Machine learning

Manreet Kaur

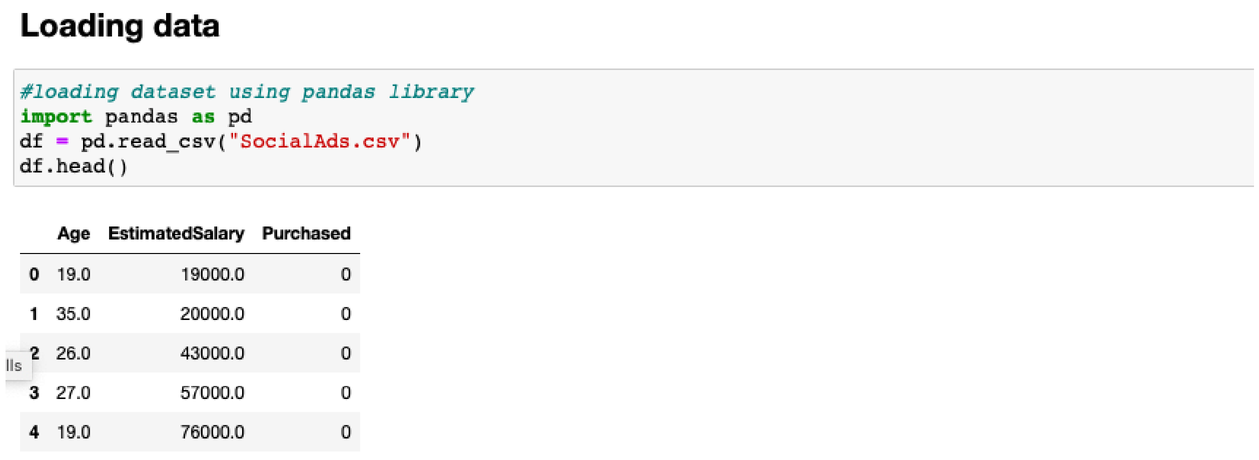
Introduction:

In this assignment a dataset about a company was given with attributes Age, Estimated salary and Purchased. Using this dataset, we need to predict whether the user will purchase the product or not. The 5 Support vector machine models will be developed for predicting the target binary categorical variable that is Purchased. These models will be then evaluated to get the best fit model.

Stepwise formation of machine learning project to get the intended outcomes:

1. **Frame the problem and looking at big picture** – In this lab problem statement and dataset were given as the business objective was to develop a model to predict whether customer will purchase the product of company or not based on their age and estimated salary. The machine learning system which we need to design for this assignment is classification because we are predicting binary categorical variable.

2. **Get the Data:** The dataset in this case was provided by instructor in form of csv file so I just downloaded csv file in my machine and opened in Jupiter notebooks using pandas library in following command in Jupiter notebook:



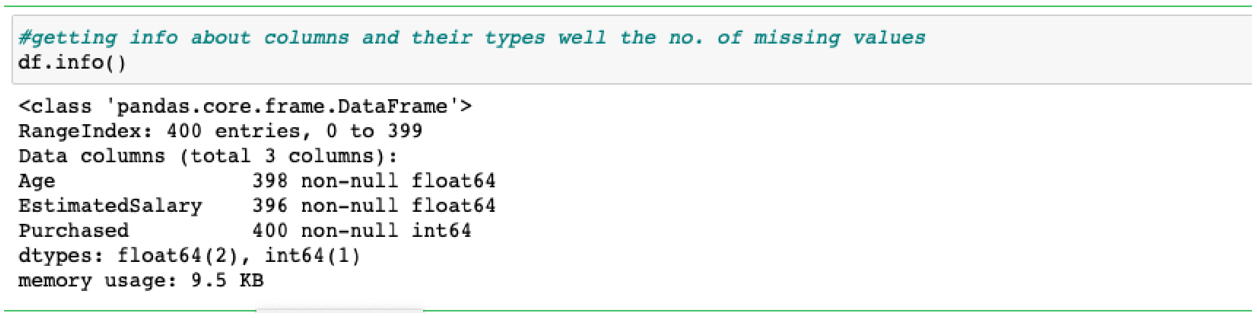
Loading data

```
#loading dataset using pandas library
import pandas as pd
df = pd.read_csv("SocialAds.csv")
df.head()
```

	Age	EstimatedSalary	Purchased
0	19.0	19000.0	0
1	35.0	20000.0	0
2	26.0	43000.0	0
3	27.0	57000.0	0
4	19.0	76000.0	0

After loading the dataset and creating a data frame df using pandas in notebook, now next step is to take a quick look at the structure of dataset. First, we got to know dataset has 3 attributes, out of which both independent attributes are numerical.
Let’s use the info () function to know if there are any null values and types of variables in dataset.

