Supervised Machine Learning: Classification Project by Sisay Menji

In [144]:
```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
%matplotlib inline
pd.options.display.float_format = '{:,.2f}'.format
```

In [145]:
```
col_names = ["age","workclass", "fnlwgt", "education", "education_num",
             "marital_status", "occupation", "relationship","race", "sex",
             "capital_gain","capital_loss","hours_per_week","native_country","in
```

In [146]:
```
df = pd.read_table("adult.data", sep=",",names=col_names)
```

In [147]:
```
df.head()
```

Out[147]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	ra
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	Wh
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Wh
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	Wh
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Bla
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Bla

Data descriptions

The data for the project is taken from [UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Adult)](https://archive.ics.uci.edu/ml/datasets/Adult). The data was extracted from 1994 Census database by Barry Becker.The data is used to predict whether a persom makers over 50K a year or not.The data contains a cleaned records. The data contains 15 variables and 32561 observations. The variables included are:

- **<=50**: a binary variable that indicates whether the person makes over 50K or not
- **age**: a continuous variable showing age
- **workclass**: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

- **fnlwgt**: continuous survey weight variable
- **education**: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- **education-num**: continuous.
- **marital-status**: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- **occupation**: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- **relationship**: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- **race**: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- **sex**: Female, Male.
- **capital-gain**: continuous.
- **capital-loss**: continuous.
- **hours-per-week**: continuous.
- **native-country**: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

Objectives of the analysis

The analysis will focus on building various classification models to predict whether a person makes over 50K or not. In this regard the analysis will conduct the following:

1. Compare different classification models (logistic regression, KNN, and Random Forest) and chose the best model
2. For the chosen best model use oversampling techniques to compensate for the unbalanced sample size
3. Assess whether oversampling the minority class helps increase recall and reduce precision
4. Understand which variables are important in explaining whether a person earns above 50K per year or not

As the data is a survey data it has a weight variable used to get popoulation estimates. In this analysis I will focus on sample estimates and ignore the survey weight.

In [148]:
```
df.drop(columns=["fnlwgt"], inplace=True)
```

Descriptive Statistics

In [149]:
```
cols = df.columns
```

In [150]:
```
df.describe()
```

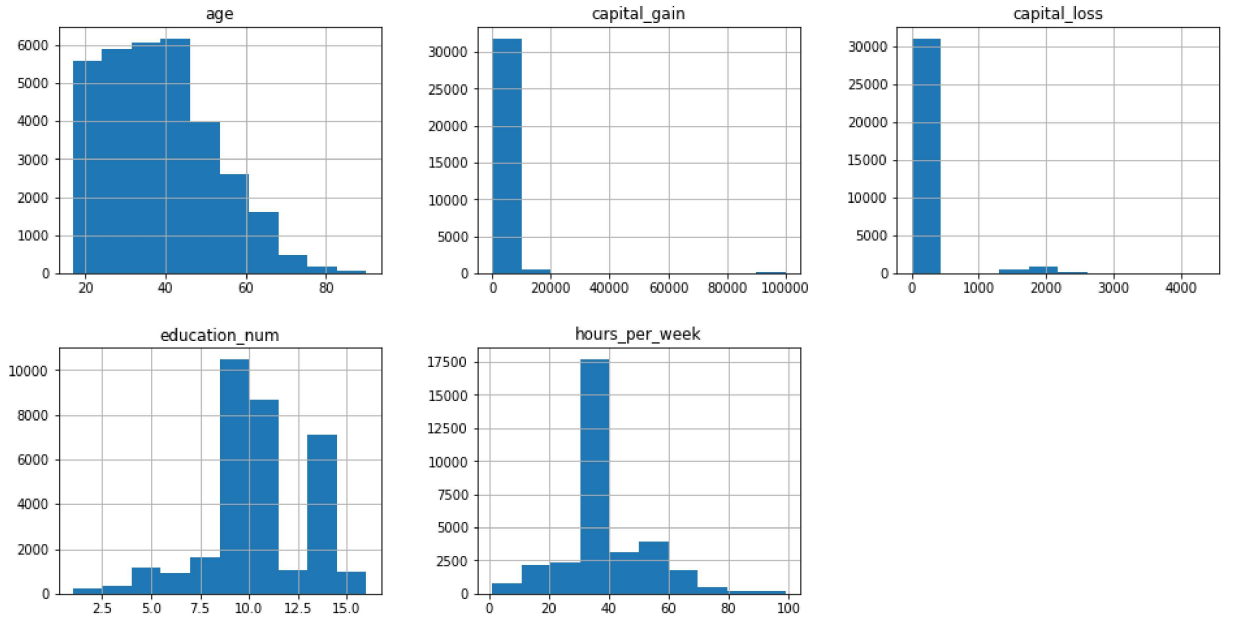
Out[150]:

