Data Science II with python (Class notes)

STAT 303-2

Arvind Krishna

2023-01-03

Table of contents

Preface

These are class notes for the course STAT303-2. This is not the course text-book. You are required to read the relevant sections of the book as mentioned on the course website.

The course notes are currently being written, and will continue to being developed as the course progresses (just like the course textbook last quarter). Please report any typos / mistakes / inconsistencies / issues with the class notes / class presentations in your comments here. Thank you!

Part I

Code

1 Simple Linear Regression

Read section 3.1 of the book before using these notes.

Note that in this course, lecture notes are not sufficient, you must read the book for better understanding. Lecture notes are just implementing the concepts of the book on a dataset, but not explaining the concepts elaborately.

1.1 Simple Linear Regression

```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
from matplotlib.lines import Line2D
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Develop a simple linear regression model that predicts car price based on engine size. Datasets to be used: *Car_features_train.csv, Car_prices_train.csv*

```
# We are reading training data ONLY at this point.
# Test data is already separated in another file
trainf = pd.read_csv('./Datasets/Car_features_train.csv') # Predictors
trainp = pd.read_csv('./Datasets/Car_prices_train.csv') # Response
train = pd.merge(trainf,trainp)
train.head()
```

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
0	18473	bmw	6 Series	2020	Semi-Auto	11	Diesel	145	53.3282	3.0	37980

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
1	15064	bmw	6 Series	2019	Semi-Auto	10813	Diesel	145	53.0430	3.0	33980
2	18268	bmw	6 Series	2020	Semi-Auto	6	Diesel	145	53.4379	3.0	36850
3	18480	bmw	6 Series	2017	Semi-Auto	18895	Diesel	145	51.5140	3.0	25998
4	18492	bmw	6 Series	2015	Automatic	62953	Diesel	160	51.4903	3.0	18990

1.1.1 Training with `statsmodels`

Here, we will use the `statsmodels.formula.api` module of the `statsmodels` library. The use of "API" here doesn't refer to a traditional external web API but rather an interface within the library for users to interact with and perform specific tasks. The `statsmodels.formula.api` module provides a formulaic interface to the `statsmodels` library. A formula is a compact way to specify statistical models using a formula language. This module allows users to define statistical models using formulas similar to those used in R.

So, in summary, the `statsmodels.formula.api` module provides a formulaic interface as part of the `statsmodels` library, allowing users to specify statistical models using a convenient and concise formula syntax.

```
# Let's create the model

# ols stands for Ordinary Least Squares - the name of the algorithm that optimizes Linear Reg

# data input needs the dataframe that has the predictor and the response
# formula input needs to:
    # be a string
    # have the following syntax: "response~predictor"

# Using engineSize to predict price
ols_object = smf.ols(formula = 'price~engineSize', data = train)
```

```
#Using the fit() function of the 'ols' class to fit the model, i.e., train the model
model = ols_object.fit()
```

```
#Printing model summary which contains among other things, the model coefficients
model.summary()
```

Dep. Variable:	price	**R-squared:**	0.390
Model:	OLS	**Adj. R-squared:**	0.390
Method:	Least Squares	**F-statistic:**	3177.
Date:	Tue, 16 Jan 2024	**Prob (F-statistic):**	0.00
Time:	16:46:33	**Log-Likelihood:**	-53949.
No. Observations:	4960	**AIC:**	1.079e+05
Df Residuals:	4958	**BIC:**	1.079e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-4122.0357	522.260	-7.893	0.000	-5145.896	-3098.176
engineSize	1.299e+04	230.450	56.361	0.000	1.25e+04	1.34e+04

Omnibus:	1271.986	**Durbin-Watson:**	0.517
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6490.719
Skew:	1.137	**Prob(JB):**	0.00
Kurtosis:	8.122	**Cond. No.**	7.64

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

The model equation is: $\widehat{price}$ = -4122.0357 + 12990 * `engineSize`

- R-squared is 39%. This is the proportion of variance in car price explained by `engineSize`.
- The coef of `engineSize` ($\hat{\beta}_1$) is statistically significant (p-value = 0). There is a linear relationship between X and Y.
- The 95% CI of $\hat{\beta}_1$ is [1.25e+04, 1.34e+04].
- PI is not shown here.

The coefficient of `engineSize` is 1.299e+04. - Unit change in `engineSize` increases the expected price by $ 12,990. - An increase of 3 increases the price by $ (3*1.299e+04) = $ 38,970.

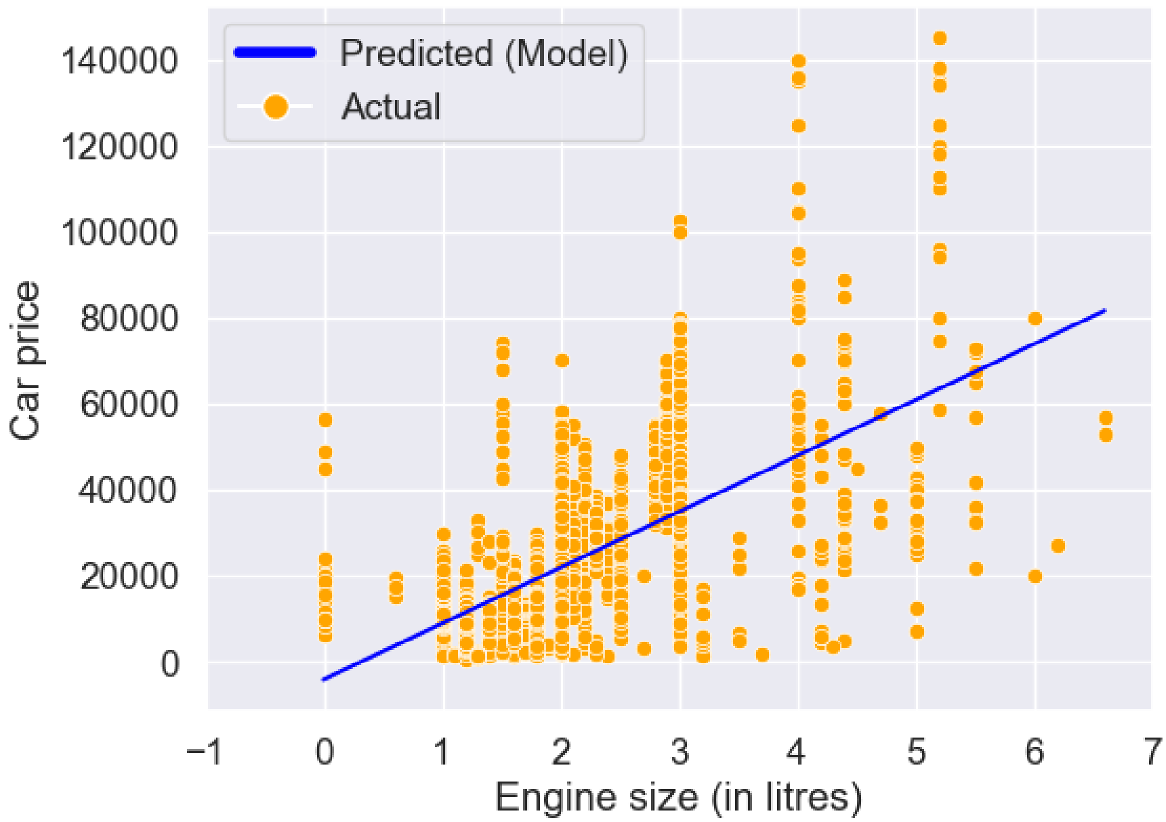
The coefficients can also be returned directly usign the `params` attribute of the `model` object returned by the `fit()` method of the `ols` class:

```
model.params
```

```
Intercept     -4122.035744
engineSize    12988.281021
dtype: float64
```

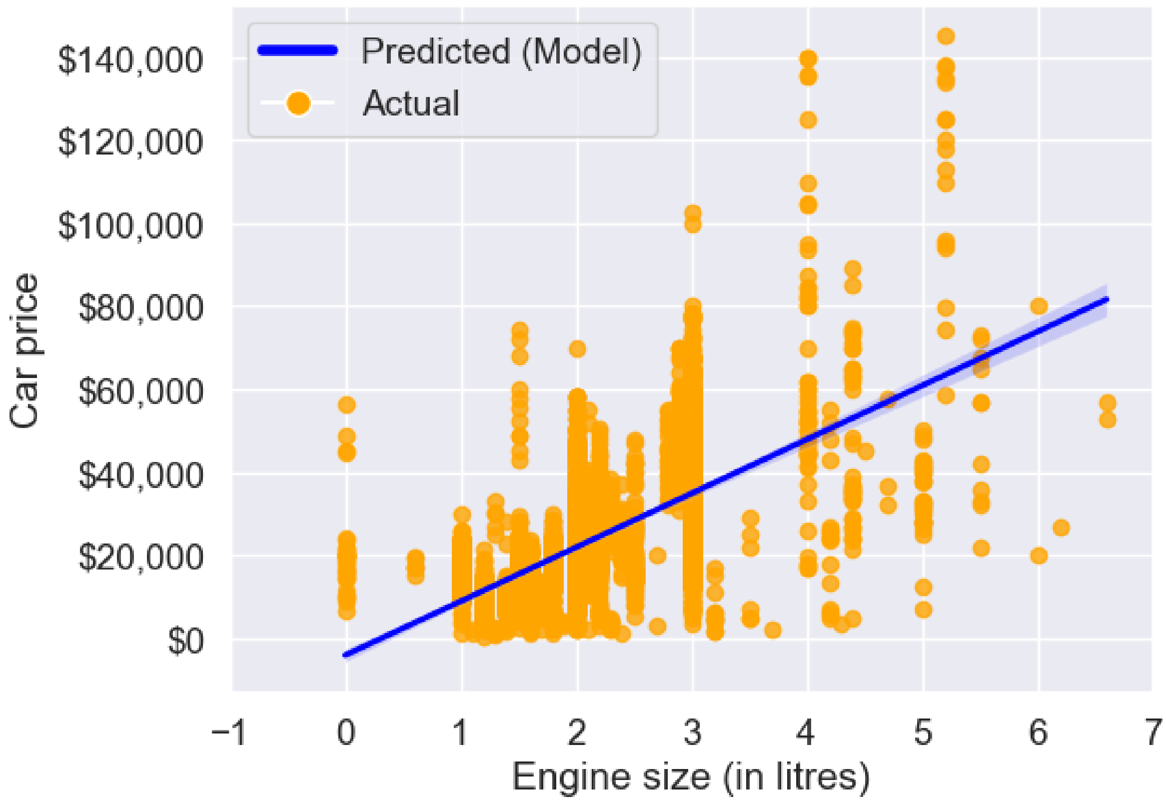
Visualize the regression line

```
sns.set(font_scale=1.25)
ax = sns.scatterplot(x = train.engineSize, y = train.price,color = 'orange')
sns.lineplot(x = train.engineSize, y = model.fittedvalues,color = 'blue')
plt.xlim(-1,7)
plt.xlabel('Engine size (in litres)')
plt.ylabel('Car price')
legend_elements = [Line2D([0], [0], color='blue', lw=4, label='Predicted (Model)'),
                   Line2D([0], [0], marker='o', color='w', label='Actual',
                          markerfacecolor='orange', markersize=10)]
ax.legend(handles=legend_elements, loc='upper left');
```



Note that the above plot can be made directly using the seaborn function `regplot()`. The function `regplot()` fits a simple linear regression model with `y` as the response, and `x` as the predictor, and then plots the model over a scatteplot of the data.

```
ax = sns.regplot(x = 'engineSize', y = 'price', data = train, color = 'orange',line_kws={"co
plt.xlim(-1,7)
plt.xlabel('Engine size (in litres)')
plt.ylabel('Car price')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.legend(handles=legend_elements, loc='upper left');
#Note that some of the engineSize values are 0. They are incorrect, and should ideally be im
```



The light shaded region around the blue line in the above plot is the confidence interval.

Predict the car price for the cars in the test dataset. Datasets to be used: *Car_features_test.csv, Car_prices_test.csv*

Now that the model has been trained, let us evaluate it on unseen data. Make sure that the columns names of the predictors are the same in train and test datasets.

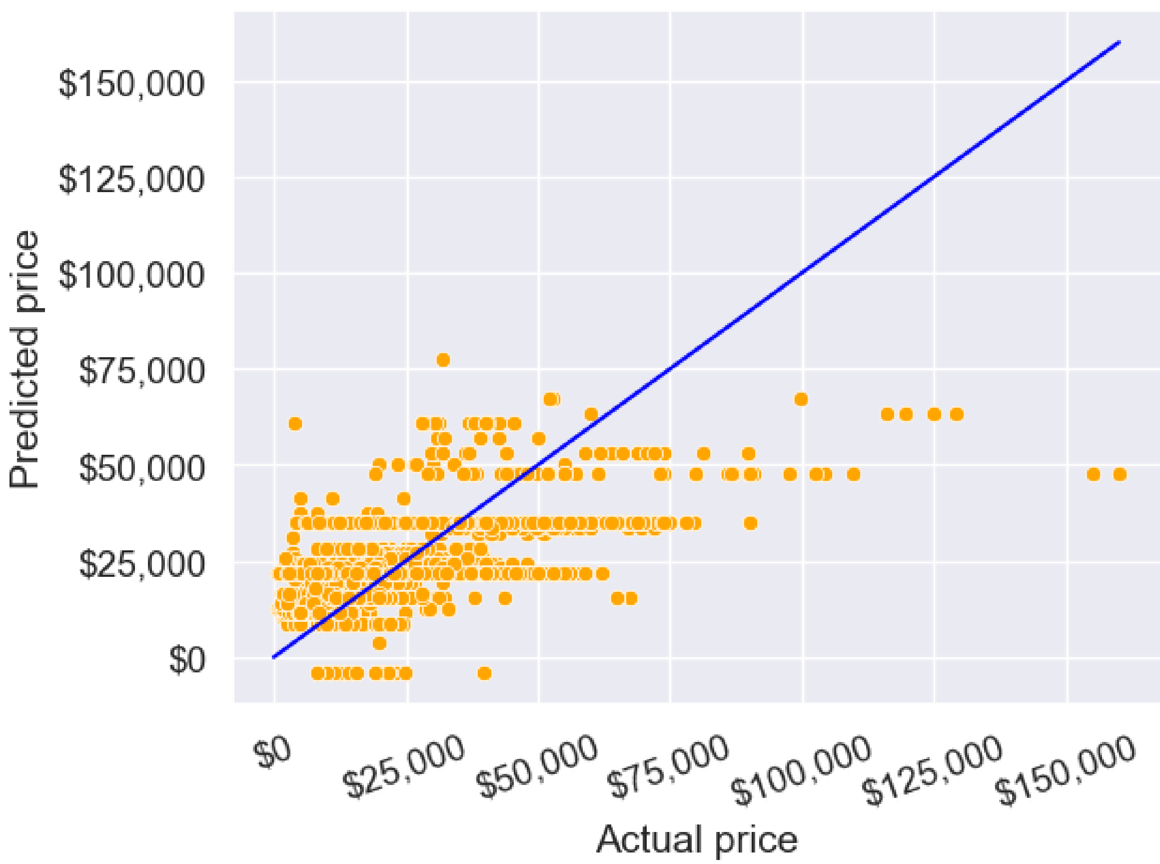
```
# Read the test data
testf = pd.read_csv('./Datasets/Car_features_test.csv') # Predictors
```

```
testp = pd.read_csv('./Datasets/Car_prices_test.csv') # Response
test = pd.merge(testf, testp)
```

```
#Using the predict() function associated with the 'model' object to make predictions of car p
pred_price = model.predict(testf)#Note that the predict() function finds the predictor 'engin
```

Make a visualization that compares the predicted car prices with the actual car prices

```
sns.scatterplot(x = testp.price, y = pred_price, color = 'orange')
#In case of a perfect prediction, all the points must lie on the line x = y.
ax = sns.lineplot(x = [0,testp.price.max()], y = [0,testp.price.max()],color='blue') #Plottin
plt.xlabel('Actual price')
plt.ylabel('Predicted price')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.xaxis.set_major_formatter('${x:,.0f}')
plt.xticks(rotation=20);
```



The prediction doesn't look too good. This is because we are just using one predictor - engine size. We can probably improve the model by adding more predictors when we learn multiple linear regression.

What is the RMSE of the predicted car price on unseen data?

```
np.sqrt(((testp.price - pred_price)**2).mean())
```

```
12995.106451548696
```

The root mean squared error in predicting car price is around $13k.

What is the residual standard error based on the training data?

```
np.sqrt(model.mse_resid)
```

```
12810.109175214138
```

The residual standard error on the training data is close to the RMSE on the test data. This shows that the performance of the model on unknown data is comparable to its performance on known data. This implies that the model is not overfitting, which is good! In case we overfit a model on the training data, its performance on unknown data is likely to be worse than that on the training data.

Find the confidence and prediction intervals of the predicted car price

```
#Using the get_prediction() function associated with the 'model' object to get the intervals
intervals = model.get_prediction(testf)
```

```
#The function requires specifying alpha (probability of Type 1 error) instead of the confiden
intervals.summary_frame(alpha=0.05)
```

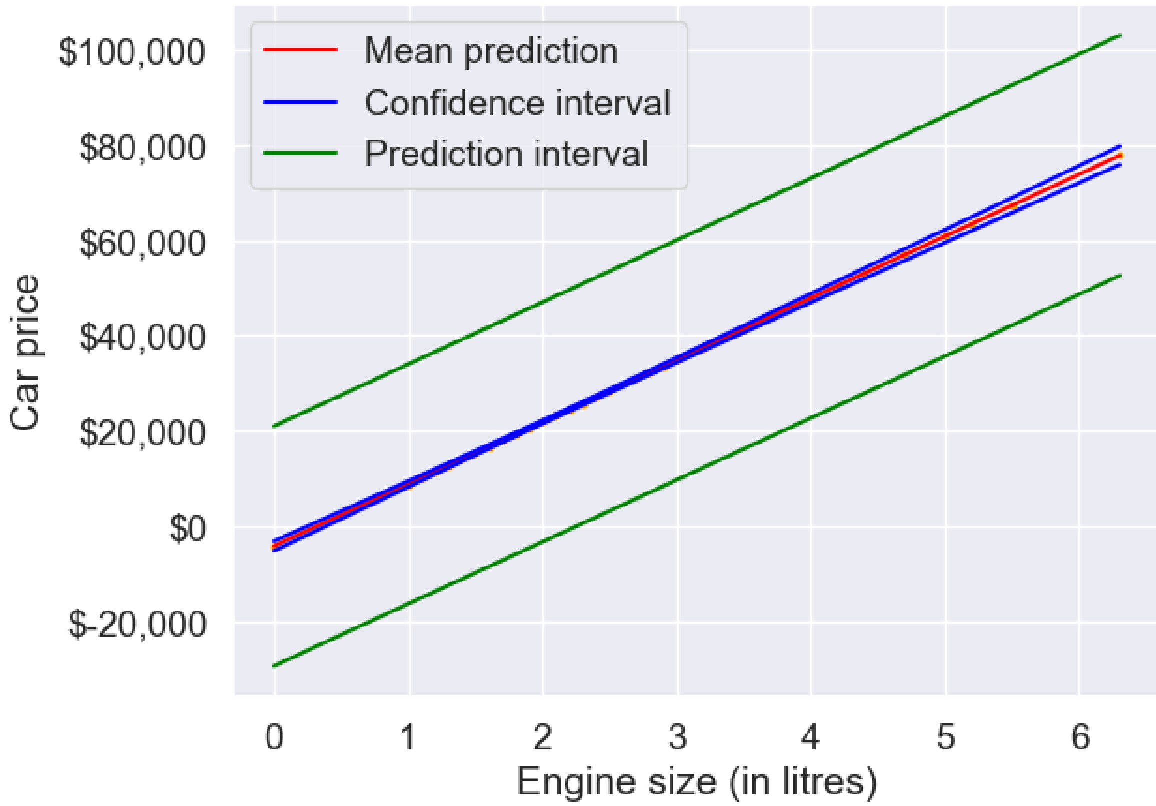
	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	34842.807319	271.666459	34310.220826	35375.393812	9723.677232	59961.937406
1	34842.807319	271.666459	34310.220826	35375.393812	9723.677232	59961.937406
2	34842.807319	271.666459	34310.220826	35375.393812	9723.677232	59961.937406
3	8866.245277	316.580850	8245.606701	9486.883853	-16254.905974	33987.396528
4	47831.088340	468.949360	46911.740050	48750.436631	22700.782946	72961.393735
...	...	...	...	...	...	...
2667	47831.088340	468.949360	46911.740050	48750.436631	22700.782946	72961.393735

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
2668	34842.807319	271.666459	34310.220826	35375.393812	9723.677232	59961.937406
2669	8866.245277	316.580850	8245.606701	9486.883853	-16254.905974	33987.396528
2670	21854.526298	184.135754	21493.538727	22215.513869	-3261.551421	46970.604017
2671	21854.526298	184.135754	21493.538727	22215.513869	-3261.551421	46970.604017

Show the regression line predicting car price based on engine size for test data. Also show the confidence and prediction intervals for the car price.

```
interval_table = intervals.summary_frame(alpha=0.05)
```

```
ax = sns.scatterplot(x = testf.engineSize, y = pred_price,color = 'orange', s = 10)
sns.lineplot(x = testf.engineSize, y = pred_price, color = 'red')
sns.lineplot(x = testf.engineSize, y = interval_table.mean_ci_lower, color = 'blue')
sns.lineplot(x = testf.engineSize, y = interval_table.mean_ci_upper, color = 'blue')
sns.lineplot(x = testf.engineSize, y = interval_table.obs_ci_lower, color = 'green')
sns.lineplot(x = testf.engineSize, y = interval_table.obs_ci_upper, color = 'green')

legend_elements = [Line2D([0], [0], color='red', label='Mean prediction'),
                   Line2D([0], [0], color='blue', label='Confidence interval'),
                  Line2D([0], [0], color='green', label='Prediction interval')]
ax.legend(handles=legend_elements, loc='upper left')
plt.xlabel('Engine size (in litres)')
plt.ylabel('Car price')
ax.yaxis.set_major_formatter('${x:,.0f}');
```



1.1.2 Training with sklearn

```
# Create the model as an object

model = LinearRegression() # No inputs, this will change for other models

# Train the model - separate the predictor(s) and the response for this!
X_train = train[['engineSize']]
y_train = train[['price']]

# Note that both are dfs, NOT series - necessary to avoid errors

model.fit(X_train, y_train)

# Check the slight syntax differences
    # predictors and response separate
    # We need to manually slice the predictor column(s) we want to include
```

```
    # No need to assign to an output

# Return the parameters
print("Coefficient of engine size = ", model.coef_) # slope
print("Intercept = ", model.intercept_) # intercept

# No .summary() here! - impossible to do much inference; this is a shortcoming of sklearn
```

```
Coefficient of engine size =  [[12988.28102112]]
Intercept =  [-4122.03574424]
```

```
# Prediction

# Again, separate the predictor(s) and the response of interest
X_test = test[['engineSize']]
y_test = test[['price']].to_numpy() # Easier to handle with calculations as np array

y_pred = model.predict(X_test)

# Evaluate
model_rmse = np.sqrt(np.mean((y_pred - y_test)**2)) # RMSE
model_mae = np.mean(np.abs(y_pred - y_test)) # MAE

print('Test RMSE: ', model_rmse)
```

```
Test RMSE:  12995.106451548696
```

```
# Easier way to calculate metrics with sklearn tools

# Note that we have imported the functions 'mean_squared_error' and 'mean_absolute_error'
# from the sklearn.metrics module (check top of the code)

model_rmse = np.sqrt(mean_squared_error(y_test,y_pred))
model_mae = mean_absolute_error(y_test,y_pred)
print('Test RMSE: ', model_rmse)
print('Test MAE: ', model_mae)
```

```
Test RMSE:  12995.106451548696
Test MAE:  9411.325912951994
```

```
y_pred_train = model.predict(X_train)
print('Train R-squared:', r2_score(y_train, y_pred_train))
print('Test R-squared:', r2_score(y_test, y_pred))
```

```
Train R-squared: 0.39049842625794573
Test R-squared: 0.3869900378620146
```

Note: Why did we repeat the same task in two different libraries?

- `statsmodels` and `sklearn` have different advantages - we will use both for our purposes
  - `statsmodels` returns a lot of statistical output, which is very helpful for inference (coming up next) but it has a limited variety of models.
  - With `statsmodels`, you may have columns in your DataFrame in addition to predictors and response, while with `sklearn` you need to make separate objects consisting of only the predictors and the response.
  - `sklearn` includes many models (`Lasso` and `Ridge` this quarter, many others next quarter) and helpful tools/functions (like metrics) that statsmodels does not but it does not have any inference tools.

1.1.3 Training with `statsmodels.api`

Earlier we had used the `statsmodels.formula.api` module, where we had to put the regression model as a formula. We can also use the `statsmodels.api` module to develop a regression model. The syntax of training a model with the `OLS()` function in this module is similar to that of `sklearn`'s `LinearRegression()` function. However, the order in which the predictors and response are specified is different. The formula-style syntax of the `statsmodels.formula.api` module is generally preferred. However, depending on the situation, the `OLS()` syntax of `statsmodels.api` may be preferred.

Note that you will manually need to add the predictor *(a column of ones)* corresponding to the intercept to train the model with this method.

```
# Create the model as an object

# Train the model - separate the predictor(s) and the response for this!
X_train = train[['engineSize']]
y_train = train[['price']]

X_train_with_intercept = np.concatenate((np.ones(X_train.shape[0]).reshape(-1,1), X_train), a

model = sm.OLS(y_train, X_train_with_intercept).fit()
```

```
# Return the parameters
print(model.params)
```

```
const    -4122.035744
x1       12988.281021
dtype: float64
```

The model summary and all other attributes and methods of the `model` object are the same as that with the object created using the `statsmodels.formula.api` module.

```
model.summary()
```

Dep. Variable:	price	**R-squared:**	0.390
Model:	OLS	**Adj. R-squared:**	0.390
Method:	Least Squares	**F-statistic:**	3177.
Date:	Mon, 08 Jan 2024	**Prob (F-statistic):**	0.00
Time:	11:17:55	**Log-Likelihood:**	-53949.
No. Observations:	4960	**AIC:**	1.079e+05
Df Residuals:	4958	**BIC:**	1.079e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	-4122.0357	522.260	-7.893	0.000	-5145.896	-3098.176
x1	1.299e+04	230.450	56.361	0.000	1.25e+04	1.34e+04

Omnibus:	1271.986	**Durbin-Watson:**	0.517
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6490.719
Skew:	1.137	**Prob(JB):**	0.00
Kurtosis:	8.122	**Cond. No.**	7.64

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

2 Multiple Linear Regression

Read section 3.2 of the book before using these notes.

Note that in this course, lecture notes are not sufficient, you must read the book for better understanding. Lecture notes are just implementing the concepts of the book on a dataset, but not explaining the concepts elaborately.

2.1 Multiple Linear Regression

```
# importing libraries
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

Develop a multiple linear regression model that predicts car price based on engine size, year, mileage, and mpg. Datasets to be used: *Car_features_train.csv, Car_prices_train.csv*

```
# Reading datasets
trainf = pd.read_csv('./Datasets/Car_features_train.csv')
trainp = pd.read_csv('./Datasets/Car_prices_train.csv')
train = pd.merge(trainf,trainp)
train.head()
```

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
0	18473	bmw	6 Series	2020	Semi-Auto	11	Diesel	145	53.3282	3.0	37980
1	15064	bmw	6 Series	2019	Semi-Auto	10813	Diesel	145	53.0430	3.0	33980
2	18268	bmw	6 Series	2020	Semi-Auto	6	Diesel	145	53.4379	3.0	36850
3	18480	bmw	6 Series	2017	Semi-Auto	18895	Diesel	145	51.5140	3.0	25998

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
4	18492	bmw	6 Series	2015	Automatic	62953	Diesel	160	51.4903	3.0	18990

2.1.1 Training the model

```
#Using the ols function to create an ols object. 'ols' stands for 'Ordinary least squares'
ols_object = smf.ols(formula = 'price~year+mileage+mpg+engineSize', data = train)
model = ols_object.fit()
model.summary()
```

Dep. Variable:	price	**R-squared:**	0.660
Model:	OLS	**Adj. R-squared:**	0.660
Method:	Least Squares	**F-statistic:**	2410.
Date:	Mon, 29 Jan 2024	**Prob (F-statistic):**	0.00
Time:	03:10:20	**Log-Likelihood:**	-52497.
No. Observations:	4960	**AIC:**	1.050e+05
Df Residuals:	4955	**BIC:**	1.050e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-3.661e+06	1.49e+05	-24.593	0.000	-3.95e+06	-3.37e+06
year	1817.7366	73.751	24.647	0.000	1673.151	1962.322
mileage	-0.1474	0.009	-16.817	0.000	-0.165	-0.130
mpg	-79.3126	9.338	-8.493	0.000	-97.620	-61.006
engineSize	1.218e+04	189.969	64.107	0.000	1.18e+04	1.26e+04

Omnibus:	2450.973	**Durbin-Watson:**	0.541
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	31060.548
Skew:	2.045	**Prob(JB):**	0.00
Kurtosis:	14.557	**Cond. No.**	3.83e+07

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.83e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

The model equation is: estimated car `price` = -3.661e6 + 1818 * `year` -0.15 * `mileage` - 79.31 * `mpg` + 12180 * `engineSize`

The procedure to fit the model using `sklearn` will be similar to that in simple linear regression.

```
model = LinearRegression()

X_train = train[['year','engineSize','mpg','mileage']] # Slice out the predictors
y_train = train[['price']]

model.fit(X_train,y_train)
```

2.1.2 Hypothesis test for a relationship between the response and a subset of predictors

Let us test the hypothesis if there is relationship between car price and the set of predictors: `mpg` and `year`.

```
hypothesis = '(mpg = 0, year = 0)'

model.f_test(hypothesis) # the F test of these two predictors is stat. sig.
```

```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=325.9206432972666, p=1.0499509223096256e-133, df_denom=4.96e+03, df_num=2>
```

As the p-value is low, we reject the null hypothesis, i.e., at least one of the predictors among `mpg` and `year` has a statistically significant relationship with car price.

Predict the car price for the cars in the test dataset. Datasets to be used: *Car_features_test.csv, Car_prices_test.csv*

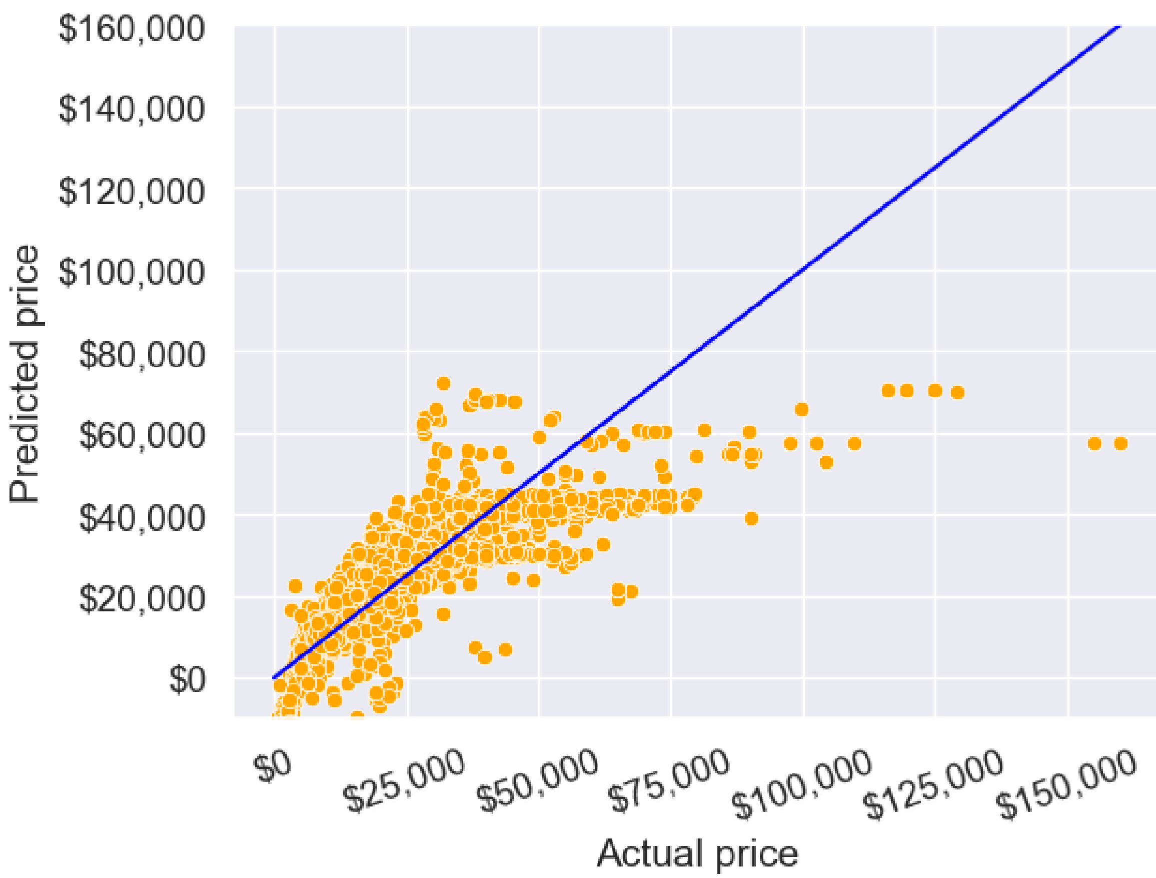
```
testf = pd.read_csv('./Datasets/Car_features_test.csv')
testp = pd.read_csv('./Datasets/Car_prices_test.csv')
```

2.1.3 Prediction

```
pred_price = model.predict(testf)
```

Make a visualization that compares the predicted car prices with the actual car prices

```
sns.set(font_scale=1.25)
sns.scatterplot(x = testp.price, y = pred_price, color = 'orange')
#In case of a perfect prediction, all the points must lie on the line x = y.
ax = sns.lineplot(x = [0,testp.price.max()], y = [0,testp.price.max()],color='blue') #Plottin
plt.xlabel('Actual price')
plt.ylabel('Predicted price')
plt.ylim([-10000, 160000])
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.xaxis.set_major_formatter('${x:,.0f}')
plt.xticks(rotation=20);
```



The prediction looks better as compared to the one with simple linear regression. This is because we have four predictors to help explain the variation in car price, instead of just one in the case of simple linear regression. Also, all the predictors have a significant relationship with price as evident from their p-values. Thus, all four of them are contributing in explaining the variation. Note the higher values of R^2 as compared to the one in the case of simple linear regression.

What is the RMSE of the predicted car price?

```
np.sqrt(((testp.price - pred_price)**2).mean())
```

```
9956.82497993548
```

What is the residual standard error based on the training data?

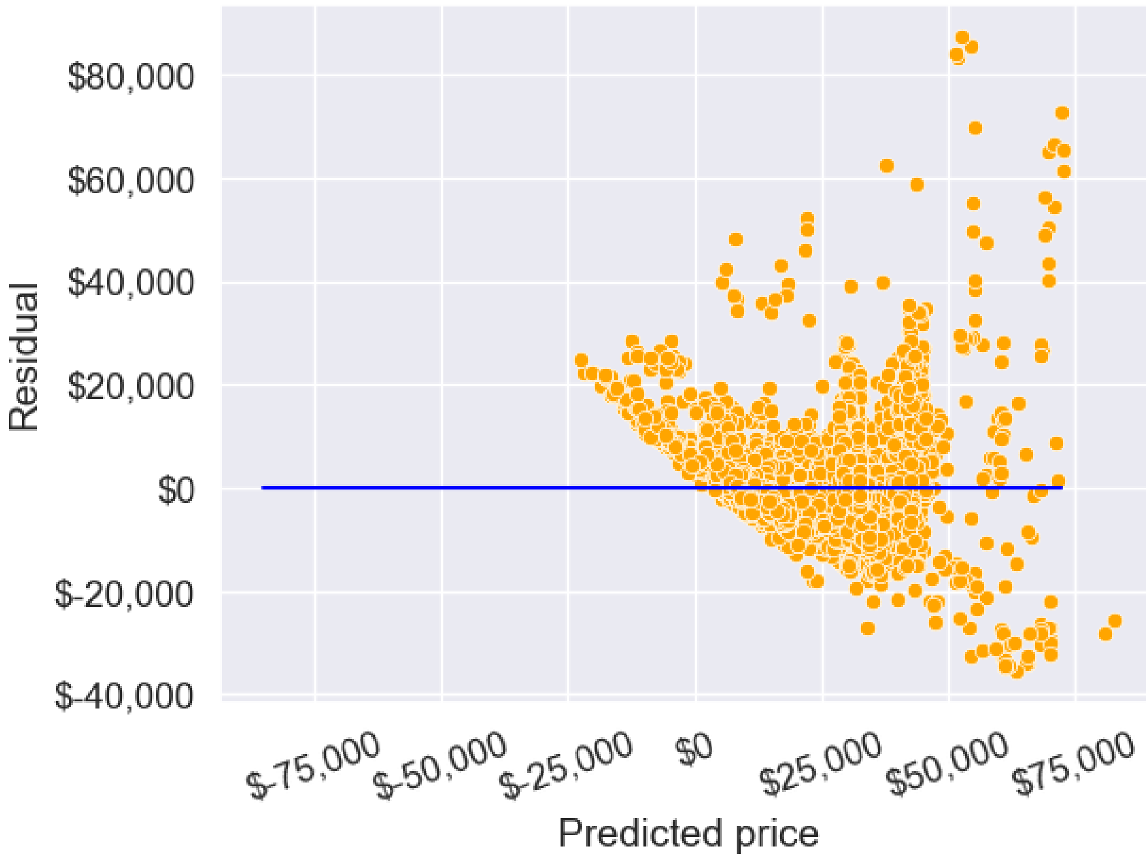
```
np.sqrt(model.mse_resid)
```

```
9563.74782917604
```

```
trainp.describe()
```

	carID	price
count	4960.000000	4960.000000
mean	15832.446169	23469.943750
std	2206.717006	16406.714563
min	12002.000000	450.000000
25%	13929.250000	12000.000000
50%	15840.000000	18999.000000
75%	17765.750000	30335.750000
max	19629.000000	145000.000000

```
sns.scatterplot(x = model.fittedvalues, y=model.resid,color = 'orange')
ax = sns.lineplot(x = [pred_price.min(),pred_price.max()],y = [0,0],color = 'blue')
plt.xlabel('Predicted price')
plt.ylabel('Residual')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.xaxis.set_major_formatter('${x:,.0f}')
plt.xticks(rotation=20);
```



2.1.4 Effect of adding noisy predictors on R^2

Will the explained variation (R-squared) in car price always increase if we add a variable?

Should we keep on adding variables as long as the explained variation (R-squared) is increasing?

```
#Using the ols function to create an ols object. 'ols' stands for 'Ordinary least squares'
np.random.seed(1)
train['rand_col'] = np.random.rand(train.shape[0])
ols_object = smf.ols(formula = 'price~year+mileage+mpg+engineSize+rand_col', data = train)
model = ols_object.fit()
model.summary()
```

Table 2.3: OLS Regression Results

Dep. Variable:	price	R-squared:	0.661
Model:	OLS	Adj. R-squared:	0.660
Method:	Least Squares	F-statistic:	1928.
Date:	Tue, 27 Dec 2022	Prob (F-statistic):	0.00
Time:	01:07:38	Log-Likelihood:	-52497.
No. Observations:	4960	AIC:	1.050e+05
Df Residuals:	4954	BIC:	1.050e+05
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-3.662e+06	1.49e+05	-24.600	0.000	-3.95e+06	-3.37e+06
year	1818.1672	73.753	24.652	0.000	1673.578	1962.756
mileage	-0.1474	0.009	-16.809	0.000	-0.165	-0.130
mpg	-79.2837	9.338	-8.490	0.000	-97.591	-60.976
engineSize	1.218e+04	189.972	64.109	0.000	1.18e+04	1.26e+04
rand_col	451.1226	471.897	0.956	0.339	-474.004	1376.249

Omnibus:	2451.728	Durbin-Watson:	0.541
Prob(Omnibus):	0.000	Jarque-Bera (JB):	31040.331
Skew:	2.046	Prob(JB):	0.00
Kurtosis:	14.552	Cond. No.	3.83e+07

Adding a variable with random values to the model (`rand_col`) increased the explained variation (R^2). This is because the model has one more parameter to tune to reduce the residual squared error (RSS). However, the p-value of `rand_col` suggests that its coefficient is zero. Thus, using the model with `rand_col` may give poorer performance on unknown data, as compared to the model without `rand_col`. This implies that it is not a good idea to blindly add variables in the model to increase R^2.

3 Variable interactions and transformations

Read sections 3.3.1 and 3.3.2 of the book before using these notes.

Note that in this course, lecture notes are not sufficient, you must read the book for better understanding. Lecture notes are just implementing the concepts of the book on a dataset, but not explaining the concepts elaborately.

3.1 Variable interactions

```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
trainf = pd.read_csv('./Datasets/Car_features_train.csv')
trainp = pd.read_csv('./Datasets/Car_prices_train.csv')
testf = pd.read_csv('./Datasets/Car_features_test.csv')
testp = pd.read_csv('./Datasets/Car_prices_test.csv')
train = pd.merge(trainf,trainp)
test = pd.merge(testf,testp)
train.head()
```

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
0	18473	bmw	6 Series	2020	Semi-Auto	11	Diesel	145	53.3282	3.0	37980
1	15064	bmw	6 Series	2019	Semi-Auto	10813	Diesel	145	53.0430	3.0	33980
2	18268	bmw	6 Series	2020	Semi-Auto	6	Diesel	145	53.4379	3.0	36850
3	18480	bmw	6 Series	2017	Semi-Auto	18895	Diesel	145	51.5140	3.0	25998
4	18492	bmw	6 Series	2015	Automatic	62953	Diesel	160	51.4903	3.0	18990

Until now, we have assumed that the association between a predictor X_j and response Y does not depend on the value of other predictors. For example, the multiple linear regression model that we developed in Chapter 2 assumes that the average increase in price associated with a unit increase in engineSize is always $12,180, regardless of the value of other predictors. However, this assumption may be incorrect.

3.1.1 Variable interaction between continuous predictors

We can relax this assumption by considering another predictor, called an interaction term. Let us assume that the average increase in `price` associated with a one-unit increase in `engineSize` depends on the model `year` of the car. In other words, there is an interaction between `engineSize` and `year`. This interaction can be included as a predictor, which is the product of `engineSize` and `year`. *Note that there are several possible interactions that we can consider. Here the interaction between* `engineSize` *and* `year` *is just an example.*

```
#Considering interaction between engineSize and year
ols_object = smf.ols(formula = 'price~year*engineSize+mileage+mpg', data = train)
model = ols_object.fit()
model.summary()
```

Table 3.2: OLS Regression Results

Dep. Variable:	price	R-squared:	0.682
Model:	OLS	Adj. R-squared:	0.681
Method:	Least Squares	F-statistic:	2121.
Date:	Tue, 24 Jan 2023	Prob (F-statistic):	0.00
Time:	15:28:11	Log-Likelihood:	-52338.
No. Observations:	4960	AIC:	1.047e+05
Df Residuals:	4954	BIC:	1.047e+05
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	5.606e+05	2.74e+05	2.048	0.041	2.4e+04	1.1e+06
year	-275.3833	135.695	-2.029	0.042	-541.405	-9.361
engineSize	-1.796e+06	9.97e+04	-18.019	0.000	-1.99e+06	-1.6e+06
year:engineSize	896.7687	49.431	18.142	0.000	799.861	993.676
mileage	-0.1525	0.008	-17.954	0.000	-0.169	-0.136
mpg	-84.3417	9.048	-9.322	0.000	-102.079	-66.604

Omnibus:	2330.413	Durbin-Watson:	0.524
Prob(Omnibus):	0.000	Jarque-Bera (JB):	29977.437
Skew:	1.908	Prob(JB):	0.00
Kurtosis:	14.423	Cond. No.	7.66e+07

Note that the R-squared has increased as compared to the model in Chapter 2 since we added a predictor.

The model equation is:

$$price = \beta_0 + \beta_1 * year + \beta_2 * engineSize + \beta_3 * (year * engineSize) + \beta 4 * mileage + \beta_5 * mpg, \tag{3.1}$$

or

$$price = \beta_0 + \beta_1 * year + (\beta_2 + \beta_3 * year) * engineSize + \beta 4 * mileage + \beta_5 * mpg, \tag{3.2}$$

or

$$price = \beta_0 + \beta_1 * year + \tilde{\beta} * engineSize + \beta 4 * mileage + \beta_5 * mpg, \tag{3.3}$$

Since $\tilde{\beta}$ is a function of `year`, the association between `engineSize` and `price` is no longer a constant. A change in the value of `year` will change the association between `price` and `engineSize`.

