

Problem Statement

Mini-Project 2 on SVM: Apply the Support vector machine for classification on a dataset obtained from UCI ML repository. For Example: Fruits Classification or Soil Classification or Leaf Disease Classification.

Background

Breast cancer is the most common cancer amongst women in the world. It accounts for 25% of all cancer cases, and affected over 2.1 Million people in 2015 alone. It starts when cells in the breast begin to grow out of control. These cells usually form tumors that can be seen via X-ray or felt as lumps in the breast area.

Early diagnosis significantly increases the chances of survival. The key challenges against it's detection is how to classify tumors into malignant (cancerous) or benign(non cancerous). A tumor is considered malignant if the cells can grow into surrounding tissues or spread to distant areas of the body. A benign tumor does not invade nearby tissue nor spread to other parts of the body the way cancerous tumors can. But benign tumors can be serious if they press on vital structures such as blood vessels or nerves.

Machine Learning technique can dramatically improve the level of diagnosis in breast cancer. Research shows that experienced physicians can detect cancer by 79% accuracy, while a 91 %(sometimes up to 97%) accuracy can be achieved using Machine Learning techniques.

Dataset

Dataset is obtained from sklearn. It has around 30 features and one target variable.

List of features are

- ❖ mean radius = mean of distances from center to points on the perimeter
- ❖ mean texture = standard deviation of gray-scale values
- ❖ mean perimeter = mean size of the core tumor
- ❖ mean area =
- ❖ mean smoothness = mean of local variation in radius lengths
- ❖ mean compactness = mean of $\text{perimeter}^2 / \text{area} - 1.0$
- ❖ mean concavity = mean of severity of concave portions of the contour
- ❖ mean concave points = mean for number of concave portions of the contour
- ❖ mean symmetry =
- ❖ mean fractal dimension = mean for "coastline approximation" - 1
- ❖ radius error = standard error for the mean of distances from center to points on the perimeter
- ❖ texture error = standard error for standard deviation of gray-scale values

- ❖ perimeter error =
- ❖ area error =
- ❖ smoothness error = standard error for local variation in radius lengths
- ❖ compactness error = $\text{standard error for perimeter}^2 / \text{area} - 1.0$
- ❖ concavity error = standard error for severity of concave portions of the contour
- ❖ concave points error = standard error for number of concave portions of the contour
- ❖ symmetry error =
- ❖ fractal dimension error = standard error for "coastline approximation" - 1
- ❖ worst radius = "worst" or largest mean value for mean of distances from center to points on the perimeter
- ❖ worst texture = "worst" or largest mean value for standard deviation of gray-scale values
- ❖ worst perimeter =
- ❖ worst smoothness = "worst" or largest mean value for local variation in radius lengths
- ❖ worst compactness = "worst" or largest mean value for $\text{perimeter}^2 / \text{area} - 1.0$
- ❖ worst concavity = "worst" or largest mean value for severity of concave portions of the contour

- ❖ worst concave points = "worst" or largest mean value for number of concave portions of the contour
- ❖ worst fractal dimension = "worst" or largest mean value for "coastline approximation" - 1

It has 596 rows (Instances) and 31 columns(Features)

Methodology

There are no categorical values in data so need of label encoding them.

