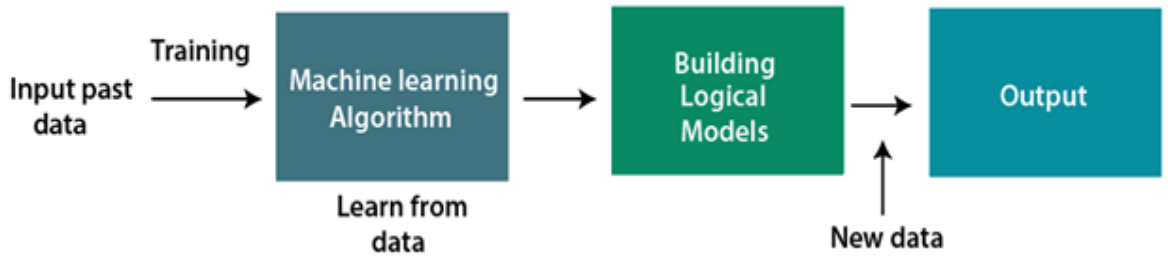
Introduction to machine Learning

Machine Learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed. The term machine learning was first introduced by **Arthur Samuel** in **1959**.
Machine learning is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that makes it more similar to humans: The ability to learn. Machine learning is actively being used today, perhaps in many more places than one would expect.

How machine learning work:

A Machine Learning system learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it. The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.
Suppose we have a complex problem, where we need to perform some predictions, so instead of writing a code for it, we just need to feed the data to generic algorithms, and with the help of these algorithms, machine builds the logic as per the data and predict the output. Machine learning has changed our way of thinking about the problem. The below block diagram explains the working of Machine Learning algorithm:



Need for machine learning:

The need for machine learning is increasing day by day. The reason behind the need for machine learning is that it is capable of doing tasks that are too complex for a person to implement directly. As a human, we have some limitations as we cannot

access the huge amount of data manually, so for this, we need some computer systems and here comes machine learning to make things easy for us.

We can train machine learning algorithms by providing them with a huge amount of data and let them explore the data, construct the models, and predict the required output automatically. The performance of the machine learning algorithm depends on the amount of data, and it can be determined by the cost function. With the help of machine learning, we can save both time and money.

Some key points which shows the importance of machine learning:

- Solving complex problems, which are difficult for a human.
- Rapid increment in the production of data.
- Decision making in various sectors including finance.
- Finding hidden patterns and extracting useful information from data.

Some common applications of machine learning:

- **E-commerce product recommendation:**

  One of the prominent elements of typically any e-commerce website is product recommendation, which involves the sophisticated use of machine learning algorithms. Websites track customer behavior based on past purchases, browsing habits, and cart history and then recommend products using machine learning and AI.

- **Catching email spam:**

  One of the most popular applications of machine learning that everyone is familiar with is in detecting email spam. Email service providers build applications with spam filters that use an ML algorithm to classify an incoming email as spam and direct it to the spam folder.

- **Self-Driving Cars:**

  Self-driving cars use an unsupervised learning algorithm that heavily relies on machine learning techniques. This algorithm enables the vehicle to collect information from cameras and sensors about its surroundings, understand it, and choose what actions to perform.

- **Online fraud Detection:**

  One of the most essential applications of machine learning is fraud detection. Every time a customer completes a transaction, the machine learning model carefully examines their profile in search of any unusual patterns to detect online fraud.

- **Stock market trading:**

  When it comes to the stock market and day trading, machine learning employs algorithmic trading to extract important data to automate or support crucial investment activities. Successful portfolio management, and choosing when to buy and sell stocks are some tasks accomplished using ML.

- **Catching malware:**

  The process of using machine learning (ML) to detect malware consists of two basic stages. First, analyzing suspicious activities in an Android environment to generate a suitable collection of features; second, training the system to use the machine and deep learning (DL) techniques on the generated features to detect future cyberattacks in such environments.

- **Speech Recognition:**

  ML software can make measurements of words spoken using a collection of numbers that represent the speech signal. Popular applications that employ speech recognition include Amazon's Alexa, Apple's Siri, and Google Maps.

- **Image Recognition:**

  One of the most notable machine learning applications is image recognition, which is a method for cataloging and detecting an object or feature in a digital image. In addition, this technique is used for further analysis, such as pattern recognition, face detection, and face recognition.

- **Cancer prognosis and prediction:**

  As ML algorithms can identify critical traits in complicated datasets, it is applied in cancer research. It is used to construct prediction models using techniques like Artificial Neural Networks (ANNs), Bayesian Networks (BNs), and Decision Trees (DTs). This helps in precise decision-making and modeling of the evolution and therapy of malignant diseases.

Classification of machine learning

It is classified into:

1. Supervised machine learning.
2. Unsupervised machine learning.

Supervised machine

Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.
Supervised learning further divided into two categories:

- Regression.
- Classification.

Unsupervised machine learning:

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.
Supervised learning further divided into two categories:

- Clustering.
- Association.

Scikit-Learn:

scikit-learn, also known as sklearn, is a popular open-source machine learning library for Python. It provides a wide range of tools and algorithms for various tasks in machine learning, including classification, regression, clustering, dimensionality reduction, and more. Here are some key features and components of scikit-learn:

1. Consistent API: scikit-learn provides a consistent and intuitive API for building and using machine learning models. It follows the same pattern for all algorithms, making it easy to switch between different models without major code modifications.

2. Comprehensive Algorithms: The library offers a wide range of machine learning algorithms, including linear regression, logistic regression, support vector machines (SVM), random forests, gradient boosting, k-means clustering, and more. These algorithms are implemented efficiently and optimized for large-scale datasets.

3. Data Preprocessing: scikit-learn provides various tools for data preprocessing, including handling missing values, feature scaling, encoding categorical variables, and feature extraction. These preprocessing techniques help to prepare the data before training the models.

4. Model Evaluation: The library offers metrics and scoring functions for evaluating the performance of machine learning models. It includes metrics for classification (e.g., accuracy, precision, recall, F1-score), regression (e.g., mean squared error, R-squared), and clustering (e.g., silhouette coefficient). It also supports techniques like cross-validation and hyperparameter tuning for robust model evaluation.

5. Feature Selection and Dimensionality Reduction: scikit-learn provides methods for feature selection, such as selecting the most important features or removing irrelevant features. It also offers dimensionality reduction techniques like principal component analysis (PCA) and manifold learning methods to reduce the number of features while preserving important information.

6. Integration with NumPy and pandas: scikit-learn seamlessly integrates with other popular Python libraries like NumPy and pandas. It accepts NumPy arrays and pandas DataFrames as input, making it convenient to work with data in those formats.

7. Community and Documentation: scikit-learn has a vibrant community, with active development and ongoing support. It provides comprehensive documentation, including user guides, API references, tutorials, and examples, to help users get started and understand the library's capabilities.

Overall, scikit-learn is widely used for its simplicity, versatility, and extensive collection of machine learning tools. It is suitable for both beginners and experienced practitioners, enabling them to build and deploy machine learning models effectively.

Installation using pip:

Pip install scikit-learn

Installation using conda:

Conda install scikit-learn

After the installation is complete, you can import scikit-learn in your python script or interactive environment.

