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

Linear regression

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: $\hat{price}$ = -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.

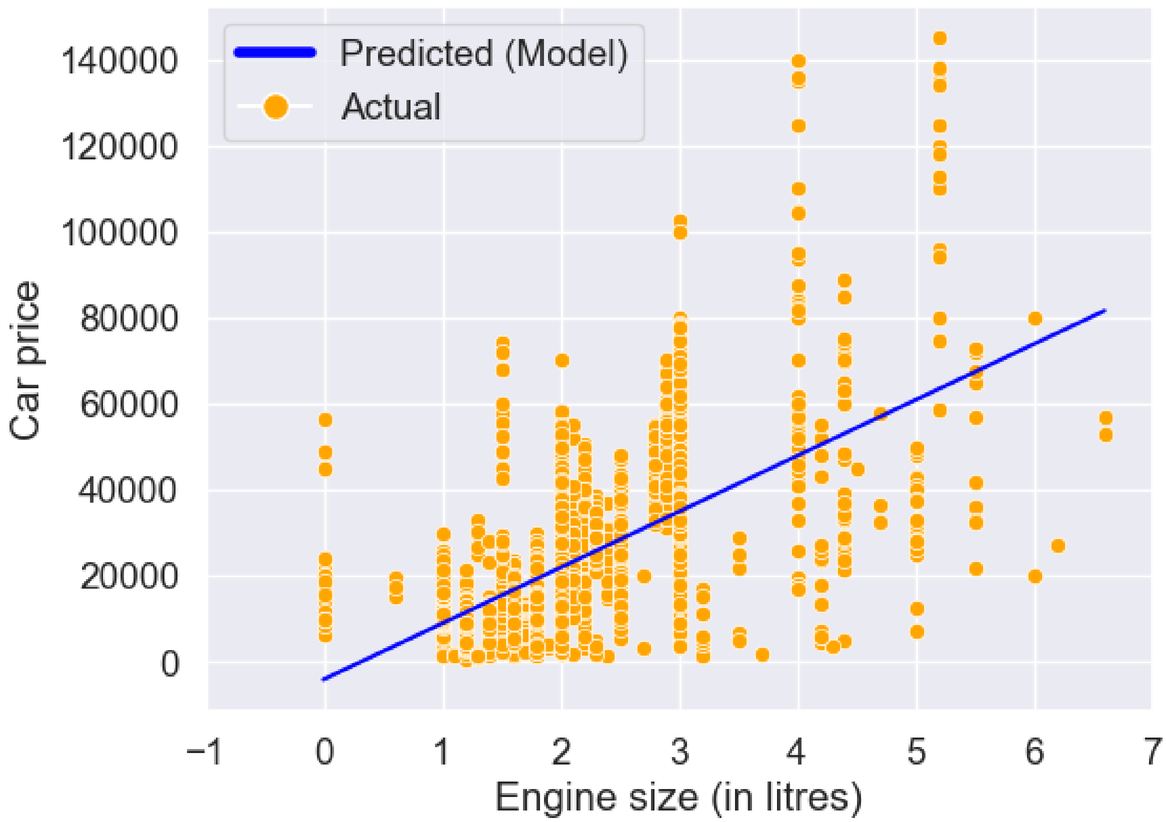
The coefficients can also be returned directly usign the `params` attribute of the `model` object returned by the `fit()` method of the `ols` class:

```
model.params
```