Substituting the values of the coefficients:

`price` = 5.606e5 - 275.3833*year* + *(-1.796e6+896.7687year)*`engineSize` -0.1525*mileage* - *84.3417*`mpg`

Thus, for cars launched in the year 2010, the average increase in price for one liter increase in engine size is -1.796e6 + 896.7687 * 2010 ≈ \$6,500, assuming all the other predictors are constant. However, for cars launched in the year 2020, the average increase in price for one liter increase in engine size is -1.796e6 + 896.7687*2020 ≈ \$15,500 , assuming all the other predictors are constant.

Similarly, the equation can be re-arranged as:

`price` = 5.606e5 +(-275.3833+896.7687*engineSize)*`year` -1.796e6*engineSize -0.1525*`mileage` -84.3417*`mpg`

Thus, for cars with an engine size of 2 litres, the average increase in price for a one year newer model is -275.3833+896.7687 * 2 ≈ \$1500, assuming all the other predictors are constant.

However, for cars with an engine size of 3 litres, the average increase in price for a one year newer model is -275.3833+896.7687 * 3 ≈ \$2400, assuming all the other predictors are constant.

```
#Computing the RMSE of the model with the interaction term
pred_price = model.predict(testf)
np.sqrt(((testp.price - pred_price)**2).mean())
```

```
9423.598872501092
```

Note that the RMSE is lower than that of the model in Chapter 2. This is because the interaction term between `engineSize` and `year` is significant and relaxes the assumption of constant association between price and engine size, and between price and year. This added flexibility makes the model better fit the data. Caution: Too much flexibility may lead to overfitting!

Note that interaction terms corresponding to other variable pairs, and higher order interaction terms (such as those containing 3 or 4 variables) may also be significant and improve the model fit & thereby the prediction accuracy of the model.

3.1.2 Including qualitative predictors in the model

Let us develop a model for predicting `price` based on `engineSize` and the qualitative predictor `transmission`.

```
#checking the distribution of values of transmission
train.transmission.value_counts()
```

```
Manual       1948
Automatic    1660
Semi-Auto    1351
Other           1
Name: transmission, dtype: int64
```

Note that the *Other* category of the variable *transmission* contains only a single observation, which is likely to be insufficient to train the model. We'll remove that observation from the training data. Another option may be to combine the observation in the *Other* category with the nearest category, and keep it in the data.

```
train_updated = train[train.transmission!='Other']
```

```
ols_object = smf.ols(formula = 'price ~ engineSize + transmission', data = train_updated)
model = ols_object.fit()
model.summary()
```

Table 3.5: OLS Regression Results

Dep. Variable:	price	R-squared:	0.459
Model:	OLS	Adj. R-squared:	0.458
Method:	Least Squares	F-statistic:	1400.
Date:	Tue, 24 Jan 2023	Prob (F-statistic):	0.00
Time:	15:28:21	Log-Likelihood:	-53644.
No. Observations:	4959	AIC:	1.073e+05
Df Residuals:	4955	BIC:	1.073e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3042.6765	661.190	4.602	0.000	1746.451	4338.902
transmission[T.Manual]	-6770.6165	442.116	-15.314	0.000	-7637.360	-5903.873
transmission[T.Semi-Auto]	4994.3112	442.989	11.274	0.000	4125.857	5862.765
engineSize	1.023e+04	247.485	41.323	0.000	9741.581	1.07e+04

Omnibus:	1575.518	Durbin-Watson:	0.579
Prob(Omnibus):	0.000	Jarque-Bera (JB):	11006.609
Skew:	1.334	Prob(JB):	0.00
Kurtosis:	9.793	Cond. No.	11.4

Note that there is no coefficient for the *Automatic* level of the variable `Transmission`. If a car doesn't have *Manual* or *Semi-Automatic* transmission, then it has an *Automatic* transmission. Thus, the coefficient of *Automatic* will be redundant, and the dummy variable corresponding to *Automatic* transmission is dropped from the model.

The level of the categorical variable that is dropped from the model is called the baseline level. Here *Automatic* transmission is the baseline level. The coefficients of other levels of `transmission` should be interpreted with respect to the baseline level.

Q: Interpret the intercept term

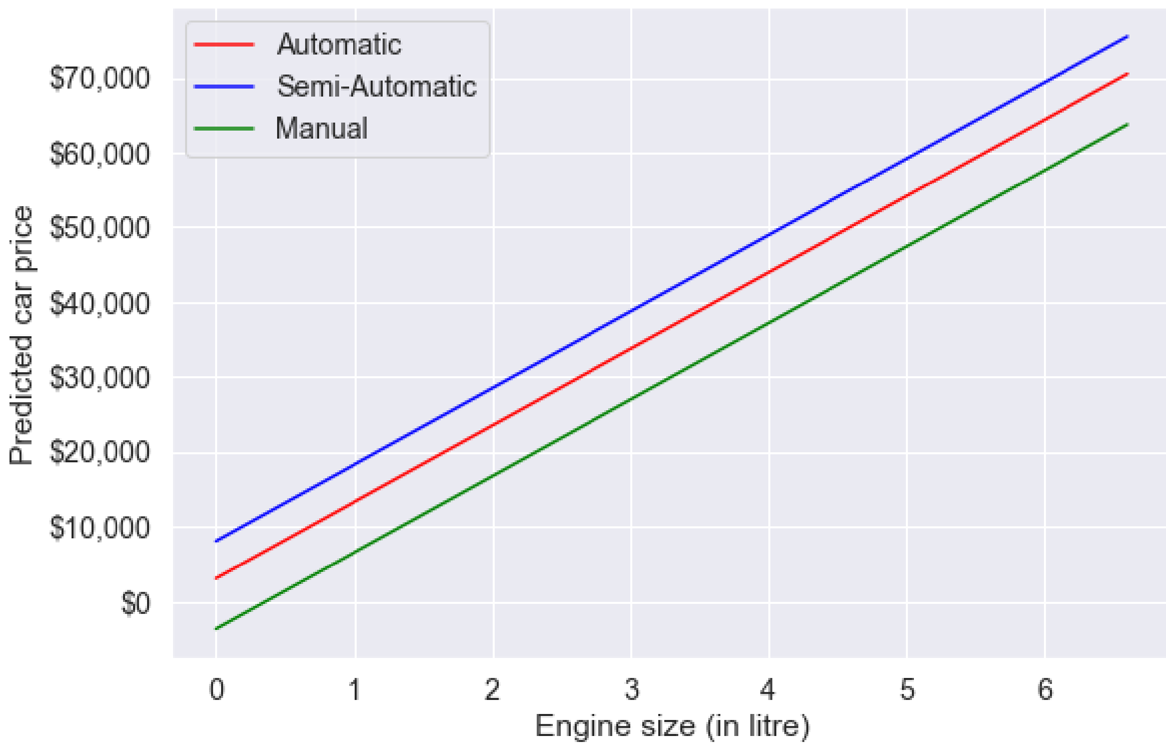
Ans: For the hypothetical scenario of a car with zero engine size and *Automatic* transmission, the estimated mean car price is $\approx$ \$3042.

Q: Interpret the coefficient of `transmission[T.Manual]`

Ans: The estimated mean price of a car with manual transmission is $\approx$ \$6770 less than that of a car with *Automatic* transmission.

Let us visualize the developed model.

```
#Visualizing the developed model
plt.rcParams["figure.figsize"] = (9,6)
sns.set(font_scale = 1.3)
x = np.linspace(train_updated.engineSize.min(),train_updated.engineSize.max(),100)
ax = sns.lineplot(x = x, y = model.params['engineSize']*x+model.params['Intercept'], color =
sns.lineplot(x = x, y = model.params['engineSize']*x+model.params['Intercept']+model.params[
sns.lineplot(x = x, y = model.params['engineSize']*x+model.params['Intercept']+model.params[
plt.legend(labels=["Automatic","Semi-Automatic", "Manual"])
plt.xlabel('Engine size (in litre)')
plt.ylabel('Predicted car price')
ax.yaxis.set_major_formatter('${x:,.0f}')
```



Based on the developed model, for a given engine size, the car with a semi-automatic transmission is estimated to be the most expensive on average, while the car with a manual transmission is estimated to be the least expensive on average.

Changing the baseline level: By default, the baseline level is chosen as the one that comes first if the levels are arranged in alphabetical order. However, you can change the baseline level by specifying one explicitly.

Internally, statsmodels uses the patsy package to convert formulas and data to the matrices that are used in model fitting. You may refer to this section in the patsy documentation to specify a particular level of the categorical variable as the baseline.

For example, suppose we wish to change the baseline level to *Manual* transmission. We can specify this in the formula as follows:

```
ols_object = smf.ols(formula = 'price~engineSize+C(transmission, Treatment("Manual"))', data
model = ols_object.fit()
model.summary()
```

Table 3.8: OLS Regression Results

Dep. Variable:	price	R-squared:	0.459
Model:	OLS	Adj. R-squared:	0.458
Method:	Least Squares	F-statistic:	1400.
Date:	Tue, 24 Jan 2023	Prob (F-statistic):	0.00
Time:	15:28:39	Log-Likelihood:	-53644.
No. Observations:	4959	AIC:	1.073e+05
Df Residuals:	4955	BIC:	1.073e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.9
Intercept	-3727.9400	492.917	-7.563	0.000	-4694.275	-27
C(transmission, Treatment("Manual"))[T.Automatic]	6770.6165	442.116	15.314	0.000	5903.873	763
C(transmission, Treatment("Manual"))[T.Semi-Auto]	1.176e+04	473.110	24.867	0.000	1.08e+04	1.2
engineSize	1.023e+04	247.485	41.323	0.000	9741.581	1.0

Omnibus:	1575.518	Durbin-Watson:	0.579
Prob(Omnibus):	0.000	Jarque-Bera (JB):	11006.609
Skew:	1.334	Prob(JB):	0.00
Kurtosis:	9.793	Cond. No.	8.62

3.1.3 Including qualitative predictors and their interaction with continuous predictors in the model

Note that the qualitative predictor leads to fitting 3 parallel lines to the data, as there are 3 categories.

However, note that we have made the constant association assumption. The fact that the lines are parallel means that the average increase in car price for one litre increase in engine size does not depend on the type of transmission. This represents a potentially serious limitation of the model, since in fact a change in engine size may have a very different association on the price of an automatic car versus a semi-automatic or manual car.

This limitation can be addressed by adding an interaction variable, which is the product of `engineSize` and the dummy variables for semi-automatic and manual transmissions.

```
#Using the ols function to create an ols object. 'ols' stands for 'Ordinary least squares'
ols_object = smf.ols(formula = 'price~engineSize*transmission', data = train_updated)
model = ols_object.fit()
model.summary()
```

Table 3.11: OLS Regression Results

Dep. Variable:	price	R-squared:	0.479
Model:	OLS	Adj. R-squared:	0.478
Method:	Least Squares	F-statistic:	909.9
Date:	Sun, 22 Jan 2023	Prob (F-statistic):	0.00
Time:	22:55:55	Log-Likelihood:	-53550.
No. Observations:	4959	AIC:	1.071e+05
Df Residuals:	4953	BIC:	1.072e+05
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3754.7238	895.221	4.194	0.000	1999.695	5509.753
transmission[T.Manual]	1768.5856	1294.071	1.367	0.172	-768.366	4305.538
transmission[T.Semi-Auto]	-5282.7164	1416.472	-3.729	0.000	-8059.628	-2505.805
engineSize	9928.6082	354.511	28.006	0.000	9233.610	1.06e+04
engineSize:transmission[T.Manual]	-5285.9059	646.175	-8.180	0.000	-6552.695	-4019.117
engineSize:transmission[T.Semi-Auto]	4162.2428	552.597	7.532	0.000	3078.908	5245.578

Omnibus:	1379.846	Durbin-Watson:	0.622
Prob(Omnibus):	0.000	Jarque-Bera (JB):	9799.471
Skew:	1.139	Prob(JB):	0.00
Kurtosis:	9.499	Cond. No.	30.8

The model equation for the model with interactions is:

Automatic transmission: `price` = 3754.7238 + 9928.6082 * `engineSize`,

Semi-Automatic transmission: `price` = 3754.7238 + 9928.6082 * `engineSize` + (-5282.7164+4162.2428*`engineSize`),

Manual transmission: `price` = 3754.7238 + 9928.6082 * `engineSize` +(1768.5856-5285.9059 * `engineSize`),

or

Automatic transmission: `price` = 3754.7238 + 9928.6082 * `engineSize`,

Semi-Automatic transmission: `price` = -1527 + 7046 * `engineSize`,

Manual transmission: `price` = 5523 + 4642 * `engineSize`

Q: Interpret the coefficient of manual tranmission, i.e., the coefficient of `transmission[T.Manual]`.

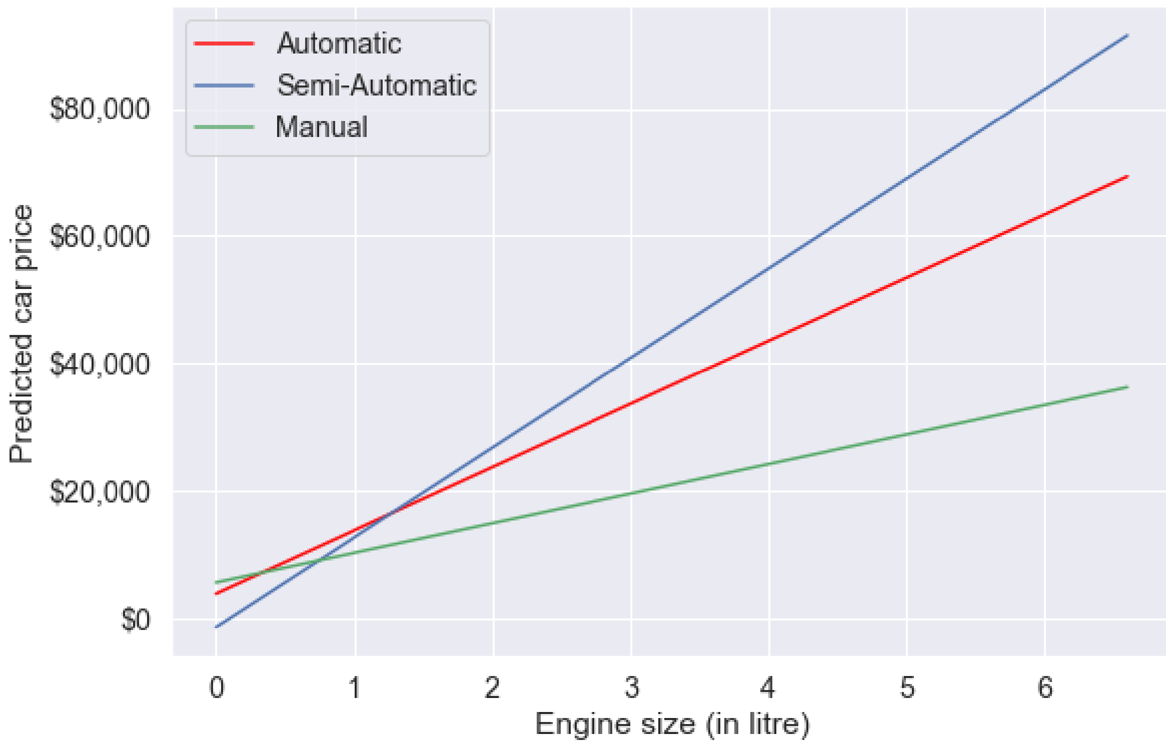
A: For a hypothetical scenario of zero engine size, the estimated mean `price` of a car with *Manual* `transmission` is ≈ \$1768 more than the estimated mean `price` of a car with *Automatic* `transmission`.

Q: Interpret the coefficient of the interaction between engine size and manual transmission, i.e., the coefficient of `engineSize:transmission[T.Manual]`.

A: For a unit (or a litre) increase in `engineSize` , the increase in estimated mean `price` of a car with *Manual* transmission is ≈ \$5285 less than the increase in estimated mean `price` of a car with *Automatic* transmission.

```
#Visualizing the developed model with interaction terms
plt.rcParams["figure.figsize"] = (9,6)
sns.set(font_scale = 1.3)
x = np.linspace(train_updated.engineSize.min(),train_updated.engineSize.max(),100)
ax = sns.lineplot(x = x, y = model.params['engineSize']*x+model.params['Intercept'], label='A
plt.plot(x, (model.params['engineSize']+model.params['engineSize:transmission[T.Semi-Auto]'])
plt.plot(x, (model.params['engineSize']+model.params['engineSize:transmission[T.Manual]'])*x+
plt.legend(loc='upper left')
plt.xlabel('Engine size (in litre)')
```

```
plt.ylabel('Predicted car price')
ax.yaxis.set_major_formatter('${x:,.0f}')
```



Note the interaction term adds flexibility to the model.

The slope of the regression line for semi-automatic cars is the largest. This suggests that increase in engine size is associated with a higher increase in car price for semi-automatic cars, as compared to other cars.

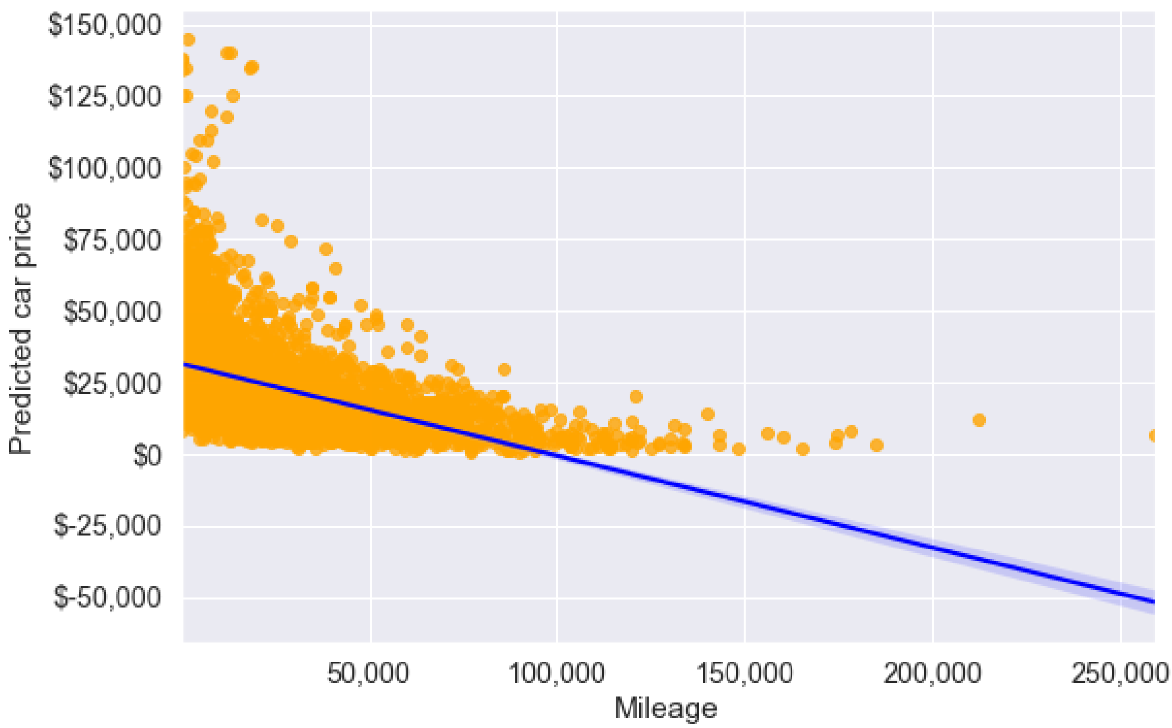
3.2 Variable transformations

So far we have considered only a linear relationship between the predictors and the response. However, the relationship may be non-linear.

Consider the regression plot of `price` on `mileage`.

```
ax = sns.regplot(x = train_updated.mileage, y =train_updated.price,color = 'orange', line_kws
plt.xlabel('Mileage')
plt.ylabel('Predicted car price')
```

```
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.xaxis.set_major_formatter('{x:,.0f}')
```




```
#R-squared of the model with just mileage
model = smf.ols('price~mileage', data = train_updated).fit()
model.rsquared
```

```
0.22928048993376182
```

From the first scatterplot, we see that the relationship between `price` and `mileage` doesn't seem to be linear, as the points do not lie on a straight line. Also, we see the regression line (or the curve), which is the best fit line doesn't seem to fit the points well. However, `price` on average seems to decrease with `mileage`, albeit in a non-linear manner.

3.2.1 Quadratic transformation

So, we guess that if we model price as a quadratic function of `mileage`, the model may better fit the points (or the curve may better fit the points). Let us transform the predictor `mileage` to include $mileage^2$ (i.e., perform a quadratic transformation on the predictor).

```
#Including mileage squared as a predictor and developing the model
ols_object = smf.ols(formula = 'price~mileage+I(mileage**2)', data = train_updated)
model = ols_object.fit()
model.summary()
```

Table 3.14: OLS Regression Results

Dep. Variable:	price	R-squared:	0.271
Model:	OLS	Adj. R-squared:	0.271
Method:	Least Squares	F-statistic:	920.6
Date:	Sun, 22 Jan 2023	Prob (F-statistic):	0.00
Time:	23:26:05	Log-Likelihood:	-54382.
No. Observations:	4959	AIC:	1.088e+05
Df Residuals:	4956	BIC:	1.088e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3.44e+04	332.710	103.382	0.000	3.37e+04	3.5e+04
mileage	-0.5662	0.017	-33.940	0.000	-0.599	-0.534
I(mileage ** 2)	2.629e-06	1.56e-07	16.813	0.000	2.32e-06	2.94e-06

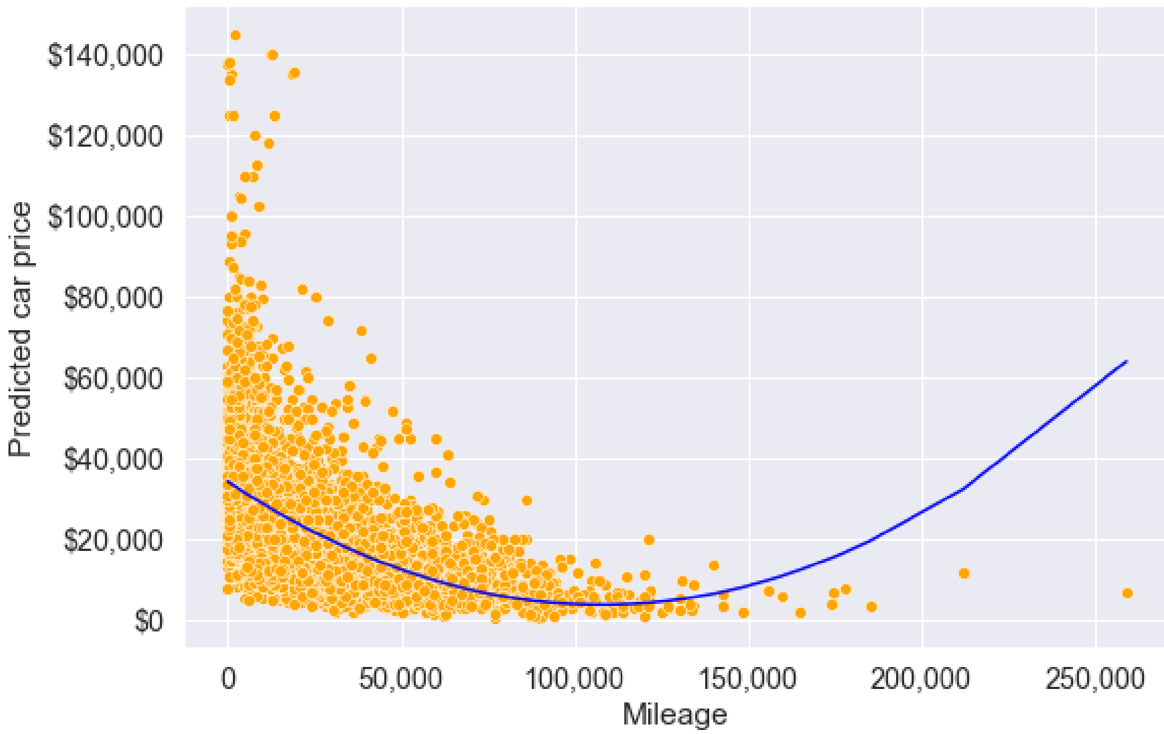
Omnibus:	2362.973	Durbin-Watson:	0.325
Prob(Omnibus):	0.000	Jarque-Bera (JB):	22427.952
Skew:	2.052	Prob(JB):	0.00
Kurtosis:	12.576	Cond. No.	4.81e+09

Note that in the formula specified within the `ols()` function, the `I()` operator isolates or insulates the contents within I(…) from the regular formula operators. Without the `I()` operator, `mileage**2` will be treated as the interaction of `mileage` with itself, which is `mileage`. Thus, to add the square of `mileage` as a separate predictor, we need to use the `I()` operator.

Let us visualize the model fit with the quadratic transformation of the predictor - `mileage`.

```
#Visualizing the regression line with the model consisting of the quadratic transformation o
pred_price = model.predict(train_updated)
ax = sns.scatterplot(x = 'mileage', y = 'price', data = train_updated, color = 'orange')
sns.lineplot(x = train_updated.mileage, y = pred_price, color = 'blue')
plt.xlabel('Mileage')
```

```
plt.ylabel('Predicted car price')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.xaxis.set_major_formatter('{x:,.0f}')
```



The above model seems to better fit the data (as compared to the model without transformation) at least upto mileage around 125,000. The R^2 of the model with the quadratic transformation of `mileage` is also higher than that of the model without transformation indicating a better fit.

3.2.2 Cubic transformation

Let us see if a cubic transformation of `mileage` can further improve the model fit.

```
#Including mileage squared and mileage cube as predictors and developing the model
ols_object = smf.ols(formula = 'price~mileage+I(mileage**2)+I(mileage**3)', data = train_upda
model = ols_object.fit()
model.summary()
```

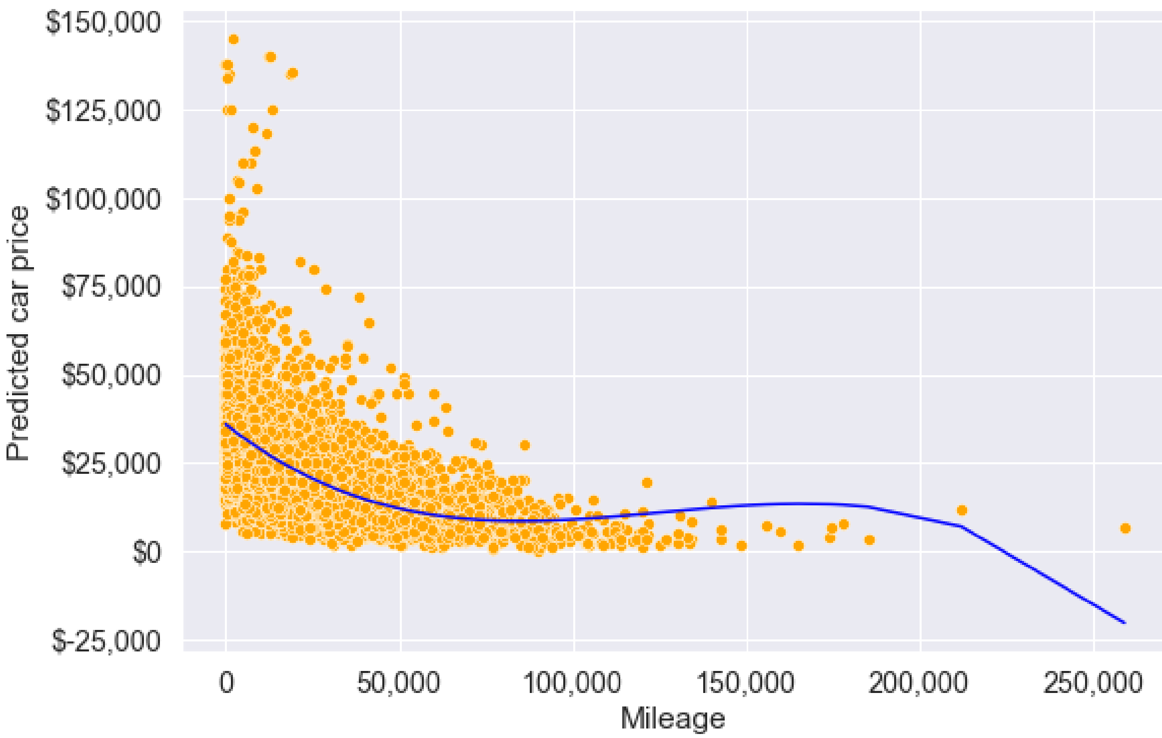
Table 3.17: OLS Regression Results

Dep. Variable:	price	R-squared:	0.283
Model:	OLS	Adj. R-squared:	0.283
Method:	Least Squares	F-statistic:	652.3
Date:	Sun, 22 Jan 2023	Prob (F-statistic):	0.00
Time:	23:33:27	Log-Likelihood:	-54340.
No. Observations:	4959	AIC:	1.087e+05
Df Residuals:	4955	BIC:	1.087e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3.598e+04	371.926	96.727	0.000	3.52e+04	3.67e+04
mileage	-0.7742	0.028	-27.634	0.000	-0.829	-0.719
I(mileage ** 2)	6.875e-06	4.87e-07	14.119	0.000	5.92e-06	7.83e-06
I(mileage ** 3)	-1.823e-11	1.98e-12	-9.199	0.000	-2.21e-11	-1.43e-11

Omnibus:	2380.788	Durbin-Watson:	0.321
Prob(Omnibus):	0.000	Jarque-Bera (JB):	23039.307
Skew:	2.065	Prob(JB):	0.00
Kurtosis:	12.719	Cond. No.	7.73e+14

```
#Visualizing the model with the cubic transformation of mileage
pred_price = model.predict(train_updated)
ax = sns.scatterplot(x = 'mileage', y = 'price', data = train_updated, color = 'orange')
sns.lineplot(x = train_updated.mileage, y = pred_price, color = 'blue')
plt.xlabel('Mileage')
plt.ylabel('Predicted car price')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.xaxis.set_major_formatter('{x:,.0f}')
```



Note that the model fit with the cubic transformation of `mileage` seems slightly better as compared to the models with the quadratic transformation, and no transformation of mileage, for mileage up to 180k. However, the model should not be used to predict car prices of cars with a mileage higher than 180k.

Let's update the model created earlier (in the beginning of this chapter) to include the transformed predictor.

```
#Model with an interaction term and a variable transformation term
ols_object = smf.ols(formula = 'price~year*engineSize+mileage+mpg+I(mileage**2)', data = tra
model = ols_object.fit()
model.summary()
```

Table 3.20: OLS Regression Results

Dep. Variable:	price	R-squared:	0.702
Model:	OLS	Adj. R-squared:	0.702
Method:	Least Squares	F-statistic:	1947.
Date:	Sun, 22 Jan 2023	Prob (F-statistic):	0.00
Time:	23:42:13	Log-Likelihood:	-52162.
No. Observations:	4959	AIC:	1.043e+05

Df Residuals:	4952	BIC:	1.044e+05
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.53e+06	2.7e+05	5.671	0.000	1e+06	2.06e+06
year	-755.7419	133.791	-5.649	0.000	-1018.031	-493.453
engineSize	-2.022e+06	9.72e+04	-20.803	0.000	-2.21e+06	-1.83e+06
year:engineSize	1008.6993	48.196	20.929	0.000	914.215	1103.184
mileage	-0.3548	0.014	-25.973	0.000	-0.382	-0.328
mpg	-54.7450	8.896	-6.154	0.000	-72.185	-37.305
I(mileage ** 2)	1.926e-06	1.04e-07	18.536	0.000	1.72e-06	2.13e-06

Omnibus:	2355.448	Durbin-Watson:	0.562
Prob(Omnibus):	0.000	Jarque-Bera (JB):	38317.404
Skew:	1.857	Prob(JB):	0.00
Kurtosis:	16.101	Cond. No.	6.40e+12

Note that the R-squared has increased as compared to the model with just the interaction term.

```
#Computing RMSE on test data
pred_price = model.predict(testf)
np.sqrt(((testp.price - pred_price)**2).mean())
```

```
9074.494088619422
```

Note that the prediction accuracy of the model has further increased, as the RMSE has reduced. The transformed predictor is statistically significant and provides additional flexibility to better capture the trend in the data, leading to an increase in prediction accuracy.

3.3 `PolynomialFeatures()`

The function `PolynomialFeatures()` from the `sklearn` library can be used to generate a predictor matrix that includes all interactions and transformations upto a degree `d`.

```
X_train = train[['mileage', 'engineSize', 'year', 'mpg']]
y_train = train[['price']]
X_test = test[['mileage', 'engineSize', 'year', 'mpg']]
y_test = test[['price']]
```

3.3.1 Generating polynomial features

Let us generate polynomial features upto degree 2. This will include all the two-factor interactions, and all squared terms of degree 2.

```
poly = PolynomialFeatures(2, include_bias = False) # Create the object - degree is 2

# Generate the polynomial features
X_train_poly = poly.fit_transform(X_train)
```

Note that the `LinearRegression()` function adds the intercept by default *(check the* [*fit_intercept*] *argument)*. Thus, we have put `include_bias = False` while generating the polynomial features, as we don't need the intercept. The term `bias` here refers to the intercept *(you will learn about `bias` in detail in STAT303-3)*. Another option is to include the intercept while generating the polynomial features, and put `fit_intercept = False` in the `LinearRegression()` function.

Below are the polynomial features generated by the `PolynomialFeatures()` functions.

```
poly.get_feature_names_out()
```

```
array(['mileage', 'engineSize', 'year', 'mpg', 'mileage^2',
       'mileage engineSize', 'mileage year', 'mileage mpg',
       'engineSize^2', 'engineSize year', 'engineSize mpg', 'year^2',
       'year mpg', 'mpg^2'], dtype=object)
```

3.3.2 Fitting the model

```
model = LinearRegression()
model.fit(X_train_poly, y_train)
```

```
LinearRegression()
```

3.3.3 Testing the model

```
X_test_poly = poly.fit_transform(X_test)
```

```
#RMSE
np.sqrt(mean_squared_error(y_test, model.predict(X_test_poly)))
```

```
8896.175508213777
```

Note that the polynomial features have helped reduced the RMSE further.

4 Logistic regression

Read sections 4.1 - 4.3 of the book before using these notes.

Note that in this course, lecture notes are not sufficient, you must read the book for better understanding. Lecture notes are just implementing the concepts of the book on a dataset, but not explaining the concepts elaborately.

4.1 Theory Behind Logistic Regression

Logistic regression is the go-to linear classification algorithm for two-class problems. It is easy to implement, easy to understand and gets great results on a wide variety of problems, even when the expectations the method has for your data are violated.

4.1.1 Description

Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the **`Sigmoid function`** was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$\frac{1}{1+e^{-x}}$$

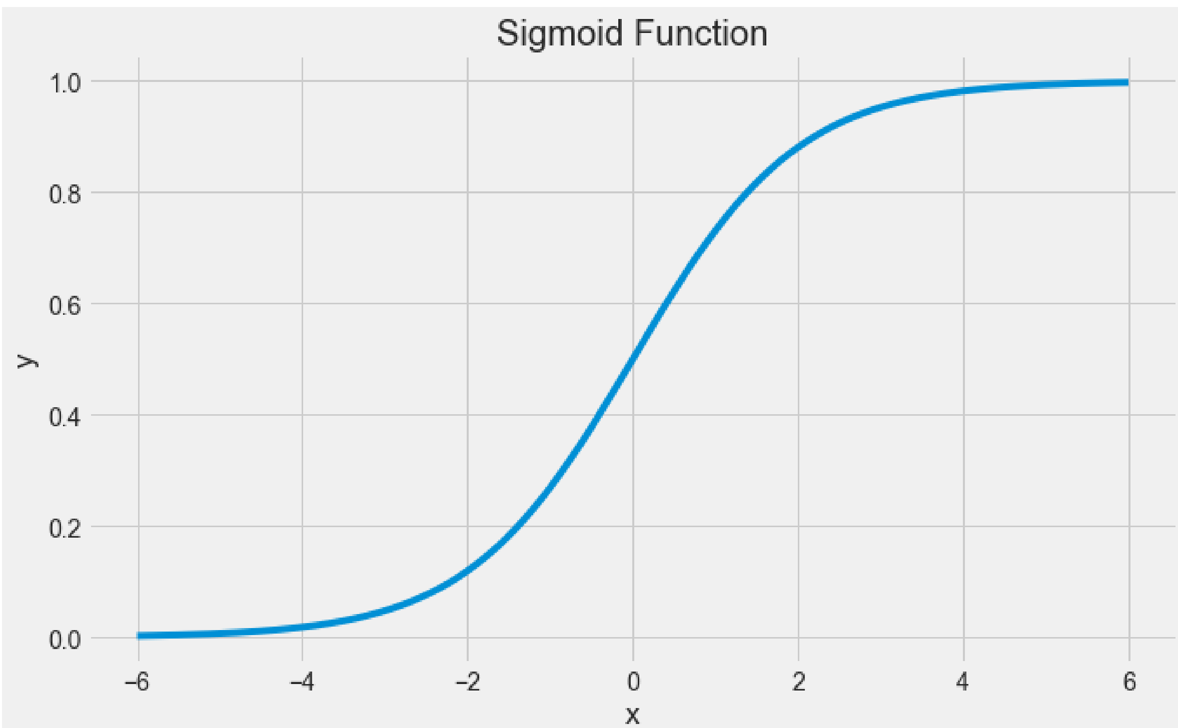
e is the base of the natural logarithms and x is value that you want to transform via the logistic function.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as sm
```

```python
from sklearn.metrics import precision_recall_curve, roc_curve, auc, accuracy_score
from sklearn.linear_model import LogisticRegression
```

```python
%matplotlib inline
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
x = np.linspace(-6, 6, num=1000)
plt.figure(figsize=(10, 6))
plt.plot(x, (1 / (1 + np.exp(-x))))
plt.xlabel("x")
plt.ylabel("y")
plt.title("Sigmoid Function")
```

```
Text(0.5, 1.0, 'Sigmoid Function')
```



The logistic regression equation has a very similar representation like linear regression. The difference is that the output value being modelled is binary in nature.

$$\hat{p} = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x_1}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_1}}$$

or

$$\hat{p} = \frac{1.0}{1.0 + e^{-(\hat{\beta}_0 + \hat{\beta}_1 x_1)}}$$

$\hat{\beta}_0$ is the estimated intercept term

$\hat{\beta}_1$ is the estimated coefficient for x_1

$\hat{p}$ is the predicted output with real value between 0 and 1. To convert this to binary output of 0 or 1, this would either need to be rounded to an integer value or a cutoff point be provided to specify the class segregation point.

4.1.2 Learning the Logistic Regression Model

The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation.

Maximum-likelihood estimation is a common learning algorithm used by a variety of machine learning algorithms, although it does make assumptions about the distribution of your data (more on this when we talk about preparing your data).

The best coefficients should result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class. The intuition for maximum-likelihood for logistic regression is that a search procedure seeks values for the coefficients (Beta values) that maximize the likelihood of the observed data. In other words, in MLE, we estimate the parameter values (Beta values) which are the most likely to produce that data at hand.

Here is an analogy to understand the idea behind Maximum Likelihood Estimation (MLE). Let us say, you are listening to a song (data). You are not aware of the singer (parameter) of the song. With just the musical piece at hand, you try to guess the singer (parameter) who you feel is the most likely (MLE) to have sung that song. Your are making a maximum likelihood estimate! Out of all the singers (parameter space) you have chosen them as the one who is the most likely to have sung that song (data).

We are not going to go into the math of maximum likelihood. It is enough to say that a minimization algorithm is used to optimize the best values for the coefficients for your training data. This is often implemented in practice using efficient numerical optimization algorithm (like the Quasi-newton method).

When you are learning logistic, you can implement it yourself from scratch using the much simpler gradient descent algorithm.

4.1.3 Preparing Data for Logistic Regression

The assumptions made by logistic regression about the distribution and relationships in your data are much the same as the assumptions made in linear regression.

Much study has gone into defining these assumptions and precise probabilistic and statistical language is used. My advice is to use these as guidelines or rules of thumb and experiment with different data preparation schemes.

Ultimately in predictive modeling machine learning projects you are laser focused on making accurate predictions rather than interpreting the results. As such, you can break some assumptions as long as the model is robust and performs well.

- **Binary Output Variable:** This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.
- **Remove Noise:** Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
- **Gaussian Distribution:** Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.
- **Remove Correlated Inputs:** Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
- **Fail to Converge:** It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

4.2 Logistic Regression: Scikit-learn vs Statsmodels

Python gives us two ways to do logistic regression. Statsmodels offers modeling from the perspective of statistics. Scikit-learn offers some of the same models from the perspective of machine learning.

So we need to understand the difference between statistics and machine learning! Statistics makes mathematically valid inferences about a population based on sample data. Statistics answers the question, "What is the evidence that X is related to Y?" Machine learning has the goal of optimizing predictive accuracy rather than inference. Machine learning answers the question, "Given X, what prediction should we make for Y?"

Let us see the use of `statsmodels` for logistic regression. We'll see scikit-learn later in the course, when we learn methods that focus on prediction.

4.3 Training a logistic regression model

Read the data on social network ads. The data shows if the person purchased a product when targeted with an ad on social media. Fit a logistic regression model to predict if a user will purchase the product based on their characteristics such as age, gender and estimated salary.