Import scikit-learn

Train-test split:

The train-test split is a technique used in machine learning to divide a dataset into two separate subsets: the training set and the testing set. The training set is used to train the machine learning model, while the testing set is used to evaluate the model's performance on unseen data.

Syntax for train-test split using `train_test_split` function:

```
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Explanation:

1. Import the `train_test_split` function from the `sklearn.model_selection` module:
   ```python
   from sklearn.model_selection import train_test_split
   ```
2. Call the `train_test_split` function and pass the input features (`X`) and the target variable (`y`) as arguments, along with additional parameters:
   ```python
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  ```
   `X`: The input features of the dataset.
   `y`: The corresponding target variable.
   **`test_size`:** This parameter specifies the proportion of the dataset that should be allocated to the testing set. It can be a float value (0.0 to 1.0) representing the percentage of the data, or an integer representing the absolute number of samples.
   **`random_state`:** This parameter sets the random seed, which ensures reproducibility of the split. It allows you to obtain the same split every time you run the code with the same seed value. It's optional but recommended for reproducibility purposes.
   ```

3. The `train_test_split` function returns four subsets of the data:
   `X_train`: The training set's input features.
   `X_test`: The testing set's input features.
   `y_train`: The training set's target variable.
   `y_test`: The testing set's target variable.

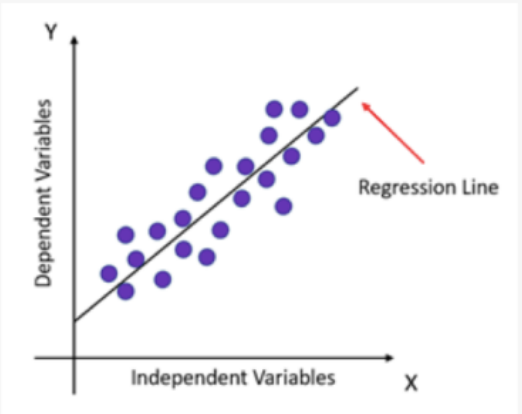
After the split, you can use `X_train` and `y_train` to train your machine learning model, and then use `X_test` to make predictions or evaluate the model's performance by comparing the predicted values to `y_test`.

Remember to adapt the code to your specific dataset and task, ensuring that `X` and `y` contain the appropriate data.

Supervised machine learning

Linear Regression:

Regression is a statistical technique that relates a dependent variable to one or more independent variables. A regression model is able to show whether changes observed in the dependent variable are associated with changes in one or more of the independent variables.



Linear regression line

Linear regression can be expressed mathematically as:

$y = b_0 + b_1 x + \varepsilon$

Here,

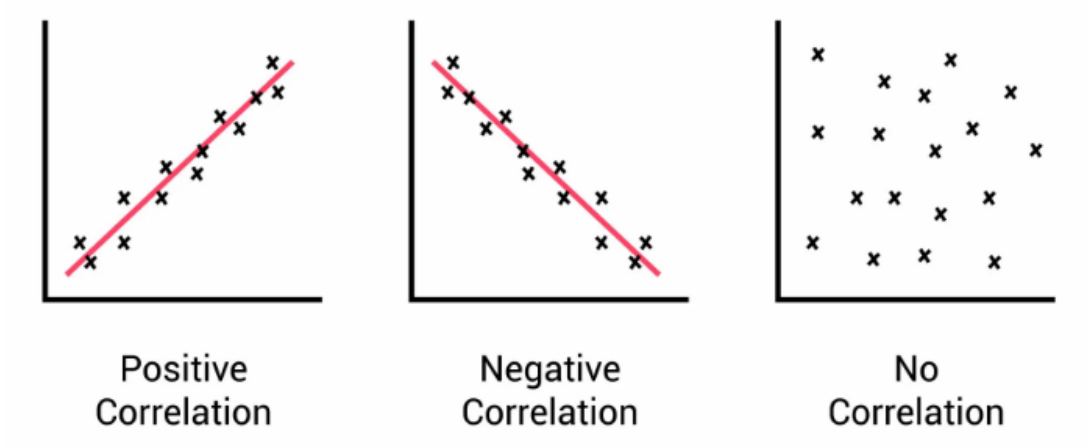
- y= Dependent variable
- x= Independent variable

- b_0= Intercept of the line
- b_1= Linear regression coefficient (slope of the line)
- ε= Random error

The last parameter, random error ε, is required as the best fit line also doesn't include the data points perfectly.

Linear regression line:

A linear line showing the relationship between the dependent and independent variables is called a regression line. A regression line can show two types of relationship:



Positive linear relationship:

If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship. Equation: $y = b_0+b_1x$

Negative linear relationship:

If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship. Equation: $y = -b_0+b_1x$

Zero correlation:

A zero correlation exists when there is no relationship between two variables.

Simple linear regression:

It is a type of regression algorithm that models the relationship between a dependent variable and a single independent variable.

The key point in simple linear regression is that the dependent variable must be a continuous value. However, the independent variable can be measured on

continuous and categorical values. Simple linear regression algorithm has mainly two objectives:

- Model the relationship between the two variables: Such as the relationship between Income and expenditure, experience and salary, etc.
- Forecasting new observations: Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

A simple straight-line equation involving slope (dy/dx) and intercept (an integer/continuous value) is utilized in simple Linear Regression.

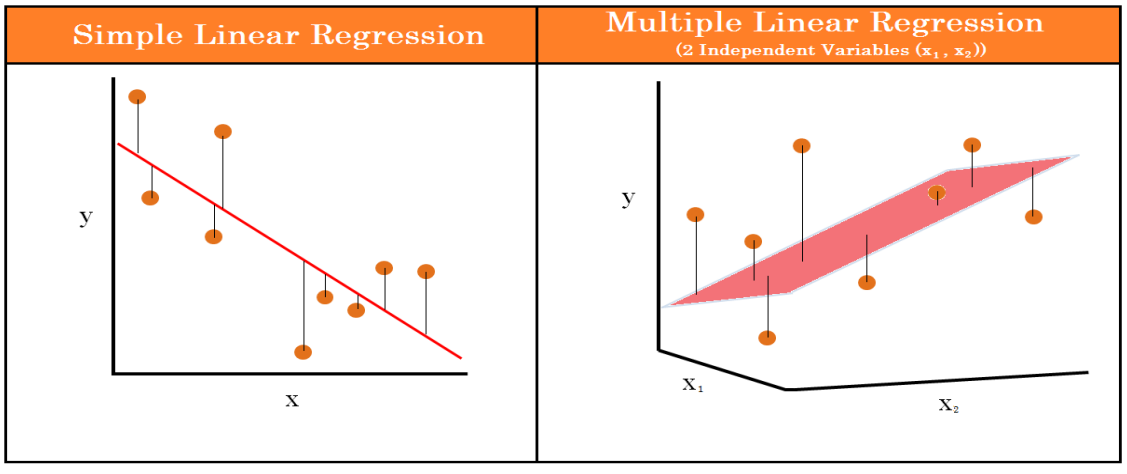
Here a simple form is:

y=mx+c where y denotes the output x is the independent variable, and c is the intercept when x=0. With this equation, the algorithm trains the model of machine learning and gives the most accurate output.

Multiple linear regression:

When a number of independent variables more than one, the governing linear equation applicable to regression takes a different form like:

y= c+m1x1+m2x2... mnxn which represents the coefficient responsible for impact of different independent variables x1, x2 etc. This machine learning algorithm, when applied, finds the values of coefficients m1, m2, etc., and gives the best fitting line.



Simple linear vs multiple linear regression

Advantages:

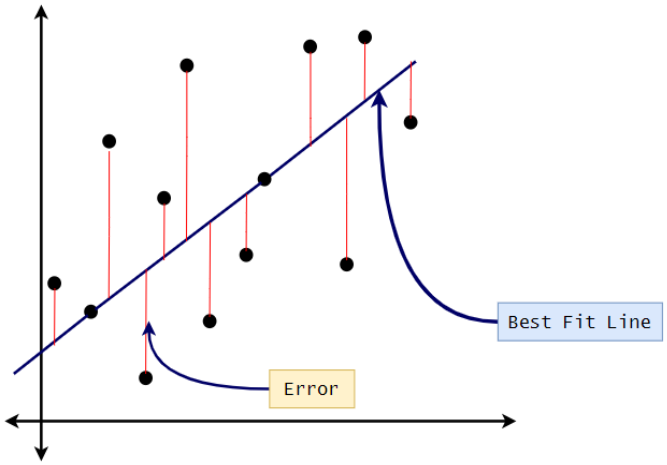
- Simple to implement.

- Perform well on the data with linear relationship.

Disadvantages:

- Not suitable for data having non-linear relationship.
- Underfitting issue.
- Sensitive to outliers.

Find the best fit line:



- In linear regression our purpose is to find the best fit line(model). So, here" loss function" comes into picture.

Loss function:

- Loss function measures how far an estimated value is from its true value, it is helpful to determine which model performs better and which parameters are better.

Loss function/MSE $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

- From the above formula, the error difference is squared for each value and then the average of the sum of squares of error gives us the cost function. It is also referred to as Mean Square Error (MSE).
- Low loss value → High accuracy.
- High loss value → Low accuracy.
- We can improve the model by some optimization technique called "gradient descent".

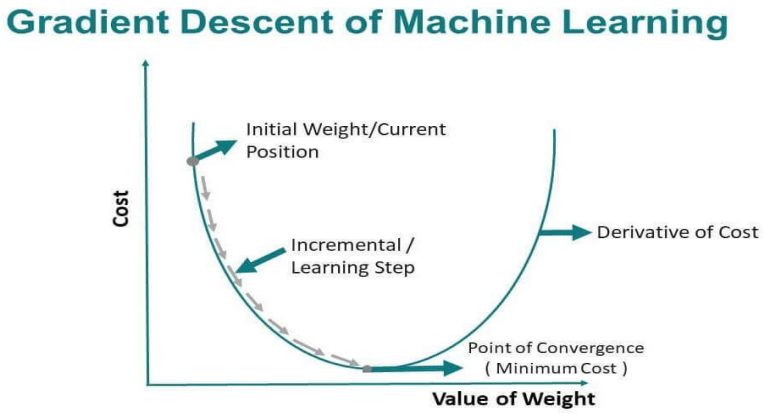
Residuals:

The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and

so cost function will be high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively updating the values to reach the minimum cost function.



Model performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called optimization. It can be achieved by the R-squared method.

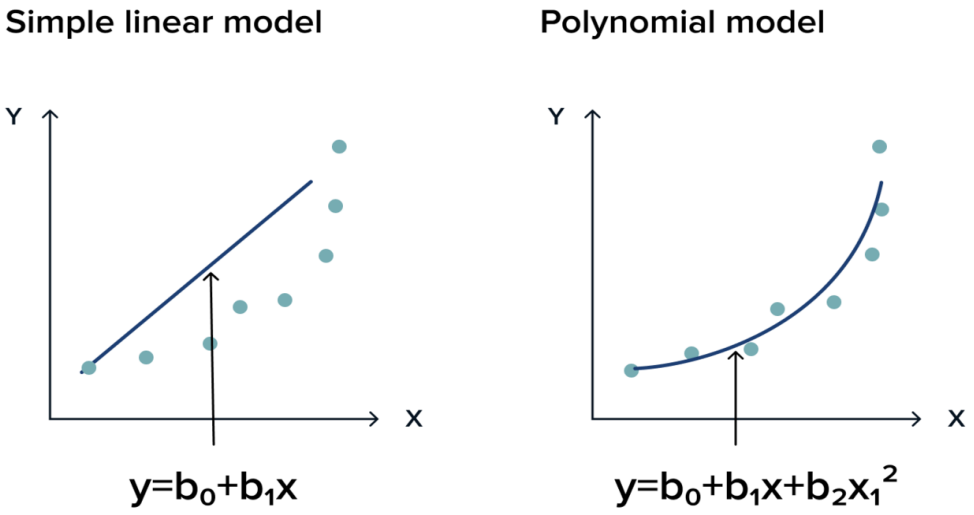
Polynomial regression:

- It is a regression algorithm that model the relationship between a dependent variable(y) and independent variable(x) as nth degree polynomial.

