BLOG ON

GLOBAL POWER PLANT

SUBMITTED BY

NAINCY JOSHI

Power Plant

India is a land of beautiful dams and power plants. A power plant is basically an industrial location that is utilized for the generation and distribution of power. There are three types of power plants in India- Hydro Electric power plants, Thermal power plants, Nuclear Power plants. So, let's know about these power plants in detail.

RENEWABLE SOURCE OF ENERGY

- Solar power
- Wind power
- Biomass
- Small hydro
- Waste-to-energy

NON-RENEWABLE SOURCE OF ENERGY

- Coal
- Crude oil
- Natural gas
- Uranium

DATASET DETAILS

- We have detail about the power plant in India.
- We have 908 Rows and 25 Columns.
- country
- country_long name
- gppd_idnr
- capacity_mw
- latitude
- longitude
- primary_fuel
- other_fuel1
- other_fuel2
- other_fuel3
- commissioning_years
- Source

- url
- geolocation_source
- wepp_id
- year_of_capacity_data
- generation_gwh_2013
- generation_gwh_2014
- generation_gwh_2015
- generation_gwh_2016
- generation_gwh_2017
- generation_data_source
- estimated_generation_gwh

EXPLORATORY DATA ANALYSIS:-

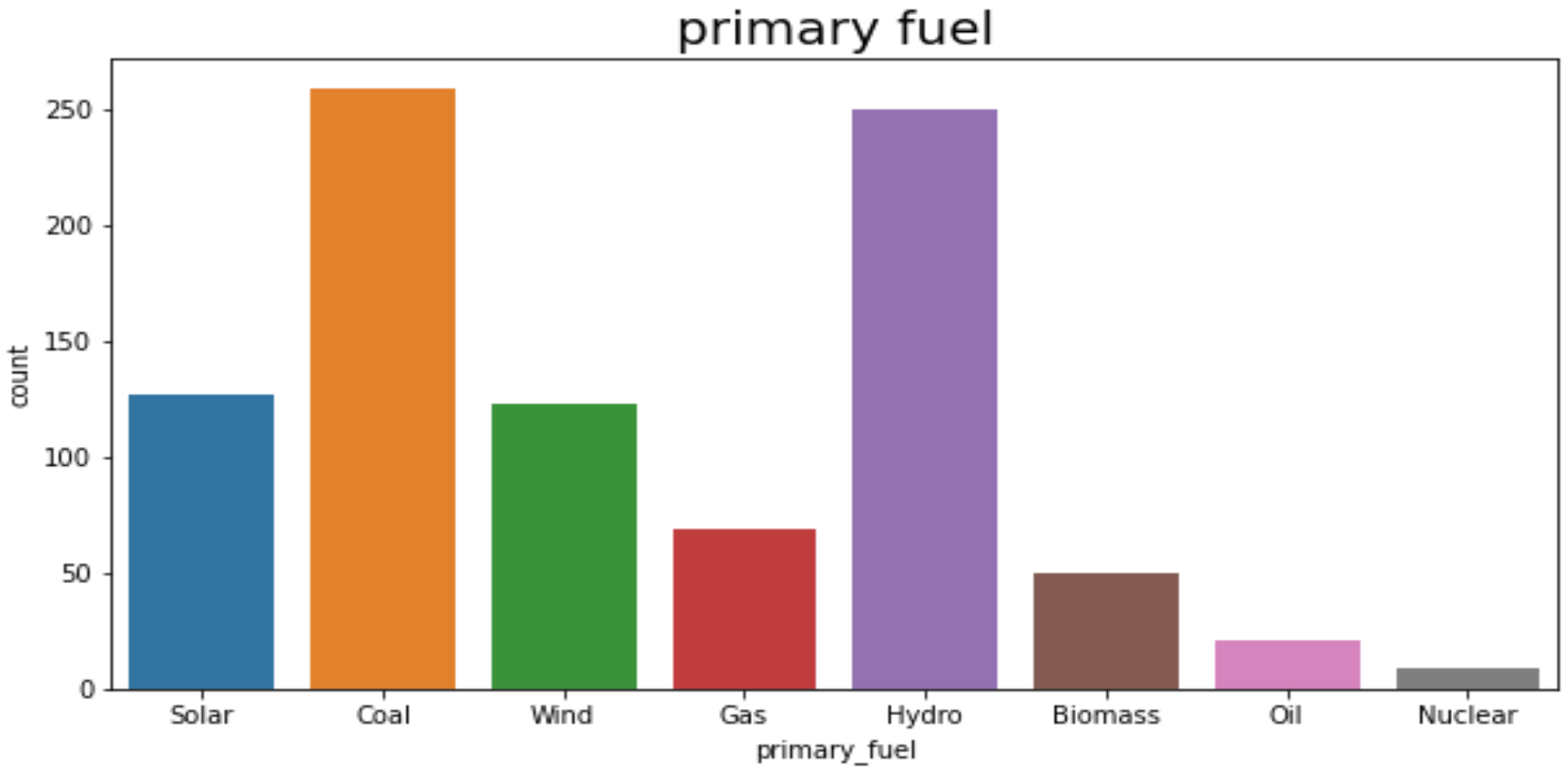
Finding Missing Value:data.isna().sum()
latitude 46
longitude 46
other_fuel1 709
other_fuel2 907
other_fuel3 908
commissioning_year 380
owner 566
geolocation_source 19
wepp_id 908
year_of_capacity_data 388
generation_gwh_2013 524
generation_gwh_2014 507
generation_gwh_2015 483
generation_gwh_2016 471
generation_gwh_2017 465
generation_data_source 458
estimated_generation_gwh 908

