Innovation Centre for Education

Application of Machine Learning in Industries

Student Exercise Guide

Course code IAM03SG192v1.0

Industrial Applications of Machine Learning Lab

Contents

Exercise 1. Introduction to pandas' package

Estimated time:

00:30 minutes

What this exercise is about:

This exercise covers the following:

- Get started with pandas' package
- Explore basic functions available
- Simple dataframe manipulations.

What you should be able to do:

At the end of this exercise, you should be able to: Start pandas package to create dataframes and basic manipulation operations.

Introduction:

Since pandas is a large library with many different specialist features and functions, these exercises focus mainly on the fundamentals of manipulating data (indexing, grouping, aggregating, cleaning), making use of the core Data Frame and Series objects. A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Requirements:

- Anaconda 3
- Windows 7 and above
- Spyder

Instructor exercise overview:

Procedure:

Step 1: Import pandas under the name pd.

```
>import pandas as pd
```

Step 2: A pandas DataFrame can be created using the following constructor −pandas.DataFrame(data, index, columns). Create the following Python dictionary data and Python list labels:

```
>data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'dog'],
      'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
      'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
 'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}
>labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

Step 3: To create a DataFrame df we use the following function and pass the dictionary data which has the index labels.

```
>df = pd.DataFrame(data, index=labels)
```

Step 4: Display a summary of the basic information about this DataFrame and its data.

```
>df.info()#...or...
>df.describe()
```

Step 5: To return the first 5 rows of the DataFrame we use head function to check the top contents of the df.

```
> df.head()
```

Step 6: shape returns a tuple representing the dimensionality of the DataFrame. Tuple (a,b), where a represents the number of rows and b represents the number of columns.

```
>df.shape
```

Step 7: Selecting individual columns in dataframe. We can use [] and columns names in double quotes to select the required columns.

```
>df  [‘age’]
>df[['animal', 'age']]
```

Step 8: We can also slice the dataframe using indexes we created or using row and columns numbers. We use loc to slice based on index and iloc for integer-based slicing.

```
>print df.loc[:,'age']#select all rows for a specific column
```

```
>print df.loc[['a','b','f','h'],['animal’, ‘visits']]# Select few rows for multiple columns, say list[]
```

```
>print df.iloc[:4] #select all rows for 4rdcolumn, the format is .iloc(row number, column number)
```

```
>print df.iloc[1:5, 2:4] #integer slicing
```

Step 9: We can sort the data using the function sort_values()

```
>df.sort_values(‘age’,ascending=False)
```

Step 10: Summation functions in dataframes

```
>df['visits'].sum()#Calculate the sum of all visits
```

```
>df['age'].mean() # Mean age of all animals
```

```
>df['animal'].value_counts() #Count the number of each type of animal.
```

End of Exercise.

Exercise 2. Introduction to NumPy

Estimated time:

00:30 minutes

What this exercise is about:

This exercise covers the following:

- Understanding NumPy
- NumPy function
- Create NumPy arrays, indexing, slicing, etc.

What you should be able to do:

At the end of this exercise, you should be able to: Understanding NumPy and how it works.

Introduction:

One of the main benefits is its extensive set of libraries, a collection of routines and functions that help data scientists perform complex tasks quite effortlessly without writing a single line of code. One such important function is numerical Python aka NumPy which is a fundamental library, well known for high-performance multi-dimensional array and can be used for different mathematical functions like linear algebra, Fourier Transformations, etc. as well as logical operations.

This exercise takes a lowdown on understanding NumPy and its functions, including steps to create NumPy arrays, indexing, slicing, etc.

Requirements:

- Anaconda 3
- Windows 7 and above
- Spyder

Instructor exercise overview:

Procedure:

Step 1: Import numpy under the name np:

```
>import numpy as np
```

Step 2: Lists Of Lists for CSV Data: Before using NumPy, we'll first try to work with the data using Python and the csv package. We can read in the file using the csv.reader object, which will allow us to read in and split up all the content from the ssv file.

```
>import csv

>with open('winequality-red.csv', 'r') as f:

>wines = list(csv.reader(f, delimiter=';'))

>print(wines[:3])
```

The data has been read into a list of lists. Each inner list is a row from the ssv file. As you may have noticed, each item in the entire list of lists is represented as a string, which will make it harder to do computations.

Step 3: Creating numpy arrays:

- from python lists

>my_list = [1, 2, 3, 4, 5]

>my_numpy_list = np.array(my_list)

>my_numpy_list

- using arange() built-in function: This generates 10 digits of values from index 0 to 10.It is important to note that the arange() function can also take 3 arguments. The third argument signifies the step size of the operation.

>my_list = np.arange(10)#or

>my_list = np.arange(0,10)#or

>my_list = np.arange(0,11,2)

- using linspace() built-in function: The linspace() function returns numbers evenly spaced over a specified intervals. Say we want 15 evenly spaced points from 1 to 3.

>lin_arr = np.linspace(1, 3, 15)

Step 4: Generating an array of random numbers in NumPy: We can generate an array of random numbers using rand(), randn() or randint() functions.

Using random.rand(), we can generate an array of random numbers of the shape we pass to it from uniform distribution over 0 to 1.

>my_rand = np.random.rand(4)

>my_rand = np.random.rand(5, 4) #two-dimensional array of 5rows and 4columns

We can use the randint() function to generate an array of integers. The randint() function can take up to 3 arguments; the low(inclusive), high(exclusive) and size of the array.

>np.random.randint(20) #generates a random integer exclusive of 20

>np.random.randint(2, 20) #generates a random integer including 2 but excluding 20

>np.random.randint(2, 20, 7) #generates 7 random integers including 2 but excluding 20

Step 5: Indexing/Slicing NumPy Arrays: We now know how to create arrays, but unless we can retrieve results from them, there isn't a lot we can do with NumPy. We can use array indexing to select individual elements, groups of elements, or entire rows and columns.We'll again work with the wines array:Let's select the element at row 3 and column 4.

>wines[2,3]

If we instead want to select the first three items from the fourth column, we can do it using a colon (:).

>wines[0:3,3]

>wines[:,3]

>wines[3,:]

Step 6: Assigning Values to NumPy Arrays: We can also use indexing to assign values to certain elements in arrays. We can do this by assigning directly to the indexed value >wines[1,5] = 10

We can do the same for slices. To overwrite an entire column

>wines[:,10] = 50

Step 7: Reshaping NumPy Arrays: We can change the shape of arrays while still preserving all of their elements. This often can make it easier to access array elements. The simplest reshaping is to flip the axes, so rows become columns, and vice versa. We can accomplish this with the numpy.transpose function

>np.transpose(wines).shape

We can use the numpy.ravel function to turn an array into a one-dimensional representation. It will essentially flatten an array into a long sequence of values:

>wines.ravel()

Here's an example where we can see the ordering of numpy.ravel:

>array_one = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>array_one.ravel()

Converting one-dimensional array to two-dimensional

>arr = np.random.rand(25)

>arr.reshape(5,5)

The reshape() can only convert to equal number or rows and columns and must together be equal to equal to the number of elements. In the example above, arr contained 25 elements hence can only be reshaped to a 5X5 matrix.

Step 8: NumPy Data Types: You can find the data type of a NumPy array by accessing the dtype property.

>wines.dtype

NumPy has several different data types, which mostly map to Python data types, like float, and str.You can use the numpy.ndarray.astype method to convert an array to a different type.

```
>wines.astype(int)
```

Step 9: NumPy Array Operations.

1. Single Array Math: If you do any of the basic mathematical operations (/, *, -, +, ^) with an array and a value, it will apply the operation to each of the elements in the array.Let's say we want to add 10 points to each quality score because we're drunk and feeling generous. Here's how we'd do that:

```
>wines[:,11] + 10
```

2. Multiple Array Math: It's also possible to do mathematical operations between arrays. This will apply the operation to pairs of elements. For example, if we add the quality column to itself, here's what we get:

```
>wines[:,11] + wines[:,11]
```

```
>wines[:,10] * wines[:,11]
```

Step 10: NumPy Array Methods: In addition to the common mathematical operations, NumPy also has several methods that you can use for more complex calculations on arrays. An example of this is the numpy.ndarray.sum method. This finds the sum of all the elements in an array by default:

```
>wines[:,11].sum()
```

The total of all of our quality ratings is 154.1788. We can pass the axis keyword argument into the sum method to find sums over an axis. If we call sum across the wines matrix, and pass in axis=0, we'll find the sums over the first axis of the array. This will give us the sum of all the values in every column. This may seem backwards that the sums over the first axis would give us the sum of each column, but one way to think about this is that the specified axis is the one "going away". So if we specify axis=0, we want the rows to go away, and we want to find the sums for each of the remaining axes across each row:

```
>wines.sum(axis=0)
```

We can verify that we did the sum correctly by checking the shape. The shape should be 12, corresponding to the number of columns:

```
>wines.sum(axis=0).shape
```

If we pass in axis=1, we'll find the sums over the second axis of the array. This will give us the sum of each row:

```
>wines.sum(axis=1)
```

There are several other methods that behave like the sum method, including:

- numpy.ndarray.mean — finds the mean of an array.
- numpy.ndarray.std — finds the standard deviation of an array.

- numpy.ndarray.min — finds the minimum value in an array.
- numpy.ndarray.max — finds the maximum value in an array.

Step 11: NumPy Array Comparisons: NumPy makes it possible to test to see if rows match certain values using mathematical comparison operations like <, >, >=, <=, and ==. For example, if we want to see which wines have a quality rating higher than 5, we can do this:

```
>wines[:,11] > 5
```

Subsetting : One of the powerful things we can do with a Boolean array and a NumPy array is select only certain rows or columns in the NumPy array. For example, the below code will only select rows in wines where the quality is over 7-

```
>high_quality = wines[:,11] > 7
```

```
>wines[high_quality,:][:3,:]
```

We select only the rows where high_quality contains a True value, and all of the columns. This subsetting makes it simple to filter arrays for certain criteria. For example, we can look for wines with a lot of alcohol and high quality. In order to specify multiple conditions, we have to place each condition in parentheses, and separate conditions with an ampersand (&):

```
>high_quality_and_alcohol = (wines[:,10] > 10) & (wines[:,11] > 7)
```

```
>wines[high_quality_and_alcohol,10:]
```

End of Exercise.

Exercise 3. Wine Quality Prediction

Estimated time:

30:00 minutes

What this exercise is about:

This notebook demonstrates how to predict the quality of the wine based on certain physicochemical (inputs) and sensory (the output) variables.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean and prepare the data set and perform Exploratory Data Analysis.
- How to build different classifier models to predict the wine quality.
- Compare different models select the best model based on accuracy

Introduction:

Currently the wine quality is done by human testers. To speed up the production process, the machine learning techniques can be adopted. Let's build a model to predict the quality of the wine.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
> import pandas as pd, numpy as np

> from sklearn.preprocessing import StandardScaler

> from sklearn.linear_model import LinearRegression

> from sklearn.neighbors import KNeighborsClassifier

> from sklearn.ensemble import RandomForestClassifier

> from sklearn.svm import SVC

> from sklearn import model_selection

> from sklearn.model_selection import train_test_split
```

> import warnings

> warnings.filterwarnings('ignore')

> import matplotlib.pyplot as plt

> import seaborn as sns

Step 2: Download dataset from

https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009/download/gqNqzdSwu7p4knmgywCj%2Fversions%2FOhz2Dynugco5o8aokFXA%2Ffiles%2Fwinequality-red.csv?datasetVersionNumber=2 and load to pandas data frame.

> wine = pd.read_csv("winequality-red.csv")

> wine.head()

Step 3: Data preparation and cleaning.

Let's check the dimensions of the dataframe

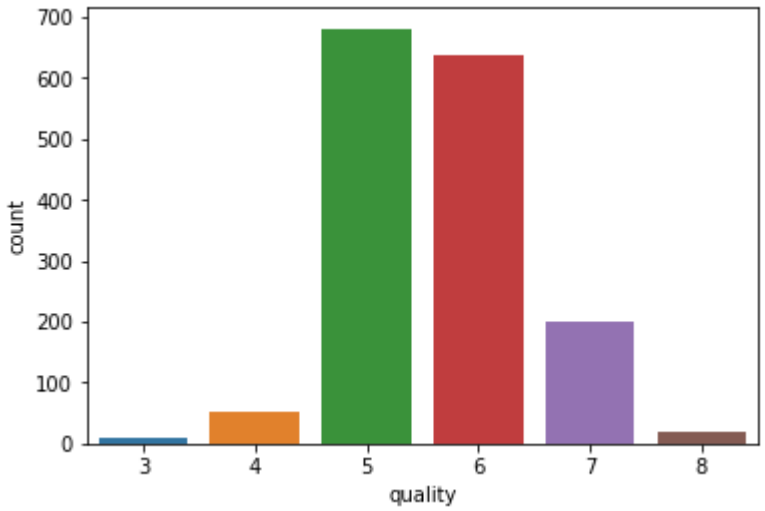
>wine.shape

Let's check the missing values in the dataframe

round(100*(wine.isnull().sum()/len(wine.index)), 2)

```
fixed acidity           0.0
volatile acidity        0.0
citric acid             0.0
residual sugar          0.0
chlorides               0.0
free sulfur dioxide     0.0
total sulfur dioxide    0.0
density                 0.0
pH                      0.0
sulphates               0.0
alcohol                 0.0
quality                 0.0
dtype: float64
```

>sns.countplot(x='quality', data=wine)



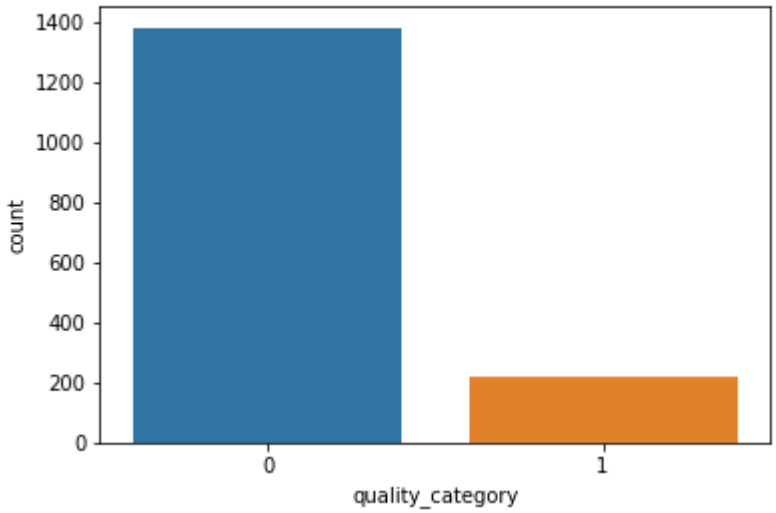
#Classify quality into three groups

```
>def classify_quality(x):
    >if x <= 6:
        >quality = 0
    >else:
        >quality = 1
    >return quality
>wine['quality_category'] = wine.quality.apply(classify_quality)
>wine.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	quality_category
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0

```
>sns.countplot(x='quality_category', data=wine)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xe471208>
```

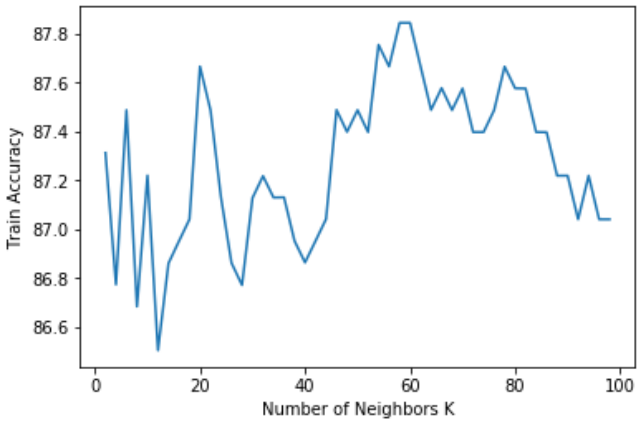



```
>wine.drop('quality', axis = 1, inplace=True)
>X = wine.drop('quality_category', axis = 1)
>y = wine['quality_category']
```

Step 4: Rescaling and Split the data set into test and train.

```
>X_std = StandardScaler().fit_transform(X)
# Splitting the data into train and test
>X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.30, random_state=100)
>neighbors = [x for x in list(range(1,100)) if x % 2 == 0]
>cv_scores = []
# Perform 10-fold cross validation on training set for odd values of k:
>seed=123
>for k in neighbors:
  > k_value = k+1
   >knn = KNeighborsClassifier(n_neighbors = k_value, weights='uniform', p=2, metric='euclidean')
   >kfold = model_selection.KFold(n_splits=10, random_state=seed)
   >scores = model_selection.cross_val_score(knn, X_train, y_train, cv=kfold, scoring='accuracy')
   >cv_scores.append(scores.mean()*100)
```

```
>optimal_k = neighbors[cv_scores.index(max(cv_scores))]

>print(optimal_k)

#print(( "The optimum number of neighbors is %d with %0.1f%%" % (optimal_k, cv_scores[optimal_k])))

>plt.plot(neighbors, cv_scores)

>plt.xlabel('Number of Neighbors K')

>plt.ylabel('Train Accuracy')

>plt.show()
```



Step 5: Model Building

```
## Use KNN Classifier. Let this be base model.

