```python
# %matplotlib widget
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn import preprocessing
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay, precision_recall_curve, PrecisionRecallDisplay, classification_
from sklearn.svm import SVC, SVR
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.ensemble import AdaBoostClassifier
import seaborn as sns

from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import accuracy_score, precision_score, classification_report
```

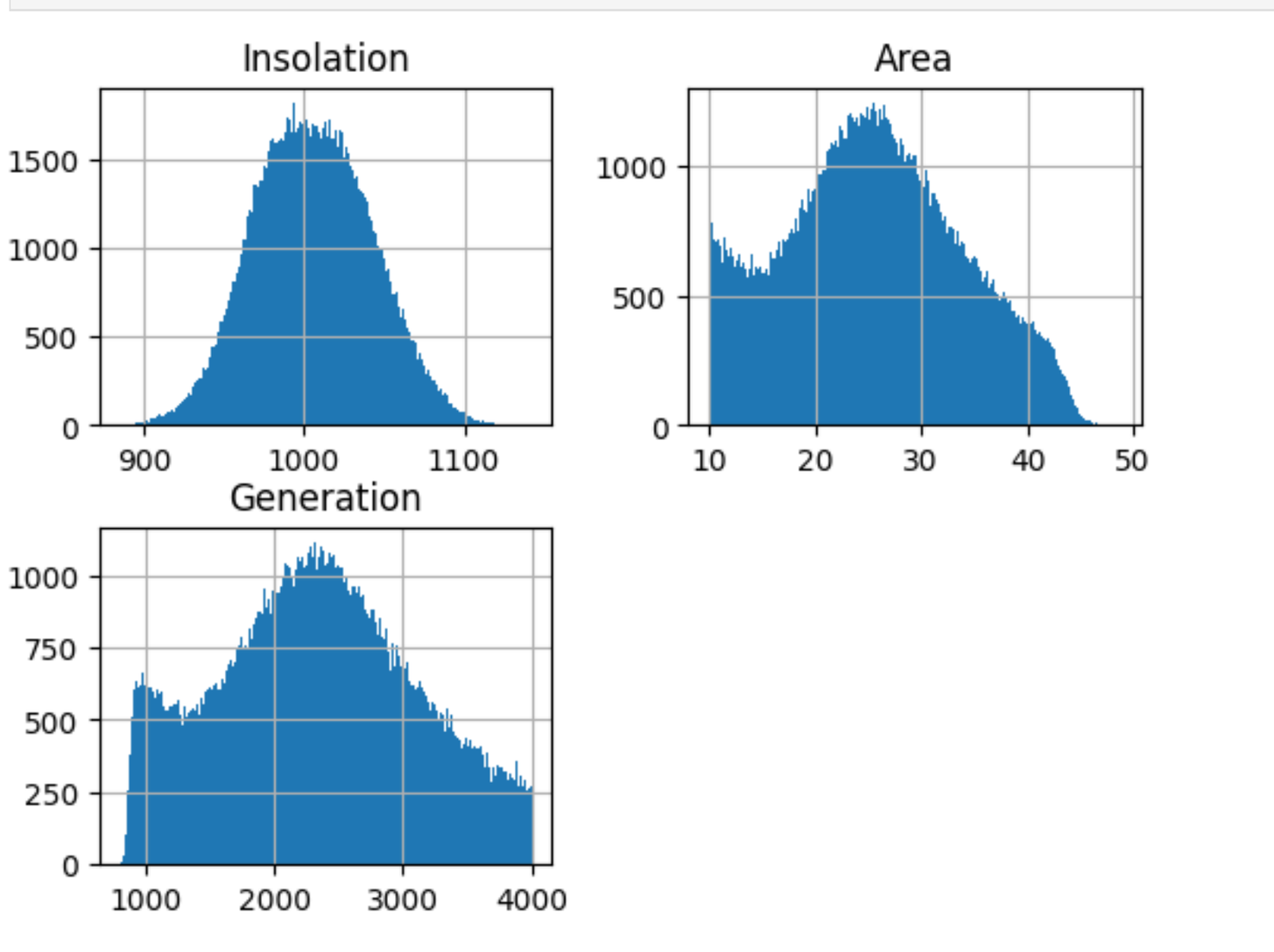
Reading the solar-potential dataset

```python
read_full = pd.read_csv("solar-potential.csv")
read_full = read_full[read_full.Generation <= 4000]
read_full = read_full.drop("OBJECTID", axis=1)

print(read_full.head())
print(read_full.count())
```

```
   Insolation     Area  Generation  CO2 Saving kg per year
0    984.0645  30.2502   2679.1321               1454.7687  \
1    945.7500  10.2481    872.2964                473.6570
2    974.5625  11.6229   1019.4489                553.5607
3    966.0000  16.9759   1475.8834                801.4047
4    975.8387  29.4817   2589.2466               1405.9609

   Solar PV Capacity kWp    Suitability  SHAPE.STArea()  SHAPE.STLength()
0                 4.3215  well suitable        45.86250         31.920914  \
1                 1.4640  well suitable        49.37000         32.031072
2                 1.6604  well suitable        48.64625         33.574782
3                 2.4251  well suitable        49.34000         32.216502
4                 4.2117  well suitable        47.42250         32.271527

                                          geo_shape   Latitude  Longitude
0  {"coordinates": [[[-2.5771009761244272, 51.483...  51.483829  -2.577005
1  {"coordinates": [[[-2.5765328225610578, 51.483...  51.483843  -2.576479
2  {"coordinates": [[[-2.576270566071664, 51.4840...  51.484132  -2.576356
3  {"coordinates": [[[-2.5765726627120293, 51.483...  51.483798  -2.576499
4  {"coordinates": [[[-2.5769878354028837, 51.483...  51.483732  -2.577043
Insolation                133129
Area                      133129
Generation                133129
CO2 Saving kg per year    133129
Solar PV Capacity kWp     133129
Suitability               133129
SHAPE.STArea()            133129
SHAPE.STLength()          133129
geo_shape                 133129
Latitude                  133129
Longitude                 133129
dtype: int64
```

