Predict whether income exceeds $50k/yr using Machine Learning Algorithms

Name: Aleena Alby
Student Id: 11865340
Faculty of engineering, Environment and Computing (Coventry University)
albya@uni.coventry.ac.uk

Abstract— **Economic inequality is a prominent problem in the United States, and many citizens desire a fair share of the country's wealth. To maintain stable economic circumstances in the country, the majority of people must earn similar amounts of money. Many governments are attempting to understand the important regions and possibilities in order to examine what elements cause people to earn a living, therefore stabilizing the nation's revenue. We use the US census data from the UCI machine learning repository for this project. To categorize whether a person earns more than $50,000 or less than $50,000, machine learning approaches such as Logistic Regression, Support Vector Machines, and Decision Trees are employed. Precision, Recall, and F1 score were utilized as assessment measures because this is a classification problem. Decision tree performed the best out of the three models tested, with 86 percent accuracy, 79 percent macro F1, and 87 percent weighted average F1.**

Keywords—Logistic Regression, Support Vector Machines, Decision Trees, Accuracy, Precision, Recall and F1 score

I. INTRODUCTION

In today's environment, data is everywhere, and machine learning algorithms may be used to anticipate or categorize any situation. Training machine learning models is straightforward and quick because to the large-scale developed architecture like server less computing and rapid processing. In many nations, the issue of economic inequality is a serious concern. Governments cannot fix the problem immediately by providing cash assistance to the impoverished. Governments must understand the elements that influence an individual's ability to generate money. People in the United States want everyone to earn the same amount of money and expect a fair share of the riches in society. The federal government must gather census data and visualize and anticipate what characteristics and variables are important for projecting individual income.

The goal of this research is to perform a detailed analysis in order to determine the important components that are required to increase an individual's income. Such an analysis can be effective in determining the essential aspects and regions that can considerably boost an individual's income level. We use machine learning methods including Logistic Regression, Support Vector Machines, and Decision Trees in our research. Precision, Recall, and F1 score are used to evaluate the algorithms.

The report is organized as follows: an introduction, a review of the literature, methodologies, experimental setup, findings, and a conclusion.

II. LITERATURE REVIEW

The study 'Income categorization using Adult Census Data' [1] uses a census data set to determine whether an individual's income is greater than $50,000 or less than $50,000. The authors employed Logistic Regression, Nave Bayes, Decision Trees, k-Nearest Neighbor, SVM, and Gradient Boosting among other machine learning models. When all of the algorithms were compared, the best accuracy was 87 percent.

The authors of the research 'decision tree classifier to forecast income levels' [2] utilized a random forest classifier to categorize income between $50,000 and $50,000. The information comes from the UCI machine learning repository, which contains 32,251 people with 13 characteristics from the 1994 census collection. When compared to decision trees and nave bayes classifiers, the random forest classifier was more accurate. On the test data, the prediction model was 85 percent accurate. The top five features are shown using the decision trees algorithm's feature importance technique.

In the paper ‘Machine Learning on UCI Adult data set using various classifier algorithms and scaling up the accuracy using Extreme Gradient Boosting’ [3] used various machine learning algorithms like XGBOOST, Random Forest and stacking of models for predicting the income greater than 50k or less than 50k. XGBOOST algorithm has performed well with 87% accuracy.

The authors of the publication 'Income Prediction through Support Vector Machines' [4] employed principal components analysis and support vector machines to categorize income that is more than or equal to $50,000. The information was obtained from the United States Census Bureau. By utilizing PCA before training the algorithm, the authors were able to attain an accuracy of 84% and a 60% reduction in computing time.

Taking into account past studies, all of the researchers utilized the same data, which was obtained from the UCI machine learning library. For our study, we'll employ the same data and apply machine learning models that are routinely used, such as logistic regression, decision trees, and random and support vector machines. We try to find best parameters for the model using GridSearchCV by hyperparameter tuning and analyze will our results match or we can achieve greater accuracy compared to previous results.

III. DATASET AND METHODOLOGIES

A. Data set

The Data set was captured from the UCI machine learning repository. The Data set consists of income census data of United states nation. The dataset consists of 48842 rows and 14 columns. The target value consists of two values which are more than 50k and less than 50k.