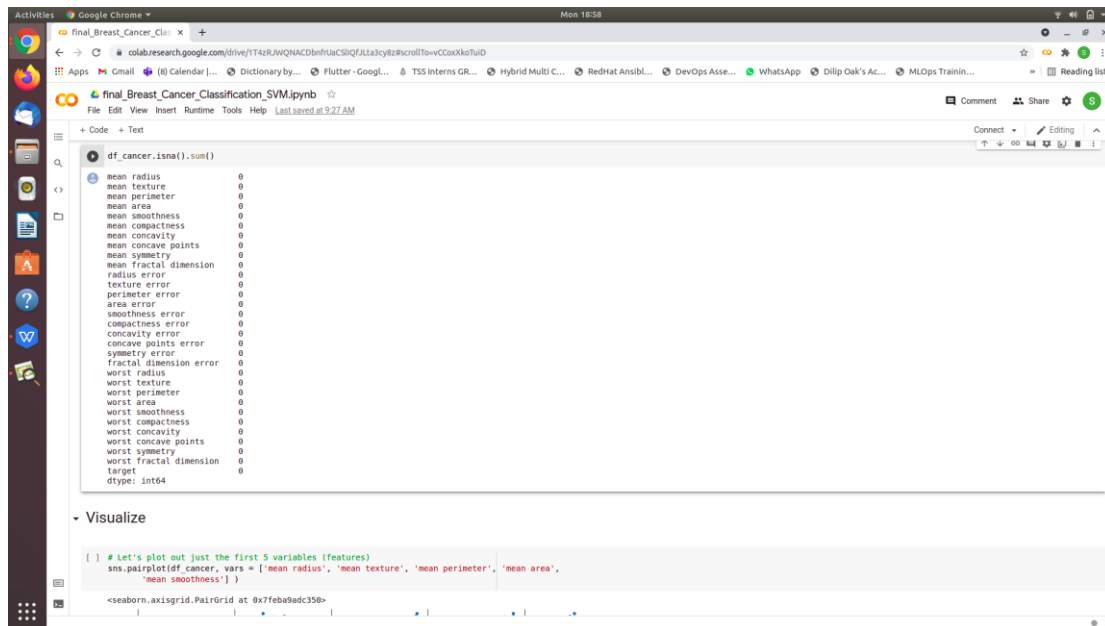
df_cancer.shape

```
(569, 31)
```

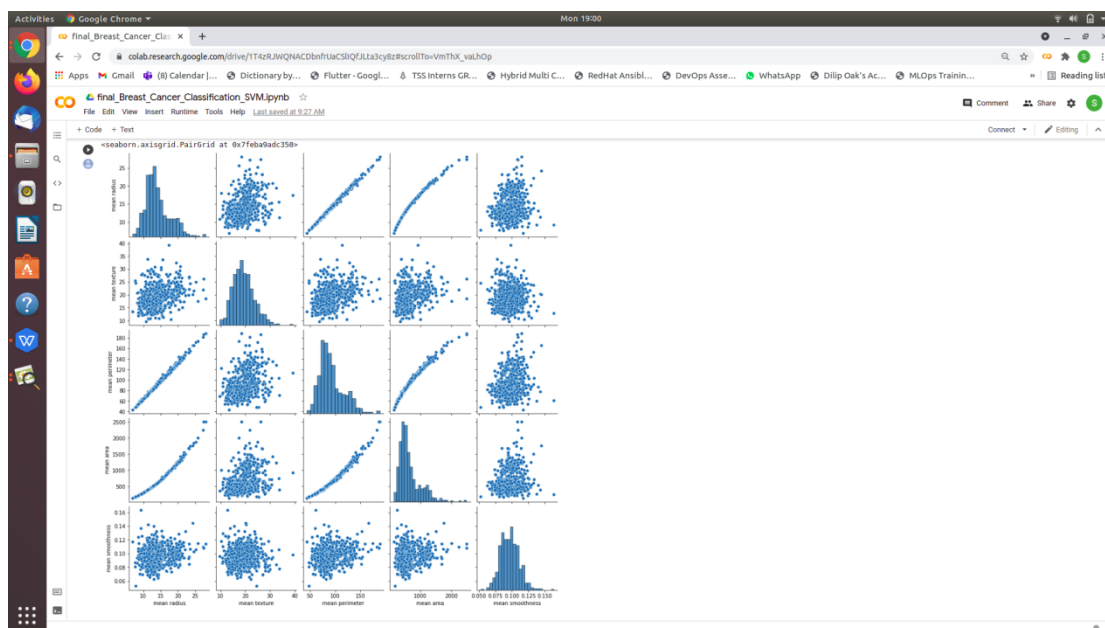
df_cancer.dtypes

Feature	dtype
mean radius	float64
mean texture	float64
mean perimeter	float64
mean area	float64
mean smoothness	float64
mean compactness	float64
mean concavity	float64
mean concave points	float64
mean symmetry	float64
mean fractal dimension	float64
radius error	float64
texture error	float64
perimeter error	float64
area error	float64
smoothness error	float64
compactness error	float64
concavity error	float64
concave points error	float64
symmetry error	float64
