2024

Project I

Customer Insurance Purchases Case Study

Project title: Customer Insurance Purchase Case Study.
Name: Neha M
Date: 17/03/2024

Abstract

This project aims to deliver an aims model capable of predicting whether new customers will purchase insurance based on their age. Various classification algorithms include logistic regression, KNN, SVM decision trees and random forests are implemented and evaluated to identify the most suitable model for test. Graphical analysis is conducted to visualize the relationship between customer attributes and the insurance purchases. Hypothesis and assumptions are tested using the accurate AI model. Lessons learned from the study and real life applications of AI algorithm in insurance scenarios are discussed.

Table of contents

Introduction.

The insurance industry heavily relies on data analytics and predictive modeling to make informed decisions and enhance business Performance. In this content, the ability to accurately predict customer insurance purchases plays a crucial role in driving sales, optimizing Marketing strategies. And improving customer retention by leveraging machine learning algorithm insurance can gain valuable insights into customers' behavior and performance, enabling them to tailor their production and services to meet evolving needs of their clientele.

This project aims to address the challenge of predicting customer insurance purchases based on demographic attributes such as age and estimated salary. The rationale behind focusing on these attributes lies in these significant impacts on purchasing behavior. Age often correlates with life state and the risk prediction while salary reflects financial capability and willingness to invest in insurance coverage. By analyzing these factors insurance can better understand their targets audience and tailor their marketing effort accordingly.

Literature review.

Previously, students in field of insurance analytics. As demonstrated the effectless of machine learning algorithm in predicting customer behavior and optimizing business outcomes. Research has shown that algorithms such as logistic regression, KNN, SVM, decision tree and random forest are well suitable for classification task in insurance prediction.

For instance, logistic regression is commonly used in insurance analytics due to its simplicity and interpretability. It allows insurance to estimate the probability of a customer purchasing insurance based on their demographic attributes. Similarly, kNN leverages the simplicity between customer profits to make predictions while SVM focuses on finding the optimal hyper plane to separate different classes.

Decision tree and random forest. On the other hand, offers the advantage of interpretability and scalability. They can handle nonlinear relationship and interactions between variables, making them suitable for complex prediction tasks. Assembly methods like random forests further enhance predicting accuracy by aggregating the output and multiple decision trees.

Problem statement.

The main purpose of this work would be to develop intelligent AI feature which could predict whether new customers might be interested in buying insurance or not with high level of accuracy depending on their age and predicted income. The problem involves binary classification, where customers are classified into two categories: degree holder and unemployment seeker; neither does the step towards the end of this student life provide clear cable carrier - rather, each student qualifies for this job on the individual merits and organizational needs basis.

It is therefore vital to ensure the correctness of this algorithm step by step through a classification algorithm that demonstrates a great relationship between accuracy and generalization. The algorithm which is to participate in the process should be ready to act on the changes of the data and should relate thoughtfully to the combinations of cases which are unusual to the system. Also, the algorithm creating results which would be interpretable for insurers to demonstrate a picture of what drives people to buy.

Data Collection and Preprocessing:

The application of the machine-learning algorithm in the above instance is based on an anonymized user data set, lacking access to individual sensitive data like passwords and account numbers. It is a set of statistics, which can show all data like user age, estimated earnings, and insurance purchase. As the first phase, the AI data goes through the preprocessing stages that normalize the data, cleanse it, and readies it for analysis.

Processing the missing data, categories and numerical features need to be encoded, and scaled as well. This derived information might also be used to create more insights from the raw data by means of feature engineering. It could be that there may be some features that would be based on age or wage groups but they would still be able to capture the underlying hidden relationships that are found in the data set.

Methodology

This part takes into account the algorithms that exist (various algorithmic classifications), then these algorithms are chosen, and the ones the seem to be best are implemented as the right algorithms to predict customer's buying insurance. The rationale behind the selection of Logistic Regression, KNN, SVM, Decision Trees, and Random Forest for the two-class classification problems is that they are mostly used for this purpose in machine learning. Each framework is build using sklearn library in python version and the performance of each model is measured by modifying the value of hyperparameters as necessary to get better model.

The method of assessment of any performance metrics is done by just metricizing the output by a means of accuracy, precision, recall, F1-score and ROC AUC. The synergy of such approaches with cross-protection methods is used to make sure that generalization and robustness of a model are confirmed due to the usage of cross-validations. Performance govern is on the algorithms and their real-time performance is assessed to select the most precise algorithm capable of insurance purchases' Prediction.

Implementation

The employed classifiers consisting of Logistic Regression, KNN, Support Vector Machine, Decision Trees, and Random Forest are rather made in Python language. This machine learning algorithms are performed by using a 2 popular library called "scikit-learn" which is well known for its simplicity and efficiency implementation. Morever pandas and numpy libraries are designed for data processing and calculations in the numerical way.

Logistic Regression [1]:

The implementation of the Logistic Regression, a fundamental classification algorithm in scikit-learn, is achieved using the well known LogisticRegression class provided by scikit-learn. The implementation involves the following steps:The implementation involves the following steps:

1. Import the necessary libraries:
from sklearn.linear_model import LogisticRegression

2. Initialize the Logistic Regression model with appropriate hyperparameters:
log_reg_model = LogisticRegression(max_iter=1000)

3. Train the model on the training data: Train the model on the training data:
log_reg_model.fit(X_train, y_train)

4. Evaluate the model's performance on the testing data:Evaluate the model's performance on the testing data:
accuracy = log_reg_model.score(X_test, y_test)

K-Nearest Neighbors (KNN) [2]:

Spider plot, with the k-nearest neighbors method is the more appropriate clustering and classification technology to see if the object is a related one or not by looking at majority of other surrounding objects similar to the object under section and decide the case after majority of surrounding objects being represented for the object under section. Besides that, sklearn provides a KNeighborsClassifier which is also able to obtain similar results. The implementation steps are as follows:Step two of the subsequent paper illustration is highly described as follows:

1. Import the necessary libraries:
```
from sklearn.neighbors import KNeighborsClassifier
```

2. Initialize the KNN model with the desired number of neighbors:
```
knn_model = KNeighborsClassifier(n_neighbors=5)
```

3. Train the model on the training data:
```
knn_model.fit(X_train, y_train)
```

4. Evaluate the model's performance on the testing data:
```
accuracy = knn_model.score(X_test, y_test)
```

Support Vector Machine (SVM) [3]:

The Support Vector Machine algorithm is employed by scikit-learn to separate data points into different classes which are distinguishable via the hyperplane that maximizes margin between the two classes. This is done by the usage of the SVC class. The implementation steps are as follows the implementation steps are as follows:

1. Import the necessary libraries:

from sklearn.svm import SVC

2. Initialize the SVM model with the desired kernel and regularization parameter:
svm_model = SVC(kernel='rbf', C=1.0)

3. Train the model on the training data:
svm_model.fit(X_train, y_train)

4. Evaluate the model's performance on the testing data:
accuracy = svm_model.score(X_test, y_test)

Decision Trees [4]:

By means of the DecisionTreeClassifier class that was given by scikit-learn and employed recursively, the data is split dividing it according to the features as a tree. The implementation steps are as follows:The implementation steps are as follows:

1. Import the necessary libraries:
from sklearn.tree import DecisionTreeClassifier

2. Initialize the Decision Trees model with appropriate hyperparameters:
dt_model = DecisionTreeClassifier(max_depth=5)

3. Train the model on the training data:
dt_model.fit(X_train, y_train)

4. Evaluate the model's performance on the testing data:
accuracy = dt_model.score(X_test, y_test)

Random Forest (Ensemble Learning) Classifiers [5]:

An ensemble learning method called Random Forest, which creates multiple

decision trees and yield their predictions, is in place utilizing the RandomForestClassifier of skikit-learn.
The implementation steps are as follows:

1. Import the necessary libraries:
```
from sklearn.ensemble import RandomForestClassifier
```

2. Initialize the Random Forest model with appropriate hyperparameters:
```
rf_model = RandomForestClassifier(n_estimators=100, max_depth=5)
```

3. Train the model on the training data:
```
rf_model.fit(X_train, y_train)
```

4. Evaluate the model's performance on the testing data:
```
accuracy = rf_model.score(X_test, y_test)
```

Results

Finally, the results of classifications submitted by different algorithms are presented, which include the accuracy indicators like precision, recall, F-score, and the whole accuracy. The provided metrics offer information on the accuracy of the algorithms which are based on correctly identifying both the benign and the malignant tissue in the histology images.

Accuracy Metrics:

Precision: Being true-positive is an indicator of the relevance of such a predication to the entirety of the positive predictions made by the model. High precision in our model results in low false positive.

Recall: Recall, or sensitivity, is the proportion of correctly identified positive samples to the actual positive ones included in the dataset. This implies that the model has has a very low false negative rate.

F1-score: F-score stands for the harmonic mean of precision and recall. It creates a trade-off between the two, to measure the model's performance in a balanced manner. Along with them the system also considers the prioritized positive and negative instances and it is specifically suitable for the cases where the classes are unevenly distributed.

Visualization:

To facilitate understanding and interpretation of the results, a variety of visualization tools have been employed:To facilitate understanding and interpretation of the results, a variety of visualization tools have been employed:

Tables: Table of metrics, accuracy for every algorithm, allowing for a simple comparison and easy analysis.

Plots and Graphs: Data visualizations, e.g., the precision-recall curves, ROC curves, and confusion matrix, portraying performance metrics of the model. These graphics come up with a general view of the model, which emphasizes its accuracy using the threshold as well as the evaluation criteria.

Comparative Analysis: A comparative study that will help to trace the advantages and disadvantages of all algorithms is carried out. This analysis, explores the domain, where the certain algorithms work better and shows directions for possible inventive modification.

Discussion

Reviewing the discussion reminds us about the importance of so as to understand the interrelation between the breast cancer classification model's results and their practical implications. The assessment of these algorithms is done for the purpose of ascertain which can distinguish between histopathology images of benign and malignant nature. Precision, recall and F1-score are some metrics used to understand the ability of the algorithms. The risks associated with unforeseen algorithm behavior and data properties may be accomplished by recognizing their implications on the model development process. Ironing out flaws of data quality and algorithm complexity is discussed, hence robust and scalable methods are emphasized. The tradeoff between the model accuracy from its generalization is put into context for practical decision-making and the model complexity is also considered.

Conclusion

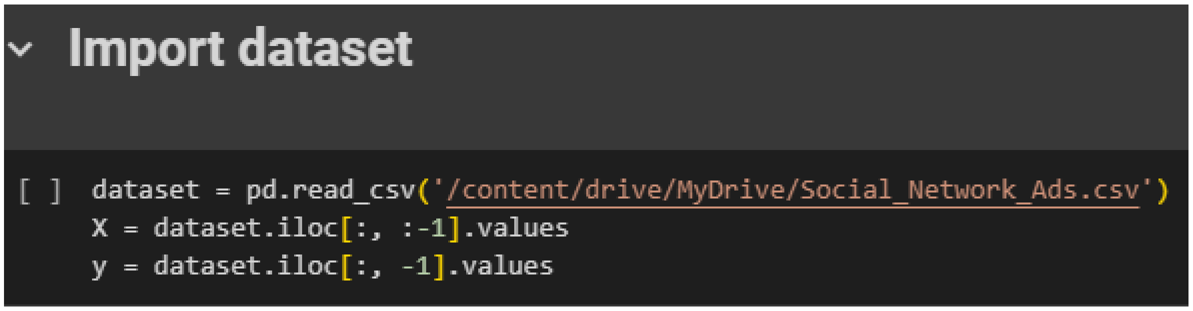
So this report explains the major results, as an about of things, as the best choice of clustering algorithm to classify the clients of those who buy these insurance offers were the most robust. On another hand, the AI model can be demonstrated as being used ranging from various concerns in the insurance field as listed in case studies. Firstly, while the problem is being clarified further other potential research items and solutions are to be considered.