Histogram of Relevant Attributes

```python
read_full.hist(column=["Insolation","Area","Generation"],bins=200)
# ax = read_full.hist(column=["Insolation"], bins=200)
# plt.xlabel("Insolation Irradiance kWh/m2/year")
# plt.ylabel("Frequency")
# plt.title("Insolation Histogram")
plt.show()
```



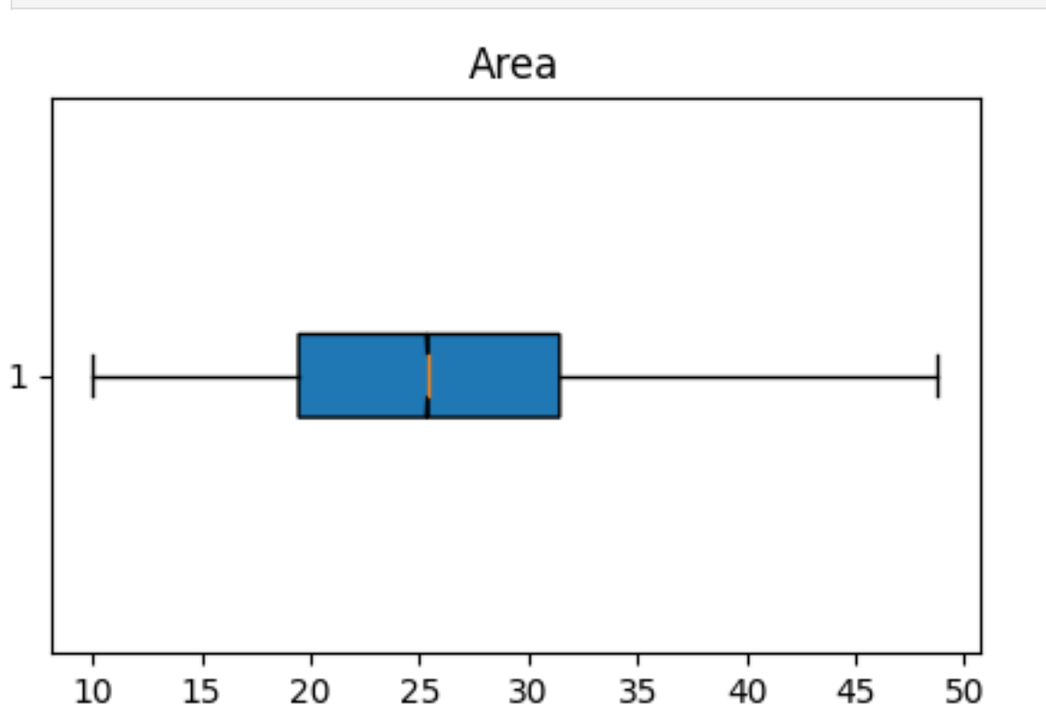
Splitting data for different models

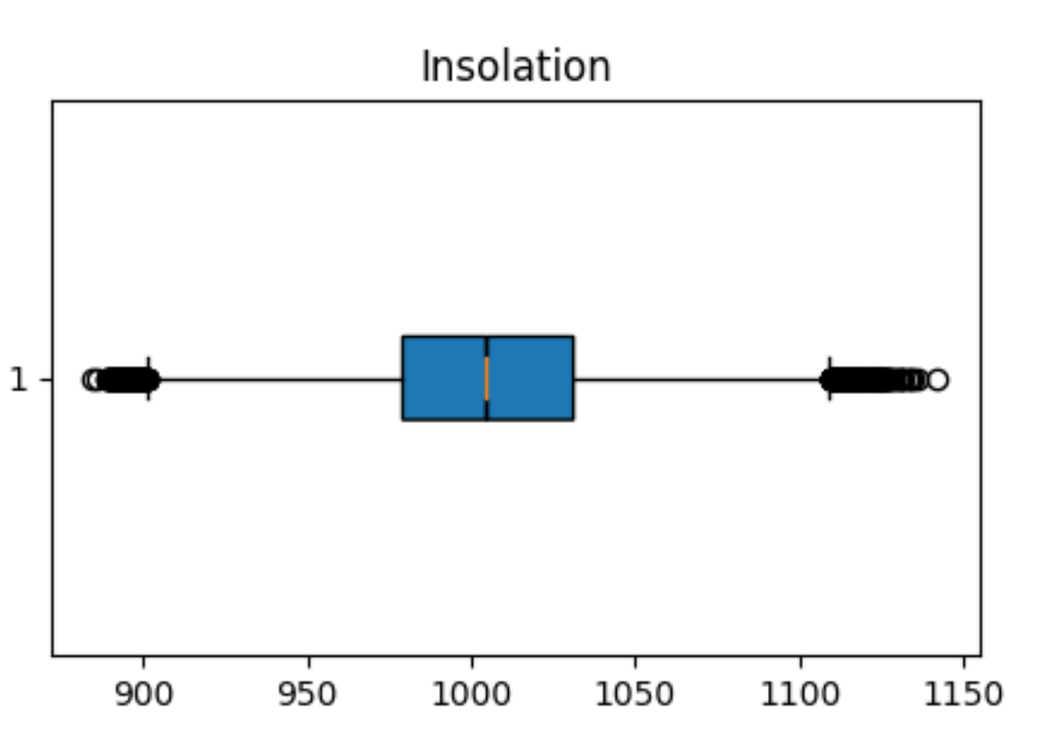
```python
X = read_full[["Insolation", "Area"]].copy()
y = read_full["Suitability"].copy()
# scaleX = StandardScaler().fit(X)
# X = scaleX.transform(X)
train_x, test_x, train_y, test_y = train_test_split(X,y, test_size=0.2)



X_gen = read_full[["Insolation", "Area"]].copy()
y_gen = read_full["Generation"].copy()
# scaleX_gen = StandardScaler().fit(X_gen)
# X_gen = scaleX_gen.transform(X_gen)
X_gen_train, X_gen_test, y_gen_train, y_gen_test = train_test_split(X_gen,y_gen, test_size=0.2)



# X_corr = read_full["geo_point_2d"].str.split(", ", expand = True)
buff = read_full['Insolation']
buff = buff.rename("Insolation/m2")
print(buff.head())