fractal dimension error	float64
worst radius	float64
worst texture	float64
worst perimeter	float64
worst area	float64
worst smoothness	float64
worst compactness	float64
worst concavity	float64
worst concave points	float64
worst symmetry	float64
worst fractal dimension	float64
target	object

There are no null values so data cleaning step is not required.

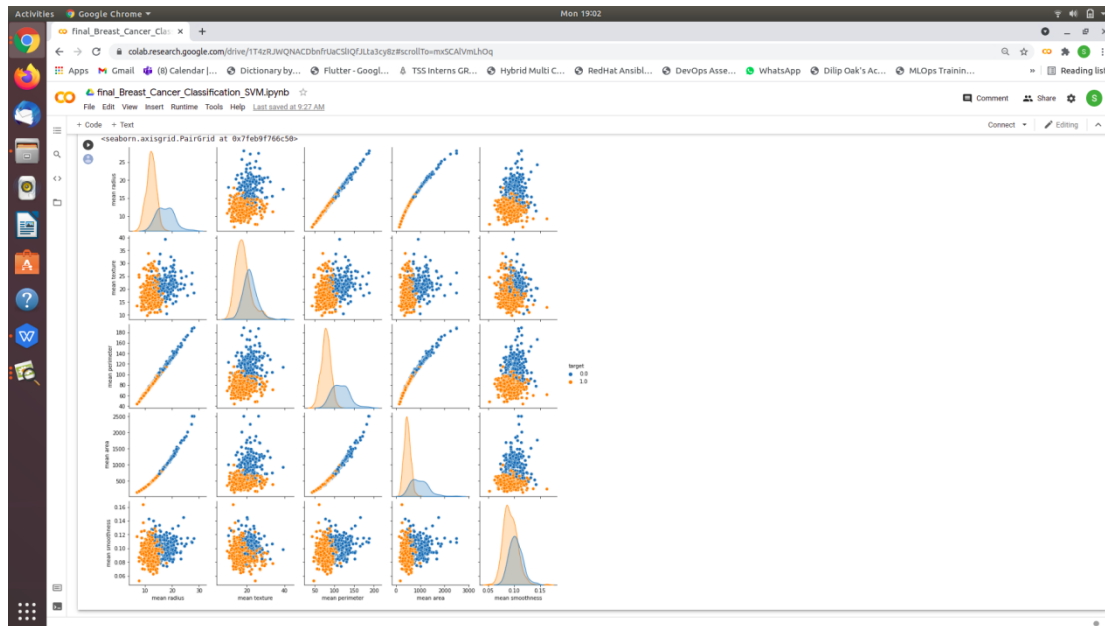


Data Visualization



The above plots show the relationship between our features. But the only problem with them is that they do not show us which of the "dots" is Malignant and which is Benign.