$$y= c+m_1x_1 + m_2x_1^2 + m_3x_1^3 + m_4x_1^4 + \ldots\ldots\ldots m_nx_1^n$$

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.I
- Hence, "In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,4,..,n) and then modeled using a linear model.

- Polynomial regression is required because if we apply a linear model on a linear dataset, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a non-linear dataset, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, where data points are arranged in a non-linear fashion, we need the Polynomial Regression model.



- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.

Assumptions of Linear Regression:

Normally most statistical tests and results rely upon some specific assumptions regarding the variables involved. Naturally, if these assumptions are not considered, the results will not be reliable. Linear Regression also comes under the same consideration. There are some common assumptions to be considered while using Linear Regression:

1. **Linearity:** The models of Linear Regression models must be linear in the sense that the output must have a linear association with the input values, and it only suits data that has a linear relationship between the two entities.

2. **Homoscedasticity:** Homoscedasticity means the standard deviation and the variance of the residuals difference of $(y - \hat{y})^2$ must be the same for any value of x. Multiple Linear Regression assumes that the amount of error in the residuals is similar at each point of the linear model. We can check the Homoscedasticity using Scatter plots.
3. **Non-multicollinearity:** The data should not have multicollinearity, which means the independent variables should not be highly correlated with each other. If this occurs, it will be difficult to identify those specific variables which actually contribute to the variance in the dependent variable. We can check the data for this using a correlation matrix.
4. **No Autocorrelation:** When data are obtained across time, we assume that successive values of the disturbance component are momentarily independent in the conventional Linear Regression model. When this assumption is not followed, the situation is referred to as autocorrelation.
5. **Not applicable to Outliers:** The value of the dependent variable cannot be estimated for a value of an independent variable which lies outside the range of values in the sample data.

All the above assumptions are critical because if they are not followed, they can lead to drawing conclusions that may become invalid and unreliable.

Evaluation metrics for regression

Mean Absolute Error(MAE):

- MAE is a simple metric which calculates the absolute difference between actual and predicted values.
- To better understand, let's take an example where you have input data and output data and use Linear Regression, which draws a best-fit line.
- Now you have to find the MAE of your model which is basically a mistake made by the model known as an error. Now find the difference between the actual value and predicted value that is an absolute error but we have to find the mean absolute of the complete dataset.
- so, sum all the errors and divide them by a total number of observations, and this is MAE. And we aim to get a minimum MAE because this is a loss.

MAE equation, $\text{MAE}=\frac{1}{N}\sum\left|y - \hat{y}\right|$

N= Total number of data points, y=Actual output, $\hat{y}$= Predicted output.

$\sum|y - \hat{y}|$= Sum of actual values of residuals.

Advantages of MAE

- The MAE you get is in the same unit as the output variable.
- It is most Robust to outliers.

Disadvantages of MAE

- The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

Mean Squared Error(MSE):

- MSE represents the squared distance between actual and predicted values. We perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.

  MSE equation: MSE=$\frac{1}{N}(y - \hat{y})^2$

Advantages of MSE

- The graph of MSE is differentiable, so you can easily use it as a loss function.

Disadvantages of MSE

- The value you get after calculating MSE is a squared unit of output. for example, the output variable is in meter(m) then after calculating MSE the output we get is in meter squared.
- If you have outliers in the dataset then it penalizes the outliers most and the calculated MSE is bigger. So, in short, It is not Robust to outliers which were an advantage in MAE.

Root Mean Squared Error(RMSE):

- As RMSE is clear by the name itself, that it is a simple square root of mean squared error.

  RMSE Equation: RMSE= $\sqrt{\frac{1}{N}(y - \hat{y})^2}$

Advantages of RMSE

- The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

Disadvantages of RMSE

- It is not that robust to outliers as compared to MAE.

R-Squared(R2):

- R squared is a popular metric for identifying model accuracy. It tells how close the data points to the fitted line generated by a regression algorithm.

$$\text{R-Squared(R2)} = 1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2} = 1 - \frac{Sum\ of\ squared\ residuals(SSR)}{Total\ sum\ of\ squares(SST)}$$

Here,

- 'y' is the actual target value.
- '$\hat{y}$' is the predicted target value.
- '$\bar{y}$' is the mean of the actual target value.

- R^2 score ranges from 0 to 1. The closer to 1 the R^2, the better the regression model is. If R^2 is equal to 0, the model is not performing better than a random model. If R^2 is negative, the regression model is erroneous.
- But the problem is when we add an irrelevant feature in the dataset then at that time R2 sometimes starts increasing which is incorrect.

Adjusted R-Squared(R2):

- Adjusted R^2 is the same as standard R^2 except that it penalizes models when additional features are added.
- To counter the problem which is faced by R-square, Adjusted r-square penalizes adding more independent variables which don't increase the explanatory power of the regression model.
- The value of adjusted r-square is always less than or equal to the value of r-square.
- It ranges from 0 to 1, the closer the value is to 1, the better it is.
- It measures the variation explained by only the independent variables that actually affect the dependent variable.

Adjusted R-Squared Equation: $$R^2_{Adjusted} = \left[\frac{(1-R^2)(n-1)}{n-k-1}\right]$$

Where,

- n is the number of data points.
- K is the number of independent variables.

Example
Model Building

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```
#importing dataset
df = pd.read_csv("med-insurance.csv")
```