```
train = pd.read_csv('./Datasets/Social_Network_Ads_train.csv') #Develop the model on train da
test = pd.read_csv('./Datasets/Social_Network_Ads_test.csv') #Test the model on test data
```

```
train.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15755018	Male	36	33000	0
1	15697020	Female	39	61000	0
2	15796351	Male	36	118000	1
3	15665760	Male	39	122000	1
4	15794661	Female	26	118000	0

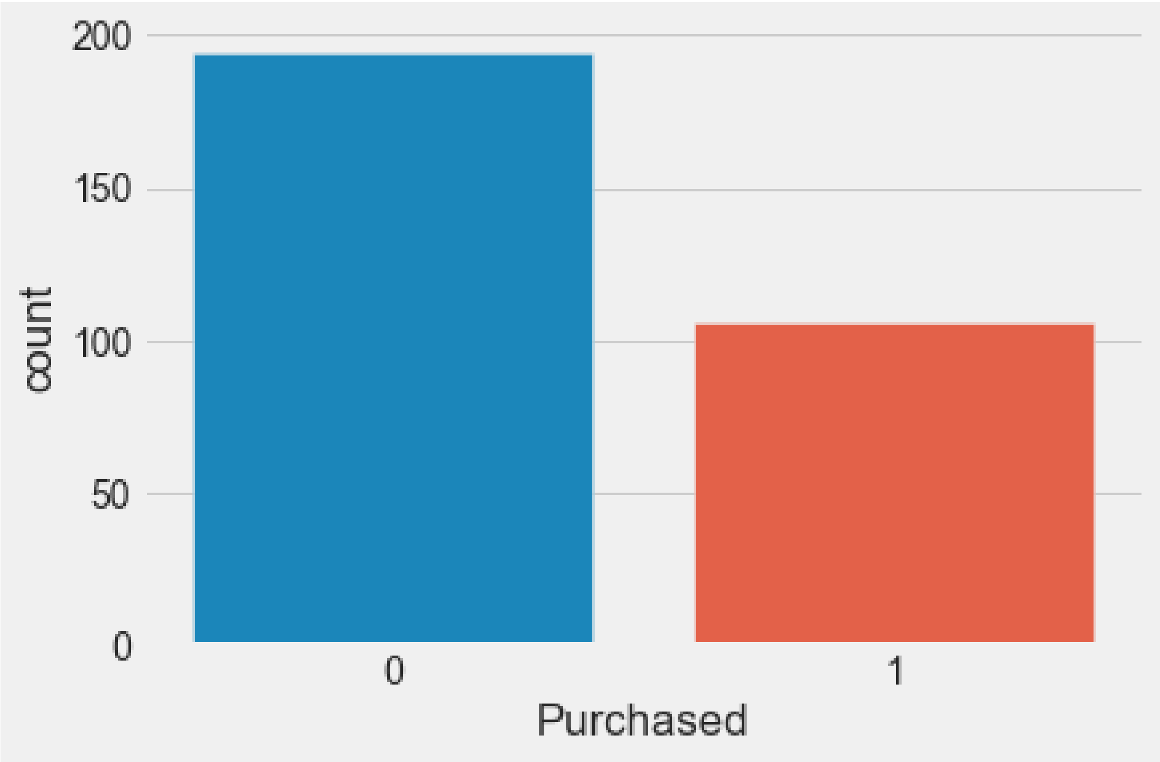
4.3.1 Examining the Distribution of the Target Column

Make sure our target is not severely imbalanced.

```
train.Purchased.value_counts()
```

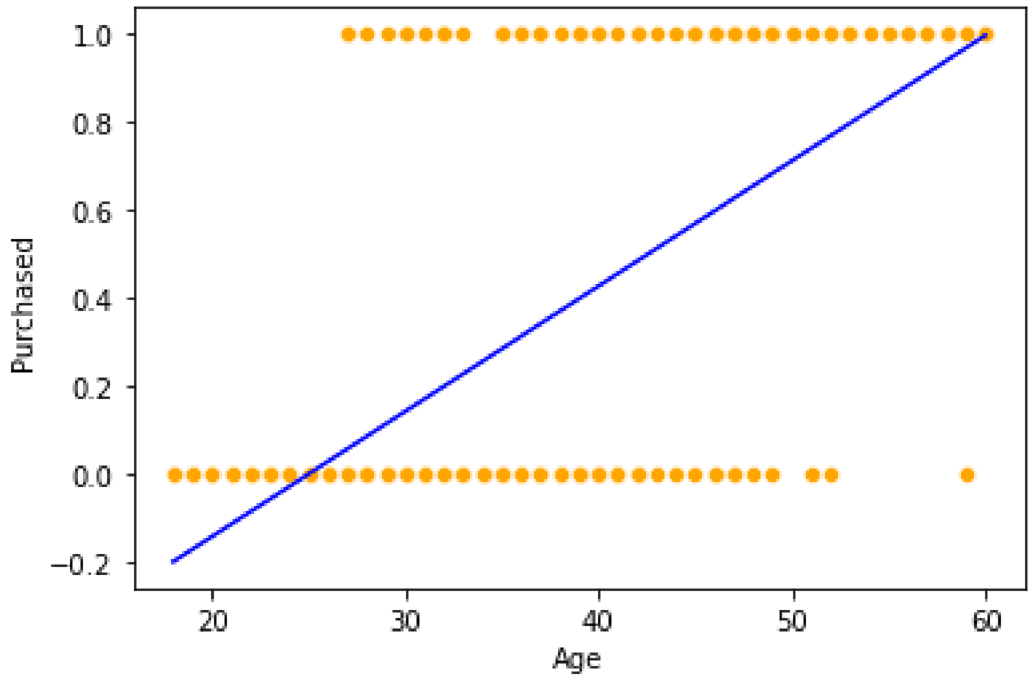
```
0    194
1    106
Name: Purchased, dtype: int64
```

```
sns.countplot(x = 'Purchased',data = train);
```



Let us try to fit a linear regression model, instead of logistic regression. We fit a linear regression model to predict probability of purchase based on age.

```
sns.scatterplot(x = 'Age', y = 'Purchased', data = train, color = 'orange') #Visualizing data
lm = sm.ols(formula = 'Purchased~Age', data = train).fit() #Developing linear regression mode
sns.lineplot(x = 'Age', y= lm.predict(train), data = train, color = 'blue') #Visualizing mode
```



Note the issues with the linear regression model:

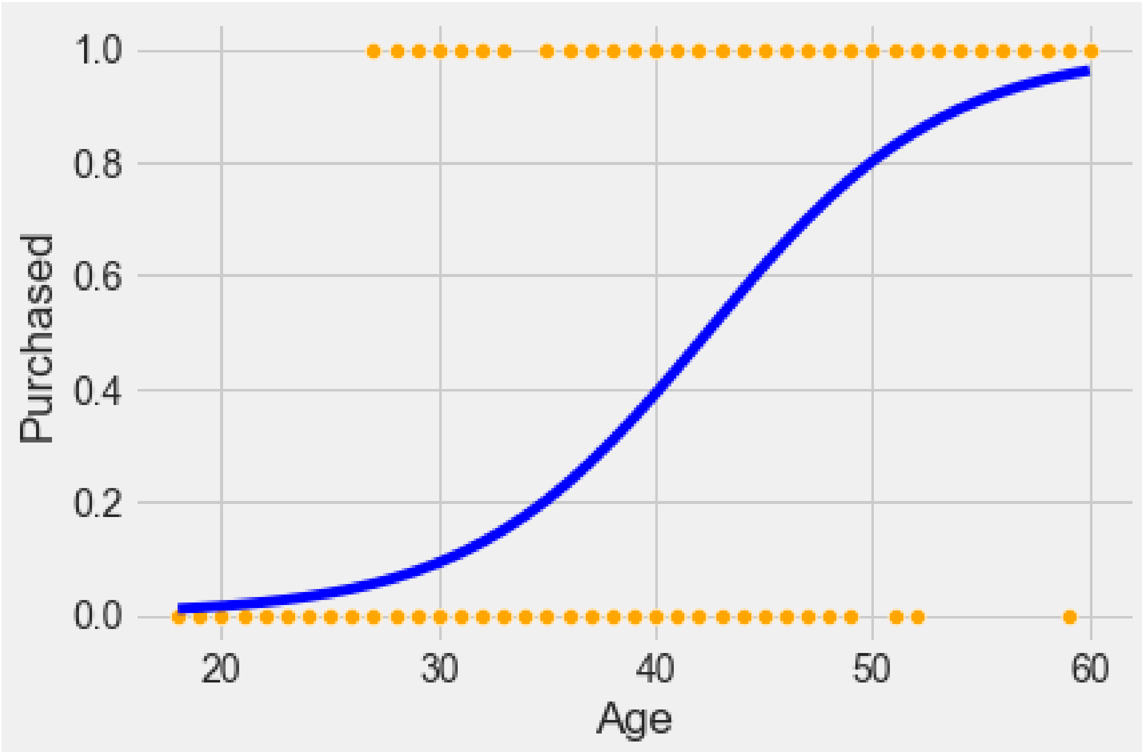
1. The regression line goes below 0 and over 1. However, probability of purchase must be in [0,1].
2. The linear regression model does not seem to fit the data well.

4.3.2 Fitting the logistic regression model

Now, let us fit a logistic regression model to predict probability of purchase based on `Age`.

```
sns.scatterplot(x = 'Age', y = 'Purchased', data = train, color = 'orange') #Visualizing data
logit_model = sm.logit(formula = 'Purchased~Age', data = train).fit() #Developing logistic re
sns.lineplot(x = 'Age', y= logit_model.predict(train), data = train, color = 'blue') #Visuali
```

```
Optimization terminated successfully.
         Current function value: 0.430107
         Iterations 7
```



As logistic regression uses the sigmoid function, the probability stays in [0,1]. Also, it seems to better fit the points as compared to linear regression.

```
logit_model.summary()
```

Table 4.2: Logit Regression Results

Dep. Variable:	Purchased	No. Observations:	300
Model:	Logit	Df Residuals:	298
Method:	MLE	Df Model:	1
Date:	Tue, 19 Apr 2022	Pseudo R-squ.:	0.3378
Time:	16:46:02	Log-Likelihood:	-129.03
converged:	True	LL-Null:	-194.85
Covariance Type:	nonrobust	LLR p-value:	1.805e-30

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-7.8102	0.885	-8.825	0.000	-9.545	-6.076
Age	0.1842	0.022	8.449	0.000	0.141	0.227

Interpret the coefficient of age

For a unit increase in age, the log odds of purchase increase by 0.18, or the odds of purchase get multiplied by exp(0.18) = 1.2

Is the increase in probability of purchase constant with a unit increase in age?

No, it depends on age.

Is gender associated with probability of purchase?

```
logit_model_gender = sm.logit(formula = 'Purchased~Gender', data = train).fit()
logit_model_gender.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.648804
         Iterations 4
```

Table 4.4: Logit Regression Results

Dep. Variable:	Purchased	No. Observations:	300
Model:	Logit	Df Residuals:	298
Method:	MLE	Df Model:	1
Date:	Tue, 19 Apr 2022	Pseudo R-squ.:	0.001049
Time:	16:46:04	Log-Likelihood:	-194.64
converged:	True	LL-Null:	-194.85
Covariance Type:	nonrobust	LLR p-value:	0.5225

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.5285	0.168	-3.137	0.002	-0.859	-0.198
Gender[T.Male]	-0.1546	0.242	-0.639	0.523	-0.629	0.319

No, assuming a significance level of $\alpha = 5\%$, `Gender` is not associated with probability of default, as the p-value for `Male` is greater than 0.05.

4.4 Confusion matrix and classification accuracy

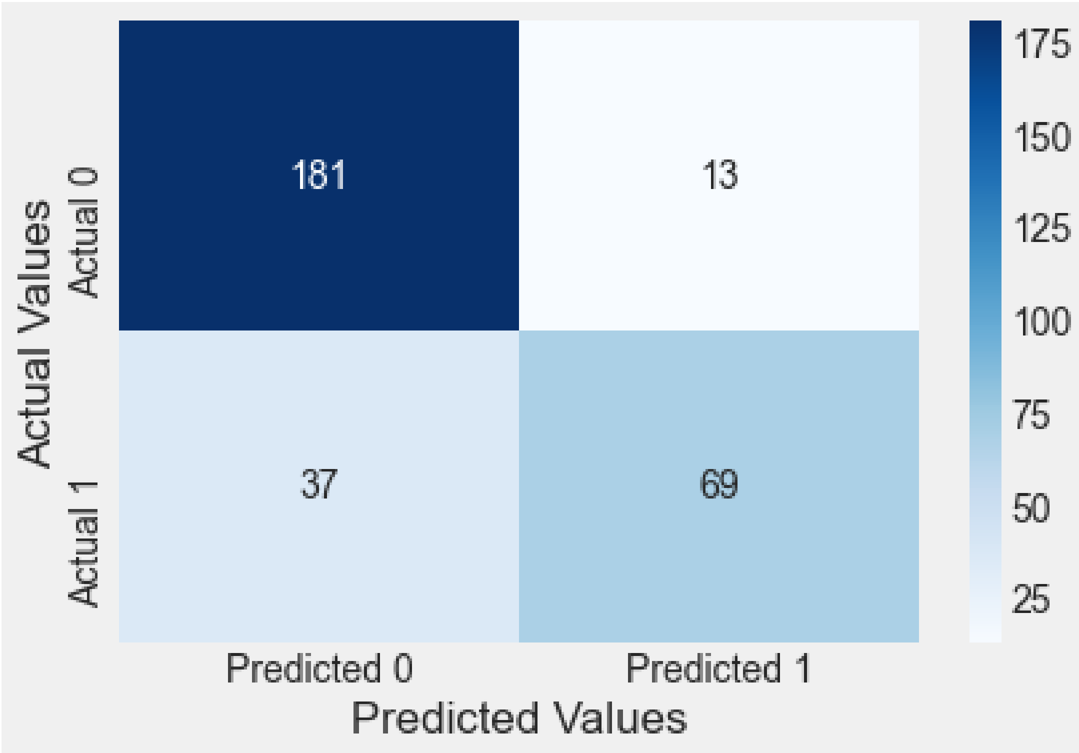
A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class.

```
#Function to compute confusion matrix and prediction accuracy on training data
def confusion_matrix_train(model,cutoff=0.5):
    # Confusion matrix
    cm_df = pd.DataFrame(model.pred_table(threshold = cutoff))
    #Formatting the confusion matrix
    cm_df.columns = ['Predicted 0', 'Predicted 1']
    cm_df = cm_df.rename(index={0: 'Actual 0',1: 'Actual 1'})
    cm = np.array(cm_df)
    # Calculate the accuracy
    accuracy = (cm[0,0]+cm[1,1])/cm.sum()
    sns.heatmap(cm_df, annot=True, cmap='Blues', fmt='g')
    plt.ylabel("Actual Values")
    plt.xlabel("Predicted Values")
    print("Classification accuracy = {:.1%}".format(accuracy))
```

Find the confusion matrix and classification accuracy of the model with `Age` as the predictor on training data.

```
cm = confusion_matrix_train(logit_model)
```

```
Classification accuracy = 83.3%
```



Confusion matrix:

- Each row: actual class
- Each column: predicted class

First row: Non-purchasers, the negative class:

- 181 were correctly classified as Non-purchasers. **True negatives**.
- Remaining 13 were wrongly classified as Non-purchasers. **False positive**

Second row: Purchasers, the positive class:

- 37 were incorrectly classified as Non-purchasers. **False negatives**
- 69 were correctly classified Purchasers. **True positives**

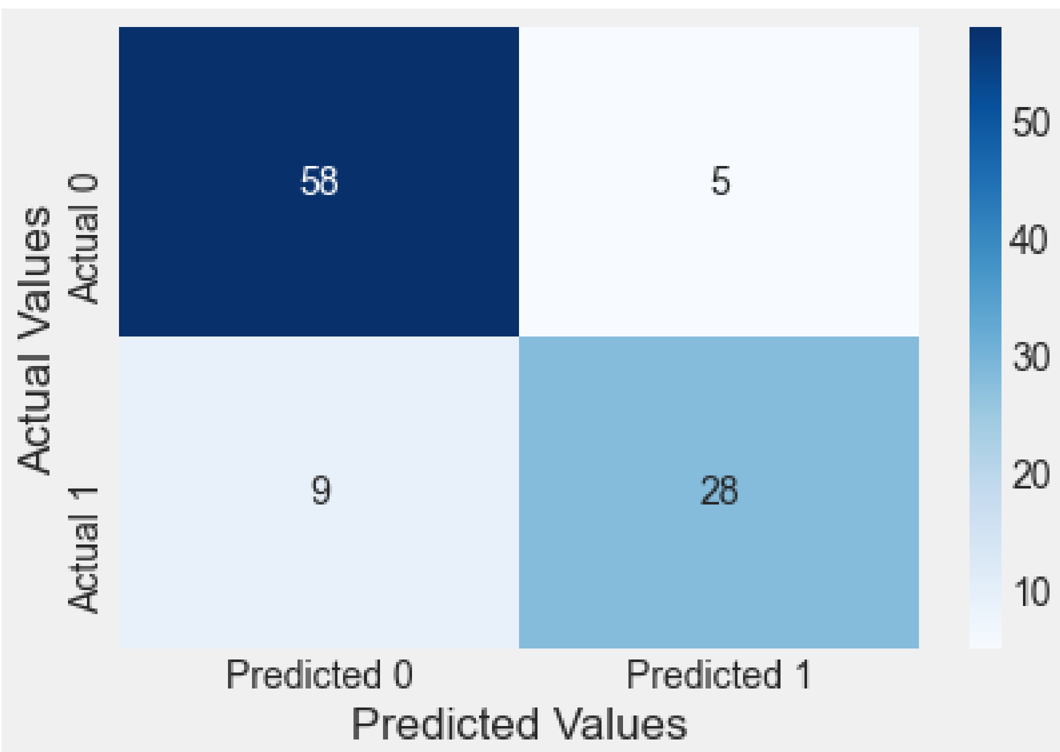
```
#Function to compute confusion matrix and prediction accuracy on test data
def confusion_matrix_test(data,actual_values,model,cutoff=0.5):
#Predict the values using the Logit model
    pred_values = model.predict(data)
# Specify the bins
    bins=np.array([0,cutoff,1])
#Confusion matrix
```

```
    cm = np.histogram2d(actual_values, pred_values, bins=bins)[0]
    cm_df = pd.DataFrame(cm)
    cm_df.columns = ['Predicted 0','Predicted 1']
    cm_df = cm_df.rename(index={0: 'Actual 0',1:'Actual 1'})
    accuracy = (cm[0,0]+cm[1,1])/cm.sum()
    sns.heatmap(cm_df, annot=True, cmap='Blues', fmt='g')
    plt.ylabel("Actual Values")
    plt.xlabel("Predicted Values")
    print("Classification accuracy = {:.1%}".format(accuracy))
```

Find the confusion matrix and classification accuracy of the model with `Age` as the predictor on test data.

```
confusion_matrix_test(test,test.Purchased,logit_model)
```

```
Classification accuracy = 86.0%
```



The model classifies a bit more accurately on test data as compared to the training data, which is a bit unusual. However, it shows that the model did not overfit on training data.

Include `EstimatedSalary` as a predictor in the above model

```
logit_model2 = sm.logit(formula = 'Purchased~Age+EstimatedSalary', data = train).fit()
logit_model2.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.358910
         Iterations 7
```

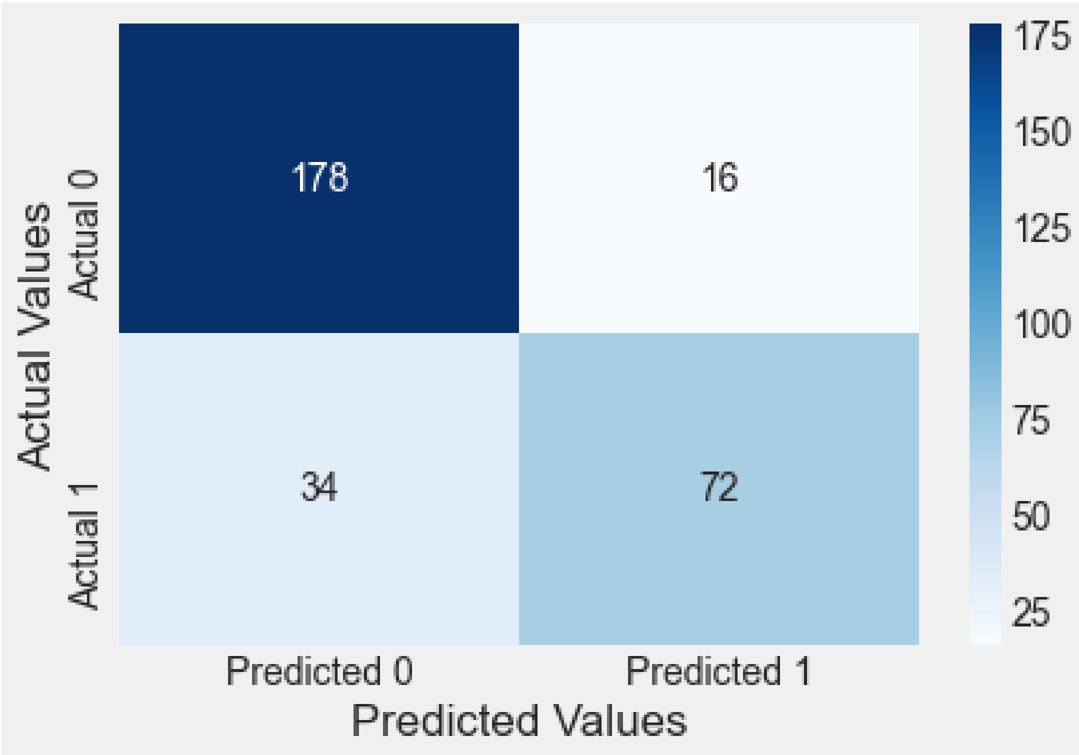
Table 4.6: Logit Regression Results

Dep. Variable:	Purchased	No. Observations:	300
Model:	Logit	Df Residuals:	297
Method:	MLE	Df Model:	2
Date:	Tue, 14 Feb 2023	Pseudo R-squ.:	0.4474
Time:	12:03:29	Log-Likelihood:	-107.67
converged:	True	LL-Null:	-194.85
Covariance Type:	nonrobust	LLR p-value:	1.385e-38

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-11.9432	1.424	-8.386	0.000	-14.735	-9.152
Age	0.2242	0.028	7.890	0.000	0.168	0.280
EstimatedSalary	3.48e-05	6.15e-06	5.660	0.000	2.27e-05	4.68e-05

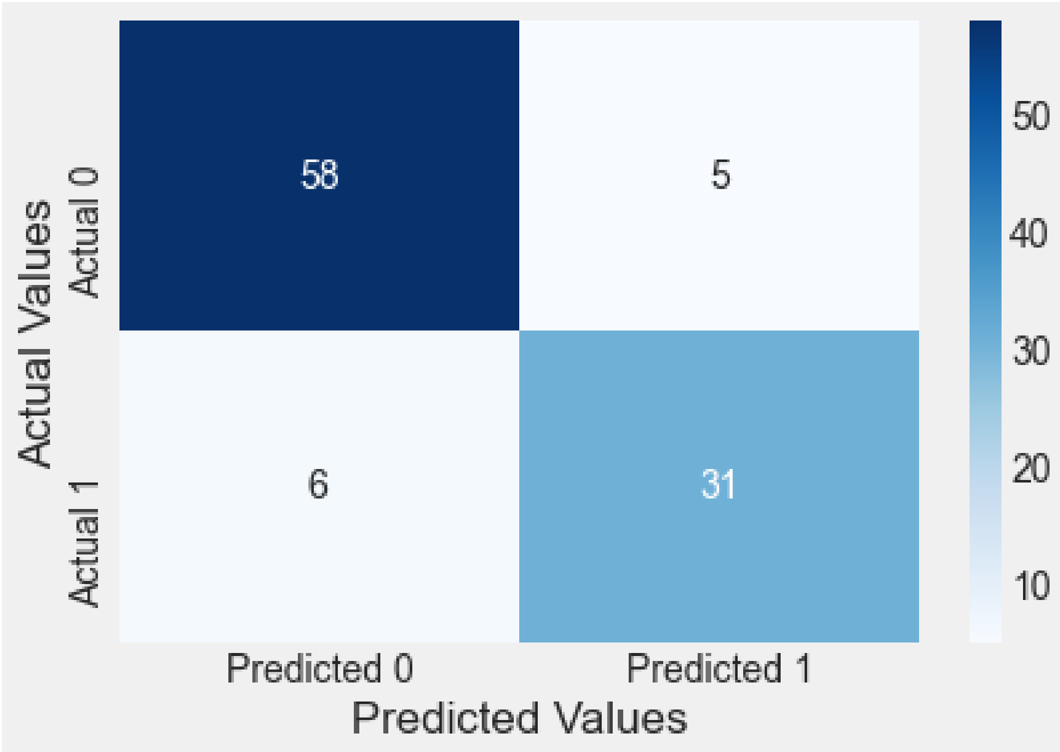
```
confusion_matrix_train(logit_model2)
```

```
Classification accuracy = 83.3%
```




```
confusion_matrix_test(test,test.Purchased,logit_model2)
```

```
Classification accuracy = 89.0%
```



The log likelihood of the model has increased, while also increasing the prediction accuracy on test data, which shows that the additional predictor is helping explain the response better, without overfitting the data.

Include `Gender` as a predictor in the above model

```
logit_model = sm.logit(formula = 'Purchased~Age+EstimatedSalary+Gender', data = train).fit()
logit_model.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.357327
         Iterations 7
```

Table 4.8: Logit Regression Results

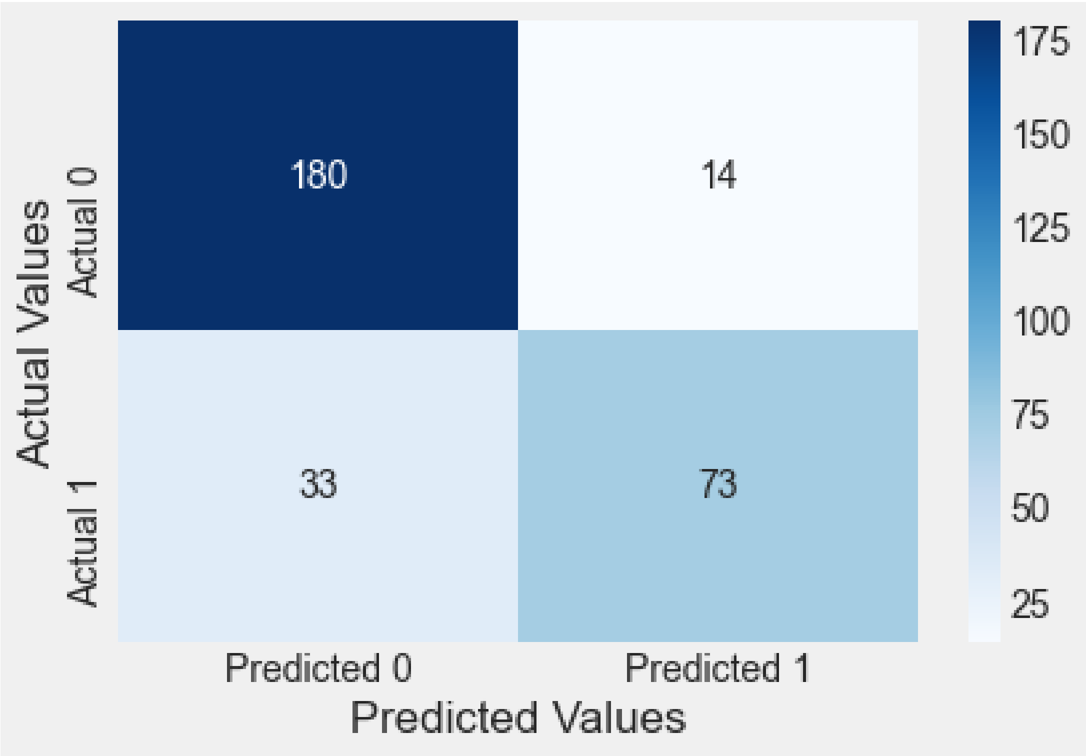
Dep. Variable:	Purchased	No. Observations:	300
Model:	Logit	Df Residuals:	296
Method:	MLE	Df Model:	3
Date:	Tue, 14 Feb 2023	Pseudo R-squ.:	0.4498
Time:	12:17:28	Log-Likelihood:	-107.20

converged:	True	LL-Null:	-194.85
Covariance Type:	nonrobust	LLR p-value:	9.150e-38

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-12.2531	1.478	-8.293	0.000	-15.149	-9.357
Gender[T.Male]	0.3356	0.346	0.970	0.332	-0.342	1.013
Age	0.2275	0.029	7.888	0.000	0.171	0.284
EstimatedSalary	3.494e-05	6.17e-06	5.666	0.000	2.29e-05	4.7e-05

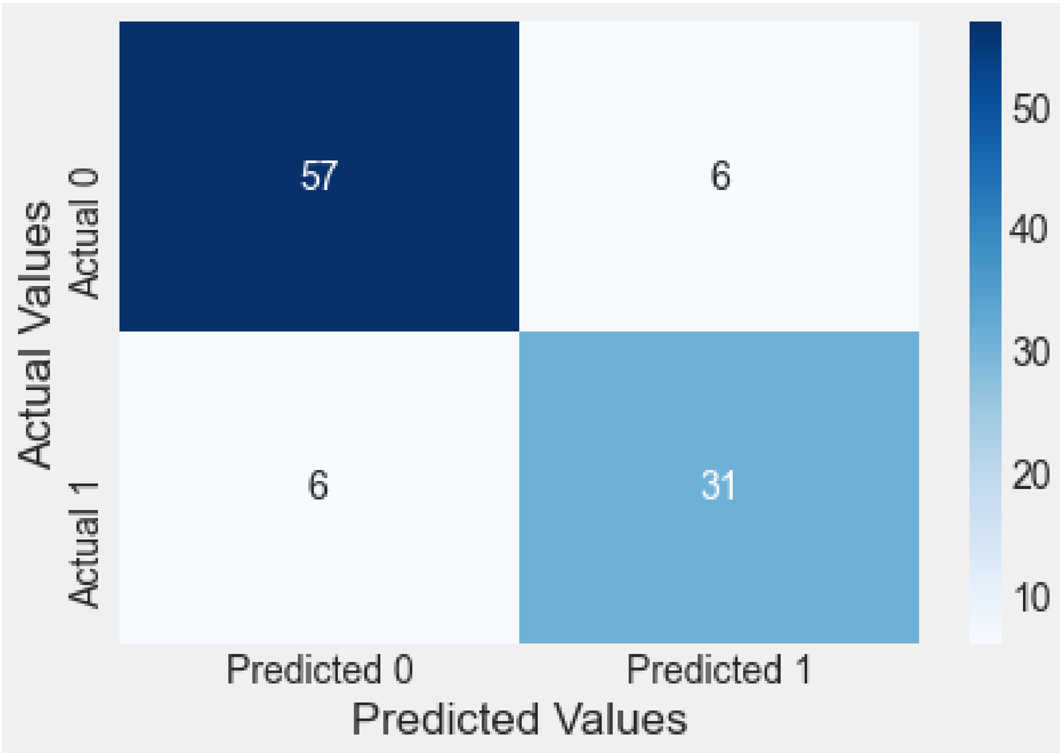
```
confusion_matrix_train(logit_model)
```

```
Classification accuracy = 84.3%
```




```
confusion_matrix_test(test,test.Purchased,logit_model)
```

```
Classification accuracy = 88.0%
```



`Gender` is a statistically insignificant predictor, and including it slightly lowers the classification accuracy on test data. Note that the classification accuracy on training data will continue to increase on adding more predictors, irrespective of their relevance *(similar to the idea of RSS on training data in linear regression).*

Is there a residual in logistic regression?

No, since the response is assumed to have a Bernoulli distribution, instead of a normal distribution.

Is the odds ratio for a unit increase in a predictor X_j, a constant (assuming that the rest of the predictors are held constant)?

Yes, the odds ratio in this case will e^{β_j}

4.5 Variable transformations in logistic regression

Read the dataset *diabetes.csv* that contains if a person has diabetes (`Outcome = 1`) based on health parameters such as BMI, blood pressure, age etc. Develop a model to predict the probability of a person having diabetes based on their age.

```
data = pd.read_csv('./Datasets/diabetes.csv')
```

```
data.head()
```

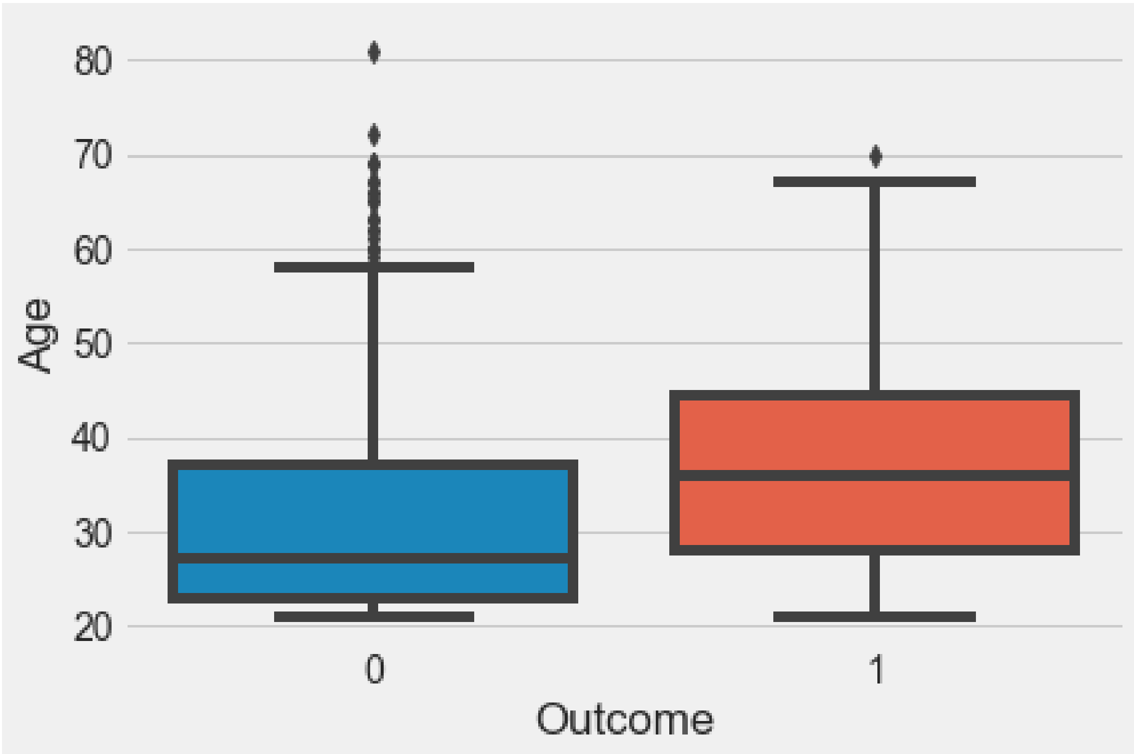
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

Randomly select 80% of the observations to create a training dataset. Create a test dataset with the remaining 20% observations.

```
#Creating training and test datasets
np.random.seed(2)
train = data.sample(round(data.shape[0]*0.8))
test = data.drop(train.index)
```

Does `Age` seem to distinguish `Outcome` levels?

```
sns.boxplot(x = 'Outcome', y = 'Age', data = train)
```

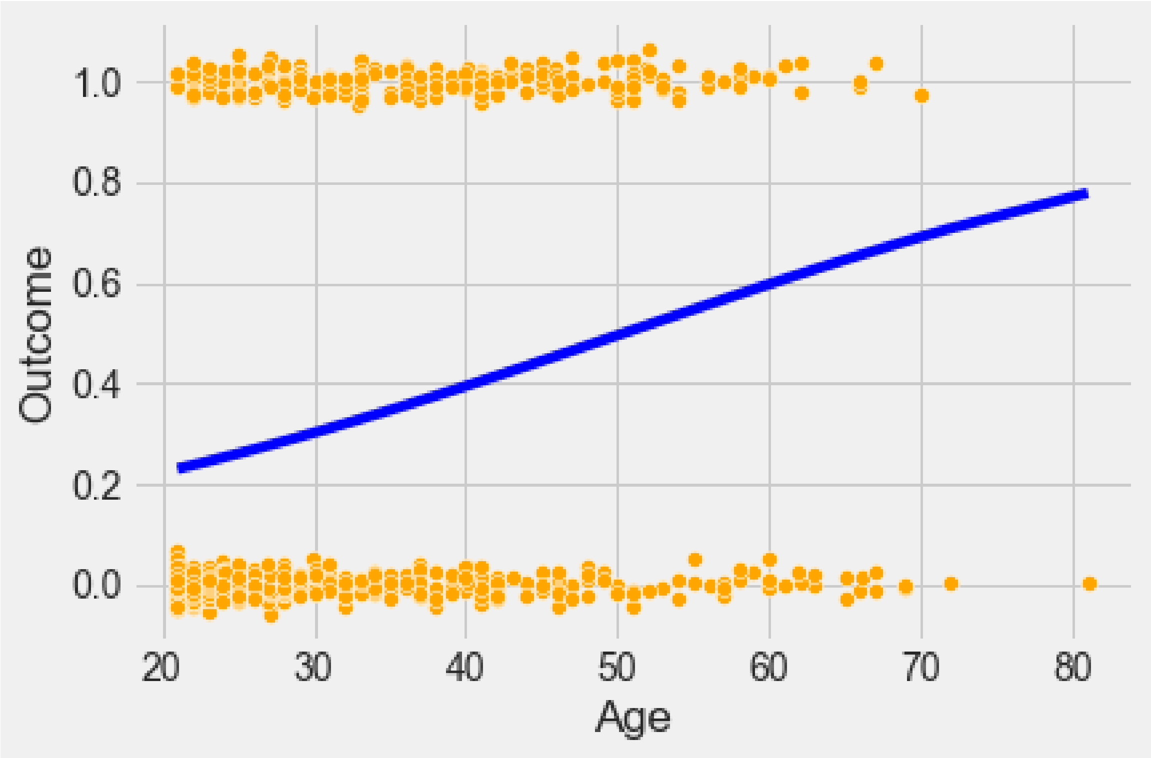


Yes it does!

Develop and visualize a logistic regression model to predict `Outcome` using `Age`.

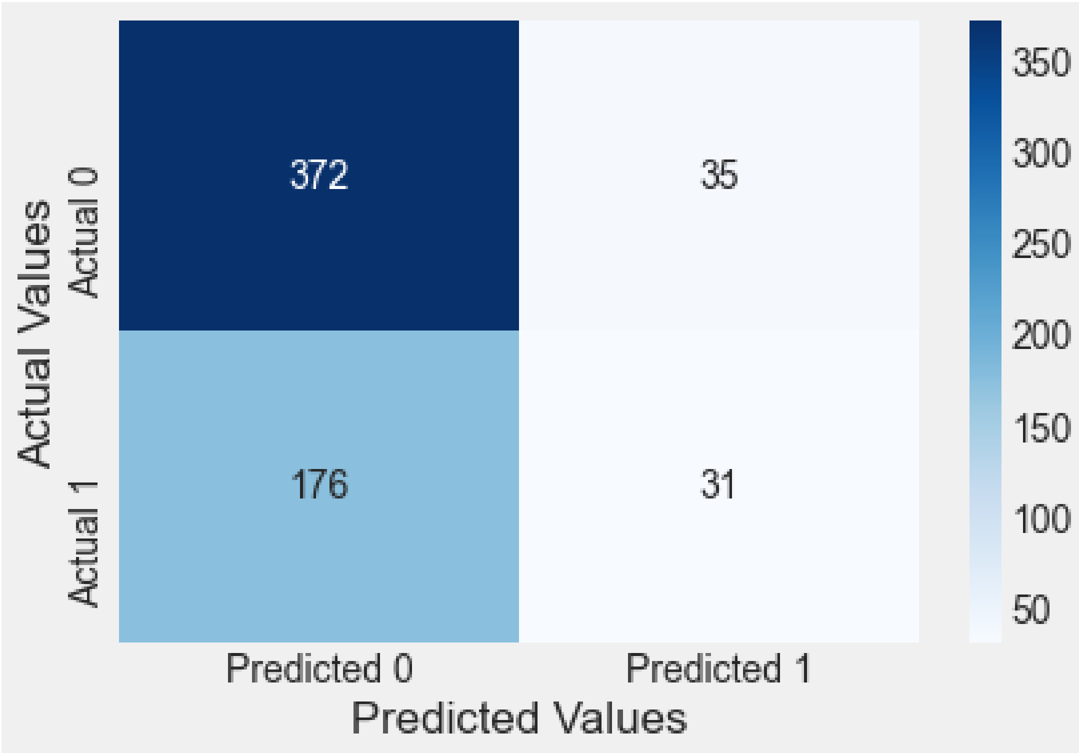
```
#Jittering points to better see the density of points in any given region of the plot
def jitter(values,j):
    return values + np.random.normal(j,0.02,values.shape)
sns.scatterplot(x = jitter(train.Age,0), y = jitter(train.Outcome,0), data = train, color =
logit_model = sm.logit(formula = 'Outcome~Age', data = train).fit()
sns.lineplot(x = 'Age', y= logit_model.predict(train), data = train, color = 'blue')
print(logit_model.llf) #Printing the log likelihood to compare it with the next model we buil
```

```
Optimization terminated successfully.
         Current function value: 0.612356
         Iterations 5
-375.9863802089716
```




```
confusion_matrix_train(logit_model)
```

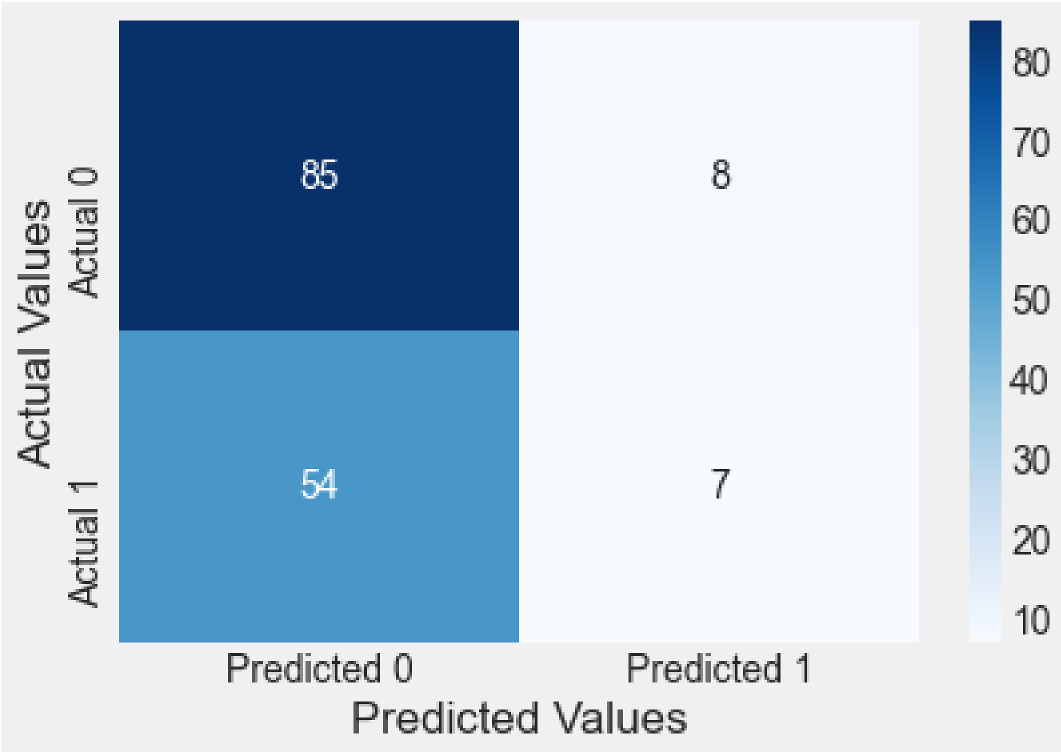
```
Classification accuracy = 65.6%
```



Classification accuracy on train data = 66%

```
confusion_matrix_test(test,test.Outcome,logit_model)
```

```
Classification accuracy = 59.7%
```



Classification accuracy on test data = 60%

Can a tranformation of `Age` provide a more accurate model?

Let us visualize how the probability of people having diabetes varies with `Age`. We will bin `Age` to get the percentage of people having diabetes within different `Age` bins.

```
#Binning Age
binned_age = pd.qcut(train['Age'],11,retbins=True)
train['age_binned'] = binned_age[0]
```

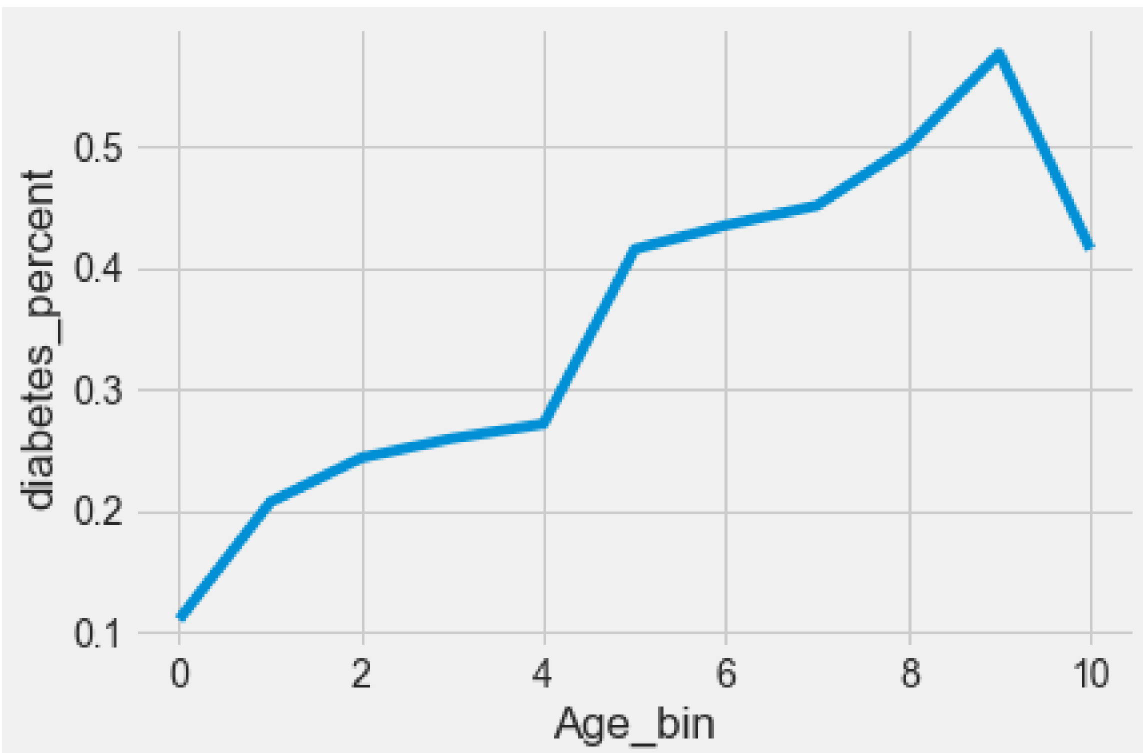
```
#Finding percentage of people having diabetes in each Age bin
age_data = train.groupby('age_binned')['Outcome'].agg([('diabetes_percent','mean'),('nobs','c
age_data
```

	age_binned	diabetes_percent	nobs
0	(20.999, 22.0]	0.110092	109
1	(22.0, 23.0]	0.206897	29
2	(23.0, 25.0]	0.243243	74
3	(25.0, 26.0]	0.259259	27

	age_binned	diabetes_percent	nobs
4	(26.0, 28.0]	0.271186	59
5	(28.0, 31.0]	0.415094	53
6	(31.0, 35.0]	0.434783	46
7	(35.0, 39.0]	0.450980	51
8	(39.0, 43.545]	0.500000	54
9	(43.545, 52.0]	0.576271	59
10	(52.0, 81.0]	0.415094	53

```
#Visualizing percentage of people having diabetes with increasing Age (or Age bins)
sns.lineplot(x = age_data.index, y= age_data['diabetes_percent'])
plt.xlabel('Age_bin')
```

```
Text(0.5, 0, 'Age_bin')
```



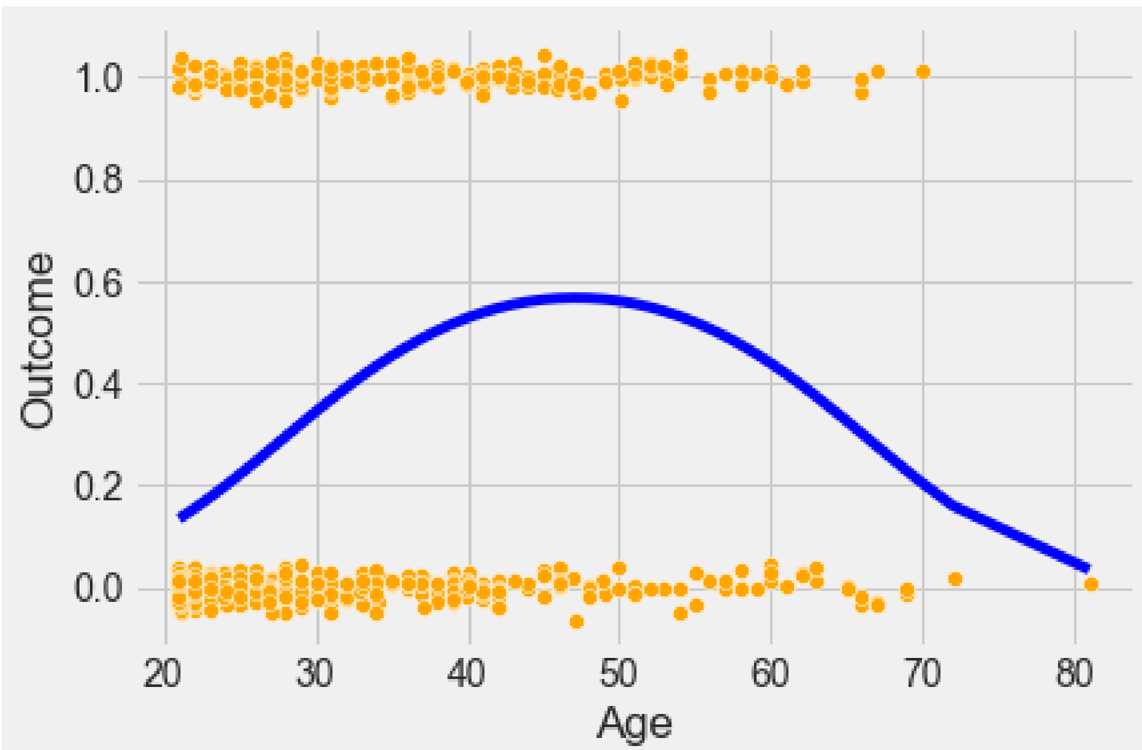
We observe that the probability of people having diabetes does **not** keep increasing monotonically with age. People with ages 52 and more have a lower probability of having diabetes than people in the immediately younger `Age` bin.