X_corr = pd.concat([read_full["Latitude"],read_full["Longitude"],buff],axis=1)
X_corr = X_corr.rename(columns={"Latitude":"Lat",1:"Longitude",'':"Insolation/m2"})
print(X_corr)
y_corr = read_full["Generation"]/read_full["Area"]
scaleX_corr = StandardScaler().fit(X_corr)
X_corr = scaleX_corr.transform(X_corr)
X_corr_train, X_corr_test, y_corr_train, y_corr_test = train_test_split(X_corr,y_corr, test_size=0.2)
```

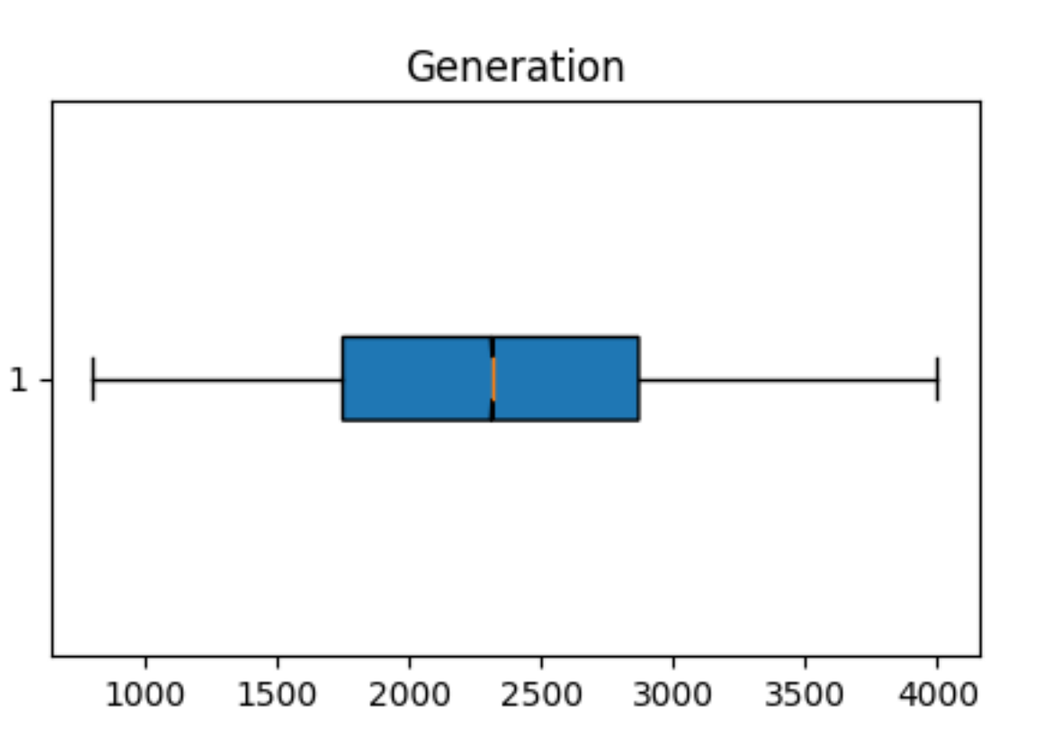
```
0    984.0645
1    945.7500
2    974.5625
3    966.0000
4    975.8387
Name: Insolation/m2, dtype: float64
              Lat  Longitude  Insolation/m2
0       51.483829  -2.577005       984.0645
1       51.483843  -2.576479       945.7500
2       51.484132  -2.576356       974.5625
3       51.483798  -2.576499       966.0000
4       51.483732  -2.577043       975.8387
...           ...        ...            ...
158143  51.406617  -2.583654       967.0000
158144  51.422928  -2.573583      1030.7727
158145  51.444919  -2.552712       946.3077
158146  51.444889  -2.552765       999.4000
158147  51.406365  -2.584466       962.2143

[133129 rows x 3 columns]
```