>knn = KNeighborsClassifier(n_neighbors = 58)

>knn.fit(X_train, y_train)
```

Output:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=58, p=2,
           weights='uniform')
```

```
>Y_pred = knn.predict(X_test)

>acc_train = round(knn.score(X_train, y_train) * 100, 2)

>acc_val = round(knn.score(X_test, y_test) * 100, 2)

>print("Accuracy of training dataset: " + str(acc_train))

>print("Accuracy of test dataset: "+ str(acc_val))
```

```
Accuracy of training dataset: 88.29
Accuracy of test dataset: 88.96
```

Let's check accuracy of Randomforest classifier

>rfc = RandomForestClassifier(n_estimators=200)

>rfc.fit(X_train, y_train)

>pred_rfc = rfc.predict(X_test)

>acc_train = round(rfc.score(X_train, y_train) * 100, 2)

>acc_val = round(rfc.score(X_test, y_test) * 100, 2)

>print("Accuracy of training dataset: " + str(acc_train))

>print("Accuracy of test dataset: "+ str(acc_val))

```
Accuracy of training dataset: 100.0
Accuracy of test dataset: 91.04
```

Let's check accuracy of SV classifier

>svc = SVC(kernel='rbf')

>svc.fit(X_train, y_train)

>pred_svc = svc.predict(X_test)

>acc_train = round(svc.score(X_train, y_train) * 100, 2)

>acc_val = round(svc.score(X_test, y_test) * 100, 2)

>print("Accuracy of training dataset: " + str(acc_train))

>print("Accuracy of test dataset: "+ str(acc_val))

```
Accuracy of training dataset: 89.63
Accuracy of test dataset: 88.12
```

Step 6: Summary

> 1. We have built KNN, RandomForest and SVC classifiers models.

> 2. We can cleary observe that the RandomForest model gives more accurate results predicting the wine quality.

> Apply other classification models and check the accuracy.

End of Exercise.

Exercise 4. Housing Price Prediction

Estimated time:

60:00 minutes

What this exercise is about:

This notebook demonstrates how to build a regression model to predict the price of a house given its specifications.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean data and convert categorial data to scalar.
- Build and train lasso regression model for house price prediction.
- Next steps to boost the performance of the model / prediction.

Introduction:

A real estate company wants to buy houses at prices lesser than their actual value and sell at higher prices. The company wants to use data analytics to identify the factors which are significant in predicting the price of the house. Let's build a model for identifying the features that affect the price of the house.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

Step 2: Download dataset from

https://www.kaggle.com/c/house-prices-advanced-regression-techniques/download/EGpnHbJBO1XGF5B35WNU%2Fversions%2FHOSha0bPv4HEBGjo9O9x%2Ffiles%2Ftrain.csvand load to pandas data frame

```
>housing = pd.read_csv("train.csv")

>housing.head()
```

Step 3: Data Preparation by splitting the data into train and test.

Let's check the dimensions of the dataframe

>housing.shape

#checking duplicates

>sum(housing.duplicated(subset = 'Id')) == 0

No duplicate values

#Data Cleaning

Let's check the missing values in the dataframe

>round(100*(housing.isnull().sum()/len(housing.index)), 2)

```
Id               0.00
MSSubClass       0.00
MSZoning         0.00
LotFrontage     17.74
LotArea          0.00
Street           0.00
Alley           93.77
LotShape         0.00
LandContour      0.00
Utilities        0.00
LotConfig        0.00
LandSlope        0.00
Neighborhood     0.00
Condition1       0.00
Condition2       0.00
BldgType         0.00
HouseStyle       0.00
OverallQual      0.00
OverallCond      0.00
YearBuilt        0.00
YearRemodAdd     0.00
RoofStyle        0.00
RoofMatl         0.00
Exterior1st      0.00
Exterior2nd      0.00
MasVnrType       0.55
MasVnrArea       0.55
ExterQual        0.00
ExterCond        0.00
Foundation       0.00
BsmtQual         2.53
```

BsmtQual	2.53
BsmtCond	2.53
BsmtExposure	2.60
BsmtFinType1	2.53
BsmtFinSF1	0.00
BsmtFinType2	2.60
BsmtFinSF2	0.00
BsmtUnfSF	0.00
TotalBsmtSF	0.00
Heating	0.00
HeatingQC	0.00
CentralAir	0.00
Electrical	0.07
1stFlrSF	0.00
2ndFlrSF	0.00
LowQualFinSF	0.00
GrLivArea	0.00
BsmtFullBath	0.00
BsmtHalfBath	0.00
FullBath	0.00
HalfBath	0.00
BedroomAbvGr	0.00
KitchenAbvGr	0.00
KitchenQual	0.00
TotRmsAbvGrd	0.00
Functional	0.00
Fireplaces	0.00
FireplaceQu	47.26
GarageType	5.55
GarageYrBlt	5.55
GarageFinish	5.55

```
GarageArea        0.00
GarageQual        5.55
GarageCond        5.55
PavedDrive        0.00
WoodDeckSF        0.00
OpenPorchSF       0.00
EnclosedPorch     0.00
3SsnPorch         0.00
ScreenPorch       0.00
PoolArea          0.00
PoolQC           99.52
Fence            80.75
MiscFeature      96.30
MiscVal           0.00
MoSold            0.00
YrSold            0.00
SaleType          0.00
SaleCondition     0.00
SalePrice         0.00
dtype: float64
```

Step 4: Model Building.

we will drop the columns having more than 40% NA values.

>housing = housing.drop(housing.loc[:,list(round(100*(housing.isnull().sum()/len(housing.index)), 2)>40)].columns, 1)

let's look at the statistical aspects of the dataframe

>housing.describe()

Output:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46.549315
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	161.319273
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	0.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000

BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr
1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
567.240411	1057.429452	1162.626712	346.992466	5.844521	1515.463699	0.425342	0.057534	1.565068	0.382877	2.866438
441.866955	438.705324	386.587738	436.528436	48.623081	525.480383	0.518911	0.238753	0.550916	0.502885	0.815778
0.000000	0.000000	334.000000	0.000000	0.000000	334.000000	0.000000	0.000000	0.000000	0.000000	0.000000
223.000000	795.750000	882.000000	0.000000	0.000000	1129.500000	0.000000	0.000000	1.000000	0.000000	2.000000
477.500000	991.500000	1087.000000	0.000000	0.000000	1464.000000	0.000000	0.000000	2.000000	0.000000	3.000000
808.000000	1298.250000	1391.250000	728.000000	0.000000	1776.750000	1.000000	0.000000	2.000000	1.000000	3.000000
2336.000000	6110.000000	4692.000000	2065.000000	572.000000	5642.000000	3.000000	2.000000	3.000000	2.000000	8.000000

KitchenAbvGr	TotRmsAbvGrd	Fireplaces	GarageYrBlt	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch
1460.000000	1460.000000	1460.000000	1379.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
1.046575	6.517808	0.613014	1978.506164	1.767123	472.980137	94.244521	46.660274	21.954110	3.409589	15.060959
0.220338	1.625393	0.644666	24.689725	0.747315	213.804841	125.338794	66.256028	61.119149	29.317331	55.757415
0.000000	2.000000	0.000000	1900.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	5.000000	0.000000	1961.000000	1.000000	334.500000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	6.000000	1.000000	1980.000000	2.000000	480.000000	0.000000	25.000000	0.000000	0.000000	0.000000
1.000000	7.000000	1.000000	2002.000000	2.000000	576.000000	168.000000	68.000000	0.000000	0.000000	0.000000
3.000000	14.000000	3.000000	2010.000000	4.000000	1418.000000	857.000000	547.000000	552.000000	508.000000	480.000000

PoolArea	MiscVal	MoSold	YrSold	SalePrice
460.000000	1460.000000	1460.000000	1460.000000	1460.000000
2.758904	43.489041	6.321918	2007.815753	180921.195890
40.177307	496.123024	2.703626	1.328095	79442.502883
0.000000	0.000000	1.000000	2006.000000	34900.000000
0.000000	0.000000	5.000000	2007.000000	129975.000000
0.000000	0.000000	6.000000	2008.000000	163000.000000
0.000000	0.000000	8.000000	2009.000000	214000.000000
738.000000	15500.000000	12.000000	2010.000000	755000.000000

>housing['SalePrice'].describe()

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

More than 95% of the below columns are same i.e. there is no variation in the data. Hence they are not useful for analysis.

```
>housing.drop(['Condition2', 'BsmtFinType2', 'BsmtFinSF2', 'Heating', 'LandContour', 'Street', 'Utilities', 'LandSlope', 'RoofMatl', 'LowQualFinSF', 'MiscVal', 'PoolArea', 'CentralAir'], axis=1, inplace=True)

# Quality measurements are stored as text but we can convert them to

# numbers where a higher number means higher quality.


>qual_dict = {None: 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5}

>housing["ExterQual"] = housing["ExterQual"].map(qual_dict).astype(int)

>housing["ExterCond"] = housing["ExterCond"].map(qual_dict).astype(int)


>housing["BsmtQual"] = housing["BsmtQual"].map(qual_dict).astype(int)

>housing["BsmtCond"] = housing["BsmtCond"].map(qual_dict).astype(int)

>housing["KitchenQual"] = housing["KitchenQual"].map(qual_dict).astype(int)

# housing["FireplaceQu"] = housing["FireplaceQu"].map(qual_dict).astype(int)


>housing["GarageQual"] = housing["GarageQual"].map(qual_dict).astype(int)

>housing["GarageCond"] = housing["GarageCond"].map(qual_dict).astype(int)

>housing["HeatingQC"] = housing["HeatingQC"].map(qual_dict).astype(int)

>bsmt_expo = {None: 0, "No": 1, "Mn": 2, "Av": 3, "Gd": 4}

>housing["BsmtExposure"] = housing["BsmtExposure"].map(bsmt_expo).astype(int)

>pave_drive = {"N": 0, "P": 1, "Y": 2}

>housing["PavedDrive"] = housing["PavedDrive"].map(pave_drive).astype(int)

## fill the missing values with mode for below columns

>housing['GarageType'] = housing['GarageType'].transform(lambda x: x.fillna(x.mode()[0]))

>housing['GarageYrBlt'] = housing['GarageYrBlt'].transform(lambda x: x.fillna(x.mode()[0]))

>housing['LotFrontage'] = housing['LotFrontage'].transform(lambda x: x.fillna(x.mode()[0]))

>grge_finish = {None: 0, "No": 1, "Unf": 2, "RFn": 3, "Fin": 4}

>housing["GarageFinish"] = housing["GarageFinish"].map(grge_finish).astype(int)
```

```
>electrical = {None: 0, "Mix": 1, "FuseP": 2, "FuseF": 3, "FuseA": 4, "SBrkr": 5}

>housing["Electrical"] = housing["Electrical"].map(electrical).astype(int)

# 4 -> Good

# 3 -> Avg

# 2 -> Below Avg / Low quality

# 1 -> Low Quality

# 0 -> No basement

>bsmt_finish = {None: 0, "Unf": 1, "LwQ": 2, "BLQ": 2, "Rec": 3, "ALQ": 3, "GLQ": 4}

>housing["BsmtFinType1"] = housing["BsmtFinType1"].map(bsmt_finish).astype(int)

## Convert YearBuilt to Age of House in Years

>housing["AgeInYears"] = 2019 - housing['YrSold']

>housing.drop(['YearBuilt'], axis=1, inplace=True)

## Convert YrSold and MoSold to Years

>housing["YearsSinceSold"] = 2019 - (housing['YrSold'] + housing['MoSold']/12)

>housing.drop(['MoSold', 'YrSold'], axis=1, inplace=True)

>housing["YearsSinceSold"] = housing["YearsSinceSold"].astype(int)

>housing['TotalPorchArea'] = housing['OpenPorchSF'] + housing['EnclosedPorch'] + housing['3SsnPorch'] +
housing['ScreenPorch']


>housing.drop(['OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch'], axis=1, inplace=True)

## We have GrFlrArea which is sum of 1stFlrSF and 2ndFlrSF. Hence drop the 2ndFlrSF.

>housing.drop(['2ndFlrSF'], axis=1, inplace=True)


# More than 50% of data doesn't have MasVnrType. Hence not considering this column and corresponding
area as well.

>housing.drop(['MasVnrType', 'MasVnrArea' ], axis=1, inplace=True)

#housing['MasVnrType'] = housing['MasVnrType'].transform(lambda x: x.fillna(x.mode()[0]))

#housing['MasVnrArea'] = housing['MasVnrArea'].transform(lambda x: x.fillna('0')).astype(int)
```

Step 5: Checking for outliers

>numeric_features = housing.select_dtypes(include=[np.number])

>numeric_features.columns

Output :

```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearRemodAdd', 'ExterQual', 'ExterCond', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtUnfSF',
       'TotalBsmtSF', 'HeatingQC', 'Electrical', '1stFlrSF', 'GrLivArea',
       'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
       'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'SalePrice', 'AgeInYears',
       'YearsSinceSold', 'TotalPorchArea'],
      dtype='object')
```

Checking outliers at 25%,50%,75%,90%,95% and 99%

housing[numeric_features.columns].describe(percentiles=[.25,.5,.75,.90,.95,.99])

Output :

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearRemodAdd	ExterQual	ExterCond	BsmtQual	BsmtCond
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.00000	1460.000000	1460.000000	1460.000000
mean	730.500000	56.897260	68.267123	10516.828082	6.099315	5.575342	1984.865753	3.39589	3.083562	3.489041	2.934932
std	421.610009	42.300571	22.356355	9981.264932	1.382997	1.112799	20.645407	0.57428	0.351054	0.876478	0.552159
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1950.000000	2.00000	1.000000	0.000000	0.000000
25%	365.750000	20.000000	60.000000	7553.500000	5.000000	5.000000	1967.000000	3.00000	3.000000	3.000000	3.000000
50%	730.500000	50.000000	63.000000	9478.500000	6.000000	5.000000	1994.000000	3.00000	3.000000	4.000000	3.000000
75%	1095.250000	70.000000	79.000000	11601.500000	7.000000	6.000000	2004.000000	4.00000	3.000000	4.000000	3.000000
90%	1314.100000	120.000000	92.000000	14381.700000	8.000000	7.000000	2006.000000	4.00000	4.000000	4.000000	3.000000
95%	1387.050000	160.000000	104.000000	17401.150000	8.000000	8.000000	2007.000000	4.00000	4.000000	5.000000	3.000000
99%	1445.410000	190.000000	137.410000	37567.640000	10.000000	9.000000	2009.000000	5.00000	4.000000	5.000000	4.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	5.00000	5.000000	5.000000	4.000000

1. outlier treatment

removing (statistical) outliers

>Q1 = housing['LotArea'].quantile(0.05)

>Q3 = housing['LotArea'].quantile(0.95)

>IQR = Q3 - Q1

>housing = housing[(housing['LotArea'] >= Q1 - 1.5*IQR) & (housing['LotArea'] <= Q3 + 1.5*IQR)]

```
>Q1 = housing['BsmtFinSF1'].quantile(0.05)
>Q3 = housing['BsmtFinSF1'].quantile(0.95)
>IQR = Q3 - Q1
>housing = housing[(housing['BsmtFinSF1'] >= Q1 - 1.5*IQR) & (housing['BsmtFinSF1'] <= Q3 + 1.5*IQR)]

>Q1 = housing['TotalBsmtSF'].quantile(0.05)
>Q3 = housing['TotalBsmtSF'].quantile(0.95)
>IQR = Q3 - Q1
>housing = housing[(housing['TotalBsmtSF'] >= Q1 - 1.5*IQR) & (housing['TotalBsmtSF'] <= Q3 + 1.5*IQR)]

>Q1 = housing['BsmtUnfSF'].quantile(0.05)
>Q3 = housing['BsmtUnfSF'].quantile(0.95)
>IQR = Q3 - Q1
>housing = housing[(housing['BsmtUnfSF'] >= Q1 - 1.5*IQR) & (housing['BsmtUnfSF'] <= Q3 + 1.5*IQR)]

>Q1 = housing['1stFlrSF'].quantile(0.05)
>Q3 = housing['1stFlrSF'].quantile(0.95)
>IQR = Q3 - Q1
>housing = housing[(housing['1stFlrSF'] >= Q1 - 1.5*IQR) & (housing['1stFlrSF'] <= Q3 + 1.5*IQR)]

>Q1 = housing['GrLivArea'].quantile(0.05)
>Q3 = housing['GrLivArea'].quantile(0.95)
>IQR = Q3 - Q1
>housing = housing[(housing['GrLivArea'] >= Q1 - 1.5*IQR) & (housing['GrLivArea'] <= Q3 + 1.5*IQR)]

>Q1 = housing['GarageArea'].quantile(0.05)
>Q3 = housing['GarageArea'].quantile(0.95)
>IQR = Q3 - Q1
```

