MACHINE LEARNING & DATA ANALYSIS
FINAL PROJECT REPORT
On

"COVID-19 INFECTION CLASSIFICATION WITH MACHINE LEARNING"

Submitted By
HEMENDRA JAMPALA (00695281)

Under The Guidance of
PROF. TRAVIS MILLBURN

UNIVERSITY OF NEW HAVEN
Tagliatela College of Engineering
Department of Data Science
300 Boston Post Rd, West Haven, CT 06516

1. PROJECT OBJECTIVE:-

- Coronavirus disease 2019 (COVID-19) is a contagious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2).
- Worlwide Total cases : 67 M / Recovered: 43 M / Deaths: 1.54M.
- Initially prompt and accurate molecular diagnosis of COVID-19 was very challenging.
- Below map visualizes corona impact on world

- The use of modern technology with AI and ML dramatically improves the screening, prediction, contact tracing, forecasting, and drug/vaccine development with extreme reliability.
- We will create a best Model, which helps to predict whether the patient will be Covid-19 Positive or Negative.

2. DATASET DETAILS:-

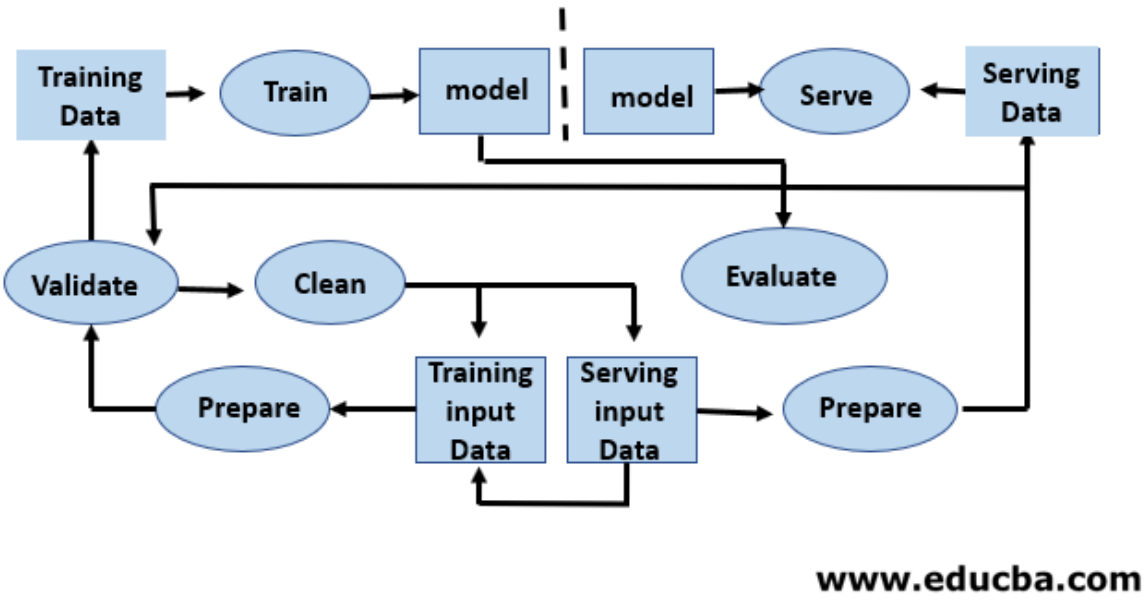
The Dataset is from Kaggle, Dataset was released by the Mexican government.

It has 20K Samples and 23 Features.

Features in the dataset are ‘age', 'patient_type', 'contact_other_covid', 'intubed', 'pneumonia', 'pregnancy', 'diabetes', 'copd', 'asthma', 'inmsupr', 'hypertension', 'other_disease', ‘gender', 'cardiovascular', 'obesity', 'renal_chronic', 'tobacco', 'covid_res', 'icu‘

3. APPROACH:-

The following steps are implemented to build a required supervised machine learning model to predict the covid result.



The steps followed to achieve best ML model as follows

1. Data Loading & Analysis
2. Data Visualizations
3. Data Preprocessing
4. Splitting the Data
5. Model Training
6. Comparing Model Results
7. Conclusion.

3.1 Loading Dataset and Analysis:-

Initially, we need to install and import all the basic necessary libraries like Pandas, Numpy, Scikit-learn, pyplot, Seaborn etc into Jupyter Notebook.