Box Plot for Relevant Attributes

```python
fig1 = plt.figure(figsize =(5, 3))
fig2 = plt.figure(figsize =(5, 3))
fig3 = plt.figure(figsize =(5, 3))

# Creating plot
ax1 = fig1.add_subplot()
ax2 = fig2.add_subplot()
ax3 = fig3.add_subplot()

bp1 = ax1.boxplot(read_full["Area"], patch_artist = True, notch = 'True', vert = 0, )
bp2 = ax2.boxplot(read_full["Insolation"], patch_artist = True, notch = 'True', vert = 0)
bp3 = ax3.boxplot(read_full["Generation"], patch_artist = True, notch = 'True', vert = 0)

ax1.title.set_text('Area')
ax2.title.set_text('Insolation')
ax3.title.set_text('Generation')
# show plot
plt.show()
```

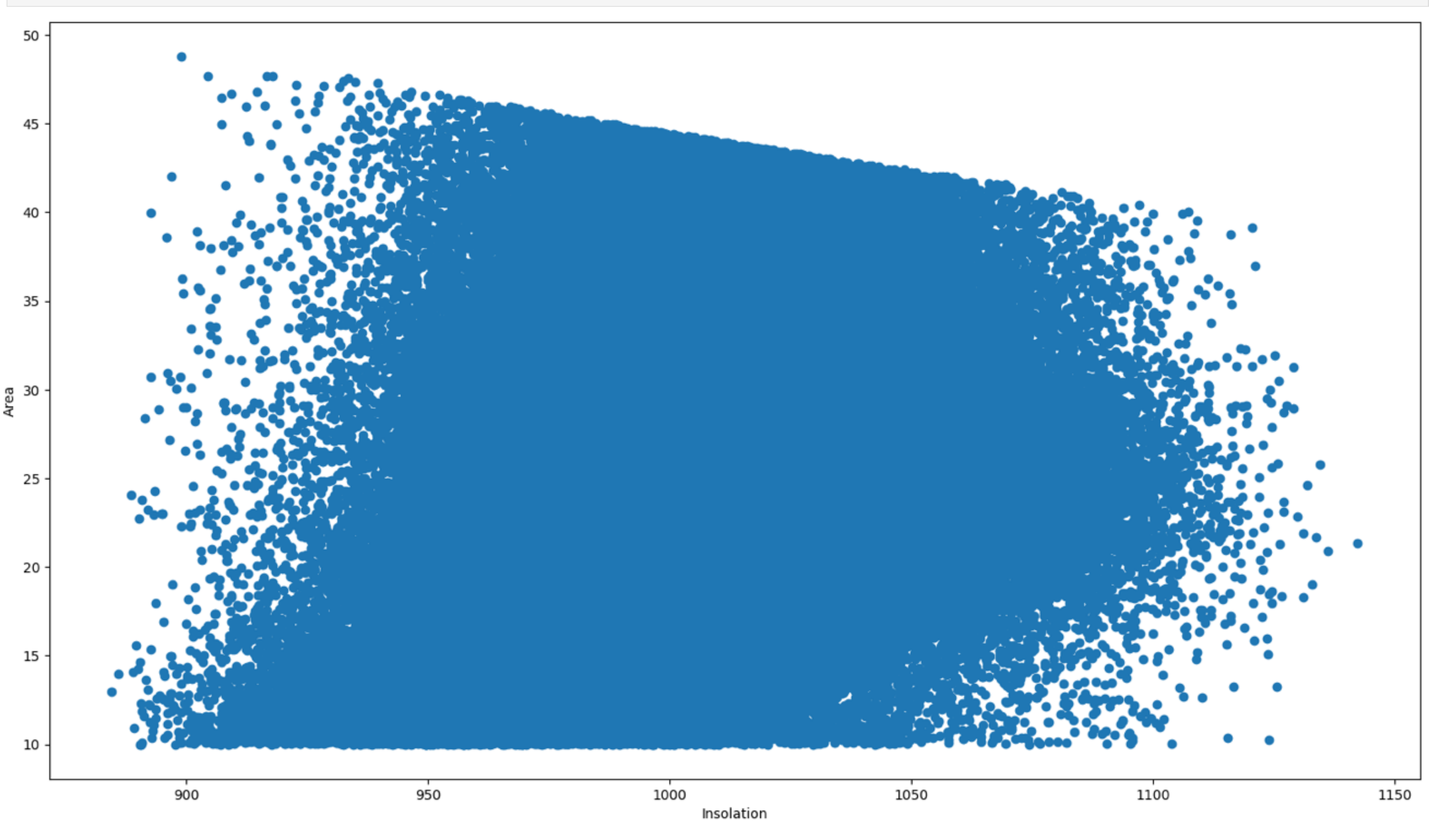






Scatter Plot for Insolation vs Area

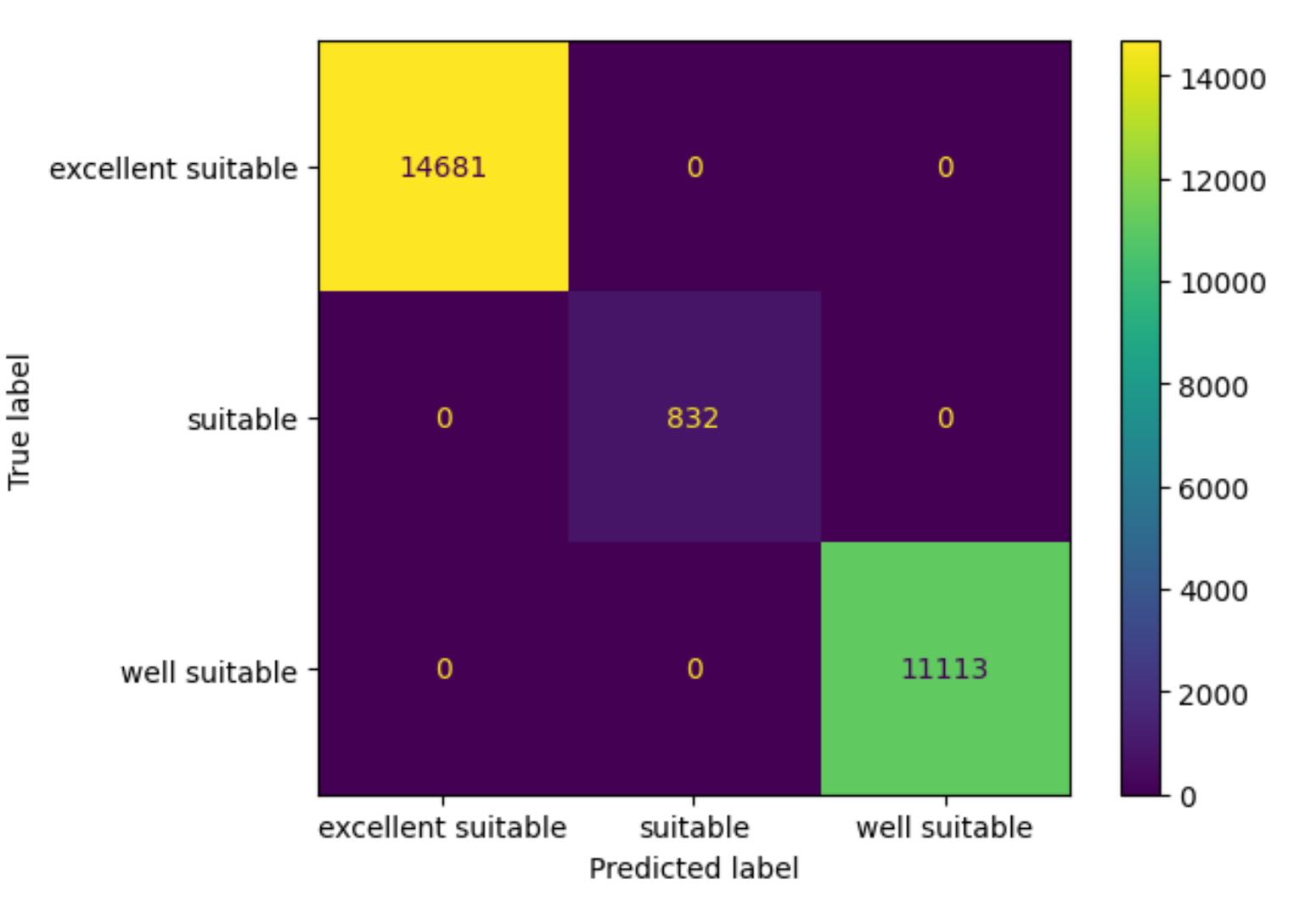
```python
fig, ax = plt.subplots(figsize = (18,10))

ax.scatter(read_full["Insolation"], read_full["Area"])
plt.xlabel("Insolation")
plt.ylabel("Area")

plt.show()
```