>housing = housing[(housing['GarageArea'] >= Q1 - 1.5*IQR) & (housing['GarageArea'] <= Q3 + 1.5*IQR)]

>Q1 = housing['TotalPorchArea'].quantile(0.05)

>Q3 = housing['TotalPorchArea'].quantile(0.95)

>IQR = Q3 - Q1

>housing = housing[(housing['TotalPorchArea'] >= Q1 - 1.5*IQR) & (housing['TotalPorchArea'] <= Q3 + 1.5*IQR)]

>housing[numeric_features.columns].describe(percentiles=[.25,.5,.75,.90,.95,.99])

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearRemodAdd	ExterQual	ExterCond	BsmtQual	BsmtCond
count	1443.000000	1443.000000	1443.000000	1443.000000	1443.000000	1443.000000	1443.000000	1443.000000	1443.000000	1443.000000	1443.000000
mean	731.543313	57.013167	68.037422	9770.485793	6.090783	5.573805	1984.835759	3.392931	3.083853	3.483714	2.933472
std	421.870317	42.313692	21.334077	4236.330058	1.376720	1.113182	20.655017	0.571034	0.352181	0.877507	0.554651
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1950.000000	2.000000	1.000000	0.000000	0.000000
25%	366.500000	20.000000	60.000000	7500.000000	5.000000	5.000000	1967.000000	3.000000	3.000000	3.000000	3.000000
50%	733.000000	50.000000	63.000000	9416.000000	6.000000	5.000000	1994.000000	3.000000	3.000000	4.000000	3.000000
75%	1096.500000	70.000000	79.000000	11475.000000	7.000000	6.000000	2004.000000	4.000000	3.000000	4.000000	3.000000
90%	1313.800000	120.000000	92.000000	14115.000000	8.000000	7.000000	2006.000000	4.000000	4.000000	4.000000	3.000000
95%	1386.900000	160.000000	103.000000	16539.700000	8.000000	8.000000	2007.000000	4.000000	4.000000	5.000000	3.000000
99%	1445.580000	190.000000	134.000000	25205.780000	10.000000	9.000000	2009.000000	5.000000	4.000000	5.000000	4.000000
max	1460.000000	190.000000	313.000000	36500.000000	10.000000	9.000000	2010.000000	5.000000	5.000000	5.000000	4.000000

>housing.shape

Step 6: Get Dummy Variables

>categoricals = housing.select_dtypes(exclude=[np.number])

>categoricals.columns

```
Index(['MSZoning', 'LotShape', 'LotConfig', 'Neighborhood', 'Condition1',
       'BldgType', 'HouseStyle', 'RoofStyle', 'Exterior1st', 'Exterior2nd',
       'Foundation', 'Functional', 'GarageType', 'SaleType', 'SaleCondition'],
      dtype='object')
```

>mszoning = pd.get_dummies(housing['MSZoning'], prefix='MSZoning', drop_first=True)

Adding the results to the master dataframe

>housing = pd.concat([housing, mszoning], axis=1)

>lotshape = pd.get_dummies(housing['LotShape'], prefix='LotShape', drop_first=True)

Adding the results to the master dataframe

```
>housing = pd.concat([housing, lotshape], axis=1)

>lotconfig = pd.get_dummies(housing['LotConfig'], prefix='LotConfig', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, lotconfig], axis=1)

>neighbrhd = pd.get_dummies(housing['Neighborhood'], prefix='Neighborhood', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, neighbrhd], axis=1)

>condition = pd.get_dummies(housing['Condition1'], prefix='Condition1', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, condition], axis=1)

>bldgtype = pd.get_dummies(housing['BldgType'], prefix='BldgType', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, bldgtype], axis=1)

>housestype = pd.get_dummies(housing['HouseStyle'], prefix='HouseStyle', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, housestype], axis=1)

>roofstyle = pd.get_dummies(housing['RoofStyle'], prefix='RoofStyle', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, roofstyle], axis=1)

>ext1style = pd.get_dummies(housing['Exterior1st'], prefix='Exterior1st', drop_first=True)
# Adding the results to the master dataframe
```

```
>housing = pd.concat([housing, ext1style], axis=1)

>ext2style = pd.get_dummies(housing['Exterior2nd'], prefix='Exterior2nd', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, ext2style], axis=1)

>func = pd.get_dummies(housing['Functional'], prefix='Functional', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, func], axis=1)

>grgtype = pd.get_dummies(housing['GarageType'], prefix='GarageType', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, grgtype], axis=1)

>found = pd.get_dummies(housing['Foundation'], prefix='Foundation', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, found], axis=1)


>sltype = pd.get_dummies(housing['SaleType'], prefix='SaleType', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, sltype], axis=1)

>slcond = pd.get_dummies(housing['SaleCondition'], prefix='SaleCondition', drop_first=True)
# Adding the results to the master dataframe
>housing = pd.concat([housing, slcond], axis=1)

### Drop the Originial Columns
```

```
>housing.drop(['MSZoning', 'LotShape', 'LotConfig', 'Neighborhood', 'Condition1',
    > 'BldgType', 'HouseStyle', 'RoofStyle', 'Exterior1st', 'Exterior2nd',
    > 'Foundation', 'Functional', 'GarageType', 'SaleType', 'SaleCondition'], axis=1, inplace=True)
>housing.shape
>housing.head()
```

Output :

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearRemodAdd	ExterQual	ExterCond	BsmtQual	BsmtCond	BsmtExposure	BsmtFinTyp
0	1	60	65.0	8450	7	5	2003	4	3	4	3	1	
1	2	20	80.0	9600	6	8	1976	3	3	4	3	4	
2	3	60	68.0	11250	7	5	2002	4	3	4	3	2	
3	4	70	60.0	9550	7	5	1970	3	3	3	4	1	
4	5	60	84.0	14260	8	5	2000	4	3	4	3	3	

5 rows × 158 columns

```
>y = housing['SalePrice']
>X = housing.drop(['SalePrice', 'Id'], axis=1)
>housing.head()
```

Output :

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearRemodAdd	ExterQual	ExterCond	BsmtQual	BsmtCond	BsmtExposure	BsmtFinTyp
0	1	60	65.0	8450	7	5	2003	4	3	4	3	1	
1	2	20	80.0	9600	6	8	1976	3	3	4	3	4	
2	3	60	68.0	11250	7	5	2002	4	3	4	3	2	
3	4	70	60.0	9550	7	5	1970	3	3	3	4	1	
4	5	60	84.0	14260	8	5	2000	4	3	4	3	3	

5 rows × 158 columns

 # split into train and test

```
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split


# storing column names in cols, since column names are (annoyingly) lost after
# scaling (the df is converted to a numpy array)
cols = X.columns
X = pd.DataFrame(scale(X))
```

```
X.columns = cols

X.columns



X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    train_size=0.7,
                                    test_size = 0.3, random_state=100)
```

Step 7: Use of lasso model for feature selection

```
# list of alphas to tune
>params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}


>lasso = Lasso()


# cross validation
>model_cv = GridSearchCV(estimator = lasso,
                >param_grid = params,
                >scoring= 'neg_mean_absolute_error',
                >cv = folds,
                >return_train_score=True,
               >verbose = 1)


>model_cv.fit(X_train, y_train)
>cv_results = pd.DataFrame(model_cv.cv_results_)
>cv_results.head()
```

Output:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score
0	0.116009	0.015464	0.000903	0.000492	0.0001	{'alpha': 0.0001}	-17626.939807	-18394.947060	-20848.781010	-18401.551660
1	0.106923	0.007771	0.001582	0.000197	0.001	{'alpha': 0.001}	-17626.924835	-18394.935408	-20848.763349	-18401.533270
2	0.106674	0.005007	0.001003	0.000634	0.01	{'alpha': 0.01}	-17626.775215	-18394.818891	-20848.586733	-18401.349371
3	0.109758	0.012592	0.000702	0.000602	0.05	{'alpha': 0.05}	-17626.112086	-18394.301053	-20847.801754	-18400.525306
4	0.105817	0.006856	0.003127	0.006253	0.1	{'alpha': 0.1}	-17625.288397	-18393.653822	-20846.819401	-18399.489845

 # plotting mean test and train scoes with alpha

cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

plotting

>plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])

>plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])

>plt.xlabel('alpha')

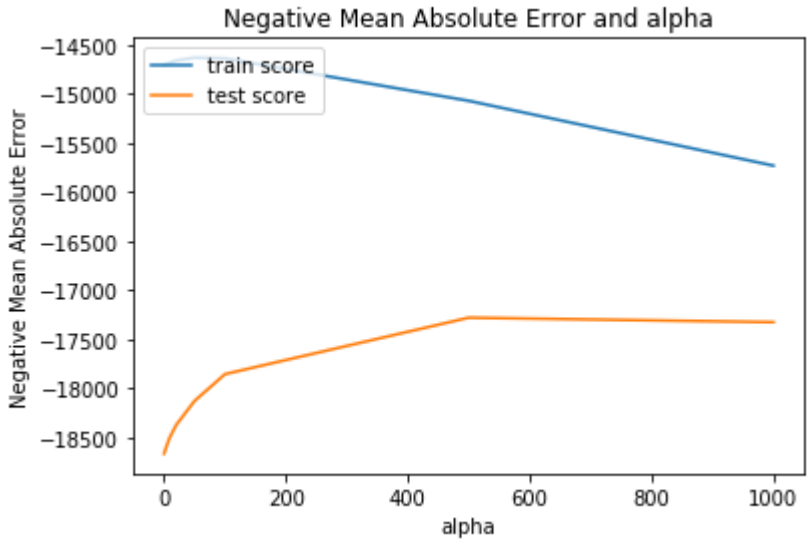
>plt.ylabel('Negative Mean Absolute Error')

>plt.title("Negative Mean Absolute Error and alpha")

>plt.legend(['train score', 'test score'], loc='upper left')

>plt.show()

Output:



Step8: Model evaluation

>alpha = 525

>lasso = Lasso(alpha=alpha)

>lasso.fit(X_train, y_train)

predict

>y_train_pred = lasso.predict(X_train)

>print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))

>y_test_pred = lasso.predict(X_test)

>print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred))

Output:

```
[('constant', 180482.404),
 ('MSSubClass', -3863.4),
 ('LotFrontage', 1569.351),
 ('LotArea', 6375.355),
 ('OverallQual', 12164.502),
 ('OverallCond', 5331.486),
 ('YearRemodAdd', 294.484),
 ('ExterQual', 3080.931),
 ('ExterCond', -4.53),
 ('BsmtQual', 3161.709),
 ('BsmtCond', -2441.001),
 ('BsmtExposure', 4044.004),
 ('BsmtFinType1', 0.0),
 ('BsmtFinSF1', 10013.186),
 ('BsmtUnfSF', -0.0),
 ('TotalBsmtSF', 6520.167),
 ('HeatingQC', 641.842),
 ('Electrical', -707.507),
 ('1stFlrSF', 0.0),
```

Step 9: Summary and next Steps

> 1. We can observe that the coefficients of insignificant features are made 0 by the lasso model.

> 2. The optimal value of lambda for lasso regression is around 500

> 3. Use the ridge regression and check it’s efficiency.

End of Exercise.

Exercise 5. Air Quality Prediction

Estimated time:

30:00 minutes

What this exercise is about:

This notebook demonstrates how to predict the relative humidity of the air (air quality) using the values of other attributes such as CO, NO2 etc.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean and prepare the data set.
- Build different models to predict the relative humidity and calculate RMSE.
- How to choose model based on the RMSE.

Introduction:

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. If the relative humidity is high, human beings sweat more. If the relative humidity is low, we feel much cooler than the actual temperature because our sweat evaporates easily. Let's predict the Relative Humidity at a given point of time, given the the values of attributes that affect the relative humidity.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
> import pandas as pd, numpy as np

> from sklearn.preprocessing import StandardScaler

> from sklearn.linear_model import LinearRegression

> from sklearn.svm import SVR

> from sklearn.ensemble import RandomForestRegressor

> from sklearn.metrics import mean_squared_error,mean_absolute_error
```

```
> from sklearn.model_selection import train_test_split

> import warnings

> warnings.filterwarnings('ignore')
```

Step 2: Download dataset from UCI website i.e. https://archive.ics.uci.edu/ml/machine-learning-databases/ozone/ and load to pandas data frame.

```
> air  = pd.read_excel("AirQualityUCI.xlsx")

> air.head()
```

Step 3: Data preparation and cleaning.

```
# Let's check the dimensions of the dataframe

>air.shape

# The NA values are represented by -200. Replace -200 with NAN.

>air = air.replace(-200, np.nan)

# Let's check the missing values in the dataframe

>round(100*(air.isnull().sum()/len(air.index)), 2)
```

Output:

```
Date             0.00
Time             0.00
CO(GT)          17.99
PT08.S1(CO)      3.91
NMHC(GT)        90.23
C6H6(GT)         3.91
PT08.S2(NMHC)    3.91
NOx(GT)         17.52
PT08.S3(NOx)     3.91
NO2(GT)         17.55
PT08.S4(NO2)     3.91
PT08.S5(O3)      3.91
T                3.91
RH               3.91
AH               3.91
dtype: float64
```

```
## we will drop the columns having more than 40% NA values.

>air = air.drop(air.loc[:,list(round(100*(air.isnull().sum()/len(air.index)), 2)>40)].columns, 1)
```

Let's drop all the rows having NAN values

>air = air.dropna()

>air.shape

>air.info()

Output:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6941 entries, 0 to 9356
Data columns (total 14 columns):
Date             6941 non-null datetime64[ns]
Time             6941 non-null object
CO(GT)           6941 non-null float64
PT08.S1(CO)      6941 non-null float64
C6H6(GT)         6941 non-null float64
PT08.S2(NMHC)    6941 non-null float64
NOx(GT)          6941 non-null float64
PT08.S3(NOx)     6941 non-null float64
NO2(GT)          6941 non-null float64
PT08.S4(NO2)     6941 non-null float64
PT08.S5(O3)      6941 non-null float64
T                6941 non-null float64
RH               6941 non-null float64
AH               6941 non-null float64
dtypes: datetime64[ns](1), float64(12), object(1)
memory usage: 813.4+ KB
```

Let's drop Date and Time

>air = air.drop(['Date', 'Time'], axis=1)

Split the train dataset into X and y

>y = air.pop('RH')

>X = air

Step 4: Rescaling and Split the data set into test and train.

Create a scaling object

>scaler = StandardScaler()

Scale these variables using 'fit_transform'

>Xstd = scaler.fit_transform(X)

split the data into train and test with test size and 30% and train size as 70%

```
>X_train, X_test, y_train, y_test=train_test_split(Xstd,y,test_size=0.3, random_state=100)

>print('Training data size:',X_train.shape)

>print('Test data size:',X_test.shape)

Training data size: (4858, 11)
Test data size: (2083, 11)
```

Step 5: Model Building

```
# Predict using LinearRegression

>lr = LinearRegression()

>lrm = lr.fit(X_train,y_train)

>y_pred_lr = lrm.predict(X_test)


>print('RMSE of Linear regression model: ', np.sqrt(mean_squared_error(y_test,y_pred_lr)))
```

Output:

```
RMSE of Linear regression model:  5.916300662074108
```

```
# Predict using Random Forest Regressor

>rf = RandomForestRegressor()

>rfm = rf.fit(X_train,y_train)

>y_pred_rf = rfm.predict(X_test)


#Calculate RMSE

>print('RMSE of Random Forest model: ',np.sqrt(mean_squared_error(y_test,y_pred_rf)))
```

Output :

```
RMSE of Random Forest model:  0.8845364654705632
```

```
# predict using SV Regressor

>svr = SVR()
```

>svm = svr.fit(X_train,y_train)

>y_pred_sv = svm.predict(X_test)

#Calculate RMSE

>print('RMSE of SVM model: ',np.sqrt(mean_squared_error(y_test,y_pred_sv)))

Output :

```
RMSE of SVM model:  3.4898148809077014
```

Step 6: Summary

> 1. RMSE is a way of measure of efficiency of the predictive models. Lower the RMSE better the model.

> 2. From our models above, it is evident that Random forest model is better in predicting the Relative Humidity.