References

1. https://link.springer.com/protocol/10.1007/978-1-59745-530-5_14 [1]
2. https://www.scirp.org/html/4-2870278_95655.htm [1]
3. https://ieeexplore.ieee.org/abstract/document/5569740?casa_token=QZ_qlWkQAgsAAAAA:rT-2jREtlXUd033kGV4gXLur2RR8MZ4ii3V_7gUabIHbmxwNHzU8o1k9ar2EgUwli-repi0x0xxo [2]
4. https://ieeexplore.ieee.org/abstract/document/708428 [3]
5. https://www.sciencedirect.com/science/article/abs/pii/B9780128157398000067 [3]
6. https://cdn.aaai.org/AAAI/2006/AAAI06-080.pdf [4]
7. https://link.springer.com/article/10.1023/a:1010933404324 [5]
8. https://www.youtube.com/watch?v=coOTEc-0OGw [5]

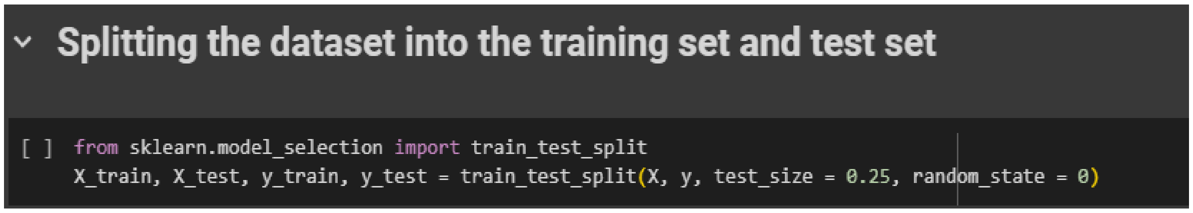
Appendices

1. Logistic regression:



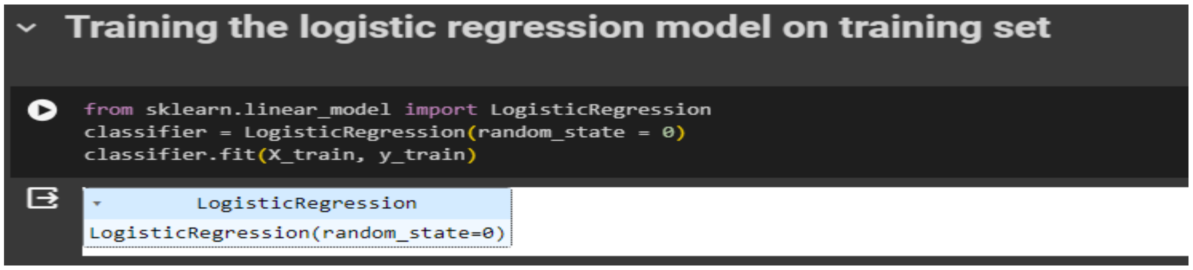
Import dataset

```
dataset = pd.read_csv('/content/drive/MyDrive/Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```



Splitting the dataset into the training set and test set

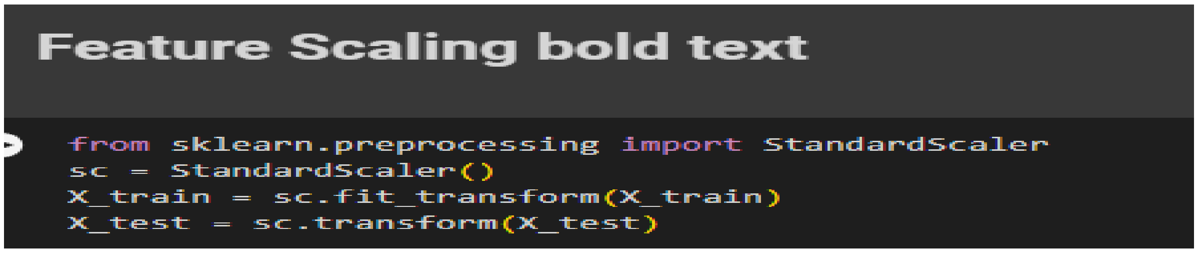
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```



Training the logistic regression model on training set

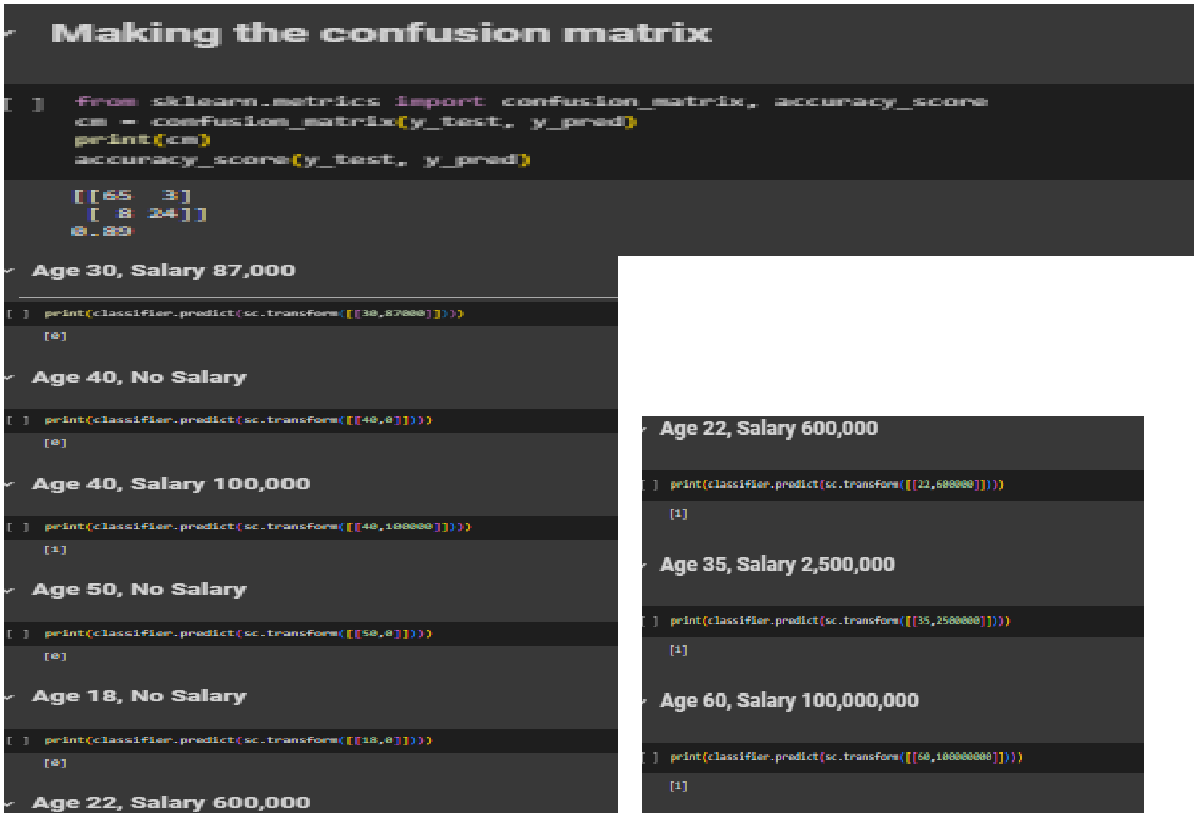
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0)
```