EDA CONCLUSION REMARK:-

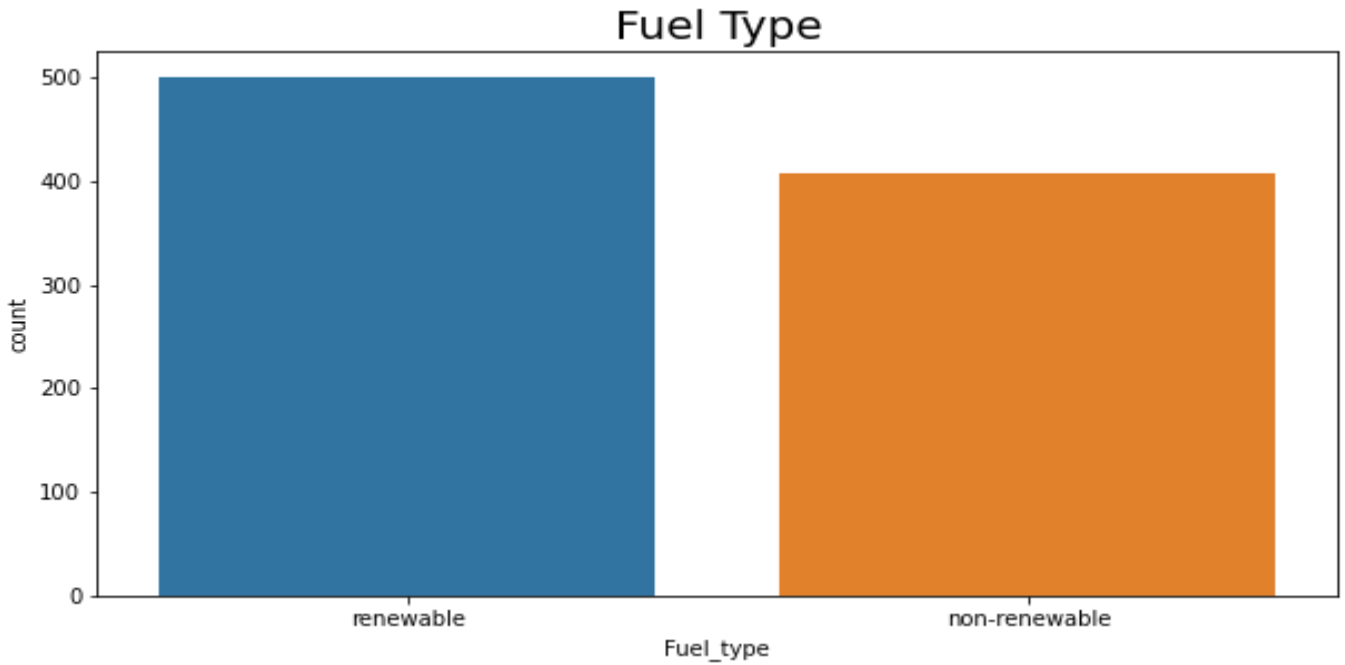
- There are lot of missing Value
- There are 908 rows and in many columns there is no data so we will drop that column
- 'wepp_id','estimated_generation_gwh','other_fuel3 ' have 908 missing value i.e equals to the row
- 'other_fuel2' has 907 missing value and 'other_fuel1' have more than 80% of missing value
- The column more than 50% missing value are 'generation_gwh_2013','generation_gwh_2014','ge neration_gwh_2015', 'generation_gwh_2016','generation_gwh_2017' and the Owner
- We can also delete country_long because country also contain same information