Code for Random Forest Classifier

```python
rfc = RandomForestClassifier(n_estimators=50, max_features="sqrt", max_samples=0.7)

# Fit RFC and predict using the testing set
rfc.fit(train_x, train_y)
pred2 = rfc.predict(test_x)

# Performance Report of rfc
print(f"Accuracy Score of Random Forest Classifier: {accuracy_score(pred2,test_y)*100}%")
cm = confusion_matrix(test_y, pred2, labels = rfc.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rfc.classes_)
disp.plot()
# fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 8))
# rfc_disp = RocCurveDisplay.from_estimator(rfc, test_x, test_y, ax=ax1)
# prec, recall, _ = precision_recall_curve(test_y, pred2, pos_label=rfc.classes_[1])
# pr_display = PrecisionRecallDisplay(precision=prec, recall=recall)

print(classification_report(test_y, pred2, target_names=["suitable", "well suitable", "excellent suitable"]))
```

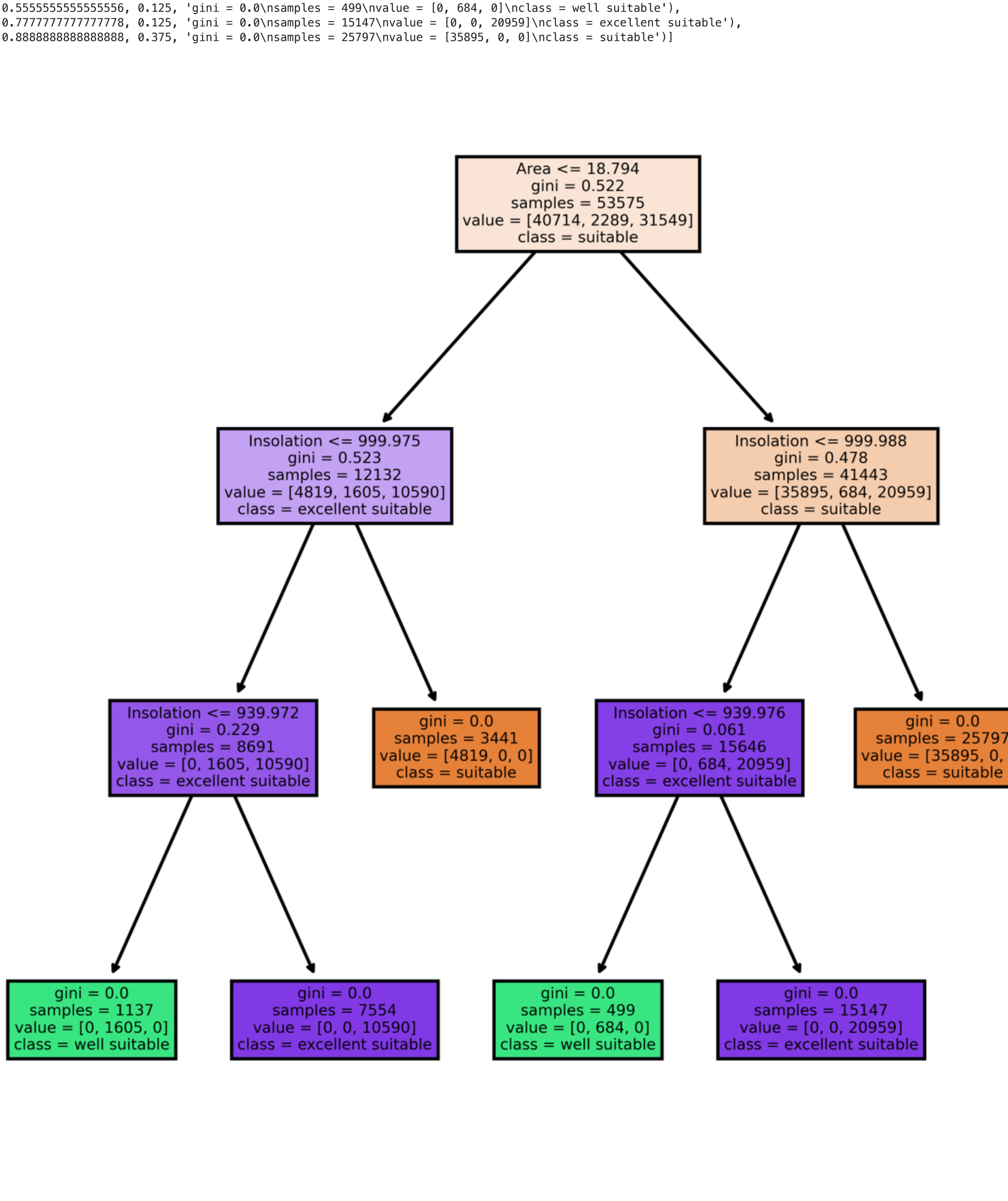
```
Accuracy Score of Random Forest Classifier: 100.0%
                    precision    recall  f1-score   support

          suitable       1.00      1.00      1.00     14681
     well suitable       1.00      1.00      1.00       832
excellent suitable       1.00      1.00      1.00     11113

          accuracy                           1.00     26626
         macro avg       1.00      1.00      1.00     26626
      weighted avg       1.00      1.00      1.00     26626
```