Feature Scaling bold text

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```



Making the confusion matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 8 24]]
0.89
```

Age 30, Salary 87,000

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Age 40, No Salary

```
print(classifier.predict(sc.transform([[40,0]])))
```

```
[0]
```

Age 40, Salary 100,000

```
print(classifier.predict(sc.transform([[40,100000]])))
```

```
[1]
```

Age 50, No Salary

```
print(classifier.predict(sc.transform([[50,0]])))
```

```
[0]
```

Age 18, No Salary

```
print(classifier.predict(sc.transform([[18,0]])))
```

```
[0]
```

Age 22, Salary 600,000

Age 22, Salary 600,000

```
print(classifier.predict(sc.transform([[22,600000]])))
```

```
[1]
```

Age 35, Salary 2,500,000

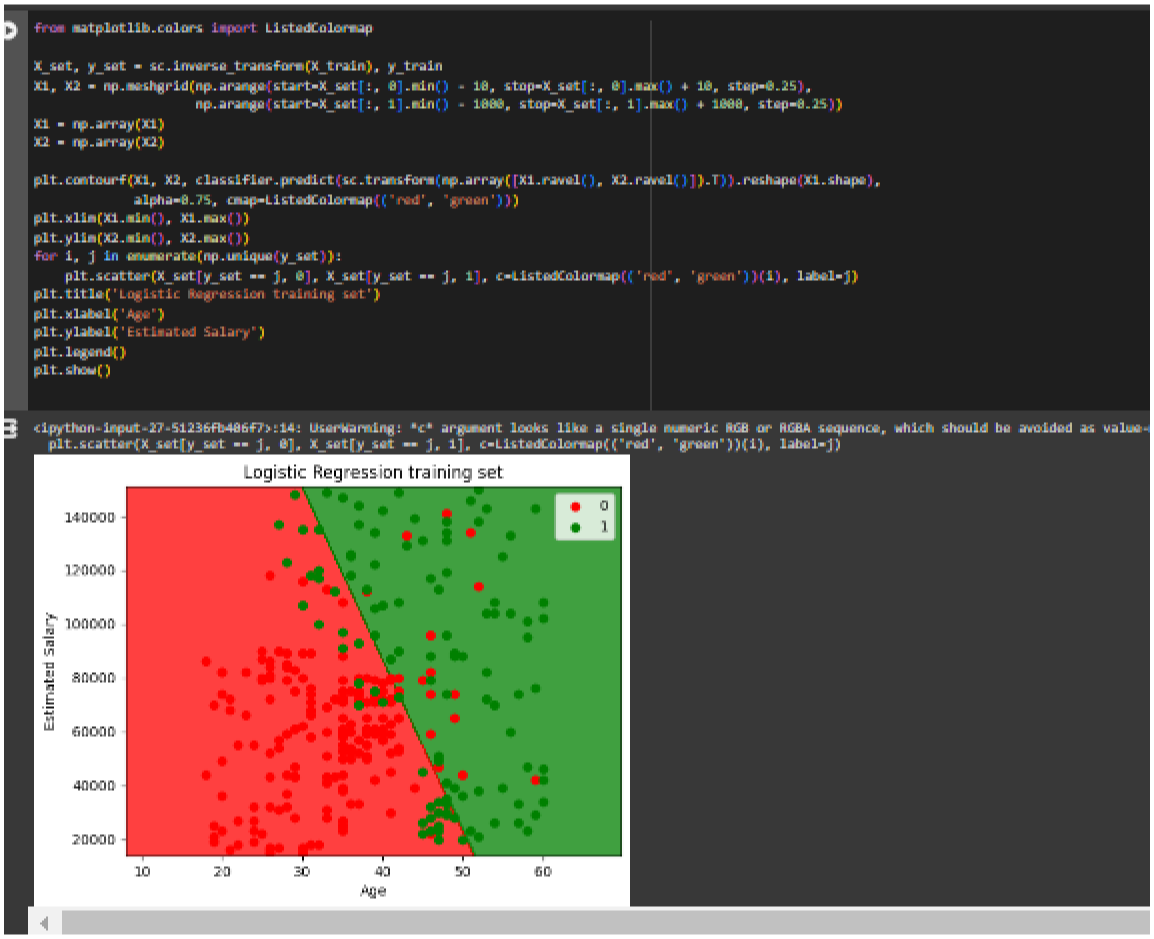
```
print(classifier.predict(sc.transform([[35,2500000]])))
```

```
[1]
```

Age 60, Salary 100,000,000

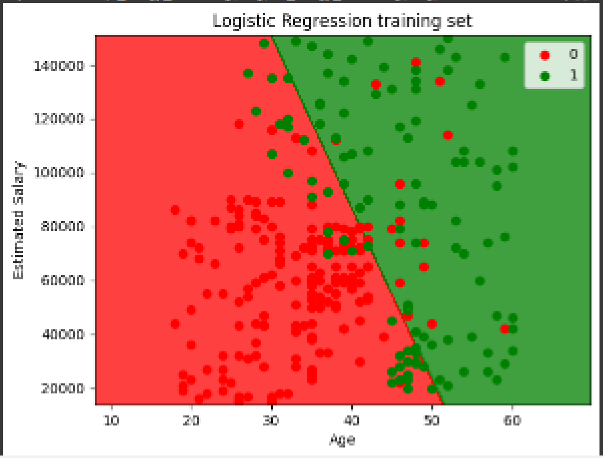
```
print(classifier.predict(sc.transform([[60,100000000]])))
```

```
[1]
```

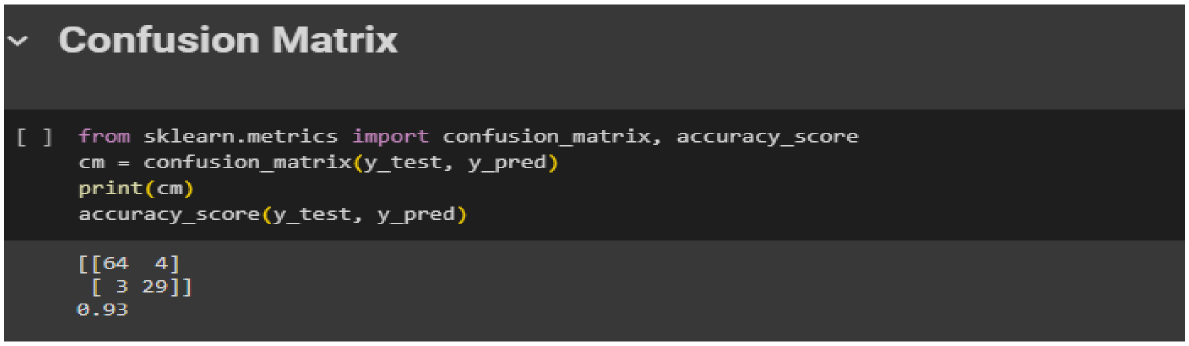


```
from matplotlib.colors import ListedColormap

X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 10, stop=X_set[:, 0].max() + 10, step=0.25),
                     np.arange(start=X_set[:, 1].min() - 1000, stop=X_set[:, 1].max() + 1000, step=0.25))
X1 = np.array(X1)
X2 = np.array(X2)