We will add one column ‘Fuel_type’based upon the primary fuel type

- In the Fuel_type column we will divide the primary_fuel into Renewable_fuel and Non-Renewable_fuel.
- The Fuel_type will become the ‘target column’ which has binary classification problem because the column contain 2 variable i.e renewable and the non-renewable_fuel



The higgest no of plants are of Coal,hydro,solar , wind and the least no of plants are of nuclear and oil



There are more number of renewable plants than the non-renewable plant

Removing the NAN values

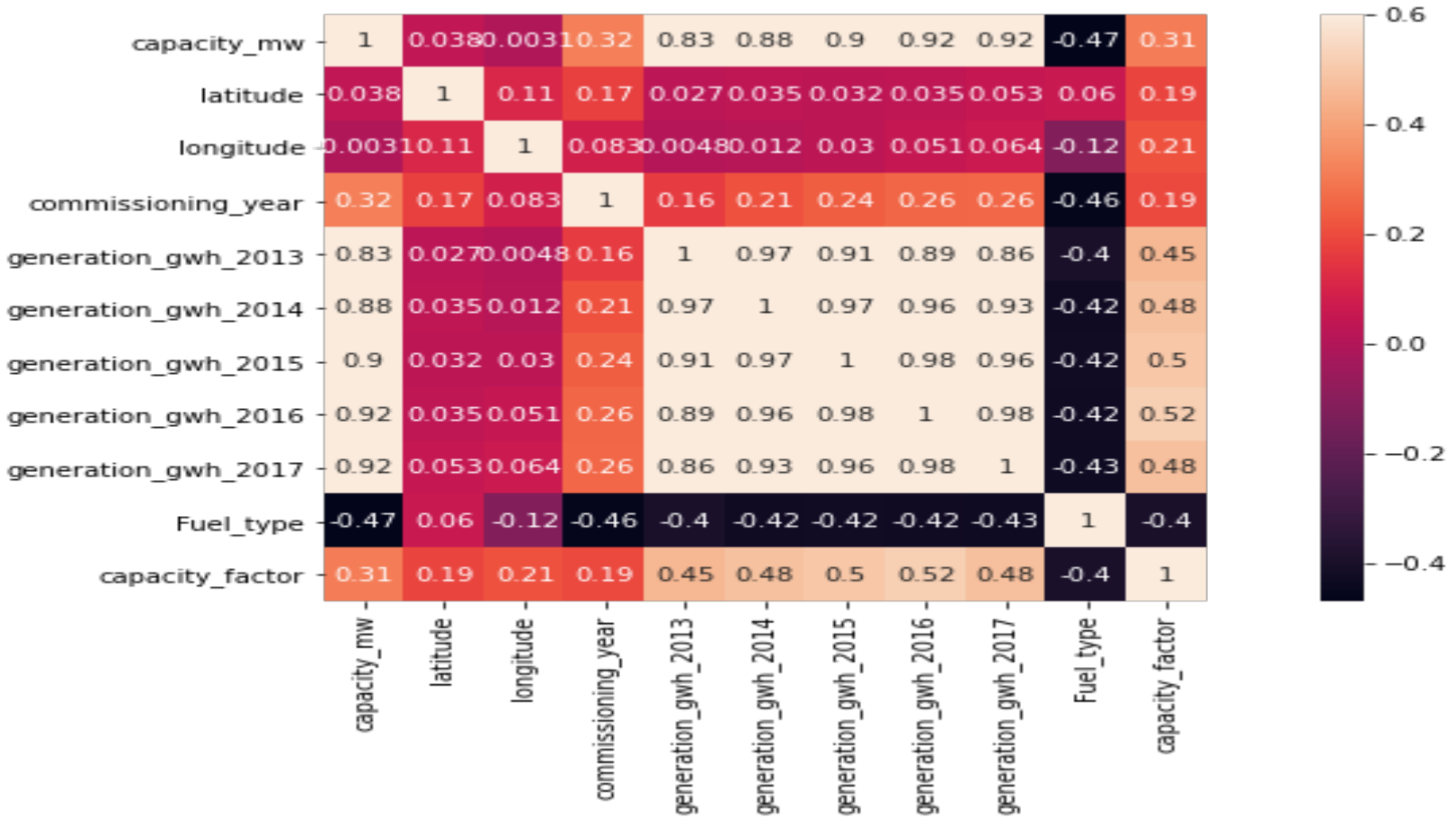
- MIN_CAPACITY_FACTOR = 0.01
- MAX_CAPACITY_FACTOR = 1.0
- We will set the Min and the max_capacity_factor to replace the nan values.
- Converting 2016 generation into capacity factor and remove rows with erroneous capacity factors
- data['capacity_factor'] = data.apply(lambda row:row['generation_gwh_2016']/(24.0*365.0*0.001*row['capacity_mw']),axis=1)
- data = data[data.capacity_factor >= MIN_CAPACITY_FACTOR]
- data = data[data.capacity_factor <= MAX_CAPACITY_FACTOR]

“Data Preprocessing and Pipeline”

