seconds
```

===

Quick Data Glance

In [2]:
```
df = pd.read_csv("ecommercecust.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	email	address	avatar	avgsessionlength
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.5C
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.93
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.0C
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.31
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.33

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   email                500 non-null    object 
 1   address              500 non-null    object 
 2   avatar               500 non-null    object 
 3   avgsessionlength     500 non-null    float64
 4   timeonapp            500 non-null    float64
 5   timeonwebsite        500 non-null    float64
 6   lengthofmembership   500 non-null    float64
 7   yearlyamountspent    500 non-null    float64
dtypes: float64(5), object(3)
memory usage: 31.4+ KB
```

In [5]:
```
df.dtypes.value_counts()
```

Out[5]:
```
float64    5
object     3
dtype: int64
```

In [6]:
```
# Descriptive Statistical Analysis
df.describe(include="all")
```

Out[6]:

	email	address	avatar	avgsessionlength	tim
count	500	500	500	500.00	
unique	500	500	138	NaN	
top	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	SlateBlue	NaN	
freq	1	1	7	NaN	
mean	NaN	NaN	NaN	33.05	
std	NaN	NaN	NaN	0.99	
min	NaN	NaN	NaN	29.53	
25%	NaN	NaN	NaN	32.34	
50%	NaN	NaN	NaN	33.08	
75%	NaN	NaN	NaN	33.71	
max	NaN	NaN	NaN	36.14	

In [7]:
```
# Descriptive Statistical Analysis
df.describe(include=["int", "float"])
```

Out[7]:

	avgsessionlength	timeonapp	timeonwebsite	lengthofmembership	yearlyamountsp
count	500.00	500.00	500.00	500.00	500
mean	33.05	12.05	37.06	3.53	499
std	0.99	0.99	1.01	1.00	79
min	29.53	8.51	33.91	0.27	256
25%	32.34	11.39	36.35	2.93	445
50%	33.08	11.98	37.07	3.53	498
75%	33.71	12.75	37.72	4.13	549
max	36.14	15.13	40.01	6.92	765

In [8]:
```
# Descriptive Statistical Analysis
df.describe(include="object")
```

Out[8]:

	email	address	avatar
count	500	500	500
unique	500	500	138
top	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	SlateBlue
freq	1	1	7

In [9]:
```
df.shape
```

Out[9]:
```
(500, 8)
```

In [10]:
```
df.columns
```

Out[10]:
```
Index(['email', 'address', 'avatar', 'avgsessionlength', 'timeonapp', 'timeonwebsi
te', 'lengthofmembership', 'yearlyamountspent'], dtype='object')
```