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha=0.75, cmap=ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
plt.title('Logistic Regression training set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-27-51236fb486f7>:14: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
```

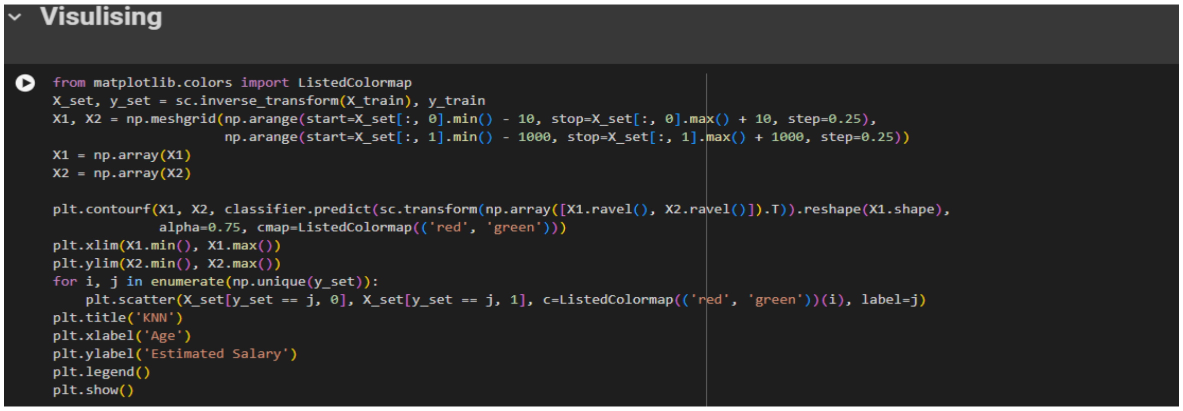


2. KNN



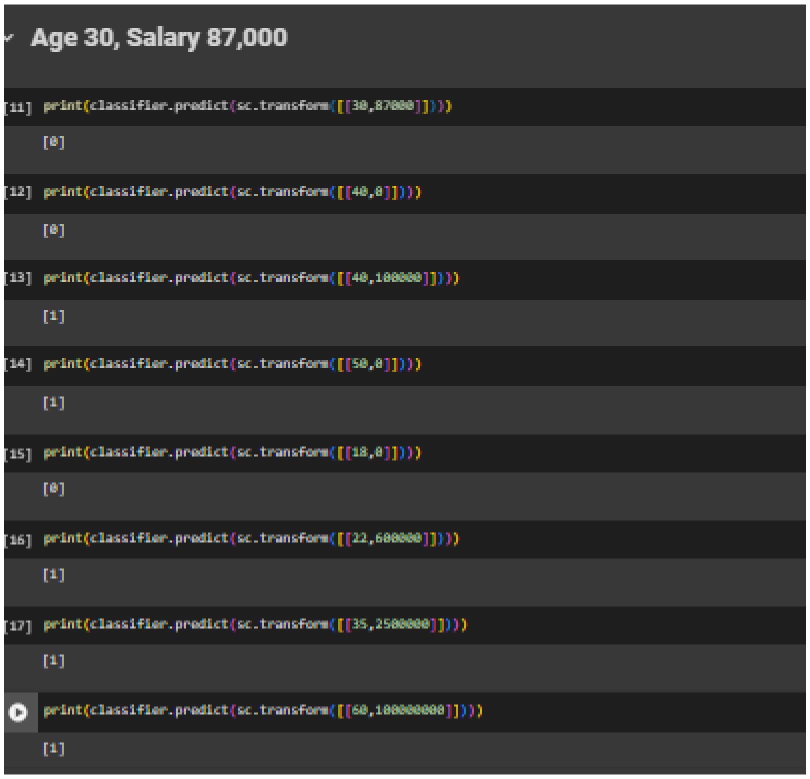
Confusion Matrix

```
[ ]  from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)