	age	education_num	capital_gain	capital_loss	hours_per_week
count	32,561.00	32,561.00	32,561.00	32,561.00	32,561.00
mean	38.58	10.08	1,077.65	87.30	40.44
std	13.64	2.57	7,385.29	402.96	12.35
min	17.00	1.00	0.00	0.00	1.00
25%	28.00	9.00	0.00	0.00	40.00
50%	37.00	10.00	0.00	0.00	40.00
75%	48.00	12.00	0.00	0.00	45.00
max	90.00	16.00	99,999.00	4,356.00	99.00

The description for numeric variables above showed that age is a variable with a minimum of 17, mean of 38.6 and maximum of 90. Education has a mean of 10 years and a minimum and maximum of 1 and 16 respectively. Capital gain has a mean of 1077.65 and a standard deviation of 7,385 which shows it has higher variance. Similarly capital los has a mean of 87.30 and a std. of 402.96. Hours per week is around 40 for most observations with a mean of 40.44 and a std. of 12.35. The following histograms show the distribution of the variables.

As can be seen from the histogram except education years (education_num) and hours per week all variables are skewed to the left.

In [151]:
```
df.hist(figsize=(16,8), layout=(2,3));
```



In [152]:
```
df.describe(include='object')
```

Out[152]:

	workclass	education	marital_status	occupation	relationship	race	sex	native
count	32561	32561	32561	32561	32561	32561	32561	
unique	9	16	7	15	6	5	2	
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	Unit
freq	22696	10501	14976	4140	13193	27816	21790	

As the above table shows except sex and income which are binary other object variables has 5 or more unique values.The categorical columns later will be divided into three categories:

- binary: having two observations
- ordinal:education which shows order
- categorical: other variables

Our outcome variable (income) is unambalanced data with 76% (24720) individuals earning 50K or below and 24% (7841) individuals earning above 50K.

In [153]:
```
df["income"].value_counts(normalize=True)
```

Out[153]:
```
 <=50K    0.76
 >50K     0.24
Name: income, dtype: float64
```

In [154]:
```
df.groupby("income").agg("mean").T
```

Out[154]:

income	<=50K	>50K
age	36.78	44.25
education_num	9.60	11.61
capital_gain	148.75	4,006.14
capital_loss	53.14	195.00
hours_per_week	38.84	45.47

In [155]:
```
pd.crosstab(df.income, df.workclass,normalize=True,margins=True)
```

Out[155]:

workclass	?	Federal-gov	Local-gov	Never-worked	Private	Self-emp-inc	Self-emp-not-inc	State-gov	Without-pay	All
income										
<=50K	0.05	0.02	0.05	0.00	0.54	0.02	0.06	0.03	0.00	0.76
>50K	0.01	0.01	0.02	0.00	0.15	0.02	0.02	0.01	0.00	0.24
All	0.06	0.03	0.06	0.00	0.70	0.03	0.08	0.04	0.00	1.00

In [156]:
```
# relationship between income and workclass
pd.crosstab(df.income, df['workclass']).apply(lambda r: r/r.sum(), axis=0)
```

