EDA, FE and Classification Model (Census Income Dataset)

Aryan Shriva

Linkedin: https://www.linkedin.com/in/aryan-shriva-0811/ (https://www.linkedin.com/in/aryan-shriva-0811/)

GitHub https://github.com/AryanShriva/Machine-Learning (https://github.com/AryanShriva/Machine-Learning)

For the code please check out my Machine Learning repository on GitHub

1. EDA and FE

1. Data Profiling
2. Stastical analysis
3. Graphical Analysis
4. Data Cleaning
5. Data Scaling

2. Logistic Regression Model

1. Linear Regression Model
2. Performance metrics for above model
3. Hyper-Parameter Tuning for above model

3. Support Vector Classifier Model

1. Support Vector Classifier Model
2. Performance metrics for above model
3. Hyper-Parameter Tuning for above model

Dataset: https://archive.ics.uci.edu/ml/datasets/Census+Income (https://archive.ics.uci.edu/ml/datasets/Census+Income)

1.0 Importing required libraries

In [ ]:
```
### Pandas and Numpy
import pandas as pd
import numpy as np

### MongoDB Library
import pymongo

### Machine Learning libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, classification

### To ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

2.0 Retrieving data from MongoDB

In [2]:
```
### Retriving data from Mongodb
### creating connection with MongoDB

client = pymongo.MongoClient("mongodb+srv://{username}:{password}@clustershu
```

In [3]:
```
db=client['Census_income']
collection=db['Census_income_data']
```

In [4]:
```
### Locating our collection and data in MongoDb using find() method
data_from_mongodb=collection.find()
```

In [5]:
```
### converting data from MongoDb to Dataframe in pandas
data_mongodb=pd.DataFrame(data_from_mongodb)
```

In [6]:
```
### first 5 records in dataset
data_mongodb.head()
```

Out[6]:

	_id	index	age	workclass	fnlwgt	education	education_num	n
0	63635b0506652a035edadc4c	31	20	Private	266015	Some_college	10	N
1	63635b0506652a035edadc2d	0	39	other	77516	Bachelors	13	N
2	63635b0506652a035edadc3a	13	32	Private	205019	other	12	N
3	63635b0506652a035edadc3d	16	25	other	176756	HS_grad	9	N
4	63635b0506652a035edadc2e	1	50	other	83311	Bachelors	13	Marrie

In [7]:
```
### dropping _id and index feature from dataset imported from MongoDB
data_mongodb.drop(['_id','index'], axis=1, inplace=True)
data_mongodb.head()
```

Out[7]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	rela
0	20	Private	266015	Some_college	10	Never_married	Sales	C
1	39	other	77516	Bachelors	13	Never_married	Adm_clerical	Not_
2	32	Private	205019	other	12	Never_married	Sales	Not_
3	25	other	176756	HS_grad	9	Never_married	other	C
4	50	other	83311	Bachelors	13	Married_civ_spouse	Exec_managerial	

3.0 Model and Evaluation

3.1 Seperating Independent and Dependent features

In [8]:
```
### Splitting data into independent feature dataframe and dependent feature
X=data_mongodb.iloc[:,:-1]
y=data_mongodb.iloc[:,-1]
X.head()
```

Out[8]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	rela
0	20	Private	266015	Some_college	10	Never_married	Sales	C
1	39	other	77516	Bachelors	13	Never_married	Adm_clerical	Not_
2	32	Private	205019	other	12	Never_married	Sales	Not_
3	25	other	176756	HS_grad	9	Never_married	other	C
4	50	other	83311	Bachelors	13	Married_civ_spouse	Exec_managerial	

In [9]:
```
y.head()
```

Out[9]:
```
0    0
1    0
2    0
3    0
4    0
Name: salary, dtype: int64
```

3.2 Train Test Split

In [10]:
```
### random state train test split will be same with all people using random_

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
X_train.head()
```

Out[10]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation
34576	30	Private	23778	Some_college	10	Never_married	Exec_managerial
33148	41	Private	112763	Some_college	10	Married_civ_spouse	Adm_clerical
2109	20	Private	241752	HS_grad	9	Married_civ_spouse	other
33501	35	Private	211494	Bachelors	13	Never_married	Exec_managerial
47110	24	Private	408585	other	4	Married_civ_spouse	other

In [11]:
```
y_train.head()
```

Out[11]:
```
34576    0
33148    0
2109     0
33501    0
47110    0
Name: salary, dtype: int64
```

In [12]:
```
X_test.head()
```

Out[12]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	re
3436	52	Private	48925	Some_college	10	Married_civ_spouse	Adm_clerical	
16332	27	Private	31659	Bachelors	13	Married_civ_spouse	other	
39798	67	Private	187553	other	4	Divorced	Prof_specialty	Nc
12405	45	other	255559	HS_grad	9	Never_married	Adm_clerical	Nc
7584	32	Private	169955	Some_college	10	Married_civ_spouse	Other_service	

In [13]:
```
y_test.head()
```

Out[13]:
```
3436     0
16332    1
39798    0
12405    0
7584     0
Name: salary, dtype: int64
```

In [14]:
```