Column Name	Column description
Age	The age of the person
Work class	The type of class of employment
Fnlwgt	Final weight
Education	The highest education of the person
Education-num	Numerical value of the education
Marital-status	Indicates marital-status of the person
Occupation	Occupation of the person
Relationship	Represents how this person this related to others
Race	Describes the race of the person
Sex	Describes the sex of the person
Capital-gain	Capital-gain of the person
Capital-loss	Capital-loss of the person
Hours-per-work	Hours worked per person in the week
Native-country	Country of origin of the person
Income (target variable)	<=50k, >50k

B. Methodologies

Logistic Regression: Logistic Regression is a classification algorithm which is used for classification purpose. Logistic regression is similar to linear regression with sigmoid function applied. The loss function applied to the logistic regression is log loss. Logistic regression is best used for binary outcome i.e., given an input it should predict the class.

Support Vector Machines: Support vector machines are supervised learning models which are used to classification. Support vectors are used to gain more confidence in classification. Greater the margin line distance between two classes the stronger the model

Decision Tree: Decision Tree are supervised learning algorithms used for both classification and regression. Decision tree algorithms are non-parametric models which learns and create rules from the data. Decision tree are prone to overfitting and can be avoided by deciding the depth of the tree and pre-pruning techniques.

IV. EXPERIMENT SETUP

We begin the procedure by importing all of the required libraries. We'll need Panda's libraries to read the dataset, load it into a data frame, and conduct computations on it for our experiment. To plot bar plots and conduct visualizations on the data, we load the Matplotlib and Seaborn libraries. Next, we import the Sklearn(scikit-learn) package, which allows us to utilize and train machine learning models as well as evaluate them using metrics such as the Classification report and the Confusion matrix. GridSearchCV is used to do hyperparameter tweaking for models in order to choose the optimal parameters. GridSearchCV is imported from sklearn model selection package. The below code snippet shows the necessary libraries we imported.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", color_codes=True)
%matplotlib inline

from sklearn.model_selection import GridSearchCV
import cv2

from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

Code snippet 1: Importing Libraries

We use Pandas' read_csv function to read the data into the data frame after loading the relevant libraries. There are two different files for training and testing data in the file. Both the training and testing data are loaded into separate data frames named 'dftrain' and 'dftest' respectively.

Using info() function we can know the attributes in the DataFrame, type of attributes, number of rows and columns in a DataFrame. Below is the screen shot of dftrain.info() executed cell.

```
dftrain.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 32560 entries, 1 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32560 non-null  int64
 1   workclass       32560 non-null  object
 2   fnlwgt          32560 non-null  float64
 3   education       32560 non-null  object
 4   education-num   32560 non-null  int64
 5   maritalstatus   32560 non-null  object
 6   occupation      32560 non-null  object
 7   relationship    32560 non-null  object
 8   race            32560 non-null  object
 9   sex             32560 non-null  object
 10  capital-gain    32560 non-null  int64
 11  capital-loss    32560 non-null  int64
 12  hours-per-week  32560 non-null  int64
 13  native-country  32560 non-null  object
 14  salary          32560 non-null  int64
dtypes: float64(1), int64(6), object(8)
memory usage: 5.0+ MB
```

Output 1: dftrian.info()

As per the above screen shot, we can see the DataFrame consists of 15 columns and 32560 entries. The target/ dependent variable is 'salary' and remaining are independent variables. Now we check the test data using dftest.info ().

```
dftest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16280 entries, 1 to 16280
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             16280 non-null  int64
 1   workclass       16280 non-null  object
 2   fnlwgt          16280 non-null  float64
 3   education       16280 non-null  object
 4   education-num   16280 non-null  int64
 5   maritalstatus   16280 non-null  object
 6   occupation      16280 non-null  object
 7   relationship    16280 non-null  object
 8   race            16280 non-null  object
 9   sex             16280 non-null  object
 10  capital-gain    16280 non-null  int64
 11  capital-loss    16280 non-null  int64
 12  hours-per-week  16280 non-null  int64
 13  native-country  16280 non-null  object
 14  salary          16280 non-null  int64
dtypes: float64(1), int64(6), object(8)
memory usage: 2.0+ MB
```

Output 2: dftest.info()

As per the above output screen shot, we can see the DataFrame consists of 16280 rows and 15 columns. So that train data consists of 32560 rows and test data consists of 16280 rows.

I noticed missing values tagged with '?' in the first few rows of both the dftrain and dftest DataFrames, so I generated a distinct value for '?' called other and replaced '?' with 'other'. In the appendix first output screen shot, you'll find the first few rows of both train and test screenshots.