A quadratic transformation of `Age` may better fit the above trend

```
#Model with the quadratic transformation of Age
def jitter(values,j):
    return values + np.random.normal(j,0.02,values.shape)
sns.scatterplot(x = jitter(train.Age,0), y = jitter(train.Outcome,0), data = train, color =
logit_model = sm.logit(formula = 'Outcome~Age+I(Age**2)', data = train).fit()
sns.lineplot(x = 'Age', y= logit_model.predict(train), data = train, color = 'blue')
logit_model.llf
```

```
Optimization terminated successfully.
         Current function value: 0.586025
         Iterations 6
```

```
-359.81925590230185
```




```
logit_model.summary()
```

Table 4.12: Logit Regression Results

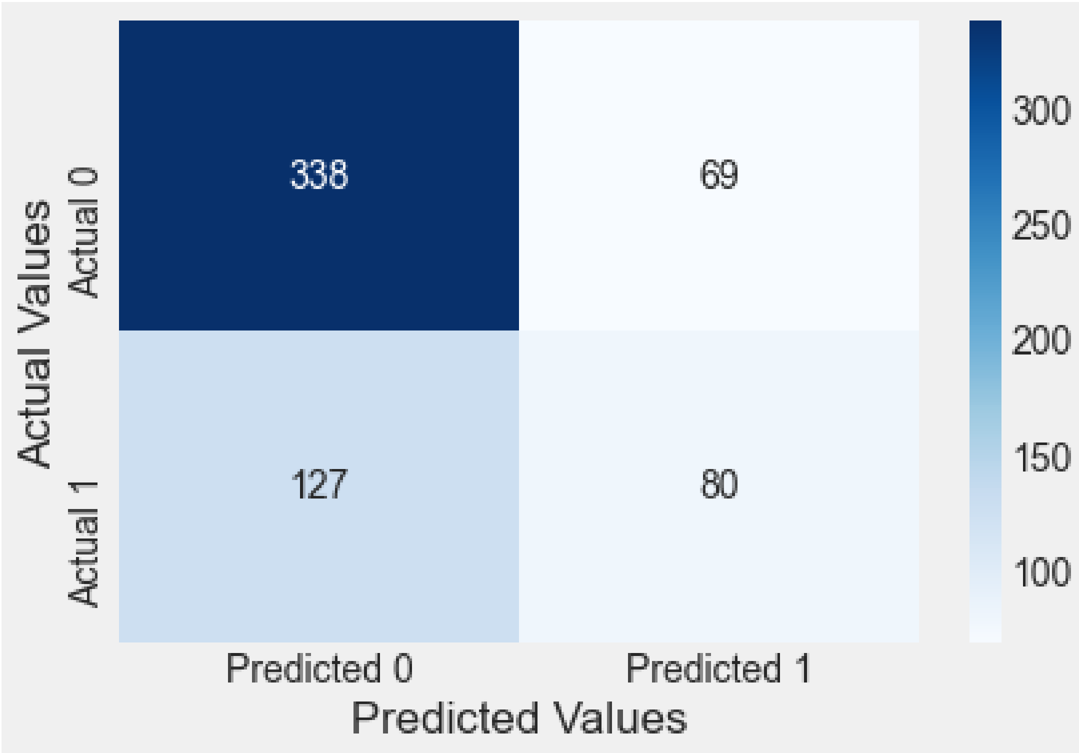
Dep. Variable:	Outcome	No. Observations:	614
Model:	Logit	Df Residuals:	611
Method:	MLE	Df Model:	2
Date:	Tue, 14 Feb 2023	Pseudo R-squ.:	0.08307
Time:	12:25:54	Log-Likelihood:	-359.82
converged:	True	LL-Null:	-392.42
Covariance Type:	nonrobust	LLR p-value:	6.965e-15

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-6.6485	0.908	-7.320	0.000	-8.429	-4.868
Age	0.2936	0.048	6.101	0.000	0.199	0.388
I(Age ** 2)	-0.0031	0.001	-5.280	0.000	-0.004	-0.002

The log likelihood of the model is higher and both the predictors are statistically significant indicating a better model fit. However, the model may also be overfitting. Let us check the model accuracy on test data.

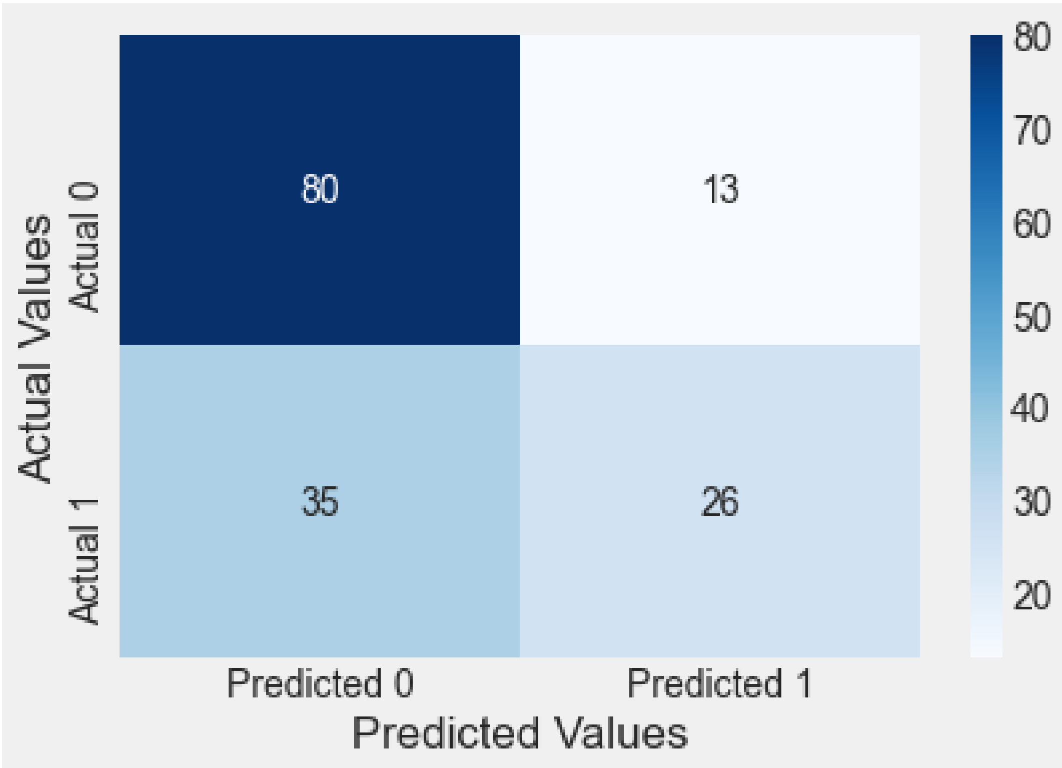
```
confusion_matrix_train(logit_model)
```

```
Classification accuracy = 68.1%
```




```
confusion_matrix_test(test,test.Outcome,logit_model)
```

```
Classification accuracy = 68.8%
```

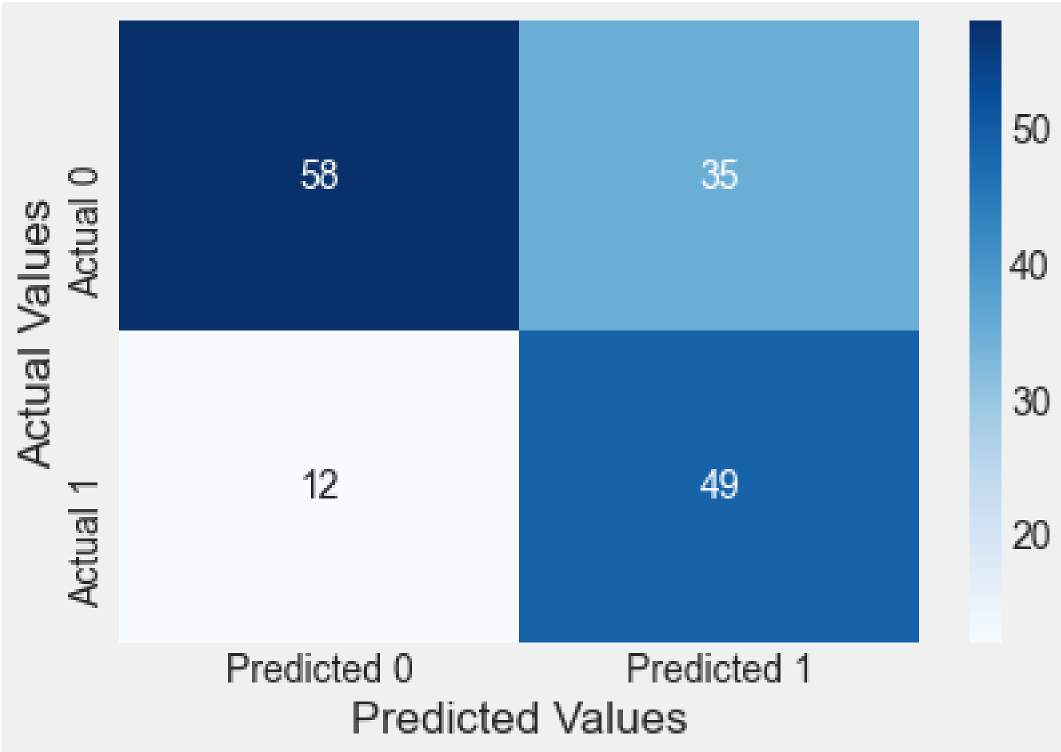


The classification accuracy on test data has increased to 69%. However, the number of *false positives* have increased. But in case of diabetes, *false negatives* are more concerning than *false positives*. This is because if a person has diabetes, and is told that they do not have diabetes, their condition may deteriorate. If a person does not have diabetes, and is told that they have diabetes, they may take unnecessary precautions or tests, but it will not be as harmful to the person as in the previous case. So, in this problem, we will be more focused on reducing the number of *false negatives*, instead of reducing the *false positives* or increasing the overall classification accuracy.

We can decrease the cutoff for classifying a person as having diabetes to reduce the number of false negatives.

```
#Reducing the cutoff for classifying a person as diabetic to 0.3 (instead of 0.5)
confusion_matrix_test(test,test.Outcome,logit_model,0.3)
```

```
Classification accuracy = 69.5%
```



Note that the changed cut-off reduced the number of *false negatives*, but at the cost of increasing the *false positives*. However, the stakeholders may prefer the reduced cut-off to be safer.

Is there another way to transform `Age`?

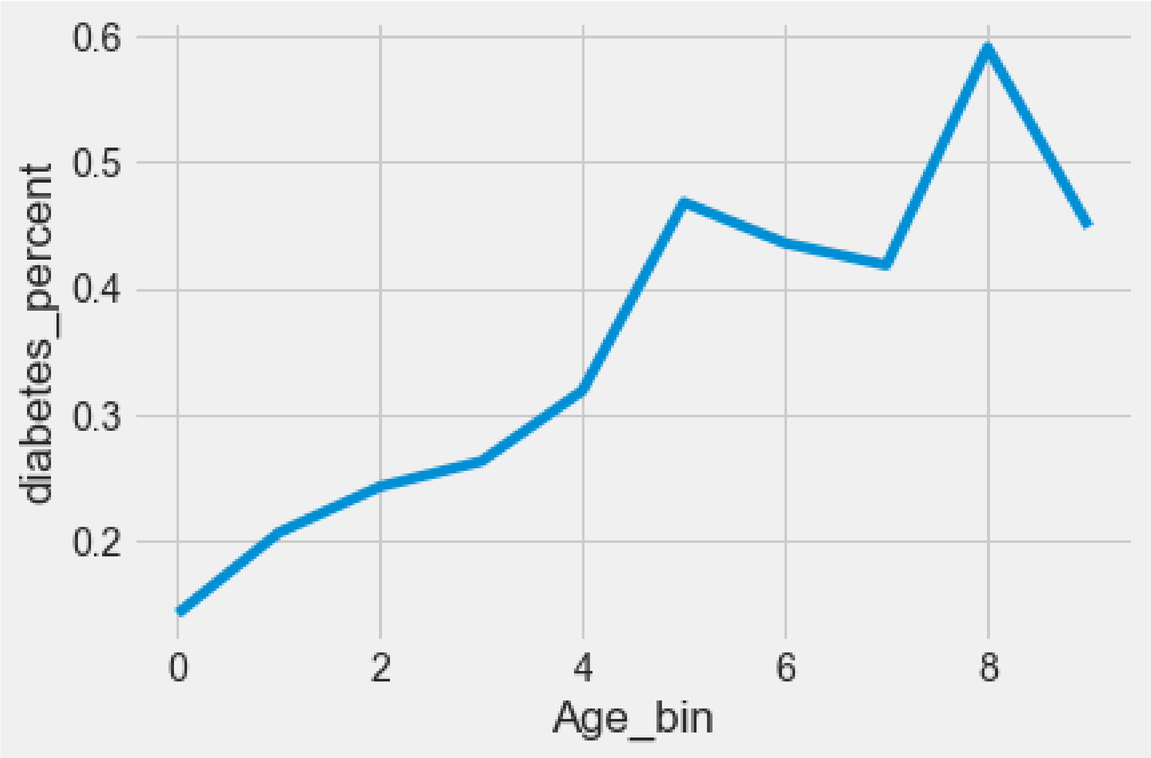
Yes, binning age into bins that have similar proportion of people with diabetes may provide a better model fit.

```
#Creating a function to bin age so that it can be applied to both the test and train datasets
def var_transform(data):
    binned_age = pd.qcut(train['Age'],10,retbins=True)
    bins = binned_age[1]
    data['age_binned'] = pd.cut(data['Age'],bins = bins)
    dum = pd.get_dummies(data.age_binned,drop_first = True)
    dum.columns = ['age'+str(x) for x in range(1,len(bins)-1)]
    data = pd.concat([data,dum], axis = 1)
    return data
```

```
#Binning age using the function var_transform()
train = var_transform(train)
test = var_transform(test)
```

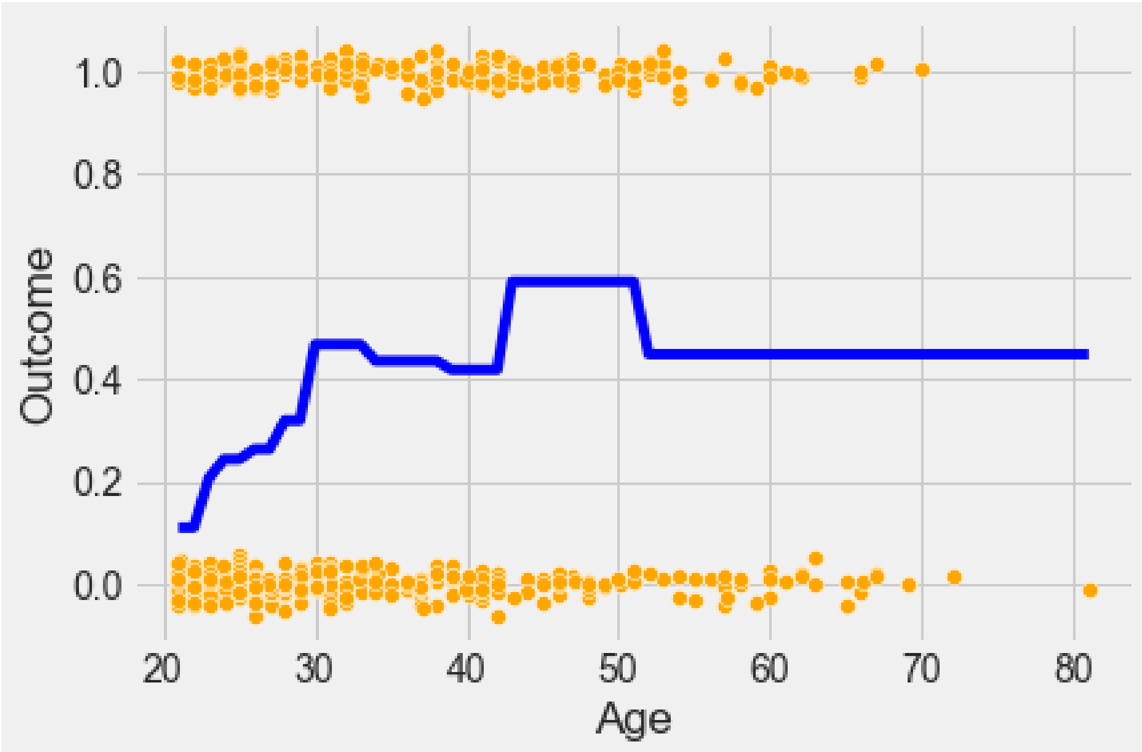
```
#Re-creating the plot of diabetes_percent vs age created earlier, just to check if the functi
age_data = train.groupby('age_binned')['Outcome'].agg([('diabetes_percent','mean'),('nobs','c
sns.lineplot(x = age_data.index, y= age_data['diabetes_percent'])
plt.xlabel('Age_bin')
```

```
Text(0.5, 0, 'Age_bin')
```




```
#Model with binned Age
def jitter(values,j):
    return values + np.random.normal(j,0.02,values.shape)
sns.scatterplot(x = jitter(train.Age,0), y = jitter(train.Outcome,0), data = train, color =
logit_model = sm.logit(formula = 'Outcome~' + '+'.join(['age'+str(x) for x in range(1,10)]),
sns.lineplot(x = 'Age', y= logit_model.predict(train), data = train, color = 'blue')
```

```
Optimization terminated successfully.
         Current function value: 0.585956
         Iterations 6
```




```
logit_model.summary()
```

Table 4.14: Logit Regression Results

Dep. Variable:	Outcome	No. Observations:	614
Model:	Logit	Df Residuals:	604
Method:	MLE	Df Model:	9
Date:	Sun, 19 Feb 2023	Pseudo R-squ.:	0.08318
Time:	14:19:51	Log-Likelihood:	-359.78
converged:	True	LL-Null:	-392.42
Covariance Type:	nonrobust	LLR p-value:	1.273e-10

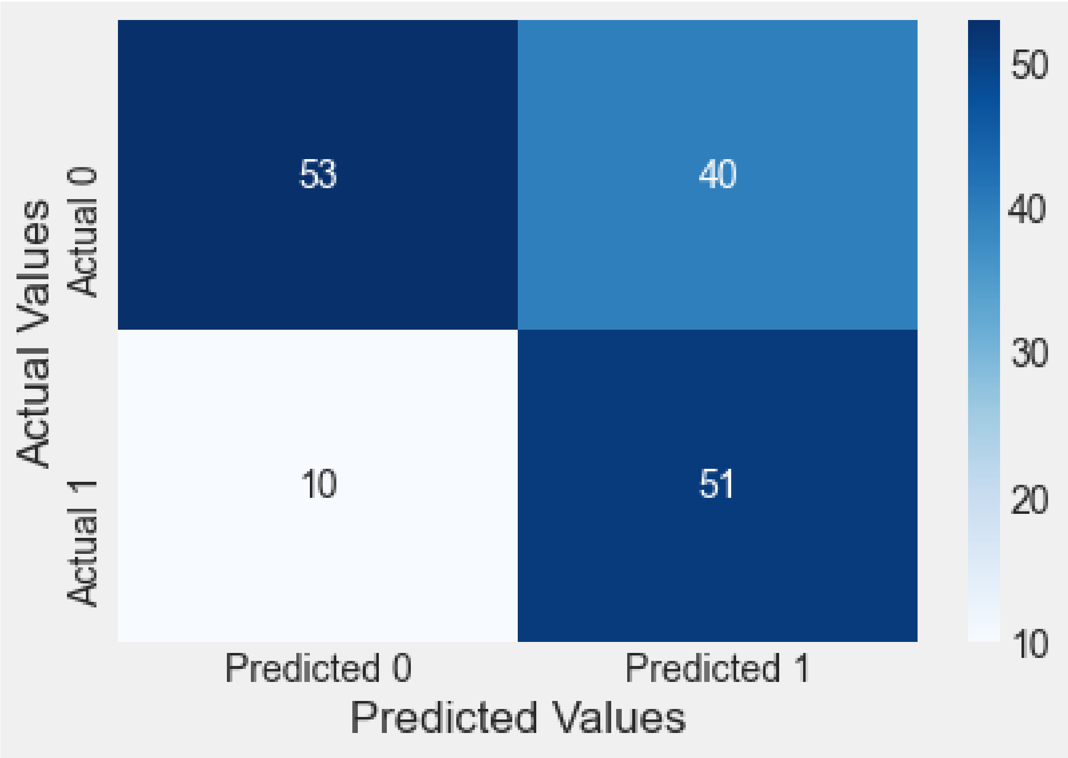
	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-2.0898	0.306	-6.829	0.000	-2.690	-1.490
age1	0.7461	0.551	1.354	0.176	-0.334	1.826
age2	0.9548	0.409	2.336	0.019	0.154	1.756
age3	1.0602	0.429	2.471	0.013	0.219	1.901
age4	1.3321	0.438	3.044	0.002	0.474	2.190

age5	1.9606	0.398	4.926	0.000	1.180	2.741
age6	1.8303	0.399	4.586	0.000	1.048	2.612
age7	1.7596	0.410	4.288	0.000	0.955	2.564
age8	2.4544	0.402	6.109	0.000	1.667	3.242
age9	1.8822	0.404	4.657	0.000	1.090	2.674

Note that the probability of having diabetes for each age bin is a constant, as per the above plot.

```
confusion_matrix_test(test,test.Outcome,logit_model,0.3)
```

```
Classification accuracy = 67.5%
```



Binning `Age` provides a similar result as compared to the model with the quadratic transformation of `Age`.

```
train.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
158	2	88	74	19	53	29.0	0.229	22
251	2	129	84	0	0	28.0	0.284	27
631	0	102	78	40	90	34.5	0.238	24
757	0	123	72	0	0	36.3	0.258	52
689	1	144	82	46	180	46.1	0.335	46

```
#Model with the quadratic transformation of Age and more predictors
logit_model_diabetes = sm.logit(formula = 'Outcome~Age+I(Age**2)+Glucose+BloodPressure+BMI+Di
logit_model_diabetes.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.470478
         Iterations 6
```

Table 4.17: Logit Regression Results

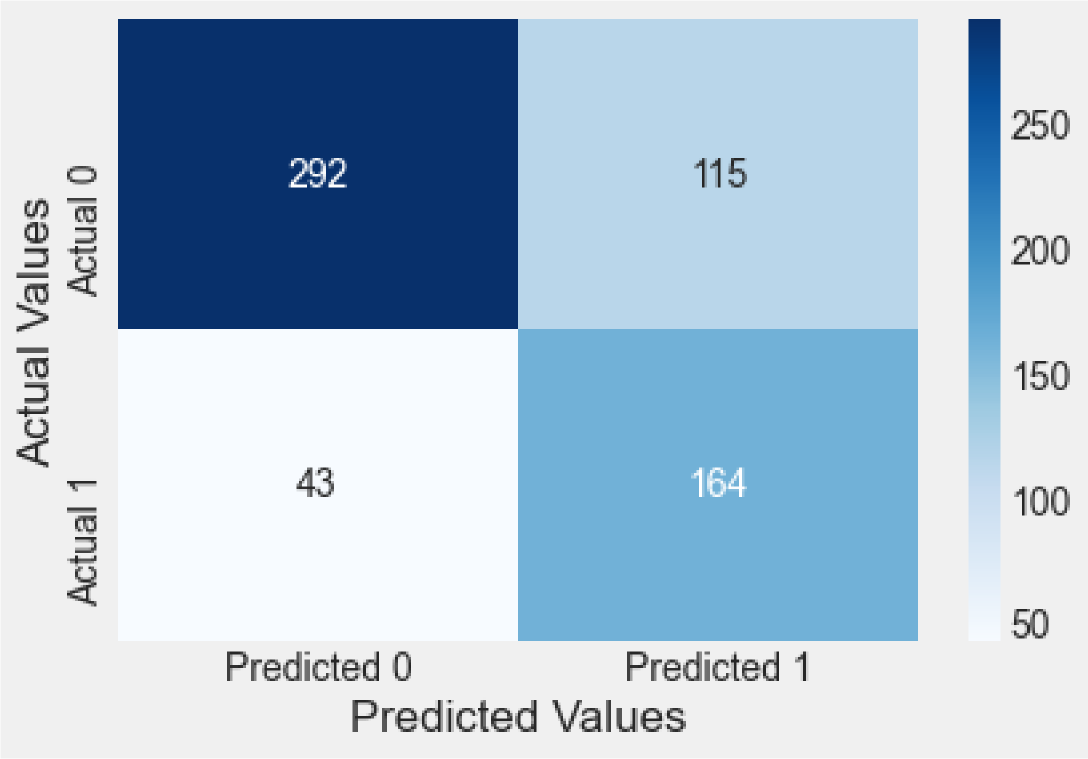
Dep. Variable:	Outcome	No. Observations:	614
Model:	Logit	Df Residuals:	607
Method:	MLE	Df Model:	6
Date:	Thu, 23 Feb 2023	Pseudo R-squ.:	0.2639
Time:	10:26:00	Log-Likelihood:	-288.87
converged:	True	LL-Null:	-392.42
Covariance Type:	nonrobust	LLR p-value:	5.878e-42

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-12.3347	1.282	-9.621	0.000	-14.847	-9.822
Age	0.2852	0.056	5.121	0.000	0.176	0.394
I(Age ** 2)	-0.0030	0.001	-4.453	0.000	-0.004	-0.002
Glucose	0.0309	0.004	8.199	0.000	0.024	0.038
BloodPressure	-0.0141	0.006	-2.426	0.015	-0.025	-0.003
BMI	0.0800	0.016	4.978	0.000	0.049	0.112
DiabetesPedigreeFunction	0.7138	0.322	2.213	0.027	0.082	1.346

Adding more predictors has increased the log likelihood of the model as expected.

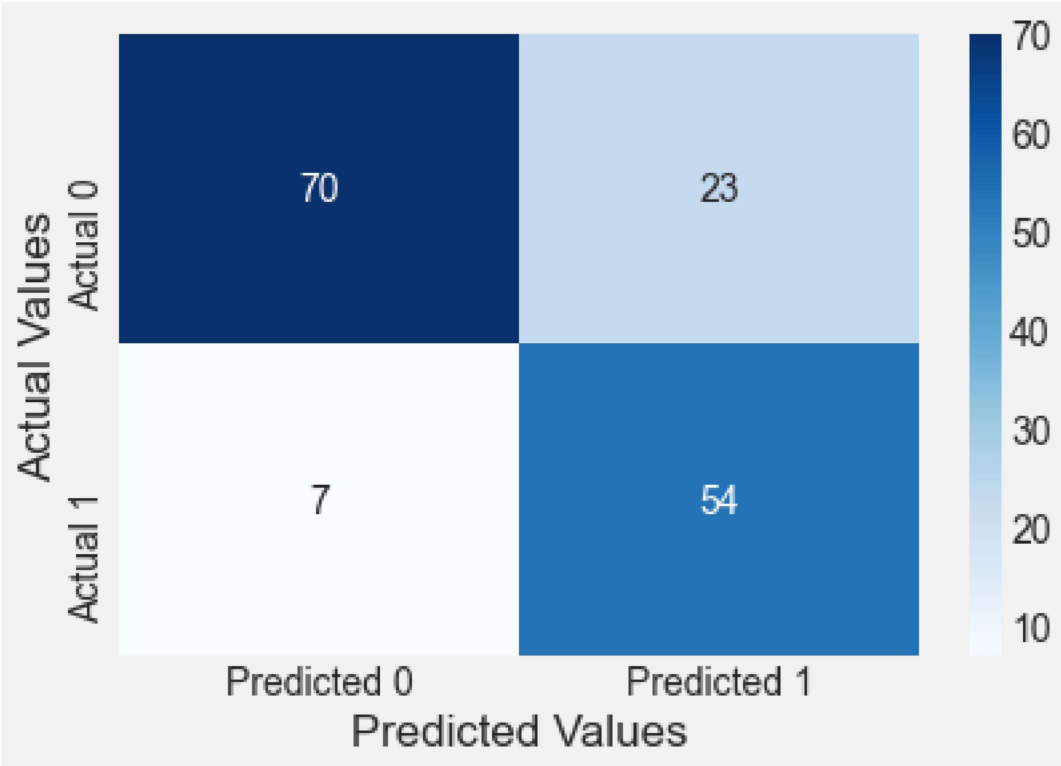
```
confusion_matrix_train(logit_model_diabetes,cutoff=0.3)
```

```
Classification accuracy = 74.3%
```




```
confusion_matrix_test(test,test.Outcome,logit_model_diabetes,0.3)
```

```
Classification accuracy = 80.5%
```



The model with more predictors also has lesser number of *false negatives*, and higher overall classification accuracy.

How many bins must you make for `Age` to get the most accurate model?
If the number of bins are too less, the trend may not be captured accurately. If the number of bins are too many, it may lead to overfitting of the model. There is an optimal value of the number of bins that captures the trend, but does not overfit. A couple of ways of estimating the optimal number of bins can be:

1. The number of bins for which the trend continues to be "almost" the same for several samples of the data.

2. Testing the model on multiple test datasets.

Optimizing the number of bins for each predictor may be a time-consuming exercises. You may do it for your course project. However, we will not do it here in the class notes.

4.6 Performance Measurement

We have already seen the confusion matrix, and classification accuracy. Now, let us see some other useful performance metrics that can be computed from the confusion matrix. The metrics below are computed for the confusion matrix immediately above this section *(or the confusion matrix on test data corresponding to the model `logit_model_diabetes`)*.

4.6.1 Precision-recall

Precision measures the accuracy of positive predictions. Also called the `precision` of the classifier

$$\text{precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

==> 70.13%

`Precision` is typically used with `recall` (`Sensitivity` or `True Positive Rate`). The ratio of positive instances that are correctly detected by the classifier.

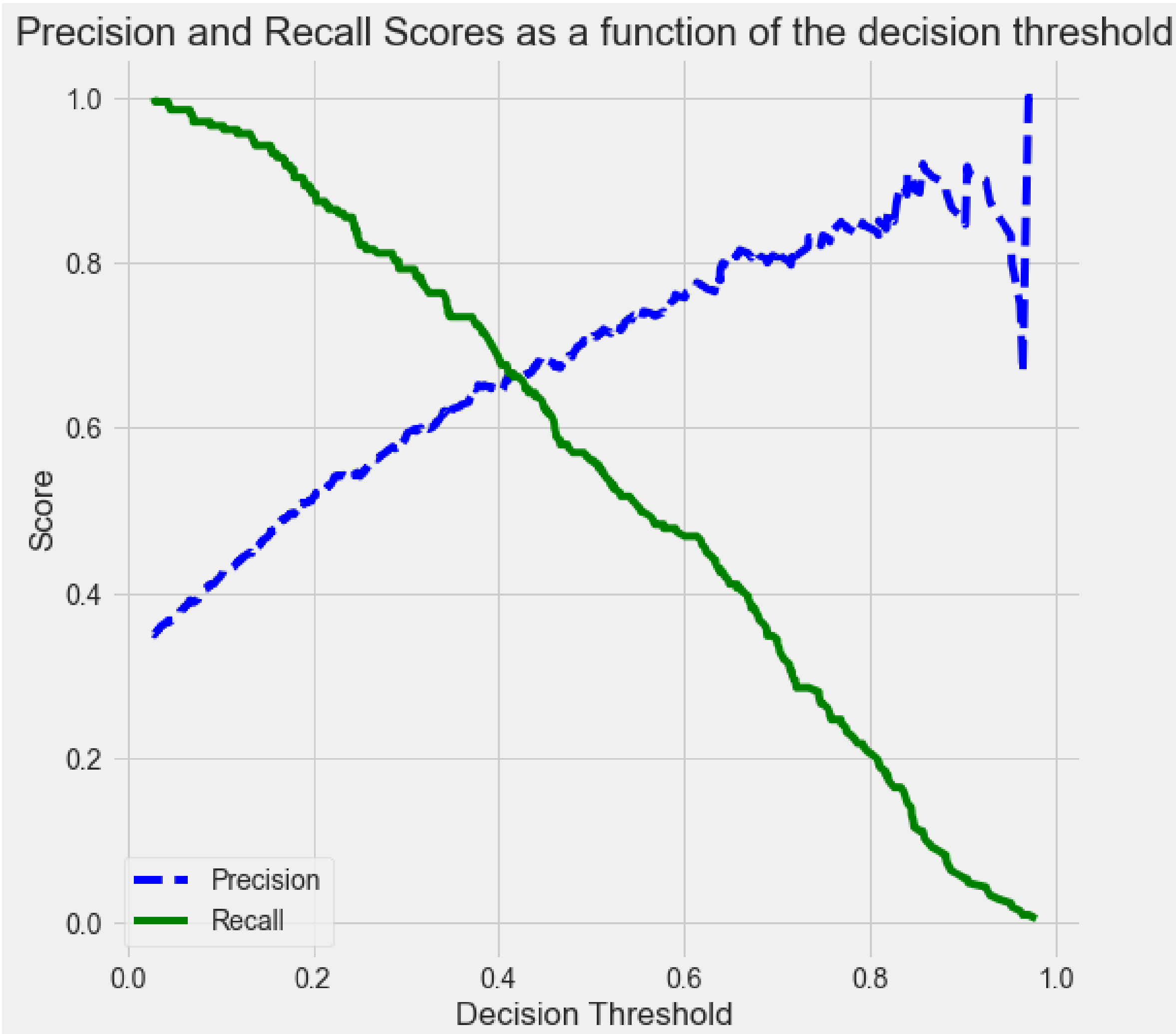
$$\text{recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

==> 88.52%

Precision / Recall Tradeoff: Increasing precision reduces recall and vice versa.

Visualize the precision-recall curve for the model `logit_model_diabetes`.

```
y=train.Outcome
ypred = logit_model_diabetes.predict(train)
p, r, thresholds = precision_recall_curve(y, ypred)
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.figure(figsize=(8, 8))
    plt.title("Precision and Recall Scores as a function of the decision threshold")
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.ylabel("Score")
    plt.xlabel("Decision Threshold")
    plt.legend(loc='best')
    plt.legend()
plot_precision_recall_vs_threshold(p, r, thresholds)
```



As the decision threshold probability increases, the precision increases, while the recall decreases.

Q: How are the values of the `thresholds` chosen to make the precision-recall curve?

Hint: Look at the documentation for `precision_recall_curve`.

4.6.2 The Receiver Operating Characteristics (ROC) Curve

A **ROC(Receiver Operator Characteristic Curve)** is a plot of sensitivity (True Positive Rate) on the y axis against (1−specificity) (False Positive Rate) on the x axis for varying values of the threshold t. The 45° diagonal line connecting (0,0) to (1,1) is the ROC curve corresponding to random chance. The ROC curve for the gold standard is the line connecting (0,0) to (0,1) and (0,1) to (1,1).

```
&lt;IPython.core.display.Image object&gt;
```

```
&lt;IPython.core.display.Image object&gt;
```

An animation to demonstrate how an ROC curve relates to sensitivity and specificity for all possible cutoffs (Source)

High Threshold:

- High specificity
- Low sensitivity

Low Threshold

- Low specificity
- High sensitivity

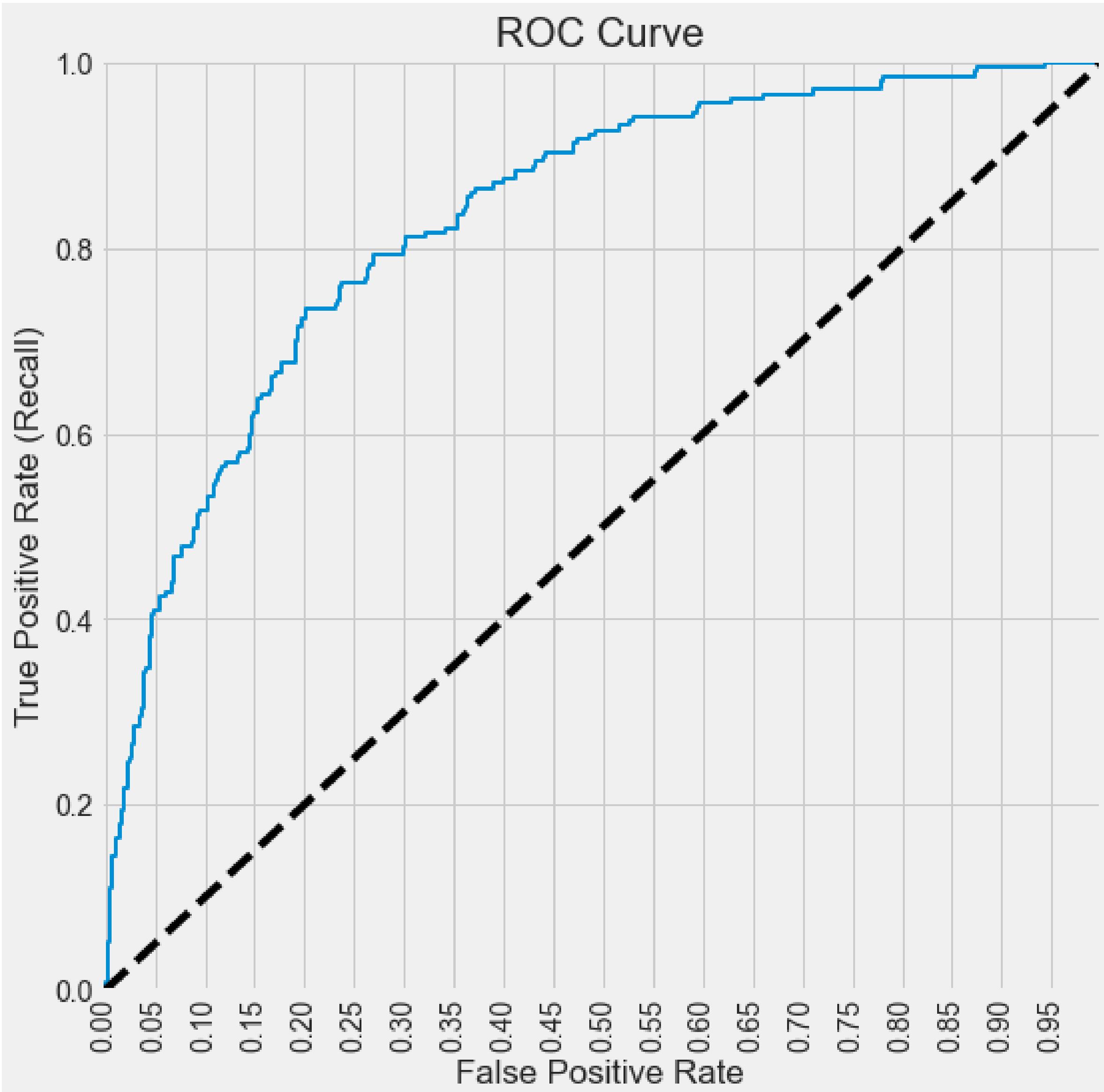
The area under ROC is called *Area Under the Curve(AUC)*. AUC gives the rate of successful classification by the logistic model. To get a more in-depth idea of what a ROC-AUC curve is and how is it calculated, here is a good blog link.

Here is good post by google developers on interpreting ROC-AUC, and its advantages / disadvantages.

Visualize the ROC curve and compute the ROC-AUC for the model `logit_model_diabetes`.

```
y=train.Outcome
ypred = logit_model_diabetes.predict(train)
fpr, tpr, auc_thresholds = roc_curve(y, ypred)
print(auc(fpr, tpr))# AUC of ROC
def plot_roc_curve(fpr, tpr, label=None):
    plt.figure(figsize=(8,8))
    plt.title('ROC Curve')
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.005, 1, 0, 1.005])
    plt.xticks(np.arange(0,1, 0.05), rotation=90)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate (Recall)")

fpr, tpr, auc_thresholds = roc_curve(y, ypred)
plot_roc_curve(fpr, tpr)
```

```
0.8325914847653979
```



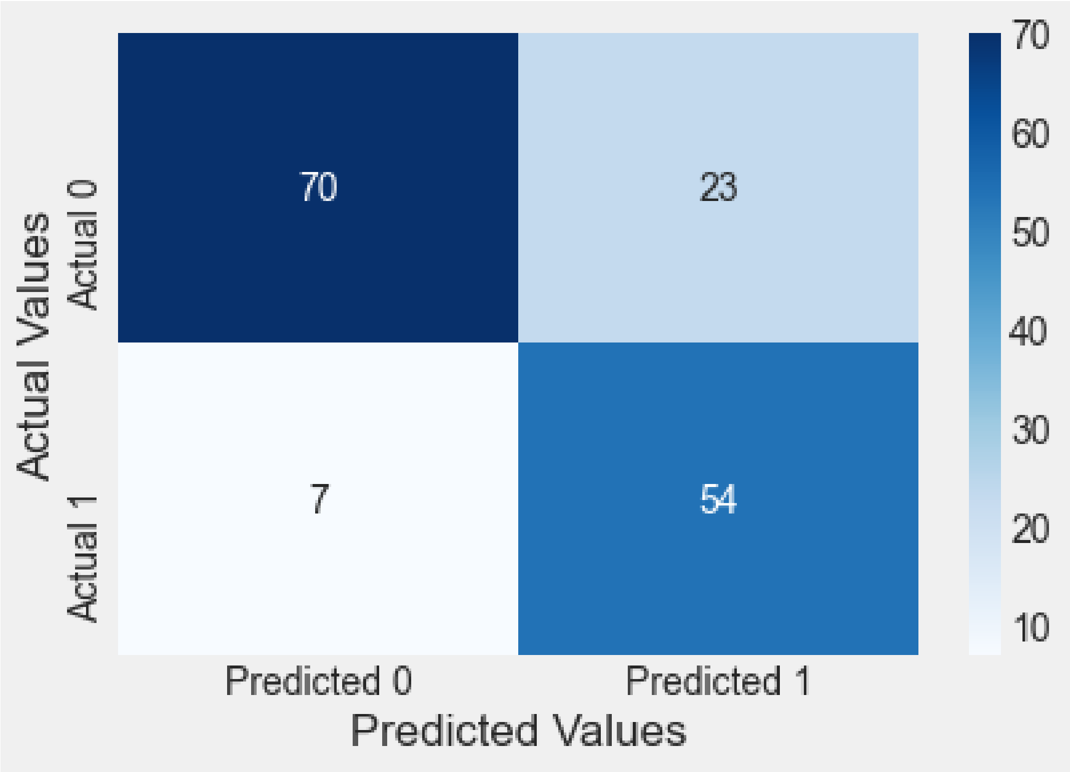
Q: How are the values of the `auc_thresholds` chosen to make the ROC curve? Why does it look like a step function?

Below is a function that prints the confusion matrix along with all the performance metrics we discussed above for a given decision threshold probability, on train / test data. Note that ROC-AUC does not depend on a decision threshold probability.