Visualisation of an Individual Tree in RFC

```python
from sklearn import tree

fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (6,6), dpi=800)
tree.plot_tree(rfc.estimators_[0],
               feature_names = ["Insolation","Area"],
               class_names=["suitable","well suitable", "excellent suitable"],
               filled = True)
# fig.savefig('rf_individualtree.png')
```

```
Out[ ]: [Text(0.5555555555555556, 0.875, 'Area <= 18.794\ngini = 0.522\nsamples = 53575\nvalue = [40714, 2289, 31549]\nclass = suitable'),
 Text(0.3333333333333333, 0.625, 'Insolation <= 999.975\ngini = 0.523\nsamples = 12132\nvalue = [4819, 1605, 10590]\nclass = excellent suitable'),
 Text(0.2222222222222222, 0.375, 'Insolation <= 939.972\ngini = 0.229\nsamples = 8691\nvalue = [0, 1605, 10590]\nclass = excellent suitable'),
 Text(0.1111111111111111, 0.125, 'gini = 0.0\nsamples = 1137\nvalue = [0, 1605, 0]\nclass = well suitable'),
 Text(0.3333333333333333, 0.125, 'gini = 0.0\nsamples = 7554\nvalue = [0, 0, 10590]\nclass = excellent suitable'),
 Text(0.4444444444444444, 0.375, 'gini = 0.0\nsamples = 3441\nvalue = [4819, 0, 0]\nclass = suitable'),
 Text(0.7777777777777778, 0.625, 'Insolation <= 999.988\ngini = 0.478\nsamples = 41443\nvalue = [35895, 684, 20959]\nclass = suitable'),
 Text(0.6666666666666666, 0.375, 'Insolation <= 939.976\ngini = 0.061\nsamples = 15646\nvalue = [0, 684, 20959]\nclass = excellent suitable'),
 Text(0.5555555555555556, 0.125, 'gini = 0.0\nsamples = 499\nvalue = [0, 684, 0]\nclass = well suitable'),
 Text(0.7777777777777778, 0.125, 'gini = 0.0\nsamples = 15147\nvalue = [0, 0, 20959]\nclass = excellent suitable'),
 Text(0.8888888888888888, 0.375, 'gini = 0.0\nsamples = 25797\nvalue = [35895, 0, 0]\nclass = suitable')]
```



Code for first SVR

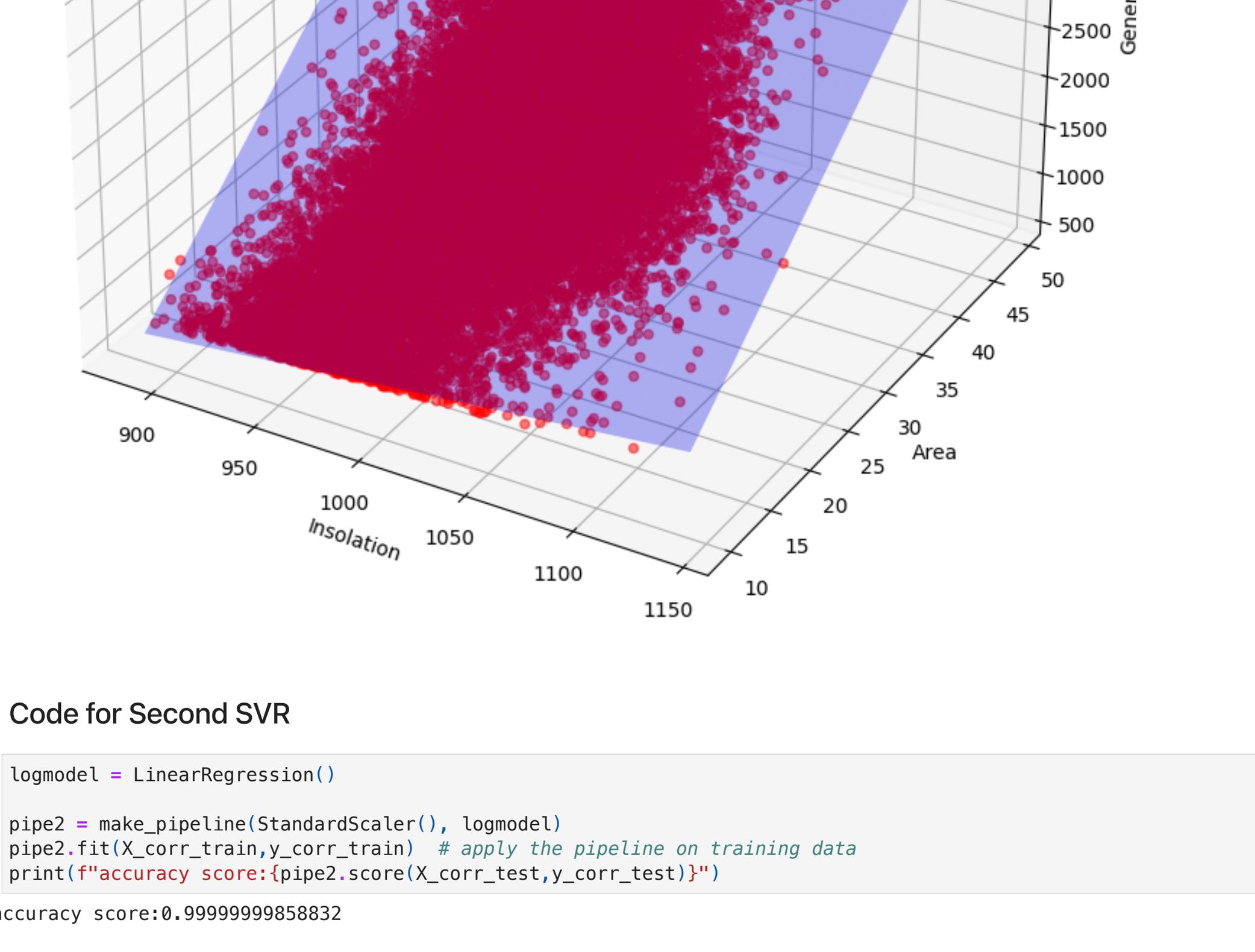
```python
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

logmodel = LinearRegression()

pipe = make_pipeline(StandardScaler(), logmodel)
pipe.fit(X_gen_train,y_gen_train)  # apply the pipeline on training data
print(f"accuracy score:{pipe.score(X_gen_test,y_gen_test)}")
```

```
accuracy score:0.9988942272462243
```