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

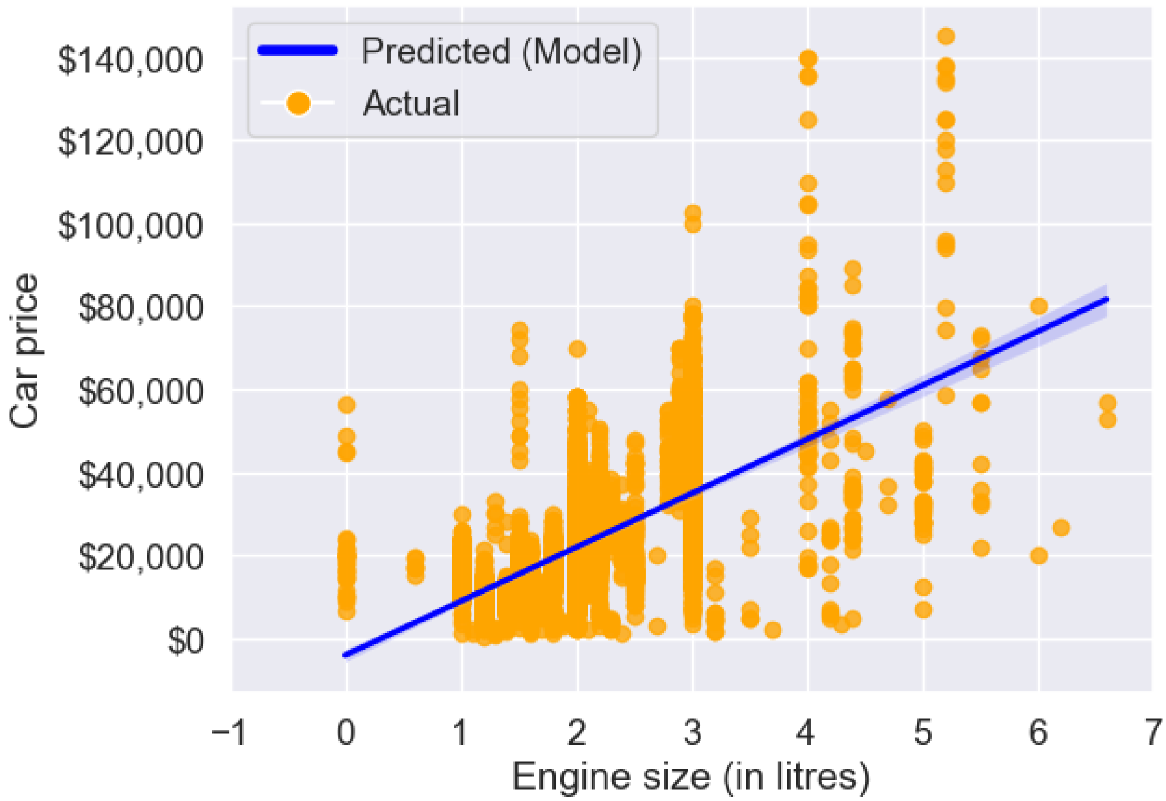
Visualize the regression line

```
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.

```python
ax = sns.regplot(x = 'engineSize', y = 'price', data = train, color = 'orange',line_kws={"col
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.