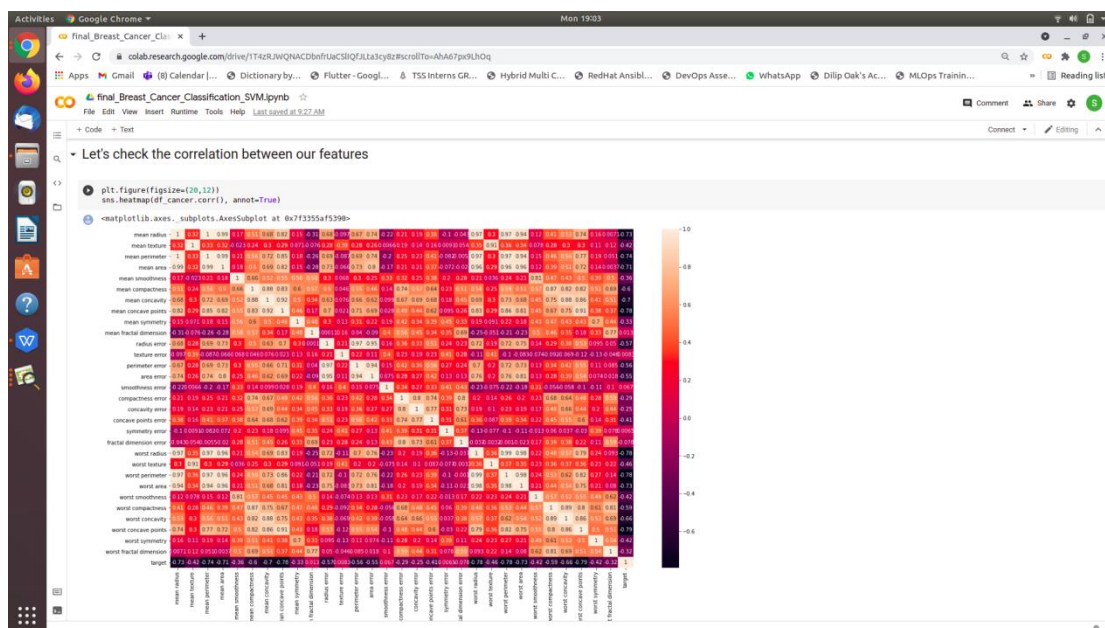
This issue will be addressed below by using "target" variable as the "hue" for the plots.



1.0 (Orange) = Benign (No Cancer)

0.0 (Blue) = Malignant (Cancer)

Co-relation using HeatMap



There is a strong correlation between the mean radius and mean perimeter, mean area and mean perimeter. So it's better to drop column mean perimeter to avoid overfitting.

Normalisation

Why Normalize?

Many machine learning algorithms attempt to find trends in the data by comparing features of data points. However, there is an issue when the features are on drastically different scales.

For example, consider a dataset of houses. Two potential features might be the number of rooms in the house, and the total age of the house in years. A machine learning algorithm could try to predict which house would be best for you. However, when the algorithm compares data points, the feature with the larger scale will completely dominate the other.