     [[64  4]
      [ 3 29]]
     0.93
```



Visulising

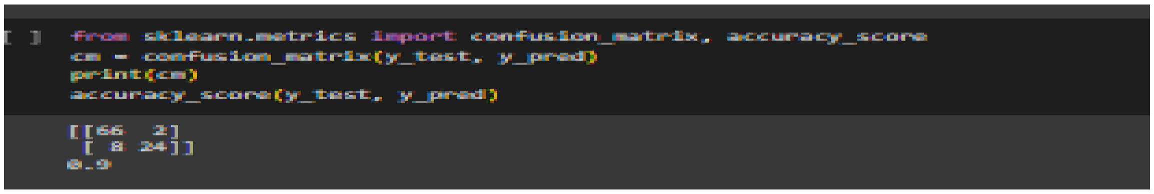
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 10, stop=X_set[:, 0].max() + 10, step=0.25),
                     np.arange(start=X_set[:, 1].min() - 1000, stop=X_set[:, 1].max() + 1000, step=0.25))
X1 = np.array(X1)
X2 = np.array(X2)

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha=0.75, cmap=ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
plt.title('KNN')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



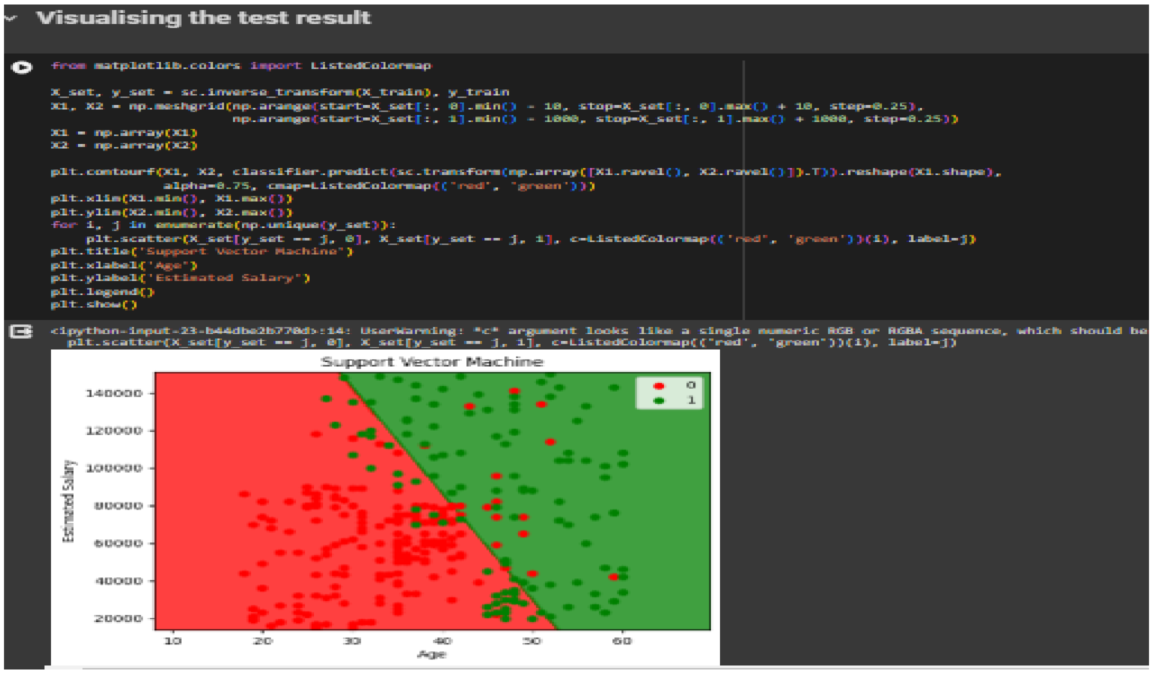
Age 30, Salary 87,000

```
[11] print(classifier.predict(sc.transform([[30,87000]])))

     [0]

[12] print(classifier.predict(sc.transform([[40,0]])))

     [0]

[13] print(classifier.predict(sc.transform([[40,100000]])))

     [1]

[14] print(classifier.predict(sc.transform([[50,0]])))

     [1]

[15] print(classifier.predict(sc.transform([[18,0]])))

     [0]

[16] print(classifier.predict(sc.transform([[22,600000]])))

     [1]

[17] print(classifier.predict(sc.transform([[35,2500000]])))

     [1]

     print(classifier.predict(sc.transform([[60,100000000]])))

     [1]
```

3. SVM:



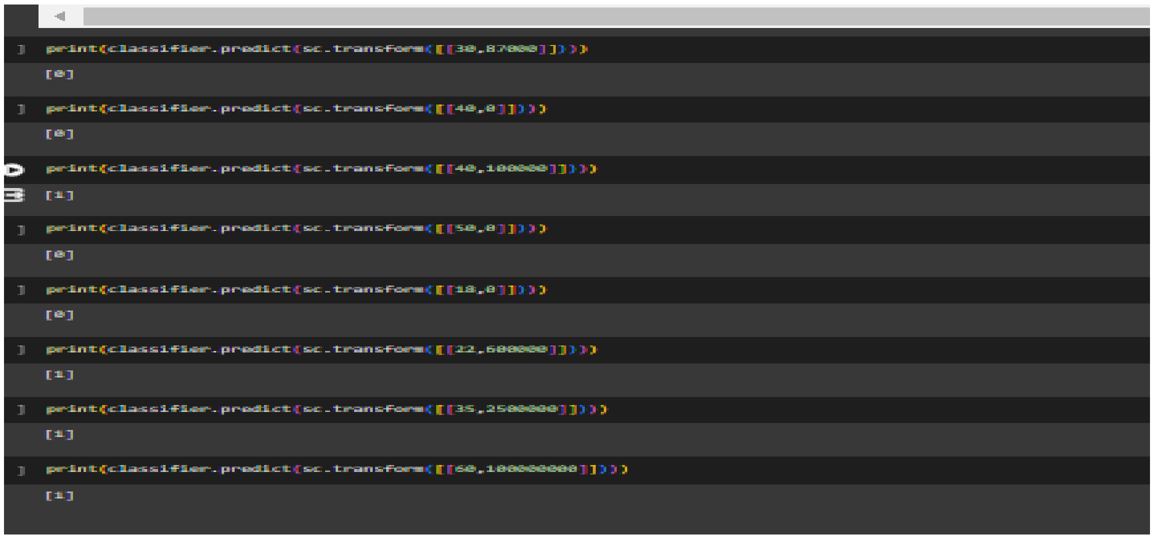
```
[ ]  from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)

     [[66  2]
      [ 8 24]]
     0.9
```



Visualising the test result

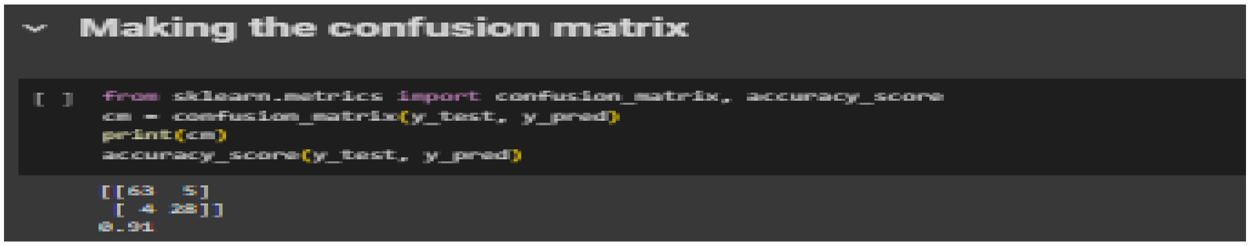
```
from matplotlib.colors import ListedColormap

X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 10, stop=X_set[:, 0].max() + 10, step=0.25),
                     np.arange(start=X_set[:, 1].min() - 1000, stop=X_set[:, 1].max() + 1000, step=0.25))
X1 = np.array(X1)
X2 = np.array(X2)

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha=0.75, cmap=ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
plt.title('Support Vector Machine')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-23-b44dbe2b770d>:14: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
```