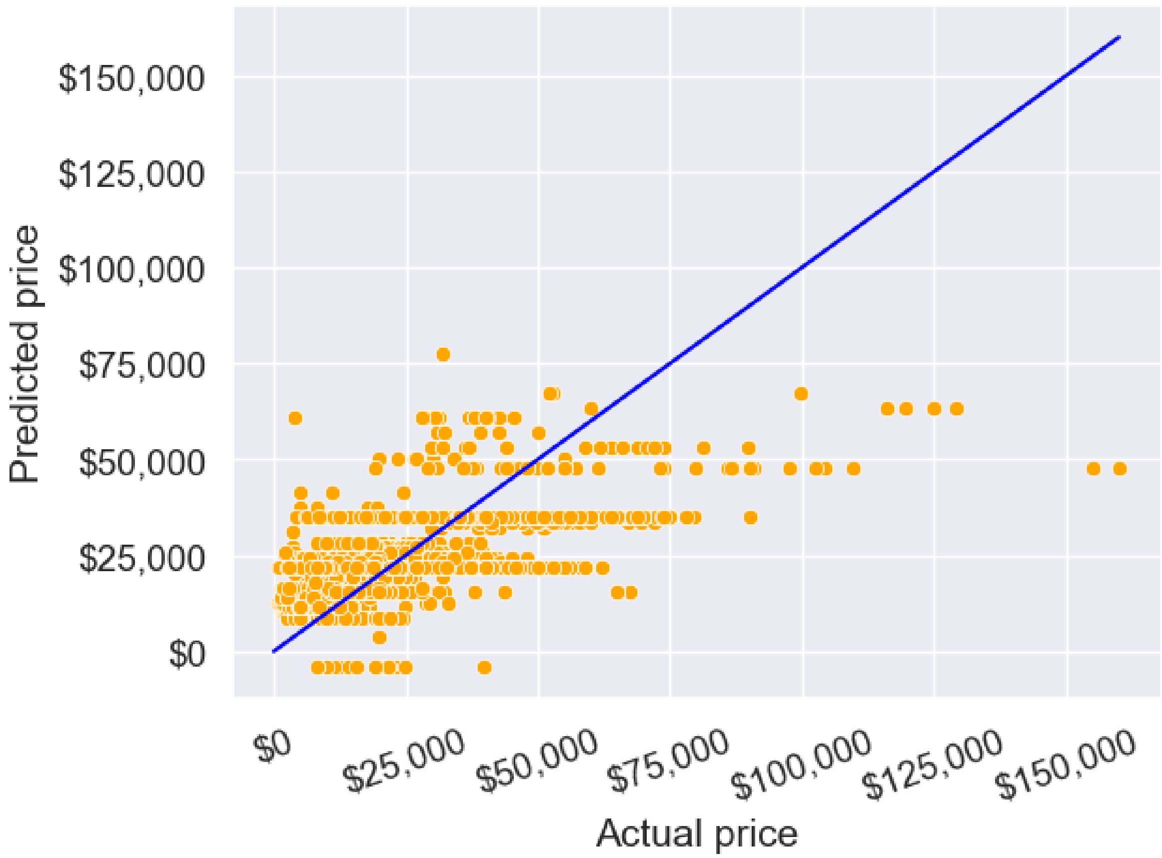
```python
# Read the test data
testf = pd.read_csv('./Datasets/Car_features_test.csv') # Predictors
```

```
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

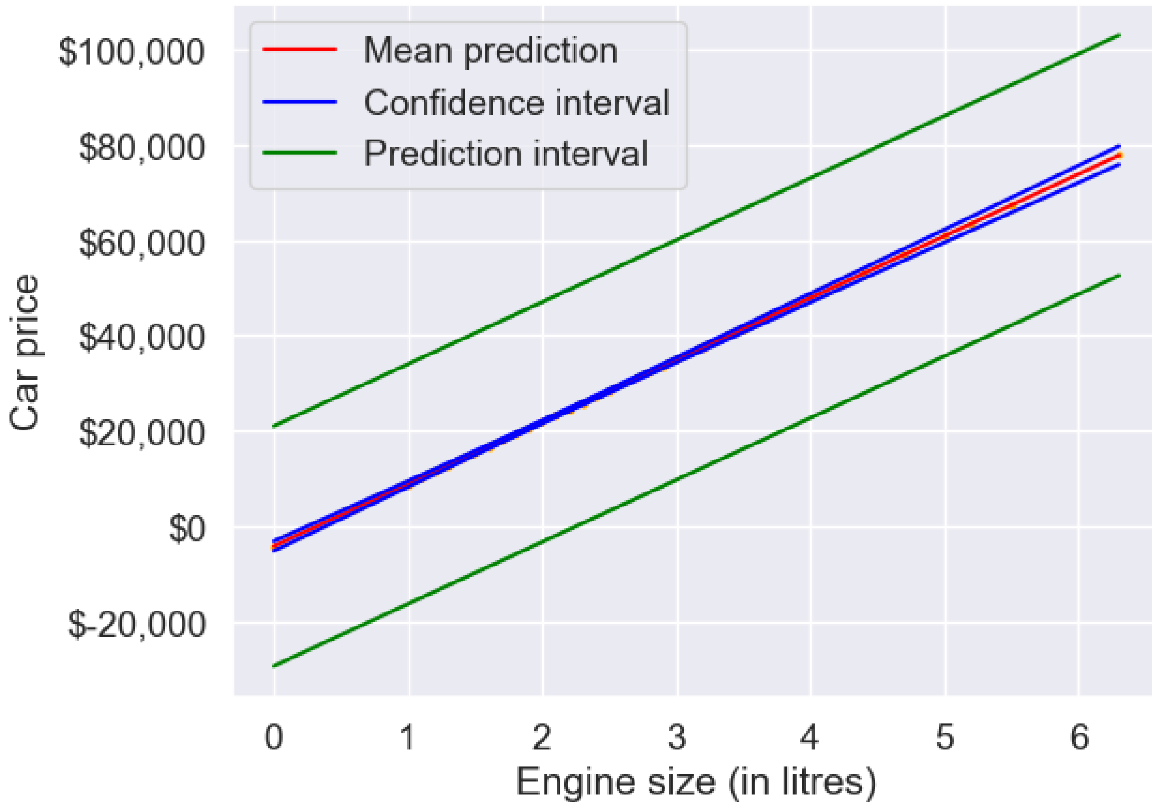
	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
2668	34842.807319	271.666459	34310.220826	35375.393812	9723.677232	59961.937406
2669	8866.245277	316.580850	8245.606701	9486.883853	-16254.905974	33987.396528