Out[156]:

workclass	?	Federal-gov	Local-gov	Never-worked	Private	Self-emp-inc	Self-emp-not-inc	State-gov	Without-pay
income									
<=50K	0.90	0.61	0.71	1.00	0.78	0.44	0.72	0.73	1.00
>50K	0.10	0.39	0.29	0.00	0.22	0.56	0.28	0.27	0.00

In [157]:
```
pd.crosstab(df.income, df['education']).apply(lambda r: r/r.sum(), axis=0)
```

Out[157]:

education	10th	11th	12th	1st-4th	5th-6th	7th-8th	9th	Assoc-acdm	Assoc-voc	Bachelors	Doctorate	HS-grad	M
income													
<=50K	0.93	0.95	0.92	0.96	0.95	0.94	0.95	0.75	0.74	0.59	0.26	0.84	
>50K	0.07	0.05	0.08	0.04	0.05	0.06	0.05	0.25	0.26	0.41	0.74	0.16	

As the above two tables shows the only working category higher income people excel compared to low income (<=50K) is the self employed category and higher income group has higher education level generally.

In [158]:
```
pd.crosstab(df.income, df['education_num']).apply(lambda r: r/r.sum(), axis=0)
```

Out[158]:

education_num	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
income															
<=50K	1.00	0.96	0.95	0.94	0.95	0.93	0.95	0.92	0.84	0.81	0.74	0.75	0.59	0.44	0.
>50K	0.00	0.04	0.05	0.06	0.05	0.07	0.05	0.08	0.16	0.19	0.26	0.25	0.41	0.56	0.

In [159]:
```
pd.crosstab(df.income, df['marital_status']).apply(lambda r: r/r.sum(), axis=0)
```

Out[159]:

marital_status	Divorced	Married-AF-spouse	Married-civ-spouse	Married-spouse-absent	Never-married	Separated	Widowed
income							
<=50K	0.90	0.57	0.55	0.92	0.95	0.94	0.91
>50K	0.10	0.43	0.45	0.08	0.05	0.06	0.09

In [160]:
```
pd.crosstab(df.income, df['occupation']).apply(lambda r: r/r.sum(), axis=0)
```

Out[160]:

occupation	?	Adm-clerical	Armed-Forces	Craft-repair	Exec-managerial	Farming-fishing	Handlers-cleaners	Machine-op-inspct	Other-service	P hou s
income										
<=50K	0.90	0.87	0.89	0.77	0.52	0.88	0.94	0.88	0.96	(
>50K	0.10	0.13	0.11	0.23	0.48	0.12	0.06	0.12	0.04	(

In [161]:
```
pd.crosstab(df.income, df['race']).apply(lambda r: r/r.sum(), axis=0)
```

Out[161]:

race	Amer-Indian-Eskimo	Asian-Pac-Islander	Black	Other	White
income					
<=50K	0.88	0.73	0.88	0.91	0.74
>50K	0.12	0.27	0.12	0.09	0.26

In [162]:
```
pd.crosstab(df.income, df['native_country']).apply(lambda r: r/r.sum(), axis=0)
```

Out[162]:

native_country	?	Cambodia	Canada	China	Columbia	Cuba	Dominican-Republic	Ecuador	El-Salvador
income									
<=50K	0.75	0.63	0.68	0.73	0.97	0.74	0.97	0.86	0.92
>50K	0.25	0.37	0.32	0.27	0.03	0.26	0.03	0.14	0.08

2 rows × 42 columns

The above tables show high income people are from professional speciality occupation and asian pacific islander and white races.The table by native country also confirms the above data.

Data preparation

The following data preparation activities are conducted to have a single data set that is fit for all the different classification models to be trained and tested.