```
==================================================================
```

In [11]:
```
df.drop(['email', 'address', 'avatar'], axis=1, inplace=True)
```

In [12]:
```
df.head()
```

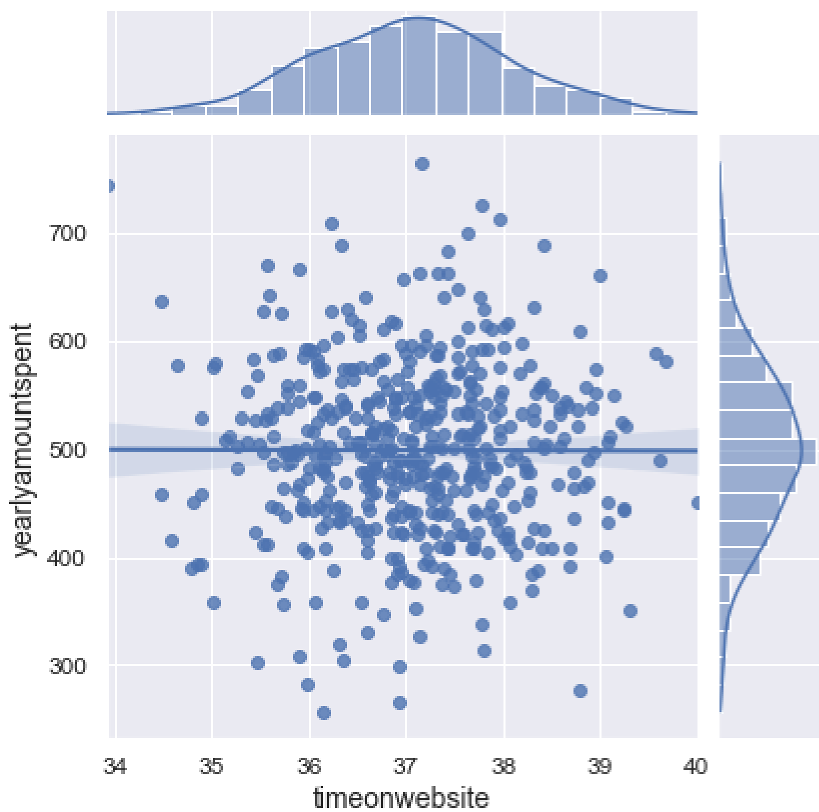
Out[12]:

	avgsessionlength	timeonapp	timeonwebsite	lengthofmembership	yearlyamountspent
0	34.50	12.66	39.58	4.08	587.95
1	31.93	11.11	37.27	2.66	392.20
2	33.00	11.33	37.11	4.10	487.55
3	34.31	13.72	36.72	3.12	581.85
4	33.33	12.80	37.54	4.45	599.41

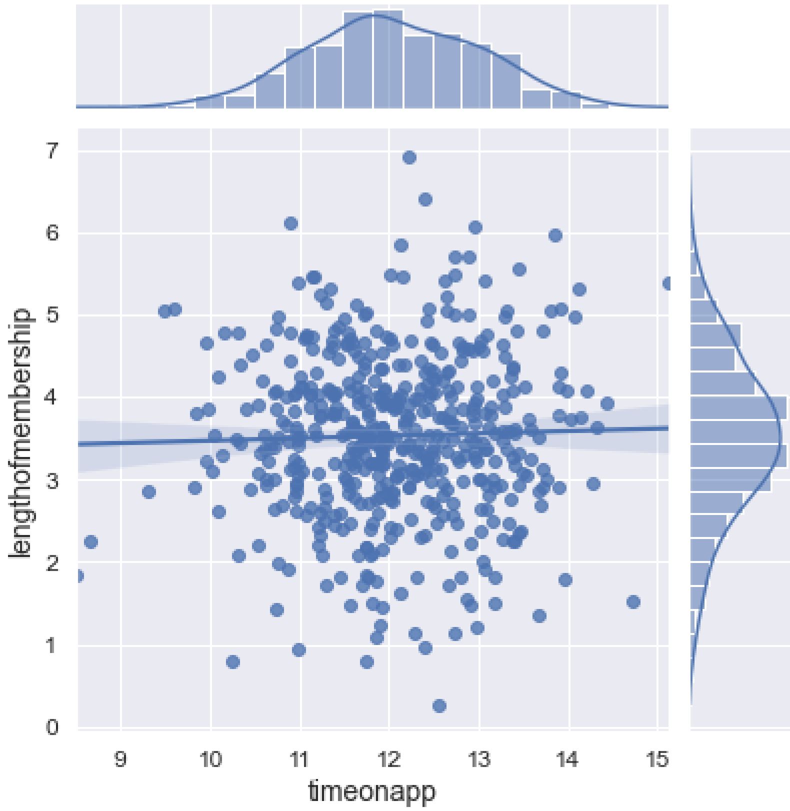
==

Data Visualization

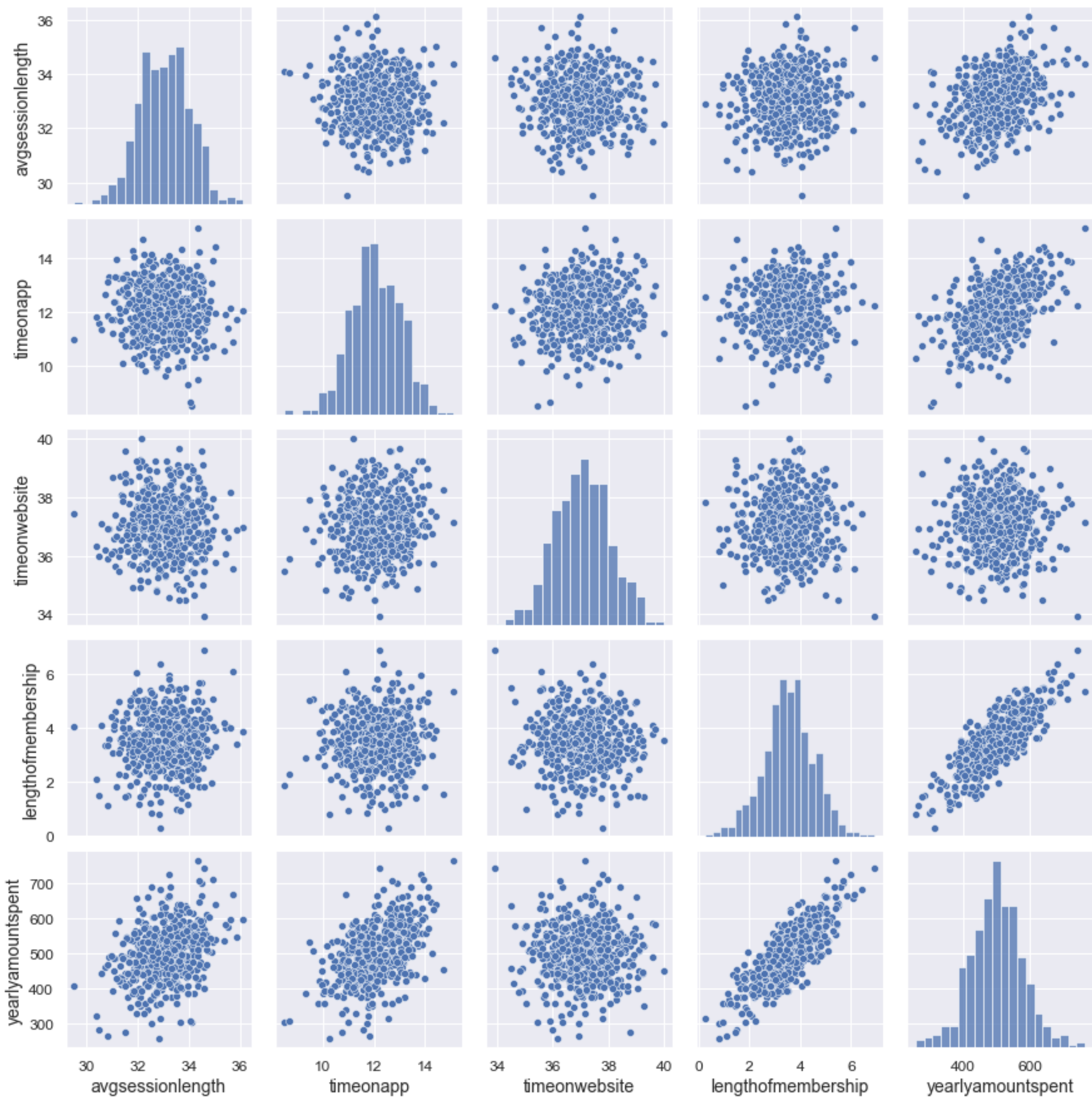
In [13]:
```
sns.jointplot(x=df.timeonwebsite, y=df.yearlyamountspent, kind='reg')
plt.show()
```



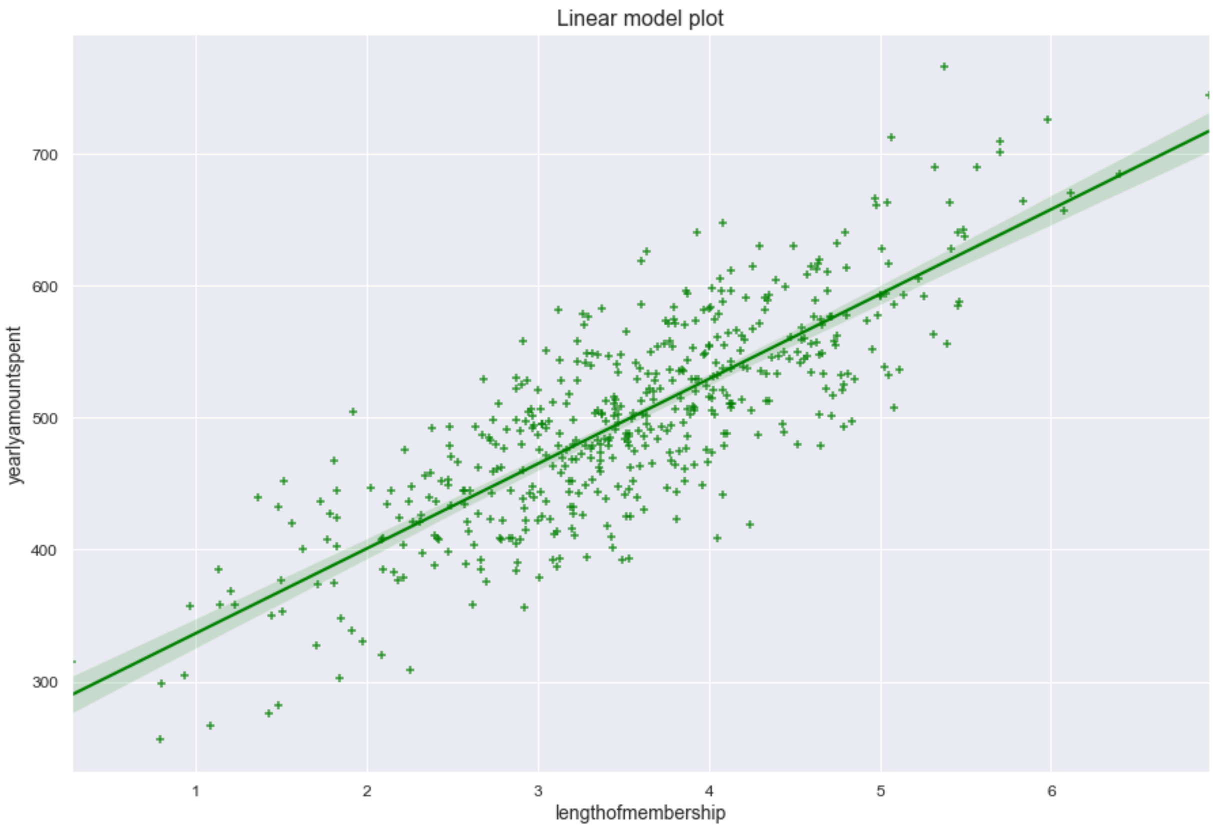
In [14]:
```
sns.jointplot(x=df.timeonapp, y=df.lengthofmembership, kind='reg')
plt.show()
```



In [15]:
```
sns.pairplot(df)
plt.show()
```



In [16]:
```
plt.figure(figsize=(15, 10))
sns.regplot(x='lengthofmembership', y='yearlyamountspent', data=df, color='green',
plt.title("Linear model plot", size=16)
plt.show()
```



==

Linear Regression

Let's first understand what exactly `Regression` means it is a statistical method used in finance, investing, and other disciplines that attempts to determine the `strength` and `character` of the relationship between one `dependent variable` (usually denoted by `Y`) and a series of other variables known as `independent variables`.
`Linear Regression` is a statistical technique where based on a set of `independent variable(s)` a dependent variable is `predicted`.

$$y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$$

y = dependent variable
β_0 = population of intercept
β_i = population of co-efficient
x = independent variable
ε_i = Random error