2670	21854.526298	184.135754	21493.538727	22215.513869	-3261.551421	46970.604017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

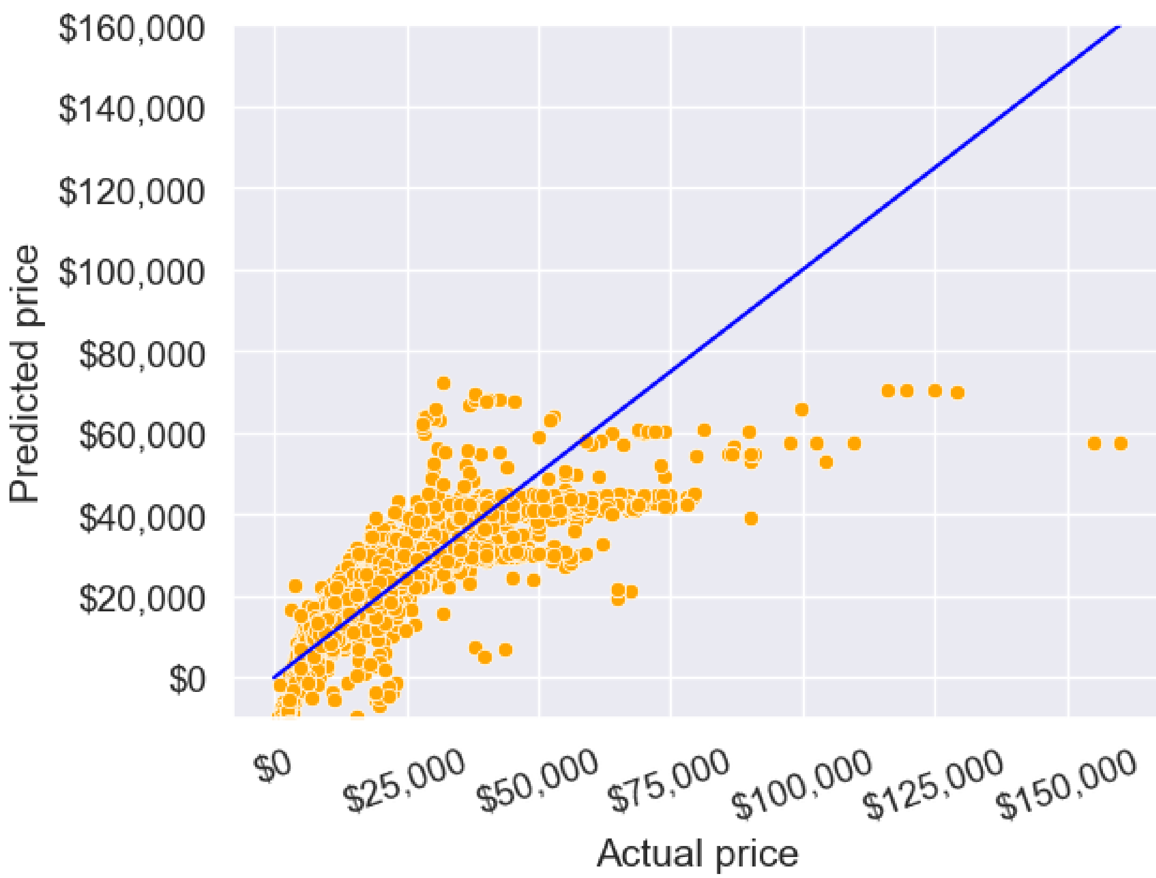
```
testf = pd.read_csv('./Datasets/Car_features_test.csv')
testp = pd.read_csv('./Datasets/Car_prices_test.csv')
```

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') #Plotti
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?

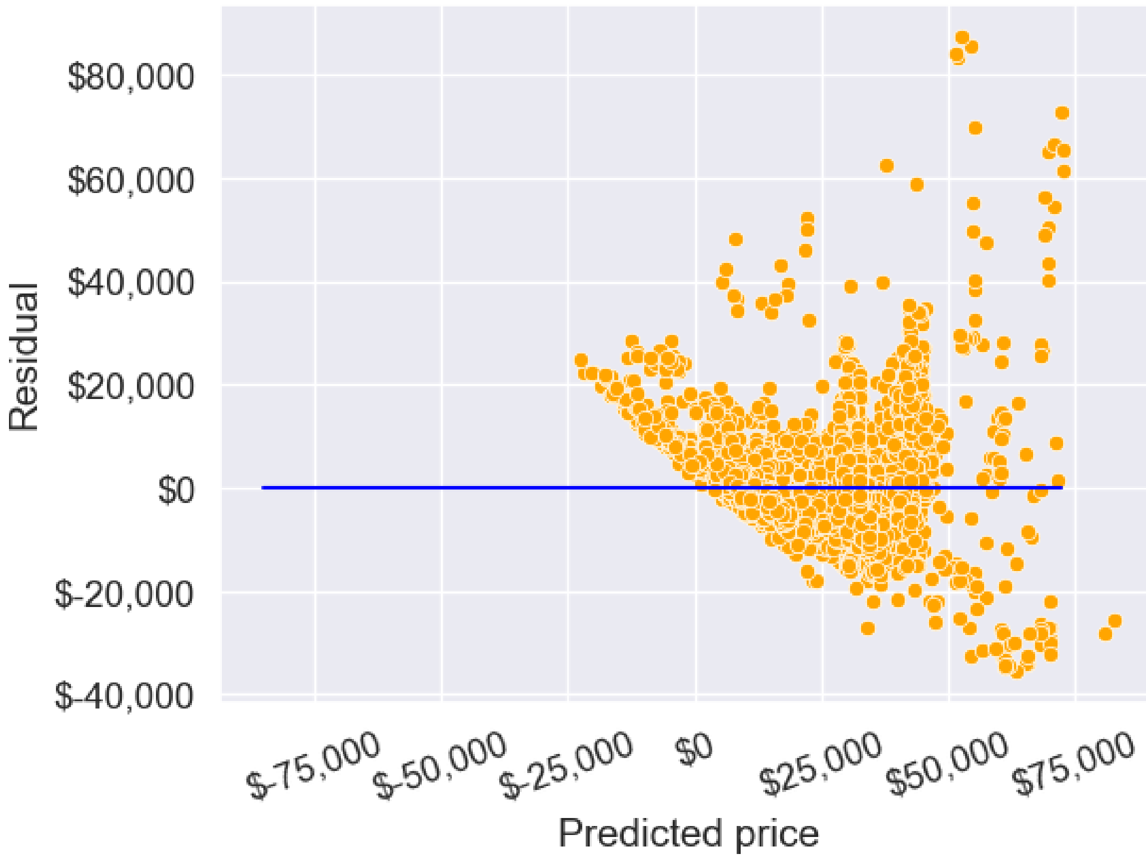
```
np.sqrt(model.mse_resid)
```