> 3.Next step is, build other models such as lasso, ridge etc and find the RMSE of the models.

End of Exercise.

Exercise 6. Bank Marketing Campaign

Estimated time:

30:00 minutes

What this exercise is about:

This notebook helps to understand the features that play significant role in making successful marketing campaign for banking service such as term deposit.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean and prepare the data set and perform exploratory data analysis.
- Build model to select the significant features for making the successful bank marketing campaign.
- How to predict if a bank marketing campaign can be successful.

Introduction:

The data is related with direct marketing campaigns of a banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. Let's use find the features that determine the success of the campaign.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

> import pandas as pd, numpy as np

> import xgboost

> from sklearn.metrics import mean_squared_error,mean_absolute_error

> from sklearn.model_selection import train_test_split

> import warnings

> warnings.filterwarnings('ignore')

> from sklearn.metrics import accuracy_score

Step 2: Download dataset from

https://www.kaggle.com/janiobachmann/bank-marketing-dataset/download/IITghj3S8idnPugoY29Y%2Fversions%2FKcTyTA1U0i4v06S1TcwY%2Ffiles%2Fbank.csv?datasetVersionNumber=1 and load to pandas data frame.

> bank = pd.read_csv("bank.xlsx")

> bank.head()

Step 3: Data preparation and cleaning.

Let's check the dimensions of the dataframe

>bank.shape

Let's check the missing values in the dataframe

>round(100*(bank.isnull().sum()/len(bank.index)), 2)

Output:

```
age          0.0
job          0.0
marital      0.0
education    0.0
default      0.0
balance      0.0
housing      0.0
loan         0.0
contact      0.0
day          0.0
month        0.0
duration     0.0
campaign     0.0
pdays        0.0
previous     0.0
poutcome     0.0
deposit      0.0
dtype: float64
```

We can see that there are no missing values.

>bank['deposit'].value_counts()

Output:

```
no     5873
yes    5289
Name: deposit, dtype: int64
```

```
# Get numeric values of boolean variables.
>bank['default'] = bank.default.map({'yes': 1 , 'no': 0})
>bank['housing'] = bank.housing.map({'yes': 1 , 'no': 0})
>bank['loan'] = bank.loan.map({'yes': 1 , 'no': 0})
>bank['deposit'] = bank.deposit.map({'yes': 1 , 'no': 0})

# Get Dummy variables of categorial variables
>for col in  ['job', 'marital', 'education', 'contact', 'month', 'poutcome']:
    >bank = pd.concat([bank.drop(col, axis=1),
              >pd.get_dummies(bank[col], prefix=col, prefix_sep='_',
              >drop_first=True, dummy_na=False)], axis=1)
# Get Dummy variables of categorial variables
>for col in  ['job', 'marital', 'education', 'contact', 'month', 'poutcome']:
   > bank = pd.concat([bank.drop(col, axis=1),
             > pd.get_dummies(bank[col], prefix=col, prefix_sep='_',
              >drop_first=True, dummy_na=False)], axis=1)
bank.head()
```

	age	default	balance	housing	loan	day	duration	campaign	previous	deposit	...	month_jul	month_jun	month_mar	month_may	month_nov	
0	59	0	2343	1	0	5	1042	1	0	1	...	0	0	0	1	0	
1	56	0	45	0	0	5	1467	1	0	1	...	0	0	0	1	0	
2	41	0	1270	1	0	5	1389	1	0	1	...	0	0	0	1	0	
3	55	0	2476	1	0	5	579	1	0	1	...	0	0	0	1	0	
4	54	0	184	0	0	5	673	2	0	1	...	0	0	0	1	0	

5 rows × 42 columns

month_oct	month_sep	poutcome_other	poutcome_success	poutcome_unknown
0	0	0	0	1
0	0	0	0	1
0	0	0	0	1
0	0	0	0	1
0	0	0	0	1

```
>bank.drop('contact_unknown', axis=1, inplace=True)
```

```
# Putting feature variable to X
```

```
>X = bank.drop('deposit',axis=1)
```

```
# Putting response variable to y
```

```
>y = bank['deposit']
```

bank.drop('pdays', axis=1, inplace=True)

Step 4: Rescaling and Split the data set into test and train.

```
# Splitting the data into train and test
```

```
>X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=100)
```

```
# train the XGBoost model
```

```
>xgb = xgboost.XGBClassifier(n_estimators=100, learning_rate=0.08, gamma=0, subsample=0.75,
                        colsample_bytree=1, max_depth=7)
```

```
>xgb.fit(X_train,y_train.squeeze().values)
```

```
# calculate and print scores for the model for top 10 features
```

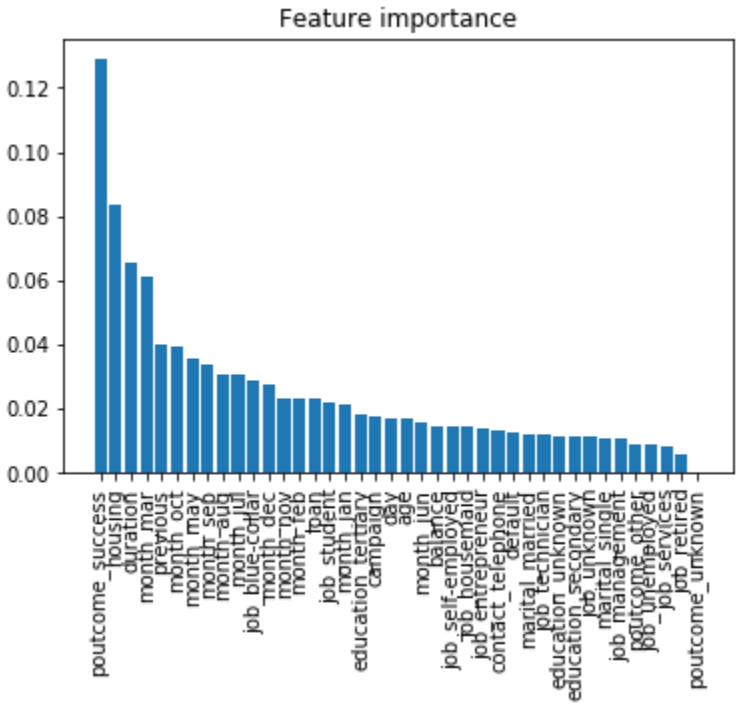
```
>y_train_preds = xgb.predict(X_train)
```

```
>y_test_preds = xgb.predict(X_test)
```

```
>print('XGB accuracy score for train: %.3f: test: %.3f' % (
      >accuracy_score(y_train, y_train_preds),
      >accuracy_score(y_test, y_test_preds)))
```

```
XGB accuracy score for train: 0.906: test: 0.847
```

```
# Get the significant features that can make the campaign successful

>headers = ["name", "score"]


>values = sorted(zip(X_train.columns, xgb.feature_importances_), key=lambda x: x[1] * -1)


>xgb_feature_importances = pd.DataFrame(values, columns = headers)

# Plot the feature importance

>x_pos = np.arange(0, len(xgb_feature_importances))

>plt.bar(x_pos, xgb_feature_importances['score'])

>plt.xticks(x_pos, xgb_feature_importances['name'])

>plt.xticks(rotation=90)

>plt.title('Feature importance')

>plt.show()
```



Step 6: Summary

> 1. The xgboost accuracy of predicting if campaign can be successful or not is .

> 2. The features that are significant in predicting if the campaign can be successful are not are

Previous_campaign, housing loan, duration of campaign and month_march.

> 3. Next step is, build other classifier models such as KNN, RandomForest, nearual network, naïve bayes.

End of Exercise.

Exercise 7. Liver Disease Prediction

Estimated time:

30:00 minutes

What this exercise is about:

This notebook demonstrates how to predict if a person has liver disease or not based on certain clinical and demographic factors

What you should be able to do:

At the end of this exercise, you will lean

- How to clean and prepare the data set.
- How to find the optimal value of neighbors for KNN model.
- Build a KNN model to predict the liver disease.

Introduction:

This data set contains 416 liver patient records and 167 non liver patient records. Patients with liver disease have been continuously increasing because of excessive consumption of alcohol, inhale of harmful gases.The objective is to build a model to predict if a person has liver disease or not based on certain clinical facts.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
> import pandas as pd, numpy as np

> from sklearn.preprocessing import StandardScaler

> from sklearn.linear_model import LinearRegression

> from sklearn.neighbors import KNeighborsClassifier

> from sklearn import model_selection

> from sklearn.model_selection import train_test_split

> import warnings
```

> warnings.filterwarnings('ignore')

> import matplotlib.pyplot as plt

Step 2: Download dataset from https://www.kaggle.com/uciml/indian-liver-patient-records/download/LyanOKbu7iChVyoOVpgo%2Fversions%2FfUsPCKqE8IhThYb2Dl4F%2Ffiles%2Findian_liver_patient.csv?datasetVersionNumber=1 and load to pandas data frame.

> liver = pd.read_csv("indian_liver_patient.csv")

> liver.head()

Step 3: Data preparation and cleaning.

Let's check the dimensions of the dataframe

>liver.shape

Let's check the missing values in the dataframe

>round(100*(liver.isnull().sum()/len(liver.index)), 2)

```
Age                         0.00
Gender                      0.00
Total_Bilirubin             0.00
Direct_Bilirubin            0.00
Alkaline_Phosphotase        0.00
Alamine_Aminotransferase    0.00
Aspartate_Aminotransferase  0.00
Total_Protiens              0.00
Albumin                     0.00
Albumin_and_Globulin_Ratio  0.69
Dataset                     0.00
dtype: float64
```

We can see that only Albumin_and_Globulin_Ratio has some missing values. Let's drop the NAN values

>liver.dropna(inplace=True)

>liver.shape

>liver['Gender'] = liver.Gender.map({'Female': 2, 'Male': 1})

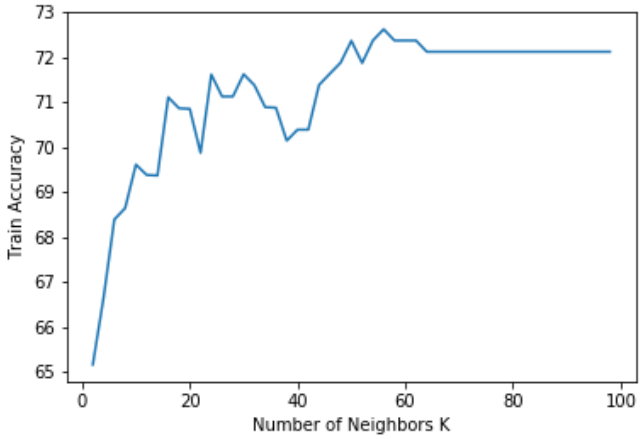
Putting feature variable to X

>X = liver.drop('Dataset',axis=1)

```
# Putting response variable to y
>y = liver['Dataset']
```

Step 4: Rescaling and Split the data set into test and train.

```
>X_std = StandardScaler().fit_transform(X)
# Splitting the data into train and test
>X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.30, random_state=100)
>neighbors = [x for x in list(range(1,100)) if x % 2 == 0]
>cv_scores = []
# Perform 10-fold cross validation on training set for odd values of k:
>seed=123
>for k in neighbors:
  > k_value = k+1
  > knn = KNeighborsClassifier(n_neighbors = k_value, weights='uniform', p=2, metric='euclidean')
   >kfold = model_selection.KFold(n_splits=10, random_state=seed)
   >scores = model_selection.cross_val_score(knn, X_train, y_train, cv=kfold, scoring='accuracy')
   >cv_scores.append(scores.mean()*100)
>optimal_k = neighbors[cv_scores.index(max(cv_scores))]
>print(optimal_k)
#print(( "The optimum number of neighbors is %d with %0.1f%%" % (optimal_k, cv_scores[optimal_k])))
>plt.plot(neighbors, cv_scores)
>plt.xlabel('Number of Neighbors K')
>plt.ylabel('Train Accuracy')
>plt.show()
```



Step 5: Model Building

>knn = KNeighborsClassifier(n_neighbors = 56)

>knn.fit(X_train, y_train)

Output:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=56, p=2,
           weights='uniform')
```

>Y_pred = knn.predict(X_test)

>acc_train = round(knn.score(X_train, y_train) * 100, 2)

>acc_val = round(knn.score(X_test, y_test) * 100, 2)

>print("Accuracy of training dataset: " + str(acc_train))

>print("Accuracy of test dataset: "+ str(acc_val))

Output:

```
Accuracy of training dataset: 73.33
Accuracy of test dataset: 70.69
```

Step 6: Summary

> 1. At neighbors 56, the accuracy of prediction of liver disease from KNN is 70.69

> 2. Apply different algorithms and check if the accuracy improves.

End of Exercise.

Exercise 8. Heart Disease Prediction

Estimated time:

00:30 minutes

What this exercise is about:

This notebook tries to predict if a person is at risk of heart disease or not based on his/her age, sex and medical history.

What you should be able to do:

At the end of this exercise, you will lean

- How to normalize, split data before building model.
- How to build model using K-nearest Neighbors algorithm.
- Performing model selection using KFold to find optimal number of neighbors.

Introduction:

The dataset has medical history, age, sex of persons at risk of having heart disease and persons not having heart disease. The objective use to use this data and build a model to predict if a person is at risk of having heart disease or not.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

>import pandas as pd

>import matplotlib.pyplot as plt

>from sklearn.neighbors import KNeighborsClassifier

>from sklearn import model_selection

>from sklearn.model_selection import train_test_split

> from sklearn.preprocessing import StandardScaler

Step 2: Download dataset from https://www.kaggle.com/ronitf/heart-disease-uci/download

and load to pandas data frame

>df = pd.read_csv(heart.csv)

>df.head()

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

>df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
target      303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.2 KB
```

The data set is clean. You can go ahead with model building.

Step 3 : Normalize the dataset and split into train and test data

># Putting feature variable to X

>X = heart.drop('target',axis=1)

># Putting response variable to y

>y = heart['target']

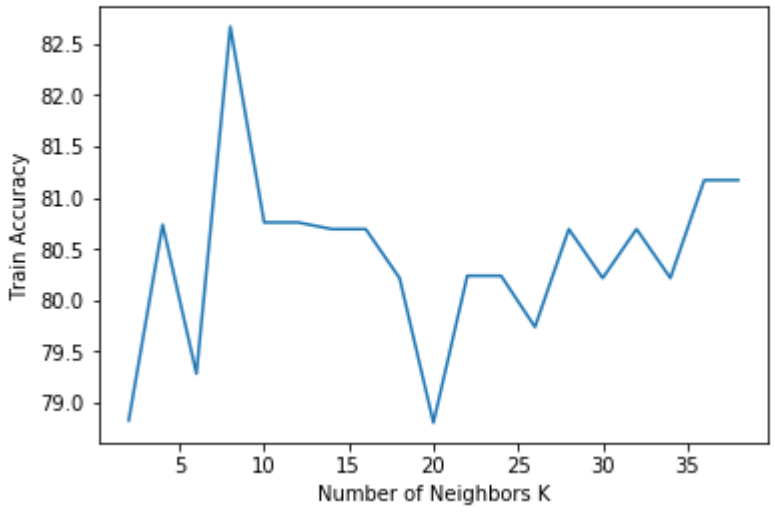
>X_std = StandardScaler().fit_transform(X)

```
>#  Splitting the data into train and test
>X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.30, random_state=101)
```

Step 4: Model Building.

```
>#  Use KNN Algorithm
>#  To find optimum value of K we build a loop witch check all the posible values for K.
>#  Neighbors
>neighbors = [x for x in list(range(1,40)) if x % 2 == 0]
>#  Create empty list that will hold cv scores
>cv_scores = []
>#  Perform 10-fold cross validation on training set for odd values of k:
>seed=123
>for k in neighbors:
>k_value = k+1
>knn = KNeighborsClassifier(n_neighbors = k_value, weights='uniform', p=2, metric='euclidean')
>kfold = model_selection.KFold(n_splits=10, random_state=seed)
>scores = model_selection.cross_val_score(knn, X_train, y_train, cv=kfold, scoring='accuracy')
>cv_scores.append(scores.mean()*100)
>optimal_k = neighbors[cv_scores.index(max(cv_scores))]
>print(( "The optimum number of neighbors is %d with %0.1f%%" % (optimal_k, cv_scores[optimal_k])))


>plt.plot(neighbors, cv_scores)
>plt.xlabel('Number of Neighbors K')
>plt.ylabel('Train Accuracy')
>plt.show()
```

```
The optimum number of neighbors is 8 with 80.2%
```



Step 5: Prediction of Heart disease and find the accuracy of prediction.