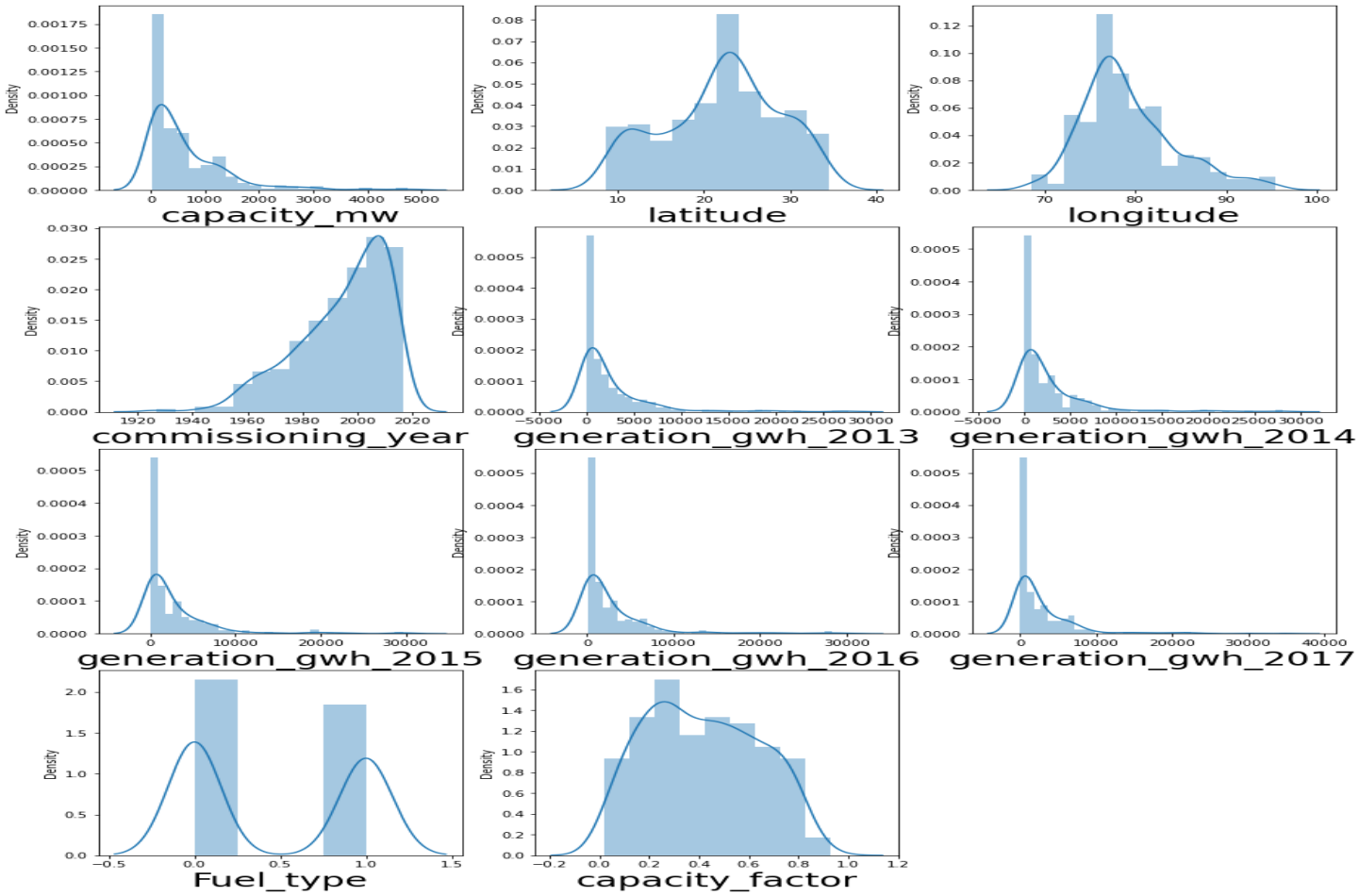
- **Remove the outliers** from the dataset. We can use the various outliers removal techniques to detect and address the outliers in the dataset. Common Practices are using the Z score method and the Inter Quartile Range method (IQR) for addressing the outliers. Note that is it important to remove all the NaN values before using the Z-Score and the IQR algorithms. Also, outlier removal must be performed on the numerical features only and not categorical variables.

The Next step would be addressing the **skewness of the data**. Data Scientists are very uptight about the skewness of the numerical features in the dataset. There are 3 types of skews distribution: Positive Skew, Negative Skew, and Normal Skew. The data analysts are very fond of normal distribution. As were are the Data Scientists in the making, so whenever we have a numerical feature in the dataset and the feature is skew we tend to make the skew more normal using some skew transformation methods. Common Practices for skew transformation are using log transformation, square root transformation, and power n transformation(Box-Cox). The valid skews are where the skew value of the feature is between -1 to +1. The features with skew greater than absolute 1 (abs(1)) are invalid.

Checking the correlation with the features to the target column



Plotting the graph to see the outlier:-



- From the figure,we can conclude that many columns have outliers.
- We will remove the outlier using Z-Score.

```
from scipy.stats import zscore
z=np.abs(zscore(data))
threshold=3
data_new=data[(z<3).all(axis=1)]
print(data.shape)
print(data_new.shape)
```

- (349, 11) data.shape()
- (332, 11) data_new.shape()

Removing skewness using LabelEncoder()