```
#Function to compute confusion matrix and prediction accuracy on test/train data
def confusion_matrix_data(data,actual_values,model,cutoff=0.5):
#Predict the values using the Logit model
    pred_values = model.predict(data)
# Specify the bins
    bins=np.array([0,cutoff,1])
#Confusion matrix
    cm = np.histogram2d(actual_values, pred_values, bins=bins)[0]
    cm_df = pd.DataFrame(cm)
    cm_df.columns = ['Predicted 0','Predicted 1']
    cm_df = cm_df.rename(index={0: 'Actual 0',1:'Actual 1'})
# Calculate the accuracy
    accuracy = (cm[0,0]+cm[1,1])/cm.sum()
    fnr = (cm[1,0])/(cm[1,0]+cm[1,1])
    precision = (cm[1,1])/(cm[0,1]+cm[1,1])
    fpr = (cm[0,1])/(cm[0,0]+cm[0,1])
    tpr = (cm[1,1])/(cm[1,0]+cm[1,1])
    fpr_roc, tpr_roc, auc_thresholds = roc_curve(actual_values, pred_values)
    auc_value = (auc(fpr_roc, tpr_roc))# AUC of ROC
    sns.heatmap(cm_df, annot=True, cmap='Blues', fmt='g')
    plt.ylabel("Actual Values")
    plt.xlabel("Predicted Values")
    print("Classification accuracy = {:.1%}".format(accuracy))
    print("Precision = {:.1%}".format(precision))
    print("TPR or Recall = {:.1%}".format(tpr))
    print("FNR = {:.1%}".format(fnr))
    print("FPR = {:.1%}".format(fpr))
    print("ROC-AUC = {:.1%}".format(auc_value))
```

```
confusion_matrix_data(test,test.Outcome,logit_model_diabetes,0.3)
```

```
Classification accuracy = 80.5%
Precision = 70.1%
TPR or Recall = 88.5%
FNR = 11.5%
FPR = 24.7%
ROC-AUC = 90.1%
```



4.7 sklearn

The `LogisticRegression()` function from the `linear_model` module of the `sklearn` library is used for fitting a logistic regression model. Note that the function as a default regularization parameter value set as `C = 1`. We'll understand the purpose of regularization later in the course.

```
train = pd.read_csv('./Datasets/Social_Network_Ads_train.csv') #Develop the model on train da
test = pd.read_csv('./Datasets/Social_Network_Ads_test.csv') #Test the model on test data

X_train = train[['Age']]
y_train = train['Purchased']

X_test = test[['Age']]
y_test = test['Purchased']

model = LogisticRegression(penalty=None) # We will talk about this input later in the quarte

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test) # Note that in sklearn, .predict returns the classes directly

print("Accuracy = ",accuracy_score(test.Purchased, y_pred)*100, "%")

# To return the prediction probabilities, we need .predict_proba

y_pred_probs = model.predict_proba(X_test)

# First col: class 0 prob, second col: class 1 prob

# We will need the probs to try different thresholds - this will be necessary for the other r
```

```
Accuracy =  86.0 %
```

5 Ridge regression and Lasso

Read section 6.2 of the book before using these notes.

Note that in this course, lecture notes are not sufficient, you must read the book for better understanding. Lecture notes are just implementing the concepts of the book on a dataset, but not explaining the concepts elaborately.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV, LogisticRegressionCV, Logis
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, accuracy_score
from sklearn.model_selection import cross_val_score, cross_val_predict
```

```
trainf = pd.read_csv('./Datasets/house_feature_train.csv')
trainp = pd.read_csv('./Datasets/house_price_train.csv')
testf = pd.read_csv('./Datasets/house_feature_test.csv')
testp = pd.read_csv('./Datasets/house_price_test.csv')
train = pd.merge(trainf,trainp)
test = pd.merge(testf,testp)
train.head()
```

	house_id	house_age	distance_MRT	number_convenience_stores	latitude	longitude	house_pri
0	210	5.2	390.5684	5	24.97937	121.54245	2724.84
1	190	35.3	616.5735	8	24.97945	121.53642	1789.29
2	328	15.9	1497.7130	3	24.97003	121.51696	556.96
3	5	7.1	2175.0300	3	24.96305	121.51254	1030.41
4	412	8.1	104.8101	5	24.96674	121.54067	2756.25

5.1 Ridge regression

Let us develop a ridge regression model to predict house price based on the five house features.

```
#Taking the log transform of house_price as house prices have a right-skewed distribution
y = np.log(train.house_price)
```

5.1.1 Standardizing the predictors

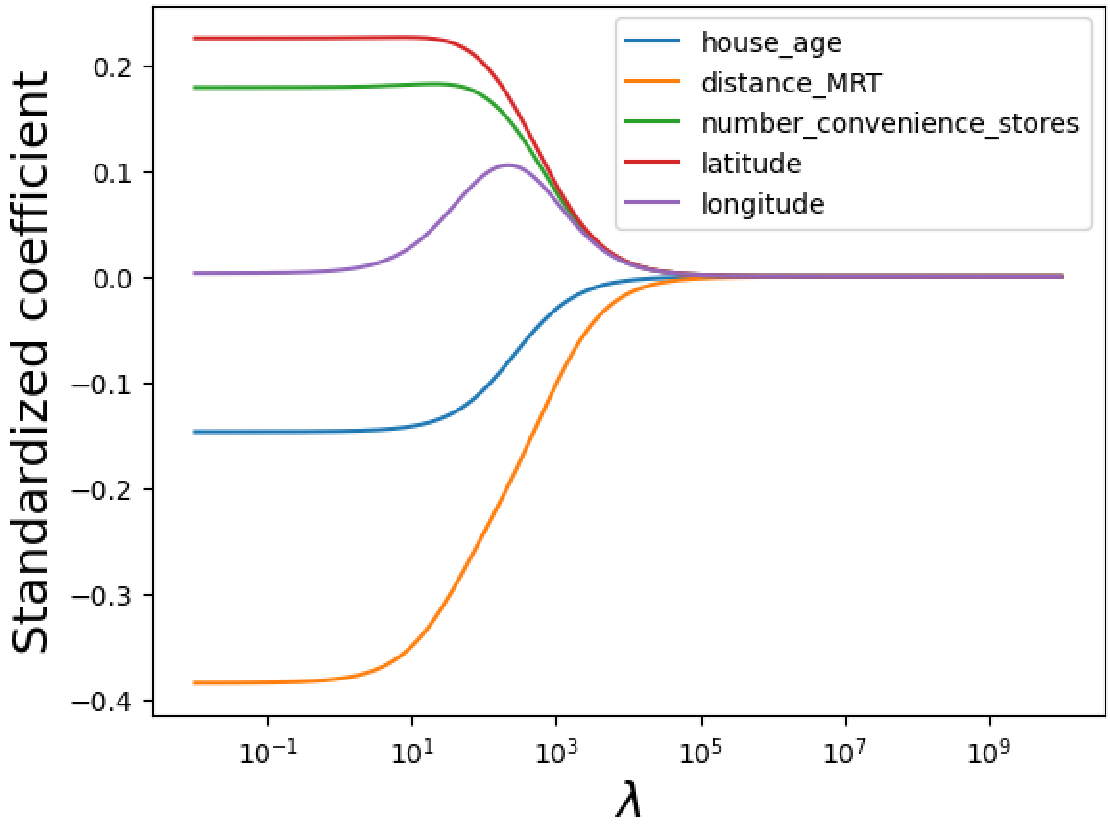
```
#Standardizing predictors so that each of them have zero mean and unit variance

#Filtering all predictors
X = train.iloc[:,1:6];

#Defining a scaler object
scaler = StandardScaler()

#The scaler object will contain the mean and variance of each column (predictor) of X.
#These values will be useful to scale test data based on the same mean and variance as obtai
scaler.fit(X)

#Using the scaler object (or the values of mean and variance stored in it) to standardize X
Xstd = scaler.transform(X)
```

5.1.2 Optimizing the tuning parameter

```
#The tuning parameter lambda is referred as alpha in sklearn

#Creating a range of values of the tuning parameter to visualize the ridge regression coeffi
#for different values of the tuning parameter
alphas = np.logspace(10,-2,200)
```

```
#Finding the ridge regression coefficients for increasing values of the tuning parameter
coefs = []
for a in alphas:
    ridge = Ridge(alpha = a)
    ridge.fit(Xstd, y)
    coefs.append(ridge.coef_)
```

```
#Visualizing the shrinkage in ridge regression coefficients with increasing values of the tun
plt.xlabel('xlabel', fontsize=18)
plt.ylabel('ylabel', fontsize=18)
plt.plot(alphas, coefs)
plt.xscale('log')
plt.xlabel('$\lambda$')
plt.ylabel('Standardized coefficient')
plt.legend(train.columns[1:6]);
```

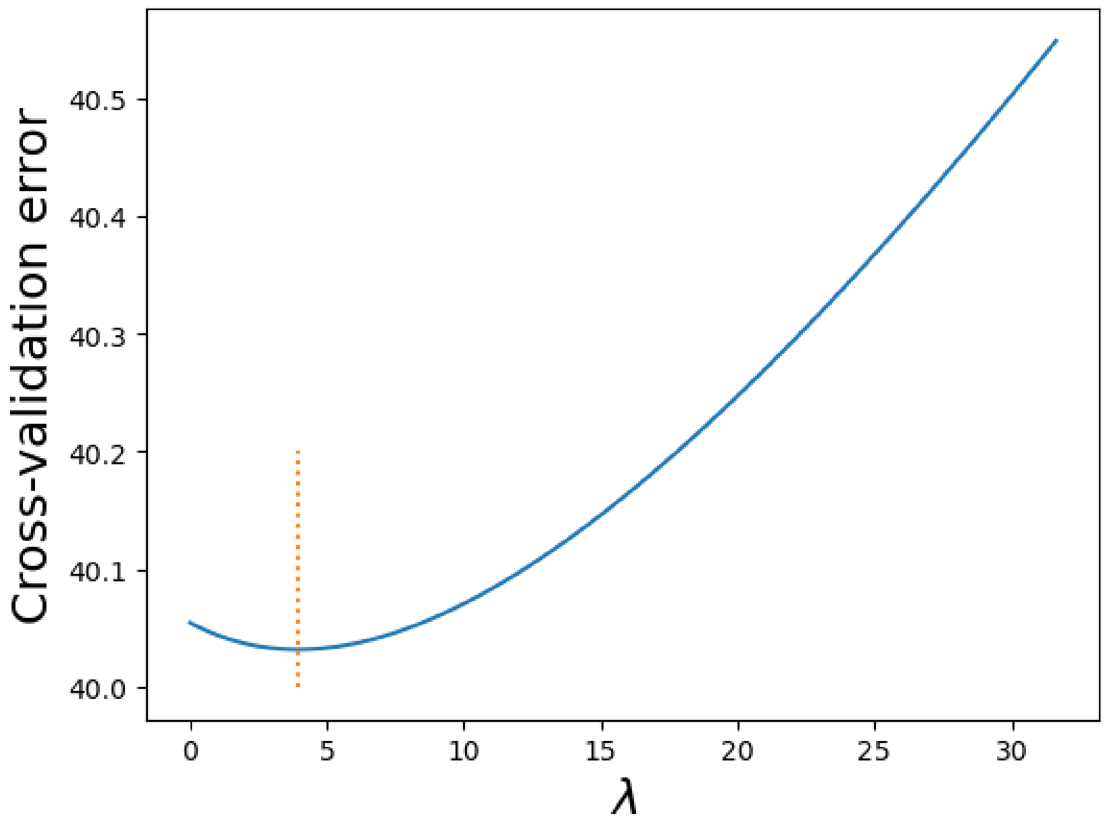



```
#Let us use cross validation to find the optimal value of the tuning parameter - lambda
#For the optimal lambda, the cross validation error will be the least

#Note that we are reducing the range of alpha so as to better visualize the minimum
alphas = np.logspace(1.5,-3,200)
ridgecv = RidgeCV(alphas = alphas,store_cv_values=True)
ridgecv.fit(Xstd, y)
```

```
#Optimal value of the tuning parameter - lambda
ridgecv.alpha_
```

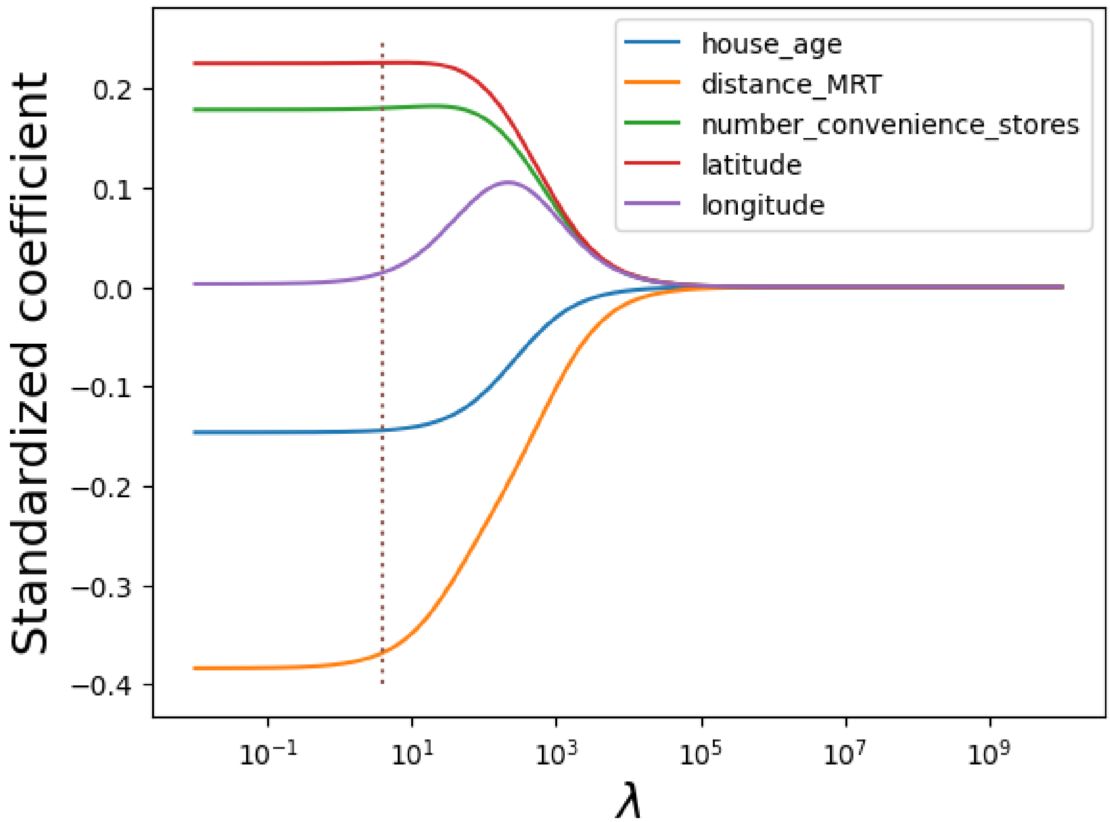
```
3.939829130085526
```

```
#Visualizing the LOOCV (leave one out cross validatation error vs lambda)
plt.xlabel('xlabel', fontsize=18)
plt.ylabel('ylabel', fontsize=18)
plt.plot(ridgecv.alphas,ridgecv.cv_values_.sum(axis=0))
plt.plot([ridgecv.alpha_,ridgecv.alpha_],[40,40.2],':')
plt.xlabel('$\lambda$')
plt.ylabel('Cross-validation error');
```



Note that the cross validation error is minimum at the optimal value of the tuning parameter.

```
#Visualizing the shrinkage in ridge regression coefficients with increasing values of the tur
alphas = np.logspace(10,-2,200)
plt.xlabel('xlabel', fontsize=18)
plt.ylabel('ylabel', fontsize=18)
plt.plot(alphas, coefs)
plt.plot([ridgecv.alpha_,ridgecv.alpha_],[-0.4,0.25],':')
plt.xscale('log')
plt.xlabel('$\lambda$')
plt.ylabel('Standardized coefficient')
plt.legend(train.columns[1:6]);
```



5.1.3 RMSE on test data

```
#Test dataset
Xtest = test.iloc[:,1:6]
```

```
#Standardizing test data
Xtest_std = scaler.transform(Xtest)
```

```
#Using the developed ridge regression model to predict on test data
ridge = Ridge(alpha = ridgecv.alpha_)
ridge.fit(Xstd, y)
pred=ridge.predict(Xtest_std)
```

```
#RMSE on test data
np.sqrt(((np.exp(pred)-test.house_price)**2).mean())
```

```
405.64878431933295
```

Note that the RMSE is similar to the one obtained using least squares regression on all the five predictors. This is because the coefficients were required to shrink very slightly for the best ridge regression fit. This may happen when we have a low number of predictors, where most of them are significant. Ridge regression is likely to perform better than least squares in case of a large number of predictors, where an OLS model will be prone to overfitting.

5.1.4 Model coefficients & R-squared

```
#Checking the coefficients of the ridge regression model
ridge.coef_
```

```
array([-0.14444475, -0.3683359 ,  0.17988341,  0.22567002,  0.01429926])
```

Note that none of the coefficients are shrunk to zero. The coefficient of `longitude` is smaller than the rest, but not zero.

```
#R-squared on train data for the ridge regression model
r2_score(ridge.predict(Xstd),y)
```

```
0.6993726041206049
```

```
#R-squared on test data for the ridge regression model
r2_score(pred,np.log(test.house_price))
```

```
0.757276231336096
```

5.2 Lasso

Let us develop a lasso model to predict house price based on the five house features.

5.2.1 Standardizing the predictors

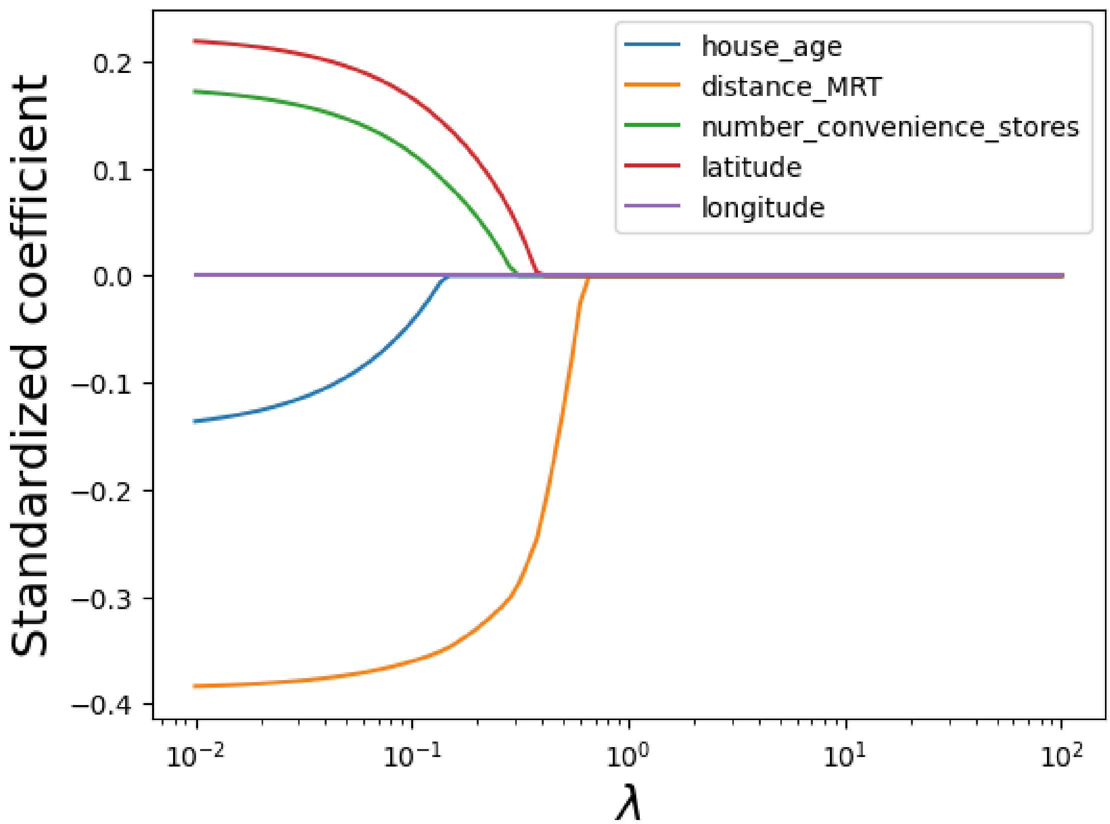
We have already standardized the predictors in the previous section. The standardized predictors are the NumPy array object `Xstd`.

5.2.2 Optimizing the tuning parameter

```
#Creating a range of values of the tuning parameter to visualize the lasso coefficients
#for different values of the tuning parameter
alphas = np.logspace(2,-2,100)
```

```
#Finding the lasso coefficients for increasing values of the tuning parameter
lasso = Lasso(max_iter = 10000)
coefs = []

for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(Xstd, y)
    coefs.append(lasso.coef_)
```

```
#Visualizing the shrinkage in lasso coefficients with increasing values of the tuning parame
plt.xlabel('xlabel', fontsize=18)
plt.ylabel('ylabel', fontsize=18)
plt.plot(alphas, coefs)
plt.xscale('log')
plt.xlabel('$\lambda$')
plt.ylabel('Standardized coefficient')
plt.legend(train.columns[1:6]);
```

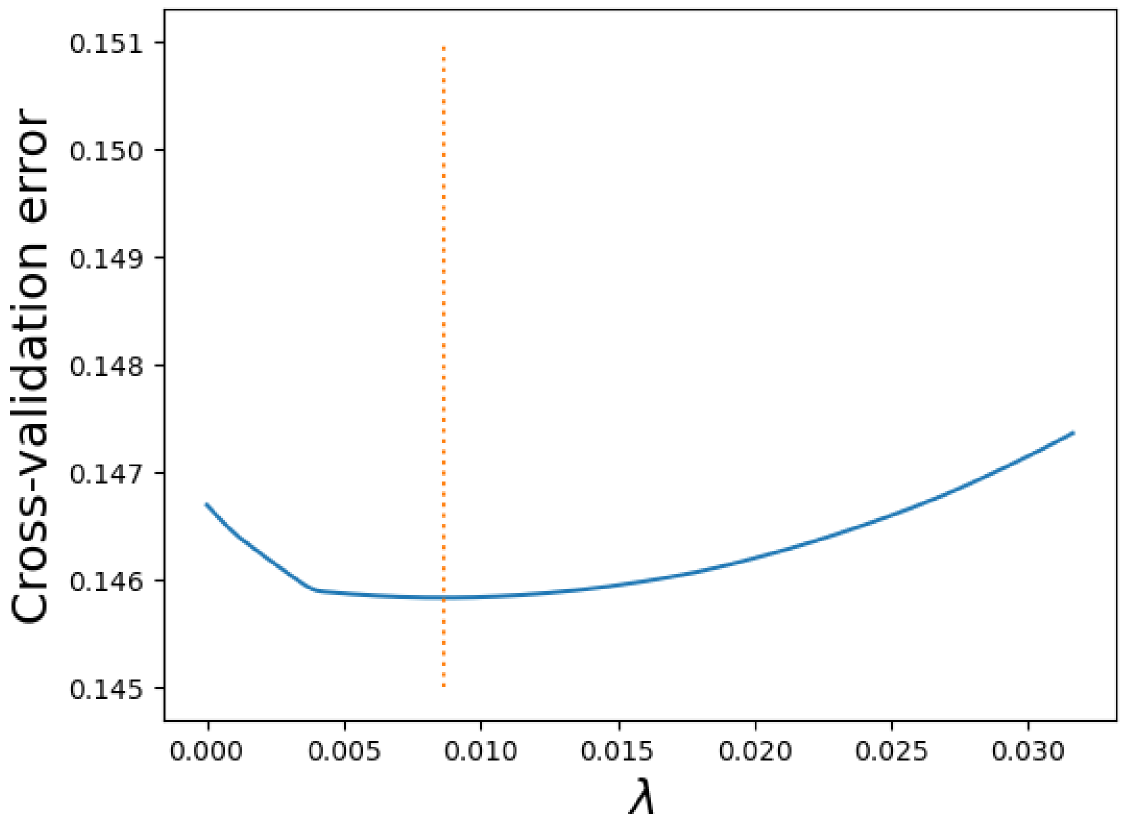


Note that lasso performs variable selection. For certain values of lambda, some of the predictor coefficients are zero, while others are non-zero. This is different than ridge regression, which only shrinks the coefficients, but doesn't do variable selection.

```
#Let us use cross validation to find the optimal value of the tuning parameter - lambda
#For the optimal lambda, the cross validation error will be the least

#Note that we are reducing the range of alpha so as to better visualize the minimum
alphas = np.logspace(-1.5,-5,200)
lassocv = LassoCV(alphas = alphas, cv = 10, max_iter = 100000)
lassocv.fit(Xstd, y)

#Optimal value of the tuning parameter - lamda
lassocv.alpha_
```

```
0.00865338307114046
```

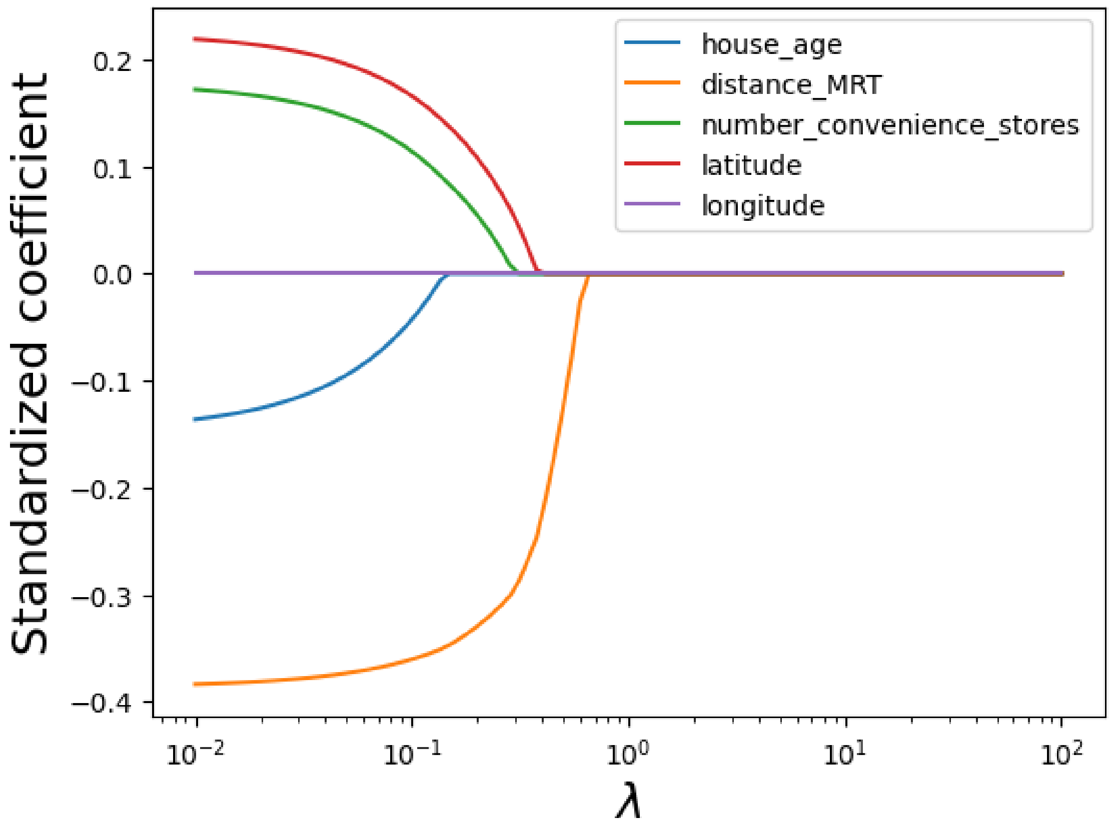
```
#Visualizing the LOOCV (leave one out cross validatation error vs lambda)
plt.xlabel('xlabel', fontsize=18)
plt.ylabel('ylabel', fontsize=18)
plt.plot(lassocv.alphas_,lassocv.mse_path_.mean(axis=1))
plt.plot([lassocv.alpha_,lassocv.alpha_],[0.145,0.151],':')
plt.xlabel('$\lambda$')
plt.ylabel('Cross-validation error');
```



The 10-fold cross validation error minimizes at lambda = 0.009.

```
#Visualizing the shrinkage in lasso coefficients with increasing values of the tuning parame
alphas = np.logspace(2,-2,100)
plt.xlabel('xlabel', fontsize=18)
plt.ylabel('ylabel', fontsize=18)
plt.plot(alphas, coefs)
plt.xscale('log')
plt.xlabel('$\lambda$')
```

```
plt.ylabel('Standardized coefficient')
plt.legend(train.columns[1:6]);
```



5.2.3 RMSE on test data

```
#Using the developed lasso model to predict on test data
lasso = Lasso(alpha = lassocv.alpha_)
lasso.fit(Xstd, y)
pred=lasso.predict(Xtest_std)
```

```
#RMSE on test data
np.sqrt(((np.exp(pred)-test.house_price)**2).mean())
```

```
400.8580108804818
```

5.2.4 Model coefficients & R-squared

```
#Checking the coefficients of the lasso model
lasso.coef_
```

```
array([-0.13758288, -0.38414914,  0.17276584,  0.21970825,  0.        ])
```

Note that the coefficient of `longitude` is shrunk to zero. Lasso performs variable selection.

```
#R-squared on train data for the lasso model
r2_score(lasso.predict(Xstd),y)
```

```
0.6931007715680897
```

```
#R-squared on test data for the lasso model
r2_score(pred,np.log(test.house_price))
```

```
0.7526968660283655
```

5.3 Lasso/Ridge Classification

The Ridge and Lasso penalties are added from inside the same LogisticRegression object, they don't have their own objects like they do in regression.

```
# Data
train = pd.read_csv('Datasets/Social_Network_Ads_train.csv')
test = pd.read_csv('Datasets/Social_Network_Ads_test.csv')
```

```
# Predictors and response
X_train = train[['Age', 'EstimatedSalary']]
y_train = train['Purchased']

X_test = test[['Age', 'EstimatedSalary']]
y_test = test['Purchased']
```

```
# Creating the model
    # penalty=None means regular logistic Regression
    # penalty=l2 means Ridge Classification
    # penalty=l1 means Lasso Classification
    # C = 1/lambda

model = LogisticRegression(penalty='l2', C = 1)
```

```
# Scale
sc = StandardScaler()
sc.fit(X_train)
X_train_scaled = sc.transform(X_train)
X_test_scaled = sc.transform(X_test)

# Train
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled) # threshold = 0.5 here

# Evaluate
print(accuracy_score(test.Purchased, y_pred)*100)

# Probs
y_pred_probs = model.predict_proba(X_test_scaled)
```

```
88.0
```

5.3.1 Cross-validation to find optimal C

```
# a list of possible C values
Cs = np.logspace(-1,1)

# Cs =  the C values we want to try out
# cv = number of folds, 3,5,10 - if no input given, 5-fold
# penalty = Ridge or Lasso
model_cv = LogisticRegressionCV(Cs = Cs, cv=5, penalty='l2')

model_cv.fit(X_train_scaled, y_train)

model_cv.C_[0]
```

```
1.3894954943731375
```

```
model = LogisticRegression(penalty='l2', C = model_cv.C_[0])
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled) # threshold = 0.5 here

# Evaluate
print(accuracy_score(test.Purchased, y_pred)*100)
```

```
88.0
```

6 Cross-validation

Read section 5.1 of the book before using these notes.

Note that in this course, lecture notes are not sufficient, you must read the book for better understanding. Lecture notes are just implementing the concepts of the book on a dataset, but not explaining the concepts elaborately.

- The aim of the notebook is to introduce how to use some low-level cross-validation tools.
- Why? Because unlike Lasso, Ridge and LogisticRegression, most models in sklearn don't have a CV version.
- In that case, you need to CV yourself with the tools in this notebook.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge, Lasso, LogisticRegression # No CV versions of the obj
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import mean_squared_error, mean_absolute_error, accuracy_score, roc_curv
precision_score, recall_score, confusion_matrix
from sklearn.model_selection import cross_val_score, cross_val_predict
```

6.1 Regression

```
trainf = pd.read_csv('Datasets/house_feature_train.csv')
trainp = pd.read_csv('Datasets/house_price_train.csv')
testf = pd.read_csv('Datasets/house_feature_test.csv')
testp = pd.read_csv('Datasets/house_price_test.csv')
train = pd.merge(trainf,trainp)
test = pd.merge(testf,testp)
train.head()
```

	house_id	house_age	distance_MRT	number_convenience_stores	latitude	longitude	house_pri
0	210	5.2	390.5684	5	24.97937	121.54245	2724.84
1	190	35.3	616.5735	8	24.97945	121.53642	1789.29
2	328	15.9	1497.7130	3	24.97003	121.51696	556.96
3	5	7.1	2175.0300	3	24.96305	121.51254	1030.41
4	412	8.1	104.8101	5	24.96674	121.54067	2756.25

```
# Data

# Train
y_train = np.log(train.house_price) # Response (log taken to account for the skewed dist. of
X_train = train.iloc[:,1:6] # Slice out the predictors

# Test
y_test = np.log(test.house_price) # Response (log taken to account for the skewed dist. of ho
X_test = test.iloc[:,1:6] # Slice out the predictor

# Scale both
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Let's tune the lambda of a Ridge model, with 5-fold CV.

# For that, we need to loop through lambda (alpha) values.
# However, we don't need to loop through folds - we will use a function for that! - cross_val

alphas = np.logspace(-1,1,200)

cv_results = []

for alpha in alphas: # For each alpha
    model = Ridge(alpha=alpha) # Create the model
    cv_results.append(cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='neg_root

# Note that the input is the model object, the data, number of folds and the metric
# If you don't specify the scoring, it will use r-squared for regression and accuracy for cla
# The output is an array of k values, k being the number of folds (cv input)
```

```
# For each alpha value, 5 RMSE values

# Take the mean of each row to find avg cv score for each alpha
# Negative sign because the scoring input has "neg" in the previous cell
rmses = -np.array(cv_results).mean(axis=1)

# Index of the minimum CV RMSE
np.argmin(rmses)

alphas[np.argmin(rmses)]

# Note the same alpha as in RidgeCV example in the previous notebook
```

```
4.768611697714469
```

```
# Now we need to create one final Ridge model with the optimized alpha value

model = Ridge(alpha=alphas[np.argmin(rmses)])

model.fit(X_train_scaled, y_train)

# Predict
# Evaluate
```

```
Ridge(alpha=4.768611697714469)
```

6.2 Classification

```
# Data
train = pd.read_csv('Datasets/Social_Network_Ads_train.csv')
test = pd.read_csv('Datasets/Social_Network_Ads_test.csv')

# Predictors and response
X_train = train[['Age', 'EstimatedSalary']]
y_train = train['Purchased']

X_test = test[['Age', 'EstimatedSalary']]
y_test = test['Purchased']
```

```
# Scale
sc = StandardScaler()
sc.fit(X_train)
X_train_scaled = sc.transform(X_train)
X_test_scaled = sc.transform(X_test)
```

```
# CV a logistic regression model

# a list of possible C values
Cs = [0.001, 0.01, 0.1, 1, 10, 100]

cv_results = []

for C in Cs:
    model = LogisticRegression(penalty='l2', C=C)
    cv_results.append(cross_val_score(model, X_train_scaled, y_train, cv=10))

# Scoring not given, default metric is accuracy (you can use recall, precision etc.)
```

```
# For each C, 10 accuracy values

accs = np.array(cv_results).mean(axis=1)

Cs[np.argmax(accs)] # best C - Same as the output of LogisticRegressionCV in the previous no

# Train the final model
# predict
# Evaluate
```

```
1
```

- Important question: How were these accuracies calculated? With a threhold of 0.5
- What if we want to change/optimize the threshold in this process as well? Then `cross_val_score()` is not enough, we need to change the function!

```
# CV a logistic regression model - but do not return the accuracy metric for each fold
    # Return the PREDICTIONS FOR EACH FOLD

# a list of possible C values
Cs = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
```

```
cv_results = []

for C in Cs:
    model = LogisticRegression(penalty='l2', C=C)
    cv_results.append(cross_val_predict(model, X_train_scaled, y_train, cv=10, method='predic

# Cross_val_predict function has an optional input: method
```

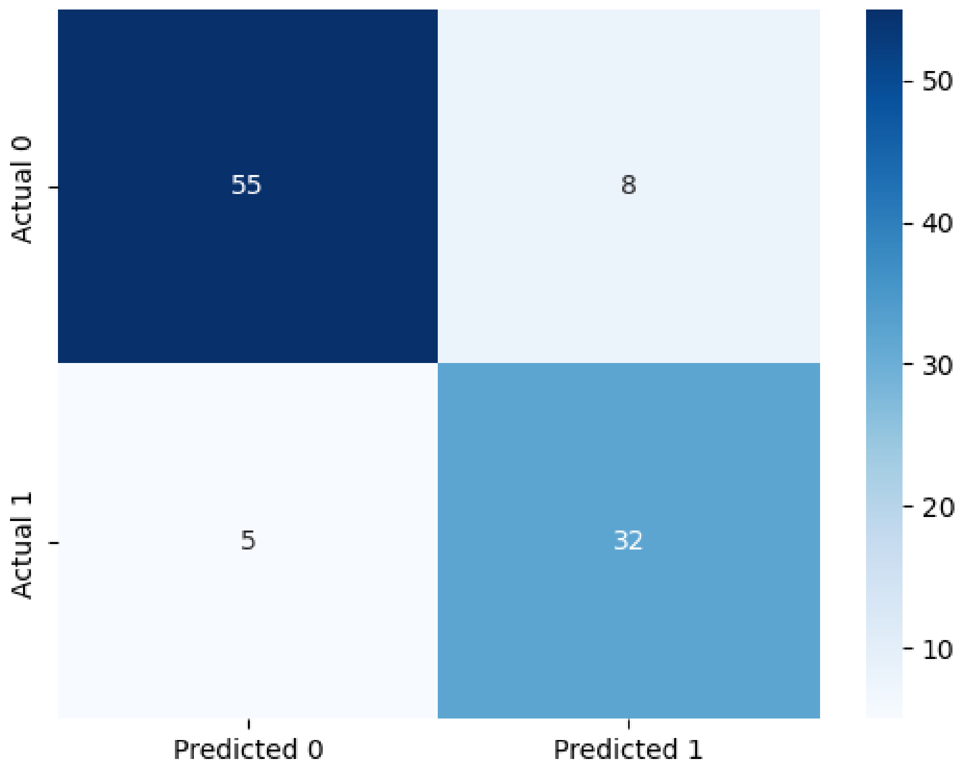
```
threshold_hyperparam_vals = np.arange(0,1.01,0.01)
C_hyperparam_vals = np.logspace(-3.5, 1)
accuracy_iter = pd.DataFrame(columns = {'threshold':[], 'C':[], 'accuracy':[]})
iter_number = 0

for c_val in C_hyperparam_vals:
    predicted_probability = cross_val_predict(LogisticRegression(C = c_val), X_train_scaled,
                                              y_train, cv = 5, method = 'predict_proba')

    for threshold_prob in threshold_hyperparam_vals:
        predicted_class = predicted_probability[:,1] > threshold_prob
        predicted_class = predicted_class.astype(int)

        #Computing the accuracy
        accuracy = accuracy_score(predicted_class, y_train)*100
        accuracy_iter.loc[iter_number, 'threshold'] = threshold_prob
        accuracy_iter.loc[iter_number, 'C'] = c_val
        accuracy_iter.loc[iter_number, 'accuracy'] = accuracy
        iter_number = iter_number + 1
```

```
# Parameters for highest accuracy
optimal_C = accuracy_iter.sort_values(by = 'accuracy', ascending = False).iloc[0,:]['C']
optimal_threshold = accuracy_iter.sort_values(by = 'accuracy', ascending = False).iloc[0, :]

#Optimal decision threshold probability
print("Optimal decision threshold = ", optimal_threshold)

#Optimal C
print("Optimal C = ", optimal_C)
```

```
Optimal decision threshold =  0.41000000000000003
Optimal C =  0.06250551925273976
```

```
model = LogisticRegression(C = optimal_C).fit(X_train_scaled, y_train)
test_pred = model.predict_proba(X_test_scaled)[:,1]

y_pred_optimal_threshold = (test_pred > optimal_threshold).astype(int)

#Computing the accuracy
print("Accuracy: ",accuracy_score(y_pred_optimal_threshold, y_test)*100)

#Computing the ROC-AUC
fpr, tpr, auc_thresholds = roc_curve(y_test, y_pred_optimal_threshold)
print("ROC-AUC: ",auc(fpr, tpr))# AUC of ROC

#Computing the precision and recall
print("Precision: ", precision_score(y_test, y_pred_optimal_threshold))
print("Recall: ", recall_score(y_test, y_pred_optimal_threshold))

#Confusion matrix
cm = pd.DataFrame(confusion_matrix(y_test, y_pred_optimal_threshold), columns=['Predicted 0'
            index = ['Actual 0', 'Actual 1'])
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g');
```

```
Accuracy:  87.0
ROC-AUC:  0.868940368940369
Precision:  0.8
Recall:  0.8648648648648649
```



- We will use `cross_val_score()` and `cross_val_predict()` repeatedly next quarter.
- There is a `cross_validate()` function that allows us to use multiple metrics at once (for example, accuracy and recall) - next quarter.

Find some more examples of using the cross validation and some other useful functions here.

7 Potential issues

Read section 3.3.3 (4, 5, & 6) of the book before using these notes.

Note that in this course, lecture notes are not sufficient, you must read the book for better understanding. Lecture notes are just implementing the concepts of the book on a dataset, but not explaining the concepts elaborately.

Let us continue with the car price prediction example from the previous chapter.

```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy import stats
from sklearn.model_selection import cross_val_predict
from patsy import dmatrices
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
trainf = pd.read_csv('./Datasets/Car_features_train.csv')
trainp = pd.read_csv('./Datasets/Car_prices_train.csv')
testf = pd.read_csv('./Datasets/Car_features_test.csv')
testp = pd.read_csv('./Datasets/Car_prices_test.csv')
train = pd.merge(trainf,trainp)
train.head()
```

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
0	18473	bmw	6 Series	2020	Semi-Auto	11	Diesel	145	53.3282	3.0	37980
1	15064	bmw	6 Series	2019	Semi-Auto	10813	Diesel	145	53.0430	3.0	33980
2	18268	bmw	6 Series	2020	Semi-Auto	6	Diesel	145	53.4379	3.0	36850
3	18480	bmw	6 Series	2017	Semi-Auto	18895	Diesel	145	51.5140	3.0	25998
4	18492	bmw	6 Series	2015	Automatic	62953	Diesel	160	51.4903	3.0	18990

```
# Considering the model developed to address assumptions in the previous chapter
# Model with an interaction term and a variable transformation term
ols_object = smf.ols(formula = 'np.log(price)~(year+engineSize+mileage+mpg)**2+I(mileage**2)
model_log = ols_object.fit()
model_log.summary()
```

Dep. Variable:	np.log(price)	**R-squared:**	0.803
Model:	OLS	**Adj. R-squared:**	0.803
Method:	Least Squares	**F-statistic:**	1834.
Date:	Sun, 10 Mar 2024	**Prob (F-statistic):**	0.00
Time:	16:51:01	**Log-Likelihood:**	-1173.8
No. Observations:	4960	**AIC:**	2372.
Df Residuals:	4948	**BIC:**	2450.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-238.2125	25.790	-9.237	0.000	-288.773	-187.652
year	0.1227	0.013	9.608	0.000	0.098	0.148
engineSize	13.8349	5.795	2.387	0.017	2.475	25.195
mileage	0.0005	0.000	3.837	0.000	0.000	0.001
mpg	-1.2446	0.345	-3.610	0.000	-1.921	-0.569
year:engineSize	-0.0067	0.003	-2.324	0.020	-0.012	-0.001
year:mileage	-2.67e-07	6.8e-08	-3.923	0.000	-4e-07	-1.34e-07
year:mpg	0.0006	0.000	3.591	0.000	0.000	0.001
engineSize:mileage	-2.668e-07	4.08e-07	-0.654	0.513	-1.07e-06	5.33e-07
engineSize:mpg	0.0028	0.000	6.842	0.000	0.002	0.004
mileage:mpg	7.235e-08	1.79e-08	4.036	0.000	3.72e-08	1.08e-07
I(mileage ** 2)	1.828e-11	5.64e-12	3.240	0.001	7.22e-12	2.93e-11

Omnibus:	711.514	**Durbin-Watson:**	0.498
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2545.807
Skew:	0.699	**Prob(JB):**	0.00
Kurtosis:	6.220	**Cond. No.**	1.73e+13

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.73e+13. This might indicate that there are
strong multicollinearity or other numerical problems.