```
>from sklearn.neighbors import KNeighborsClassifier
>knn = KNeighborsClassifier(n_neighbors = 8)
>knn.fit(X_train, y_train)
>Y_pred = knn.predict(X_test)
>acc_train = round(knn.score(X_train, y_train) * 100, 2)
>acc_val = round(knn.score(X_test, y_test) * 100, 2)
>print("Accuracy of training dataset: " + str(acc_train))
>print("Accuracy of test dataset: "+ str(acc_val))
```

```
Accuracy of training dataset: 87.26
Accuracy of test dataset: 83.52
```

Step 6: Summary and next Steps

> 1. The KNN model was able to predict the heart disease with a accuracy of 83.52 %.

> 2. You have learnt how to build KNN model and to perform model selection using KFold.

> 3. Next step is to try models such as Random Forest and see if accuracy improves.

End of Exercise.

Exercise 9. Credit Default Prediction

Estimated time:

00:30 minutes

What this exercise is about:

This notebook demonstrates how to build a random forest model to predict whether a given customer defaults or not

What you should be able to do:

At the end of this exercise, you will lean

- How to prepare data to build a random forest model.
- Build and train random forest model to predict credit default.
- Next steps to boost the performance of random forest model.

Introduction:

Credit default is one of the most important problems in the banking industry. There are various features which can be used to predict default, such as income, employment status, age, previous loans, payments, number of times a credit payment has been delayed by the customer etc.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

>import pandas as pd

>import numpy as np

>from sklearn.model_selection import train_test_split

> from sklearn.ensemble import RandomForestClassifier

Step 2: Download dataset from https://www.kaggle.com/uciml/default-of-credit-card-clients-datasetand load to pandas data frame

>df = pd.read_csv('credit-card-default.csv')

>df.head()

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0	0.0	0.0	689.0	
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0	3455.0	3261.0	0.0	1000.0	
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	1500.0	
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	2019.0	
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0	19146.0	19131.0	2000.0	36681.0	1

5 rows × 25 columns

> df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
ID                          30000 non-null int64
LIMIT_BAL                   30000 non-null float64
SEX                         30000 non-null int64
EDUCATION                   30000 non-null int64
MARRIAGE                    30000 non-null int64
AGE                         30000 non-null int64
PAY_0                       30000 non-null int64
PAY_2                       30000 non-null int64
PAY_3                       30000 non-null int64
PAY_4                       30000 non-null int64
PAY_5                       30000 non-null int64
PAY_6                       30000 non-null int64
BILL_AMT1                   30000 non-null float64
BILL_AMT2                   30000 non-null float64
BILL_AMT3                   30000 non-null float64
BILL_AMT4                   30000 non-null float64
```

It is evident from above information that there is no major data quality issues.

Step 3: Data Preparation by splitting the data into train and test.

>\# Putting feature variable to X

>X = df.drop('default.payment.next.month',axis=1)

>\# Putting response variable to y

>y = df['default.payment.next.month']

>\# Splitting the data into train and test

>X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)

>print('Train data set dimension: {}'.format(X_train.shape))

>print('Test data set dimension: {}'.format(X_test.shape))

>print('Train label dimension: {}'.format(y_train.shape))

>print('Test label dimension: {}'.format(y_test.shape))

```
Train data set dimension: (21000, 24)
Test data set dimension: (9000, 24)
Train label dimension: (21000,)
Test label dimension: (9000,)
```

Step 4: Model Building.

>\# Running the random forest with default parameters.

>rfc = RandomForestClassifier()

>rfc.fit(X_train,y_train)

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm start=False)
```

Step 5: Predict credit default using the model

>predictions = rfc.predict(X_test)

Step 6: Model Evaluation

>\# Importing classification report and confusion matrix from sklearn metrics

>from sklearn.metrics import classification_report,confusion_matrix, accuracy_score

>\# Let's check the report of our default model

>print(classification_report(y_test,predictions))

```
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      7058
           1       0.61      0.33      0.43      1942

   micro avg       0.81      0.81      0.81      9000
   macro avg       0.72      0.64      0.66      9000
weighted avg       0.79      0.81      0.79      9000
```

>\# Printing confusion matrix

>print(confusion_matrix(y_test,predictions))

```
[[6637  421]
 [1294  648]]
```

>print(accuracy_score(y_test,predictions))

```
0.8094444444444444
```

Step 7: Summary and next Steps

> 1. You have learnt how to build a Random Forest Model for prediction problems.

> 2. Next step is to boost performance of the model by tuning the hype rparameters such as n_estimators, max_depth, min_samples_split and min_samples_leaf

End of Exercise.

Exercise 10. Car Price Prediction

Estimated time:

60:00 minutes

What this exercise is about:

This notebook demonstrates how to build a logistic regression model to identify the features that affect the price of the car.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean the data set, create dummy variables to build the logistic regression model.
- How to build a logistic regression model to find the features that affect the car price.
- Next steps to improve the efficiency of the model.

Introduction:

One of the car companies wants to understand which factors actually affect the price of the car before entering a specific market. It has data about the specifications of different models of cars sold in that region. Using this data, it wants to predict which of these variables actually influence the price of the car.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
> import pandas as pd

> import numpy as np

> from sklearn.model_selection import train_test_split

> from sklearn.preprocessing import MinMaxScaler

> from sklearn.feature_selection import RFE

> from sklearn.linear_model import LinearRegression

> import matplotlib.pyplot as plt
```

> import seaborn as sns

> %matplotlib inline

> import statsmodels.api as sm

> from statsmodels.stats.outliers_influence import variance_inflation_factor

Step 2: Download dataset from https://www.kaggle.com/venkatabhaskar/car-price/download and load to pandas data frame

> carprice = pd.read_csv('CarPrice.csv')

Step 3: Data preparation and cleaning.

> carprice[['CarName', 'Model']] = pd.DataFrame(carprice.CarName.str.split(' ',1).tolist(), columns = ['CarName','Model'])

>carprice.CarName.replace({'maxda':'mazda','porcshce':'porsche','vokswagen':'volkswagen', 'vw':'volkswagen', 'Nissan':'nissan', 'toyouta':'toyota'}, inplace=True)

> # Before running linear regression, we need to convert everything to scalar. Hence map categorical variables to scalar.

> carprice['doornumber'] = carprice.doornumber.map({'four': 4, 'two': 2})

> carprice['cylindernumber'] = carprice.cylindernumber.map({'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'eight': 8, 'twelve': 12})

>carprice['fueltype'] = pd.get_dummies(carprice['fueltype'], drop_first=True)

>carprice = pd.concat([carprice, fueltype], axis = 1)

>carprice.drop(['fueltype'], axis = 1, inplace = True)

>aspiration = pd.get_dummies(carprice['aspiration'], drop_first=True)

>carprice = pd.concat([carprice, aspiration], axis = 1)

>carprice.drop(['aspiration'], axis = 1, inplace = True)

```
># Get the dummy variables for the feature 'enginelocation'
>enginelocation = pd.get_dummies(carprice['enginelocation'], drop_first=True)
># Add the results to the original housing dataframe
>carprice = pd.concat([carprice, enginelocation], axis = 1)
># Drop 'fueltype' as we have created the dummies for it
>carprice.drop(['enginelocation'], axis = 1, inplace = True)

># Get the dummy variables for the feature 'CarName'
>mancode = pd.get_dummies(carprice['CarName'])
># Add the results to the original housing dataframe
>carprice = pd.concat([carprice, mancode], axis = 1)
># Drop 'CarName' as we have created the dummies for it
>carprice.drop(['CarName'], axis = 1, inplace = True)

># Get the dummy variables for the feature 'carbody'
>carbody = pd.get_dummies(carprice['carbody'], drop_first=True)
># Add the results to the original housing dataframe
>carprice = pd.concat([carprice, carbody], axis = 1)
># Drop 'carbody' as we have created the dummies for it
>carprice.drop(['carbody'], axis = 1, inplace = True)

># Get the dummy variables for the feature 'enginetype'
>enginetype = pd.get_dummies(carprice['enginetype'], drop_first=True)
># Add the results to the original housing dataframe
>carprice = pd.concat([carprice, enginetype], axis = 1)
># Drop 'enginetype' as we have created the dummies for it
>carprice.drop(['enginetype'], axis = 1, inplace = True)
```

```
>carprice = pd.concat([carprice, fuelsystem], axis = 1)
>carprice.drop(['fuelsystem'], axis = 1, inplace = True)
>carprice = pd.concat([carprice, drivewheel], axis = 1)
>carprice.drop(['drivewheel'], axis = 1, inplace = True)
>carprice.drop(['Model'], axis = 1, inplace = True)
>carprice.head()
```

	car_ID	symboling	doornumber	wheelbase	carlength	carwidth	carheight	curbweight	cylindernumber	enginesize	...	rotor	2bbl	4bbl	idi	mfi	mpfi
0	1	3	2	88.6	168.8	64.1	48.8	2548	4	130	...	0	0	0	0	0	1
1	2	3	2	88.6	168.8	64.1	48.8	2548	4	130	...	0	0	0	0	0	1
2	3	1	2	94.5	171.2	65.5	52.4	2823	6	152	...	0	0	0	0	0	1
3	4	2	4	99.8	176.6	66.2	54.3	2337	4	109	...	0	0	0	0	0	1
4	5	2	4	99.4	176.6	66.4	54.3	2824	5	136	...	0	0	0	0	0	1

5 rows × 62 columns

Step 4: Split the data into train and test and apply standardization.

```
> df_train, df_test = train_test_split(carprice, train_size = 0.7, test_size = 0.3, random_state = 100)
> scaler = MinMaxScaler()
> # Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
> num_vars = ['doornumber', 'cylindernumber', 'wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight','enginesize', 'boreratio' , 'stroke', 'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg' , 'price' ]
> df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
> df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 143 entries, 122 to 8
Data columns (total 62 columns):
car_ID              143 non-null int64
symboling           143 non-null int64
doornumber          143 non-null float64
wheelbase           143 non-null float64
carlength           143 non-null float64
carwidth            143 non-null float64
carheight           143 non-null float64
curbweight          143 non-null float64
cylindernumber      143 non-null float64
enginesize          143 non-null float64
boreratio           143 non-null float64
stroke              143 non-null float64
compressionratio    143 non-null float64
horsepower          143 non-null float64
peakrpm             143 non-null float64
citympg             143 non-null float64
highwaympg          143 non-null float64
price               143 non-null float64
gas                 143 non-null uint8
turbo               143 non-null uint8
rear                143 non-null uint8
alfa-romero         143 non-null uint8
audi                143 non-null uint8
```

> y_train = df_train.pop('price')

> X_train = df_train

Step 5: Building the Model

> # Running RFE with the output number of the variable equal to 10

> lm = LinearRegression()

> lm.fit(X_train, y_train)

> rfe = RFE(lm, 10) # running RFE

> rfe = rfe.fit(X_train, y_train)> # fit_transform

> # The top 10 variables / columns that affects the price of the car as predicted by rfe

> col = X_train.columns[rfe.support_]

> col

```
Index(['carwidth', 'curbweight', 'cylindernumber', 'enginesize', 'boreratio',
       'stroke', 'rear', 'bmw', 'peugeot', 'dohcv'],
      dtype='object')
```

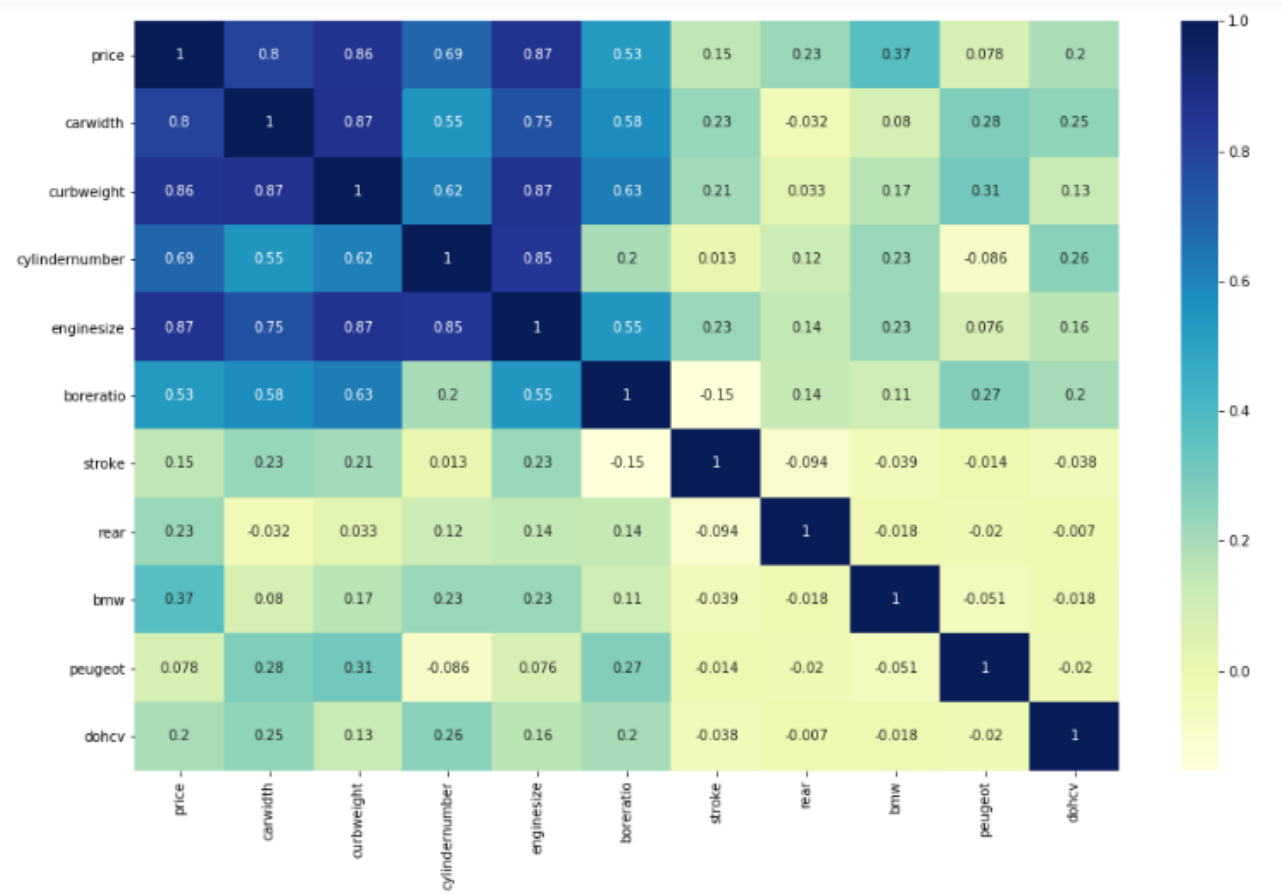
> # Let's check the correlation coefficients of top 10 features to see which variables are highly correlated

> df_corr = pd.concat([y_train, X_train[col]], axis = 1)

> plt.figure(figsize = (16, 10))

> sns.heatmap(df_corr.corr(), annot = True, cmap="YlGnBu")

> plt.show()



> # Creating X_test dataframe with RFE selected variables

> X_train_rfe = X_train[['carwidth', 'curbweight', 'enginesize', 'bmw', 'rear']]

> # Adding a constant variable

> X_train_new = sm.add_constant(X_train_rfe)

> lm = sm.OLS(y_train,X_train_new).fit() # Running the linear model

> #Let's see the summary of our linear model

> print(lm.summary())

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.886
Model:                            OLS   Adj. R-squared:                  0.882
Method:                 Least Squares   F-statistic:                     214.0
Date:                Mon, 30 Dec 2019   Prob (F-statistic):           6.87e-63
Time:                        06:59:26   Log-Likelihood:                172.52
No. Observations:                 143   AIC:                            -333.0
Df Residuals:                     137   BIC:                            -315.3
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.1813      0.017    -10.532      0.000      -0.215      -0.147
carwidth       0.3603      0.071      5.107      0.000       0.221       0.500
curbweight     0.2724      0.079      3.439      0.001       0.116       0.429
enginesize     0.4535      0.084      5.417      0.000       0.288       0.619
bmw            0.2485      0.032      7.720      0.000       0.185       0.312
rear           0.4773      0.077      6.187      0.000       0.325       0.630
==============================================================================
Omnibus:                        6.652   Durbin-Watson:                   2.029
Prob(Omnibus):                  0.036   Jarque-Bera (JB):                6.377
Skew:                           0.426   Prob(JB):                       0.0412
Kurtosis:                       3.587   Cond. No.                         21.3
==============================================================================

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

> # Calculate the VIFs for the new model

> vif = pd.DataFrame()

> X = X_train_new

> vif['Features'] = X.columns

> vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

> vif['VIF'] = round(vif['VIF'], 2)

> vif = vif.sort_values(by = "VIF", ascending = False)

> vif

	Features	VIF
1	curbweight	34.22
0	carwidth	19.95
2	enginesize	14.83
3	bmw	1.13
4	rear	1.07

> # Drop curbweight as it's vif is high.