Before Removing skewness

- capacity_mw 2.360446
- latitude -0.229299
- longitude 0.874606
- commissioning_year -0.963851
- generation_gwh_2013 3.272933
- generation_gwh_2014 3.151795
- generation_gwh_2015 3.421393
- generation_gwh_2016 3.358910
- generation_gwh_2017 3.451648
- Fuel_type 0.155864
- capacity_factor 0.145771

After Skewness removed

- capacity_mw -0.037236
- latitude 0.029099
- longitude 0.002144
- commissioning_year -0.800774
- generation_gwh_2013 0.003348
- generation_gwh_2014 0.004417
- generation_gwh_2015 0.001629
- generation_gwh_2016 0.000000
- generation_gwh_2017 0.007091
- Fuel_type 0.155864
- capacity_factor 0.000000 dtype: float64

Concluding Preprocessing remarks

- Label encoding is done for categorical target variable and ordinal type feature encoding.
- We have used Z-score to remove Outlier.
- We have used LabelEncoder() to remove the skewness()

“ Building the Machine Learning Classification Model”

- Firstly we need to **separate the features and targets**. As per the convention, we will score features in ‘x’ and ‘y’ variables. Always keep checking the shape of the arrays and the data frames, this will make your debugging easy.

```
x = data.drop("Fuel_type", axis=1)
y= data["Fuel_type"]
print(x.shape,y.shape)
(349, 9) (349,)
```

We do not have any other train dataset, thus we will randomly select some percentage of the samples as the train set and validate the model on that data. The sklearn library provides us many functions to do such splits. We will use the **train_test_split** function. We will split the data in 80:20 train test ratio.

- from sklearn.model_selection import train_test_split, cross_validate
- x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=31, test_size=.20)
- print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)
- (279, 9) (279,) (70, 9) (70,)

- The algorithms considered in this section are: **Logistic Regression**, **Random Forest**, **SVM**, **KNN**, **Decision Tree Classifier**, **Gaussian NB, SGDClassifier,Linear_SVC.**

- Let's evaluate each model in turn and provide **accuracy** and **confusion matrix scores**.

- **Random Forest Regressor** and **Decision Tree Regressor** performed best with **extremely low error** and a **good R2 score.**

- The Confusion matrix provides us with a much more detailed representation of the accuracy score and of what's going on with our labels — we know exactly which/how labels were correctly and incorrectly predicted. The accuracy of the Logistic Regression Classifier on test set is 78.57

Hyper-parameter Tunning

Performing hyper parameter tuning to get good and more accurate result from the model.

HyperParameter Tunning

In [53]:
```
from sklearn.metrics import r2_score, mean_squared_error,mean_absolute_error,roc_auc_score
from sklearn.model_selection import GridSearchCV
c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space}
search= GridSearchCV(lr, param_grid, cv = 5)
```