These are the main required libraries for building and training machine learning models. If any other libraries are required in the future, they can be imported accordingly.

The ~23k samples of data in the CSV file are loaded into Pandas dataframe using read_csv() function.

This function returns the data in the CSV file as a two-dimensional data structure with labeled axes, called dataframe as shown below:

```
In [2]: #Loading data into dataframe
        df = pd.read_csv(r'C:\Users\madhuyen\Desktop\covid-desk2.csv')

In [3]: #shape of dataframe
        df.shape

Out[3]: (20352, 23)

In [4]: #print top 3 rows
        df.head(3)
```

Out[4]:

	id	sex	patient_type	entry_date	date_symptoms	date_died	intubed	pneumonia	age	pregnancy	...	inmsupr	hypertension	other_disease	cardiova
0	0b69dc	1	2	29-05-2020	21-05-2020	9999-99-99	2	2	31	1	...	2	2	2	
1	16d202	1	2	6/6/2020	30-05-2020	9999-99-99	2	2	26	1	...	2	1	2	
2	0d32e0	1	2	3/5/2020	28-04-2020	9999-99-99	2	1	18	1	...	2	2	2	

3 rows × 23 columns

Data Analysis:-

```
for col in val_columns:
    print(df[col].value_counts())
```

```
1    20352
Name: sex, dtype: int64
2    18018
1     2334
Name: intubed, dtype: int64
2    14174
1     6178
Name: contact_other_covid, dtype: int64
2    17961
1     2391
Name: icu, dtype: int64
1    11905
2     8447
Name: covid_res, dtype: int64
```

```
[11]: df.describe()
```

[11]:

	sex	patient_type	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension
count	20352.0	20352.0	20352.000000	20352.000000	20352.000000	20352.000000	20352.000000	20352.000000	20352.000000	20352.000000	20352.000000
mean	1.0	2.0	1.885318	1.338886	50.458137	1.975138	1.705582	1.952241	1.965949	1.958382	1.672514
std	0.0	0.0	0.318645	0.473343	20.768518	0.155710	0.455792	0.213262	0.181364	0.199718	0.469308
min	1.0	2.0	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Once we loaded data successfully, we need to perform initial Data Analysis to understand the data to find any statistical observations.

Pandas is mainly used for data analysis, with the help of Pandas methods we perform various statistical tests as shown above.

3.2 DATA VISUALIZATIONS:-

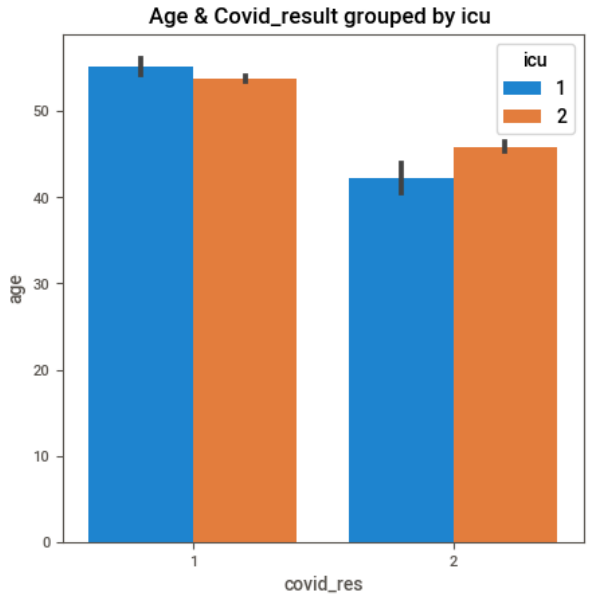
Data visualization gives us a clear idea of what the information means by giving it visual context through maps or graphs. This makes the data more natural for the human mind to comprehend and therefore makes it easier to identify trends, patterns, and outliers within large data sets.

Python has some of most interactive data visualization tools like matplotlib, seaborn with the help of pandas.