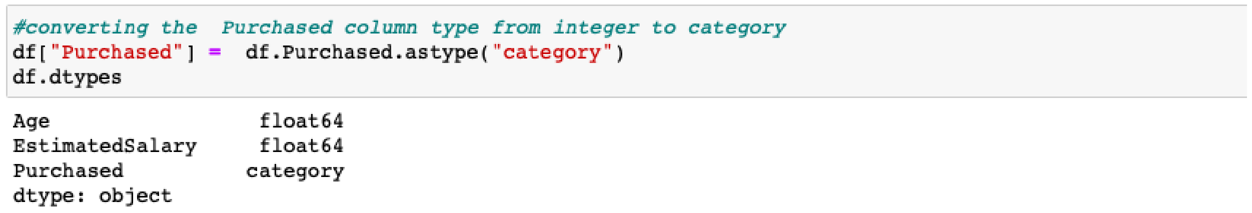
Quick look at data



```
#getting info about columns and their types well the no. of missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
Age                398 non-null float64
EstimatedSalary    396 non-null float64
Purchased          400 non-null int64
dtypes: float64(2), int64(1)
memory usage: 9.5 KB
```

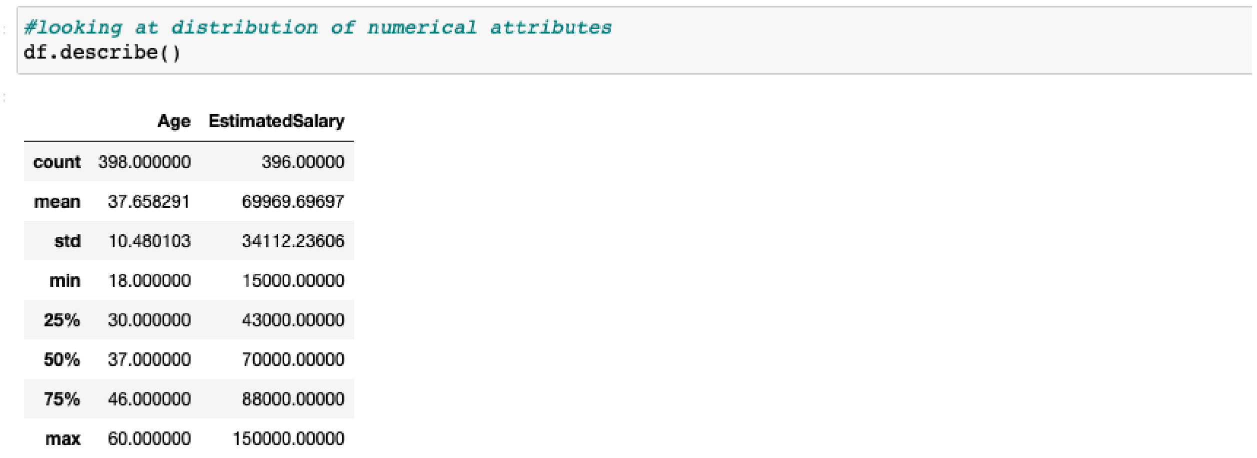
The results shows that the age column has 2 null values and estimated salary column has 4 null values in dataset and the dtype of target attribute that is Purchased is int64 whereas we know it is categorical variable, so we need to convert the dtype for this column to avoid any further problem while developing or evaluating the machine learning model.



```
#converting the  Purchased column type from integer to category
df["Purchased"] =  df.Purchased.astype("category")
df.dtypes
```

```
Age                 float64
EstimatedSalary     float64
Purchased          category
dtype: object
```

Next describe () function was used to get the summary of both numerical variables. In this summary we get the information about null values in specific numerical column and the distribution of these numerical attributes.



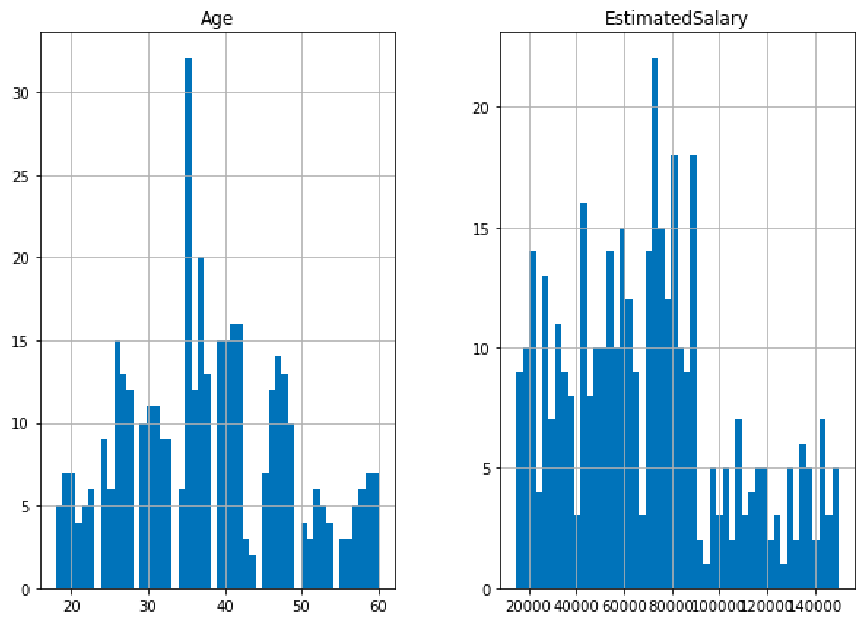
```
#looking at distribution of numerical attributes
df.describe()
```

	Age	EstimatedSalary
count	398.000000	396.00000
mean	37.658291	69969.69697
std	10.480103	34112.23606
min	18.000000	15000.00000
25%	30.000000	43000.00000
50%	37.000000	70000.00000
75%	46.000000	88000.00000
max	60.000000	150000.00000

The above results give insights like 50th percentile shows that 50 percent of people in the dataset are of age less than 37.
For visualization, Jupiter's magic command % was used to plot the histograms for attributes to gain a significant insight.

```
#checking distribution of variables
%matplotlib inline
import matplotlib.pyplot as plt
df.hist(bins=50, figsize=(10,7))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a12580f90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a174b3810>]],
      dtype=object)
```



The results shows that Age column is distributed in form of bell shape in dataset whereas distribution of Estimated salary is bit rightly skewed.
Now let’s check balance in dataset by calculating frequency of binary categories of Purchased column using value_counts( ) function.

```
#checking for balance in class predictor variable in data set
df.Purchased.value_counts()
```

```
0    257
1    143
Name: Purchased, dtype: int64
```

The above results shows that dataset is bit imbalanced. However, this imbalance in dataset is not a major concern over here because the difference in distribution of dataset between both categories is less than 30%. Therefore, the imbalance becomes a major concern if the ratio of distribution between 2 attributes is 80:20 which is not a case here.