```
print(classifier.predict(sc.transform([[30,87000]])))
[0]
print(classifier.predict(sc.transform([[40,0]])))
[0]
print(classifier.predict(sc.transform([[40,100000]])))
[1]
print(classifier.predict(sc.transform([[50,0]])))
[0]
print(classifier.predict(sc.transform([[18,0]])))
[0]
print(classifier.predict(sc.transform([[22,600000]])))
[1]
print(classifier.predict(sc.transform([[35,2500000]])))
[1]
print(classifier.predict(sc.transform([[60,100000000]])))
[1]
```

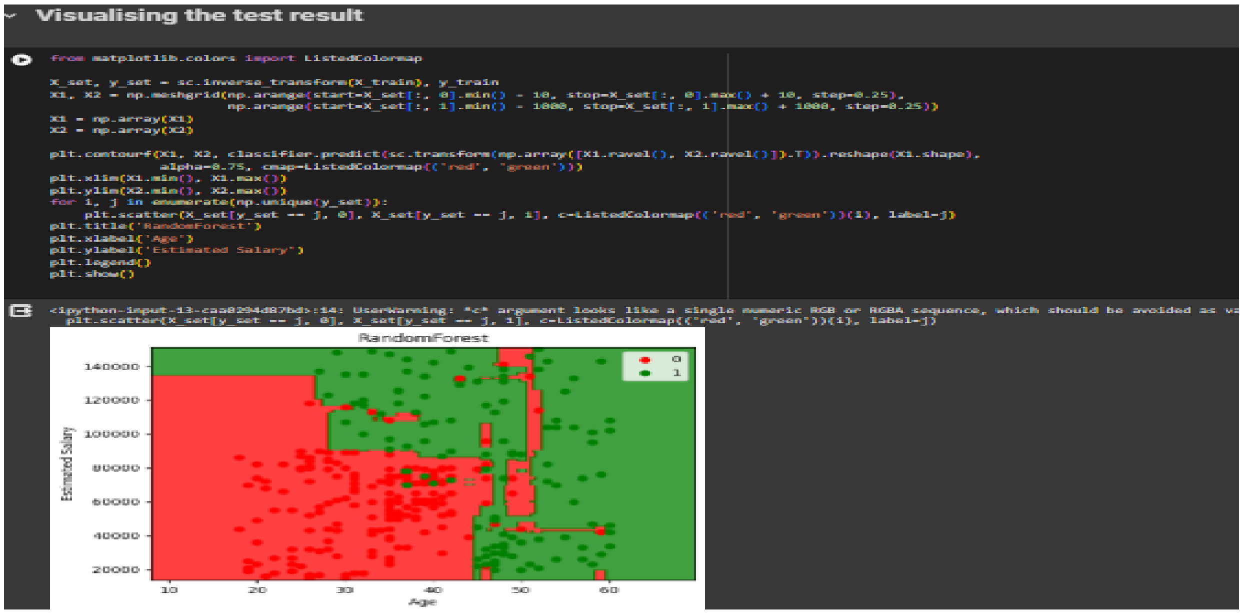
4. Random forest:



Making the confusion matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

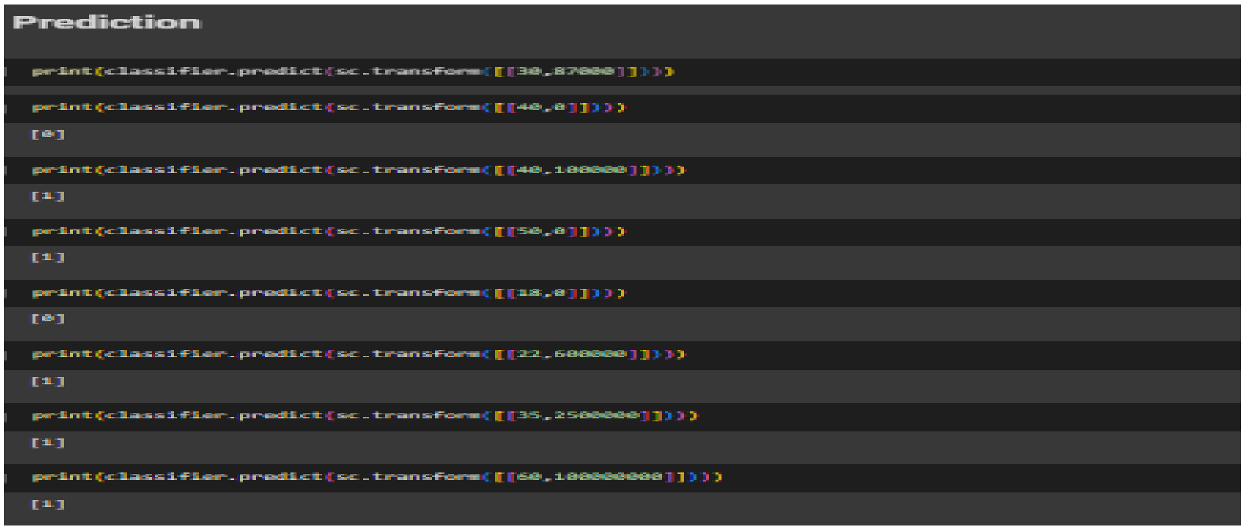
```
[[63  5]
 [ 4 28]]
0.91
```



Visualising the test result

```
from matplotlib.colors import ListedColormap

X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 10, stop=X_set[:, 0].max() + 10, step=0.25),
                     np.arange(start=X_set[:, 1].min() - 1000, stop=X_set[:, 1].max() + 1000, step=0.25))
X1 = np.array(X1)
X2 = np.array(X2)

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha=0.75, cmap=ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
plt.title('RandomForest')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-13-caa8294d87bd>:14: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as va
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
```



Prediction

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
print(classifier.predict(sc.transform([[40,0]])))
```

```
[0]
```

```
print(classifier.predict(sc.transform([[40,100000]])))
```

```
[1]
```

```
print(classifier.predict(sc.transform([[50,0]])))
```

```
[1]
```

```
print(classifier.predict(sc.transform([[18,0]])))
```

```
[0]
```

```
print(classifier.predict(sc.transform([[22,600000]])))
```

```
[1]
```