In [54]:
```
search.fit(x, y)
```

Out[54]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.00000000e-05, 8.48342898e-05, 7.19685673e-04, 6.10540230e-03,
       5.17947468e-02, 4.39397056e-01, 3.72759372e+00, 3.16227766e+01,
       2.68269580e+02, 2.27584593e+03, 1.93069773e+04, 1.63789371e+05,
       1.38949549e+06, 1.17876863e+07, 1.00000000e+08])})
```

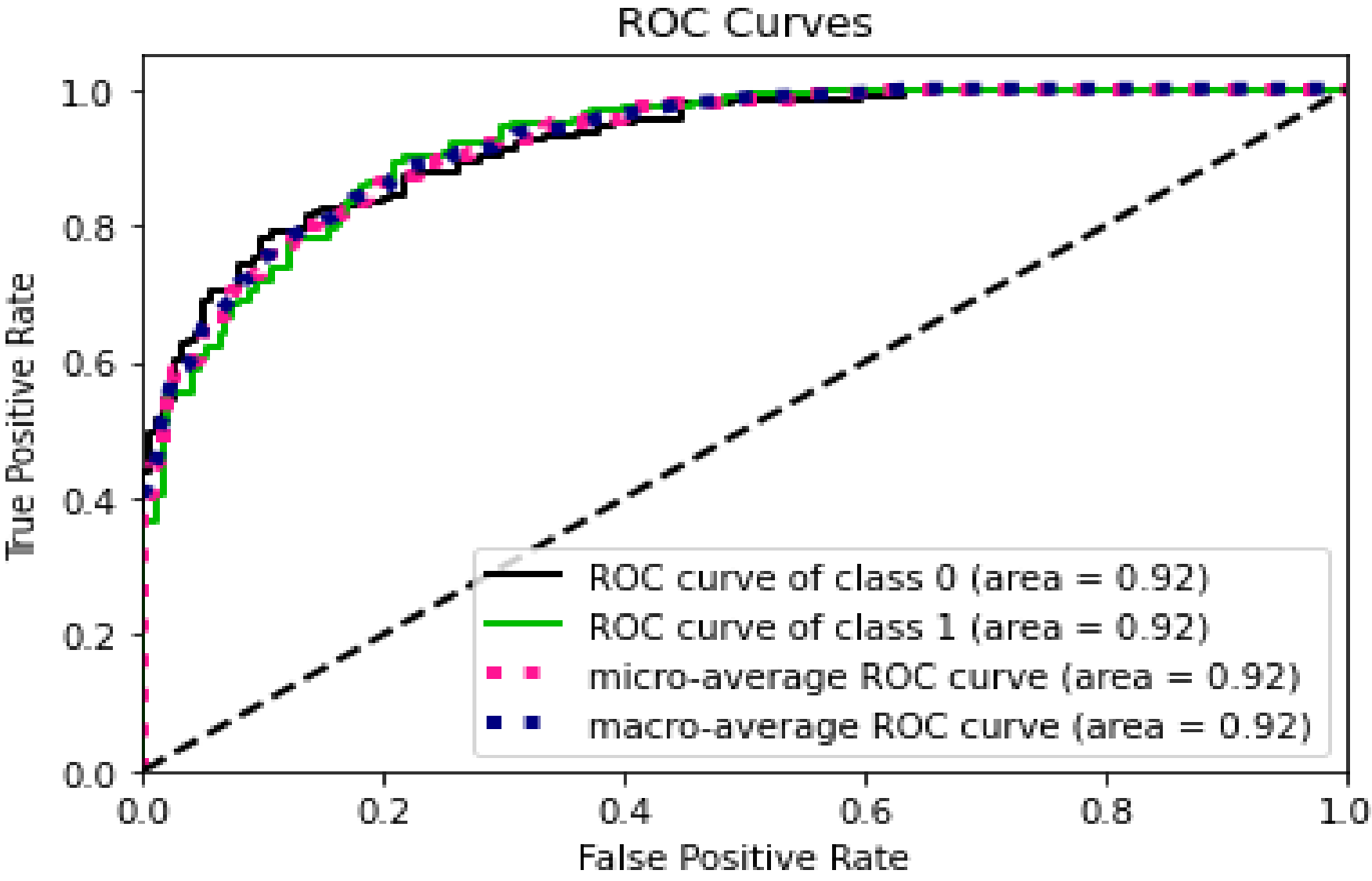
In [55]:
```
best_model = search.best_params_
best_model
```

Out[55]:
```
{'C': 0.0007196856730011522}
```

In [56]:
```
logreg = LogisticRegression(C= 0.006105402296585327)
logreg.fit(x, y)
Y_pred = logreg.predict(x)
acc_log = round(logreg.score(x, y) * 100, 2)
acc_log
```

Out[56]:
```
80.23
```

Area under ROC Curve (or AUC for short) is a performance metric for binary classification problems. The AUC represents a **model's ability to discriminate between positive and negative classes**, and is better suited to this project. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model as good as random.


ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.92)
ROC curve of class 1 (area = 0.92)
micro-average ROC curve (area = 0.92)
macro-average ROC curve (area = 0.92)

Saving the model

- We will save the model using “Joblib”.

Concluding Remark:-

- We have increased the model accuracy from 78.57 to 80.23 by doing hyper_parameter tunning.