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

Out[3]:

	age	sex	bmi	children	smoker	region	expenses
0	19	female	27.9	0	yes	southwest	16884.92
1	18	male	33.8	1	no	southeast	1725.55
2	28	male	33.0	3	no	southeast	4449.46
3	33	male	22.7	0	no	northwest	21984.47
4	32	male	28.9	0	no	northwest	3866.86

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   expenses  1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

In [6]:
```
df.describe()
```

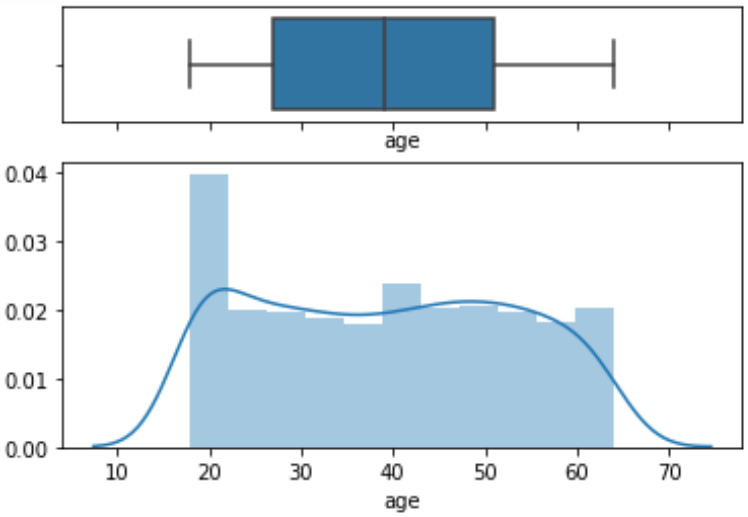
Out[6]:

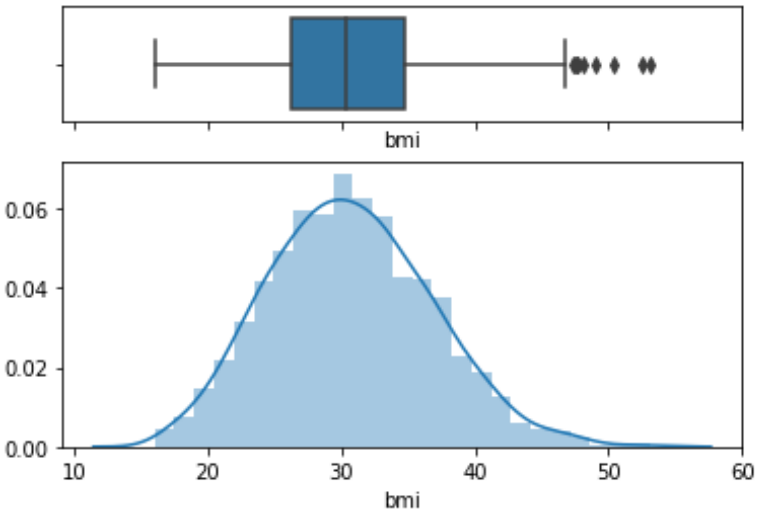
	age	bmi	children	expenses
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.665471	1.094918	13270.422414
std	14.049960	6.098382	1.205493	12110.011240
min	18.000000	16.000000	0.000000	1121.870000
25%	27.000000	26.300000	0.000000	4740.287500
50%	39.000000	30.400000	1.000000	9382.030000
75%	51.000000	34.700000	2.000000	16639.915000
max	64.000000	53.100000	5.000000	63770.430000