==

Multiple Linear Regression

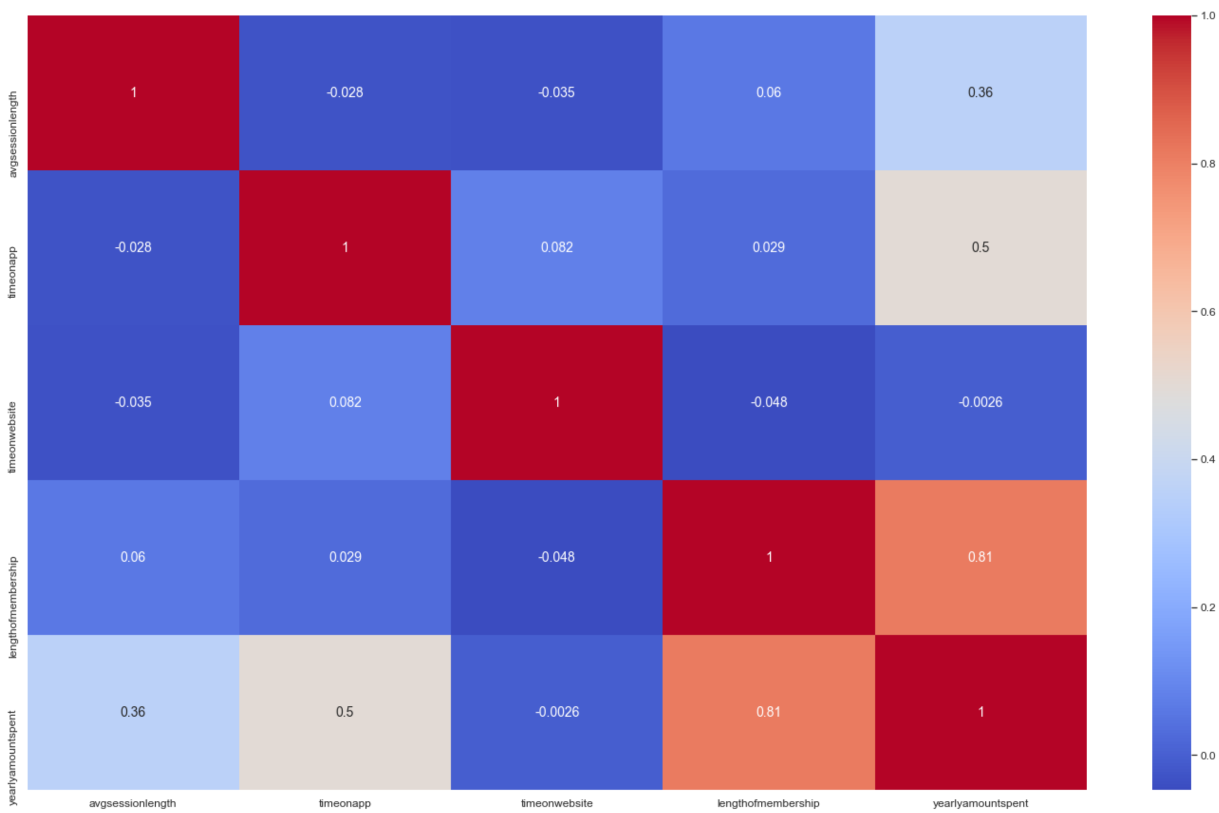
It(as the name suggests) is characterized by `multiple independent variables` (more than `1`). While you discover the simplest `fit line`, you'll be able to adjust a `polynomial or regression` toward the `mean`. And these are called `polynomial or regression` toward the `mean`.

In [17]:
```
cormat = df.corr()
cormat
```

Out[17]:

	avgsessionlength	timeonapp	timeonwebsite	lengthofmembership
avgsessionlength	1.00	-0.03	-0.03	0.06
timeonapp	-0.03	1.00	0.08	0.03
timeonwebsite	-0.03	0.08	1.00	-0.05
lengthofmembership	0.06	0.03	-0.05	1.00
yearlyamountspent	0.36	0.50	-0.00	0.81

In [18]:
```
plt.subplots(figsize=(25,15))
sns.heatmap(cormat, annot=True, annot_kws= {'size':14 }, cmap="coolwarm")
plt.show()
```



==

Multiple Linear Regression (Scikit Learn)

What if we want to predict car price using more than one variable?

If we want to use more variables in our model to predict car price, we can use **Multiple Linear Regression**. Multiple Linear Regression is very similar to Simple Linear Regression, but this method is used to explain the relationship between one continuous response (dependent) variable and **two or more** predictor (independent) variables. Most of the real-world regression models involve multiple predictors. We will illustrate the structure by using four predictor variables, but these results can generalize to any integer:

$$Y: Response\ Variable$$
$$X\_1: Predictor\ Variable\ 1$$
$$X\_2: Predictor\ Variable\ 2$$
$$X\_3: Predictor\ Variable\ 3$$
$$X\_4: Predictor\ Variable\ 4$$

$$a: intercept$$
$$b\_1: coefficients\ of\ Variable\ 1$$
$$b\_2: coefficients\ of\ Variable\ 2$$
$$b\_3: coefficients\ of\ Variable\ 3$$