The below screen shot is the code snippet for replacing '?' with 'other' in the data frames.

```
dftrain.replace(' ?', 'other', inplace=True)
dftest.replace(' ?', 'other', inplace=True)
```

Code snippet 2: replacing values

Now we check for missing values in both the training and test DataFrames. As per the below screen shot, we can see there are no missing in the data frame.

```
missingvalues(dftest)
```

	0	1
0	age	0
1	workclass	0
2	fnlwgt	0
3	education	0
4	education-num	0
5	maritalstatus	0
6	occupation	0
7	relationship	0
8	race	0
9	sex	0
10	capital-gain	0
11	capital-loss	0
12	hours-per-week	0
13	native-country	0
14	salary	0

Output 3: dftest Miss count

```
missingvalues(dftrain)
```

	0	1
0	age	0
1	workclass	0
2	fnlwgt	0
3	education	0
4	education-num	0
5	maritalstatus	0
6	occupation	0
7	relationship	0
8	race	0
9	sex	0
10	capital-gain	0
11	capital-loss	0
12	hours-per-week	0
13	native-country	0
14	salary	0

Outpu4: dftrain Miss count

Now we turn the target variable 'salary' from categorical into numerical values. In column 'salary' we replace the string '>50k' with 1 and '<=50k' with 0. Below is the code for replacing the categorical values with 0 and 1.

```
dftrain['salary'] = dftrain['salary'].apply(lambda x: 1 if x==' >50K' else 0)
dftest['salary'] = dftest['salary'].apply(lambda x: 1 if x==' >50K.' else 0)
```

Code snippet 3: Replacing target variable with numerical

A. Data visualization

Age

We first visualize the age column. The below plot is the histogram of the age variable. As per the histogram we can see there are more individuals with age between 20 and 40.

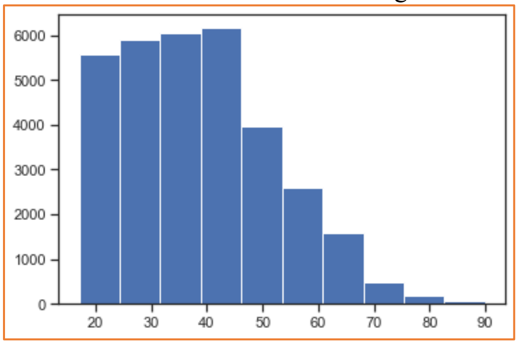


Figure 1: Histogram of Age

Workclass

We visualize the column workclass and check the values counts of the variable.

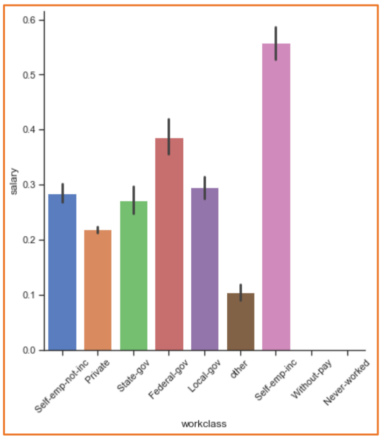


Figure 2: Factor plot of Workclass & salary

As per the above count plot there are less count for without-pay and never-worked values, so we merge these two into one value called 'never-worked'. Below is the code snippet for replacing 'Without-pay' value with 'Never-worked' value as both look similar.

```
dftrain['workclass'].replace(' Without-pay', ' Never-worked', inplace=True)
dftest['workclass'].replace(' Without-pay', ' Never-worked', inplace=True)
```

Code snippet 4: Merging classes into one class

Fnlwgt

As fnlwgt column is a continuous variable we use describe method to check the central tendency and dispersion of the data. As the central dispersion is more, I have performed scaling of the data by calculating logarithmic values. Below is the code snippet for applying logarithmic.

```
dftrain['fnlwgt'] = dftrain['fnlwgt'].apply(lambda x: np.log1p(x))
dftest['fnlwgt'] = dftest['fnlwgt'].apply(lambda x: np.log1p(x))

dftrain['fnlwgt'].describe()

count    32560.000000
mean        11.983800
std          0.630735
min          9.416216
25%         11.677019
50%         12.091582
75%         12.376050
max         14.210727
Name: fnlwgt, dtype: float64
```

Code snippet 5: log transformation to fnlwgt variable

Education

We visualize the education column by using sns factor plot function passing x value as 'education' and y value as 'salary'.