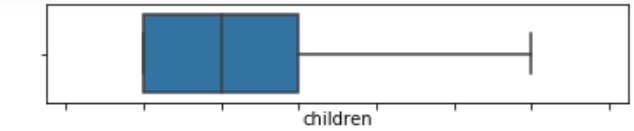
Performing EDA

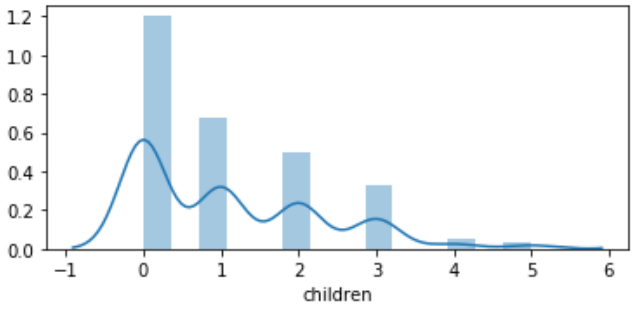
In [7]:

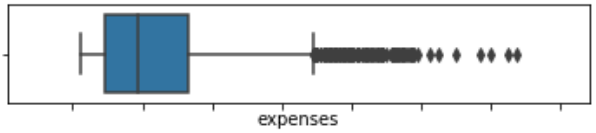
```
#for continuous variables
f,(ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratios": (0.2 , 0.5)})
sns.boxplot(df['age'], ax=ax_box)
sns.distplot(df['age'], ax=ax_hist)
plt.show()
f,(ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratios": (0.2 , 0.5)})
sns.boxplot(df['bmi'], ax=ax_box)
sns.distplot(df['bmi'], ax=ax_hist)
plt.show()
f,(ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratios": (0.2 , 0.5)})
sns.boxplot(df['children'], ax=ax_box)
sns.distplot(df['children'], ax=ax_hist)
plt.show()
f,(ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"height_ratios": (0.2 , 0.5)})
sns.boxplot(df['expenses'], ax=ax_box)
sns.distplot(df['expenses'], ax=ax_hist)
plt.show()
```

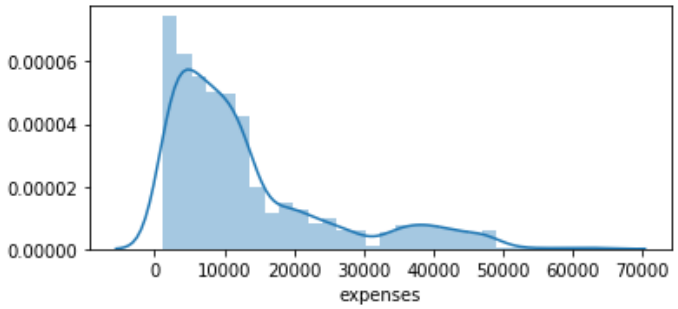








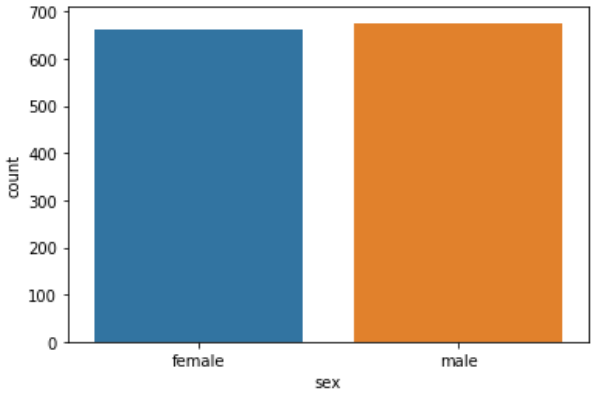


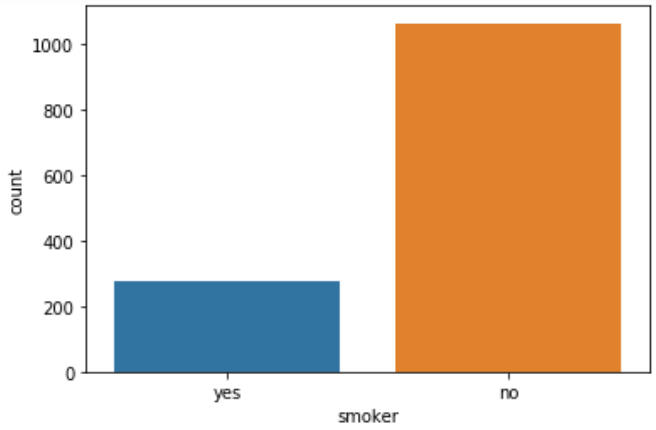


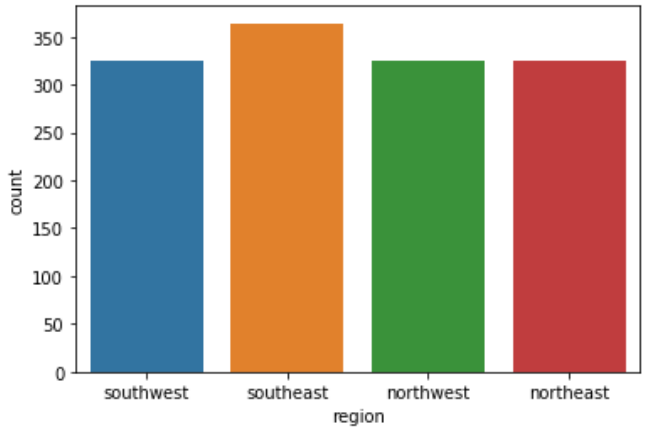

```
Observations
#Age data has a uniform distribution.
#bmi has a normal distribution with few outliers.
#expenses has an unequal distribution with more outliers.
```

In [9]:

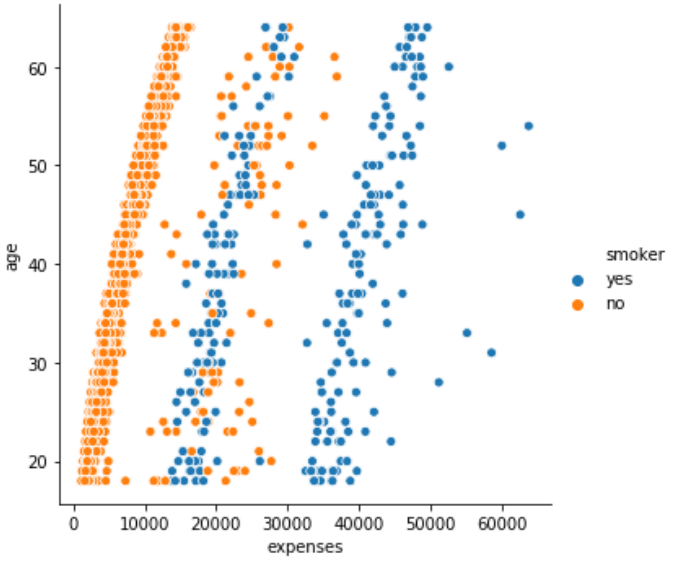
```
#for discrete variables
sns.countplot('sex', data=df)
plt.show()
sns.countplot('smoker',data=df)
plt.show()
sns.countplot('region',data=df)
plt.show()
```