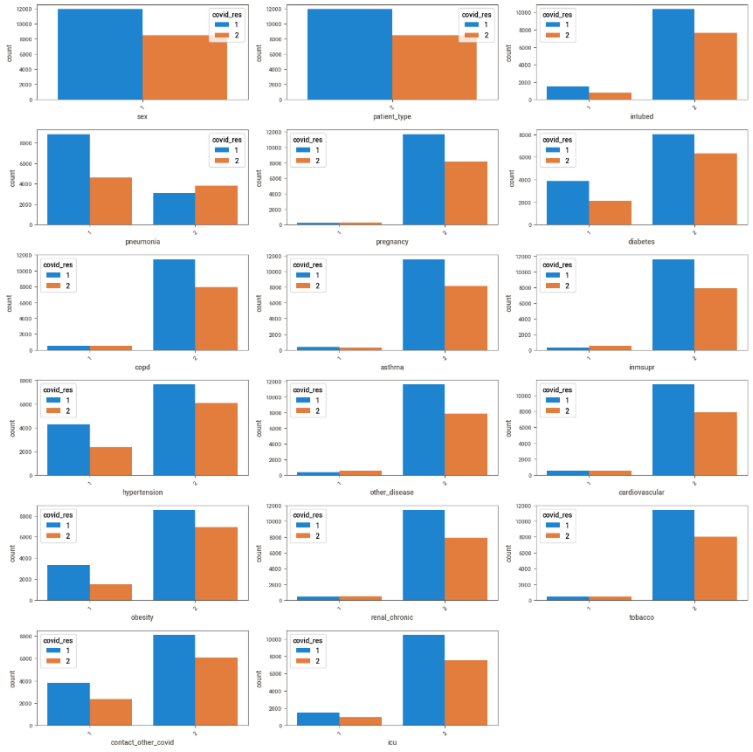
I found few advanced python libraries for Data Visualizations like pandas profiling and sweet viz for visualized data analysis.

```
In [9]: #Data Visualization using Sweet Viz library
        data_report = sv.analyze(df)

        Done! Use 'show' commands to display/save.   [100%] 00:01 -> (00:00 left)

In [10]: #data_Report html will be opened in a new tab
         data_report.show_html("data_Report.html")

         Report data_Report.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, the report IS saved in
         your notebook/colab files.
```

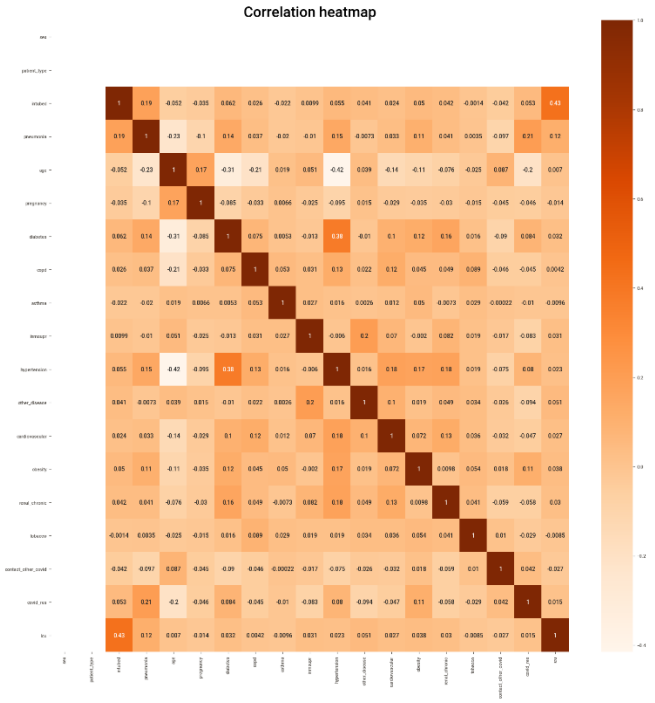
sns.barplot(y="age", x="covid_res", hue="icu", data=df)



Plot all the columns with respect to covid_res, sns.countplot(i[1],hue='covid_res',data=df)



Heatmap to observe the correlation between the features

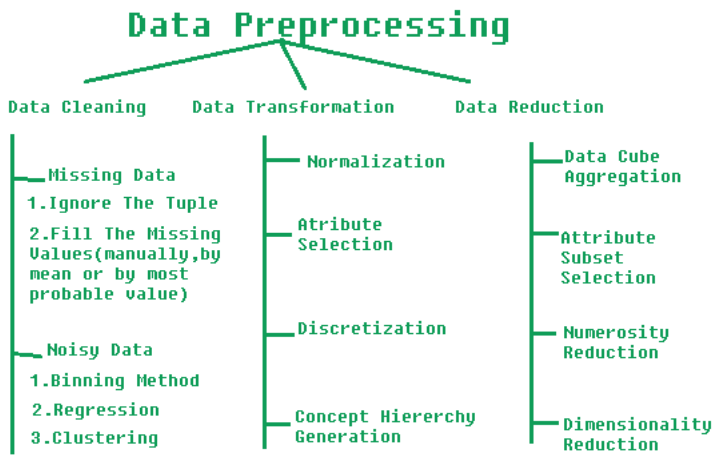


3.3 DATA PREPROCESSING:-

Preprocessing helps to increase the efficacy of our predictions. Data cleaning, transforming and reduction called as data preprocessing.