$$b\_4: coefficients\ of\ Variable\ 4$$

The equation is given by:

$$Yhat = a + b\_1 X\_1 + b\_2 X\_2 + b\_3 X\_3 + b\_4 X\_4$$

Train Test Split

We've prepared our data and we're ready to model. There's one last step before we can begin. We must split the data into features and target variable, and into training data and test data. We do this using the `train_test_split()` function. We'll put 25% of the data into our test set, and use the remaining 75% to train the model.

Notice below that we include the argument `stratify=y`. If our master data has a class split of 80/20, stratifying ensures that this proportion is maintained in both the training and test data. `=y` tells the function that it should use the class ratio found in the `y` variable (our target).

The less data you have overall, and the greater your class imbalance, the more important it is to stratify when you split the data. If we didn't stratify, then the function would split the data randomly, and we could get an unlucky split that doesn't get any of the minority class in the test data, which means we wouldn't be able to effectively evaluate our model. Worst of all, we might not even realize what went wrong without doing some detective work.

Lastly, we set a random seed so we and others can reproduce our work.

In [19]:
```
df.shape
```
Out[19]:
```
(500, 5)
```
In [20]:
```
df.head()
```
Out[20]:

	avgsessionlength	timeonapp	timeonwebsite	lengthofmembership	yearlyamountspent
0	34.50	12.66	39.58	4.08	587.95
1	31.93	11.11	37.27	2.66	392.20
2	33.00	11.33	37.11	4.10	487.55
3	34.31	13.72	36.72	3.12	581.85
4	33.33	12.80	37.54	4.45	599.41

In [21]:
```
X = df.iloc[:,0:4]
y = df.iloc[:,4]
```
In [22]:
```
X.values, y.values
```