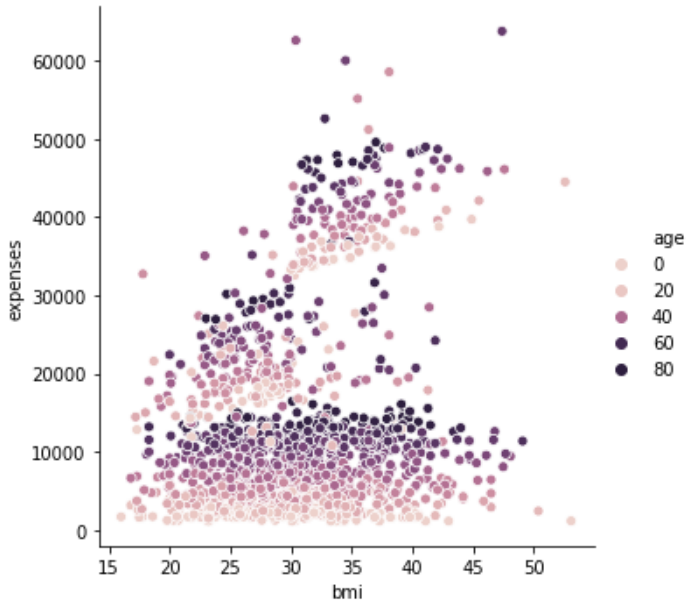


In [10]:
```
1 sns.relplot(x='expenses', y='age', data=df, hue='smoker')
2 plt.show()
```

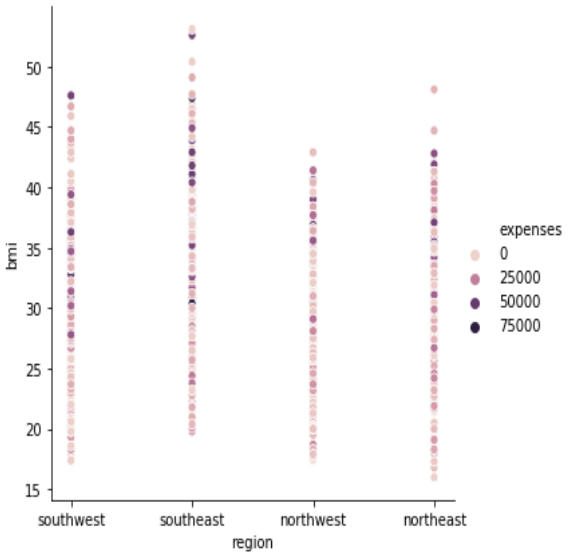



```
1 #here we observed among all age groups smokers are paying more medical expenses
```

In [12]:
```
sns.relplot(x='bmi', y='expenses', data=df, hue='age')
plt.show()
```



In [13]:
```
sns.relplot(x='region', y='bmi', data=df, hue='expenses')
plt.show()
```




```
#Here we have observed that southeast region people are having more BMI among all regions.
#And northwest people are having less BMI compared to all regions.
```

In [18]:
```
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
#Using LabelEncoder for 2 unique values
for col in ['sex','smoker']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
```

In [19]:
```
df.head()
```

Out[19]:

	age	sex	bmi	children	smoker	region	expenses
0	19	0	27.9	0	1	southwest	16884.92
1	18	1	33.8	1	0	southeast	1725.55
2	28	1	33.0	3	0	southeast	4449.46
3	33	1	22.7	0	0	northwest	21984.47
4	32	1	28.9	0	0	northwest	3866.86

In [21]:
```
dum = pd.get_dummies(df['region'], drop_first=True)
dum.head()
```

Out[21]:

	northwest	southeast	southwest
0	0	0	1
1	0	1	0
2	0	1	0
3	1	0	0
4	1	0	0

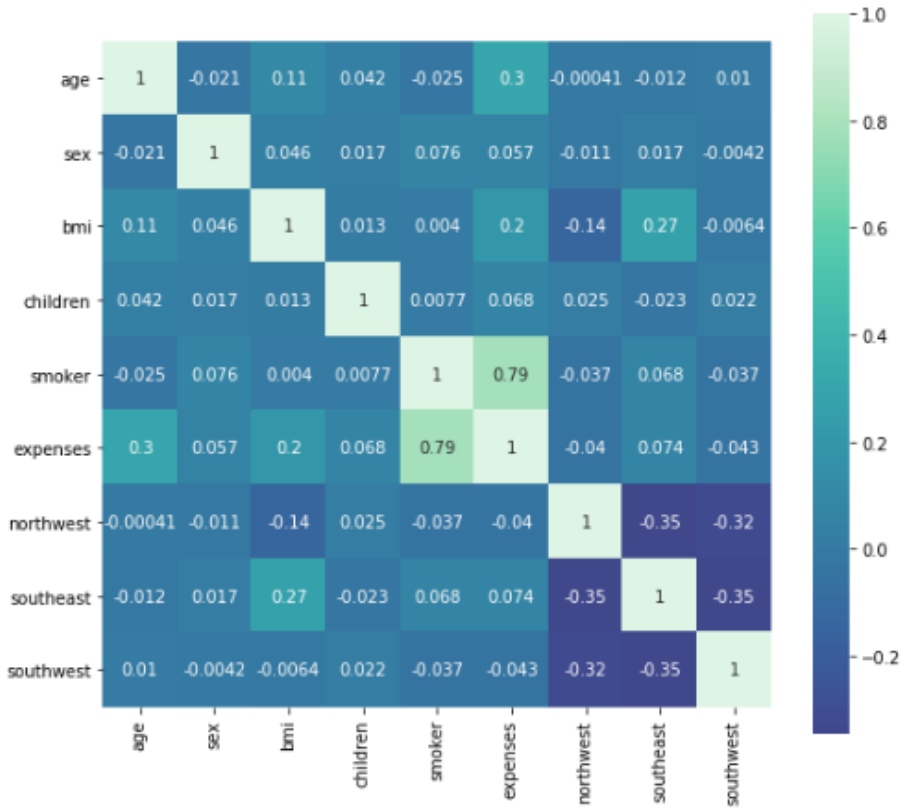
In [22]:
```
df = pd.concat([df,dum], axis=1)
df.head()
```

Out[22]:

	age	sex	bmi	children	smoker	region	expenses	northwest	southeast	southwest
0	19	0	27.9	0	1	southwest	16884.92	0	0	1
1	18	1	33.8	1	0	southeast	1725.55	0	1	0
2	28	1	33.0	3	0	southeast	4449.46	0	1	0
3	33	1	22.7	0	0	northwest	21984.47	1	0	0
4	32	1	28.9	0	0	northwest	3866.86	1	0	0

In [24]:
```
plt.figure(figsize=(9,8))
sns.heatmap(df.corr(), square=True, annot=True, cmap='mako', center = 0)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1fc87b4ae88>




```
#Here we can observe there is direct correlation between smoker and expenses i.e; almost 0.8.
#so, we can tell that smoker has to pay more medical expenses.
```

In [30]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

Linear Regression

In [31]:
```
#Modelling
#LinearRegression
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)
```

Out[31]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [32]:
```
model.intercept_
```

Out[32]: -12376.785237284665

In [33]:
```
model.coef_
```

Out[33]:
```
array([  261.28251281,   104.99524716,   348.96600937,   424.41067944,
       23627.8945956 ,  -486.46995207,  -970.61815579,  -925.06307896])
```

In [34]:
```
train_predictions = model.predict(x_train)
```

In [35]:
```
test_predictions = model.predict(x_test)
```

In [36]:
```
#evaluation metrics
from sklearn.metrics import mean_squared_error
test_RMSE = np.sqrt(mean_squared_error(y_test, test_predictions))
train_RMSE = np.sqrt(mean_squared_error(y_train, train_predictions))
print(train_RMSE, test_RMSE)
```