Code to Display Regression Line

```python
x_surf, y_surf = np.meshgrid(np.linspace(read_full.Insolation.min(), read_full.Insolation.max(), 10), np.linspace(read_full.Area.min(), read_full.Area.max(), 10))
onlyX = pd.DataFrame({'Insolation':x_surf.ravel(), 'Area':y_surf.ravel()})
fittedY=pipe.predict(onlyX)
fittedY=np.array(fittedY)

print(f"Insolation max:{read_full.Insolation.max()}/min: {read_full.Insolation.min()}")
print(f"Area max:{read_full.Area.max()}/min: {read_full.Area.min()}")
print(f"Generation max:{read_full.Generation.max()}/min: {read_full.Generation.min()}")

# readfile["Generation"].plot.line()

from mpl_toolkits.mplot3d import Axes3D


fig = plt.figure(figsize=(20,10))
### Set figure size
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_gen_test['Insolation'],X_gen_test['Area'],y_gen_test,c='red', marker='o', alpha=0.5)
ax.plot_surface(x_surf,y_surf,fittedY.reshape(x_surf.shape), color='b', alpha=0.3)
ax.set_xlabel('Insolation')
ax.set_ylabel('Area')
ax.set_zlabel('Generation')
plt.show()
```

```
Insolation max:1142.3/min: 884.6154
Area max:48.7909/min: 10.0
Generation max:3999.8745/min: 801.36
```



Code for Second SVR

```python
logmodel = LinearRegression()

pipe2 = make_pipeline(StandardScaler(), logmodel)
pipe2.fit(X_corr_train,y_corr_train)  # apply the pipeline on training data
print(f"accuracy score:{pipe2.score(X_corr_test,y_corr_test)}")
```

```
accuracy score:0.99999999858832
```

Takes in geographical data as well as insolation and outputs predicted power generation as well as coordinates as csv

```python
pred_pd = pd.read_csv("Uk_Insolation_data.csv")
pred_pd = pred_pd[["Lat","Long","Insolation"]]
print(pred_pd.head())
pred=pipe2.predict(pred_pd)
print(type(pred))
output = pd.DataFrame(pred)
print(pred)
pred_pd = pred_pd.drop('Insolation', axis=1)
output = pd.concat([pred_pd,output], axis=1)
output.to_csv("Uk_generation_output.csv")
```

```
         Lat      Long  Insolation
0  54.862330 -2.391998   1005.3090
1  57.271658 -4.075642    952.1060
2  52.234476 -4.272127   1147.0376
3  54.434273 -6.088879   1047.6035
4  50.992662 -3.070938   1158.9788
<class 'numpy.ndarray'>
[3412.92177839 3237.09221907 3881.31786988 ... 3145.90659937 3870.96632316
 3442.37388254]
/usr/local/lib/python3.10/site-packages/sklearn/base.py:413: UserWarning: X has feature names, but StandardScaler was fitted without feature names
  warnings.warn(
```