```
#Computing RMSE on test data
pred_price_log = model_log.predict(testf)
np.sqrt(((testp.price - np.exp(pred_price_log))**2).mean())
```

```
9094.209473232966
```

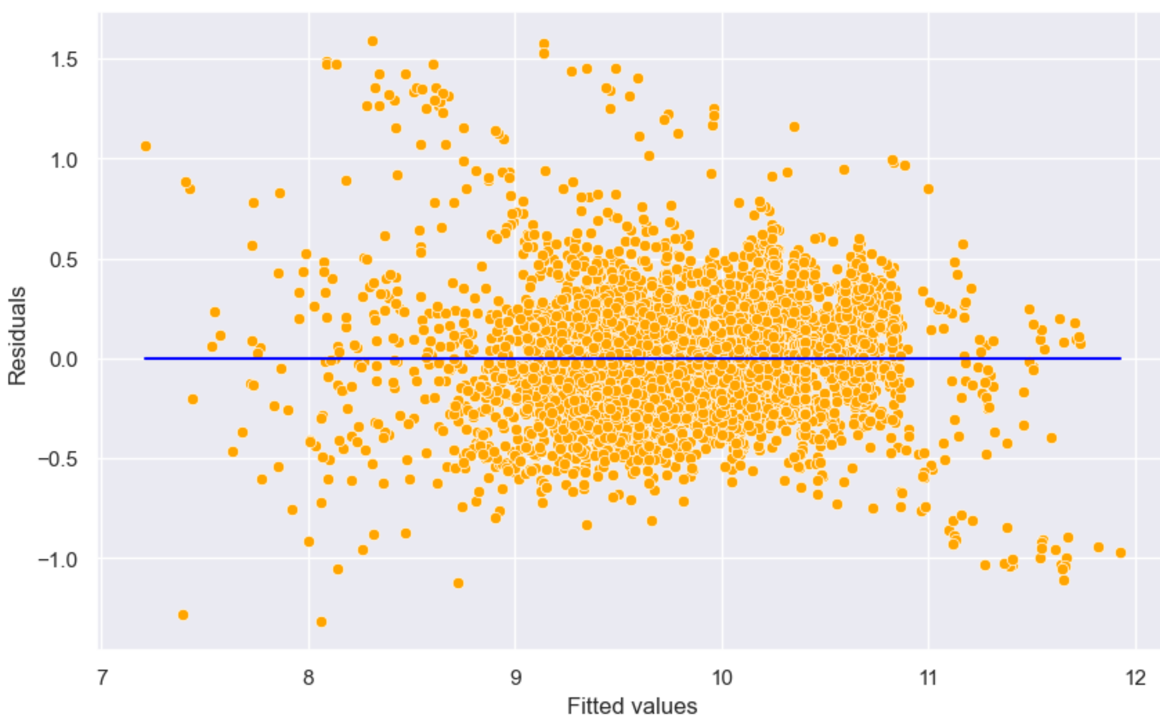
7.1 Outliers

An outlier is a point for which the true response (y_i) is far from the value predicted by the model. Residual plots can be used to identify outliers.

If the the response at the i^{th} observation is y_i, the prediction is $\hat{y}_i$, then the residual e_i is:

$$e_i = y_i - \hat{y}_i$$

```
#Plotting residuals vs fitted values
sns.set(rc={'figure.figsize':(10,6)})
sns.scatterplot(x = (model_log.fittedvalues), y=(model_log.resid),color = 'orange')
sns.lineplot(x = [model_log.fittedvalues.min(),model_log.fittedvalues.max()],y = [0,0],color
plt.xlabel('Fitted values')
plt.ylabel('Residuals');
```



Some of the errors may be high. However, it is difficult to decide how large a residual needs to be before we can consider a point to be an outlier. To address this problem, we have standardized residuals, which are defined as:

$$r_i = \frac{e_i}{RSE(\sqrt{1-h_{ii}})},$$

where r_i is the standardized residual, RSE is the residual standard error, and h_{ii} is the leverage *(introduced in the next section)* of the i^{th} observation.

Standardized residuals, allow the residuals to be compared on a *standard scale*.

Issue with standardized residuals:, If the observation corresponding to the standardized residual has a high leverage, then it will drag the regression line / plane / hyperplane towards it, thereby influencing the estimate of the residual itself.

Studentized residuals: To address the issue with standardized residuals, studentized residual for the i^{th} observation is computed as the standardized residual, but with the RSE (residual standard error) computed after removing the i^{th} observation from the data. Studentized residual, t_i for the i^{th} observation is given as:

$$t_i = \frac{e_i}{RSE_i(\sqrt{1-h_{ii}})},$$

where RSE_i is the residual standard error of the model developed on the data without the i^{th} observation.

Distribution of studentized residuals: If the regression model is appropriate such that no case is outlying because of a change in the model, then each studentized residual will follow a t distribution with (n–p–1) degrees of freedom.

As the studentized residuals follow a t distribution, we can conduct a hypothesis test to identify whether an observation is an outlier or not for a given sigificance level. Note that the test will be two-sided since we are not concerned with the sign of the residuals, but only their absolute values.

In the current example, for a signficance level of 5%, the critical t-statistic is $t(1-\frac{\alpha}{2}, n-p-1)$, as calculated below.

```
n = train.shape[0]
p = model_log.df_model
alpha = 0.05

# Critical value
stats.t.ppf(1 - alpha/2, n - p - 1)
```

```
1.9604435402730618
```

If we were conducting the test for a single observation, we'll compare the studentized residual for that observation with the critical t-statistic, and if the residual is greater than the critical value, we'll consider that observation as an outlier.

However, typically, we'll be interested in conducting this test for all observations, and thus we'll need a more conservative critical value for the same signficance level. This critical value is given by the Bonferroni correction as $t(1 - \frac{\alpha}{2n}, n - p - 1)$.

Thus, the minimum value of studentized residual for which the observation will be classified as an outlier is:

```
critical_value = stats.t.ppf(1-alpha/(2*n), n - p - 1)
critical_value
```

```
4.4200129981725365
```

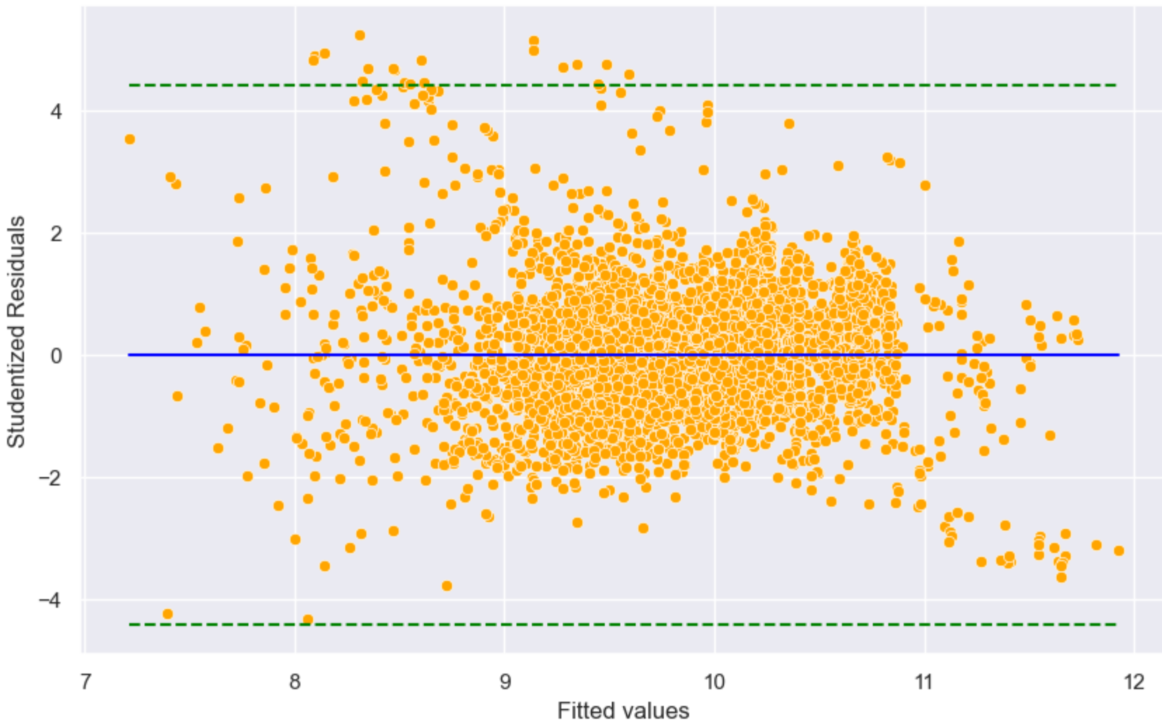
The studentized residuals can be obtained using the `outlier_test()` method of the object returned by the `fit()` method of an OLS object. Let us find the studentized residuals in our car `price` prediction model.

```
#Studentized residuals
out = model_log.outlier_test()
out
```

	student_resid	unadj_p	bonf(p)
0	-1.164204	0.244398	1.0
1	-0.801879	0.422661	1.0
2	-1.263820	0.206354	1.0
3	-0.614171	0.539131	1.0
4	0.027929	0.977720	1.0
...	...	...	...
4955	-0.523361	0.600747	1.0
4956	-0.509538	0.610398	1.0
4957	-1.718808	0.085712	1.0
4958	-0.077594	0.938154	1.0
4959	-0.482388	0.629551	1.0

Studentized residuals are in the first column of the above table. Let us plot the studentized residuals against fitted values. In the figure below, the studentized residuals above the top dotted green line and below the bottom dotted green line are outliers.

```
#Plotting studentized residuals vs fitted values
sns.scatterplot(x = (model_log.fittedvalues), y=(out.student_resid),color = 'orange')
sns.lineplot(x = [model_log.fittedvalues.min(),model_log.fittedvalues.max()],y = [0,0],color
ax = sns.lineplot(x = [model_log.fittedvalues.min(),model_log.fittedvalues.max()],y = [critic
             color = 'green')
sns.lineplot(x = [model_log.fittedvalues.min(),model_log.fittedvalues.max()],y = [-critical_v
             color = 'green')
ax.lines[1].set_linestyle("--")
ax.lines[2].set_linestyle("--")

plt.xlabel('Fitted values')
plt.ylabel('Studentized Residuals');
```



Outliers: Observations whose studentized residuals have a magnitude greater than $t(1-\frac{\alpha}{2n}, n-p-1)$.

Impact of outliers: Outliers do not have a large impact on the OLS line / plane / hyperplane as long as they don't have a high leverage *(discussed in the next section)*. However, outliers do inflate the residual standard error (RSE). RSE in turn is used to compute the standard errors of regression coefficients. As a result, statistically significant variables may appear to be insignificant, and R^2 may appear to be lower.

Are there outliers in our example?

```
#Number of points with absolute studentized residuals greater than critical_value
np.sum(np.abs(out.student_resid) > critical_value)
```

```
19
```

Let us analyze the outliers.

```
ind = (np.abs(out.student_resid) > critical_value)
pd.concat([train.loc[ind,:], np.exp(model_log.fittedvalues[ind])], axis = 1)
```

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	price
2042	18228	bmw	i3	2017	Automatic	24041	Hybrid	0	78.2726	0.0	2149
2046	17362	bmw	i3	2016	Automatic	68000	Hybrid	0	78.0258	0.0	1599
2050	19224	bmw	i3	2016	Automatic	20013	Hybrid	0	77.9310	0.0	1987
2051	13913	bmw	i3	2014	Automatic	34539	Hybrid	0	78.3838	0.0	1449
2055	16512	bmw	i3	2017	Automatic	28169	Hybrid	0	77.9799	0.0	2375
2059	15844	bmw	i3	2016	Automatic	19995	Hybrid	0	78.2825	0.0	1985
2060	12107	bmw	i3	2016	Automatic	8421	Hybrid	0	77.9125	0.0	1949
2061	18215	bmw	i3	2014	Automatic	37161	Hybrid	0	77.7505	0.0	1418
2063	15617	bmw	i3	2017	Automatic	41949	Hybrid	140	78.1907	0.0	1999
2064	18020	bmw	i3	2015	Automatic	9886	Hybrid	0	78.1810	0.0	1748
2143	12972	bmw	i8	2017	Automatic	9992	Hybrid	135	69.2767	1.5	5995
2144	13826	bmw	i8	2015	Automatic	43323	Hybrid	0	69.2683	1.5	4499
2150	18949	bmw	i8	2015	Automatic	43102	Hybrid	0	69.0922	1.5	4289
2151	18977	bmw	i8	2016	Automatic	10087	Hybrid	0	68.9279	1.5	4899
2744	18866	merc	M Class	2004	Automatic	121000	Diesel	325	29.3713	2.7	1995
3548	13149	audi	S4	2019	Automatic	4900	Diesel	145	40.7030	0.0	4500
4116	16420	audi	SQ5	2020	Automatic	1500	Diesel	145	34.7968	0.0	5645
4117	17611	audi	SQ5	2019	Automatic	1500	Diesel	145	34.5016	0.0	4880
4851	16577	bmw	Z3	2002	Automatic	16500	Petrol	325	29.7614	2.2	1499

Do you notice some unique characteristics of these observations due to which they may be outliers?

What methods you can propose to estimate the price of these outliers more accurately, which will also result in the overall reduction in RMSE?

7.2 High leverage points

High leverage points are those with an unusual value of the predictor(s). They have the potential to have a relatively higher impact on the OLS line / plane / hyperplane, as compared to the outliers.

Leverage statistic (page 99 of the book): In order to quantify an observation's leverage, we compute the leverage statistic. A large value of this statistic indicates an observation with high leverage. For simple linear regression,

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{i'=1}^{n}(x_{i'} - \bar{x})^2}. \tag{7.1}$$

It is clear from this equation that h_i increases with the distance of x_i from $\bar{x}$. A large value of h_i indicates that the i^{th} observation is distance from the center of all the other observations in terms of predictor values.

The leverage statistic h_i is always between $1/n$ and 1, and the average leverage for all the observations is always equal to $(p+1)/n$:

$$\bar{h} = \frac{p+1}{n} \tag{7.2}$$

So if a given observation has a leverage statistic that greatly exceeds $(p+1)/n$, then we may suspect that the corresponding point has high leverage.

If the i^{th} observation has a large leverage h_i, it may exercise substantial leverage in determining the fitted value $\widehat{Y}_i$, because:

- The fitted value $\widehat{Y}_i$ is a linear combination of the observed Y values, and h_i is the weight of observation Y_i in determining this fitted value.
- The larger the h_i, the smaller is the variance of the residual e_i, and the closer the fitted value $\widehat{Y}_i$ will tend to be the observed value Y_i.

Thumb rules:

- A leverage h_i is usually considered large if it is more than twice as large as the mean value $\bar{h}$.
- Another suggested guideline is that h_i values exceeding 0.5 indicate **very high leverage**, whereas those between 0.2 and 0.5 indicate moderate leverage.

Influential points: Note that if a high leverage point falls in line with the regression line, then it will not affect the regression line. However, it may inflate R-squared and increase the significance of predictors. If a high leverage point falls away from the regression line, then it is also an outlier, and will affect the regression line. The points whose presence significantly affects the regression line are called influential points. A point that is both a high leverage point and an outlier is likely to be an influential point. However, a high leverage point is not necessarily an influential point.

Source for influential points: https://online.stat.psu.edu/stat501/book/export/html/973

Let us see if there are any high leverage points in our regression `model`.

```
#Model with an interaction term and a variable transformation term
ols_object = smf.ols(formula = 'np.log(price)~(year+engineSize+mileage+mpg)**2+I(mileage**2)
model_log = ols_object.fit()
model_log.summary()
```

Dep. Variable:	np.log(price)	**R-squared:**	0.803
Model:	OLS	**Adj. R-squared:**	0.803
Method:	Least Squares	**F-statistic:**	1834.
Date:	Sun, 10 Mar 2024	**Prob (F-statistic):**	0.00
Time:	16:53:39	**Log-Likelihood:**	-1173.8
No. Observations:	4960	**AIC:**	2372.
Df Residuals:	4948	**BIC:**	2450.
Df Model:	11		
Covariance Type:	nonrobust		

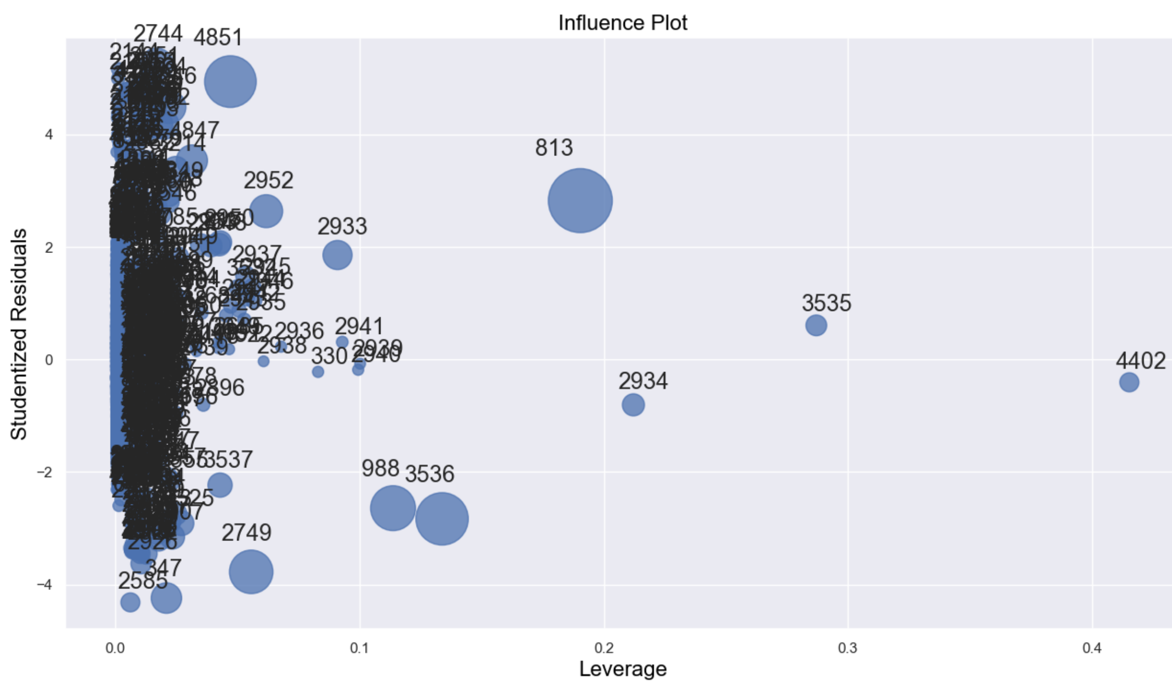
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-238.2125	25.790	-9.237	0.000	-288.773	-187.652
year	0.1227	0.013	9.608	0.000	0.098	0.148
engineSize	13.8349	5.795	2.387	0.017	2.475	25.195
mileage	0.0005	0.000	3.837	0.000	0.000	0.001
mpg	-1.2446	0.345	-3.610	0.000	-1.921	-0.569
year:engineSize	-0.0067	0.003	-2.324	0.020	-0.012	-0.001
year:mileage	-2.67e-07	6.8e-08	-3.923	0.000	-4e-07	-1.34e-07
year:mpg	0.0006	0.000	3.591	0.000	0.000	0.001
engineSize:mileage	-2.668e-07	4.08e-07	-0.654	0.513	-1.07e-06	5.33e-07
engineSize:mpg	0.0028	0.000	6.842	0.000	0.002	0.004
mileage:mpg	7.235e-08	1.79e-08	4.036	0.000	3.72e-08	1.08e-07
I(mileage ** 2)	1.828e-11	5.64e-12	3.240	0.001	7.22e-12	2.93e-11

Omnibus:	711.514	**Durbin-Watson:**	0.498
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2545.807
Skew:	0.699	**Prob(JB):**	0.00
Kurtosis:	6.220	**Cond. No.**	1.73e+13

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.73e+13. This might indicate that there are
strong multicollinearity or other numerical problems.

```
#Computing the leverage statistic for each observation
influence = model_log.get_influence()
leverage = influence.hat_matrix_diag
```

```
#Visualizng leverage against studentized residuals
sns.set(rc={'figure.figsize':(15,8)})
sm.graphics.influence_plot(model_log);
```



Let us identify the high leverage points in the data, as they may be affecting the regression line if they are outliers as well, i.e., if they are influential points. Note that there is no defined threshold for a point to be classified as a high leverage point. Some statisticians consider points having twice the average leverage as high leverage points, some consider points having thrice the average leverage as high leverage points, and so on.

```
out = model_log.outlier_test()
```

```
#Average leverage of points
average_leverage = (model_log.df_model+1)/model_log.nobs
average_leverage
```

```
0.0024193548387096775
```

Let us consider points having four times the average leverage as high leverage points.

```
#We will remove all observations that have leverage higher than the threshold value.
high_leverage_threshold = 3*average_leverage
```

```
#Number of high leverage points in the dataset
np.sum(leverage>high_leverage_threshold)
```

```
269
```

7.2.1 Identifying extrapolation using leverage

Leverage can be used to check if prediction on a particular point will lead to extrapolation.

Below is the function that can be used to find the leverage at for a particular observation `xnew`. Note that `xnew` has to be a single-dimensional array, and `X` has to be the predictor matrix (also called the design matrix).

```
def leverage_compute(xnew, X):
    return(xnew.reshape(-1, 1).T.dot(np.linalg.inv(X.T.dot(X))).dot(xnew.reshape(-1, 1))[0][0
```

As expected, the function will return the same leverage as provided by the `hat_matrix_diag` attribute of the objected returned by the `get_influence()` method of `model_log` as shown below:

```
leverage[0]
```

```
0.0026426981240353694
```

As the observation for prediction is required we need to create the predictor matrix `X` to create all the observations with the interactions specified in the model.

```
y, X = dmatrices('np.log(price)~(year+engineSize+mileage+mpg)**2+I(mileage**2)', data = trai
```

```
leverage_compute(X[0,:], X)
```

```
0.0026426973869101977
```

If the leverage for a new observation is higher than the maximum leverage among all the observations in the training dataset, then prediction at the new observation will be extrapolation.

7.3 Influential points

Observations that are both high leverage points and outliers are influential points that may affect the regression line. Let's remove these influential points from the data and see if it improves the model prediction accuracy on test data.

```
#Dropping influential points from data
train_filtered = train.drop(np.intersect1d(np.where(np.abs(out.student_resid) > critical_valu
                                              (np.where(leverage>high_leverage_threshold)[0])))
```

Note that as the Bonferroni's critical value is very conservative estimate, we have rounded off the critical value to 4, instead of 4.42.

```
train_filtered.shape
```

```
(4948, 11)
```

```
#Number of points removed as they were influential
train.shape[0]-train_filtered.shape[0]
```

```
12
```

We removed 12 influential data points from the training data.

```
#Model after removing the influential observations
ols_object = smf.ols(formula = 'np.log(price)~(year+engineSize+mileage+mpg)**2+I(mileage**2)
model_log = ols_object.fit()
model_log.summary()
```

Dep. Variable:	np.log(price)	**R-squared:**	0.815
Model:	OLS	**Adj. R-squared:**	0.814
Method:	Least Squares	**F-statistic:**	1971.
Date:	Sun, 10 Mar 2024	**Prob (F-statistic):**	0.00
Time:	16:54:08	**Log-Likelihood:**	-1027.9
No. Observations:	4948	**AIC:**	2080.
Df Residuals:	4936	**BIC:**	2158.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	-256.2339	25.421	-10.080	0.000	-306.070	-206.398
year	0.1317	0.013	10.462	0.000	0.107	0.156
engineSize	18.4650	5.663	3.261	0.001	7.364	29.566
mileage	0.0006	0.000	4.288	0.000	0.000	0.001
mpg	-1.1810	0.338	-3.489	0.000	-1.845	-0.517
year:engineSize	-0.0090	0.003	-3.208	0.001	-0.015	-0.004
year:mileage	-2.933e-07	6.7e-08	-4.374	0.000	-4.25e-07	-1.62e-07
year:mpg	0.0006	0.000	3.458	0.001	0.000	0.001
engineSize:mileage	-4.316e-07	4e-07	-1.080	0.280	-1.21e-06	3.52e-07
engineSize:mpg	0.0048	0.000	11.537	0.000	0.004	0.006
mileage:mpg	7.254e-08	1.75e-08	4.140	0.000	3.82e-08	1.07e-07
I(mileage ** 2)	1.668e-11	5.53e-12	3.017	0.003	5.84e-12	2.75e-11

Omnibus:	718.619	**Durbin-Watson:**	0.521
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2512.509
Skew:	0.714	**Prob(JB):**	0.00
Kurtosis:	6.185	**Cond. No.**	1.75e+13

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.75e+13. This might indicate that there are strong multicollinearity or other numerical problems.

Let us compare the square root of 5-fold cross-validated mean squared error of the model with and without the influential points.

```
y, X = dmatrices('np.log(price)~(year+engineSize+mileage+mpg)**2+I(mileage**2)', data = trai
np.sqrt(mean_squared_error(np.exp(cross_val_predict(LinearRegression(), X, y)), np.exp(y)))
```

```
9811.74078331643
```

```
y, X = dmatrices('np.log(price)~(year+engineSize+mileage+mpg)**2+I(mileage**2)', data = trai
np.sqrt(mean_squared_error(np.exp(cross_val_predict(LinearRegression(), X, y)), np.exp(y)))
```

```
9800.202063309154
```

Why can't we use `cross_val_score()` instead of `cross_val_predict()` here?

There seems to be a slight improvement in prediction error after removing influential points. Note that none of the points had "very high leverage", and thus the change is not substantial.

Note that we obtain a higher R-squared value of 81.5% as compared to 80% with the complete data. Removing the influential points helped obtain a slightly better model fit. However, that may also happen just by reducing observations.

```
#Computing RMSE on test data
pred_price_log = model_log.predict(testf)
np.sqrt(((testp.price - np.exp(pred_price_log))**2).mean())
```

```
8922.977452912108
```

The RMSE on test data has also reduced. This shows that some of the influential points were impacting the regression line. With those points removed, the model better captures the general trend in the data.

7.3.1 Influence on single fitted value (DFFITS)

- A useful measure of the influence that the i^{th} observation has on the fitted value $\hat{Y}_i$ is:

$$(DFFITS)_i = \frac{\hat{Y}_i - \hat{Y}_{i(i)}}{\sqrt{MSE_i h_i}} \tag{7.3}$$

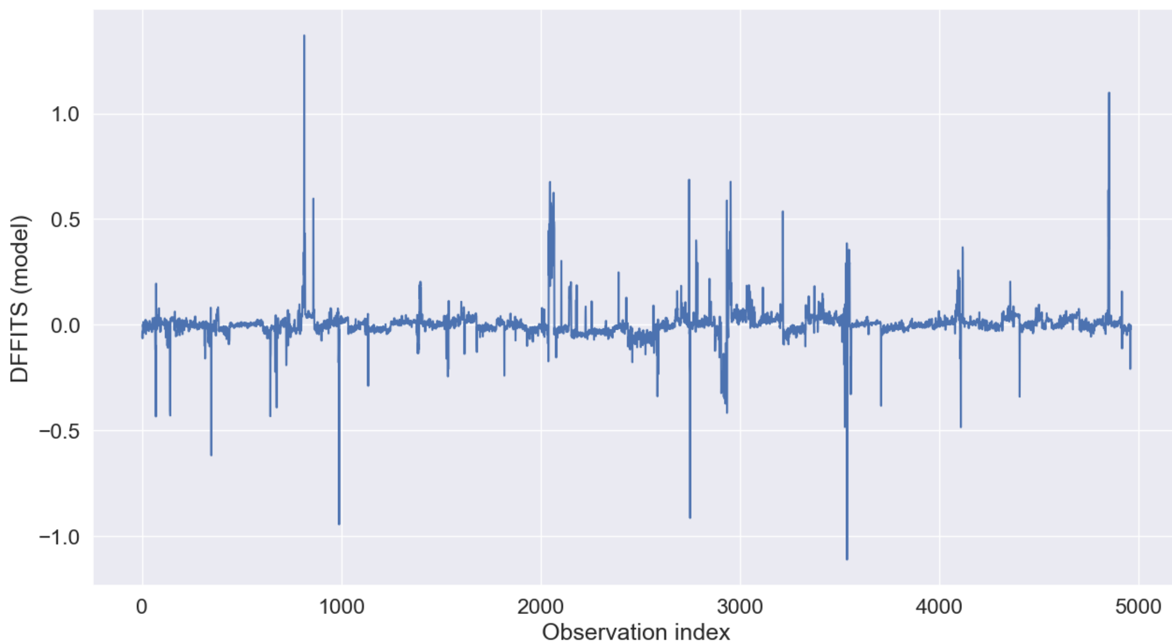
- Note that the denominator in the above fraction is the estimated standard deviation of $\hat{Y}_i$, but uses the error mean square when the i^{th} observation is omitted.
- $DFFITS$ for the i^{th} observation represents the number of estimated standard deviations of $\hat{Y}_i$ that the fitted value $\hat{Y}_i$ increases or decreases with the inclusion of the i^{th} observation in fitting the regression model.
- It can be shown that:

$$(DFFITS)_i = t_i\sqrt{\frac{h_i}{1-h_i}} \tag{7.4}$$

where t_i is the studentized deleted residual for the i^{th} observation.

- We can see that if an observation has high leverage and is an outlier, it is likely to be influential
- For large datasets, an observation is considered influential if the magnitude of $DFFITS$ for it exceeds $2\sqrt{\frac{p}{n}}$

```
sns.set(font_scale =1.5)
sns.lineplot(x = range(train.shape[0]), y = influence.dffits[0])
plt.xlabel('Observation index')
plt.ylabel('DFFITS (model)');
```



Let us analyze the point with the highest $DFFITS$.

```
np.where(influence.dffits[0]>1)
```

```
(array([ 813, 4851], dtype=int64),)
```

```
train.loc[813,:]
```

```
carID                12454
brand                   vw
model            Caravelle
year                  2012
transmission     Semi-Auto
mileage             212000
fuelType            Diesel
tax                    325
mpg                34.4424
engineSize             2.0
price                11995
Name: 813, dtype: object
```

```
train.loc[train.model == ' Caravelle','mileage'].describe()
```

```
count        65.000000
mean      25638.692308
std       42954.135726
min          10.000000
25%        3252.000000
50%        6900.000000
75%       30414.000000
max      212000.000000
Name: mileage, dtype: float64
```

```
# Prediction with model developed based on all points
ols_object = smf.ols(formula = 'np.log(price)~(year+engineSize+mileage+mpg)**2+I(mileage**2)
                     data = train)
model_log = ols_object.fit();
np.exp(model_log.predict(train.loc[[813],:]))
```

```
813    5502.647323
dtype: float64
```

```
# Prediction with model developed based on all points except the 813th point
ols_object = smf.ols(formula = 'np.log(price)~(year+engineSize+mileage+mpg)**2+I(mileage**2)
                     data = train.drop(index = 813))
model_log = ols_object.fit();
np.exp(model_log.predict(train.loc[[813],:]))
```

```
813    4581.374593
dtype: float64
```

Let us see the leverage and studentized residual for this observation.

```
# Leverage
leverage[813]
```

```
0.19038697461006687
```

```
# Studentized residual
out.student_resid[813]
```

```
2.823478041409651
```

Do you notice what may be contributing to the high influence of this point?

7.3.2 Influence on all fitted values (Cook's distance)

In contrast to $DFFITS$, which considers the influence of the i^{th} observation on the fitted value $\hat{Y}_i$, Cook's distance considers the influence of the i^{th} observation on all n the fitted values:

$$D_i = \frac{\sum_{j=1}^{n}(\hat{Y}_j - \hat{Y}_{j(i)})^2}{pMSE} \tag{7.5}$$

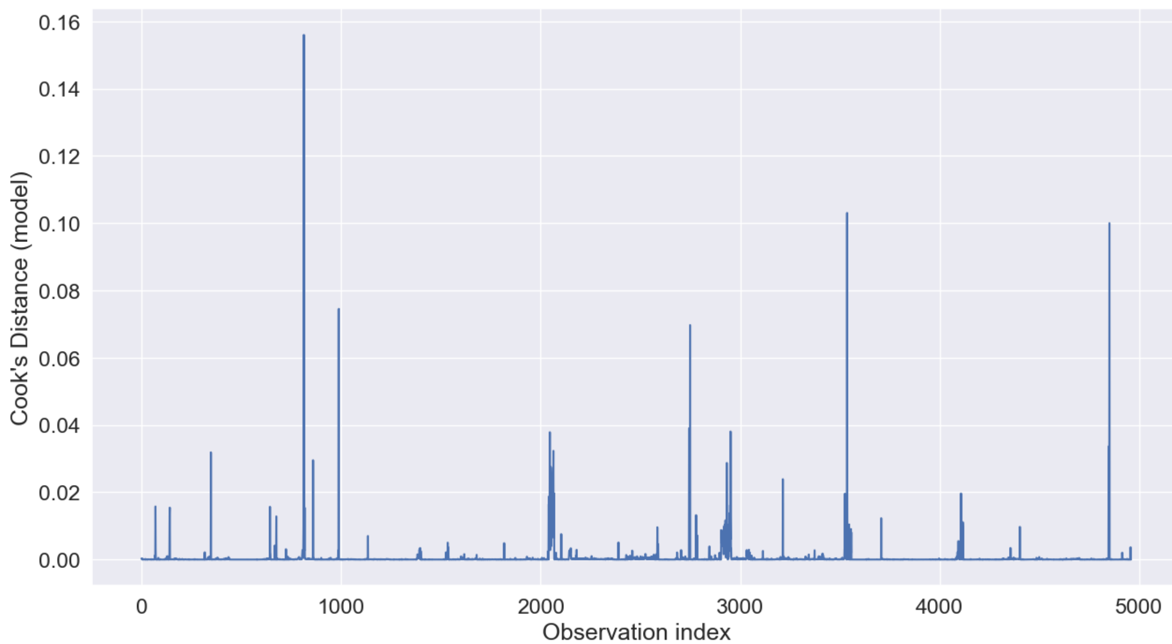
It can be shown that:

$$D_i = \frac{e_i^2}{pMSE}\left[\frac{h_i}{(1-h_i)^2}\right] \tag{7.6}$$

The larger h_i or e_i, the larger is D_i. D_i can be related to the $F(p, n-p)$ distribution. If the percentile value is 50% or more, the observation is considered as highly influential.

Cook's distance is considered high if it is greater than 0.5 and extreme if it is greater than 1.

```
sns.set(font_scale =1.5)
sns.lineplot(x = range(train.shape[0]), y = influence.cooks_distance[0])
plt.xlabel('Observation index')
plt.ylabel("Cook's Distance (model)");
```




```
# Point with the highest Cook's distance
np.where(influence.cooks_distance[0]>0.15)
```

```
(array([813], dtype=int64),)
```

The critical Cook's distance value for a point to be highly influential in this dataset is:

```
stats.f.ppf(0.5, 11, 4949)
```

```
0.9402181103263811
```

Thus, we don't have any highly influential points in the dataset.

7.3.3 Influence on regression coefficients (DFBETAS)

- $DFBETAS$ measures the influence of the i^{th} observation on the regression coefficient.
- $DFBETAS$ of the i^{th} observation on the k^{th} regression coefficient is:

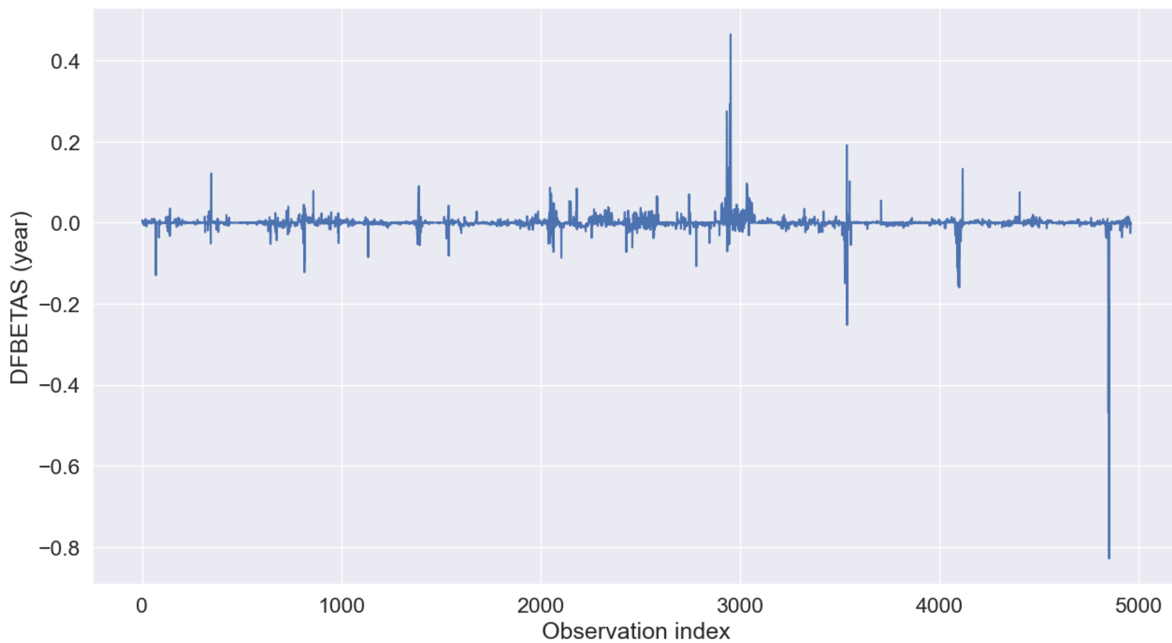
$$(DFBETAS)_{k(i)} = \frac{\hat{\beta}_k - \hat{\beta}_{k(i)}}{\sqrt{MSE_i c_k}} \tag{7.7}$$

where c_k is the k^{th} diagonal element of $(X^TX)^{-1}$.

For large datasets, an observation is considered influential if $DFBETAS$ exceeds $\frac{2}{\sqrt{n}}$.

Below is the plot of $DFBETAS$ for the `year` predictor against the observation index.

```
sns.set(font_scale =1.5)
sns.lineplot(x = range(train.shape[0]), y = influence.dfbetas[:,1])
plt.xlabel('Observation index')
plt.ylabel("DFBETAS (year)");
```



Let us analyze the point with the highest magnitude of $DFBETAS$.

```
np.where(influence.dfbetas[:,1]<-0.8)
```

```
(array([4851], dtype=int64),)
```

```
train.year.describe()
```

```
count    4960.000000
mean     2016.737903
std         2.884035
min      1997.000000
25%      2016.000000
50%      2017.000000
75%      2019.000000
max      2020.000000
Name: year, dtype: float64
```

```
train.loc[train.year<=2002,:]
```

	carID	brand	model	year	transmission	mileage	fuelType	tax	mpg	engineSize	pri
330	13200	audi	A8	1997	Automatic	122000	Petrol	265	19.3511	4.2	465
732	13988	vw	Beetle	2001	Manual	47729	Petrol	330	32.5910	2.0	249
3157	18794	ford	Puma	2002	Manual	108000	Petrol	230	38.5757	1.7	219
3525	19395	merc	S Class	2001	Automatic	108800	Diesel	325	31.5473	3.2	169
3532	17531	merc	S Class	1999	Automatic	34000	Petrol	145	24.8735	3.2	599
3533	18761	merc	S Class	2001	Automatic	66000	Petrol	570	24.7744	3.2	449
3535	18813	merc	S Class	1998	Automatic	43534	Petrol	265	23.2962	6.0	199
3536	17891	merc	S Class	2002	Automatic	24000	Petrol	570	20.7968	5.0	699
3707	18746	hyundi	Santa Fe	2002	Manual	94000	Petrol	325	30.2671	2.4	120
4091	12995	merc	SLK	1998	Automatic	113557	Petrol	265	31.8368	2.3	199
4094	19585	merc	SLK	2001	Automatic	69234	Petrol	325	30.8839	2.0	399
4096	14265	merc	SLK	2001	Automatic	48172	Petrol	325	29.7058	2.3	399
4097	15821	merc	SLK	2002	Automatic	61400	Petrol	325	29.6568	2.3	399
4098	13021	merc	SLK	2001	Automatic	91000	Petrol	325	30.3248	2.3	395
4099	12660	merc	SLK	2001	Automatic	42087	Petrol	325	29.9404	2.3	449
4101	17521	merc	SLK	2002	Automatic	75034	Petrol	325	30.1380	2.3	499
4107	13977	merc	SLK	2000	Automatic	87000	Petrol	265	27.2998	3.2	149
4108	18679	merc	SLK	2000	Automatic	113237	Petrol	270	26.8765	3.2	399
4109	14598	merc	SLK	2001	Automatic	64476	Petrol	325	27.4628	3.2	499
4847	17268	bmw	Z3	1997	Manual	49000	Petrol	270	34.9548	1.9	395
4848	12137	bmw	Z3	1999	Manual	58000	Petrol	270	35.3077	1.9	395
4849	13288	bmw	Z3	1999	Manual	74282	Petrol	245	35.4143	1.9	399
4850	19172	bmw	Z3	2001	Manual	60000	Petrol	325	30.7305	2.2	595
4851	16577	bmw	Z3	2002	Automatic	16500	Petrol	325	29.7614	2.2	149

Let us see the leverage and studentized residual for this observation.

```
# Leverage
leverage[4851]
```

```
0.047120455781282225
```

```
# Studentized residual
out.student_resid[4851]
```

```
4.938606329343604
```

Do you see what makes this point influential?

7.4 Collinearity

Collinearity refers to the situation when two or more predictor variables have a high linear association. Linear association between a pair of variables can be measured by the correlation coefficient. Thus the correlation matrix can indicate some potential collinearity problems.

7.4.1 Why and how is collinearity a problem

(Source: page 100-101 of book)

The presence of collinearity can pose problems in the regression context, since it can be difficult to separate out the individual effects of collinear variables on the response.

Since collinearity reduces the accuracy of the estimates of the regression coefficients, it causes the standard error for $\hat{\beta}_j$ to grow. Recall that the t-statistic for each predictor is calculated by dividing $\hat{\beta}_j$ by its standard error. Consequently, collinearity results in a decline in the t-statistic. As a result, **in the presence of collinearity, we may fail to reject $H_0 : \beta_j = 0$. This means that the power of the hypothesis test—the probability of correctly detecting a non-zero coefficient—is reduced by collinearity.**

7.4.2 How to measure collinearity/multicollinearity

(Source: page 102 of book)

Unfortunately, not all collinearity problems can be detected by inspection of the correlation matrix: it is possible for collinearity to exist between three or more variables even if no pair of variables has a particularly high correlation. We call this situation multicollinearity. Instead of inspecting the correlation matrix, a better way to assess multicollinearity is to compute the variance inflation factor (VIF). The VIF is variance inflation factor the ratio of the variance of $\hat{\beta}_j$ when fitting the full model divided by the variance of $\hat{\beta}_j$ if fit on its own. The smallest possible value for VIF is 1, which indicates the complete absence of collinearity. Typically in practice there is a small amount of collinearity among the predictors. As a rule of thumb, a **VIF value that exceeds 5 or 10 indicates a problematic amount of collinearity.**

The estimated variance of the coefficient β_j, of the j^{th} predictor X_j, can be expressed as:

$$v\hat{a}r(\hat{\beta}_j) = \frac{(\hat{\sigma})^2}{(n-1)v\hat{a}r(X_j)}.\frac{1}{1-R^2_{X_j|X_{-j}}},$$

where $R^2_{X_j|X_{-j}}$ is the R-squared for the regression of X_j on the other covariates (a regression that does not involve the response variable Y).

In case of simple linear regression, the variance expression in the equation above does not contain the term $\frac{1}{1-R^2_{X_j|X_{-j}}}$, as there is only one predictor. However, in case of multiple linear regression, the variance of the estimate of the j^{th} coefficient ($\hat{\beta}_j$) gets inflated by a factor of $\frac{1}{1-R^2_{X_j|X_{-j}}}$ *(Note that in the complete absence of collinearity,* $R^2_{X_j|X_{-j}} = 0$*, and the value of this factor will be 1).*

Thus, the Variance inflation factor, or the VIF for the estimated coefficient of the j^{th} predictor X_j is:

$$VIF(\hat{\beta}_j) = \frac{1}{1-R^2_{X_j|X_{-j}}} \tag{7.8}$$

```
#Correlation matrix
train.corr()
```

	carID	year	mileage	tax	mpg	engineSize	price
carID	1.000000	0.006251	-0.001320	0.023806	-0.010774	0.011365	0.012129
year	0.006251	1.000000	-0.768058	-0.205902	-0.057093	0.014623	0.501296
mileage	-0.001320	-0.768058	1.000000	0.133744	0.125376	-0.006459	-0.478705

	carID	year	mileage	tax	mpg	engineSize	price
tax	0.023806	-0.205902	0.133744	1.000000	-0.488002	0.465282	0.144652
mpg	-0.010774	-0.057093	0.125376	-0.488002	1.000000	-0.419417	-0.369919
engineSize	0.011365	0.014623	-0.006459	0.465282	-0.419417	1.000000	0.624899
price	0.012129	0.501296	-0.478705	0.144652	-0.369919	0.624899	1.000000

Let us compute the Variance Inflation Factor (VIF) for the four predictors.