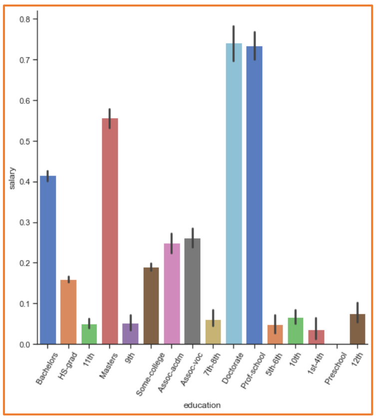


Figure 3: Factor plot for education and salary

As per the above factor plot, we can see primary school is divided into grades so we merger all the grades into primary. The below function is used to replace all the grades with one value called 'Primary'

```
def primary(x):
    if x in [' 1st-4th', ' 5th-6th', ' 7th-8th', ' 9th', ' 10th', ' 11th', ' 12th']:
        return ' Primary'
    else:
        return x

dftrain['education'] = dftrain['education'].apply(primary)
dftest['education'] = dftest['education'].apply(primary)
```

Code snippet 6: Merging all the grades to primary

The below factor plot of education after replacing all the grades with 'Primary'.

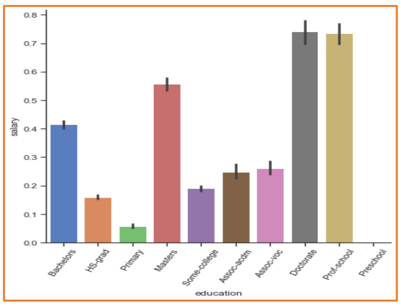


Figure 4: Factor plot for education and salary

Education num

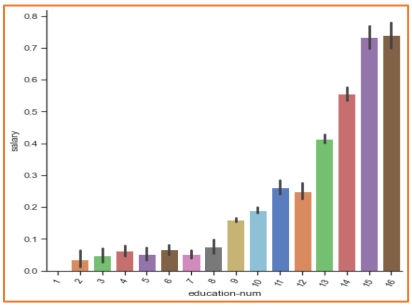


Figure 5: Factor plot for Education-num and salary

The bar plot looks good and for each level of education the count is more as it indicates individuals with highest level of education earn more than 50k

Marital Status

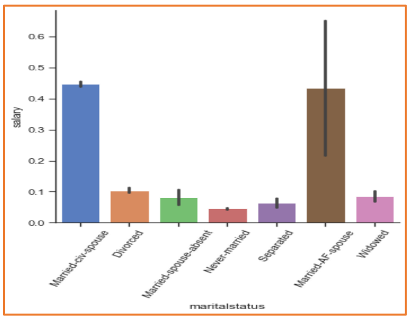


Figure 6: Factor plot for marital status and salary

As Married-AF-spouse features and Married-civ-spouse are similar we can merge them into one class.

```
dftrain['maritalstatus'].replace(' Married-AF-spouse', ' Married-civ-spouse', inplace=True)
dftest['maritalstatus'].replace(' Married-AF-spouse', ' Married-civ-spouse', inplace=True)
```

Code snippet 7: Replacing unique values

After merging we plot the factor plot between marital status and salary column

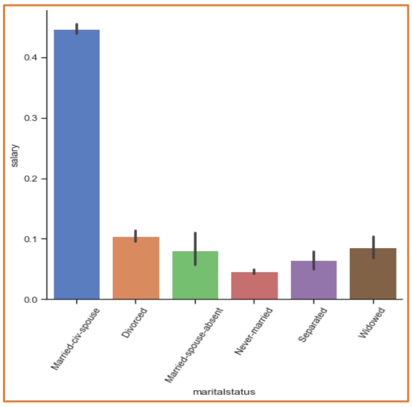


Figure 7: Factor plot for marital status and salary

Country

As per the country column factor plot attached in appendix there are many countries and if we perform one hot encoding there can be many features forming. So, we replace the countries with one geographical country or continent name or status of the country.

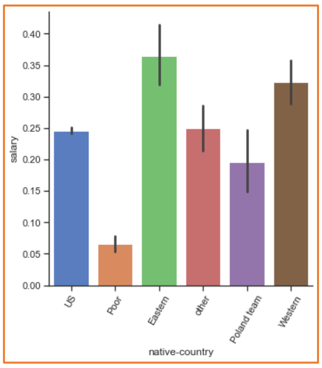


Figure 12: Factor plot for native-country and salary

We now join both train and test data and perform one hot encoding to transform all the categorical columns to numerical columns and again separate the joined data into train and test data.

```
joint.head()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	salary	workclass: Federal-gov	workclass: Local-gov	workclass: Never-worked	...	race: Other	race: White	sex: Female	sex: Male	native-country:Eastern	n country:P
1	50	11.330348	13	0	0	13	0	0	0	0	...	0	1	0	1	0	
2	38	12.281398	9	0	0	40	0	0	0	0	...	0	1	0	1	0	
3	53	12.366157	7	0	0	40	0	0	0	0	...	0	0	0	1	0	
4	28	12.732013	13	0	0	40	0	0	0	0	...	0	0	1	0	0	
5	37	12.558780	14	0	0	40	0	0	0	0	...	0	1	1	0	0	

5 rows × 64 columns

Output 5: sample date of joined after one hot encoding

This is the screen shot of the first 5 rows of the joined data after performing one hot encoding. The total columns after one-hot encoding are 64.

Now we again split the data into train and test by using below code.