```
Out[22]: (array([[34.49726773, 12.65565115, 39.57766802,  4.08262063],
                [31.92627203, 11.10946073, 37.26895887,  2.66403418],
                [33.00091476, 11.33027806, 37.11059744,  4.1045432 ],
                ...,
                [32.64677668, 11.49940906, 38.33257633,  4.95826447],
                [33.32250105, 12.39142299, 36.84008573,  2.33648467],
                [33.71598092, 12.41880832, 35.77101619,  2.73515957]]),
          array([587.95105397, 392.20493344, 487.54750487, 581.85234404,
                599.40609205, 637.10244792, 521.57217476, 549.90414611,
                570.20040896, 427.1993849 , 492.60601272, 522.33740461,
                408.64035107, 573.41586733, 470.4527333 , 461.7807422 ,
                457.84769594, 407.70454755, 452.31567548, 605.0610388 ,
                534.70574381, 419.93877484, 436.51560573, 519.34098913,
                700.91709162, 423.17999168, 619.89563986, 486.83893477,
                529.53766534, 554.72208383, 497.5866713 , 447.68790654,
                588.71260551, 491.07322368, 507.44183234, 521.88357317,
                347.77692663, 490.73863214, 478.17033405, 537.84619527,
                532.75178758, 501.87443028, 591.19717818, 547.24434342,
                448.22982919, 549.86059046, 593.91500297, 563.67287336,
                479.73194909, 416.35835358, 725.58481406, 442.66725174,
                384.62657157, 451.45744687, 522.40414126, 483.67330802,
                520.89879445, 453.16950235, 496.65070807, 547.36514059,
                616.85152297, 507.212569  , 613.59932337, 483.15972078,
                540.26340041, 765.51846194, 553.60153468, 469.3108615 ,
                408.62018783, 451.57568516, 444.96655165, 595.8228367 ,
                418.1500811 , 534.7771881 , 578.24160506, 478.71935687,
                444.2859075 , 544.77986372, 488.78606109, 475.75906779,
                489.812488  , 462.89763615, 596.43017262, 338.31986264,
                533.51493526, 536.77189936, 487.37930602, 473.72896651,
                547.12593175, 505.11334354, 449.07031944, 611.0000251 ,
                515.82881485, 439.07476674, 514.08895775, 543.34016626,
                521.14295181, 614.71533383, 507.39006179, 495.29944255,
                518.06455798, 390.10327297, 420.73767324, 492.10505239,
                410.06961106, 497.51368333, 494.55186109, 378.33090691,
                570.45172591, 549.00822693, 459.28512346, 492.94505307,
                424.76263551, 422.42677588, 642.10157873, 413.37178311,
                479.23109291, 593.07724134, 506.54730705, 571.30749488,
                576.31117737, 576.8025474 , 514.23952072, 495.17595045,
                514.33655827, 541.22658399, 516.83155668, 468.44573723,
                548.2803202 , 431.61773376, 552.94034545, 573.30622226,
                452.627255  , 542.7115581 , 407.80403064, 482.35357032,
                529.23009012, 433.0487691 , 476.19141335, 439.99787994,
                448.93329321, 472.99224667, 463.92351299, 350.05820016,
                460.06127739, 505.77114032, 463.4849954 , 479.73193765,
                424.18549429, 465.88931271, 426.77521599, 684.16343102,
                555.89259539, 657.01992394, 595.80381888, 503.97837905,
                586.15587018, 744.2218671 , 512.82535813, 528.22380937,
                468.91350132, 357.59143942, 536.42310453, 490.20659998,
                550.04758058, 513.45057119, 497.81193001, 578.98625858,
                506.53639314, 501.74923331, 421.96679419, 439.89128048,
                666.12559173, 298.76200786, 465.17662331, 373.8857237 ,
                532.71748568, 554.90078302, 537.77316254, 501.10024523,
                517.16513559, 557.52927361, 493.71919298, 452.12262509,
                577.27345498, 485.92313052, 425.74509203, 537.2150527 ,
                524.63796461, 478.88539132, 612.3852299 , 476.76672415,
                505.11963753, 545.94549214, 434.02169975, 424.67528101,
```

