New Package Introduction - Introduction to Supervised Learning and Regression

**Scikit-Learn**

Some of the common functions used from the Scikit-learn library in the hands-on applications related to the first lecture are listed below:

1) **from sklearn.linear\_model import LinearRegression**

A predictive model can be fitted, using linear regression, to an observed dataset of target variable and the independent variables.

sklearn.linear\_model.**LinearRegression**(*\**, *fit\_intercept=True*, *normalize='deprecated'*, *copy\_X=True*, *n\_jobs=None*, *positive=False*)

Example

from sklearn.linear\_model import LinearRegression  
model = LinearRegression()  
model.fit(xtrain, ytrain)

You can refer to the LinearRegression sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html).

2) **from sklearn.model\_selection import train\_test\_split**

Machine learning algorithms that are applicable to prediction-based algorithms and applications are fitted on one part of the data, called training data, and are evaluated on the other (smaller) part of the data, called test data. We split the data into train and test sets using the train\_test\_split function.

sklearn.model\_selection.**train\_test\_split**(*\*arrays*, *test\_size=None*, *train\_size=None*, *random\_state=None*, *shuffle=True*, *stratify=None*)

Example

from sklearn.model\_selection import train\_test\_split  
# Creating the training and testing data from X and Y   
xtrain,xtest,ytrain,ytest = train\_test\_split(X,Y,test\_size=0.2, random\_state=42)

You can refer to the train\_test\_split sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html).

3) **from sklearn.preprocessing import MinMaxScaler**

MinMaxscaler is a scaling technique for continuous variables used in machine learning. The minimum and maximum values are scaled to be 0 and 1, respectively, by the MinMaxscaler.

sklearn.preprocessing.**MinMaxScaler**(*feature\_range=(0, 1)*, *\**, *copy=True*, *clip=False*)

Example

from sklearn.preprocessing import MinMaxScalerscaler = MinMaxScaler()  
# scaling the training data  
scaled\_train = scaler.fit\_transform(x\_train)

You can refer to the MinMaxscaler sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html).

4) **from sklearn.model\_selection import cross\_val\_score**

In applied machine learning, cross-validation is mostly used to evaluate how well a machine-learning model performs on untrained data. That is, to use a small sample to assess how the model will generally perform when used to generate predictions on data that was not utilized during the model's training.

To start the cross-validation process, the data is divided into k folds and shuffled (to avoid any unintended ordering problems). The following procedure is followed for each of the k-folds:

1. A model is trained using of the folds as training data
2. The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure).

sklearn.model\_selection.**cross\_val\_score**(*estimator*, *X*, *y=None*, *\**, *groups=None*, *scoring=None*, *cv=None*, *n\_jobs=None*, *verbose=0*, *fit\_params=None*, *pre\_dispatch='2\*n\_jobs'*, *error\_score=nan*)

You can refer to the cross\_val\_score sklearn documentation for a better understanding of the parameters and attributes [here](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html).

**Statistical functions (statsmodels):**

A Python package called statsmodels offers classes and functions for estimating a variety of statistical models, running statistical tests, and exploring statistical data.

Statsmodels can be installed easily by running the below command in your Jupyter notebook:

**Installation:**

!pip install statsmodels

Once the command runs successfully, restart the kernel, and then we can import the library by running the following command in a Jupyter notebook

import statsmodels.api as sm

5)**Ordinary Least Squares(OLS):**

A popular method for calculating the coefficients of linear regression equations that represent the relationship between one or more independent quantitative variables and a dependent variable is ordinal least squares regression (OLS). It's applicable for simple or multiple linear regression.

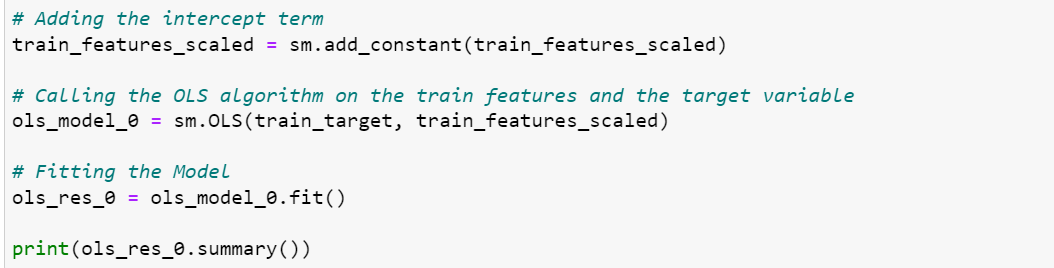
OLS(endog, exog, missing, hasconst); here endog = dependent variable, exog = independent variables

You can refer to the documentation for a better understanding of the parameters and attributes [here](https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.html).

**To fit and summarize the model**

model = sm.OLS(y, x)  
res = model.fit()  
print(res.summary())

**Below we can  find the function implementation in the BigMart\_Sales\_Prediction case study:**

****

6)**Variance inflation factor(VIF):**

The variance inflation factor measures how much the variance of parameter estimates will rise when the exog idx (x) variable is added to the linear regression. It is a measurement of the exog design matrix's multicollinearity.