1. Variable management: the valuation education is dropped as education_num serves the same purpose. Since education_num is an int64 it is converted into object data type for later recoding. Convert income into 1 and 0 vriable with 0 for income <=50K
2. Scaling the numeric variables
3. Encoding categorical variables:

In [163]:
```
#1. Variable management
df.drop(columns=["education"], inplace=True)
df["education_num"] = df["education_num"].astype(str)
obj_cols = list(set(df.loc[:, df.dtypes==object].columns) - set(["income"]))
df["income"] = df["income"].str.replace("<=50K","0").str.replace(" >50K","1").a
```

In [164]:
```
#2 Min Max scaling numeric variables
num_cols = df.loc[:,df.dtypes=='int64'].columns
num_cols = list(set(num_cols) - set("education_num",))
```

In [165]:
```
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
for n in num_cols:
    df[n] = mm.fit_transform(np.array(df[n]).reshape(-1,1))
```

In [166]:
```
# 3. Encoding categorical variables
obj_cols
```

Out[166]:
```
['workclass',
 'sex',
 'relationship',
 'occupation',
 'education_num',
 'race',
 'native_country',
 'marital_status']
```

In [167]:
```
[(i, len(df[i].unique())) for i in obj_cols]
```

Out[167]:
```
[('workclass', 9),
 ('sex', 2),
 ('relationship', 6),
 ('occupation', 15),
 ('education_num', 16),
 ('race', 5),
 ('native_country', 42),
 ('marital_status', 7)]
```

In [168]:
```
binary_cols = [i for i in obj_cols if len(df[i].unique())==2]
ordinal_cols = ["education_num"]
categorical_cols = list(set(obj_cols).difference(binary_cols,ordinal_cols))
print(" Binary columns: ", binary_cols,"\n","Ordinal columns: ", ordinal_cols,"
```

```
 Binary columns:  ['sex']
 Ordinal columns:  ['education_num']
 Categorical columns:  ['workclass', 'relationship', 'occupation', 'race', 'nat
ive_country', 'marital_status']
```

In [169]:
```
from sklearn.preprocessing import LabelBinarizer, LabelEncoder, OrdinalEncoder
lb = LabelBinarizer()
le = LabelEncoder()
oe = OrdinalEncoder()
```

In [170]:
```
df["education_num"] = oe.fit_transform(np.array(df["education_num"]).reshape(-1
```

In [171]:
```
for b in binary_cols:
    df[b] = lb.fit_transform(df[b])
df = pd.get_dummies(df, columns = categorical_cols, drop_first=True)
```

Train and test split

In [172]:
```
from sklearn.model_selection import StratifiedShuffleSplit
feature_cols = list(set(df.columns).difference(["income"]))
strat_shuff = StratifiedShuffleSplit(n_splits=1, random_state=42, test_size=0.3
train_idex, test_index = next(strat_shuff.split(df[feature_cols], df["income"])
```

In [173]:
```
x_train, x_test, y_train, y_test = df.loc[train_idex, feature_cols], df.loc[tes
        df.loc[train_idex,"income"], df.loc[test_index, "income"]
```

In [174]:
```
y_train.value_counts(normalize=True)
```

Out[174]:
```
0   0.76
1   0.24
Name: income, dtype: float64
```

In [175]:
```
y_test.value_counts(normalize=True)
```

Out[175]:
```
0   0.76
1   0.24
Name: income, dtype: float64
```

Classification models

In this section, I will implement different classification models to better predict whether a person earns more than 50K or not (i.e. class 1). The modeling approach will follow the following steps:

1. In all models GridSearchCV will be used to get the best paramter specification
2. All models will be compared using accuracy, f1 score and auc_score
3. The best model will be used for prediction and understanding which variables are important

Logistic regression

In [176]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

param_grid = {
    "solver": ['lbfgs', 'liblinear'],
    "penalty": ["l1","l2", "elasticnet", "none"],
    "C" :np.arange(0.5,5,1)
}
lr = GridSearchCV(LogisticRegression(n_jobs=-1, random_state=44), param_grid=pa
```

In [177]:
```
import warnings
warnings.filterwarnings(action="ignore", category=Warning)
```

In [178]:
```
lr.fit(x_train, y_train);
```

In [179]:
```
lr.best_params_
```

Out[179]:
```
{'C': 3.5, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [180]:
```
from sklearn.metrics import accuracy_score, f1_score, recall_score, classificat
def error_measures(y_t, y_p, label):
    return pd.Series(
        {
            'Accuracy': accuracy_score(y_t, y_p),
            'Recall': recall_score(y_t,y_p),
            'F1 Score': f1_score(y_t,y_p),
            'Roc AuC': roc_auc_score(y_t,y_p)
        },
        name=label
    )
```