We cannot fit and evaluate machine learning algorithms on raw data; instead, we must transform the data to meet the requirements of individual machine learning algorithms. More than that, we must choose a representation for the data that best exposes the unknown underlying structure of the prediction problem to the learning algorithms in order to get the best performance given our available resources on a predictive modeling project.

Given that we have standard implementations of highly parameterized machine learning algorithms in open source libraries, fitting models has become routine. As such, the most challenging part of each predictive modeling project is how to prepare the one thing that is unique to the project.



Applied below preprocessing techniques to the data before feeding it into our model.

- Null value treatment – Fill NA with Mean/Median.
- Label Encoding -Handling Categorical Variables
- Feature Scaling – Standard Scalar

from sklearn.preprocessing import RobustScaler scaler = RobustScaler() x_df_scaled = scaler.fit_transform(X)

22]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_df_scaled = scaler.fit_transform(X)
```

In [19]:
```
# Apply our encoding logic even though we don't have any categorical data in df
for column in df.columns:
    if df[column].dtype == type(object):
        le = sklearn.preprocessing.LabelEncoder()
        df[column] = le.fit_transform(df[column])
```

In [18]:
```
#count null values in each column
df.isnull().sum()
```

Out[18]:
```
sex                  0
patient_type         0
intubed              0
pneumonia            0
age                  0
pregnancy            0
diabetes             0
copd                 0
asthma               0
inmsupr              0
hypertension         0
other_disease        0
cardiovascular       0
obesity              0
renal_chronic        0
tobacco              0
contact_other_covid  0
covid_res            0
icu                  0
dtype: int64
```

3.4 Splitting the Data:-

Before getting into data splitting to train and test data, we have to create feature (X) and target (y) variables from the dataset. In the suicide dataset, I considered suicide_rate of country as the target variable and the rest of them are considered as the features. The code snippet of this task is as follows:

4. Splitting the Data

]:
```
X = df.drop(columns=['covid_res'])
y = df[['covid_res']]

X.shape, y.shape
```

]: ((20352, 18), (20352, 1))

After forming the input (X) and target (y) variables, the entire dataset needs to be split into train and test datasets. We train our model on the train data and test the accuracy of built model prediction or classification on the test data. By splitting the dataset, we are not doing any changes to the test data which gives unaltered results of our model efficiency.

The dataset splitting is done by using predefined 'train_test_split()' function from scikit-learn as shown below. And the dataset is split into 80% of train data and 20% of test data, basically an 80-20 split.

```
In [23]: # Splitting the dataset into train and test sets: 80-20 split

         from sklearn.model_selection import train_test_split

         X_train,X_test,y_train,y_test=train_test_split(x_df_scaled,y,test_size=0.2,random_state=0,stratify=y)
         X_train.shape, X_test.shape