```
352.55010816, 662.96108781, 560.56016062, 467.50190043,
504.87043239, 590.56271965, 443.96562681, 392.49739919,
568.71757593, 712.39632681, 413.29599918, 562.08204539,
412.0129313 , 468.6684656 , 496.55408164, 548.51852928,
536.13089686, 558.42725718, 357.86371864, 529.0566632 ,
387.35707274, 528.93361857, 420.91615953, 496.93344626,
519.3729768 , 591.43773557, 502.4097853 , 604.33484007,
555.06839405, 256.67058229, 547.11098236, 461.92087689,
458.37691065, 436.28349815, 532.93521884, 512.55253436,
630.42276323, 463.74598112, 493.18021625, 501.20917268,
501.92826487, 376.33690076, 421.32663126, 538.77493348,
398.16346853, 571.47103412, 451.62861054, 490.6004425 ,
591.78108943, 409.07047205, 563.44603567, 647.61945573,
448.34042501, 518.78648309, 523.63393514, 393.85737099,
426.15454771, 503.38788729, 482.60246733, 524.79762757,
574.65484337, 574.74721966, 660.42518429, 375.39845541,
640.18774001, 514.00981785, 376.49684072, 484.51980911,
614.72963763, 567.47501053, 554.00309343, 399.9838716 ,
479.17285149, 585.9318443 , 540.99573911, 628.04780393,
582.49192373, 640.7861664 , 446.41867337, 570.63009809,
423.3083341 , 616.66028602, 530.36246889, 442.36311738,
511.97985999, 560.44379217, 475.26342373, 374.26967454,
463.59141803, 471.60288439, 626.01867266, 432.47206125,
356.61556789, 467.4278485 , 503.21739312, 378.47356645,
584.21831349, 451.72786332, 557.634109  , 432.72071784,
506.42385997, 510.15981728, 587.57479948, 282.47124572,
473.94985742, 489.9080531 , 541.97220376, 266.08634095,
494.68715581, 689.78760417, 387.53471631, 441.89663152,
604.84131882, 302.18954781, 479.61481167, 506.13234244,
319.9288698 , 528.30922503, 610.12803313, 584.10588505,
466.42119882, 404.82452887, 564.79096901, 596.51669797,
368.65478495, 542.41247673, 478.2621264 , 473.36049557,
559.19904795, 447.18764431, 505.23006828, 557.25268675,
422.36873661, 445.06218551, 442.06441376, 533.04006018,
424.2028271 , 498.63559849, 330.59444603, 443.44186006,
478.60091594, 440.00274755, 357.78311075, 476.13924687,
501.1224915 , 592.6884532 , 486.0834255 , 576.02524413,
442.72289157, 461.79095906, 488.38752578, 593.15640148,
392.81034498, 443.19722103, 535.48077519, 533.39655379,
532.12744911, 558.94811239, 508.77190674, 403.7669021 ,
640.5840619 , 461.62827839, 382.41610787, 561.87465767,
444.57614413, 401.03313522, 384.32605714, 527.78299576,
482.14499688, 594.27448342, 502.0925279 , 407.65717876,
708.93518487, 531.96155055, 521.24078024, 447.3690272 ,
385.15233799, 430.58888256, 418.6027421 , 478.95140476,
483.79652206, 538.94197453, 486.16379907, 385.09500707,
527.78378976, 547.19074935, 410.60294395, 583.97780197,
474.53232944, 414.93506065, 550.81336773, 458.78113168,
407.54216801, 581.30893288, 546.55666686, 503.17508519,
549.13157329, 482.83098586, 557.60826205, 484.87696494,
669.9871405 , 547.70998858, 537.8252823 , 408.21690177,
663.07481761, 506.37586675, 528.4193297 , 632.12358814,
488.27029797, 508.73574095, 411.18696357, 409.09452619,
467.80092437, 512.16586639, 608.27181662, 589.02648976,
444.05382657, 493.18126139, 532.72480546, 275.91842065,
511.03878605, 438.41774201, 475.72506791, 483.54319387,
```