After gaining useful insights, next we need to spit the dataset into testing and training using scikit learn library. This library was used because it uses random generator seed so which does not produce different sample set while multiple runs of the program.

Spliting dataset

```
#spliting the dataset into training and testing
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(df, test_size=0.2, random_state=42)
train_set.head(5)
```

	Age	EstimatedSalary	Purchased
3	27.0	57000.0	0
18	46.0	28000.0	1
202	39.0	134000.0	1
250	44.0	39000.0	0
274	57.0	26000.0	1

3. **Discover and Visualize the Data to gain insights:**
   For visualization, the testing set was kept aside, and a copy of training dataset is created so as to avoid any kind of loss in data.

Preparing dataset for machine learning model

```
#having the copy of training dataframe to apply transformations
df_train =train_set.copy()
df_train.columns
```

```
Index(['Age', 'EstimatedSalary', 'Purchased'], dtype='object')
```

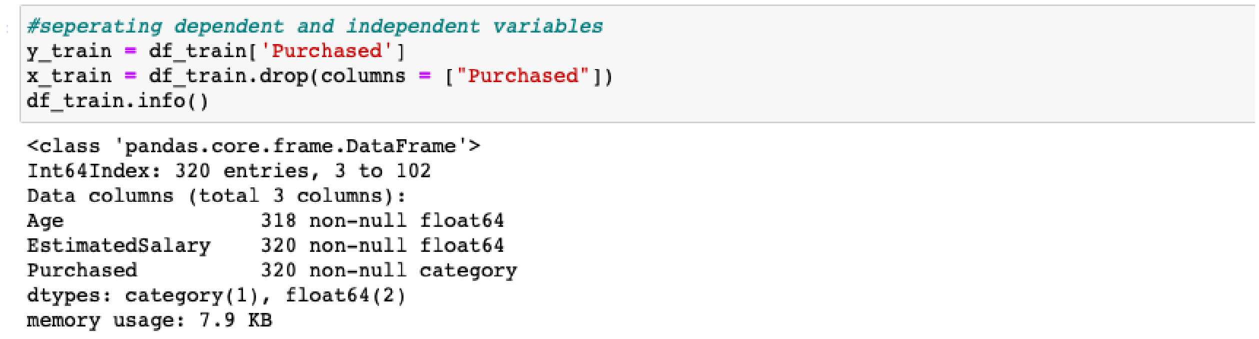
Let's observe the balance of categories in the training set

```
#checking for balance in class predictor variable in training set
train_set.Purchased.value_counts()
```

```
0    205
1    115
Name: Purchased, dtype: int64
```

Here as well we can observe that dataset is bit imbalanced but this imbalancing proportion is not much high. Therefore, there are no-any concerns or issues while modeling or evaluating dataset in terms of balance between categories.

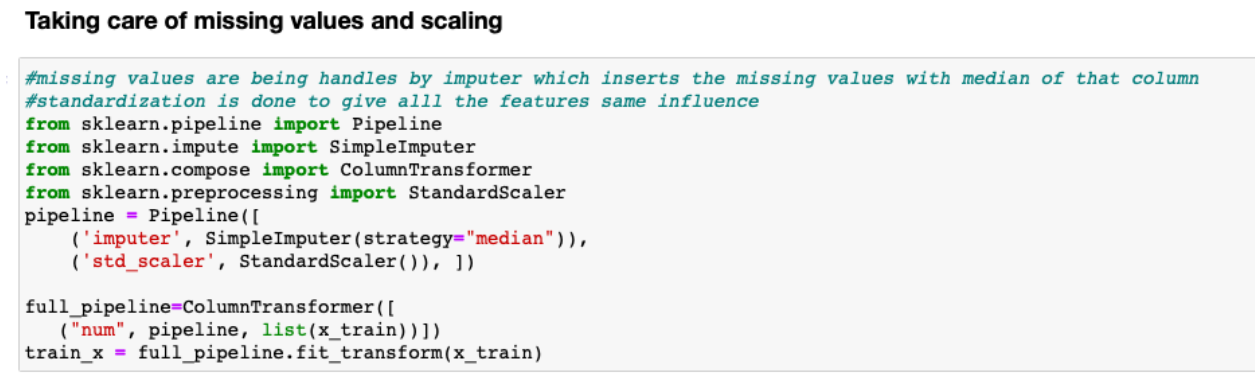
4. **Prepare the data to better expose the underlying data patterns to machine learning algorithm:**
   For preparing the data first we need to separate the independent columns from dependent columns. Here we can use drop () function for dropping the target column that is Purchased to get the dataset with dependent columns.



```
#seperating dependent and independent variables
y_train = df_train['Purchased']
x_train = df_train.drop(columns = ["Purchased"])
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 320 entries, 3 to 102
Data columns (total 3 columns):
Age                318 non-null float64
EstimatedSalary    320 non-null float64
Purchased          320 non-null category
dtypes: category(1), float64(2)
memory usage: 7.9 KB
```

After this separation now we need to take care of missing values and scaling of variables in training dataset.