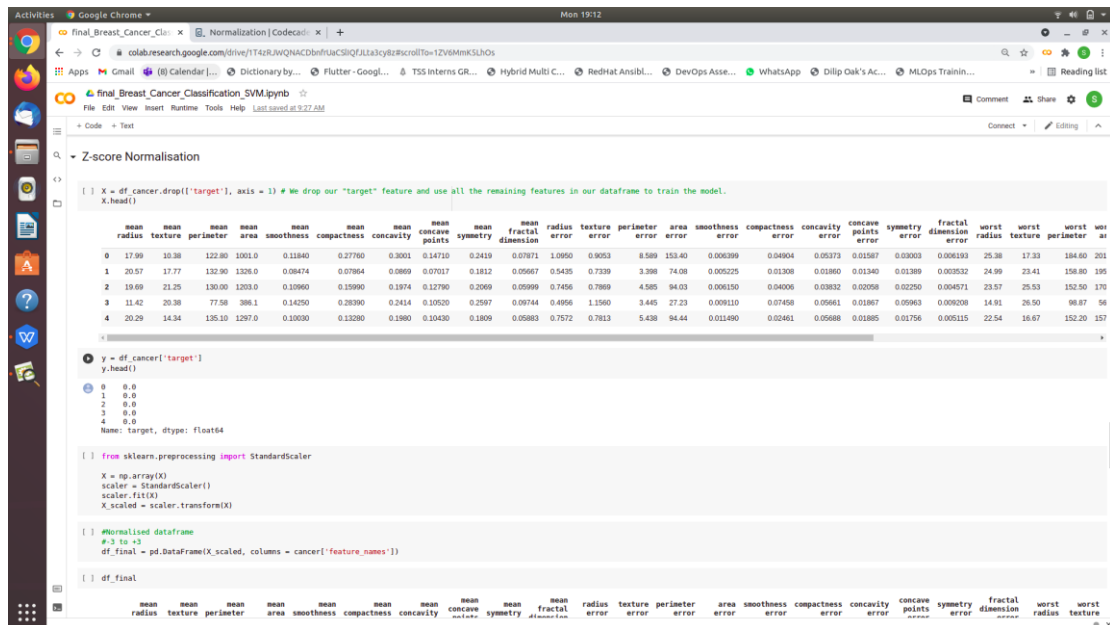
Z-Score Normalization

Z-score normalization is a strategy of normalizing data that avoids this outlier issue. The formula for Z-score normalization is below:

$$(\text{value} - \mu) / \sigma$$

Here, μ is the mean value of the feature and σ is the standard deviation of the feature. If a value is exactly equal to the mean of all the values of the feature, it will be normalized to 0. If it is below the mean, it will be a negative number, and if it is above the mean it will be a positive number. The size of those negative and positive numbers is determined by the standard deviation of the original feature. If the unnormalized data had a large standard deviation, the normalized values will be closer to 0.

Before Normalisation



```
[ ] X = df.cancer.drop('target', axis = 1) # We drop our "target" feature and use all the remaining features in our dataframe to train the model.
X.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error	area error	smoothness error	compactness error	concavity error	concave points error	symmetry error	fractal dimension error	worst radius	worst texture	worst perimeter	worst area
0	17.99	10.38	122.80	1001.0	0.11840	0.27790	0.3001	0.14710	0.2419	0.07071	1.0950	0.9053	8.589	153.40	0.006399	0.04804	0.05373	0.01367	0.03003	0.006193	25.38	17.33	184.60	201
1	20.57	17.77	132.90	1326.0	0.106474	0.07964	0.0869	0.07017	0.1812	0.05067	0.5435	0.7339	3.368	74.08	0.005225	0.01308	0.01890	0.01340	0.01389	0.003532	24.99	23.41	158.86	195
2	19.09	21.29	136.00	1203.0	0.10960	0.13974	0.12790	0.2059	0.05999	0.7149	0.7699	4.585	94.03	0.006150	0.04006	0.03832	0.02058	0.02250	0.00471	23.57	25.53	152.50	178	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.15520	0.2597	0.08744	0.4056	1.1560	1.445	27.23	0.009110	0.07458	0.05961	0.01867	0.05963	0.002008	14.91	26.50	96.87	56
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0.7572	0.7813	5.438	94.44	0.012490	0.02461	0.05688	0.01885	0.01756	0.005115	22.54	16.67	152.20	157

```
[ ] y = df.cancer['target']
y.head()
```

```
[ ] from sklearn.preprocessing import StandardScaler
X = np.array(X)
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

```
[ ] #Normalised dataframe
# 3 to 3
df_final = pd.DataFrame(X_scaled, columns = cancer['feature_names'])
```

```
[ ] df_final
```