```
X = train[['mpg','year','mileage','engineSize']]
```

```
X.columns[1:]
```

```
Index(['year', 'mileage', 'engineSize'], dtype='object')
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
X = add_constant(X)
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

for i in range(len(X.columns)):
    vif_data.loc[i,'VIF'] = variance_inflation_factor(X.values, i)

print(vif_data)
```

```
      feature           VIF
0       const  1.201579e+06
1         mpg  1.243040e+00
2        year  2.452891e+00
3     mileage  2.490210e+00
4  engineSize  1.219170e+00
```

As all the values of VIF are close to one, we do not have the problem of multicollinearity in the model. Note that the VIF of `year` and `mileage` is relatively high as they are the most correlated.

Q1: Why is the VIF of the constant so high?

Q2: Why do we need to include the constant while finding the VIF?

7.4.3 Manual computation of VIF

```
#Manually computing the VIF for year
ols_object = smf.ols(formula = 'price~mpg', data = train)
model_log = ols_object.fit()
model_log.summary()
```

Dep. Variable:	price	**R-squared:**	0.137
Model:	OLS	**Adj. R-squared:**	0.137
Method:	Least Squares	**F-statistic:**	786.0
Date:	Wed, 06 Mar 2024	**Prob (F-statistic):**	1.14e-160
Time:	17:04:39	**Log-Likelihood:**	-54812.
No. Observations:	4960	**AIC:**	1.096e+05
Df Residuals:	4958	**BIC:**	1.096e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	4.144e+04	676.445	61.258	0.000	4.01e+04	4.28e+04
mpg	-374.2975	13.351	-28.036	0.000	-400.471	-348.124

Omnibus:	2132.208	**Durbin-Watson:**	0.320
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	13751.995
Skew:	1.942	**Prob(JB):**	0.00
Kurtosis:	10.174	**Cond. No.**	158.

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

```
(13.351/9.338)**2
```

```
2.044183378279958
```

```
#Manually computing the VIF for year
ols_object = smf.ols(formula = 'price~year+mpg+engineSize+mileage', data = train)
model_log = ols_object.fit()
model_log.summary()
```

Dep. Variable:	price	**R-squared:**	0.660
Model:	OLS	**Adj. R-squared:**	0.660
Method:	Least Squares	**F-statistic:**	2410.
Date:	Wed, 06 Mar 2024	**Prob (F-statistic):**	0.00
Time:	17:01:18	**Log-Likelihood:**	-52497.
No. Observations:	4960	**AIC:**	1.050e+05
Df Residuals:	4955	**BIC:**	1.050e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	-3.661e+06	1.49e+05	-24.593	0.000	-3.95e+06	-3.37e+06
year	1817.7366	73.751	24.647	0.000	1673.151	1962.322
mpg	-79.3126	9.338	-8.493	0.000	-97.620	-61.006
engineSize	1.218e+04	189.969	64.107	0.000	1.18e+04	1.26e+04
mileage	-0.1474	0.009	-16.817	0.000	-0.165	-0.130

Omnibus:	2450.973	**Durbin-Watson:**	0.541
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	31060.548
Skew:	2.045	**Prob(JB):**	0.00
Kurtosis:	14.557	**Cond. No.**	3.83e+07

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.83e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

```
#Manually computing the VIF for year
ols_object = smf.ols(formula = 'year~mpg+engineSize+mileage', data = train)
model_log = ols_object.fit()
model_log.summary()
```

Dep. Variable:	year	**R-squared:**	0.592
Model:	OLS	**Adj. R-squared:**	0.592
Method:	Least Squares	**F-statistic:**	2400.
Date:	Wed, 06 Mar 2024	**Prob (F-statistic):**	0.00
Time:	17:00:13	**Log-Likelihood:**	-10066.
No. Observations:	4960	**AIC:**	2.014e+04
Df Residuals:	4956	**BIC:**	2.017e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
Intercept	2018.3135	0.140	1.44e+04	0.000	2018.039	2018.588
mpg	0.0095	0.002	5.301	0.000	0.006	0.013
engineSize	0.1171	0.037	3.203	0.001	0.045	0.189
mileage	-9.139e-05	1.08e-06	-84.615	0.000	-9.35e-05	-8.93e-05

Omnibus:	2949.664	**Durbin-Watson:**	1.161
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	63773.271
Skew:	-2.426	**Prob(JB):**	0.00
Kurtosis:	19.883	**Cond. No.**	1.91e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.91e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

```
#VIF for year
1/(1-0.592)
```

```
2.4509803921568625
```

Note that year and mileage have a high linear correlation. Removing one of them should decrease the standard error of the coefficient of the other, without significanty decrease R-squared.

```
ols_object = smf.ols(formula = 'price~mpg+engineSize+mileage+year', data = train)
model_log = ols_object.fit()
model_log.summary()
```

Table 7.6: OLS Regression Results

Dep. Variable:	price	R-squared:	0.660
Model:	OLS	Adj. R-squared:	0.660
Method:	Least Squares	F-statistic:	2410.
Date:	Tue, 07 Feb 2023	Prob (F-statistic):	0.00
Time:	21:39:45	Log-Likelihood:	-52497.
No. Observations:	4960	AIC:	1.050e+05
Df Residuals:	4955	BIC:	1.050e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-3.661e+06	1.49e+05	-24.593	0.000	-3.95e+06	-3.37e+06
mpg	-79.3126	9.338	-8.493	0.000	-97.620	-61.006
engineSize	1.218e+04	189.969	64.107	0.000	1.18e+04	1.26e+04
mileage	-0.1474	0.009	-16.817	0.000	-0.165	-0.130
year	1817.7366	73.751	24.647	0.000	1673.151	1962.322

Omnibus:	2450.973	Durbin-Watson:	0.541
Prob(Omnibus):	0.000	Jarque-Bera (JB):	31060.548
Skew:	2.045	Prob(JB):	0.00
Kurtosis:	14.557	Cond. No.	3.83e+07

Removing mileage from the above regression.

```
ols_object = smf.ols(formula = 'price~mpg+engineSize+year', data = train)
model_log = ols_object.fit()
model_log.summary()
```

Table 7.9: OLS Regression Results

Dep. Variable:	price	R-squared:	0.641
Model:	OLS	Adj. R-squared:	0.641
Method:	Least Squares	F-statistic:	2951.
Date:	Tue, 07 Feb 2023	Prob (F-statistic):	0.00
Time:	21:40:00	Log-Likelihood:	-52635.
No. Observations:	4960	AIC:	1.053e+05
Df Residuals:	4956	BIC:	1.053e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-5.586e+06	9.78e+04	-57.098	0.000	-5.78e+06	-5.39e+06
mpg	-101.9120	9.500	-10.727	0.000	-120.536	-83.288
engineSize	1.196e+04	194.848	61.392	0.000	1.16e+04	1.23e+04
year	2771.1844	48.492	57.147	0.000	2676.118	2866.251

Omnibus:	2389.075	Durbin-Watson:	0.528
Prob(Omnibus):	0.000	Jarque-Bera (JB):	26920.051
Skew:	2.018	Prob(JB):	0.00
Kurtosis:	13.675	Cond. No.	1.41e+06

Note that the standard error of the coefficient of *year* has reduced from 73 to 48, without any large reduction in R-squared.

7.4.4 When can we overlook multicollinearity?

- The severity of the problems increases with the degree of the multicollinearity. Therefore, if there is only moderate multicollinearity *(5 < VIF < 10)*, we may overlook it.
- Multicollinearity affects only the standard errors of the coefficients of collinear predictors. Therefore, if multicollinearity is not present for the predictors that we are particularly interested in, we may not need to resolve it.
- Multicollinearity affects the standard error of the coefficients and thereby their p-values, but in general, it does not influence the prediction accuracy, except in the case that the coefficients are so unstable that the predictions are outside of the domain space of the response. If our sole aim is prediction, and we don't wish to infer the statistical significance of predictors, then we may avoid addressing multicollinearity. "*The fact that some or all predictor variables are correlated among themselves does not, in general, inhibit our ability to obtain a good fit nor does it tend to affect inferences about mean responses or predictions of new observations, provided these inferences are made within the region of observations" - Neter, John, Michael H. Kutner, Christopher J. Nachtsheim, and William Wasserman. "Applied linear statistical models." (1996): 318.*

Part II

Assignments

8 Assignment 1 (Section 20)

Instructions

1. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.
2. Do not write your name on the assignment.
3. Write your code in the **Code cells** and your answers in the **Markdown cells** of the Jupyter notebook. Ensure that the solution is written neatly enough to for the graders to understand and follow.
4. Use Quarto to render the **.ipynb** file as HTML. You will need to open the command prompt, navigate to the directory containing the file, and use the command: `quarto render filename.ipynb --to html`. Submit the HTML file.
5. The assignment is worth 100 points, and is due on **Wednesday, 24th January 2024 at 11:59 pm**.
6. There is a **bonus** question worth 15 points.
7. **Five points are properly formatting the assignment**. The breakdown is as follows:
   - Must be an HTML file rendered using Quarto **(1 point)**. *If you have a Quarto issue, you must mention the issue & quote the error you get when rendering using Quarto in the comments section of Canvas, and submit the ipynb file.*
   - No name can be written on the assignment, nor can there be any indicator of the student's identity—e.g. printouts of the working directory should not be included in the final submission. **(1 point)**
   - There aren't excessively long outputs of extraneous information (e.g. no printouts of entire data frames without good reason, there aren't long printouts of which iteration a loop is on, there aren't long sections of commented-out code, etc.) **(1 point)**
   - Final answers to each question are written in the Markdown cells. **(1 point)**
   - There is no piece of unnecessary / redundant code, and no unnecessary / redundant text. **(1 point)**

8. The maximum possible score in the assigment is 100 + 15 (bonus question) + 5 (proper formatting) = 120 out of 100. There is no partial credit for some parts of the bonus question.

8.1 1) Case Studies: Regression vs Classification and Prediction vs Inference (16 points)

8.1.1 1a)

For each case below, explain (1) whether it is a classification or a regression problem and (2) whether the main purpose is prediction or inference. **You need justify your answers for credit.**

8.1.2 1b)

You work for a company that is interested in conducting a marketing campaign. The goal of your project is to identify individuals who are likely to respond positively to a marketing campaign, based on observations of demographic variables (such as age, gender, income etc.) measured on each individual. **(2+2 points)**

8.1.3 1c)

For the same company, now you are working on a different project. This one is focused on understanding the impact of advertisements in different media types on the company sales. For example, you are interested in the following question: *'How large of an increase in sales is associated with a given increase in radio and TV advertising?'* **(2+2 points)**

8.1.4 1d)

A company is selling furniture and they are interested in the finding the association between demographic characteristics of customers (such as age, gender, income etc.) and if they would purchase a particular company product. **(2+2 points)**

8.1.5 1e)

We are interested in forecasting the % change in the USD/Euro exchange rate using the weekly changes in the stock markets of a number of countries. We collect weekly data for all of 2023. For each week, we record the % change in the USD/Euro, the % change in the US market, the % change in the British market, and the % change in the German market. **(2+2 points)**

8.2 2) Examples for Different Regression Metrics: RMSE vs MAE (8 points)

8.2.1 2a)

Describe a regression problem, where it will be more proper to evaluate the model performance using the root mean squared error (RMSE) metric as compared to the mean absolute error (MAE) metric. **You need to justify your answer for credit. (4 points)**

Note: You are not allowed to use the datasets and examples covered in the lectures.

8.2.2 2b)

Describe a regression problem, where it will be more proper to evaluate the model performance using the mean absolute error (MAE) metric as compared to the root mean squared error (RMSE) metric. **You need to justify your answer for credit. (4 points)**

Note: You are not allowed to use the datasets and examples covered in the lectures.

8.3 3) Modeling the Petrol Consumption in U.S. States (61 points)

Read **petrol_consumption_train.csv**. Assume that each observation is a U.S. state. For each observation, the data has the following variables as its five columns:

`Petrol_tax`: Petrol tax (cents per gallon)

`Per_capita_income`: Average income (dollars)

`Paved_highways`: Paved Highways (miles)

`Prop_license`: Proportion of population with driver's licenses

`Petrol_consumption`: Consumption of petrol (millions of gallons)

8.3.1 3a)

Create a pairwise plot of all the variables in the dataset. **(1 point)** Print the correlation matrix of all the variables as well. **(1 point)** Which variable has the highest linear correlation with `Petrol_consumption`? **(2 points)**

Note: Remember that a pairwise plot is a visualization tool that you can find in the seaborn library.

8.3.2 3b)

Fit a simple linear regression model to predict `Petrol_consumption` using the column you found in **part a** as the only predictor. Print the model summary. **(4 points)**

8.3.3 3c)

What is the increase in petrol consumption for an increase of 0.05 in the predictor? **(4 points)**

8.3.4 3d)

Does petrol consumption have a statistically significant relationship with the predictor? **You need to justify your answer for credit**. **(4 points)**

8.3.5 3e)

How much of the variation in petrol consumption can be explained by its linear relationship with the predictor? **(3 points)**

8.3.6 3f)

Predict the petrol consumption for a state in which 50% of the population has a driver's license. **(3 points)** What are the confidence interval **(3 points)** and the prediction interval **(3 points)** for your prediction? Which interval is wider? **(1 points)** Why? **(2 points)**

8.3.7 3g)

Predict the petrol consumption for a state in which 10% of the population has a driver's license. **(3 points)** Are you getting a reasonable outcome? **(1 point)** Why or why not? **(2 points)**

8.3.8 3h)

What is the residual standard error of the model? **(3 points)**

8.3.9 3i)

Using the trained model, predict the petrol consumption of the observations in **petrol_consumption_test.csv** **(2 points)** and find the RMSE. **(2 points)** What is the unit of this RMSE value? **(1 point)**

8.3.10 3j)

Based on the answers to **part g** and **part h**, do you think the model is overfitting? **You need to justify your answer for credit. (4 points)**

8.3.11 3k)

Make a scatterplot of `Petrol_consumption` vs. the predictor using **petrol_consumption_test.csv**. **(1 point)** Over the scatterplot, plot the regression line **(2 points)**, the prediction interval **(2 points)**, and the confidence interval. **(2 points)**

Make sure that regression line, prediction interval lines, and confidence interval lines have different colors. **(1 point)** Display a legend that correctly labels the lines as well. **(1 point)** Note that you need two lines of the same color to plot an interval.

8.3.12 3l)

Find the correlation between `Petrol_consumption` and the rest of the variables in **petrol_consumption_train.csv**. Which column would have the lowest R-squared value when used as the predictor for a Simple Linear Regression model to predict `Petrol_consumption`? Note that you can directly answer this question from the correlation values and do not need to develop any more linear regression models. **(3 points)**

8.4 4) Reproducing the Results with Scikit-Learn (15 points)

8.4.1 4a)

Using the same datasets, same response and the same predictor as **Question 3**, reproduce the following outputs with scikit-learn:

- Model RMSE for test data **(3 points)**
- R-squared value of the model **(3 points)**
- Residual standard error of the model **(3 points)**

Note that you are only allowed to use scikit-learn, pandas, and numpy tools for this question. Any other libraries will not receive any credit.

8.4.2 4b)

Which of the model outputs from **Question 3** cannot be reproduced using scikit-learn? Give two answers. **(2+2 points)** What does this tell about scikit-learn? **(2 points)**

8.5 5) Bonus Question (15 points)

Please note that the bonus question requires you to look more into the usage of the tools we covered in class and it will be necessary to do your own research. We strongly suggest attempting it after you are done with the rest of the assignment.

8.5.1 5a)

Fit a simple linear regression model to predict `Petrol_consumption` based on the predictor in **Question 3**, but **without an intercept term**. **(5 points - no partial credit)**

Without an intercept means that the equation becomes $Y = \beta_1 X$. The intercept term, β_0, becomes 0.

Note: You must answer this part correctly to qualify for the bonus points in the following parts.

8.5.2 5b)

Predict the petrol consumption for the observations in **petrol_consumption_test.csv** using the model without an intercept and find the RMSE. **(1+2 points)** Then, print the summary and find the R-squared. **(2 points)**

8.5.3 5c)

The RMSE for the models with and without the intercept are similar, which indicates that both models are almost equally good. However, the R-squared for the model without intercept is much higher than the R-squared for the model with the intercept. Why? Justify your answer. **(5 points - no partial credit)**

9 Assignment 1 (Sections 21 & 22)

Instructions

1. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.
2. Do not write your name on the assignment.
3. Write your code in the **Code cells** and your answers in the **Markdown cells** of the Jupyter notebook. Ensure that the solution is written neatly enough to for the graders to understand and follow.
4. Use Quarto to render the **.ipynb** file as HTML. You will need to open the command prompt, navigate to the directory containing the file, and use the command: `quarto render filename.ipynb --to html`. Submit the HTML file.
5. The assignment is worth 100 points, and is due on **Wednesday, 24th January 2024 at 11:59 pm**.
6. There is a **bonus** question worth 15 points.
7. **Five points are properly formatting the assignment**. The breakdown is as follows:
   - Must be an HTML file rendered using Quarto **(1 point)**. *If you have a Quarto issue, you must mention the issue & quote the error you get when rendering using Quarto in the comments section of Canvas, and submit the ipynb file.*
   - No name can be written on the assignment, nor can there be any indicator of the student's identity—e.g. printouts of the working directory should not be included in the final submission. **(1 point)**
   - There aren't excessively long outputs of extraneous information (e.g. no printouts of entire data frames without good reason, there aren't long printouts of which iteration a loop is on, there aren't long sections of commented-out code, etc.) **(1 point)**
   - Final answers to each question are written in the Markdown cells. **(1 point)**
   - There is no piece of unnecessary / redundant code, and no unnecessary / redundant text. **(1 point)**

8. The maximum possible score in the assigment is 100 + 15 (bonus question) + 5 (proper formatting) = 120 out of 100. There is no partial credit for some parts of the bonus question.

9.1 1) Case Studies: Regression vs Classification and Prediction vs Inference (16 points)

For each case below, explain (1) whether it is a classification or a regression problem and (2) whether the main purpose is prediction or inference. **You need justify your answers for credit.**

9.1.1 1a)

You work for a company that is interested in conducting a marketing campaign. The goal of your project is to identify individuals who are likely to respond positively to a marketing campaign, based on observations of demographic variables (such as age, gender, income etc.) measured on each individual. **(2+2 points)**

9.1.2 1b)

For the same company, now you are working on a different project. This one is focused on understanding the impact of advertisements in different media types on the company sales. For example, you are interested in the following question: *'How large of an increase in sales is associated with a given increase in radio and TV advertising?'* **(2+2 points)**

9.1.3 1c)

A company is selling furniture and they are interested in the finding the association between demographic characteristics of customers (such as age, gender, income etc.) and if they would purchase a particular company product. **(2+2 points)**

9.1.4 1d)

We are interested in forecasting the % change in the USD/Euro exchange rate using the weekly changes in the stock markets of a number of countries. We collect weekly data for all of 2023. For each week, we record the % change in the USD/Euro, the % change in the US market, the % change in the British market, and the % change in the German market. **(2+2 points)**

9.2 2) Examples for Different Regression Metrics: RMSE vs MAE (8 points)

9.2.1 2a)

Describe a regression problem, where it will be more proper to evaluate the model performance using the root mean squared error (RMSE) metric as compared to the mean absolute error (MAE) metric. **You need to justify your answer for credit. (4 points)**

Note: You are not allowed to use the datasets and examples covered in the lectures.

9.2.2 2b)

Describe a regression problem, where it will be more proper to evaluate the model performance using the mean absolute error (MAE) metric as compared to the root mean squared error (RMSE) metric. **You need to justify your answer for credit. (4 points)**

Note: You are not allowed to use the datasets and examples covered in the lectures.

9.3 3) Simple Linear Regression: Formulation (3 points)

When asked to state the simple linear regression model, a students wrote it as follows: $E(Y_i) = \beta_0 + \beta_1 X_i + \epsilon_i$. Is this correct **(1 point)**? Justify your answer **(2 points)**.

9.4 4) Modeling the Petrol Consumption in U.S. States (58 points)

Read **petrol_consumption_train.csv**. Assume that each observation is a U.S. state. For each observation, the data has the following variables as its five columns:

`Petrol_tax`: Petrol tax (cents per gallon)

`Per_capita_income`: Average income (dollars)

`Paved_highways`: Paved Highways (miles)

`Prop_license`: Proportion of population with driver's licenses

`Petrol_consumption`: Consumption of petrol (millions of gallons)

9.4.1 4a)

Create a pairwise plot of all the variables in the dataset. **(1 point)** Print the correlation matrix of all the variables as well. **(1 point)** Which variable has the highest linear correlation with `Petrol_consumption`? **(1 point)**

Note: Remember that a pairwise plot is a visualization tool that you can find in the seaborn library.

9.4.2 4b)

Fit a simple linear regression model to predict `Petrol_consumption` using the column you found in **part a** as the only predictor. Print the model summary. **(3 points)**

9.4.3 4c)

When asked for a point estimate of the expected petrol consumption for a state where the proportion of population with driver's license is 54.4%, a person gave the estimate 488 million gallons because that is the mean value of `Petrol_consumption` for the two observations of `Prop_license = 0.544` pieces in the dataset. Is there an issue with this approach? Explain. **(2 points)** If there is an issue, then suggest a better approach and use it to estimate the expected petrol consumption for a state where the proportion of population with driver's license is 54.4%. **(2 points)**

9.4.4 4d)

What is the increase in petrol consumption for an increase of 0.05 in the predictor? **(3 points)**

9.4.5 4e)

Does petrol consumption have a statistically significant relationship with the predictor? **You need to justify your answer for credit**. **(3 points)**

9.4.6 4f)

How much of the variation in petrol consumption can be explained by its linear relationship with the predictor? **(2 points)**

9.4.7 4g)

Predict the petrol consumption for a state in which 50% of the population has a driver's license. **(2 points)** What are the confidence interval **(2 points)** and the prediction interval **(2 points)** for your prediction? Which interval is wider? **(1 points)** Why? **(2 points)**

9.4.8 4h)

Predict the petrol consumption for a state in which 10% of the population has a driver's license. **(3 points)** Are you getting a reasonable outcome? **(1 point)** Why or why not? **(2 points)**

9.4.9 4i)

What is the residual standard error of the model? **(3 points)**

9.4.10 4j)

Using the trained model, predict the petrol consumption of the observations in **petrol_consumption_test.csv** **(2 points)** and find the RMSE. **(2 points)** What is the unit of this RMSE value? **(1 point)**

9.4.11 4k)

Based on the answers to **part i** and **part j**, do you think the model is overfitting? **You need to justify your answer for credit. (3 points)**

9.4.12 4l)

Make a scatterplot of `Petrol_consumption` vs. the predictor using **petrol_consumption_test.csv**. **(1 point)** Over the scatterplot, plot the regression line **(1 point)**, the prediction interval **(2 points)**, and the confidence interval. **(2 points)**

Make sure that regression line, prediction interval lines, and confidence interval lines have different colors. **(1 point)** Display a legend that correctly labels the lines as well. **(1 point)** Note that you need two lines of the same color to plot an interval.

9.4.13 4m)

The dataset consists of 40 US States. If you combine this data with the data of the remaining 10 US States, are you likely to obtain narrower confidence and prediction intervals in the plot developed in the previous question, for the same level of confidence? Justify your answer. **(2 points)**.

If yes, then can you gaurantee that the width of these intervals will reduce? Justify your answer. If no, then can you gaurantee that the width of these intervals will not reduce? Justify your answer. **(2 points)**

9.4.14 4n)

Find the correlation between `Petrol_consumption` and the rest of the variables in **petrol_consumption_train.csv**. Which column would have the lowest R-squared value when used as the predictor for a Simple Linear Regression model to predict `Petrol_consumption`? Note that you can directly answer this question from the correlation values and do not need to develop any more linear regression models. **(2 points)**

9.5 5) Reproducing the Results with Scikit-Learn (15 points)

9.5.1 5a)

Using the same datasets, same response and the same predictor as **Question 4**, reproduce the following outputs with scikit-learn:

- Model RMSE for test data **(3 points)**
- R-squared value of the model **(3 points)**
- Residual standard error of the model **(3 points)**

Note that you are only allowed to use scikit-learn, pandas, and numpy tools for this question. Any other libraries will not receive any credit.

9.5.2 5b)

Which of the model outputs from **Question 4** cannot be reproduced using scikit-learn? Give two answers. **(2+2 points)** What does this tell about scikit-learn? **(2 points)**

9.6 6) Bonus Question (15 points)

Please note that the bonus question requires you to look more into the usage of the tools we covered in class and it will be necessary to do your own research. We strongly suggest attempting it after you are done with the rest of the assignment.

9.6.1 6a)

Fit a simple linear regression model to predict `Petrol_consumption` based on the predictor in **Question 4**, but **without an intercept term**. **(5 points - no partial credit)**

Without an intercept means that the equation becomes $Y = \beta_1 X$. The intercept term, β_0, becomes 0.

Note: You must answer this part correctly to qualify for the bonus points in the following parts.

9.6.2 6b)

Predict the petrol consumption for the observations in **petrol_consumption_test.csv** using the model without an intercept and find the RMSE. **(1+2 points)** Then, print the summary and find the R-squared. **(2 points)**

9.6.3 6c)

The RMSE for the models with and without the intercept are similar, which indicates that both models are almost equally good. However, the R-squared for the model without intercept is much higher than the R-squared for the model with the intercept. Why? Justify your answer. **(5 points - no partial credit)**

10 Assignment 2

Instructions

1. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.

2. Write your code in the **Code cells** and your answers in the **Markdown cells** of the Jupyter notebook. Ensure that the solution is written neatly enough to for the graders to understand and follow.

3. Use Quarto to render the **.ipynb** file as HTML. You will need to open the command prompt, navigate to the directory containing the file, and use the command: `quarto render filename.ipynb --to html`. Submit the HTML file.

4. The assignment is worth 100 points, and is due on **Sunday, 4th February 2024 at 11:59 pm**.

5. **Five points are properly formatting the assignment**. The breakdown is as follows:

   - Must be an HTML file rendered using Quarto **(1 point)**. *If you have a Quarto issue, you must mention the issue & quote the error you get when rendering using Quarto in the comments section of Canvas, and submit the ipynb file.*
   - No name can be written on the assignment, nor can there be any indicator of the student's identity—e.g. printouts of the working directory should not be included in the final submission. **(1 point)**
   - There aren't excessively long outputs of extraneous information (e.g. no printouts of entire data frames without good reason, there aren't long printouts of which iteration a loop is on, there aren't long sections of commented-out code, etc.) **(1 point)**
   - Final answers to each question are written in the Markdown cells. **(1 point)**
   - There is no piece of unnecessary / redundant code, and no unnecessary / redundant text. **(1 point)**

6. The maximum possible score in the assigment is 105 + 5 (proper formatting) = 110 out of 100.

10.1 1) Multiple Linear Regression (24 points)

A study was conducted on 97 male patients with prostate cancer who were due to receive a radical prostatectomy (complete removal of the prostate). The **prostate.csv** file contains data on 9 measurements taken from these 97 patients. Each row (observation) represents a patient and each column (variable) represents a measurement. The description of variables can be found here: https://rafalab.github.io/pages/649/prostate.html

10.1.1 1a)

Fit a linear regression model with `lpsa` as the response and all the other variables as the predictors. Print its summary. **(2 points)** Write down the **optimal equation** that predicts `lpsa` using the predictors. **(2 points)**

10.1.2 1b)

Is the **overall regression** statistically significant? In other words, is there a statistically significant relationship between the response and at least one predictor? **You need to justify your answer for credit. (2 points)**

10.1.3 1c)

What does the optimal coefficient of `svi` tell us as a numeric output? Make sure you include the predictor, (`svi`) the response (`lpsa`) and the other predictors in your answer. **(2 points)**

10.1.4 1d)

Check the p-values of `gleason` and `age`. Are these predictors statistically significant? **You need to justify your answer for credit. (2 points)**

10.1.5 1e)

Check the 95% Confidence Interval of `age`. How can you relate it to its p-value and statistical significance, which you found in the previous part? **(2 points)**

10.1.6 1f)

This question requires some thinking, and bringing your 303-1 and 303-2 knowledge together.

Fit a **simple** linear regression model on `lpsa` against `gleason` and check the p-value of `gleason` using the summary. **(2 point)** Did the statistical significance of `gleason` change in the absence of other predictors? **(1 point)** Why or why not? **(3 points)**

Hints:

1) You need to compare this model with the Multiple Linear Regression model you created above.
2) Printing a correlation matrix of all the predictors should be useful.

10.1.7 1g)

Predict the `lpsa` of a **65 year old** man with `lcavol` = 1.35, `lweight` = 3.65, `lbph` = 0.1, `svi` = 0.22, `lcp` = -0.18, `gleason` = 6.75, and `pgg45` = 25. Find the 95% confidence and prediction intervals as well. **(2 points)**

10.1.8 1h)

In the Multiple Linear Regression model with all the predictors, you should see a total of five predictors that appear to be statistically insignificant. Why is it not a good idea to directly conclude that all of them are statistically insignificant? **(2 points)** Implement the additional test that concludes the statistical insignificance of all five predictors. **(2 points)**

Hint: `f_test()` method

10.2 2) Multiple Linear Regression with Variable Transformations (22 points)

The **infmort.csv** file has the infant mortality data of different countries in the world. The `mortality` column represents the infant mortality rate with "deaths per 1000 infants" as the unit. The `income` column represents the per capita income in USD. The other columns should be self-explanatory. (This is an old dataset, as can be seen from some country names.)

10.2.1 2a)

Start your analysis by creating (i) a boxplot of log(`mortality`) for each `region` and (ii) a boxplot of `income` for each `region`. Note that the `region` column has the continent names. **(3 points)**

Note: You need to use np.log, which is the natural log. This is to better distinguish the mortality values.

10.2.2 2b)

In the previous part, you should see that Europe has the lowest infant mortality rate on average, but it also has the highest per capita income on average. Our goal is to see if Europe still has the lowest mortality rate if we remove the effect of income. We will try to find an answer for the rest of this question.

Create four scatter plots: (i) `mortality` against `income`, (ii) log(`mortality`) against `income`, (iii) `mortality` against log(`income`), and (iv) log(`mortality`) against log(`income`). **(3 points)** Based on the plots, create an appropriate model to predict the mortality rate as a function of per capita income. Print the model summary. **(2 points)**

10.2.3 2c)

Update the model you created in the previous part by adding `region` as a predictor. Print the model summary. **(2 points)**

10.2.4 2d)

Use the model developed in the previous part to compute a new `adjusted_mortality` variable for each observation in the data. **(5 points)** Adjusted mortality rate is the mortality rate after removing the estimated effect of income. You need to calculate it with the following steps:

- Multiply the (transformed) income column with its optimal coefficient. This is the estimated effect of income.
- Subtract the product from the (transformed) response column. This removes the estimated effect of income.
- You may need to do a inverse transformation to calculate the actual adjusted mortality rate values.

Make a boxplot of log(`adjusted_mortality`) for each `region`. **(2 points)**

10.2.5 2e)

Using the plots in parts **a** and **d**, answer the following questions:

(i) Does Europe still have the lowest mortality rate on average after removing the effect of income?

(ii) How did the distribution of values among different continents change after removing the effect of income? How did the comparison of different continents change? Does any non-European country have a lower mortality rate than all the European countries after removing the effect of income?

(5 points)

10.3 3) Variable Transformations and Interactions (38 points)

The **soc_ind.csv** dataset contains many social indicators of a number of countries. Each row is a country and each column is a social indicator. The column names should be clear on what the variables represent. The GDP per capita will be the response variable throughout this question.

10.3.1 3a)

Using correlations, find out the most useful predictor for a simple linear regression model with `gdpPerCapita` as the response. You can ignore categorical variables for now. Let that predictor be P. **(2 points)**

10.3.2 3b)

Create a scatterplot of `gdpPerCapita` vs P. Does the relationship between `gdpPerCapita` and P seem linear or non-linear? **(2 points)**

10.3.3 3c)

If the relationship in the previous part is non-linear, create three models:

- Only with P
- With P and its quadratic term
- With P, its quadratic term and its cubic term

(2x3 = 6 points)

Compare the R-squared values of the models. **(2 points)**

10.3.4 3d)

On the same figure:

- create the scatterplot in part b.
- plot the linear regression line (only using P)
- plot the polynomial regression curve that includes the quadratic and cubic terms.
- add a legend to distinguish the linear and cubic fits.

(6 points)

10.3.5 3e)

Develop a model to predict `gdpPerCapita` using P and `continent` as predictors. (No higher-order terms.)

1. Which continent creates the baseline? **(2 points)** Write down its equation. **(2 points)**
2. For a given value of P, are there any continents that **do not** have a statistically significant difference of predicted `gdpPerCapita` from the baseline continent? If yes, then which ones, and why? If no, then why not? You need to justify your answers for credit. **(4 points)**

10.3.6 3f)

The model developed in the previous part has a limitation. It assumes that the increase in predicted `gdpPerCapita` with a unit increase in P does not depend on the `continent`.

Eliminate this limitation by including the interaction of `continent` with P in the model. Print the model summary of the model with interactions. **(2 points)** Which continent has the closest increase in predicted `gdpPerCapita` to the baseline continent with a unit increase in P. Which continent has the furthest? **You need to justify your answers for credit. (5 points)**

10.3.7 3g)

Using the model developed in the previous part, plot the regression lines of all the continents on the same figure. Put `gdpPerCapita` on the y-axis and P on the x-axis. **(4 points)** Use a legend to color-code the continents. **(1 point)**

10.4 4) Prediction with Sklearn (21 points)

Using the **soc_ind.csv** dataset and **only sklearn and pandas**, train a Linear Regression model. You need the following steps:

- `gdpPerCapita` is the response. **(2 points)**
- `Index`, `geographic_location` and `country` columns are not necessary. **(2 points)**
- All the remaining columns are predictors. **(2 points)**
- `continent` column needs to be one-hot-encoded. **(2 points)**
- Since the numeric values have different orders of magnitude, you need to scale the dataset. You can use StandardScaler from sklearn.preprocessing for this. Create an object (just like a model) and use .fit_transform with the data as the input. **(4 points)**
- Train a LinearRegression model. Use the entire dataset as the training data. **(3 points)**
- Get the predictions for the training data. **(3 points)**
- Calculate the RMSE and MAE. **(3 points)**

For this question, you only need to calculate the training performance. In the future, we will see how to split a dataset into training and test sets.

11 Assignment 3 (Section 20)

Instructions

1. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.
2. Write your code in the **Code cells** and your answers in the **Markdown cells** of the Jupyter notebook. Ensure that the solution is written neatly enough to for the graders to understand and follow.
3. Use Quarto to render the **.ipynb** file as HTML. You will need to open the command prompt, navigate to the directory containing the file, and use the command: `quarto render filename.ipynb --to html`. Submit the HTML file.
4. The assignment is worth 100 points, and is due on **Tuesday, 20th February 2024 at 11:59 pm**.
5. There is a **bonus** question worth 12 points.
6. **Five points are properly formatting the assignment**. The breakdown is as follows:
   - Must be an HTML file rendered using Quarto **(1 point)**. *If you have a Quarto issue, you must mention the issue & quote the error you get when rendering using Quarto in the comments section of Canvas, and submit the ipynb file.*
   - No name can be written on the assignment, nor can there be any indicator of the student's identity—e.g. printouts of the working directory should not be included in the final submission. **(1 point)**
   - There aren't excessively long outputs of extraneous information (e.g. no printouts of entire data frames without good reason, there aren't long printouts of which iteration a loop is on, there aren't long sections of commented-out code, etc.) **(1 point)**
   - Final answers to each question are written in the Markdown cells. **(1 point)**
   - There is no piece of unnecessary / redundant code, and no unnecessary / redundant text. **(1 point)**
7. The maximum possible score in the assigment is 103 + 12 + 5 = 120 out of 100.

11.1 Introduction (0 points)

Read the **train.csv**, **test1.csv**, and **test2.csv**. All datasets are about direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls, where bank clients were called to subscribe for a term deposit. Each observation is a phone call and each column is a variable about the client or the phone call. The columns are described as follows:

1. `age`: Age of the client
2. `education`: Education level of the client
3. `day`: Day of the month the call is made
4. `month`: Month of the call
5. `y`: did the client subscribe to a term deposit? (This is the classification response.)
6. `duration`: Call duration, in seconds. This variable highly affects the output (e.g., if duration=0 then y='no'). Yet, the `duration` is not known before a call is performed. Also, after the end of the call `y` is usually known. Therefore, it is a better idea that this variable should only be used for inference purposes and should be discarded if the intention is to have a realistic predictive model.

(Source: UCI Data Archive. Please use the given datasets for the assignment, not the raw data from the source. It is just for reference.)

11.2 1) Investigating the Effect of Call Duration on Subscription Probability (44 points)

11.2.1 1a)

First of all, you need a numeric response value for your statsmodels functions. (Numeric, not Boolean!) Convert the response column, **y**, of the training data (**train.csv**) into 0s for 'no' and 1s for 'yes'. **(2 points)**

11.2.2 1b)

Using the training data and **statsmodels**, train a logistic regression model to predict if the client subscribed to a term deposit using the call **duration**. Print its summary. **(2 points)**

You need to use this model to answer all the remaining parts of this question.

11.2.3 1c)

Is the effect of the call duration on the probability of the client subscription statistically significant? Justify your answer. **(2 points)**

11.2.4 1d)

What is the probability of the client subscribing after a 5-minute marketing call? (Note that the duration variable is in given in seconds.) **(3 points)**

11.2.5 1e)

How many **minutes** are necessary to have at least 95% probability of the client subscribing? Print it in the following format: **"... minutes or higher" (5 points: 4 points for the calculation, 1 point for formatting)**

11.2.6 1f)

What is the percentage increase in the odds of a client subscribing when the call duration increases by a **minute**? **(4 points)**

11.2.7 1g)

How many **minutes** need to be added to a call to double the odds of a client subscribing? **(3 points)**

11.2.8 1h)

After exploring the model coefficients, it is time to see the limitations of this rather simplistic model.

What is the maximum call duration (in minutes) after which the client refused to subscribe? (Note that you need the dataset itself for this question, not the model.) **(2 points)** What is the subscription probability that the model predicts for that client? **(2 points)**

11.2.9 1i)

Use a scatterplot to visualize the data. You need to plot the classes against the duration values. **(2 points)** Add some small random noise (also called jitter) on the class values to visualize as many observations as possible. **(1 point)** On top of that, add the curve that the model fits to this data. **(2 points)** You should see a sigmoid fit without its bottom end. Does it look like the duration of the call is enough by itself to predict the client subscription? **(1 point)** Why or why not? **(1 point)**

11.2.10 1j)

Predict the accuracy and recall using a threshold of 0.5 for **test_data1.csv** and **test_data2.csv**. You should print 4 numbers. **(4 points)** You should see very different values for the different metrics. Explain why this is happening and which metric is a better evaluation of the prediction performance. **(3 points)**

Hint: Checking the value counts of the response variable might be a good idea.

11.2.11 1k)

Repeat the previous question with a threshold of 0.3. Did the accuracy change much? How about recall? Explain why the results changed (or not) in terms of the confusion matrix elements. **(5 points: 1 point for the calculation, 2 points for the recall explanation, 2 points for the accuracy explanation)**

11.3 2) Exploring Variable Interactions (10 points)

11.3.1 2a)

Using the training data and statsmodels, train a logistic regression model to predict if the client subscribed to a term deposit using the **education** level and the **age**. Assume that the effect of age on the log-odds **depends on** the education level of the client. Print the summary. **(2 points)**

You need to use this model to answer all the remaining parts of this question.

11.3.2 2b)

People with which level of education have the highest percentage increase in odds of a client subscribing with a unit increase in age? Justify your answer. **(4 points)**

11.3.3 2c)

What is the maximum age of a client with tertiary education to have 15% subscription probability or lower? **(4 points)**

11.4 3) Model Development and Evaluation (29 points)

11.4.1 3a)

Using the training data and statsmodels, train a logistic regression model to predict if the client subscribed to a term deposit using **age, education, day, and month.** The model must have:

- A minimum of 75.0% accuracy for all three datasets. (train.csv, test1.csv, and test2.csv) **(3 points)**
- A minimum of 50.0% recall for all three datasets. **(6 points)**

Print the model summary. **(1 point)** For all three datasets, print the accuracy scores, recall scores and confusion matrices. **(3 points)**

Notes:

1) You cannot use duration as a predictor. The reason is explained in the description of the dataset in Introduction. **(No credit from the entire question for models that use duration.)**
2) Explore some interactions and transformations. You do not need to go too high. (Still ok if you do and pass the given cutoffs.)
3) You are free to choose the decision threshold as you wish. However, you must use the same threshold for all three datasets. **(No credit from the entire question for using different thresholds.)**
4) No rounding. (For example, a recall of 49.9% is not considered correct.)

You need to use this model (and threshold, unless stated otherwise) to answer all the remaining parts of this question.

11.4.2 3b)

What is the probability that the model will predict a higher probability for a client who will subscribe compared to a client who will not? Justify your answer. **(3 points)**

11.4.3 3c)

Assume that you want to project all your prediction results for **test1.csv and test2.csv** to real-life profits. Assume that:

- Only the clients who are predicted to subscribe are called.
- A client who is called and subscribes returns a profit of $100.
- A client who is called and does not subscribe returns a loss of $10.

What is the net profit of the results? **(3 points)** Note that you need to use the confusion matrices printed in part a.

11.4.4 3d)

Using the same assumptions in part c, find the threshold that would maximize the net profit. **(5 points)** Use the training data for this.

This is probably the most challenging part of this assignment. Here are some suggestions:

- You do not need to calculate results for every possible threshold. You should already have some metrics calculated for a large array of thresholds in part b.
- Use those metrics and the proportion of class 1 observations in the dataset to find the net profit for all thresholds.
- Find the index of the highest profit to find the threshold, again using the threshold array from part b.

11.4.5 3e)

Using the new threshold you found in the previous part, calculate the net profit using **test1.csv and test2.csv**. **(4 points)**

11.4.6 3f)

Just an intuitive question: In a real-life setting like this, would you prefer the threshold you found in part a that maximizes some mathematical concepts or the threshold you found in part d that maximizes your profit? **(1 point)**

11.5 4) Sklearn (20 points)

Using train.csv and **only sklearn, pandas, and numpy**, train a Logistic Regression model. You need the following steps:

- The response is still y. **(1 point)**
- Predictors are education, month, day and age. **(1 point)**
- Numerical predictors need to be transformed to **all** their second-order polynomial versions. **(3 points)**
- Categorical predictors need to be one-hot-encoded. **(2 points)** They should not interact with the numerical predictors. **(2 points)**
- Afterwards, the all the predictors needs to be standard scaled. **(3 points)**

Print the accuracy and recall for both training and test data using a threshold of 0.11. **(5 points)** Use **test1.csv** as the test dataset. **Remember that the test dataset needs to go through the exact same transformation pipeline as the training dataset. (3 points)**

11.6 5) Bonus: Data Visualization with Precision, Recall and FPR (12 points)

11.6.1 5a)

Plot the ROC curve for the model you trained in Question 3. Mark the point that corresponds to the decision threshold you found in 3d. (You can use a red solid point or whatever shape/color you want, as long as it is clearly marked.) Make sure you have the axis labels and the x=y line for comparison. **(3 points)**

11.6.2 5b)

Convert the previous plot to a scatter plot of TPR against FPR and color-code each point based on the corresponding profit. **(5 points)**

11.6.3 5c)

Plot the Precision-Recall curve for the model you trained in Question 3. Mark the decision threshold you found in 3d as **a dashed vertical line**. Make sure you have the axis labels. **(4 points)**

12 Assignment 3 (Section 21 & 22)

Instructions

1. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.
2. Write your code in the *Code* cells and your answer in the *Markdown* cells of the Jupyter notebook. Ensure that the solution is written neatly enough to understand and grade.
3. Use Quarto to print the *.ipynb* file as HTML. You will need to open the command prompt, navigate to the directory containing the file, and use the command: `quarto render filename.ipynb --to html`. Submit the HTML file.
4. The assignment is worth 100 points, and is due on **Tuesday, 20th February 2024 at 11:59 pm**.
5. **Five points are properly formatting the assignment**. The breakdown is as follows:
   - Must be an HTML file rendered using Quarto **(1 point)**. *If you have a Quarto issue, you must mention the issue & quote the error you get when rendering using Quarto in the comments section of Canvas, and submit the ipynb file.*
   - No name can be written on the assignment, nor can there be any indicator of the student's identity—e.g. printouts of the working directory should not be included in the final submission. **(1 point)**
   - There aren't excessively long outputs of extraneous information (e.g. no printouts of entire data frames without good reason, there aren't long printouts of which iteration a loop is on, there aren't long sections of commented-out code, etc.) **(1 point)**
   - Final answers to each question are written in the Markdown cells. **(1 point)**
   - There is no piece of unnecessary / redundant code, and no unnecessary / redundant text. **(1 point)**

Data description

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls, where bank clients were called to subscribe for a term deposit.

There is one train data - *train.csv*, which you will use to develop a model. There are two test datasets - *test1.csv* and *test2.csv*, which you will use to test your model. Each observation is a phone call and each column is a variable about the client or the phone call. Each dataset has the following attributes about the clients called in the marketing campaign:

1. `age`: Age of the client
2. `education`: Education level of the client
3. `day`: Day of the month the call is made
4. `month`: Month of the call
5. `y`: did the client subscribe to a term deposit?
6. `duration`: Call duration, in seconds. This attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the `duration` is not known before a call is performed. Also, after the end of the call `y` is obviously known. Thus, this input should only be included for inference purposes and should be discarded if the intention is to have a realistic predictive model.

(Source: UCI Data Archive. Please use the given datasets for the assignment, not the raw data from the source. It is just for reference.)

Instructions / suggestions for answering questions

(1) **Instruction:** Use *train.csv* for all questions, unless otherwise stated.

(2) **Suggestion 1:** You may use the functions in the class notes for printing the confusion matrix and the overall classification accuracy based on test / train data.

(3) **Suggestion 2:**: If you make variable transformations, you will need to do it for all the three datasets. Your code will be a bit concise if you make a function containing all the transformations, and then call it for the training and the two test datasets. You can put this function in the beginning of the code and keep adding transformations to it as you proceed with the assignment. You may need transformations in questions (1) and (13).

12.1 1)

Read the datasets. Make an appropriate visualization to visualize how the proportion of clients subscribing to a term deposit change with increasing call duration.

(4 points)

Hints:

1. Bin `duration` to create `duration_binned`. Group the data to find the fraction of clients responding positively to the marketing campaign for each bin in `duration_binned`. Make a lineplot of percentage of clients subscribing to a term deposit vs `duration_binned`, where the bins in `duration_binned` are arranged in increasing order of duration.
2. You may choose an appropriate number of bins & type of binning that helps you visualize well.
3. You may also think of other ways of visualization. You don't need to stick with this one.

12.2 2) Predictor `duration`

Based on the plot in (1), comment whether `duration` seems to be a useful variable to predict if the client will subscribe to a term deposit.

(1 point)

12.3 3) Model based on `duration`

Develop a logistic regression model to predict if the client subscribed to a term deposit based on call `duration`. Use the model to make a lineplot showing the probability of the client subscribing to a term deposit based on call `duration`.

(3 points)

Note

Answer questions 4 to 11 based on the regression model developed in (3).

12.4 4) Model significance

Is the regression model in statistically significant? Justify your answer.

(1 point for code, 1 point for answer)

12.5 5) Subscription probability in 5 minutes

What is the probability that the client subscribes to a term deposit with a 5-minute marketing call? Note that the call `duration` in data is given in *seconds*.

(2 points)

12.6 6) Call duration for subscription

What is the minimum call duration (in minutes) for which a client has a 95% or higher chance of subscribing to a term deposit?

(3 points)

12.7 7) Maximum call duration

What is the maximum call duration (in minutes) in which a client refused to subscribe to a term deposit? What was the probability of the client subscribing to the term deposit in that call?

(3 points)

12.8 8) Percent increase in odds

What is the percentage increase in the odds of a client subscribing to a term deposit when the call `duration` increases by a minute?

(3 points)

12.9 9) Doubling the subscription odds

How much must the call `duration` increase (in minutes) so that it doubles the odds of the client subscribing to a term deposit.

(3 points)

12.10 10) Classification accuracy

What is minimum overall classification accuracy of the model among the classification accuracies on *train.csv*, *test1.csv* and *test2.csv*? Consider a threshold of 30% when classifying observations.

(2 + 1 + 1 points)

12.11 11) Recall

What is the minimum *Recall* of the model among the *Recall* performance on *train.csv*, *test1.csv* and *test2.csv*? Consider a decision threshold probability of 30% when classifying observations.

Here, *Recall* is the proportion of clients predicted to subscribe to a term deposit among those who actually subscribed.

(3 points)

12.12 12) Subscription probability based on `age` and `education`

Develop a logistic regression model to predict the probability of a client subscribing to a term deposit based on `age`, `education` and the two-factor interaction between `age` and `education`. Based on the model, answer:

a. People with which type of `education` *(primary / secondary / tertiary / unknown)* have the highest percentage increase in odds of subscribing to a term deposit with a unit increase in `age`? Justify your answer.
b. What is the percentage increase in odds of a person subscribing to a term deposit for a unit increase in `age`, if the person has *tertiary* `education`.
c. What is the percentage increase in odds of a person subscribing to a term deposit for a unit increase in `age`, if the person has *primary* `education`.

(1 point for developing the model, 3 points for (a), 3 points for (b), 3 points for (c))

12.13 13) Model development

Develop a logistic regression model *(using train.csv)* to predict the probability of a client subscribing to a term deposit based on `age`, `education`, `day` and `month`. The model must have:

a. Minimum overall classification accuracy of 75% among the classification accuracies on *train.csv*, *test1.csv* and *test2.csv*.

b. Minimum recall of 50% among the recall performance on *train.csv*, *test1.csv* and *test2.csv*.

For all the three datasets - *train.csv*, *test1.csv* and *test2.csv*, print the:

1. Model summary (only for *train.csv*),
2. Confusion matrices,
3. Overall classification accuracies, and
4. Recall

Note that:

1. You cannot use `duration` as a predictor because its value is determined after the marketing call ends. However, after the call ends, we already know whether the client responded positively or negatively. That is why we have used `duration` only for inference in the previous questions. It helped us understand the effect of the length of the call on marketing success.
2. It is possible to develop the model satisfying constrains (a) and (b) with just appropriate transformation(s) of the predictor(s). However, you may consider interactions if you wish. Justify the transformations, if any, with visualizations.
3. You are free to choose any value of the decision threshold probability for classifying observations. However, you must use the same threshold on all the three datasets.

(10 points)

12.14 14) ROC-AUC

Report the probability that the model will predict a higher probability of response for a customer who signs up for the term deposit as compared to the customer who does not sign up, i.e., the ROC-AUC of the developed model in (13).

Hint: Use the functions `roc_curve`, and `auc` from the `sklearn.metrics` module

(3 points)

12.15 15) Net-profit

Suppose that the model developed in (13) is used to predict the clients in *test1.csv* and *test2.csv* who will respond positively to the campaign. Only those clients who are predicted to respond positively are called during the marketing campaign. Assume that:

1. A profit of \$100 is associated with a client who responds positively to the campaign,
2. A loss of \$10 is associated with a client who responds negatively to the campaign

What is the net profit from the campaign? Use the confusion matrices printed in (13).

(4 points)

12.16 16) Decision threshold probability

Based on the profit and loss associated with client responses specified in (15), and the model developed in (13), find the decision threshold probability of classification, such that the net profit is maximized. Use *train.csv*

Proceed as follows:

1. You would have obtained FPR and TPR for all potential decision threshold probabilities in (14).
2. Formulate an expression quantifying the net profit per client, in terms of FPR, TPR, and the overall response rate, i.e., proportion of people actually subscribing to the term deposit.
3. Find the decision threshold probability that maximizes the expression in (2).

(5 points)

12.17 17) Net profit based on new decision threshold probability

Using the new decision threshold probability obtained in (16), answer (15), i.e., what is the net-profit associated with the clients in *test1.csv* and *test2.csv* if a marketing campaign is performed. Again, only those clients who are predicted to respond positively, based on the new decision threshold probability, are called during the marketing campaign

Also, print the confusion matrices for predictions on *test1.csv* and *test2.csv* with the new threshold probability.

(4 points)

12.18 18) Model preference

Was the classification accuracy of the model in (13) higher than that of the model in (17)? If yes, then should you prefer the model in (13) for the marketing campaign? Why or why not?

Note: The model in (17) is the same as in (13), except with a different decision threshold probability

(3 points)

12.19 19) ROC curve

Plot the ROC curve for the model developed in (13). Mark the point on the curve corresponding to the decision threshold probability identified in (16).

Note that the ROC curve is independent of the decision threshold probability used by the model for prediction

(3 points)

12.20 20) Profit with TPR / FPR

Make a scatterplot of TPR vs FPR, and color the points based on net profit per client.

You can use the following code to make the plot if you have the relevant metrics in `tpr`, `fpr`, and `net_profit`

(1 point)