Taking care of missing values and scaling

```
#missing values are being handles by imputer which inserts the missing values with median of that column
#standardization is done to give alll the features same influence
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()), ])

full_pipeline=ColumnTransformer([
   ("num", pipeline, list(x_train))])
train_x = full_pipeline.fit_transform(x_train)
```

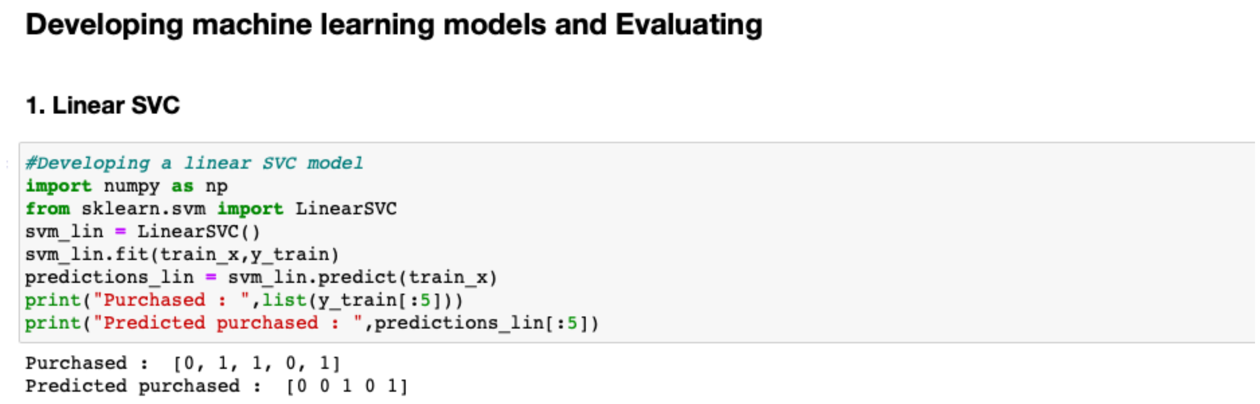
The missing values in dataset are imputed with median of that column using SimpleImputer () function. The columns are scaled using StandardScalar () function. The scaling is important because SVM models are sensitive to scaling. A pipeline is created for both tasks of scaling and imputing missing values, then this pipeline is used in ColumnTranformer () function to apply the both pipeline functions on columns of dataset.

5. Explore many different models and shortlist the best one:

Now when the data is prepared, we go for formation of machine learning algorithm.

a. Linear SVC:

The linear support vector classifier separates the both the binary classes by a straight line and this line is drawn in such a way that it is far away from closest training instances as possible. It uses the soft margin classification to discover a good balance between a large street and limited margin violations.



Developing machine learning models and Evaluating

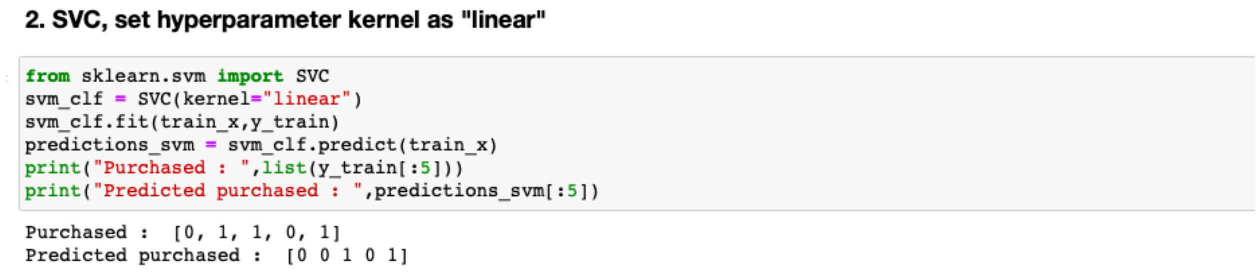
1. Linear SVC

```
#Developing a linear SVC model
import numpy as np
from sklearn.svm import LinearSVC
svm_lin = LinearSVC()
svm_lin.fit(train_x,y_train)
predictions_lin = svm_lin.predict(train_x)
print("Purchased : ",list(y_train[:5]))
print("Predicted purchased : ",predictions_lin[:5])
```

```
Purchased :  [0, 1, 1, 0, 1]
Predicted purchased :  [0 0 1 0 1]
```

b. SVC, hyperparameter kernel as linear:

Next, we are trying the SVC model with kernel = “linear”, first let me explain what kernel is, it is a mathematical technique that makes it possible to get the same results as we get by adding polynomial feature. This is the most efficient and less expensive way to transform the data into higher dimensions of polynomial degree. Here we are using kernel = linear which means it separates the data with single line. The SVC model with kernel = “linear” works much faster as compared to other kernels which makes it quite efficient method if the dataset contains large number of features whereas LinearSVC () is much faster than SVC with kernel = linear because of its lower training time complexity.