In [181]:
```
scores = pd.DataFrame()
```

In [182]:
```
y_p = lr.predict(x_test)
lr_scores = error_measures(y_test, y_p, "Logistic Regression")
scores = pd.DataFrame(lr_scores)
scores
```

Out[182]:

	Logistic Regression
Accuracy	0.85
Recall	0.61
F1 Score	0.66
Roc AuC	0.77

In [183]:
```
print(classification_report(y_test, y_p))
```

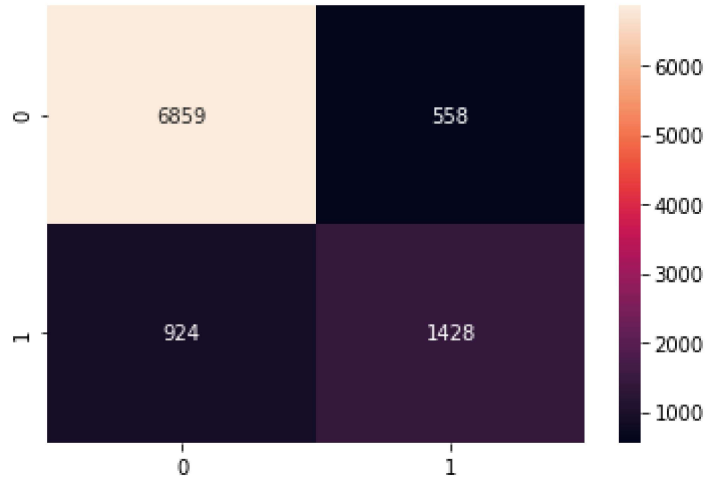
```
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      7417
           1       0.72      0.61      0.66      2352

    accuracy                           0.85      9769
   macro avg       0.80      0.77      0.78      9769
weighted avg       0.84      0.85      0.84      9769
```

In [184]:
```
import seaborn as sb
```

In [185]:
```
cm = {}
cm['lr'] = confusion_matrix(y_test,y_p)
```

In [186]:
```
print("Logistic regression confusion matrix")
sb.heatmap(confusion_matrix(y_test,y_p), annot=True, fmt='d');
```

```
Logistic regression confusion matrix
```



As the results above show logistic regression has an accuracy of 85%, recall of 61% and f1 score of 66%. For the class we are interested in (class 1), the precesion, recall and f1-score are lower 72%, 61% and 66% respectively.

KNN classification

In [187]:
```
from sklearn.neighbors import KNeighborsClassifier

param_grid = {
    'n_neighbors': np.arange(5,50,5),
    'weights': ['uniform','distance']
}
knn = GridSearchCV(KNeighborsClassifier(n_jobs=-1), param_grid=param_grid)
```

In [188]:
```
knn.fit(x_train, y_train)
```

Out[188]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=-1,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid={'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40,
45]),
                         'weights': ['uniform', 'distance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [189]:
```
knn.best_params_
```

Out[189]:
```
{'n_neighbors': 40, 'weights': 'uniform'}
```

In [190]:
```
y_p = knn.predict(x_test)
knn_scores = error_measures(y_test, y_p, "KNN")
scores = pd.concat([scores,knn_scores], axis=1)
scores
```

Out[190]:

	Logistic Regression	KNN
Accuracy	0.85	0.83
Recall	0.61	0.57
F1 Score	0.66	0.62
Roc AuC	0.77	0.74