```
train = joint.head(dftrain.shape[0])
test = joint.tail(dftest.shape[0])
```

Code snippet 8: train and test data

We perform scaling of the data using standard scalar library. Below is the code for scaling of the data using standard scaler.

```
Xtrain = train.drop('salary', axis=1)
ytrain = train['salary']

Xtest = test.drop('salary', axis=1)
ytest = test['salary']

scaler = StandardScaler()
scaler.fit(Xtrain)
Xtrain = scaler.transform(Xtrain)
Xtest = scaler.transform(Xtest)
```

Code snippet 9: scaling the data

B. Machine Learning Models

Logistic Regression

Below is the code snippet for initiating and fitting logistic regression.

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
param_grid = {'C': [0.1, 0.4, 0.7, 1, 4, 7, 10]}
grid1 = GridSearchCV(lr, param_grid).fit(Xtrain, ytrain)
print("Grid Logistic Regression: ", grid1.best_score_, grid1.best_params_)

Grid Logistic Regression:  0.8525184275184277 {'C': 0.1}

ypred=grid1.predict(Xtest)
```

Code snippet 10: Logistic regression model

Support Vector Machines

Below is the code snippet for fitting machine learning model.

```
from sklearn.svm import SVC
svc = SVC()
svc.fit(Xtrain, ytrain)

SVC()

ypred=svc.predict(Xtest)
```

Code snippet 11: SVM model

Decision Tree

Below is the code for fitting decision tree classifier.

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
param_grid = {'max_depth': [10, 40, 70, 100, 400, 700, None],
              'criterion': ['gini','entropy']}
grid1 = GridSearchCV(dtc, param_grid).fit(Xtrain, ytrain)
print("Grid DTC: ", grid1.best_score_, grid1.best_params_)

Grid DTC:  0.855497542997543 {'criterion': 'gini', 'max_depth': 10}

ypred=grid1.predict(Xtest)
```

Code snippet 12: Decision Tree model

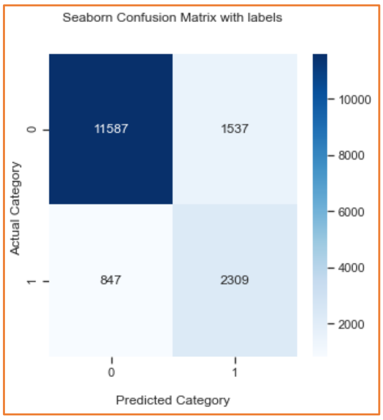
V. RESULTS

In this section we analyze the results of each model used. We use both classification report and confusion matrix to analyze the results and compare the models. By visualizing the classification report we can get accuracy, recall and F1 score of each model.

Logistic Regression

	precision	recall	f1-score	support
0	0.93	0.88	0.91	13124
1	0.60	0.73	0.66	3156
accuracy			0.85	16280
macro avg	0.77	0.81	0.78	16280
weighted avg	0.87	0.85	0.86	16280

As per the classification report of Logistic Regression the accuracy is 85% and weighted average F1 score id 86%.

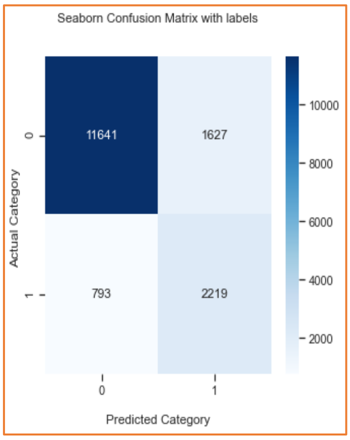


As per the confusion matrix of logistic regression the model has misclassified classes i.e., the false positives and false negatives are more.

Support Vector Machines

	precision	recall	f1-score	support
0	0.94	0.88	0.91	13268
1	0.58	0.74	0.65	3012
accuracy			0.85	16280
macro avg	0.76	0.81	0.78	16280
weighted avg	0.87	0.85	0.86	16280

As per the Support vector Machines classification report the accuracy of the model is 85% and weighted average f1 score is 86%.

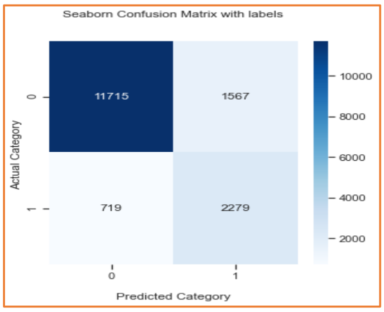


As per the confusion matrix of the support vector machines the false positive are less compared to logistic regression. The false negatives are more compared to the logistic regression.

Decision Tree

	precision	recall	f1-score	support
0	0.94	0.88	0.91	13282
1	0.59	0.76	0.67	2998
accuracy			0.86	16280
macro avg	0.77	0.82	0.79	16280
weighted avg	0.88	0.86	0.87	16280

According to the classification report of the decision tree the accuracy achieved is 86% and weighted average F1 score is 87%.



As per the confusion matrix the false negatives are less compared to the previous two models and as per the classification report and confusion matrix. The decision tree classifier best.

VI. CONCLUSION

We investigated how to use machine learning models on US census data to classify whether an individual's income exceeds $50,000 or not in this study.According to the results, the decision tree performed the best, with an accuracy of 86%. Previous study employed decision trees and reached an accuracy of 85 percent, while we achieved an accuracy of 86 percent via feature modification and scaling the data. The authors used the XGBoost algorithm and reached an accuracy of 87 percent, therefore we will apply boosting techniques in the future to improve accuracy and reduce false negatives and false positives..

REFERENCES

1. Vidya Chockalingam, Sejal Shah and Ronit Shaw:, "Income Classification using Adult Census Data", [online] Available: https://cseweb.ucsd.edu/classes/wi17/cse258-a/reports/a120.pdf.
2. Sisay Menji Bekena, "Using decision tree classifier to predict income levels", *Munich Personal RePEc Archive*, July 2017.
3. Mohammed Topiwalla:, "Machine Learning on UCI Adult data Set Using Various Classifier Algorithms And Scaling Up The Accuracy Using Extreme Gradient Boosting" in , University of SP Jain School of Global Management.
4. Alina Lazar:, "Income Prediction via Support Vector Machine", *International Conference on Machine Learning and Applications - ICMLA 2004*, 16-18 December 2004.
5. Dataset Link: https://archive.ics.uci.edu/ml/datasets/census+income

APPENDIX

Code screen shots:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", color_codes=True)
%matplotlib inline
```