> X_train_new.drop(['curbweight'], axis=1, inplace=True)

> # Adding a constant variable

> X_train_lm = sm.add_constant(X_train_new)

> lm = sm.OLS(y_train,X_train_lm).fit() # Running the linear model

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.877
Model:                            OLS   Adj. R-squared:                  0.873
Method:                 Least Squares   F-statistic:                     245.3
Date:                Mon, 30 Dec 2019   Prob (F-statistic):           1.18e-61
Time:                        06:59:38   Log-Likelihood:                 166.60
No. Observations:                 143   AIC:                            -323.2
Df Residuals:                     138   BIC:                            -308.4
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.1868      0.018    -10.496      0.000      -0.222      -0.152
carwidth       0.5195      0.055      9.400      0.000       0.410       0.629
enginesize     0.6316      0.068      9.251      0.000       0.497       0.767
bmw            0.2526      0.033      7.563      0.000       0.187       0.319
rear           0.4650      0.080      5.810      0.000       0.307       0.623
==============================================================================
Omnibus:                        9.053   Durbin-Watson:                   2.079
Prob(Omnibus):                  0.011   Jarque-Bera (JB):                8.915
Skew:                           0.576   Prob(JB):                       0.0116
Kurtosis:                       3.410   Cond. No.                         16.3
==============================================================================

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

> vif = pd.DataFrame()

> X = X_train_new

> vif['Features'] = X.columns

> vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

> vif['VIF'] = round(vif['VIF'], 2)

> vif = vif.sort_values(by = "VIF", ascending = False)

> vif

	Features	VIF
1	enginesize	8.84
0	carwidth	8.33
2	bmw	1.13
3	rear	1.07

Step 7: Summary and Next Steps

> 1. Engine Size, Car Width, Brand and Engine Location Rear are the 4 factors which influence the car Price.

> 2. Explore how to evaluate this model (calculate R2 score)

End of Exercise.

Exercise 11. Media Content Problem

Estimated time:

00:30 minutes

What this exercise is about:

This notebook tries to identify the reason for reduction in viewership of a show telecasted by a media company like prime videos, Voot, Netflix etc.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean/prepare data. Learn how to derive new features from existing.
- Build and train model to identify the reason for reduction in viewership.
- Next steps to boost the performance of model.

Introduction:

There is a decline in number of people watching a show telecasted by a media company. There could be various reasons likeongoing cricket match, twist in the story, disappearance of key character impacted the number of viewership.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

>import pandas as pd

>import numpy as np

>from sklearn.linear_model import LinearRegression

>import statsmodels.api as sm

Step 2: Download dataset from https://www.kaggle.com/ashydv/media-company/download

and load to pandas data frame

>df = pd.read_csv('mediacompany.csv)

>media = media.drop('Unnamed: 7',axis = 1)

>df.head()

	Date	Views_show	Visitors	Views_platform	Ad_impression	Cricket_match_india	Character_A
0	3/1/2017	183738	1260228	1706478	1060860448	0	0
1	3/2/2017	193763	1270561	1690727	1031846645	0	0
2	3/3/2017	210479	1248183	1726157	1010867575	0	0
3	3/4/2017	240061	1492913	1855353	1079194579	1	0
4	3/5/2017	446314	1594712	2041418	1357736987	0	0

Step 3: Data Preparation and derive new features from existing features.

># Converting date to Pandas datetime format

>media['Date'] = pd.to_datetime(media['Date'])

># Extract day of week from date

>media['Day_of_week'] = media['Date'].dt.dayofweek

>media.head()

	Date	Views_show	Visitors	Views_platform	Ad_impression	Cricket_match_india	Character_A	Day_of_week
0	2017-03-01	183738	1260228	1706478	1060860448	0	0	2
1	2017-03-02	193763	1270561	1690727	1031846645	0	0	3
2	2017-03-03	210479	1248183	1726157	1010867575	0	0	4
3	2017-03-04	240061	1492913	1855353	1079194579	1	0	5
4	2017-03-05	446314	1594712	2041418	1357736987	0	0	6

># create a Weekend feature, with value 1 for weekends and 0 for weekdays

>def isweekend(i):

> if i == 5 or i == 6: return 1

> else: return 0

>media['weekend']=[isweekend(i) for i in media['Day_of_week']]># Splitting the data into train and test

	Date	Views_show	Visitors	Views_platform	Ad_impression	Cricket_match_india	Character_A	Day_of_week	weekend
0	2017-03-01	183738	1260228	1706478	1060860448	0	0	2	0
1	2017-03-02	193763	1270561	1690727	1031846645	0	0	3	0
2	2017-03-03	210479	1248183	1726157	1010867575	0	0	4	0
3	2017-03-04	240061	1492913	1855353	1079194579	1	0	5	1
4	2017-03-05	446314	1594712	2041418	1357736987	0	0	6	1

Step 4 : Separate the target variable

>\# Putting feature variable to X

>X = media[['weekend', 'Character_A', 'Ad_impression']]

>\# Putting response variable to y

>y = media['Views_show']

Step 5: Model Building.

>\# Creating LinearRegression Object.

>lm = LinearRegression()

>\# fit the model to the training data

>lm.fit(X,y)

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Step 6: Predict the

>#Unlike SKLearn, statsmodels don't automatically fit a constant.

> so you need to use the method sm.add_constant(X) in order to add a constant.

> X = sm.add_constant(X)

> # create a fitted model in one line

> lm_1 = sm.OLS(y,X).fit()

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             Views_show   R-squared:                       0.803
Model:                            OLS   Adj. R-squared:                  0.795
Method:                 Least Squares   F-statistic:                     103.0
Date:                Sat, 14 Dec 2019   Prob (F-statistic):           1.05e-26
Time:                        22:28:46   Log-Likelihood:                -1004.2
No. Observations:                  80   AIC:                             2016.
Df Residuals:                      76   BIC:                             2026.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2.661e+05   4.74e+04     -5.609      0.000   -3.61e+05   -1.72e+05
weekend     1.51e+05   1.88e+04      8.019      0.000    1.14e+05    1.89e+05
Character_A -2.99e+04   2.14e+04     -1.394      0.167   -7.26e+04    1.28e+04
Ad_impression   0.0004   3.69e-05      9.875      0.000       0.000       0.000
==============================================================================
Omnibus:                        4.723   Durbin-Watson:                   1.169
Prob(Omnibus):                  0.094   Jarque-Bera (JB):                3.939
Skew:                           0.453   Prob(JB):                        0.139
Kurtosis:                       3.601   Cond. No.                     9.26e+09
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.26e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Step 7: Summary and next Steps

> 1. Weekend, Character_A, Ad_Impression are the key variables which affect the number of viewership.

> 2.You have learnt how create new feature from existing feature.

> 3. Next step is to try other variables and see if there is a change in R-squared.

End of Exercise.

Exercise 12. Online Retail Case Study

Estimated time:

60:00 minutes

What this exercise is about:

This notebook demonstrates how to classify customers of online retail company based on customers usage pattern, so that the online retail company can concentrate on most valued customers.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean the data set, remove outliers and create new feature variables.
- How to build a K-Means model for the classification problems.
- How to use find the optimal value of the cluster using silhouette analysis.

Introduction:

The online retail company from Europe wants to understand its customers to improve the efficiency and thereby revenue. By understanding the usage pattern of its customers, it can concentrate more on the valued customers which helps in customer retention. Using the unsupervised learning method, let's identify the usage pattern of its customers.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
> import pandas as pd

> import numpy as np

> from sklearn.preprocessing import StandardScaler

> from sklearn.cluster import KMeans

> from sklearn.metrics import silhouette_score

> import datetime as dt
```

> import matplotlib.pyplot as plt

Step 2: Download dataset from https://archive.ics.uci.edu/ml/datasets/online+retail and load to pandas data frame

> df = pd.read_csv("Online+Retail.csv", sep=",", encoding="ISO-8859-1")

> df.head()

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	01-12-2010 08:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	01-12-2010 08:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	01-12-2010 08:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	01-12-2010 08:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	01-12-2010 08:26	3.39	17850.0	United Kingdom

Step 3: Data preparation and cleaning.

> # Check if there are any missing values, if so, drop all the missing values.

> round(100*(df.isnull().sum())/len(df), 2)

```
InvoiceNo       0.00
StockCode       0.00
Description     0.27
Quantity        0.00
InvoiceDate     0.00
UnitPrice       0.00
CustomerID     24.93
Country         0.00
dtype: float64
```

> df = df.dropna()

> df.shape

```
(406829, 8)
```

> # Create new features such as amount, frequency and recency

> # Amount: total amount purchased

> df['amount'] = df['Quantity']*df['UnitPrice']

> retail = df.groupby('CustomerID')['amount'].sum()

> retail = retail.reset_index()

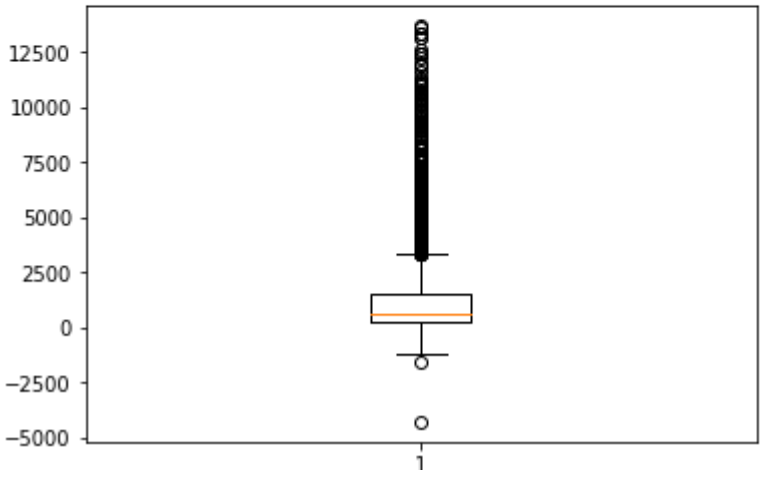
```
> # Frequency: Frequency of purchase
> frequency = df.groupby('CustomerID')['InvoiceNo'].count()
> frequency = frequency.reset_index()
> frequency.columns = ['CustomerID', 'frequency']
> # Merge the amount and frequency data frames
> retail = pd.merge(retail, frequency, on='CustomerID', how='inner')
> # Recency: Is the customer purchased recently, if so, how recent?
> # convert the invoice date to datetime
> df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'],  format='%d-%m-%Y %H:%M')
> latest_purchase = max(df['InvoiceDate'])
> df['diff'] = latest_purchase - df['InvoiceDate']
> df['diff'] = df['diff'].dt.days
> recency = df.groupby('CustomerID')['diff'].min()
> recency = recency.reset_index()
> retail = pd.merge(retail, recency, on='CustomerID', how='inner')
> retail.columns = ['CustomerID', 'amount', 'frequency', 'recency']
> retail.head()
```

	CustomerID	amount	frequency	recency
0	12346.0	0.00	2	325
1	12347.0	4310.00	182	1
2	12348.0	1797.24	31	74
3	12349.0	1757.55	73	18
4	12350.0	334.40	17	309

Step 4: Check for outliers in the newly created retail data frame.

```
> plt.boxplot(retail['amount'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x2fcfab70>,
  <matplotlib.lines.Line2D at 0x2fcfaeb8>],
 'caps': [<matplotlib.lines.Line2D at 0x2fcfaf98>,
  <matplotlib.lines.Line2D at 0x2fd05588>],
 'boxes': [<matplotlib.lines.Line2D at 0x2fcfa780>],
 'medians': [<matplotlib.lines.Line2D at 0x2fd058d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x2fd05c18>],
 'means': []}
```




```
>
```

removing (statistical) outliers

```
>
```

Q1 = retail.amount.quantile(0.05)

```
>
```

Q3 = retail.amount.quantile(0.95)

```
>
```

IQR = Q3 - Q1

```
>
```

retail = retail[(retail.amount >= Q1 - 1.5*IQR) & (retail.amount <= Q3 + 1.5*IQR)]

```
>
```

outlier treatment for recency

```
>
```

Q1 = retail.recency.quantile(0.05)

```
>
```

Q3 = retail.recency.quantile(0.95)

```
>
```

IQR = Q3 - Q1

```
>
```

retail = retail[(retail.recency >= Q1 - 1.5*IQR) & (retail.recency <= Q3 + 1.5*IQR)]

```
>
```

outlier treatment for frequency

```
>
```

Q1 = retail.frequency.quantile(0.05)

```
>
```

Q3 = retail.frequency.quantile(0.95)

```
>
```

IQR = Q3 - Q1

```
>
```

retail = retail[(retail.frequency >= Q1 - 1.5*IQR) & (retail.frequency <= Q3 + 1.5*IQR)]

Step 5: Rescaling, fit transform

> rfm_df = retail[['amount', 'frequency', 'recency']]

> # instantiate

> scaler = StandardScaler()

> # fit_transform

> rfm_df_scaled = scaler.fit_transform(rfm_df)

> rfm_df_scaled.shape

> rfm_df_scaled = pd.DataFrame(rfm_df_scaled)

> rfm_df_scaled.columns = ['amount', 'frequency', 'recency']

> rfm_df_scaled.head()

	amount	frequency	recency
0	-0.723738	-0.752888	2.301611
1	1.731617	1.042467	-0.906466
2	0.300128	-0.463636	-0.183658
3	0.277517	-0.044720	-0.738141
4	-0.533235	-0.603275	2.143188

Step 6: Model Building

> # Finding optimal number of clusters using Silhouette Analysis**.**

> # The value of the silhouette score range lies between -1 to 1.

> # A score closer to 1 indicates that the data point is very similar to other data points in the cluster,

> # A score closer to -1 indicates that the data point is not similar to the data points in its cluster.

> range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 10]

>for num_clusters in range_n_clusters:

> # intialise kmeans

> kmeans = KMeans(n_clusters=num_clusters, max_iter=50)

> kmeans.fit(rfm_df_scaled)

> cluster_labels = kmeans.labels_

```
>   # silhouette score

>   silhouette_avg = silhouette_score(rfm_df_scaled, cluster_labels)

>   print("For n_clusters={0}, the silhouette score is {1}".format(num_clusters, silhouette_avg))
```

```
For n_clusters=2, the silhouette score is 0.5411246404292333
For n_clusters=3, the silhouette score is 0.5084896296141937
For n_clusters=4, the silhouette score is 0.4816160113014515
For n_clusters=5, the silhouette score is 0.46437291959505433
For n_clusters=6, the silhouette score is 0.4169897163641209
For n_clusters=7, the silhouette score is 0.4158421705727445
For n_clusters=8, the silhouette score is 0.409404100940267
For n_clusters=10, the silhouette score is 0.3792371076160594
```

```
> # k=3 seems to be the optimal clusters, build the final model with k=3

> kmeans = KMeans(n_clusters=3, max_iter=50)

> kmeans.fit(rfm_df_scaled)
```

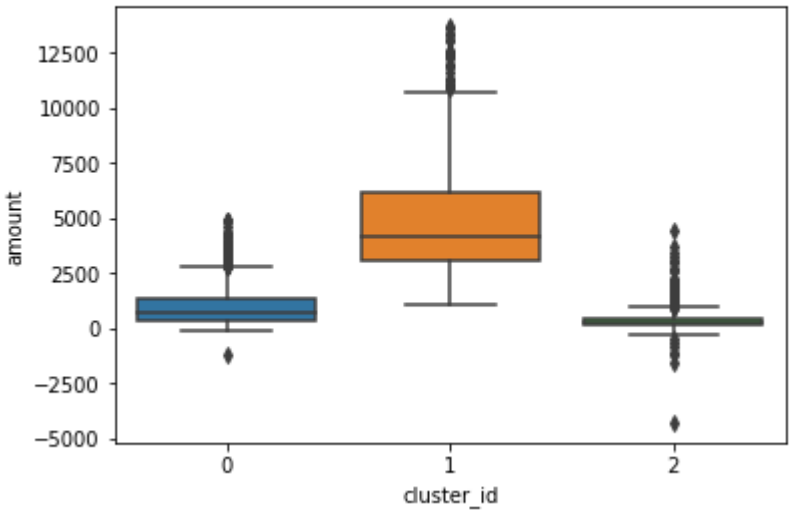
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
> retail['cluster_id'] = kmeans.labels_

> retail.head()
```

	CustomerID	amount	frequency	recency	cluster_id
0	12346.0	0.00	2	325	2
1	12347.0	4310.00	182	1	1
2	12348.0	1797.24	31	74	0
3	12349.0	1757.55	73	18	0
4	12350.0	334.40	17	309	2