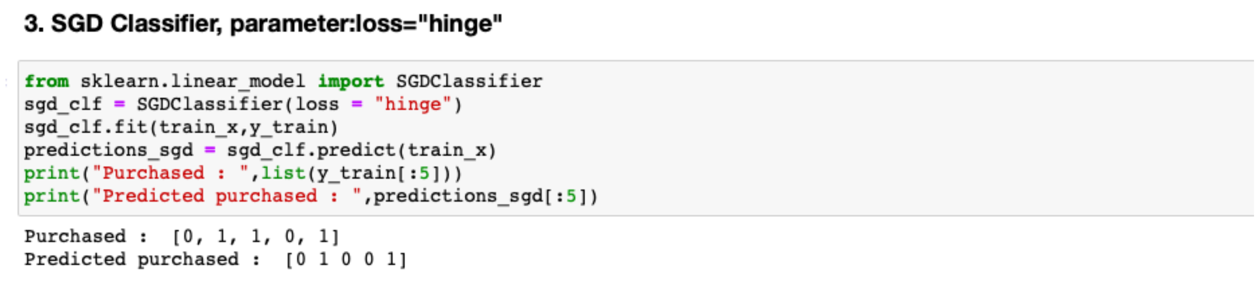
2. SVC, set hyperparameter kernel as "linear"

```
from sklearn.svm import SVC
svm_clf = SVC(kernel="linear")
svm_clf.fit(train_x,y_train)
predictions_svm = svm_clf.predict(train_x)
print("Purchased : ",list(y_train[:5]))
print("Predicted purchased : ",predictions_svm[:5])
```

```
Purchased :  [0, 1, 1, 0, 1]
Predicted purchased :  [0 0 1 0 1]
```

c. SGD Classifier, parameter loss=hinge:

SGD Classifier is a classifier that regularize linear model with stochastic gradient descent learning. The fit of model is controlled by loss parameter, here we are passing hinge to loss parameter which is a type of cost function that calculates the cost of instances on basis of their distance from the decision boundary.

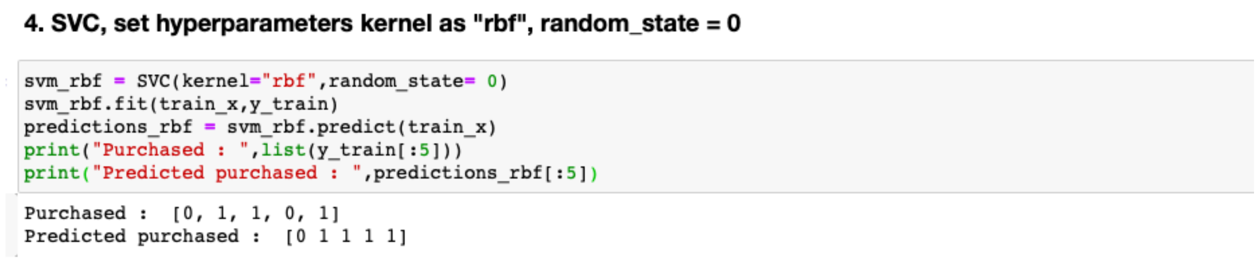


3. SGD Classifier, parameter:loss="hinge"

```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(loss = "hinge")
sgd_clf.fit(train_x,y_train)
predictions_sgd = sgd_clf.predict(train_x)
print("Purchased : ",list(y_train[:5]))
print("Predicted purchased : ",predictions_sgd[:5])
```

```
Purchased :  [0, 1, 1, 0, 1]
Predicted purchased :  [0 1 0 0 1]
```

d. **SVC, hyperparameter kernel as rbf and random_state = 0**
Here we are using kernel as rbf (Radial Basis Function), this function uses Euclidian distance between 2 data points and mean for generating a decision boundary. It uses 2 parameters gamma and C. Gamma parameter specifies the decision region. High the gamma is more curve the decision boundary is. Cis the penalty parameter that is penalty for wrong classified instances. The random_state is set to 0 which means we do not want any random number generation for shuffling the data for estimating the probabilities.

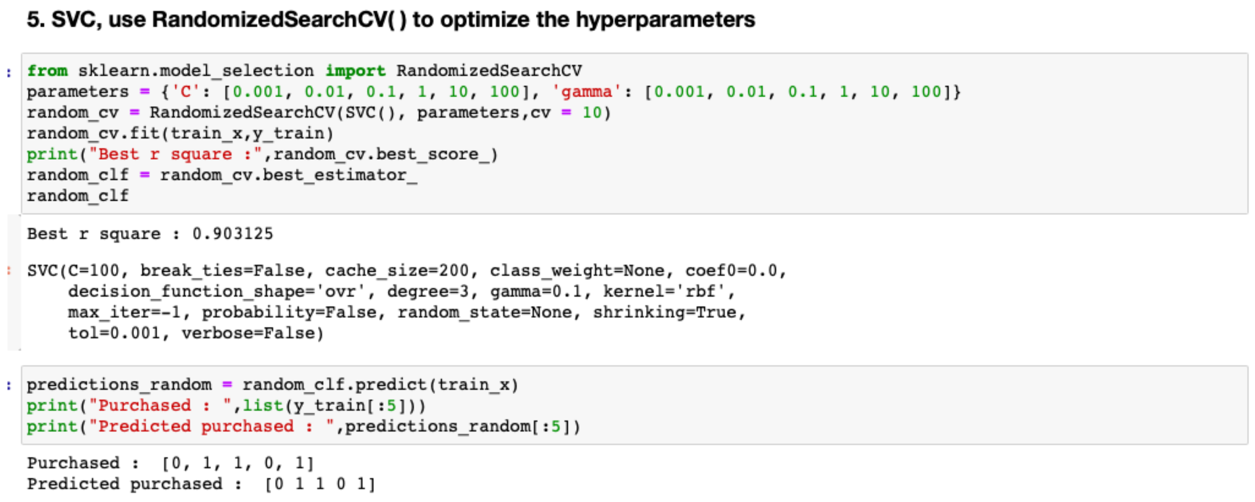


4. SVC, set hyperparameters kernel as "rbf", random_state = 0

```
svm_rbf = SVC(kernel="rbf",random_state= 0)
svm_rbf.fit(train_x,y_train)
predictions_rbf = svm_rbf.predict(train_x)
print("Purchased : ",list(y_train[:5]))
print("Predicted purchased : ",predictions_rbf[:5])
```

```
Purchased :  [0, 1, 1, 0, 1]
Predicted purchased :  [0 1 1 1 1]
```