### both will have same shape
X_train.shape, y_train.shape
```

Out[14]: ((36609, 14), (36609,))

In [15]:
```
### both will have same shape
X_test.shape, y_test.shape
```

Out[15]: ((12204, 14), (12204,))

3.3 Feature Encoding

In [16]:
```
column_trans=make_column_transformer(
        (OneHotEncoder(), ['workclass','education', 'marital_status', 'occup
        remainder='passthrough')
```

In [17]:
```
X_train=column_trans.fit_transform(X_train)
```

In [18]:
```
X_test=column_trans.transform(X_test)
```

3.4 Feature Scaling

In [19]:
```
scaler=StandardScaler()
```

In [20]:
```
X_train=scaler.fit_transform(X_train)
```

In [21]:
```
X_test=scaler.transform(X_test)
```

In [22]:
```
X_train_scaled=pd.DataFrame(X_train)
X_train_scaled.head()
```

Out[22]:

	0	1	2	3	4	5	6	7	8
0	0.573765	-0.573765	-0.444065	-0.691388	1.869548	-0.638185	-0.394399	-0.923205	1.426647
1	0.573765	-0.573765	-0.444065	-0.691388	1.869548	-0.638185	-0.394399	1.083183	-0.700944
2	0.573765	-0.573765	-0.444065	1.446367	-0.534889	-0.638185	-0.394399	1.083183	-0.700944
3	0.573765	-0.573765	2.251920	-0.691388	-0.534889	-0.638185	-0.394399	-0.923205	1.426647
4	0.573765	-0.573765	-0.444065	-0.691388	-0.534889	1.566943	-0.394399	1.083183	-0.700944

5 rows × 34 columns

In [23]:
```
X_test_scaled=pd.DataFrame(X_test)
X_test_scaled.head()
```

Out[23]:

	0	1	2	3	4	5	6	7	8
0	0.573765	-0.573765	-0.444065	-0.691388	1.869548	-0.638185	-0.394399	1.083183	-0.700944
1	0.573765	-0.573765	2.251920	-0.691388	-0.534889	-0.638185	-0.394399	1.083183	-0.700944
2	0.573765	-0.573765	-0.444065	-0.691388	-0.534889	1.566943	2.535503	-0.923205	-0.700944
3	-1.742874	1.742874	-0.444065	1.446367	-0.534889	-0.638185	-0.394399	-0.923205	1.426647
4	0.573765	-0.573765	-0.444065	-0.691388	1.869548	-0.638185	-0.394399	1.083183	-0.700944

5 rows × 34 columns

3.5 Logestic Regression Model

In [24]:
```
### model
logistic_reg=LogisticRegression()
logistic_reg
```

Out[24]: LogisticRegression()

In [25]:
```
logistic_reg.fit(X_train, y_train)
```

Out[25]: LogisticRegression()

In [26]:
```
logistic_reg_pred=logistic_reg.predict(X_test)
logistic_reg_pred
```

Out[26]: array([0, 1, 0, ..., 0, 0, 0])

In [27]:
```
confusion_mat=confusion_matrix(y_test, logistic_reg_pred)
confusion_mat
```

Out[27]:
```
array([[8618,  644],
       [1175, 1767]])
```

In [28]:
```
truly_positive=confusion_mat[0][0]
falsely_positive=confusion_mat[0][1]
falsely_negative=confusion_mat[1][0]
truly_negative=confusion_mat[1][1]
```

In [29]:
```
classification_rep_log_reg=classification_report(y_test, logistic_reg_pred)
print(classification_rep_log_reg)
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      9262
           1       0.73      0.60      0.66      2942

    accuracy                           0.85     12204
   macro avg       0.81      0.77      0.78     12204
weighted avg       0.84      0.85      0.85     12204
```

3.6 Support Vector Classifier Model

In [30]:
```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.25
```

In [31]:
```
column_trans_svc=make_column_transformer(
        (OneHotEncoder(), ['workclass','education', 'marital_status', 'occup
        remainder='passthrough')
```

In [32]:
```
X_train1=column_trans_svc.fit_transform(X_train1)
```

In [33]:
```
X_test1=column_trans_svc.transform(X_test1)
```

In [34]:
```
scaler_svc=StandardScaler()
scaler_svc
```

Out[34]:
```
StandardScaler()
```

In [35]:
```
X_train1=scaler_svc.fit_transform(X_train1)
```

In [36]:
```
X_test1=scaler_svc.transform(X_test1)
```

In [ ]:
```
svc=SVC()
svc
```
Out[38]: SVC()

In [ ]:
```
svc.fit(X_train1, y_train1)
```
Out[39]: SVC()

In [ ]:
```
svc_pred=svc.predict(X_test1)
svc_pred
```
Out[40]: array([0, 1, 0, ..., 0, 0, 0])

In [ ]:
```
confusion_mat_svc=confusion_matrix(y_test1, svc_pred)
confusion_mat_svc
```
Out[41]:
```
array([[8718,  544],
       [1256, 1686]])
```

In [ ]:
```
truly_positive=confusion_mat_svc[0][0]
falsely_positive=confusion_mat_svc[0][1]
falsely_negative=confusion_mat_svc[1][0]
truly_negative=confusion_mat_svc[1][1]
```

In [ ]:
```
classification_rep_svc=classification_report(y_test1, svc_pred)
print(classification_rep_svc)
```
```
              precision    recall  f1-score   support