```
6142.373139201786 5811.806354382952
```

In [37]:
```
model.score(x_train, y_train) #train R2
```

Out[37]:
```
0.7424103110139746
```

In [38]:
```
model.score(x_test, y_test)  #test r2
```

Out[38]:
```
0.7696351080608885
```

In [39]:
```
#checking cross vadidation score
```

In [40]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model,x,y,cv=5)
print(scores)
scores.mean()
```

```
[0.76148215 0.70650918 0.7780752  0.73273236 0.75559751]
```

Out[40]:
```
0.746879280539993
```

```
#here test accuracy and cross vadidation are nearly same.
```

Polynomial Regression

In [38]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
polynomial_converter = PolynomialFeatures(degree=2, include_bias=False)
```

In [39]:
```
X_poly = polynomial_converter.fit_transform(X)
```

In [40]:
```
X_poly.shape
```

Out[40]:
```
(1338, 44)
```

In [41]:
```
#train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X_poly, Y, test_size=0.3, random_state=30)
```

In [42]:
```
model = LinearRegression()
model.fit(X_Train, Y_Train)
```

Out[42]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [43]:
```
train_predictions = model.predict(X_Train)
```

In [44]:
```
test_predictions = model.predict(X_Test)
```

In [45]:
```
train_res = Y_Train - train_predictions
test_res = Y_Test - test_predictions
```

In [58]:
```
#train R2
model.score(X_train, y_train)
```

Out[58]:
```
0.8546283366451466
```

In [59]:
```
#test R2
model.score(X_test, y_test)
```

Out[59]:
```
0.8255318580804683
```

In [60]:
```
Scores = cross_val_score(model,x_poly,y,cv=5)
Scores
#average of MSE scores
abs(Scores.mean())
```

Out[60]:
```
0.8354827706855652
```

In [61]:
```
from sklearn.metrics import mean_absolute_error
```

In [62]:
```
MAE = mean_absolute_error(y_test, test_predictions)
MAE
```

Out[62]:
```
2964.6497128464352
```

In [63]:
```
MSE = mean_squared_error(y_test, test_predictions)
MSE
```

Out[63]:
```
25285383.23334269
```

In [64]:
```
RMSE = np.sqrt(MSE)
RMSE
```

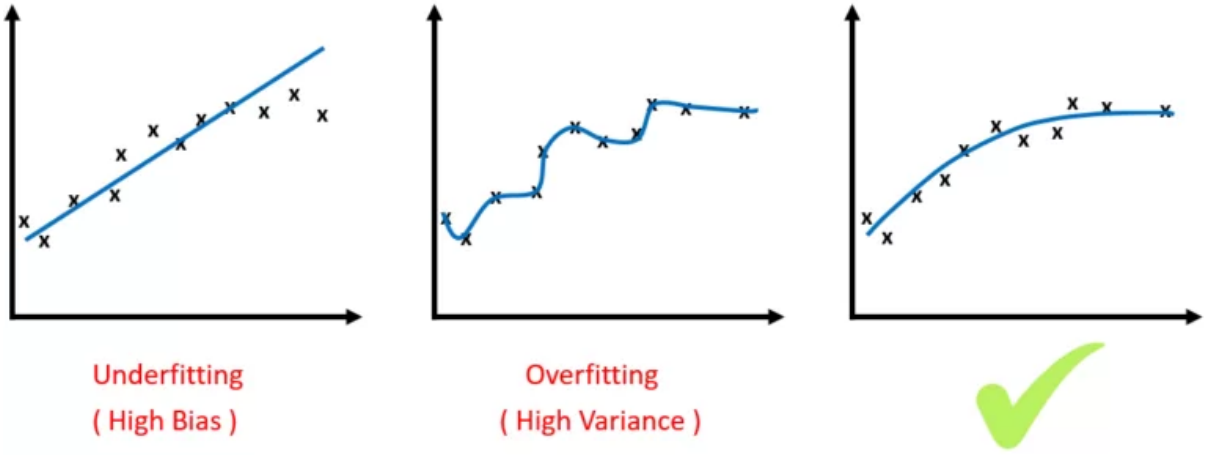
Out[64]:
```
5028.457341306844
```

References for above topics:

- https://www.javatpoint.com/machine-learning
- https://www.knowledgehut.com/blog/data-science/linear-regression-for-machine-learning
- https://www.analyticsvidhya.com/blog/2021/10/understanding-polynomial-regression-model/
- https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/

Understanding overfitting and underfitting:

- To train our machine learning model, we provide it with data to learn from. The process of plotting a series of data points and drawing a line of best fit to understand the relationship between variables is called Data Fitting. Our model is best suited when it can find all the necessary patterns in our data and avoid random data points, and unnecessary patterns called noise.
- If we allow our machine learning model to look at the data too many times, it will find many patterns in our data, including some that are unnecessary. It will learn well on the test dataset and fits very well. It will learn important patterns, but it will also learn from the noise in our data and will not be able to make predictions on other data sets.
- A scenario where a machine learning model tries to learn from the details along with the noise in the data and tries to fit each data point to a curve is called Overfitting.
- In the figure below, we can see that the model is fit for every point in our data. If new data is provided, the model curves may not match the patterns in the new data, and the model may not predict very well.

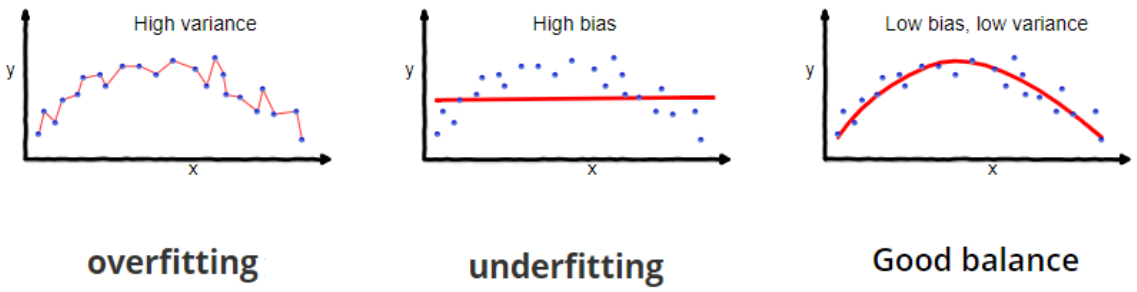


- Conversely, in the scenario where the model has not been allowed to look at our data enough times, the model will not be able to find patterns in our test data set. It won't fit our test data set properly and won't work on new data either.
- A scenario where a machine learning model can neither learn the relationship between variables in the test data nor predict or classify a new data point is called Underfitting.
- The image above shows an under equipped model. We can see that it doesn't fit the data given correctly. He did not find patterns in the data and

ignored much of the data set. It cannot work with both known and unknown data.

Bias and variance:

- Bias refers to the errors which occur when we try to fit a statistical model on real-world data which does not fit perfectly well on some mathematical model. If we use a way too simplistic a model to fit the data then we are more likely to face the situation of High Bias which refers to the case when the model is unable to learn the patterns in the data at hand and hence performs poorly.
- Variance implies the error value that occurs when we try to make predictions by using data that is not previously seen by the model. There is a situation known as high variance that occurs when the model learns noise that is present in the data.



Regularization:

When training a machine learning model, the model can be easily overfitted or under fitted. To avoid this, we use regularization in machine learning to properly fit the model to our test set. Regularization techniques help reduce the possibility of overfitting and help us obtain an optimal model. It refers to techniques used to calibrate machine learning models to minimize the adjusted loss function and avoid overfitting or underfitting.

Ridge regression:

A regression model that uses the L2 regularization technique is called Ridge regression. Ridge regression adds the “squared magnitude” of the coefficient as a penalty term to the loss function(L).

Equation: $\text{loss}=\frac{1}{N}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\lambda\sum_{i=1}^{m}w_i^2$

λ is the penalization factor. We can control the values of λ. When w_i^2 value is high the mean square value also increases. So by adding the penalty it controls the w value and doesn't allow going too high.

Lasso regression:

A regression model which uses the L1 Regularization technique is called LASSO(Least Absolute Shrinkage and Selection Operator) regression. Lasso Regression adds the "absolute value of magnitude" of the coefficient as a penalty term to the loss function(L). Lasso regression also helps us achieve feature selection by penalizing the weights to approximately equal to zero if that feature does not serve any purpose in the model.

Equation: $\text{loss}=\frac{1}{N}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\lambda\sum_{i=1}^{m}|w_i|$

The difference between Ridge and Lasso regularization is in Ridge w_i^2 is used and in Lasso $|w_i|$ absolute value is used.

Elastic Net regression:

This model is a combination of L1 as well as L2 regularization. That implies that we add the absolute norm of the weights as well as the squared measure of the weights.

Equation: $\text{loss}=\frac{1}{N}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\lambda((1-\alpha)\sum_{i=1}^{m}|w_i|+\alpha\sum_{i=1}^{m}w_i^2)$

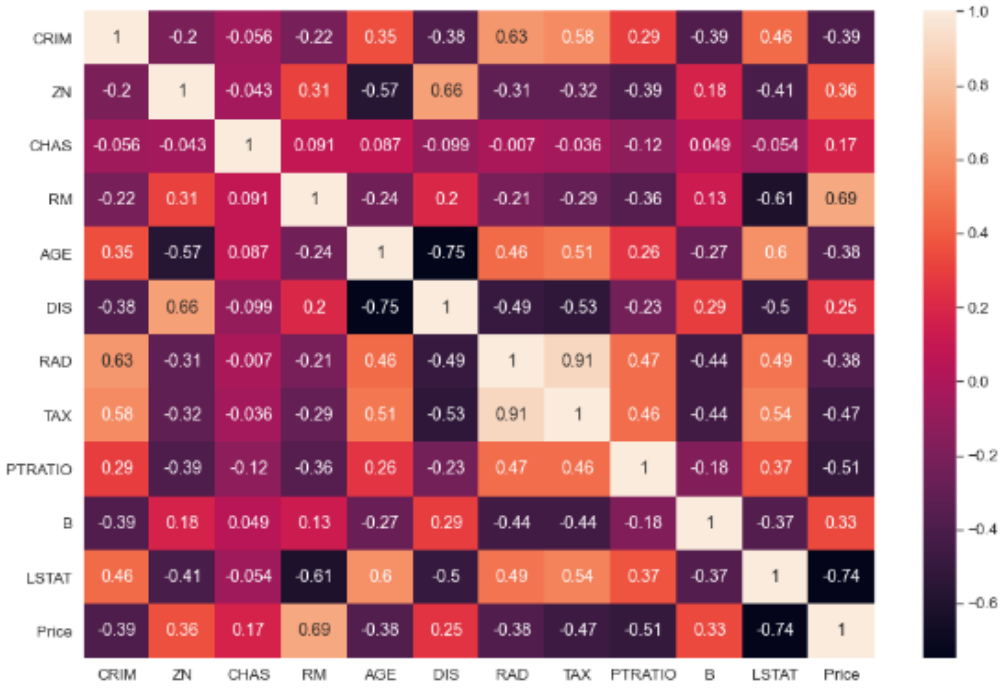
Example

Model Building

For this implementation, we will use the Boston housing dataset found in Sklearn. What we intend to see is:

```
#libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston
#data
```

```
boston = load_boston()
boston_df=pd.DataFrame(boston.data,columns=boston.feature_name
s)
#target variable
boston_df['Price']=boston.target
#Exploration
plt.figure(figsize = (10, 10))
sns.heatmap(boston_df.corr(), annot = True)
```




```
#There are cases of multicolinearity, we will drop a few
columns
boston_df.drop(columns = ["INDUS", "NOX"], inplace = True)
#we will log the LSTAT Column
boston_df.LSTAT = np.log(boston_df.LSTAT)
```

Note that we logged the LSTAT column as it doesn't have a linear relationship with the price column. Linear models assume a linear relationship between x and y variables.

Data Splitting and Scaling

```
#preview
features = boston_df.columns[0:11]
target = boston_df.columns[-1]
#X and y values
X = boston_df[features].values
y = boston_df[target].values
```

```
#splot
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=17)
print("The dimension of X_train is {}".format(X_train.shape))
print("The dimension of X_test is {}".format(X_test.shape))
#Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Output: The dimension of x_train is(354,11)
The dimension of x_test is(152,11)

Lasso Regression

Selecting Optimal Alpha Values Using Cross-Validation in Sklearn

We may need to try out different alpha values to find the optimal constraint value. For this case, we can use the cross-validation model in the sklearn package. This will try out different combinations of alpha values and then choose the best model.

```
#Using the linear CV model
from sklearn.linear_model import LassoCV
#Lasso Cross validation
lasso_cv = LassoCV(alphas = [0.0001, 0.001,0.01, 0.1, 1, 10], random_state=0).fit(X_train, y_train)

#score
print(lasso_cv.score(X_train, y_train))
print(lasso_cv.score(X_test, y_test))
```

Output: The train score for lasso model is 0.78591
The test score for lasso model is 0.76723

Ridge Regression

```
#Ridge Regression Model
ridgeReg = Ridge(alpha=10)
ridgeReg.fit(X_train,y_train)
#train and test score for ridge regression
train_score_ridge = ridgeReg.score(X_train, y_train)
test_score_ridge = ridgeReg.score(X_test, y_test)
print("The train score for ridge model is {}".format(train_score_ridge))
print("The test score for ridge model is {}".format(test_score_ridge))
```

Output: The train score ridge model is 0.78442

```
The test score for ridge model is 0.76967
```

Using an alpha value of 10, the evaluation of the model, the train, and test data indicate better performance on the ridge model.

References for above topics:

- https://www.geeksforgeeks.org/regularization-in-machine-learning/
- https://www.javatpoint.com/regularization-in-machine-learning