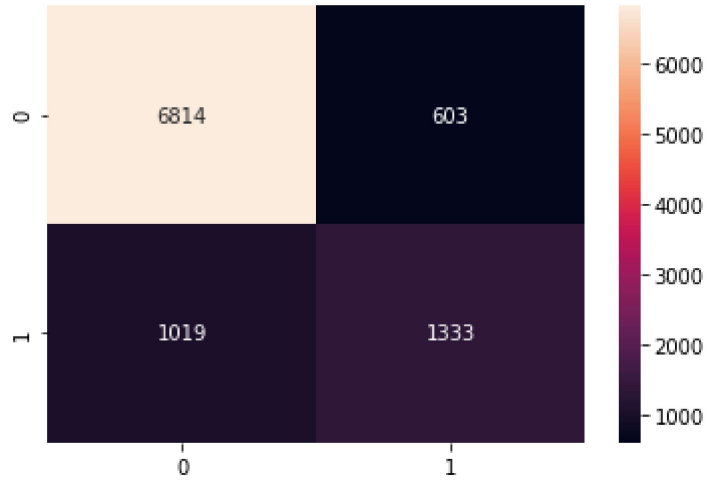
As the results aboe show KNN performance is weak compared to Logistic Regression.Below are the classification report and confusion matrix for KNN

In [191]:
```
print("KNN Classification Report")
print(classification_report(y_test,y_p))
```

```
KNN Classification Report
              precision    recall  f1-score   support

           0       0.87      0.92      0.89      7417
           1       0.69      0.57      0.62      2352

    accuracy                           0.83      9769
   macro avg       0.78      0.74      0.76      9769
weighted avg       0.83      0.83      0.83      9769
```

In [192]:
```
print("KNN confusion matrix")
cm['knn'] = confusion_matrix(y_test,y_p)
sb.heatmap(cm['knn'], annot=True, fmt='d');
```

```
KNN confusion matrix
```



Random forest classification

In [193]:
```
from sklearn.ensemble import RandomForestClassifier
param_grid = {
    'n_estimators': np.arange(50,200,20),
    'max_features': ["auto","sqrt","log2"]
}
rf = GridSearchCV(RandomForestClassifier(n_jobs=-1), param_grid=param_grid)
```

In [194]:
```
rf.fit(x_train,y_train);
```

In [195]:
```
rf.best_params_
```

Out[195]:
```
{'max_features': 'sqrt', 'n_estimators': 130}
```

In [196]:
```
y_p = rf.predict(x_test)
rf_scores = error_measures(y_test, y_p, "RF")
scores = pd.concat([scores,rf_scores], axis=1)
scores
```

Out[196]:

	Logistic Regression	KNN	RF
Accuracy	0.85	0.83	0.85
Recall	0.61	0.57	0.63
F1 Score	0.66	0.62	0.67
Roc AuC	0.77	0.74	0.77

In [197]:
```
print(classification_report(y_test, y_p))
```

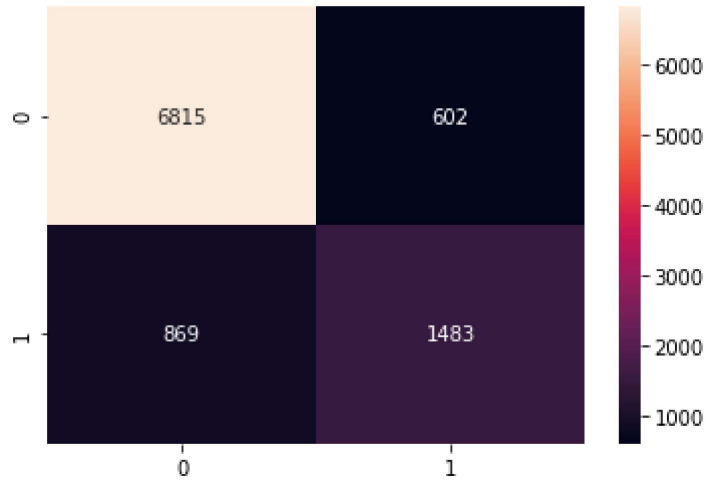
```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      7417
           1       0.71      0.63      0.67      2352

    accuracy                           0.85      9769
   macro avg       0.80      0.77      0.79      9769
weighted avg       0.84      0.85      0.85      9769
```

As the above results show Random Forest Classifer gave better results compared to logistic regression and KNN.