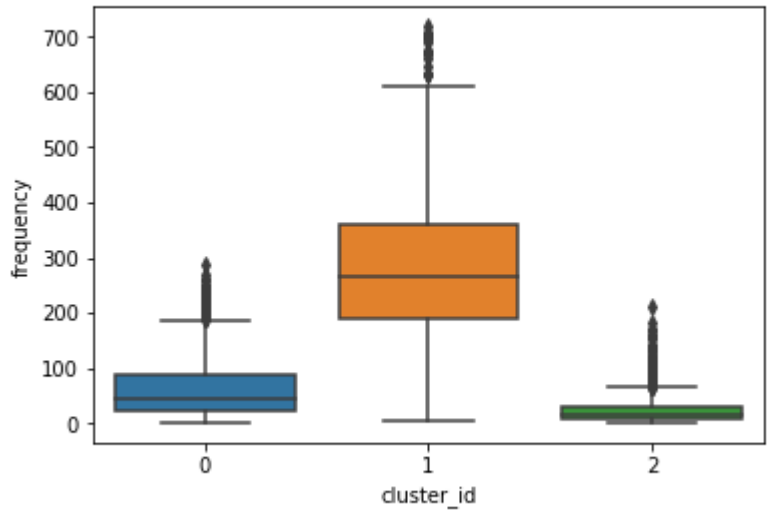
```
> print(accuracy_score(y_test,predictions))

> sns.boxplot(x='cluster_id', y='amount', data=retail)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b8e3cc0>
```



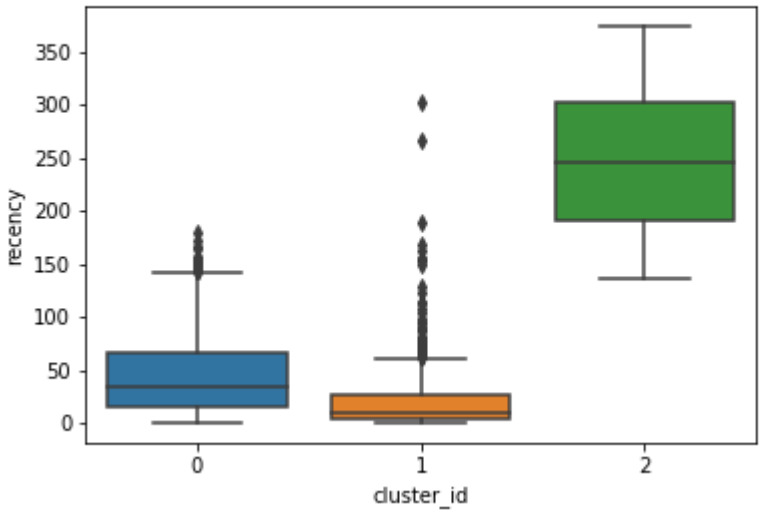
> sns.boxplot(x='cluster_id', y='frequency', data=retail)

```
<matplotlib.axes._subplots.AxesSubplot at 0x2bcb6ba8>
```



> sns.boxplot(x='cluster_id', y='recency', data=retail)

```
<matplotlib.axes._subplots.AxesSubplot at 0x2bd47a20>
```



Step 7: Summary

> 1. Customers with customer id 1 are highest amount of transactions. So they are the most valued customers.

> 2. Customers with customer_id 1 are more frequent purchasers

> 3. Customers with cutomer id 2 are less frequent and low value customers and not recent customers.

End of Exercise.

Exercise 13. Airline Passengers Prediction

Estimated time:

00:30 minutes

What this exercise is about:

This notebook demonstrates how to develop LSTM networks to predict the international airline passengers over time.

What you should be able to do:

At the end of this exercise, you will lean

- How to prepare data for building LSTM network.
- How to build LSTM network for airline passenger prediction.
- How to evaluate the LSTM model.

Introduction:

This problem is about predicting the airline passengers given the year and month. Let's apply neural network solve this prediction problem.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
>import pandas as pd

>import numpy as np

>from keras.models import Sequential

>from keras.layers import Dense

>from keras.layers import LSTM

>import math
```

```
>from sklearn.preprocessing import MinMaxScaler

>from sklearn.metrics import mean_squared_error, r2_score

>import matplotlib.pyplot as plt

>import warnings

>warnings.filterwarnings('ignore')
```

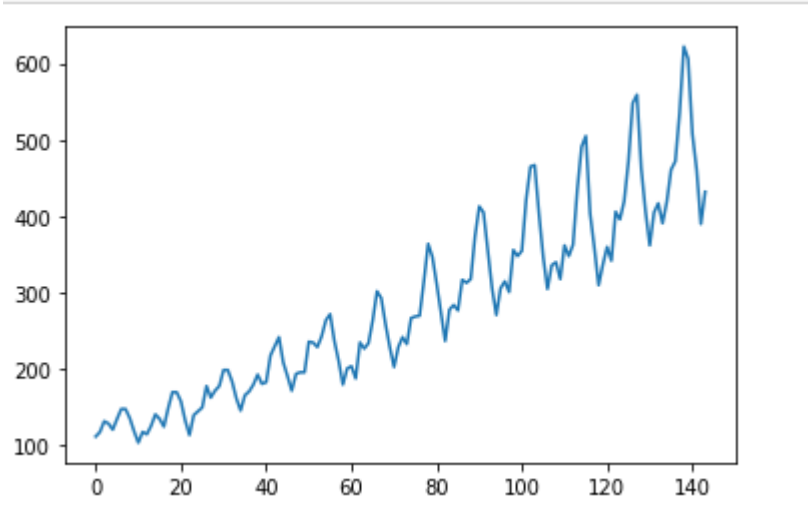
Step 2: Download dataset from https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csvand load to pandas data frame

```
>airline = pd.read_csv('airline_passengers.csv', usecols=[1], engine='python')

> airline.head()
```

Output:

	Passengers
0	112
1	118
2	132
3	129
4	121

```
# Visualize the time series data. We can see the upward trend of time series data.

>plt.plot(airline)

>plt.show()
```



```
# fix random seed for reproducibility
>np.random.seed(7)
# convert the integer values to float values, which are more suitable for modeling with a neural network.
>airline = airline.values
>airline = airline.astype('float32')
```

Step 3: Normalize and split the data into train and test

```
# normalize the dataset
>scaler = MinMaxScaler(feature_range=(0, 1))
>airline = scaler.fit_transform(airline)
# split into train and test sets
>train_size = int(len(airline) * 0.67)
>test_size = len(airline) - train_size
>train, test = airline[0:train_size,:], airline[train_size:len(airline),:]
# convert an array of values into a dataset matrix
>def create_dataset(airline, look_back=1):
    >dataX, dataY = [], []
  > for i in range(len(airline)-look_back-1):
        >a = airline[i:(i+look_back), 0]
        >dataX.append(a)
        >dataY.append(airline[i + look_back, 0])
   >return np.array(dataX), np.array(dataY)


# reshape into X=t and Y=t+1
>look_back = 1
>trainX, trainY = create_dataset(train, look_back)
>testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
```

```
>trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

>testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Step 4: Model Building.

```
# Model Building

# create and fit the LSTM network

>model = Sequential()

>model.add(LSTM(4, input_shape=(1, look_back)))

>model.add(Dense(1))

>model.compile(loss='mean_squared_error', optimizer='adam')

>model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

Output:

```
Epoch 1/100
 - 3s - loss: 0.0413
Epoch 2/100
 - 0s - loss: 0.0202
Epoch 3/100
 - 0s - loss: 0.0146
Epoch 4/100
 - 0s - loss: 0.0131
Epoch 5/100
 - 0s - loss: 0.0121
Epoch 6/100
 - 0s - loss: 0.0111
Epoch 7/100
 - 0s - loss: 0.0102
Epoch 8/100
 - 0s - loss: 0.0093
Epoch 9/100
 - 0s - loss: 0.0081
Epoch 10/100
 - 0s - loss: 0.0071
Epoch 11/100
 - 0s - loss: 0.0062
Epoch 12/100
 - 0s - loss: 0.0053
Epoch 13/100
 - 0s - loss: 0.0045
```

```
# make predictions
```

```
>trainPredict = model.predict(trainX)
>testPredict = model.predict(testX)

# invert the predictions
>trainPredict = scaler.inverse_transform(trainPredict)
>trainY = scaler.inverse_transform([trainY])
>testPredict = scaler.inverse_transform(testPredict)
>testY = scaler.inverse_transform([testY])
# calculate rmse
>trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
>print('Train Score: %.2f RMSE' % (trainScore))
>testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
> print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
>trainPredictPlot = np.empty_like(airline)
>trainPredictPlot[:, :] = np.nan
>trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
>testPredictPlot = np.empty_like(airline)
>testPredictPlot[:, :] = np.nan
>testPredictPlot[len(trainPredict)+(look_back*2)+1:len(airline)-1, :] = testPredict
# plot baseline and predictions
>plt.plot(scaler.inverse_transform(airline))
>plt.plot(trainPredictPlot, label='True')
>plt.plot(testPredictPlot, label='LSTM')
>plt.show()
```

Step 7: Summary and next Steps

> 1. We can see from above graph that LSTM does pretty good job in prediction

> 2. You have learnt how to build RNN model to for time series prediction problems.

> 3. You learnt how to evaluate a neural network model such as LSTM.

End of Exercise.

Exercise 14. Energy Efficiency Analysis

Estimated time:

30:00 minutes

What this exercise is about:

This notebook demonstrates the how to effectively design the energy requirements for residential buildings.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean and prepare the data set and perform exploratory data analysis.
- Build model to predict the Heating Load of Heating equipment.
- How to use ‘Repeated K-Fold’ to overcome over fitting problem.

Introduction:

We perform energy analysis using 12 different building shapes. The buildings differ with respect to the glazing area, the glazing area distribution, and the orientation, amongst other parameters. Use this data to predict the Heating Load of energy equipment required for the building.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

> import pandas as pd, numpy as np

> import matplotlib.pyplot as plt

> import seaborn as sns

> from sklearn.feature_selection import RFE

> from sklearn.ensemble import ExtraTreesRegressor

> from sklearn.model_selection import train_test_split

> import warnings

> warnings.filterwarnings('ignore')

```
> import xgboost

> from sklearn.metrics import accuracy_score

> from sklearn import model_selection
```

Step 2: Download dataset from http://archive.ics.uci.edu/ml/machine-learning-databases/00242/

and load to pandas data frame.

```
> energy  = pd.read_excel("ENB2012_data.xlsx")

> energy.head()
```

Output:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28

Step 3: Data cleaning and exploratory data analysis

```
# Let's check the dimensions of the dataframe

>energy.shape

# Let's check the missing values in the dataframe

>round(100*(energy.isnull().sum()/len(energy.index)), 2)
```

Output:

```
X1    0.0
X2    0.0
X3    0.0
X4    0.0
X5    0.0
X6    0.0
X7    0.0
X8    0.0
Y1    0.0
Y2    0.0
dtype: float64
```

We can See that there is no missing values

Name the columns based on the dataset metadata mentioned above

>energy.columns = ['rel_compactness', 'surface_area', 'wall_area', 'roof_area', 'overall_height',

'orientation', 'glazing_area', 'glazing_area_dist', 'heating_load', 'cooling_load']

>energy.head()

	rel_compactness	surface_area	wall_area	roof_area	overall_height	orientation	glazing_area	glazing_area_dist	heating_load	cooling_load
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28

>corr = energy.corr()

>corr.style.background_gradient(cmap='coolwarm').set_precision(2)

Output:

	rel_compactness	surface_area	wall_area	roof_area	overall_height	orientation	glazing_area	glazing_area_dist	heating_load	cooling_load
rel_compactness	1	-0.99	-0.2	-0.87	0.83	0	1.3e-17	1.8e-17	0.62	0.63
surface_area	-0.99	1	0.2	0.88	-0.86	0	1.3e-16	-3.6e-16	-0.66	-0.67
wall_area	-0.2	0.2	1	-0.29	0.28	0	-8e-19	0	0.46	0.43
roof_area	-0.87	0.88	-0.29	1	-0.97	0	-1.4e-16	-1.1e-16	-0.86	-0.86
overall_height	0.83	-0.86	0.28	-0.97	1	0	1.9e-18	0	0.89	0.9
orientation	0	0	0	0	0	1	0	0	-0.0026	0.014
glazing_area	1.3e-17	1.3e-16	-8e-19	-1.4e-16	1.9e-18	0	1	0.21	0.27	0.21
glazing_area_dist	1.8e-17	-3.6e-16	0	-1.1e-16	0	0	0.21	1	0.087	0.051
heating_load	0.62	-0.66	0.46	-0.86	0.89	-0.0026	0.27	0.087	1	0.98
cooling_load	0.63	-0.67	0.43	-0.86	0.9	0.014	0.21	0.051	0.98	1

1. As we can see from correlation matrix, overall_height, wall_area are heighly positively correlated with heating and cooling load
2. roof_area is highly negatively correlated with heating_load and cooling load.

Putting feature variable to X

>X = energy.drop(['heating_load', 'cooling_load'] ,axis=1)

Putting response variable to y

>Y = energy['heating_load']

>Y2 = energy['cooling_load']

Step 4: Model Building

#Feature Selection and Model Building

>model = ExtraTreesRegressor()

>rfe = RFE(model, 3)

>fit = rfe.fit(X, Y)

>print("Number of Features: ", fit.n_features_)

>print("Selected Features: ", fit.support_)

>print("'wall_area', 'roof_area' and 'overall_height' were top 3 selected by RFE for predicting 'heating_load'")

Output:

```
Number of Features:  3
Selected Features:  [False False  True  True  True False False False]
'wall_area', 'roof_area' and 'overall_height' were top 3 selected by RFE for predicting 'heating_load'
```

Fit the model

>seed = 7

>n_estimators = len(X)

>model.fit(X, Y)

>predictions = model.predict(X)

Evaluate the model

>kfold = model_selection.RepeatedKFold(n_splits=n_estimators, n_repeats=10, random_state=seed)

>cv_results = model_selection.cross_val_score(model, X, Y, cv=10)

>msg = "%s: %f (%f)" % ("ExtraTreesRegressor", cv_results.mean(), cv_results.std())

>print(msg)

Output:

```
ExtraTreesRegressor: 0.969070 (0.082337)
```

Summary

> 1. 'wall_area', 'roof_area' and 'overall_height' were top 3 selected for predicting 'heating_load'.

> 2. The accuracy of prediction by ExtraTreeRegressor is 97%.

> 3. Build model to predict the Cooling Load and then build model using other algorithms. Compare the models.

End of Exercise.

Exercise 15. Stock Price Prediction

Estimated time:

00:30 minutes

What this exercise is about:

This notebook demonstrates how to predict the price of a stock given the closing price, trading date of the stock.

What you should be able to do:

At the end of this exercise, you will lean

- How to prepare data for building LSTM network.
- How to build LSTM network for predicting the stock price of yahoo.
- How to evaluate the LSTM model.

Introduction:

This problem is about predicting the airline passengers given the date and time. Let's apply neural network to predict the stock price of yahoo.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
>import pandas as pd

>import numpy as np

>from keras.models import Sequential

>from keras.layers import Dense

>from keras.layers import LSTM

>import math

>from sklearn.preprocessing import MinMaxScaler
```

```
>from sklearn.metrics import mean_squared_error, r2_score

>import matplotlib.pyplot as plt

>import warnings

>warnings.filterwarnings('ignore')
```

Step 2: Download dataset from https://www.kaggle.com/dgawlik/nyse/downloadand load to pandas data frame

```
>prices = pd.read_csv('prices.csv')

>prices.head()
```

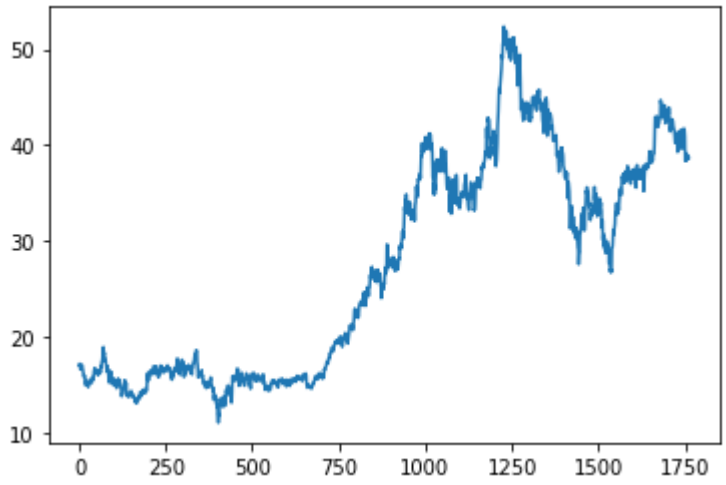
Output:

	date	symbol	open	close	low	high	volume
0	2016-01-05 00:00:00	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06 00:00:00	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07 00:00:00	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08 00:00:00	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11 00:00:00	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

Step 3: Data Preparation.

```
>yahoo = prices[prices['symbol']=='YHOO']

>yahoo_prices = yahoo.close.values.astype('float32')

>yahoo_prices = yahoo_prices.reshape(1762, 1)

>yahoo_prices.shape

>plt.plot(yahoo_prices)

>plt.show()
```

Output:




```
# fix random seed for reproducibility
>np.random.seed(7)
```

Step 4: Normalize and split the data into train and test

```
>scaler = MinMaxScaler(feature_range=(0, 1))
>yahoo_prices = scaler.fit_transform(yahoo_prices)
# split into train and test sets
>train_size = int(len(yahoo_prices) * 0.80)
>test_size = len(yahoo_prices) - train_size
>train, test = yahoo_prices[0:train_size,:], yahoo_prices[train_size:len(yahoo_prices),:]
>print(len(train), len(test))
# convert an array of values into a dataset matrix
>def create_dataset(dataset, look_back=1):
    >dataX, dataY = [], []
    >for i in range(len(dataset)-look_back-1):
        >a = dataset[i:(i+look_back), 0]
        >dataX.append(a)
       > dataY.append(dataset[i + look_back, 0])
   > return np.array(dataX), np.array(dataY)
```

reshape into X=t and Y=t+1

>look_back = 1

>trainX, trainY = create_dataset(train, look_back)

>testX, testY = create_dataset(test, look_back)

>trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

>testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

Step 5: Model Building.