e. SVC, use RandomizedSearchCV () to optimize the hyperparameters:
The RandomizedSearchCV () method is used for searching for set of hypermeters that results in best performance of model. It’s a model tuning technique that randomly sample points in a bounded domain of hyperparameter values. This method is good for finding new hyperparameter values or any new combination of hyperparameters.



5. SVC, use RandomizedSearchCV( ) to optimize the hyperparameters

```
from sklearn.model_selection import RandomizedSearchCV
parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
random_cv = RandomizedSearchCV(SVC(), parameters,cv = 10)
random_cv.fit(train_x,y_train)
print("Best r square :",random_cv.best_score_)
random_clf = random_cv.best_estimator_
random_clf
```

```
Best r square : 0.903125

SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
predictions_random = random_clf.predict(train_x)
print("Purchased : ",list(y_train[:5]))
print("Predicted purchased : ",predictions_random[:5])
```

```
Purchased :  [0, 1, 1, 0, 1]
Predicted purchased :  [0 1 1 0 1]
```

The SVC () model with best parameters was formed using best_estimator_ method of RandomizedSearchCV().

Evaluation:

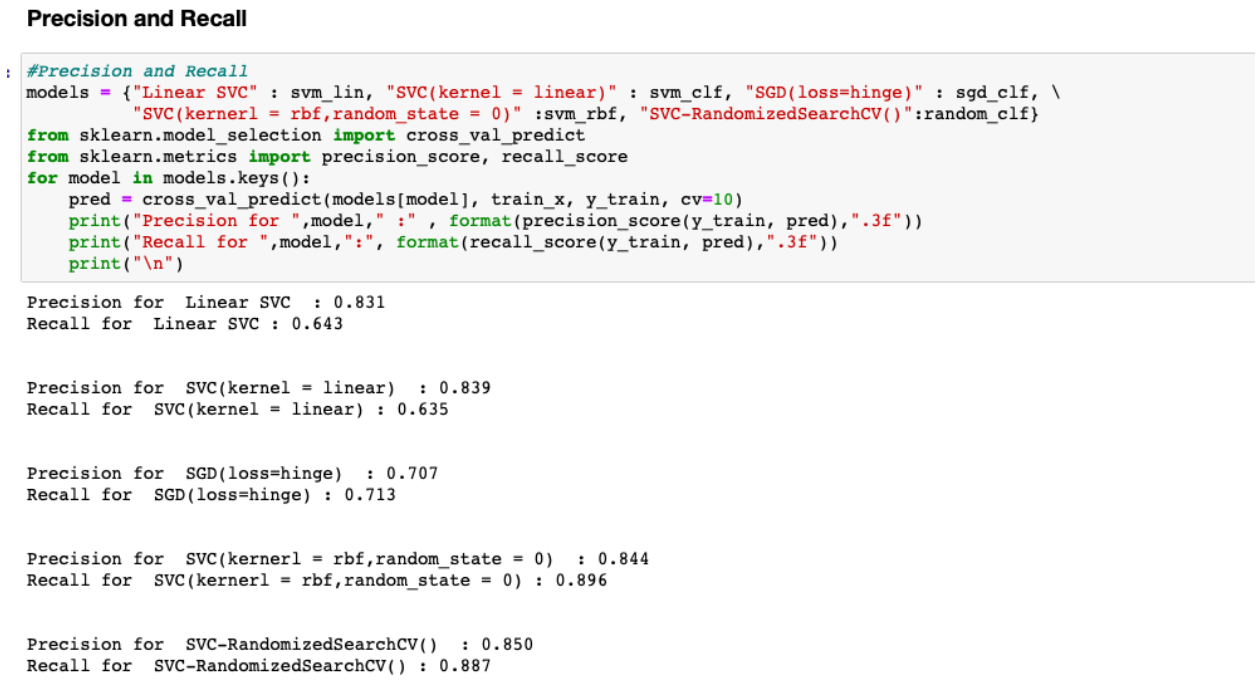
The methods used for evaluation are Precision, Recall , F1 score and ROC curve (area under curve):

Precision and Recall:

The Precision is a measure that define the %age of time a model will predict the correct positive class. In other words, it is ratio of correct positive predictions over total positive predictions by a model. Recall is the measure that defines the %age of time a model will predict correct positive predictions out of all the correct predictions.