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

```
dftrain = pd.read_csv(r"/Users/aleenaalby/Desktop/adult(1).data")
```

```
dftest = pd.read_csv(r"/Users/aleenaalby/Desktop/adulttest (1).test")
```

```
dftrain
```

	age	workclass	fnlwgt	education	education-num	maritalstatus	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	salary
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

```python
dftrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64 
 5   maritalstatus   32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64 
 11  capital-loss    32561 non-null  int64 
 12  hours-per-week  32561 non-null  int64 
 13  native-country  32561 non-null  object
 14  salary          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```python
dftest
```

	age	workclass	fnlwgt	education	education-num	maritalstatus	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	salary
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K.
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K.
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K.
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K.
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
16276	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	United-States	<=50K.
16277	64	?	321403	HS-grad	9	Widowed	?	Other-relative	Black	Male	0	0	40	United-States	<=50K.
16278	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	United-States	<=50K.
16279	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	United-States	<=50K.
16280	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	United-States	>50K.

```python
dftest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16281 entries, 0 to 16280
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             16281 non-null  int64 
 1   workclass       16281 non-null  object
 2   fnlwgt          16281 non-null  int64 
 3   education       16281 non-null  object
 4   education-num   16281 non-null  int64 
 5   maritalstatus   16281 non-null  object
 6   occupation      16281 non-null  object
 7   relationship    16281 non-null  object
 8   race            16281 non-null  object
 9   sex             16281 non-null  object
 10  capital-gain    16281 non-null  int64 
 11  capital-loss    16281 non-null  int64 
 12  hours-per-week  16281 non-null  int64 
 13  native-country  16281 non-null  object
 14  salary          16281 non-null  object
dtypes: int64(6), object(9)
memory usage: 1.9+ MB
```

```python
dftrain.replace(' ?', 'other', inplace=True)
dftest.replace(' ?', 'other', inplace=True)
```

```python
# defining function for estimating missing values in each columns
def missingvalues(df):
    missing=[]
    col_list=df.columns
    for i in col_list:
        missingvalue=df[i].isnull().sum()
        missing.append(missingvalue)
    list_of_missing=pd.DataFrame(list(zip(col_list,missing)))
    return list_of_missing
```

```
missingvalues(dftest)
```