Out[23]: ((16281, 18), (4071, 18))
```

3.5 MODEL BUILDING & TRAINING:-

Machine Learning algorithms are off two types: Supervised and Unsupervised algorithms. The type that is focused for this project is Supervised algorithms. Supervised machine learning is one of the most commonly used and successful types of machine learning. Supervised learning is used whenever we want to predict a certain outcome/label from a given set of features, and we have examples of features-label pairs.

We build a machine learning model from these features-label pairs, which comprise our training set. Our goal is to make accurate predictions for new, never-before-seen data. Further, there are two major types of supervised machine learning problems, called classification and regression. Basically, in classification, the goal is to predict a class label, which is a choice from a predefined list of possibilities.

The below supervised machine learning models considered to train the dataset in this notebook are:

- Logistic Regression
- K Neighbors Classifier
- Decision Tree Classifier
- Random Forest Classifier
- Gradient Boosting Classifier
- Bagging Classifier

The metrics considered to evaluate the model performance are Accuracy & Root Mean Squared Error.

Logistic Regression is a Classification Problem. Logistic regression most commonly used for predicting binary classes.

5.1 LogisticRegression:-

5]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state=0)
lrclf = model.fit(X_train,y_train.values.ravel())

#predicting the target value from the model for the samples
y_train_neighcls = lrclf.predict(X_train)
y_test_neighcls = lrclf.predict(X_test)

#computing the accuracy of the model performance
acc_train_lrclf = lrclf.score(X_train, y_train)
acc_test_lrclf = lrclf.score(X_test, y_test)

#computing root mean squared error (RMSE)
rmse_train_lrclf = np.sqrt(mean_squared_error(y_train, y_train_neighcls))
rmse_test_lrclf = np.sqrt(mean_squared_error(y_test, y_test_neighcls))

print('Accuracy of Training is %s & Testing is %s'%(acc_train_lrclf,acc_test_lrclf))
print('RMSE of Training is %s & Testing is %s'%(rmse_train_lrclf,rmse_test_lrclf))
#storing the results.
storeResults('LogisticRegression', acc_train_lrclf, acc_test_lrclf, rmse_train_lrclf, rmse_test_lrclf)
```

```
Accuracy of Training is 0.6626742829064554 & Testing is 0.6592974699091132
RMSE of Training is 0.5807974837183307 & Testing is 0.5836972932016105
```

- KNeighborsClassifier

K nearest neighbors is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure (e.g., distance functions). It can be performed on both classification and regression problems.

5.2 KNeighborsClassifier:-

In [26]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

neigh = KNeighborsClassifier(n_neighbors=3)
neighcls=neigh.fit(X_train,y_train.values.ravel())

#predicting the target value from the model for the samples
y_train_neighcls = neighcls.predict(X_train)
y_test_neighcls = neighcls.predict(X_test)

#computing the accuracy of the model performance
acc_train_neighcls = neighcls.score(X_train, y_train)
acc_test_neighcls = neighcls.score(X_test, y_test)

#computing root mean squared error (RMSE)
rmse_train_neighcls = np.sqrt(mean_squared_error(y_train, y_train_neighcls))
rmse_test_neighcls = np.sqrt(mean_squared_error(y_test, y_test_neighcls))

print('Accuracy of Training is %s & Testing is %s'%(acc_train_neighcls,acc_test_neighcls))
print('RMSE of Training is %s & Testing is %s'%(rmse_train_neighcls,rmse_test_neighcls))
#storing the results.
storeResults('KNeighborsClassifier', acc_train_neighcls, acc_test_neighcls, rmse_train_neighcls, rmse_test_neighcls)
```

```
Accuracy of Training is 0.714083901480253 & Testing is 0.6067305330385655
RMSE of Training is 0.5347112290944963 & Testing is 0.6271120051166574
```


- DecisionTreeClassifier

Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems. Uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

5.3. DecisionTreeClassifier:-

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

dtree = DecisionTreeClassifier(random_state=0)
dtree.fit(X_train,y_train.values.ravel())

#predicting the target value from the model for the samples
y_test_tree = dtree.predict(X_test)
y_train_tree = dtree.predict(X_train)

#computing the accuracy of the model performance
acc_train_tree = dtree.score(X_train, y_train)
acc_test_tree = dtree.score(X_test, y_test)

print('Accuracy of Training is %s & Testing is %s'%(acc_train_tree,acc_test_tree))

#computing root mean squared error (RMSE)
rmse_train_tree = np.sqrt(mean_squared_error(y_train, y_train_tree))
rmse_test_tree = np.sqrt(mean_squared_error(y_test, y_test_tree))

print('Accuracy of Training is %s & Testing is %s'%(acc_train_tree,acc_test_tree))
print('RMSE of Training is %s & Testing is %s'%(rmse_train_tree,rmse_test_tree))
#storing the results.
storeResults('DecisionTreeClassifier', acc_train_tree, acc_test_tree, rmse_train_tree, rmse_test_tree)
```



```
Accuracy of Training is 0.8043117744610282 & Testing is 0.6214689265536724
Accuracy of Training is 0.8043117744610282 & Testing is 0.6214689265536724
```

- RandomForestClassifier

Random forest is a ensemble of decision trees supervised learning algorithm. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

5.4. RandomForestClassifier

8]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)

pred_df = pd.DataFrame(rf.predict(X_test), columns=['pred'])
pred_df['actual'] = y_test.values
pred_df['residual'] = (pred_df['pred'] - pred_df['actual']).abs()