```
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:

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

or

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

or

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

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 $\approx$ \$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

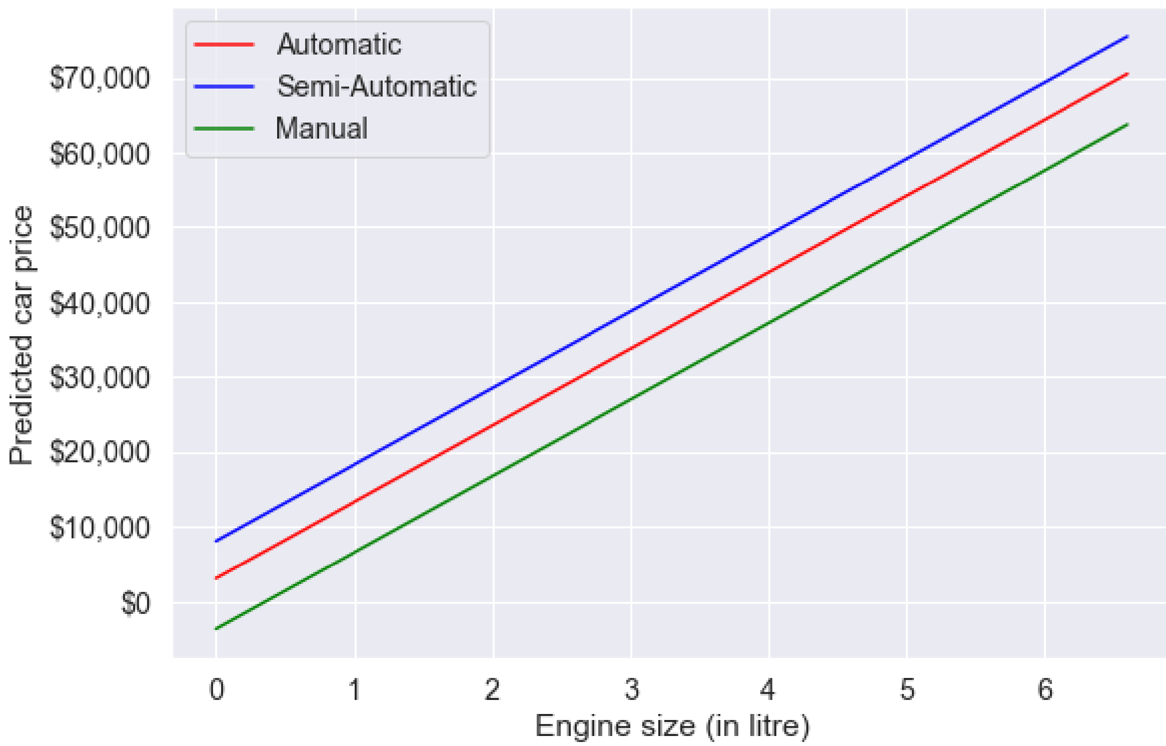
Ans: For the hypothetical scenario of a car with zero engine size and *Automatic* transmission, the estimated mean car price is ≈ \$3042.

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

Ans: The estimated mean price of a car with manual transmission is ≈ \$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]`.