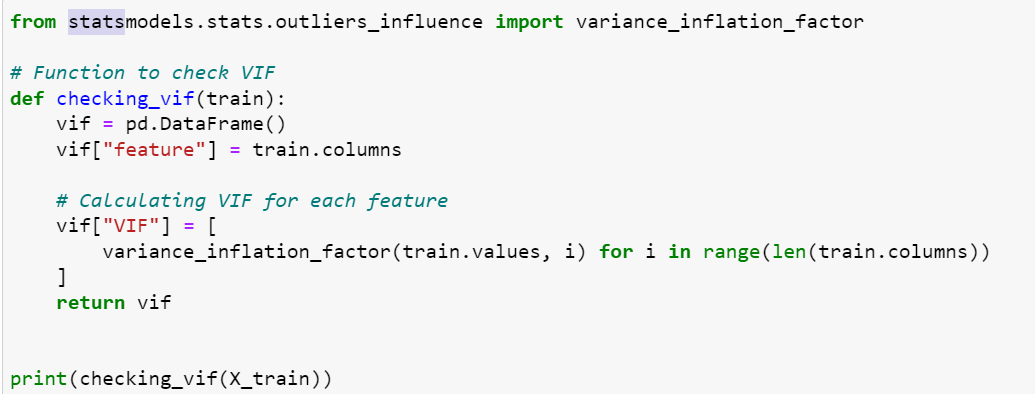
It's suggested to assume that if VIF is more than 5, the explanatory variable provided by exog idx is highly correlated with the other explanatory variables and that, as a result, the parameter estimates will have significant standard errors.

**To import the function, run the following command in a Jupyter notebook**

from stats.model.stats.outliers\_influence import variance\_inflation\_factor

You can refer to the documentation for a better understanding of the parameters and attributes [here](https://www.statsmodels.org/dev/generated/statsmodels.stats.outliers_influence.variance_inflation_factor.html?highlight=vif).

**Below we can  find the function implementation in the BigMart\_Sales\_Prediction case study:**

****

**Regression Diagnostics and specification tests:**

We don't know whether our statistical model is correctly described in many scenarios of statistical analysis. For example, when using ols we assume linearity and homoscedasticity. Some test statistics additionally assume that the errors are normally distributed or that we have a large sample. Because our results are dependent on these statistical assumptions, the results are only correct if our assumptions are correct (at least approximately).

* Homoscedasticity - If the residuals are symmetrically distributed across the regression line, then the data is said to be homoscedastic.
* Heteroscedasticity - If the residuals are not symmetrically distributed across the regression line, then the data is said to be heteroscedastic. In this case, the residuals can form a funnel shape or any other non-symmetrical shape

The diagnostic tests for linear regression are described briefly below.

7)**Heteroscedasticity tests:**

The null hypothesis for these tests is that all observations have the same error variance, i.e., the errors are homoscedastic. The tests differ in terms of the type of heteroscedasticity accepted as an alternate hypothesis.

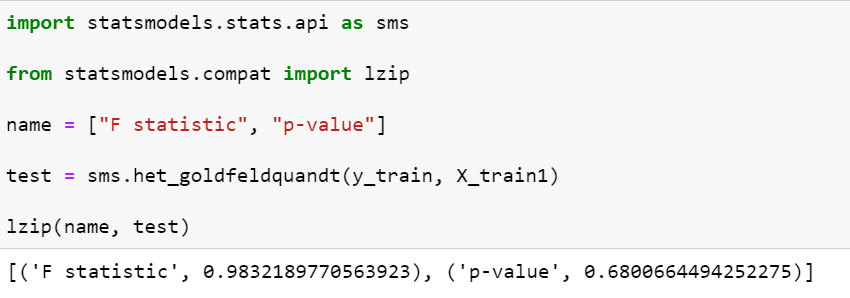
**het\_goldfeldquandt:**It is used to test the presence of heteroscedasticity in the given data.

**To import the function, run the following command in a Jupyter notebook**

from statsmodels.compat import lzip  
GQ\_test = sms.het\_goldfeldquandt(y\_train, x\_train)

You can refer to the documentation for a better understanding of the parameters and attributes [here](https://www.statsmodels.org/stable/diagnostic.html).

**Below we can find the function implementation in the superkart case study:**

****

**SciPy (Scientific Python):**

An open-source Python library called SciPy is used to solve problems in science and mathematics. It is based on the NumPy extension and offers a large variety of high-level functions for manipulating and visualizing data. It provides many user-friendly and effective numerical functions for numerical integration and optimization.

**scipy.stats package is imported as**

from scipy import stats

And in some cases, we assume that individual objects are imported as:

**f**rom scipy.stats import norm

You can refer to the documentation for a better understanding of the parameters and attributes[here](https://docs.scipy.org/doc/scipy/tutorial/stats.html).

8)**Q\_Q plot:**

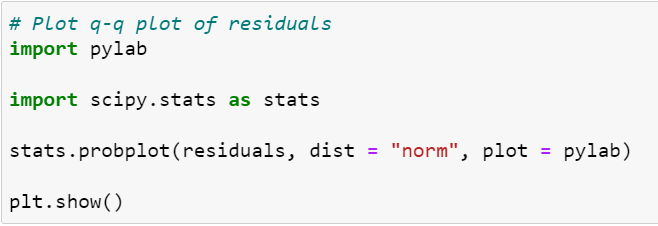
The plot obtained is known as a quantile-quantile plot, or Q\_Q plot, when the quantiles of two variables are placed against one another. With respect to the locations, this plot summarises whether or not the distributions of two variables are similar.

**To import the function, run the following command in a Jupyter notebook**

import scipy.stats as ststs  
scipy.stats.probplot(x, sparams=(), dist='norm', fit=True, plot=None, rvalue=False)

You can refer to the documentation for a better understanding of the parameters and attributes [here](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.probplot.html).

**Below we can find the function implementation in the superkart case study:**



 9)**Pearsonr:**

A linear relationship between two variables' strength and direction is measured by the Pearson correlation coefficient. Values usually fall between -1 and 1, with 1 denoting a perfectly positive correlation and -1 denoting a perfectly inverse relationship.

**To import the function, run the following command in a Jupyter notebook**

from scipy.stats import pearsonr  
scipy.stats.pearsonr(x, y, \*, alternative = 'two sided')

You can refer to the documentation for a better understanding of the parameters and attributes [here](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html).

Happy Learning!