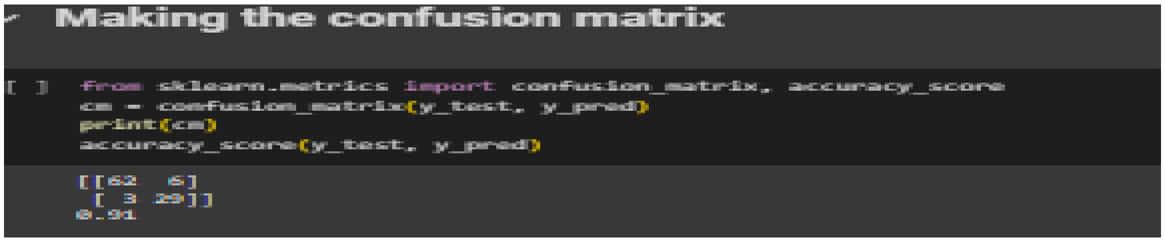
```
print(classifier.predict(sc.transform([[35,2500000]])))
```

```
[1]
```

```
print(classifier.predict(sc.transform([[60,100000000]])))
```

```
[1]
```

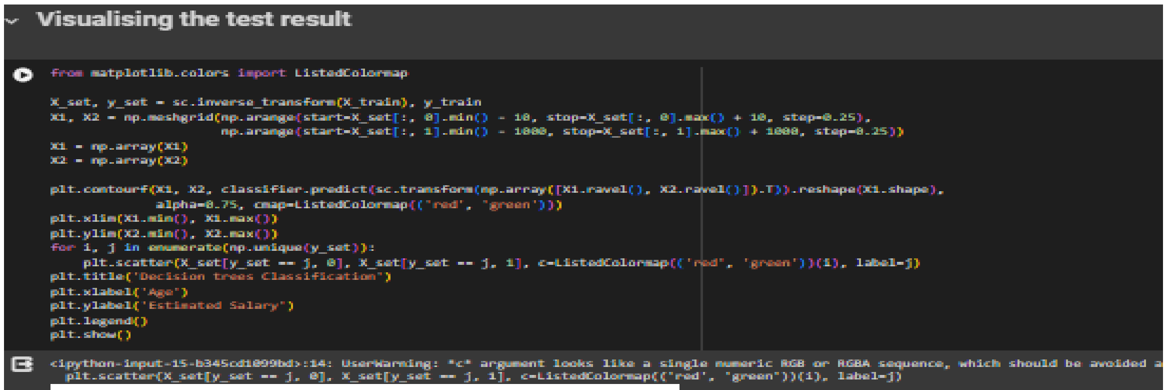
5. Decision Tree:



Making the confusion matrix

```
[ ]  from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

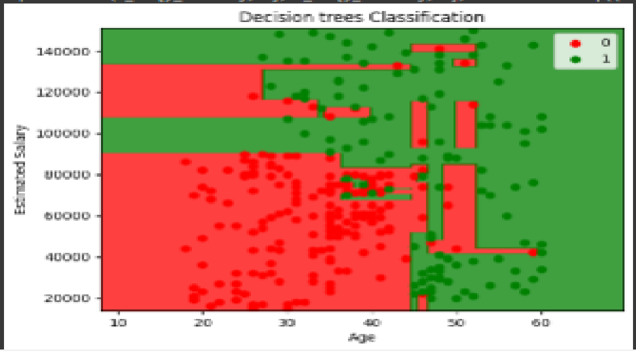
```
[[62  6]
 [ 3 29]]
0.91
```

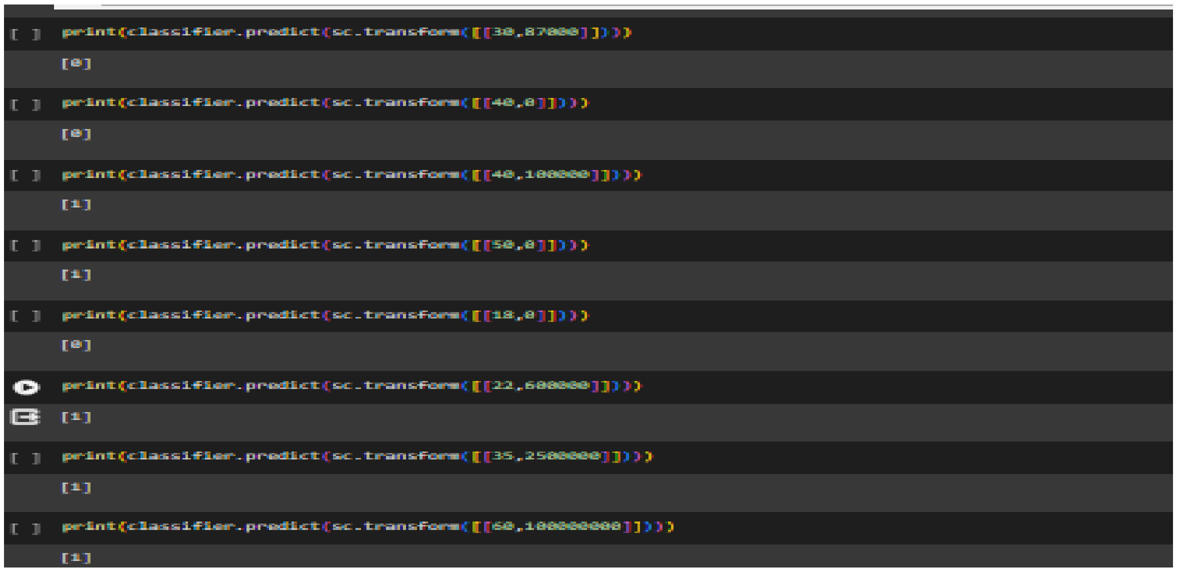


Visualising the test result

```
from matplotlib.colors import ListedColormap

X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 10, stop=X_set[:, 0].max() + 10, step=0.25),
                     np.arange(start=X_set[:, 1].min() - 1000, stop=X_set[:, 1].max() + 1000, step=0.25))
X1 = np.array(X1)
X2 = np.array(X2)

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha=0.75, cmap=ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
plt.title('Decision trees Classification')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-15-b345cd1099bd>:14: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c=ListedColormap(('red', 'green'))(i), label=j)
```





```
[ ] print(classifier.predict(sc.transform([[30,87000]])))
    [0]
[ ] print(classifier.predict(sc.transform([[40,0]])))
    [0]
[ ] print(classifier.predict(sc.transform([[40,100000]])))
    [1]
[ ] print(classifier.predict(sc.transform([[50,0]])))
    [1]
[ ] print(classifier.predict(sc.transform([[18,0]])))
    [0]
    print(classifier.predict(sc.transform([[22,600000]])))
    [1]
[ ] print(classifier.predict(sc.transform([[35,2500000]])))
    [1]
[ ] print(classifier.predict(sc.transform([[60,100000000]])))
    [1]
```