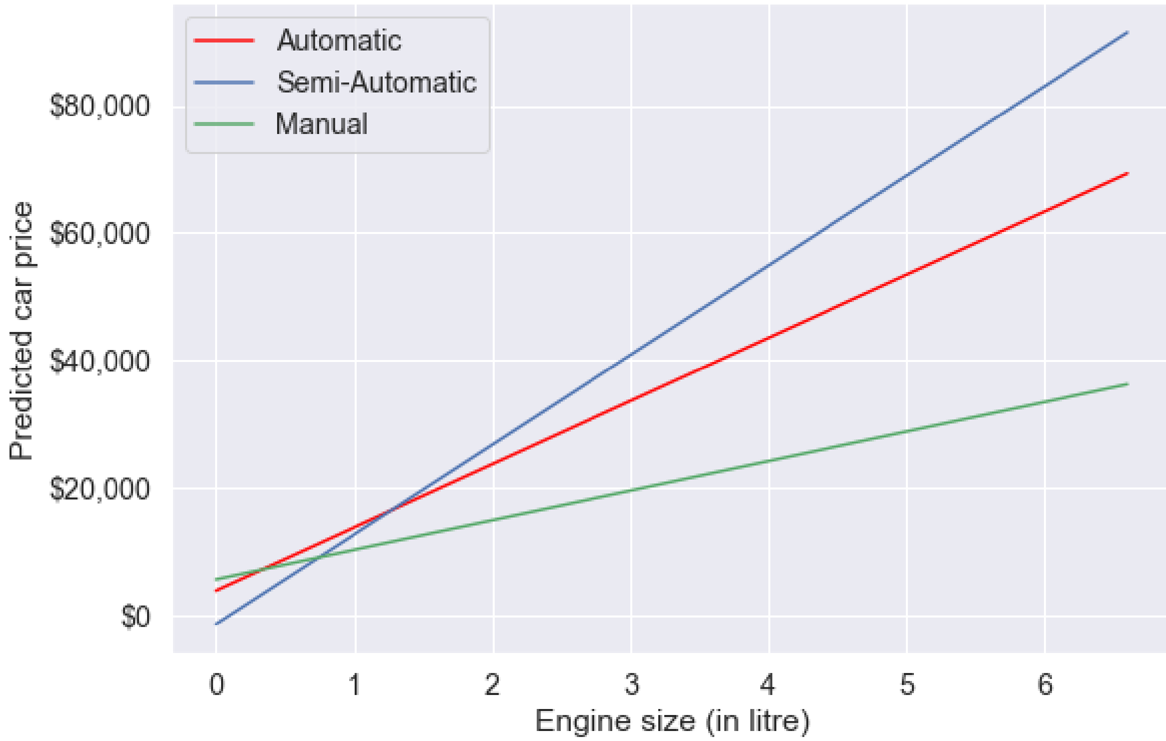
A: For a hypothetical scenario of zero engine size, the estimated mean `price` of a car with *Manual* `transmission` is ≈ \$1768 more than the estimated mean `price` of a car with *Automatic* `transmission`.

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.