#predicting the target value from the model for the samples
y_train_rf = rf.predict(X_train)
y_test_rf =rf.predict(X_test)

#computing the accuracy of the model performance
acc_train_rf = rf.score(X_train, y_train)
acc_test_rf = rf.score(X_test, y_test)

#computing root mean squared error (RMSE)
rmse_train_rf = np.sqrt(mean_squared_error(y_train, y_train_rf))
rmse_test_rf = np.sqrt(mean_squared_error(y_test, y_test_rf))

print('Accuracy of Training is %s & Testing is %s'%(acc_train_rf,acc_test_rf))
print('RMSE of Training is %s & Testing is %s'%(rmse_train_rf,rmse_test_rf))

#storing the results.
storeResults('RandomForestClassifier', acc_train_rf, acc_test_rf, rmse_train_rf, rmse_test_rf)
```

GradientBoostingClassifier:-

Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

5.5. GradientBoostingClassifier

[29]:
```
from sklearn.ensemble import GradientBoostingClassifier

clf_gb = GradientBoostingClassifier(random_state=0)
clf_gb.fit(X_train,y_train.values.ravel())

#predicting the target value from the model for the samples
y_train_tree = clf_gb.predict(X_train)
y_test_tree = clf_gb.predict(X_test)

#computing the accuracy of the model performance
acc_train_gb = clf_gb.score(X_train, y_train)
acc_test_gb = clf_gb.score(X_test, y_test)

#computing root mean squared error (RMSE)
rmse_train_gb = np.sqrt(mean_squared_error(y_train, y_train_tree))
rmse_test_gb = np.sqrt(mean_squared_error(y_test, y_test_tree))

print('Accuracy of Training is %s & Testing is %s'%(acc_train_gb,acc_test_gb))
print('RMSE of Training is %s & Testing is %s'%(rmse_train_gb,rmse_test_gb))

#storing the results.
storeResults('GradientBoostingClassifier', acc_train_gb, acc_test_gb, rmse_train_gb, rmse_test_gb)
```

```
Accuracy of Training is 0.6838032061912659 & Testing is 0.6725620240727094
RMSE of Training is 0.562313785896037 & Testing is 0.5722219638630542
```

Bagging Classifier:-

Bagging, or bootstrap aggregation, is a model aggregation technique to reduce model variance. Data is split into multiple samples with replacement called bootstrap samples. A bootstrap sample often is 3/4 of the original values and replacement resulting in repetition of values inside each model run.

5.6. BaggingClassifier:-

30]:
```
from sklearn.ensemble import BaggingClassifier

# Create a bag of estimators of size 11
dtree_bag = BaggingClassifier(base_estimator=dtree, n_estimators=100, random_state=555, n_jobs=-1)

dtree_bag.fit(X_train,y_train)

results = dtree_bag.score(X_test, y_test)

#predicting the target value from the model for the samples
y_train_dtree_bag = dtree_bag.predict(X_train)
y_test_dtree_bag = dtree_bag.predict(X_test)

#computing the accuracy of the model performance
dtree_bag_train_gb = dtree_bag.score(X_train, y_train)
dtree_bag_test_gb = dtree_bag.score(X_test, y_test)

#computing root mean squared error (RMSE)
dtree_rmse_train_gb = np.sqrt(mean_squared_error(y_train, y_train_dtree_bag))
dtree_rmse_test_gb = np.sqrt(mean_squared_error(y_test, y_test_dtree_bag))

print('Accuracy of Training is %s & Testing is %s'%(dtree_bag_train_gb,dtree_bag_test_gb))
print('RMSE of Training is %s & Testing is %s'%(dtree_rmse_train_gb,dtree_rmse_test_gb))