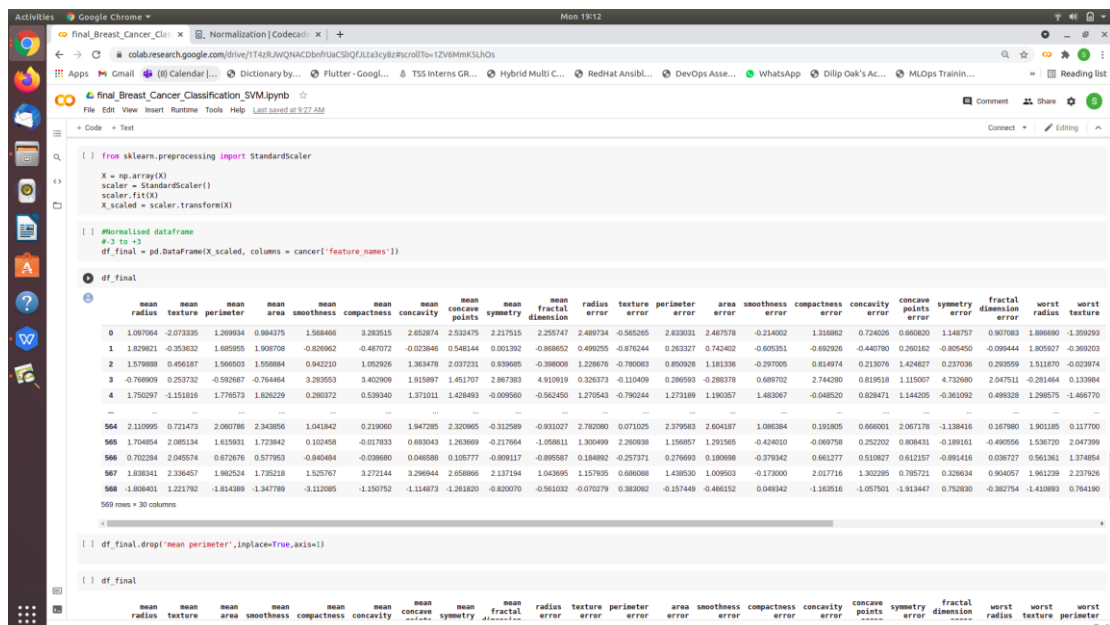
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error	area error	smoothness error	compactness error	concavity error	concave points error	symmetry error	fractal dimension error	worst radius	worst texture	worst perimeter	worst area
0	0.007064	-2.07335	1.26934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	2.489734	-0.565285	2.833031	2.487578	-0.214002	1.316862	0.724026	0.660820	1.148757	0.907083	1.866690	-1.359293		
1	1.829821	-0.353632	1.60955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	-0.986652	0.499255	-0.876244	0.263327	0.742402	-0.605351	-0.692926	-0.440780	0.260162	0.805450	-0.095444	1.805927	-0.365203		
2	1.579888	0.456187	1.560503	1.508884	0.942210	1.052926	1.363478	2.037231	0.939605	-0.988008	1.228676	-0.780083	0.850928	1.181336	-0.297005	0.814974	0.213076	1.424827	0.237036	0.293559	1.511870	-0.023974		
3	-0.769809	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	4.910919	0.326373	-0.110409	0.286593	-0.288378	0.689702	2.744280	0.819518	1.115007	4.732680	2.047511	-0.281464	0.133984		
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	-0.562450	1.270543	-0.790244	1.271189	1.190357	1.483067	-0.048520	0.828471	1.144205	-0.361092	0.499328	1.298575	-1.468770		

```
[ ] df_final
```

```
[ ] df_final.drop('mean perimeter', inplace=True, axis=1)
```

```
[ ] df_final
```

After Normalisation



```
[ ] from sklearn.preprocessing import StandardScaler
X = np.array(X)
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

```
[ ] #Normalised dataframe
# 3 to 3
df_final = pd.DataFrame(X_scaled, columns = cancer['feature_names'])
```

```
[ ] df_final
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error	area error	smoothness error	compactness error	concavity error	concave points error	symmetry error	fractal dimension error	worst radius	worst texture	worst perimeter	worst area
0	1.007064	-2.07335	1.26934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	2.489734	-0.565285	2.833031	2.487578	-0.214002	1.316862	0.724026	0.660820	1.148757	0.907083	1.866690	-1.359293		
1	1.829821	-0.353632	1.60955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	-0.986652	0.499255	-0.876244	0.263327	0.742402	-0.605351	-0.692926	-0.440780	0.260162	0.805450	-0.095444	1.805927	-0.365203		
2	1.579888	0.456187	1.560503	1.508884	0.942210	1.052926	1.363478	2.037231	0.939605	-0.988008	1.228676	-0.780083	0.850928	1.181336	-0.297005	0.814974	0.213076	1.424827	0.237036	0.293559	1.511870	-0.023974		
3	-0.769809	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	4.910919	0.326373	-0.110409	0.286593	-0.288378	0.689702	2.744280	0.819518	1.115007	4.732680	2.047511	-0.281464	0.133984		
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	-0.562450	1.270543	-0.790244	1.271189	1.190357	1.483067	-0.048520	0.828471	1.144205	-0.361092	0.499328	1.298575	-1.468770		

```
[ ] df_final
```

```
[ ] df_final.drop('mean perimeter', inplace=True, axis=1)
```

```
[ ] df_final
```