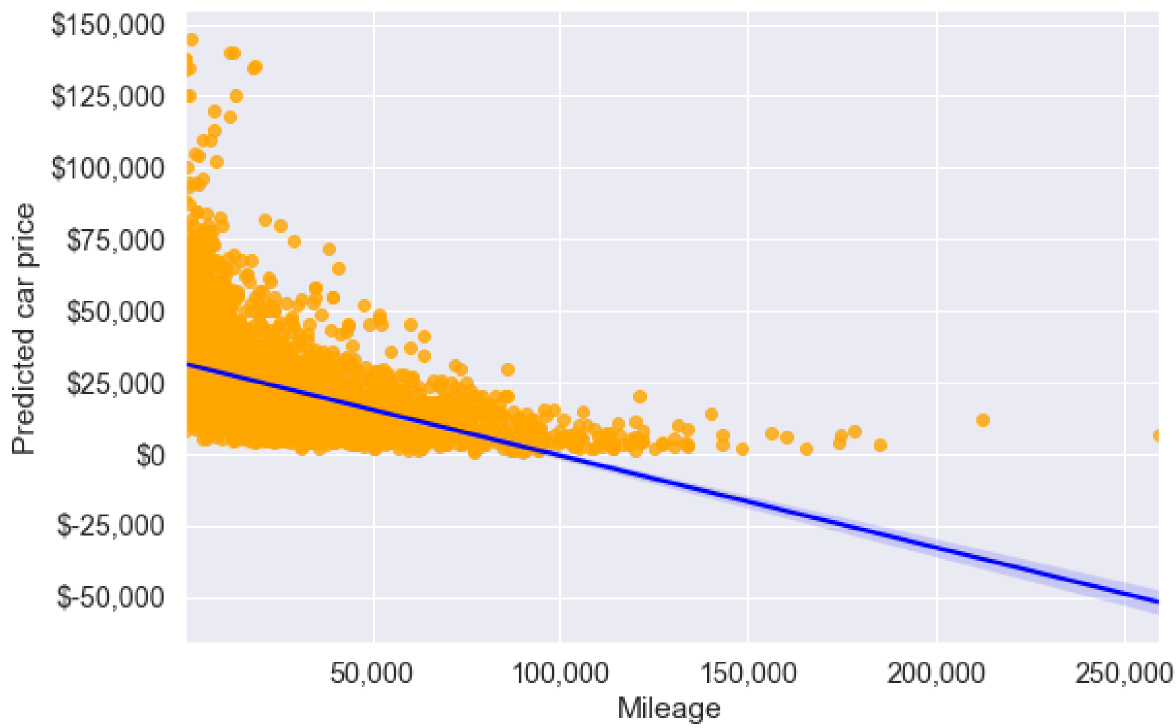
3.2 Variable transformations

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

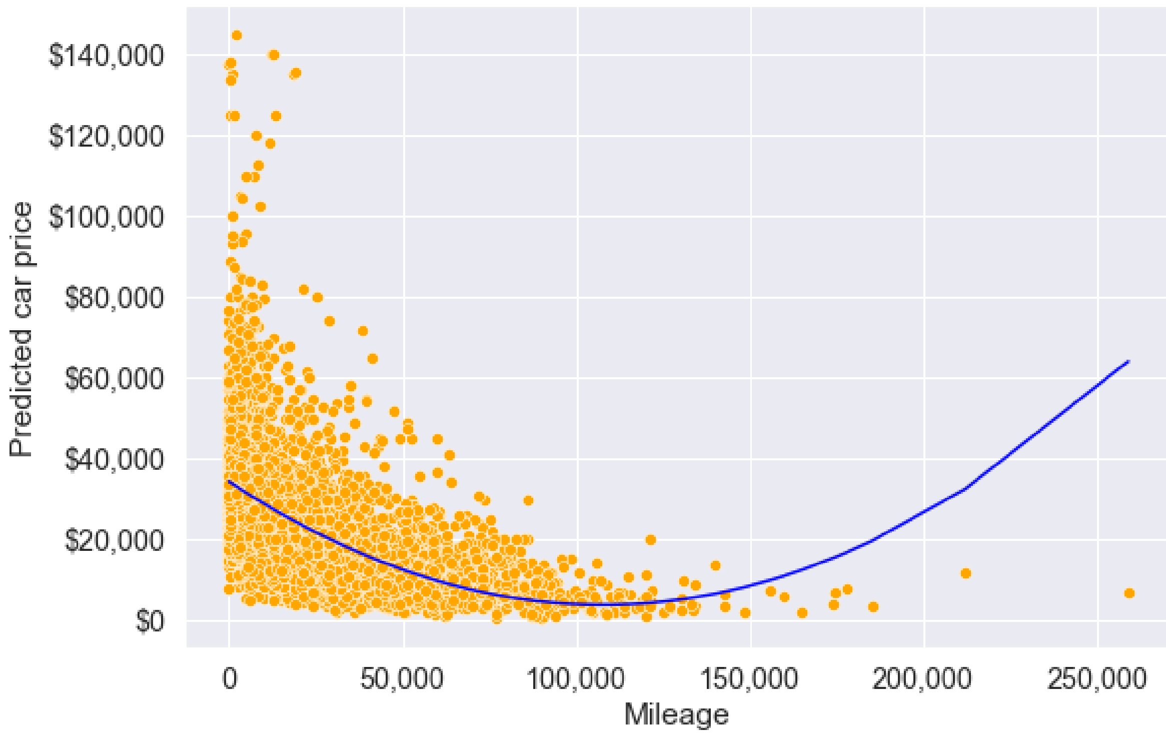
Omnibus:	2362.973	Durbin-Watson:	0.325
Prob(Omnibus):	0.000	Jarque-Bera (JB):	22427.952
Skew:	2.052	Prob(JB):	0.00
Kurtosis:	12.576	Cond. No.	4.81e+09