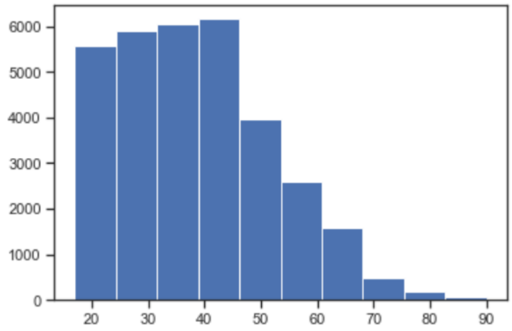
	0	1
0	age	0
1	workclass	0
2	fnlwgt	0
3	education	0
4	education-num	0
5	maritalstatus	0
6	occupation	0
7	relationship	0
8	race	0
9	sex	0
10	capital-gain	0
11	capital-loss	0
12	hours-per-week	0
13	native-country	0
14	salary	0

```
missingvalues(dftrain)
```

	0	1
0	age	0
1	workclass	0
2	fnlwgt	0
3	education	0
4	education-num	0
5	maritalstatus	0
6	occupation	0
7	relationship	0
8	race	0
9	sex	0
10	capital-gain	0
11	capital-loss	0
12	hours-per-week	0
13	native-country	0
14	salary	0

```
dftrain['salary'] = dftrain['salary'].apply(lambda x: 1 if x==' >50K' else 0)
dftest['salary'] = dftest['salary'].apply(lambda x: 1 if x==' >50K.' else 0)
```

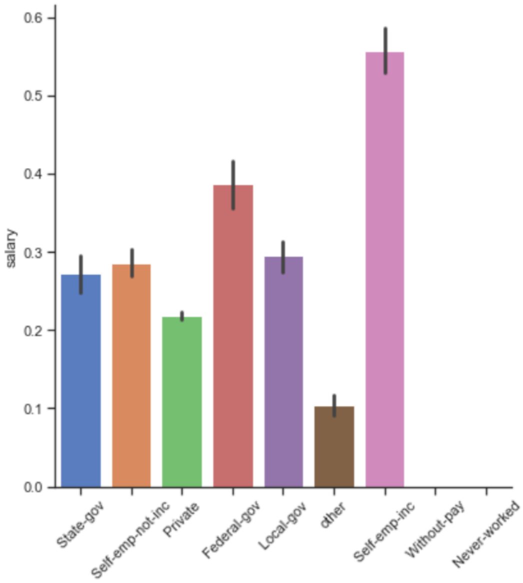
```
plt.hist(dftrain['age']);
```



```
dftrain.workclass.value_counts()
```

```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
other                 1836
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
```

```
sns.factorplot(x="workclass", y="salary", data=dftrain, kind="bar", size = 6,
palette = "muted")
plt.xticks(rotation=45);
```




```
dftrain['workclass'].replace(' Without-pay', ' Never-worked', inplace=True)
dftest['workclass'].replace(' Without-pay', ' Never-worked', inplace=True)
```

fnlgwt

```
dftrain['fnlwgt'].describe()
```

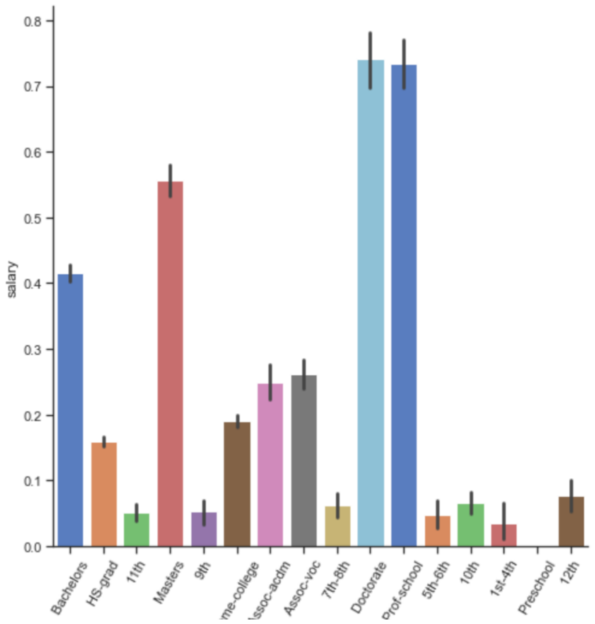
```
count    3.256100e+04
mean     1.897784e+05
std      1.055500e+05
min      1.228500e+04
25%      1.178270e+05
50%      1.783560e+05
75%      2.370510e+05
max      1.484705e+06
Name: fnlwgt, dtype: float64
```

```python
dftrain['fnlwgt'] = dftrain['fnlwgt'].apply(lambda x: np.log1p(x))
dftest['fnlwgt'] = dftest['fnlwgt'].apply(lambda x: np.log1p(x))
```

```python
dftrain['fnlwgt'].describe()
```

```
count    32561.000000
mean        11.983778
std          0.630738
min          9.416216
25%         11.676981
50%         12.091542
75%         12.376035
max         14.210727
Name: fnlwgt, dtype: float64
```