What do we mean when we say “Modeling” ?

Depending on how long we've lived in a particular place and traveled to a location, we probably have a good understanding of commute times in our area. For example, we've traveled to work/school using some combination of the metro, buses, trains, ubers, taxis, carpools, walking, biking, etc.

All humans naturally model the world around them.

Over time, our observations about transportation have built up a mental dataset and a mental model that helps us predict what traffic will be like at various times and locations. We probably use this mental model to help plan our days, predict arrival times, and many other tasks.

As data scientists we attempt to make our understanding of relationships between different quantities more precise through using data and mathematical/statistical structures.

This process is called modeling.

Models are simplifications of reality that help us to better understand that which we observe.

In a data science setting, models generally consist of an independent variable (or output) of interest and one or more dependent variables (or inputs) believed to influence the independent variable.

Model-based inference

We can use models to conduct inference.

Given a model, we can better understand relationships between an independent variable and the dependent variable or between multiple independent variables.

An example of where inference from a mental model would be valuable is:

Determining what times of the day we work best or get tired.

Prediction

We can use a model to make predictions, or to estimate a dependent variable's value given at least one independent variable's value.

Predictions can be valuable even if they are not exactly right.

Good predictions are extremely valuable for a wide variety of purposes.

An example of where prediction from a mental model could be valuable:

Predicting how long it will take to get from point A to point B.

What is the difference between model prediction and inference?

Inference is judging what the relationship, if any, there is between the data and the output.

Prediction is making guesses about future scenarios based on data and a model constructed on that data.

In this project, we will be talking about a Machine Learning Model called Support Vector Machine (SVM)

Introduction to Classification Modeling:
Support Vector Machine (SVM)

What is a Support Vector Machine (SVM)?

A Support Vector Machine (SVM) is a binary linear classification whose decision boundary is explicitly constructed to minimize generalization error. It is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression and even outlier detection.

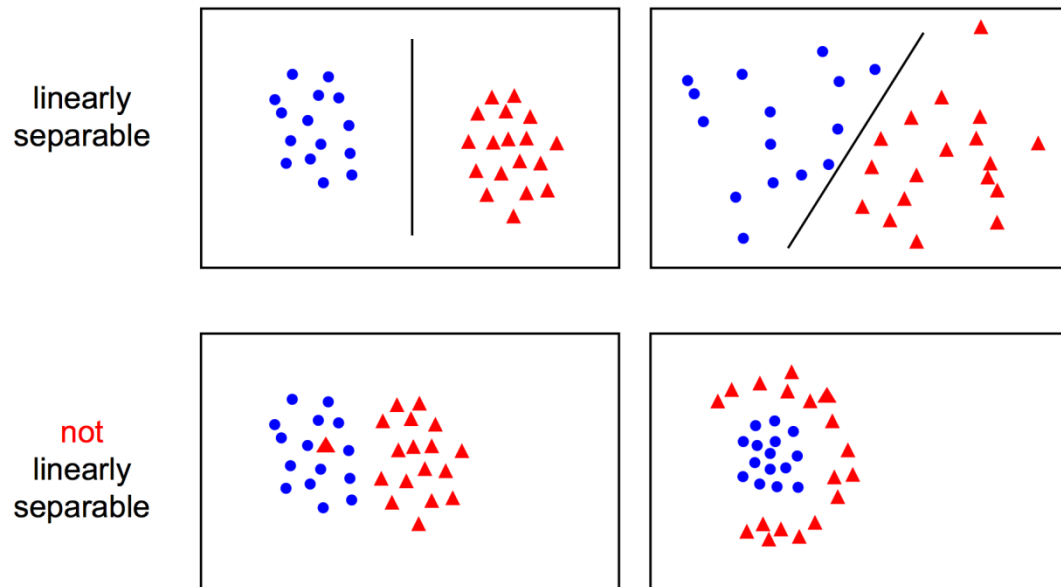
SVM is well suited for classification of complex but small or medium sized datasets.