```
       663.80369328, 544.40927216, 630.15672817, 461.11224843,
       491.9115051 , 574.41568961, 530.76671865, 581.79879768,
       556.29814117, 502.13278923, 556.18636887, 475.0716299 ,
       486.94705384, 434.14420203, 304.13559158, 571.21600483,
       583.07963566, 445.74984124, 392.99225591, 565.9943634 ,
       499.14015245, 510.5394217 , 308.52774656, 561.51653198,
       423.47053317, 513.15311185, 529.19451886, 314.4385183 ,
       478.58428601, 444.58216498, 475.01540709, 436.72055586,
       521.19531053, 478.18305971, 432.48116856, 438.30370785,
       388.94054879, 534.77148495, 537.91575292, 407.87637822,
       618.84597042, 502.77107457, 397.4205841 , 392.28524425,
       689.23569976, 543.13262629, 577.73602485, 436.58074035,
       553.99467359, 427.3565308 , 424.72877393, 541.04983096,
       469.38314617, 444.54554965, 492.5568337 , 535.32161009,
       408.95833594, 487.55545806, 487.64623174, 402.16712222,
       551.0230017 , 497.38955776, 494.63860976, 479.24741678,
       462.65651893, 515.50247966, 576.47760717, 357.85798361,
       597.73987888, 327.37795259, 510.40138845, 510.50147847,
       403.81951983, 627.60331871, 510.66179222, 573.84743772,
       529.04900413, 551.62014548, 456.46951007, 497.77864222]))
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [24]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[24]:
```
((400, 4), (100, 4), (400,), (100,))
```