```python
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(font_scale=1.5)
plt.rcParams["figure.figsize"] = (9,6)
plt.rcParams["figure.autolayout"] = True
f, ax = plt.subplots()
points = ax.scatter(fpr, tpr, c = net_profit, s=50, cmap="Blues")
f.colorbar(points, label = "Net profit ($) \n(per client)")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```

12.21 21) Precision-recall

Compare the precision and recall of the models in (13) and (17) on *train.csv*.

Note: The model in (17) is the same as in (13), except with a different decision threshold probability

(4 points)

12.22 22) Precision-recall: important metric

Based on the above comparison, which metric among precision and recall turns out to be more important for maximizing the net profit in the marketing campaign?

(1 point)

12.23 23) Precision-recall curve

Plot the precision-recall curve vs decision threshold probability for the model developed in (13). Mark the points on the curve corresponding to the decision threshold probability identified in (16).

(3 points)

12.24 24) Precision-recall vs FPR-TPR

Instead of using the FPR and TPR metrics to find the optimum decision threshold probability in (16), use the precision-recall metrics to find the same.

(5 points)

12.25 25) Sklearn

Using train.csv and **only sklearn, pandas, and numpy**, train a Logistic Regression model. You need the following steps:

- The response is still y.
- Predictors are education, month, day and age.
- Numerical predictors need to be transformed to **all** their second-order polynomial versions.
- Categorical predictors need to be one-hot-encoded. They should not interact with the numerical predictors.
- Afterwards, the all the predictors needs to be standard scaled.

Print the accuracy and recall for both training and test data using a threshold of 0.11. Use **test1.csv** as the test dataset. **Remember that the test dataset needs to go through the exact same transformation pipeline as the training dataset.**

(8 points)

13 Assignment 4

Instructions

1. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.
2. Write your code in the **Code cells** and your answers in the **Markdown cells** of the Jupyter notebook. Ensure that the solution is written neatly enough to for the graders to understand and follow.
3. Use Quarto to render the **.ipynb** file as HTML. You will need to open the command prompt, navigate to the directory containing the file, and use the command: `quarto render filename.ipynb --to html`. Submit the HTML file.
4. The assignment is worth 100 points, and is due on **Monday, 4th March 2024 at 11:59 pm**.
5. There is a **bonus** question worth 11 points.
6. **Five points are properly formatting the assignment**. The breakdown is as follows:
   - Must be an HTML file rendered using Quarto **(1 point)**. *If you have a Quarto issue, you must mention the issue & quote the error you get when rendering using Quarto in the comments section of Canvas, and submit the ipynb file.*
   - No name can be written on the assignment, nor can there be any indicator of the student's identity—e.g. printouts of the working directory should not be included in the final submission. **(1 point)**
   - There aren't excessively long outputs of extraneous information (e.g. no printouts of entire data frames without good reason, there aren't long printouts of which iteration a loop is on, there aren't long sections of commented-out code, etc.) **(1 point)**
   - Final answers to each question are written in the Markdown cells. **(1 point)**
   - There is no piece of unnecessary / redundant code, and no unnecessary / redundant text. **(1 point)**
7. The maximum possible score in the assigment is 99 + 11 + 5 = 115 out of 100.

13.1 1) Modeling the Radii of Exoplanets (40 points)

For this question, we are interested in predicting the radius of exoplanets (planets outside the Solar System) in kilometers. To achieve this goal, we will use NASA's Composite Planetary Systems dataset and different regression models. (See https://exoplanetarchive.ipac.caltech.edu for more context.)

Read all three CompositePlanetarySystems datasets - you should have one training and two test datasets. Each row is an exoplanet. `pl_rade` column represents the radius of each exoplanet as a proportion of Earth's radius, which is approximately 6,378 km.

13.1.1 a)

Develop a linear regression model (no non-linear terms) to predict `pl_rade` using all the variables in the data except `pl_name`, `disc_facility` and `disc_locale`. You can use statsmodels or sklearn. **(2 points)**

13.1.2 b)

Find the RMSE of the model using both test sets **separately**. (You need to print two RMSE values.) Note that the library you used should not make a difference here! **(2 points)**

Print the training RMSE as well for reference. **(1 point)**

13.1.3 c)

Compare the training and test RMSEs. **(1 point)** What is the issue with this model? **(1 point)**

13.1.4 d)

Train a Ridge regression model to predict `pl_rade` using the same variables as above. Optimize the hyperparameter using the RidgeCV object with LOOCV and `neg_root_mean_squared_error` scoring. What is the optimal hyperparameter value? **(5 points)**

Note:

- Keep in mind that scaling is always necessary before Ridge/Lasso regression.
- Use the following array of possible hyperparameter values: `alphas = np.logspace(2,0.5,200)`
- You have to use the RidgeCV object for this question.

13.1.5 e)

Using the optimized and trained model, print the RMSEs for the training set and both test sets. **(4 points)**

13.1.6 f)

How did the training and test performance change? Explain why the Ridge regression changed the training and test results. **(3 points)**

13.1.7 g)

Find the predictor whose coefficient is shrunk by far the most by Ridge regularization. **(3 points)**

Hint: .coef_ and .columns attributes should be helpful.

13.1.8 h)

Why did the coefficient of the predictor identified in the previous question shrunk the most? Justify your answer for credit. **(2 points)**

Hint: Correlation vector/matrix

13.1.9 i)

Visualize how the coefficients change with the change in the hyperparameter value:

- Create a line plot of coefficient values vs. the hyperparameter value.
- Color code each predictor's coefficient values.
- Use log scale where necessary.
- Use an alphas vector of `np.logspace(7,0,200)` for better visualization

(5 points)

13.1.10 j)

Recreate some of the previous steps with Lasso regression.

- **Using LassoCV only**, find the optimal hyperparameter value. **(2 points)**
  - You need a different hyperparameter array - use: `np.logspace(0,-2.5,200)`
  - Use 10-fold CV.
  - Lasso object does not have a scoring input.
- Using the optimized and trained Lasso model, print the RMSEs for the training set and both test sets. **(2 points)**
- Visualize how the coefficients change with the change in the hyperparameter value. **(2 points)**
  - Use the hyperparameter array as `np.logspace(7,-2.5,200)` for better visualization.

13.1.11 k)

Using the two figures created in parts i and j, explain how the Ridge and the Lasso models behave differently as the hyperparameter value changes. **(2 points)** What does that difference mean for the usage of the Lasso model? **(1 point)**

13.1.12 l)

Find the predictors that are eliminated by Lasso regularization. **(2 points)**

13.2 2) Improving House Prices Prediction with Higher-order Terms and Crossvalidation (29 points)

In this question, we are interested in improving the prediction performance for house prices using five predictors.

13.2.1 a)

Read the house feature and price files and create the training and test datasets. The response is log-price and the five predictors are the rest of the variables, except house_id. **(2 points)**

13.2.2 b)

In class, we saw how an entirely linear model is not enough to capture the complexity in the relationship between the response and the predictors - in other words, it is underfitting. We want to analyze how the training and test performance change as the level of model complexity increases.

Using PolynomialFeatures object, create higher-order versions of the predictors (both transformations and interactions) in the training and test data. **(3 points)** Using all predictors (linear and transformed), train a Ridge model with alpha=0.000001 **(2 points)** and store the training and test RMSEs. **(2 points)** Repeat this process from order 1 to order 6. **(2 points)**

Finally, plot the training and test RMSE values on the same figure against the order. **(1 point)** Make sure the two lineplots have different colors and a legend is included. **(1 points)**

Note:

- This question needs a loop.
- PolynomialFeatures object keeps the lower order terms (k-1 to 1) while creating new predictors of order k, so no need to concatenate.
- Don't forget to exclude the bias term created by default with PolynomialFeatures.
- Don't forget to scale (correctly) **(2 points)**
- Minimal regularization is necessary for this question, as opposed to pure Linear Regression, otherwise the test RMSE values go to infinity very quickly for higher orders.

13.2.3 c)

Which order has the best test RMSE? **(1 point)** What is the best test RMSE? **(1 point)** At which order does the overfitting start? **(1 point)**

13.2.4 d)

Repeat part b, only this time use RidgeCV to find the best amount of regularization for each order by cross-validation. Use `alphas = np.logspace(2,0.5,200)` and LOOCV. Use `neg_root_mean_squared_error` for scoring. Create the same plot as part b. **(4 points)** Describe the obvious difference between the plot in this part and the plot in part b. **(2 points)**

13.2.5 e)

What is the best test RMSE found by using higher-orders and regularization? **(1 point)** Which order achieved this test RMSE? **(1 point)** Why did this order with regularization perform better than any lower order with or (almost) without regularization? **(3 points)**

13.3 3) Systematic Elimination of Interaction Terms (30 points)

In this question, we are interested in predicting if the client subscribed to a term deposit or not after a phone call using **age** and **education** of the client and the **day** and the **month** the call took place.

Note that this is the same problem as in the previous assignment, however, **using sklearn**, we aim to make the predictive analysis with interactions more systematic.

13.3.1 a)

Read train.csv, test1.csv, and test2.csv. Prepare the training and two test datasets according the description above. **(2 points)**

13.3.2 b)

For all datasets:

- One-hot-encode the categorical predictors. **(2 points)**
- Get the interactions of **all** the predictors. (Numeric and one-hot-encoded) **(3 points)**
  - Note that there is a very quick way of doing this with PolynomialFeatures
  - Don't forget to exclude the bias.
- Scale the predictors (correctly.) **(2 points)**

13.3.3 c)

Train a Logistic Regression model with Lasso penalty. **(2 points)** The idea is to discard interactions that are not useful. Note that instead of the manual, trial-and-error way of adding interactions in statsmodels, we include all the possible interactions and then discard the useless ones here.

- Use `[0.0001,0.001, 0.01, 0.1, 1, 10, 100, 1000]` as the possible C values. **(1 point)**
- Use 10-fold cross-validation to optimize the C value. **(1 point)**
- Lasso is very useful, but it needs special algorithms, since it includes non-differentiable absolute values. Use `saga` as the solver. **(1 point)**
- For the same reason as above, the default number of iterations the algorithm takes is usually not enough for Lasso. Use `max_iter = 1000`. (Default is 500) **(1 point)**
- **This will take 10-20 minutes to run.**

13.3.4 d)

How many models in total are run by this cross-validation process? **(2 points)**

13.3.5 e)

What is the optimum C value? **(1 point)** What is the lambda (in the Lasso cost) value it corresponds to? **(1 point)**

13.3.6 f)

What is the percentage of terms (linear or interaction) that are discarded by Lasso? (**Hint:** .coef_) **(2 points)**

13.3.7 g)

Find the five terms that have the highest effect on the logodds of a subscription. Assume that we are quantifying the effect of a term with the absolute value of its coefficient. (**Hint:** .get_feature_names_out()) **(4 points)**

13.3.8 h)

Come up with real-life explanations on why the terms identified in the previous part are important. (This is an open-ended question, just make sure your answer makes sense.) **(2 points)**

13.3.9 i)

Lastly, find a threshold to get all three (training and both test) datasets above 75% accuracy and 50% recall. Note that you only worry about the threshold now. Lasso took care of finding good interactions. **(3 points)**

13.4 4) Bonus: ElasticNet (11 points)

The entire goal of this part is to get you familiar with an alternate model: ElasticNet. It is implemented by adding both Lasso and Ridge penalties to the RSS (or subtracting them from the loglikelihood.) How much Lasso and how much Ridge penalty is up to two hyperparameters.

13.4.1 a)

For regression, sklearn has its own object and its CV version for ElasticNet.

Do your own research and implement a 5-fold cross-validation for the options of 25% Lasso-75% Ridge, 50% Lasso-50% Ridge and 75% Lasso- 25% Ridge, and the alpha values of `alphas = np.logspace(10,0.1,200)`.

- Use the dataset given in the first question (with the same columns dropped.)
- You still need to scale.
- Return the best Lasso-Ridge ratio and the alpha value pair that corresponds to the best test performance.
- Note that even if you use the CV object, you have to use loops.

(8 points)

13.4.2 b)

How many models were run in the cross-validation process of two hyperparameters? **(1 point)**

13.4.3 c)

Briefly mention how you would implement ElasticNet for Logistic Regression. Again, you need to do your own research. **(2 points)**

14 Assignment 5

Instructions

1. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.

2. Write your code in the **Code cells** and your answers in the **Markdown cells** of the Jupyter notebook. Ensure that the solution is written neatly enough to for the graders to understand and follow.

3. Use Quarto to render the **.ipynb** file as HTML. You will need to open the command prompt, navigate to the directory containing the file, and use the command: `quarto render filename.ipynb --to html`. Submit the HTML file.

4. The assignment is worth 100 points, and is due on **Thursday, 14th March 2024 at 11:59 pm**.

5. **Five points are properly formatting the assignment**. The breakdown is as follows:

   - Must be an HTML file rendered using Quarto **(1 point)**. *If you have a Quarto issue, you must mention the issue & quote the error you get when rendering using Quarto in the comments section of Canvas, and submit the ipynb file.*
   - No name can be written on the assignment, nor can there be any indicator of the student's identity—e.g. printouts of the working directory should not be included in the final submission. **(1 point)**
   - There aren't excessively long outputs of extraneous information (e.g. no printouts of entire data frames without good reason, there aren't long printouts of which iteration a loop is on, there aren't long sections of commented-out code, etc.) **(1 point)**
   - Final answers to each question are written in the Markdown cells. **(1 point)**
   - There is no piece of unnecessary / redundant code, and no unnecessary / redundant text. **(1 point)**

6. The maximum possible score in the assigment is 100 + 5 = 105 out of 100.

14.1 1) Cross-validation for a Regression Task (34 points)

For this question, we are interested in using lower-level cross-validation tools for a regression task. Read the **soc_ind.csv** file. The column names should be clear on what the variables represent.

14.1.1 a)

`gdpPerCapita` will be the response for this regression analysis. Before anything else, create two density plots to see if we should use it as it is or its log-transformed version. Justify your answer with the plots. **(2 points)**

Hint: sns.kdeplot

14.1.2 b)

Create the proper response variable based on your answer in the previous part. The predictors are the rest of the variables except `Index`, `geographic_location`, and `country`. Create a predictor matrix accordingly. **(2 points)**

Using **train_test_split** from **sklearn.model_selection** create the training and test data. (You may need to read its documentation.) Use a 80%-20% train-test split and use `random_state=2` for reproducible results. **(3 points)**

14.1.3 c)

One-hot-encode and scale (in this order) both the training and the test dataset. **(2 points)**

14.1.4 d)

Using a hyperparameter vector of `np.logspace(2,0.5,200)`, cross-validate a Ridge Regression model. Use 10 folds and `neg_mean_absolute_error` as the scoring metric. **(4 points)** Save all your cross-validation (CV) scores in a numpy array. **(1 point)**

14.1.5 e)

Using the array you created in the previous part, find the optimal hyperparameter value and the best CV score that corresponds to it. **(1+1=2 points)**

14.1.6 f)

Check the best CV score you found in the previous part. What seems to be the issue with it? Remember that the response is GDP per capita of countries. (We will solve this issue later in this question.) **(2 points)**

14.1.7 g)

Create a final model with the optimal hyperparameter value you found in the previous question. Return the test MAE. **You need to return the test MAE in terms of actual GDP values for credit. (3 points)**

14.1.8 h)

Now, it is time to calculate proper MAE values for the cross-validation results and optimize the hyperparameter value based on them. **Using cross_val_predict**, return the CV predictions for all hyperparameter values. Use a hyperparameter vector of `np.logspace(2,0.5,200)`. (Same as part d.). **(4 points)** Save all the predictions in a DataFrame. **(1 point)**

14.1.9 i)

Using the DataFrame you created in the previous part, find the optimal hyperparameter value and the best CV MAE that corresponds to it. **(4 points)**

Note:

1) The MAE should be in terms of actual GDP values for credit.
2) No loops are allowed for this question. You may want to refresh your memory on .apply.

14.1.10 j)

With the hyperparameter value you found in the previous part, train a final model and print its test MAE. **(2 points)** How does it compare to the test MAE you found with cross_val_score? **(1 point)** Why do you think this is the case? **(1 point)**

14.2 2) Cross-validation for a Classification Task (36 points)

For this question, we are interested in using lower-level cross-validation tools for a classification task. Read the diabetes_train.csv and diabetes_test.csv files. The `Outcome` variable represents whether the patient has diabetes or not. The rest of the variables are medical predictors we will use to predict the outcome.

14.2.1 a)

Create the training and the test data. **(2 points)**

14.2.2 b)

Scale the datasets. **(1 point)**

14.2.3 c)

Using a hyperparameter vector of `Cs = np.logspace(2,-2,200)`, cross-validate a Lasso Classification model. Use 5 folds and the default scoring metric (which is accuracy.) **(4 points)** Save all your cross-validation (CV) scores in a numpy array. **(1 point)**

14.2.4 d)

Using the array you created in the previous part, find the optimal hyperparameter value and the best CV score that corresponds to it. **(1+1=2 points)**

14.2.5 e)

Create a final model with the optimal hyperparameter value you found in the previous question. Return the test accuracy, recall and AUC with a threshold of 0.5. **(4 points)** Which metric looks problematic? **(1 point)**

14.2.6 f)

What was the threshold cross_val_score used to return the accuracy scores? **(1 point)** How did that contribute to the problem in the previous part? **(1 point)**

14.2.7 g)

Now, it is time to return the CV predictions and optimize the threshold based on them. Using cross_val_predict, return the CV **prediction probabilities** for the best hyperparameter value your found in part d. **(3 points)** Note that you don't need any loops for this question, because you already know which C value to use.

14.2.8 h)

Using the output of the previous part, calculate and store the accuracy, recall and AUC of all possible threshold values from 0 to 1 with a stepsize of 0.001. **(4 points)**

14.2.9 i)

Plot the accuracy, recall and AUC values against the threshold on the same graph. **(2 points)** Include a legend. **(1 point)**

14.2.10 j)

In the plot, you should see a threshold value where the accuracy, recall and AUC values are all the same. Find that value. **(3 points)**

Note:

1) The metric values are the same if you round them to 2 digits after the decimal point. (That is just the rounded integer values if you multiply the metric values by 100 in the previous question.)
2) Trial-and-error will not receive any credit for this question. You need to use logical indexing.
3) **np.where** and **np.round** should be helpful. (Along with the **& operator**.)

14.2.11 k)

Using the threshold value you found in the previous question and the best hyperparameter value you found in part d, train a final Lasso Classification model and return its test accuracy, recall and AUC. **(3 points)** How do the accuracy and recall results compare to part e? **(1 point)** Did AUC change? **(1 point)** Why or why not? **(1 point)**

14.3 3) Outliers and Collinearity (30 points)

For this question, we are interested in analyzing how removing the unnecessary observations and predictors improve the prediction and inference performance of our model. Read the **Austin_Affordable_Housing_Train.csv** and **Austin_Affordable_Housing_Test.csv** files. Each row represents a housing development in Austin, TX.

The `City_Amount` variable represents the amount (in USD) provided by the city of Austin to the development and it is the response for the regression task.

14.3.1 a)

Use `Market_Rate_Units`, `Total_Affordable_Units`, `Total_Accessible_Units`, and `Total_Units` as four predictors to the linear regression model. Do not include any interaction terms and do not transform anything. Do not transform the response either.

Create the model using statsmodels. Print the model summary and the test RMSE. **(3 points)** Which predictor is statistically insignificant? **(1 point)**

14.3.2 b)

To dive deeper into the statistical significance of the four predictors, create their correlation matrix first. **(1 point)** How many pairs seem to be highly correlated? **(1 point)** Why is this matrix not useful to detect collinearity in the model? **(2 points)**

14.3.3 c)

Create the Variation Inflation Factor (VIF) table of the predictors. **(1 point)** Which predictors seem to be highly correlated? **(1 point)**

14.3.4 d)

Remove the predictor with the highest VIF and create the VIF table again. **(1 point)** Is there any collinearity left? **(1 point)**

14.3.5 e)

With the remaining predictors, create the model again. Print its summary and test RMSE. **(1 point)** How did they change? Mention test RMSE **(1 point)**, R-squared **(1 point)** and the statistical significance. **(1 point)**. What is the reason behind these changes? **(2 points)**

14.3.6 f)

Now, it is time to clean out the observations. Find the influential points in the training dataset and filter them out. **(5 points)** How many observations did you discard? **(1 point)**

14.3.7 g)

Retrain the model in part e with the clean training dataset. Print the summary and the test RMSE. **(1 point)** How do the test RMSE and R-squared compare with the results in part e? **(2 points)** Do you also see a change in statistical significance? **(1 point)** Explain the reasons behind these changes. **(2 points)**

A Regression prediction problem: Common mistakes

Below is a sample solution to the regression prediction problem that consists of conceptual mistakes, and semantic errors. This highlights some of the common mistakes that students make in their solutions.

A.1 Step 0

Assuming missing value imputation, and data cleaning *(such as converting `price` to numeric, etc.)* has been done already. The cleaned train and test datasets are `train_clean`, and `test_clean` respectively.

```
%reset

# Imputing missing values & cleaning data
%run "missing_value_imputation.ipynb"
```

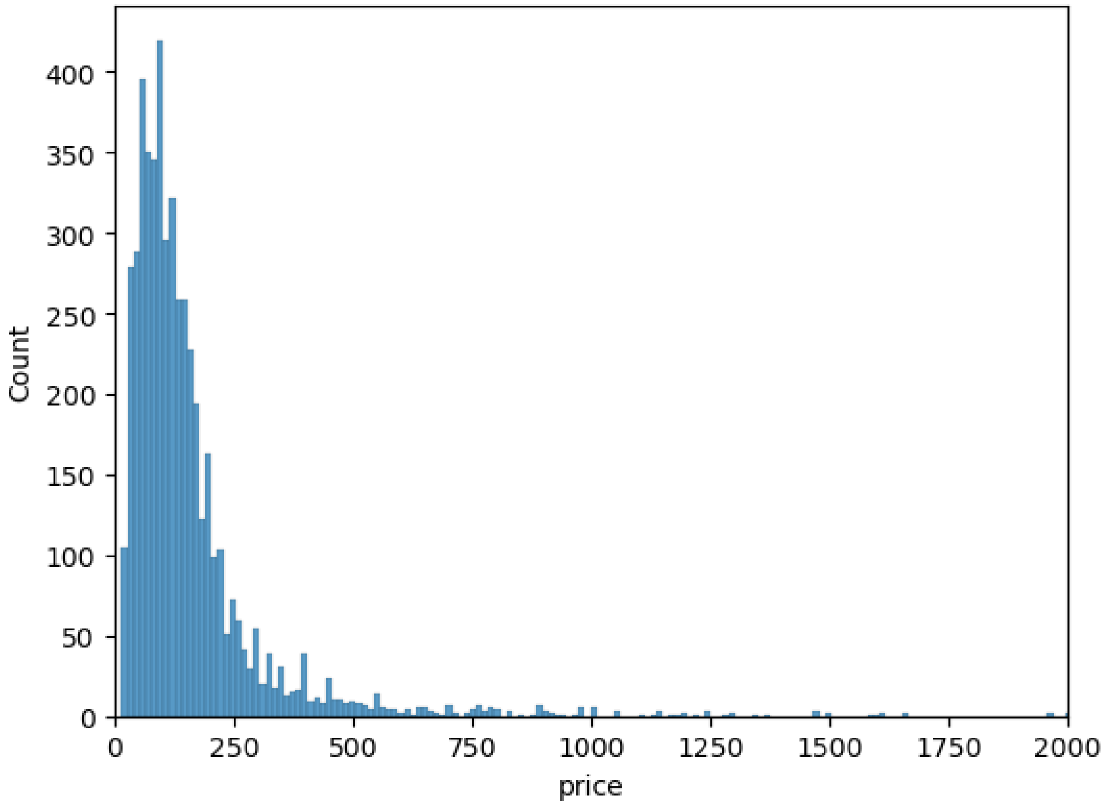
```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

A.2 Step 1

A.2.1 Response transformation

Let us visualize the distribution of the response `price`.

```
sns.histplot(train_clean.price)
plt.xlim([0,2000]);
```



As the response `price` is right-skewed, we will take the log-transform to reduce the skew.

```
train_clean['log_price'] = np.log(train_clean.price)
```

No mistake!

However, if you have mentioned something like the following, it is fine.

When plotting the price to determine if it needed a transformation you set the x-axis limit to a max of 2000, this will prevent you from understanding the extent of the skew in the data, and also prevents you from noting what may be some extreme outliers in the data (like a value of 99,000) which may require further exploration if they turn out to be influential points. The solution would be to not include a limit in the plot. Using log of price to solve the right skew issue is the correct transformation.

A.3 Step 2

A.3.1 Capping outliers

As outliers may distort the regression model, let us cap the outlying values of the transformed response.

```
#Finding upper and lower quartiles and interquartile range
q1 = np.percentile(train_clean['log_price'],25)
q3 = np.percentile(train_clean['log_price'],75)
intQ_range = q3-q1
```

```
#Tukey's fences
Lower_fence = q1 - 1.5*intQ_range
Upper_fence = q3 + 1.5*intQ_range
```

```
# Capping the outlying values
train_clean.loc[(train_clean.log_price < Lower_fence), 'log_price'] = Lower_fence
train_clean.loc[(train_clean.log_price > Upper_fence), 'log_price'] = Upper_fence
```

Mistake 1: Should find outliers with respect to the model (1 point)

Capping outliers is the wrong approach to take because outliers may not have much impact on the data if they are not influential points, and we may be able to explain these outliers easily in our model. For example we may find that these outliers have extremely large values for `accommodates` and this may be why they have such high values. Removing them prevents the model from addressing what may be things it easily could in a model. Removing points that could have been explained by the model will mean we will likely underestimate similar data points in the test data set.

A.4 Step 3

A.4.1 Combining levels of categorical predictors with very few observations

Some levels of categorical variables may have very few observations, which may lead to unreliable estimates of their regression coefficients. Thus, we will merge such levels into an 'others' category.

There are 76 levels of `neighbourhood_cleansed` some of which have very few observations.

```
train_clean.neighbourhood_cleansed.value_counts().shape
```

```
(76,)
```

```
train_clean.neighbourhood_cleansed.value_counts().tail()
```

```
South Deering    2
West Elsdon      2
Riverdale        1
Gage Park        1
Edison Park      1
Name: neighbourhood_cleansed, dtype: int64
```

```
test_clean.neighbourhood_cleansed.value_counts().shape
```

```
(76,)
```

```
test_clean.neighbourhood_cleansed.value_counts().tail()
```

```
Avalon Park        1
South Deering      1
Mount Greenwood    1
Edison Park        1
Chicago Lawn       1
Name: neighbourhood_cleansed, dtype: int64
```

Let us merge levels of `neighbourhood_cleansed` that have less than 40 observations. **Assume that 40 is a reasonable cut-off. There is no mistake in the choice of this cut-off.**

```
# Merging levels of neighbourhood_cleansed that have less than 40 observations in train data
train_clean['neighbourhood_cleansed'] = train_clean[['id','neighbourhood_cleansed']].groupby
    group_keys=False).transform(lambda x:'others' if x.count() < 40 else train_clean.loc[x.in
```

```
# Merging levels of neighbourhood_cleansed that have less than 40 observations in test data
test_clean['neighbourhood_cleansed'] = test_clean[['id','neighbourhood_cleansed']].groupby([
    group_keys=False).transform(lambda x:'others' if x.count() < 40 else test_clean.loc[x.ind
```

Similarly, we can merge levels of all such categorical variables, using appropriate cut-offs.

Mistake 2: Should keep neighbourhoods in test that are in train, instead of using the cut-off used in train (0.5 points)

Some neighbourhoods that have more than 40 observations in train data may have less than 40 observations in test data. Such neighbourhoods will be renamed as 'others' in the test data, but not in the train data, which will lead to different distinct dummy variables for neighbourhoods in train and test data. To rectify that, you may create columns for those neighbourhoods in test data, and set all values as 0. However, that will be inaccurate because those neighbourhoods actually have listings in the test data, but they were renamed as 'others'. So, the correction will be to simlpy rename all those neighbourhoods as 'others' in test data that are renamed as 'others' in the train data.

A.5 Step 4

A.5.1 Dummy variables

Let us convert categorical variables to dummy variables, as we intend to develop a ridge regression model. We will use the argument `drop_first = True` in the Pandas function `get_dummies()` as it reduces the size of the dataset without losing any information from the data.

```
# Train data
train_clean = pd.get_dummies(train_clean, drop_first = True)

# Cleaning column names
train_clean.columns = train_clean.columns.str.replace(' ', '_')
train_clean.columns = train_clean.columns.str.replace('-', '_')
train_clean.columns = train_clean.columns.str.replace('/', '_')

# Test data
test_clean = pd.get_dummies(test_clean, drop_first = True)

# Cleaning column names
test_clean.columns = test_clean.columns.str.replace(' ', '_')
test_clean.columns = test_clean.columns.str.replace('-', '_')
test_clean.columns = test_clean.columns.str.replace('/', '_')
```

Let us check if we have the same number of columns in the train and test data.

```
train_clean.shape
```

```
(5000, 96)
```

```
test_clean.shape
```

```
(3338, 91)
```

```
# Columns in train data that are not in test data
np.setdiff1d(train_clean.columns, test_clean.columns)
```

```
array(['log_price', 'neighbourhood_cleansed_Avondale',
       'neighbourhood_cleansed_Douglas',
       'neighbourhood_cleansed_Lincoln_Square', 'price'], dtype=object)
```

There are listings in 3 neighbourhoods in the train data that must also be in test data. Let us create the columns for those neighbourhoods in test data so that we have the same columns in both train and test datasets.

```
test_clean['neighbourhood_cleansed_Douglas'] = 0
test_clean['neighbourhood_cleansed_Lincoln_Square'] = 0
test_clean['neighbourhood_cleansed_Avondale'] = 0
```

No mistake!

However, if you have explained the mistake of the previous step in this step, it is fine.

A.6 Step 5

A.6.1 Ordinal variables

Here is an idea to further reduce the size of the dataset without losing any information. We will use the dummy variables to create an ordinal variable, which will have the information of all the dummy variables that correspond to the same categorical variable.

Let us replace the dummy variables of `room_type` with an ordinal variable.

```
####----Train data processing---------####

# making one big room_type column including all of the room types
train_clean['room_type'] = (train_clean['room_type_Hotel_room'] * 1 +
                   train_clean['room_type_Private_room'] * 2 + train_clean['room_type_Shared_
                           (1-(train_clean['room_type_Hotel_room']+ train_clean['room_type_Pr
                               train_clean['room_type_Shared_room']))*4  )

# Drop the dummy variables
train_clean.drop(columns = ['room_type_Hotel_room', 'room_type_Private_room', 'room_type_Shar


####----Test data processing---------####

# making one big room_type column including all of the room types
test_clean['room_type'] = (test_clean['room_type_Hotel_room'] * 1 +
                   test_clean['room_type_Private_room'] * 2 + test_clean['room_type_Shared_ro
                           (1-(test_clean['room_type_Hotel_room']+ test_clean['room_type_Priv
                               test_clean['room_type_Shared_room']))*4  )

# Drop the dummy variables
test_clean.drop(columns = ['room_type_Hotel_room', 'room_type_Private_room', 'room_type_Share
```

Similarly, other dummy variables can be converted to ordinal variables to reduce data size without losing any information.

Mistake 3: Creating unreasonable constraint (1 point)

Ordinal variables are not appropriate in this scenario because this relies on the assumption that there is an inherent hierarchy to the dummy variable which is not true. This step should be skipped (or only be applied to things we know are hierarchical in nature). Even if there was a hierarchy, another constraint it adds is that the difference between the expected response for any two consecutive levels of the hierarchy is the same.

A.7 Step 6

A.7.1 Scaling data

As we plan to develop a ridge regression model, we will scale predictors.

```
X_train = train_clean.drop(columns = ['price', 'log_price', 'id', 'host_id'])
X_test = test_clean.drop(columns = ['id', 'host_id'])
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

Let us check the shapes of train and test data to see if they are consistent.

```
X_train.shape
```

```
(5000, 90)
```

```
X_test.shape
```

```
(3338, 90)
```

Train and test datasets are consistent with regard to columns!

Mistake 4: Must use transform() on test data (1 point)

`fit_transform` should not be used on test since we must scale the test data based on the mean and variance of the columns of the train data, and not the test data.

A.8 Step 7

A.8.1 Two-factor interactions

Let us include two-factor interactions of all predictors.

```
poly = PolynomialFeatures(2, include_bias = False)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)
```

Mistake 5: Must ensure that predictors are in the same order in train and test (0.5 points)

Note that three predictors *(or columns)* were added to test data in step 4. These columns will be added to the extreme right hand side of the test data set. This implies that the order in which the columns appear in the train and test data is different. This, in turn, implies that the order in which the columns appear in the scaled train and test data sets is also different. However, we lose the column names in the scaled datasets in Step 6. So, the function `transform()` used here doesn't throw an error that the columns must be in the same order, and creates the interactions. However, the interactions created in the train and test datasets are in a different order, which will lead to incorrect predictions on the test data.

A.9 Step 8

A.9.1 Model hyperparameter optimization

Let us find the optimal value of the regularization parameter for a ridge regression model.

```
alphas = np.logspace(2,0.5,2)
modelcv = RidgeCV(alphas = alphas, scoring = 'neg_root_mean_squared_error').fit(X_train_poly
modelcv.alpha_
```

```
100.0
```

Mistake 6: Should expand search space (1 point)

If the optimal hyperparameter value is found at the edge of the search space, then the search space must be expanded in that direction. The cost function is highly likely to be minimized further if we continue search in the direction in which the cost function is decreasing.

A.10 Step 9

A.10.1 Cross-validation

Let us find the 5-fold cross validated root mean squared error (RMSE) to check if the model with the optimal regularization parameter is good, before making predictions.

```
np.exp(np.mean(-cross_val_score(Ridge(alpha = modelcv.alpha_), X_train_poly,
                train_clean.log_price, scoring = 'neg_root_mean_squared_error', n_jobs = -1)
```

```
30.617446257505094
```

The 5-fold cross-validated RMSE is only around $30. The model seems to be good!

Mistake 7: Incorrect back-transformation to units of response (1 point)

`cross_val_score()` returns 5 errors in the units of log price. Taking the exponential of the averge of these errors does not convert the error into the units of the response. Here, the function `cross_val_predict()` needs to be used to get the predictions in units of log price, then those predictions should be exponentiated to get them in the units of price, and then the cross-validated error must be obtained by comparing the cross-validated predictions in the units of price to the actual untransformed price.

A.11 Step 10

A.11.1 Model predictions

Let us use the model corresponding to the optimal regularization parameter value to make predictions.

```
test_predictions = np.exp(modelcv.predict(X_test_poly))
```

No mistake

A.12 Order of steps

Mistake 8: (1 point)

Steps 7 must come before step 6, predictors must be scaled after including the two-factor interactions.

B Datasets, assignment and project files

Datasets used in the book, assignment files, project files, and prediction problems report tempate can be found here