How does SVM classify?

It's important to start with the intuition for SVM with the **special linearly separable** classification case.

If classification of observations is **“linearly separable”**, SVM fits the **“decision boundary”** that is defined by the

largest margin between the closest points for each class. This is commonly called the “maximum margin hyperplane (MMH)” .



The advantages of support vector machines are:

- ❖ Effective in high dimensional spaces.
- ❖ Still effective in cases where number of dimensions is greater than the number of samples.
- ❖ Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- ❖ Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- ❖ If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- ❖ SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and Probabilities)

Model Training

From our dataset, let's create the target and predictor matrix

- “y” = Is the feature we are trying to predict (Output). In this case we are trying to predict if our “target” is cancerous (Malignant) or not (Benign). i.e. we are going to use the “target” feature here.
- “X” = The predictors which are the remaining columns (mean radius, mean texture, mean perimeter, mean area, mean smoothness, etc.)

Create the training and testing data

Now that we've assigned values to our “X” and “y”, the next step is to import the python library that will help us split our dataset into training and testing data.

- ❖ Training data = the subset of our data used to train our model.
- ❖ Testing data = the subset of our data that the model hasn't seen before (We will be using this dataset to test the performance of our model).

Import Support Vector Machine (SVM) Model.

Train SVM model with Training dataset.

Make a prediction using testing data on trained model.

Result

Finally we obtain all the performance matrices such as Confusion, matrix, accuracy score, precision. recall.

The confusion matrix

The confusion matrix is a table representing the performance of your model to classify labels correctly.

A confusion matrix for a binary classification task:

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual	False	True Positive

	Predicted Negative	Predicted Positive
Positive	Negative (FN)	(TP)

In a binary classifier, the "true" class is typically labeled with 1 and the "false" class is labeled with 0.

True Positive: A positive class observation (1) is correctly classified as positive by the model.

False Positive: A negative class observation (0) is incorrectly classified as positive.

True Negative: A negative class observation is correctly classified as negative.

False Negative: A positive class observation is incorrectly classified as negative.

Columns of the confusion matrix sum to the predictions by class. Rows of the matrix sum to the actual values within each class. You may encounter confusion matrices where the actual is in columns and the predicted is in the rows: the meaning is the same but the table will be reoriented.

Note: Remembering what the cells in the confusion matrix represents can be a little tricky. The first word (True or False) indicates whether or not the model was correct. The second word (Positive or Negative) indicates the *model's guess* (not the actual label!).

Accuracy – Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same.

Therefore, you have to look at other parameters to evaluate the performance of your model.

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate.

$$\text{Precision} = \frac{TP}{TP+FP}$$

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label?

$$\text{Recall} = \frac{TP}{TP+FN}$$

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

$$\text{F1 Score} = \frac{2 * (\text{Recall} * \text{Precision})}{(\text{Recall} + \text{Precision})}$$

In our case

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN) 47	False Positive (FP) 1
Actual Positive	False Negative (FN) 0	True Positive (TP) 66

Accuracy	0.99122
Precision	0.98507
Recall	1.0
F1_score	0.99248

Thus we successfully eliminated type-2 error. (False Negatives)

Conclusion

We were succesful in applying Support vector machine for classification on a dataset and obtain the performance matrix.