           0       0.87      0.94      0.91      9262
           1       0.76      0.57      0.65      2942

    accuracy                           0.85     12204
   macro avg       0.82      0.76      0.78     12204
weighted avg       0.85      0.85      0.85     12204
```

3.7 Hyper-Parameter Tuning Logistic Regression Model

In [ ]:
```
param_grid = [
    {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 5),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 500]
    }
]
```

In [ ]:
```
log_reg_hpt=LogisticRegression()
log_reg_hpt
```
Out[45]: LogisticRegression()

In [ ]:
```
hpt_log_reg=GridSearchCV(log_reg_hpt, param_grid = param_grid)
```

In [ ]:
```
best_hpt_log_reg=hpt_log_reg.fit(X_train, y_train)
best_hpt_log_reg
```

Out[47]:
```
GridSearchCV(estimator=LogisticRegression(),
             param_grid=[{'C': array([1.e-04, 1.e-02, 1.e+00, 1.e+02, 1.e+04]),
                          'max_iter': [100, 500],
                          'penalty': ['l1', 'l2', 'elasticnet', 'none'],
                          'solver': ['lbfgs', 'newton-cg', 'liblinear', 'sag',
                                     'saga']}])
```

In [ ]:
```
### getting best parameters for Logistic Regression model after gridsearchCV
print("Best parameters are {} for optimal accuracy.".format(best_hpt_log_reg
```

```
Best parameters are LogisticRegression(C=0.01, penalty='l1', solver='liblinea
r') for optimal accuracy.
```

In [ ]:
```
### getting best accuracy for Logistic Regression model after gridsearchCV
print("Best accuracy is {}".format(best_hpt_log_reg.score(X_test, y_test)))
```

```
Best accuracy is 0.8504588659455916
```

3.8 Hyper-Parameter Tuning Support Vector Classifier Model

In [37]:
```
svc_hpt=SVC()
svc_hpt
```

Out[37]:
```
SVC()
```

In [38]:
```
#### using gridsearchcv to increase model efficiency by combining above para
param_grid={'C':[1,2,3], 'kernel':['rbf']}
hpt_svc=GridSearchCV(svc_hpt, param_grid=param_grid)
```

In [39]:
```
best_hpt_svc=hpt_svc.fit(X_train1, y_train1)
best_hpt_svc
```

Out[39]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [10, 20], 'degree': [2, 3],
                         'kernel': ['linear', 'rbf', 'poly', 'sigmoid']})
```

In [40]:
```
### getting best parameters for Logistic Regression model after gridsearchCV
print("Best parameters are {} for optimal accuracy.".format(best_hpt_svc.bes
```

```
Best parameters are SVC(C=20, degree=2, kernel='poly') for optimal accuracy.
```

In [41]:
```
### getting best accuracy for Logistic Regression model after gridsearchCV
print("Best accuracy is {}".format(best_hpt_svc.score(X_test, y_test)))
```

```
Best accuracy is 0.8491478203867584
```

In [42]:
```
svc_hpt1=SVC()
svc_hpt1
```

Out[42]: SVC()

In [43]:
```
#### using gridsearchcv to increase model efficiency by combining above parai
param_grid1={'C':[1,2,3], 'kernel':['rbf']}
hpt_svc1=GridSearchCV(svc_hpt1, param_grid=param_grid1)
```

In [44]:
```
best_hpt_svc1=hpt_svc1.fit(X_train1, y_train1)
best_hpt_svc1
```

Out[44]: GridSearchCV(estimator=SVC(), param_grid={'C': [1, 2, 3], 'kernel': ['rbf']})

In [45]:
```
### getting best parameters for Logistic Regression model after gridsearchCV
print("Best parameters are {} for optimal accuracy.".format(best_hpt_svc1.be
```

```
Best parameters are SVC(C=2) for optimal accuracy.
```

In [47]:
```
### getting best accuracy for Logistic Regression model after gridsearchCV
print("Best accuracy is {}".format(best_hpt_svc1.score(X_test1, y_test1)))
```

```
Best accuracy is 0.8517699115044248
```

Note: Please refer my github repo for ROC AUC curve implementation