#storing the results.
storeResults('BaggingClassifier', dtree_bag_train_gb, dtree_bag_test_gb, dtree_rmse_train_gb, dtree_rmse_test_gb)
```

3.6 Comparing Model Results:-

Out[32]:

	ML Model	Train Accuracy	Test Accuracy	Train RMSE	Test RMSE
0	LogisticRegression	0.663	0.659	0.581	0.584
1	KNeighborsClassifier	0.714	0.607	0.535	0.627
2	DecisionTreeClassifier	0.804	0.621	0.442	0.615
3	RandomForestClassifier	0.804	0.620	0.442	0.617
4	GradientBoostingClassifier	0.684	0.673	0.562	0.572
5	BaggingClassifier	0.804	0.624	0.442	0.613

4. PYCARET:-

PyCaret is an open source, low-code machine learning library in Python that allows you to go from preparing your data to deploying your model within minutes in your choice of notebook environment.

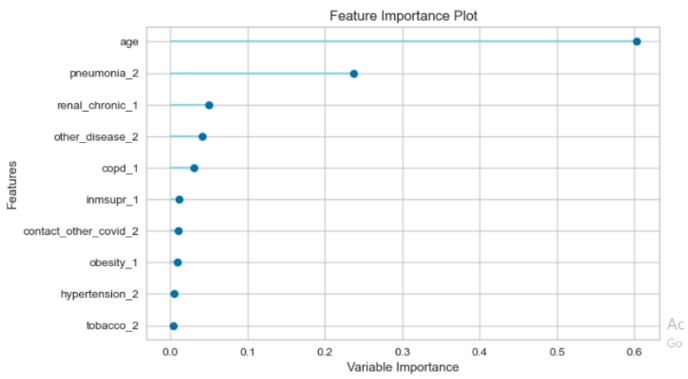
Increased Productivity:- PyCaret being a low-code library makes you more productive. With less time spent coding, you and your team can now focus on business problems.

Easy to Use:- PyCaret is simple and easy to use machine learning library that will help you to perform end-to-end ML experiments with less lines of code.

Business Ready:- PyCaret is a business ready solution. It allows you to do prototyping quickly and efficiently from your choice of notebook environment.

```
In [33]: from pycaret.classification import *
         np.seterr(divide='ignore', invalid='ignore')

Out[33]: {'divide': 'raise', 'over': 'raise', 'under': 'raise', 'invalid': 'raise'}

In [34]: clf = setup(data = df, target = 'covid_res', session_id=123, numeric_imputation = 'mean',categorical_imputation= 'mode')
```

```
In [41]: #plot feature importance
         plot_model(tuned_rf, plot='feature')
```



Feature

```
In [36]: best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ada	Ada Boost Classifier	0.6738	0.7007	0.4402	0.6627	0.5289	0.2947	0.3092	0.1760
gbc	Gradient Boosting Classifier	0.6737	0.7050	0.4471	0.6594	0.5328	0.2961	0.3093	0.3560
catboost	CatBoost Classifier	0.6714	0.6999	0.4677	0.6454	0.5422	0.2960	0.3055	19.1440
lightgbm	Light Gradient Boosting Machine	0.6706	0.6981	0.4729	0.6414	0.5443	0.2955	0.3040	0.2630
xgboost	Extreme Gradient Boosting	0.6643	0.6847	0.4677	0.6303	0.5369	0.2825	0.2903	1.0270
lda	Linear Discriminant Analysis	0.6638	0.6863	0.4561	0.6334	0.5301	0.2791	0.2883	0.0360
lr	Logistic Regression	0.6635	0.6863	0.4551	0.6332	0.5294	0.2783	0.2876	0.5330
ridge	Ridge Classifier	0.6634	0.0000	0.4534	0.6337	0.5284	0.2778	0.2873	0.0230
nb	Naive Bayes	0.6319	0.6687	0.3874	0.5878	0.4669	0.2036	0.2139	0.0210
et	Extra Trees Classifier	0.6262	0.6185	0.4606	0.5622	0.5063	0.2103	0.2131	0.5360
rf	Random Forest Classifier	0.6247	0.6364	0.4883	0.5561	0.5199	0.2139	0.2152	0.5020
knn	K Neighbors Classifier	0.6218	0.6221	0.4348	0.5587	0.4888	0.1961	0.2001	0.1360
dt	Decision Tree Classifier	0.6205	0.5959	0.4517	0.5542	0.4976	0.1978	0.2006	0.0300
svm	SVM - Linear Kernel	0.5743	0.0000	0.5444	0.6057	0.4635	0.1463	0.1775	0.1370
qda	Quadratic Discriminant Analysis	0.5406	0.5209	0.4276	0.4551	0.3467	0.0461	0.0634	0.0240

6. CONCLUSION:-

Above models helps to classify whether the patient will be tested COVID Positive or Negative.

From above models comparing results, we can conclude that Random Forest classifier gives best training and testing accuracy compared to other models for the given dataset.

Using Pycaret we get automated output without explicitly running each model.