Precision = Correct Positive (True positive) / (True Positive + False Positive)

Recall = True Positive / (True Positive + False Negative)



Precision and Recall

```
#Precision and Recall
models = {"Linear SVC" : svm_lin, "SVC(kernel = linear)" : svm_clf, "SGD(loss=hinge)" : sgd_clf, \
          "SVC(kernerl = rbf,random_state = 0)" :svm_rbf, "SVC-RandomizedSearchCV()":random_clf}
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import precision_score, recall_score
for model in models.keys():
    pred = cross_val_predict(models[model], train_x, y_train, cv=10)
    print("Precision for ",model," :" , format(precision_score(y_train, pred),".3f"))
    print("Recall for ",model,":", format(recall_score(y_train, pred),".3f"))
    print("\n")
```

```
Precision for  Linear SVC  : 0.831
Recall for  Linear SVC : 0.643


Precision for  SVC(kernel = linear)  : 0.839
Recall for  SVC(kernel = linear) : 0.635


Precision for  SGD(loss=hinge)  : 0.707
Recall for  SGD(loss=hinge) : 0.713


Precision for  SVC(kernerl = rbf,random_state = 0)  : 0.844
Recall for  SVC(kernerl = rbf,random_state = 0) : 0.896


Precision for  SVC-RandomizedSearchCV()  : 0.850
Recall for  SVC-RandomizedSearchCV() : 0.887
```

So, we would like to have more correct positive predictions that is high precision with enough high recall, from the above precisions and recall for all the 5 models it is observed that SVC model with parameters optimized by RandomizedSearchCV () kernel = rbf and random_state = 0 has highest precision. The 2nd close is the SVC model with kernel = rbf and random_state = 0 with highest recall.

F1- score:

It is the measure of model's accuracy by combining the precision and recall of model. In other words, it is harmonic mean of model's precision and recall. High is the f score, high is the accuracy of model in terms of combined effect of precision and recall.

F score = 2* (precision * recall)/(precision + recall)

F-1 score

```
for model in models.keys():
    pred = cross_val_predict(models[model], train_x, y_train, cv=10)
    print("F-1 score for ",model," :" , format(f1_score(y_train, pred),".3f"))
    print("\n")
```

```
F-1 score for  Linear SVC  : 0.725


F-1 score for  SVC(kernel = linear)  : 0.723


F-1 score for  SGD(loss=hinge)  : 0.698


F-1 score for  SVC(kernerl = rbf,random_state = 0)  : 0.869


F-1 score for  SVC-RandomizedSearchCV()  : 0.851
```

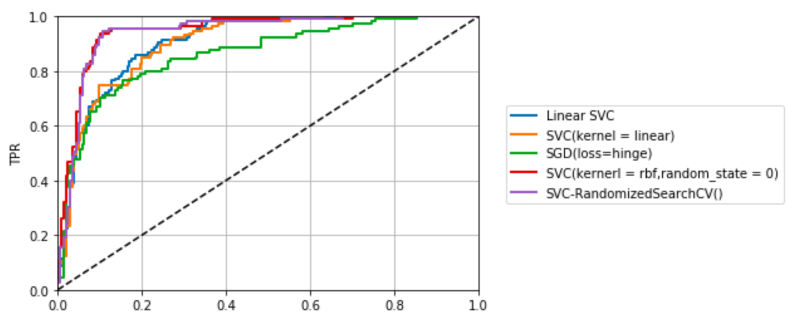
In the above results for f1 score of all the models, the svc model with kernel = rbf and random_state = 0 has highest f1 score, this shows that this model gives high accuracy in terms of combined effect of precision and recall on overall.
Now as we know, we have high precision for RandomizedSearchCV () model whereas highest f-1 score for SVC model with kernel = rbf and random_state=0. So let's go for better evaluation of these models with ROC curves and area under these curves.

ROC -curve:
ROC (Receiver Operating Characteristics) curve, this curve measures the performance for model at various thresholds. The cure is plotted using true positive rate and false positive rate of model at different threshold settings. The true positive rate determines the sensitivity of model whereas false positive rate gives the probability of false alarms while predictions.

ROC curve

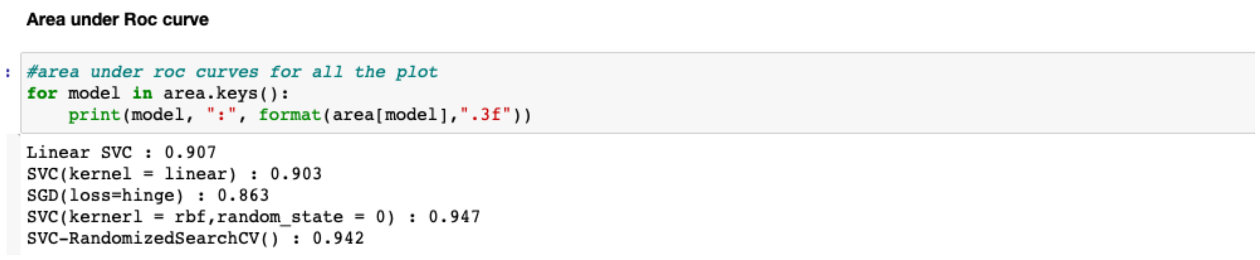
click to scroll output; double click to hide

```
#ROC curves for all the 5 models in one plot
from sklearn.metrics import roc_curve,auc
area = {}
for model in models:
    y_scores = cross_val_predict(models[model], train_x, y_train, cv=10, method="decision_function")
    fpr,tpr, thresholds = roc_curve(y_train, y_scores)
    area[model] = auc(fpr,tpr)
    plt.plot(fpr,tpr,linewidth=2, label =model)
plt.plot([0,1], [0,1], 'k--')
plt.axis([0,1,0,1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend(bbox_to_anchor=(1.05, 0.5, 0.3, 0.2), loc='upper left')
plt.grid(True)
```



The high the True positive rate, best the model is, in other words, farthest the curve from random classifier (black linear line in plot), the best fit the curve is. In the above it is observed that SVC model with kernel = rbf and random_state = 0 is the farthest one whereas model SVC with parameters from RandomizedSearchCV () is 2nd farthest. For better conclusion, lets calculate area under ROC curve.