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()
```

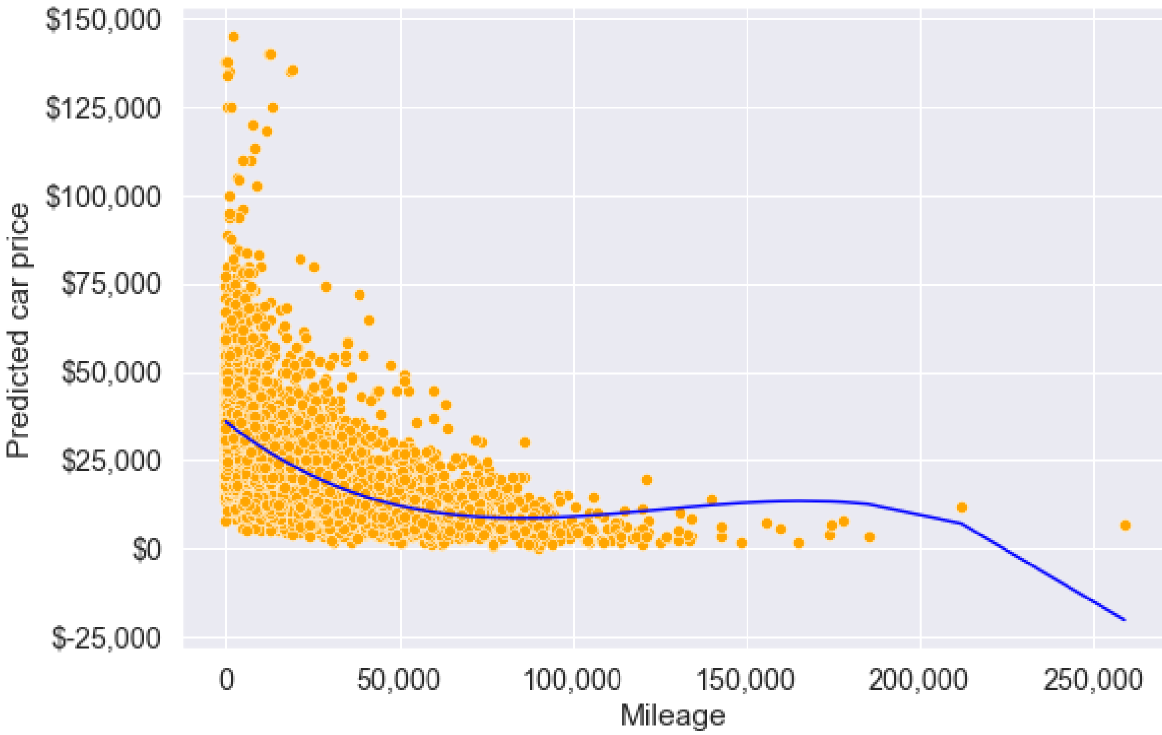
Table 3.17: OLS Regression Results

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.

Part II

Assignments

4 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.

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

4.1.1 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.**

4.1.2 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)**

4.1.3 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)**

4.1.4 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)**

4.1.5 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)**

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

4.2.1 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.

4.2.2 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.

4.3 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)

4.3.1 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.

4.3.2 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)**

4.3.3 3c)

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

4.3.4 3d)

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

4.3.5 3e)

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

4.3.6 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)**

4.3.7 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)**

4.3.8 3h)

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

4.3.9 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)**

4.3.10 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)**

4.3.11 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.

4.3.12 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)**

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

4.4.1 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.

4.4.2 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)**

4.5 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.

4.5.1 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.

4.5.2 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)**

4.5.3 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)**

5 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.

5.1 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.**

5.1.1 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)**

5.1.2 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)**

5.1.3 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)**

5.1.4 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)**

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

5.2.1 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.

5.2.2 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.

5.3 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)**.

5.4 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)

5.4.1 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.

5.4.2 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)**

5.4.3 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)**

5.4.4 4d)

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

5.4.5 4e)

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

5.4.6 4f)

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

5.4.7 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)**

5.4.8 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)**

5.4.9 4i)

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

5.4.10 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)**

5.4.11 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)**

5.4.12 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.

5.4.13 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)**

5.4.14 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)**

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

5.5.1 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.

5.5.2 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)**

5.6 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.

5.6.1 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.

5.6.2 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)**

5.6.3 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)**

6 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.

6.1 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

6.1.1 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)**

6.1.2 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)**

6.1.3 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)**

6.1.4 1d)

Check the p-values of `gleason` and `age`. Are these predictors statistically significant? **You need to justify your answer for credit. (2 points)**

6.1.5 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)**

6.1.6 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.

6.1.7 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)**

6.1.8 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

6.2 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.)

6.2.1 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.

6.2.2 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)**

6.2.3 2c)

Update the model you created in the previous part by adding `region` as a predictor. Print the model summary. **(2 points)**

6.2.4 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)**

6.2.5 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)

6.3 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.

6.3.1 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)**

6.3.2 3b)

Create a scatterplot of `gdpPerCapita` vs P. Does the relationship between `gdpPerCapita` and P seem linear or non-linear? **(2 points)**

6.3.3 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)**

6.3.4 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)

6.3.5 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)**

6.3.6 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)**

6.3.7 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)**

6.4 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.

A Datasets, assignment and project files

Datasets used in the book, assignment files, project files, and prediction problems report tempate can be found here