Model Building

create and fit the LSTM network

>model = Sequential()

>model.add(LSTM(4, input_shape=(1, look_back)))

>model.add(Dense(1))

>model.compile(loss='mean_squared_error', optimizer='adam')

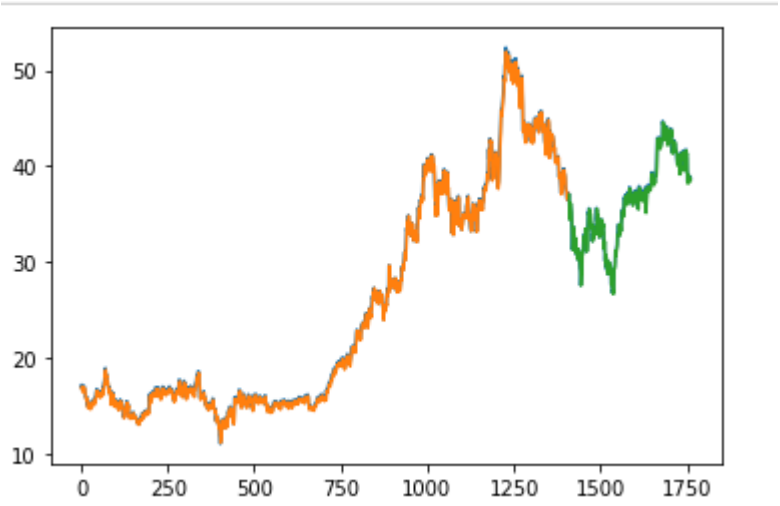
>model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)

```
Epoch 56/100
 - 3s - loss: 1.7429e-04
Epoch 57/100
 - 3s - loss: 1.7911e-04
Epoch 58/100
 - 3s - loss: 1.7982e-04
Epoch 59/100
 - 3s - loss: 1.8077e-04
Epoch 60/100
 - 4s - loss: 1.7813e-04
Epoch 61/100
 - 4s - loss: 1.7659e-04
Epoch 62/100
 - 4s - loss: 1.7562e-04
Epoch 63/100
 - 4s - loss: 1.8070e-04
Epoch 64/100
 - 4s - loss: 1.7853e-04
Epoch 65/100
 - 3s - loss: 1.7584e-04
```

Step 6: Model Evaluation

```
# make predictions
>trainPredict = model.predict(trainX)
>testPredict = model.predict(testX)


# invert the predictions
>trainPredict = scaler.inverse_transform(trainPredict)
>trainY = scaler.inverse_transform([trainY])
>testPredict = scaler.inverse_transform(testPredict)
>testY = scaler.inverse_transform([testY])
# calculate rmse
>trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
>print('Train Score: %.2f RMSE' % (trainScore))
>testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
>print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = np.empty_like(yahoo_prices)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = np.empty_like(yahoo_prices)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(yahoo_prices)-1, :] = testPredict
# plot baseline and predictions
>plt.plot(scaler.inverse_transform(yahoo_prices))
>plt.plot(trainPredictPlot, label='True')
>plt.plot(testPredictPlot, label='LSTM')
>plt.show()
```

Output:



Step 7: Summary and next Steps

> 1. We can see from above graph that LSTM does pretty good job in prediction of a stock price.

> 2.You have learnt how predict the price of stock using neural network.

> 3. Next step is, predict the price of other stocks in the dataset.

End of Exercise.

Exercise 16. Car Evaluation

Estimated time:

00:30 minutes

What this exercise is about:

This notebook demonstrates how to predict the condition of the car based on certain attributes such as maintenance cost, buying cost, boot space etc.

What you should be able to do:

At the end of this exercise, you will lean

- How to normalize, split data before building model.
- How to build model to predict the class of the car based on certain attributes.
- How to create / depict the confusion matrix.

Introduction:

The model evaluates the cars based on their overall price, maintenance cost, certain technical characteristics and comfort.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
>import pandas as pd

>import numpy as np

>from pandas import DataFrame,Series

>from sklearn.model_selection import train_test_split

>import matplotlib.pyplot as plt

>%matplotlib inline

>from sklearn.neighbors import KNeighborsClassifier

>from sklearn.preprocessing import StandardScaler

>from sklearn import model_selection
```

```
>from sklearn.metrics import confusion_matrix

>import seaborn as sns
```

Step 2: Download dataset from https://www.kaggle.com/ankiijindae/car-evaluation/download and load to pandas data frame

```
>cars = pd.read_csv('car.data.txt',encoding='utf-8',header=None)

>cars.head()
```

Output:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

Step 3 : Data Preparation

```
#The column values are as below

# Class Values:

## unacc, acc, good, vgood

# Attributes:

## buying: vhigh, high, med, low.

## maint: vhigh, high, med, low.

## doors: 2, 3, 4, 5more.

## persons: 2, 4, more.

## lug_boot: small, med, big.

## safety: low, med, high.
```

```
>cars.rename(columns =
{0:'buying_cost',1:'maintainence',2:'doors',3:'persons',4:'boot_space',5:'safety',6:'class'},inplace = True)
```

```
>class_type = {"unacc": 4, "acc": 3,'good': 2,'vgood':1}

>cars["class"] = cars["class"].map(class_type).astype(int)

>cars = pd.get_dummies(cars)

>cars.head()
```

	class	buying_cost_high	buying_cost_low	buying_cost_med	buying_cost_vhigh	maintainence_high	maintainence_low	maintainence_med	maintainence_vhi
0	4	0	0	0	1	0	0	0	
1	4	0	0	0	1	0	0	0	
2	4	0	0	0	1	0	0	0	
3	4	0	0	0	1	0	0	0	
4	4	0	0	0	1	0	0	0	

5 rows × 22 columns

Step 4: Separate Target variable and Split the data into test and train

```
# Putting feature variable to X

>X = cars.drop('class',axis=1)

# Putting response variable to y

y = cars['class']


# Splitting the data into train and test

>X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
```

Step 5: Model Building.

```
# To find optimum value of K we build a loop witch check all the posible values for K.

# Neighbors

>neighbors = [x for x in list(range(1,20))]

# Create empty list that will hold cv scores

>cv_scores = []

# Perform 10-fold cross validation on training set for odd values of k:

>seed=123

>for k in neighbors:

    >k_value = k+1
```

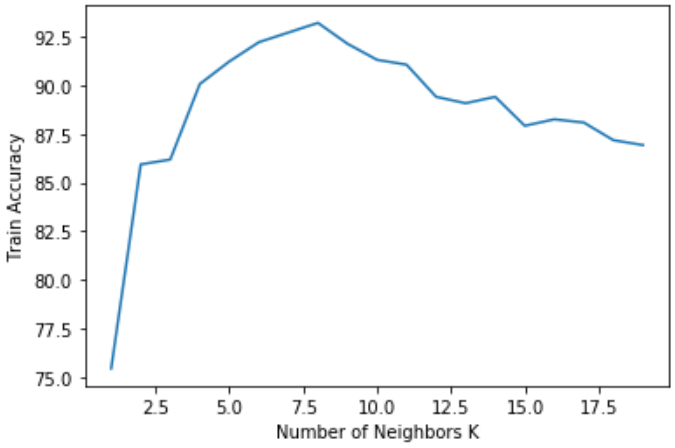
```
> knn = KNeighborsClassifier(n_neighbors = k_value, weights='uniform', p=2, metric='euclidean')
 >kfold = model_selection.KFold(n_splits=10, random_state=seed)
 >scores = model_selection.cross_val_score(knn, X_train, y_train, cv=kfold, scoring='accuracy')
> cv_scores.append(scores.mean()*100)
>optimal_k = neighbors[cv_scores.index(max(cv_scores))]
>print(( "The optimum number of neighbors is %d with %0.1f%%" % (optimal_k, cv_scores[optimal_k])))
```

Output:

```
The optimum number of neighbors is 8 with 92.1%
```

```
#Plot
>plt.plot(neighbors, cv_scores)
>plt.xlabel('Number of Neighbors K')
>plt.ylabel('Train Accuracy')
>plt.show()
```

Output:




```
# KNN
>knn = KNeighborsClassifier(n_neighbors = 8)
>knn.fit(X_train, y_train)
>y_pred = knn.predict(X_test)
>acc_train = round(knn.score(X_train, y_train) * 100, 2)
```

>acc_val = round(knn.score(X_test, y_test) * 100, 2)

>print("Accuracy of training dataset: " + str(acc_train))

>print("Accuracy of test dataset: "+ str(acc_val))

Output:

```
Accuracy of training dataset: 95.7
Accuracy of test dataset: 92.1
```

Calculate the Confusion Matrix

>confusion_mx = confusion_matrix(y_test.tolist(), y_pred.tolist())

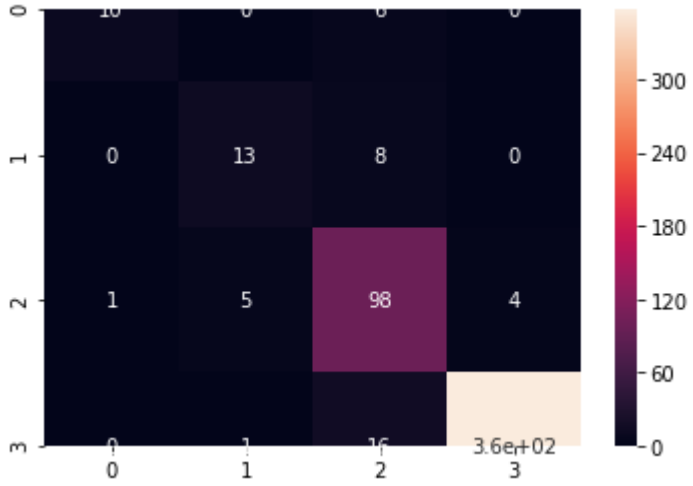
>confusion_mx

Output:

```
array([[ 10,   0,   6,   0],
       [  0,  13,   8,   0],
       [  1,   5,  98,   4],
       [  0,   1,  16, 357]], dtype=int64)
```

#Heatmap

```
> <matplotlib.axes._subplots.AxesSubplot at 0x1c8c6ab1b88>
```



Step 6: Summary and next Steps

> 1. The KNN model was able to predict the class of the car to an accuracy of 92.1%

> 2.How to create the confusion matrix and plot a heat map.

> 3. Next step is to try other machine learning models to check if the accuracy can be improved.

End of Exercise.

Exercise 17. Movie Sentiment Analysis

Estimated time:

00:30 minutes

What this exercise is about:

This notebook demonstrates how to build a sentiment analysis model to predict the sentiment towards a given movie.

What you should be able to do:

At the end of this exercise, you will lean

- How to clean and prepare the data for building Sentiment Analysis Model.
- How to build sentiment analysis model using nltk.
- How to evaluate the sentiment analysis model

Introduction:

The data set consists of movie reviews. Each record in the data set is labeled with two different sentiments i.e. 1: Positive, 0: Negative. We will build a machine learning model based on nltk to predict the sentiment towards a movie.

Requirements:

- Anaconda 3
- Windows 7 and above
- Jupyter notebook
- Nltk, download nltk data such as punkt and stopwords

Instructor exercise overview:

Procedure:

Step 1: First, import the required modules.

```
>import pandas as pd

>import numpy as np

>import matplotlib.pyplot as plt

>import bz2

>import gc

>import chardet
```

```
>import re

>import numpy as np

>import pickle

>from sklearn.model_selection import train_test_split

>import matplotlib.pyplot as plt

>from nltk.corpus import stopwords

>from nltk.tokenize import word_tokenize

>from sklearn.metrics import accuracy_score

>from sklearn.feature_extraction.text import TfidfVectorizer

>from sklearn.linear_model import SGDClassifier

>from sklearn.metrics import precision_score

>from sklearn.metrics import recall_score

>from sklearn.metrics import classification_report
```

Step 2: Download dataset from https://www.kaggle.com/marklvl/sentiment-labelled-sentences-data-set/download/dyBoTPZQ5clk6lFefCXS%2Fversions%2FzNHdKRYIdkIcw6NtxD1H%2Fdirectories%2Fsentiment%20labelled%20sentences%2Ffiles%2Fimdb_labelled.txt?datasetVersionNumber=2and load to pandas data frame

```
>imdb = pd.read_csv('imdb_labelled.txt',delimiter='\t',header=None)

>imdb.columns = ['Sentence','Class']

>imdb['index'] = imdb.index
```

Step 3: Data Preparation.

```
# Text Preprocessing

>columns = ['index','Class', 'Sentence']

>df_ = pd.DataFrame(columns=columns)

# lower string

>imdb['Sentence'] = imdb['Sentence'].str.lower()

# remove email adress
```

```
>imdb['Sentence'] = imdb['Sentence'].replace('[a-zA-Z0-9-_.]+@[a-zA-Z0-9-_.]+', '', regex=True)

# remove IP address

>imdb['Sentence'] = imdb['Sentence'].replace('((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(\.|$)){4}', '', regex=True)

# remove punctaitions and special chracters

>imdb['Sentence'] = imdb['Sentence'].str.replace('[^\w\s]','')

# remove numbers

>imdb['Sentence'] = imdb['Sentence'].replace('\d', '', regex=True)

# remove stop words

>for index, row in imdb.iterrows():

    >word_tokens = word_tokenize(row['Sentence'])

    >filtered_sentence = [w for w in word_tokens if not w in stopwords.words('english')]

  > df_ = df_.append({"index": row['index'], "Class":  row['Class'],"Sentence": " ".join(filtered_sentence[0:])},
>ignore_index=True)

>imdb = df_
```

Step 4:split the data into train and test

```
>X_train, X_test, y_train, y_test = train_test_split(imdb['Sentence'].values.astype('U'),imdb['Class'].values.astype('int32'), test_size=0.10, random_state=0)

>classes  = imdb['Class'].unique()
```

Step 5: Model Building.

```
# grid search result

>vectorizer = TfidfVectorizer(analyzer='word',ngram_range=(1,2),
>max_features=50000,max_df=0.5,use_idf=True, norm='l2')

>counts = vectorizer.fit_transform(X_train)

>vocab = vectorizer.vocabulary_

>classifier = SGDClassifier(alpha=1e-05,max_iter=50,penalty='elasticnet')

>targets = y_train

>classifier = classifier.fit(counts, targets)

>example_counts = vectorizer.transform(X_test)
```

>predictions = classifier.predict(example_counts)

Step 6: Model Evaluation

Model Evaluation

>acc = accuracy_score(y_test, predictions, normalize=True)

>hit = precision_score(y_test, predictions, average=None,labels=classes)

>capture = recall_score(y_test, predictions, average=None,labels=classes)

>print('Model Accuracy: %.2f'%acc)

>print(classification_report(y_test, predictions))

Output:

```
Model Accuracy: 0.67
              precision    recall  f1-score   support

           0       0.86      0.60      0.71        50
           1       0.50      0.80      0.62        25

    accuracy                           0.67        75
   macro avg       0.68      0.70      0.66        75
weighted avg       0.74      0.67      0.68        75
```

Step 7: Summary and next Steps

> 1. We can see the model was able to predict the sentiment with an accuracy of close to 70%

> 2. You have learnt how use nltk to build a sentiment analysis model.

> 3. Next step is, improve the accuracy of model. Explore the approaches.

End of Exercise.