Area-under curve:
Area-under curve is used for evaluation because high the area under roc curve is, high is the true positive rate for that model. Therefore, model with high auc is the best fit model.


Area under Roc curve

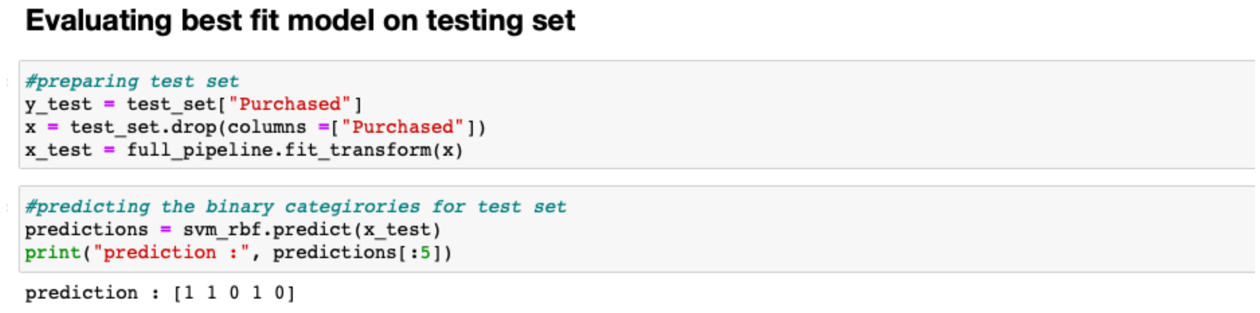
```
#area under roc curves for all the plot
for model in area.keys():
    print(model, ":", format(area[model],".3f"))
```

```
Linear SVC : 0.907
SVC(kernel = linear) : 0.903
SGD(loss=hinge) : 0.863
SVC(kernerl = rbf,random_state = 0) : 0.947
SVC-RandomizedSearchCV() : 0.942
```

From the above results, the SVC model with kernel = rbf and random_state = 0 has the highest auc which shows this model has high true positive rate.

To conclude, From the evaluation measures, it has been observed that SVC model with kernel = rbf and random_state = 0 is the best fit model.

6. Fine-tune your models and combine them into a great solution:
Now after the evaluation we found that svm model with kernel as rbf is the most promising model, now it's time to test on testing data.


Evaluating best fit model on testing set

```
#preparing test set
y_test = test_set["Purchased"]
x = test_set.drop(columns =["Purchased"])
x_test = full_pipeline.fit_transform(x)
```

```
#predicting the binary categirories for test set
predictions = svm_rbf.predict(x_test)
print("prediction :", predictions[:5])
```

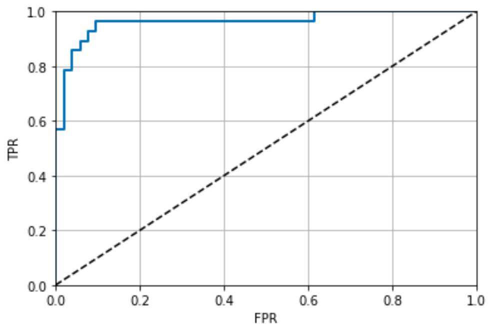
```
prediction : [1 1 0 1 0]
```

Here, we used the model for classifying the instances of the test set.

```
#precision and recall
pred_test = cross_val_predict(svm_rbf,x_test, y_test, cv=10)
print("Precision for test set :", format(precision_score(y_test, pred_test),".3f"))
print("Recall for test set :", format(recall_score(y_test, pred_test),".3f"))
print("F-1 score for test set :", format(f1_score(y_test, pred_test),".3f"))
```

```
Precision for test set : 0.889
Recall for test set : 0.857
F-1 score for test set : 0.873
```

```
#ROC curve
y_scores_test = cross_val_predict(svm_rbf,x_test, y_test, cv=10, method="decision_function")
fpr_test,tpr_test, thresholds_test = roc_curve(y_test, y_scores_test)
plt.plot(fpr_test,tpr_test,linewidth=2)
plt.plot([0,1], [0,1], 'k--')
plt.axis([0,1,0,1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.grid(True)
```



```
#Area under curve
print("Area under Roc curve for test set :", format(auc(fpr_test,tpr_test),".3f"))
```

```
Area under Roc curve for test set : 0.963
```

Using the same evaluation measure, we got that precision for the model on testing set is 88.9% which is higher as compared to all other model's precision in training set, recall is 85.7% which is enough close to the recall for model in training set. This evaluation shows that model works well but let's go for higher level of evaluation measure that is ROC curve and area under roc curve.
The ROC curve seems farther from random classifier whereas we should go for area under curve for better evaluation because numbers give the good summary for comparison. The auc here is 0.963 which is greater than all other models auc in training set, so the model gives high true positive rate for testing set as well.
Therefore, SVC model with kernel = rbf and random_state = 0 works good for testing set as well. Hence, it is best fit model.

Conclusion: Through evaluations it has shown that svc model with kernel = rbf and random_state =0 works best for the prediction of whether the user will purchase the product or not based on their age and salary.