In [198]:
```
cm['rf'] = confusion_matrix(y_test,y_p)
```

In [199]:
```
print("RF confusion matrix")
sb.heatmap(cm['rf'], annot=True, fmt='d');
```

```
RF confusion matrix
```



Oversampling minority class to improve model fit

In this section I will oversample the minorty class (individuals with income >50K) to improve the scores on the minority class.

In [200]:
```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
x_train_res, y_train_res = ros.fit_resample(x_train, y_train)
```

In [201]:
```
x_train.shape, x_train_res.shape, y_train.shape, y_train_res.shape
```

Out[201]:
```
((22792, 84), (34606, 84), (22792,), (34606,))
```

Random Forest Classifier with oversampled data

In [202]:
```
param_grid = {
    'n_estimators': np.arange(50,200,20),
    'max_features': ["auto","sqrt","log2"]
}
rf_os = GridSearchCV(RandomForestClassifier(n_jobs=-1), param_grid=param_grid)
```

In [203]:
```
rf_os.fit(x_train_res, y_train_res);
```

In [204]:
```
y_p = rf_os.predict(x_test)
rf_os_scores = error_measures(y_test, y_p, "Random Forest OS")
scores = pd.concat([scores,rf_os_scores], axis=1)
scores
```

Out[204]:

	Logistic Regression	KNN	RF	Random Forest OS
Accuracy	0.85	0.83	0.85	0.84
Recall	0.61	0.57	0.63	0.70
F1 Score	0.66	0.62	0.67	0.67
Roc AuC	0.77	0.74	0.77	0.79

In [205]:
```
print("Random Forest Oversampled classification report")
print(classification_report(y_test, y_p))
```

```
Random Forest Oversampled classification report
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      7417
           1       0.65      0.70      0.67      2352

    accuracy                           0.84      9769
   macro avg       0.78      0.79      0.78      9769
weighted avg       0.84      0.84      0.84      9769
```

In [206]:
```
feature_importance = pd.DataFrame(rf_os.best_estimator_.feature_importances_, i
feature_importance.sort_values(by="weight", ascending=False)[:10]
```

Out[206]:

	weight
age	0.23
marital_status_ Married-civ-spouse	0.12
hours_per_week	0.11
education_num	0.11
capital_gain	0.08
marital_status_ Never-married	0.05
capital_loss	0.03
relationship_ Own-child	0.02
sex	0.02
relationship_ Not-in-family	0.02

Using oversampling techniques reduced precision and increased recall which is an expected

behaviour. The feature importances table above show that age, education level, marital status and hours per week account for 56% of the feature importance.

Summary

This project tried to use different classification models to predict whether a person earns more than 50K per year based on census data. The study applied logistic regression, nearest neighbourhoods and random forest classification models. The error measure scores show that Random Forest regression performs better than the two models.

Though the selected random forest model has an accuracy of 0.85 the recall is lower 0.63. In order to increase the recall for the target class of 1 (higher income individuals) random oversampling was used to resample the training data. The results show improved recall of 0.7 with a slight decline in accuracy (0.84 from 0.85).

Results from the final (oversampled Random Forest Classifier) show that age, education level, marital status and hours per week and important in explaining whether a person earns above 50K or not.

Next steps

The study applied different classification models to understand factors explaining whether an individual earns above 50K or not. The final model used has an accuracy of 0.84, recall of 0.7. The following next steps might improve the model fit better in the future:

1. **Using different oversampling techniques**: the study applied random oversampling on the training data. Running the models with SMOTE and ADASYN
2. **Employing other models**: bagging models are not implemented in this study and considering bagging might increasethe fit compared to random forest.

========== Thank you!
==========