Linear Regression Model

In [25]:
```
lr = LinearRegression()
```

In [26]:
```
lr.fit(X_train,y_train)
```

Out[26]:
```
LinearRegression()
```

In [27]:
```
lr_pred = lr.predict(X_test)
lr_pred[0:5]
```

Out[27]:
```
array([438.46488066, 489.6618454 , 370.06954186, 513.8590556 ,
       495.69799868])
```

In [28]:
```
lr.intercept_
```

Out[28]:
```
-1060.5508096198866
```

In [29]:
```
lr.coef_
```

Out[29]:
```
array([25.88815047, 38.87046474,  0.47066154, 61.78369022])
```

Linear Regression Evaluation

In [30]:
```
tablesmlr = pd.DataFrame(data={"Actual": y_test , "Predicted": lr_pred})
```

```
tablesmlr
```

Out[30]:

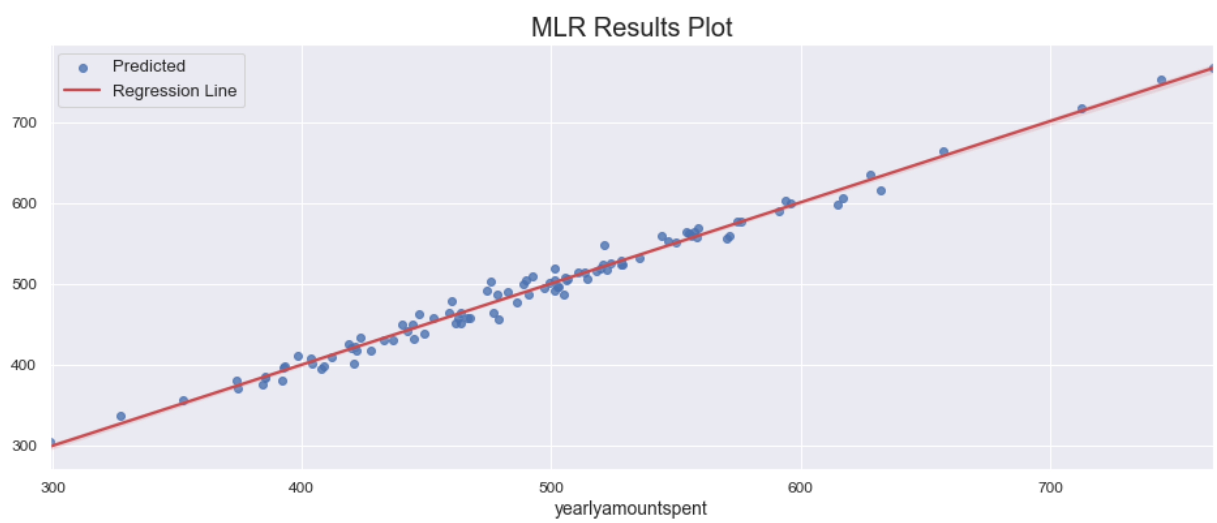
	Actual	Predicted
90	449.07	438.46
254	482.60	489.66
283	374.27	370.07
445	513.15	513.86
461	502.77	495.70
...	...	...
372	385.15	383.74
56	520.90	524.17
440	499.14	500.68
60	616.85	606.79
208	412.01	408.81

100 rows × 2 columns

In [31]:

```
fig, ax = plt.subplots(figsize=(16,6))

sns.regplot(x=y_test, y=lr_pred, data=tablesmlr, line_kws={"color":"r"})
ax.set_title("MLR Results Plot", size=20)
ax.legend(['Predicted','Regression Line'])
plt.show()
```



In [32]:

```
mse = mean_squared_error(y_test,lr_pred)
mse
```

Out[32]: 92.8901030449849

In [33]:

```
rmse = np.sqrt(mse)
rmse
```

Out[33]: 9.637951185028118

In [34]:
```
r2score = r2_score(y_test,lr_pred)
r2score
```

Out[34]: 0.9861924261981548

In [35]:
```
# Get the shape of x, to facilitate the creation of the Adjusted R^2 metric
X.shape
```

Out[35]: (500, 4)

In [36]:
```
# Number of observations is the shape along axis 0
n = X.shape[0]
# Number of features (predictors, p) is the shape along axis 1
p = X.shape[1]
```

In [37]:
```
# Number of observations is the shape along axis 0
n = X_train.shape[0]
# Number of features (predictors, p) is the shape along axis 1
p = X_train.shape[1]
```

In [38]:
```
# We find the Adjusted R-squared using the formula
adjusted_r2 = 1-(1-r2score)*(n-1)/(n-p-1)
adjusted_r2
```

Out[38]: 0.9860526026659843

In [39]:
```
lr.score(X_train, y_train)
```

Out[39]: 0.9837380400055443

In [40]:
```
lr.score(X_test, y_test)
```

Out[40]: 0.9861924261981548

==

Provide a suggestion or advice to the company to focus on either website or app (based on the coefficient of the model) in a comment in your notebook

Mobile app has more influence on amount spent, therefore the company can focus on app development.

==