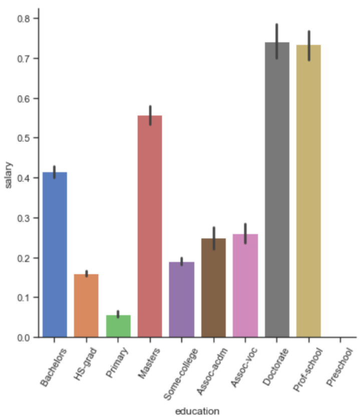
```python
sns.factorplot(x="education",y="salary",data=dftrain,kind="bar", size = 7,
palette = "muted")
plt.xticks(rotation=60);
```



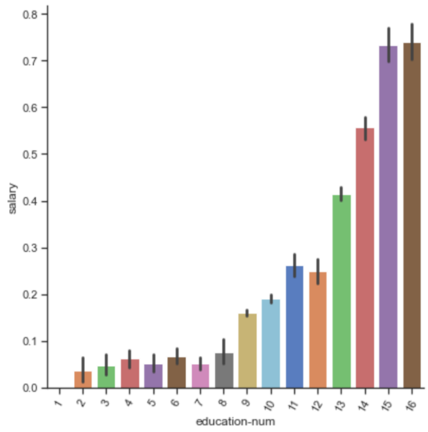

```python
def primary(x):
    if x in [' 1st-4th', ' 5th-6th', ' 7th-8th', ' 9th', ' 10th', ' 11th', ' 12th']:
        return ' Primary'
    else:
        return x
```

```python
dftrain['education'] = dftrain['education'].apply(primary)
dftest['education'] = dftest['education'].apply(primary)
```

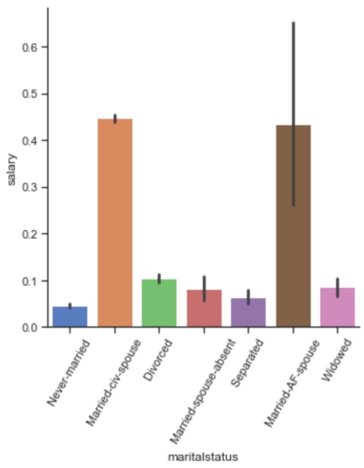
```python
sns.factorplot(x="education",y="salary",data=dftrain,kind="bar", size = 6,
palette = "muted")
plt.xticks(rotation=60);
```



```
sns.factorplot(x="education-num",y="salary",data=dftrain,kind="bar", size = 6,
palette = "muted")
plt.xticks(rotation=60);
```




```
sns.factorplot(x="maritalstatus",y="salary",data=dftrain,kind="bar", size = 5,
palette = "muted")
plt.xticks(rotation=60);
```



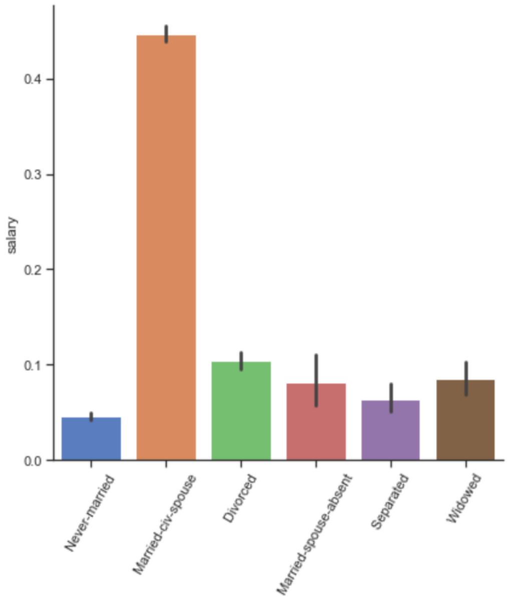

```
dftrain['maritalstatus'].value_counts()
```

```
 Married-civ-spouse        14976
 Never-married             10683
 Divorced                   4443
 Separated                  1025
 Widowed                     993
 Married-spouse-absent       418
 Married-AF-spouse            23
Name: maritalstatus, dtype: int64
```

There are very few Married-AF-spouse features. They are similar to Married-civ-spouse, so we can merge them

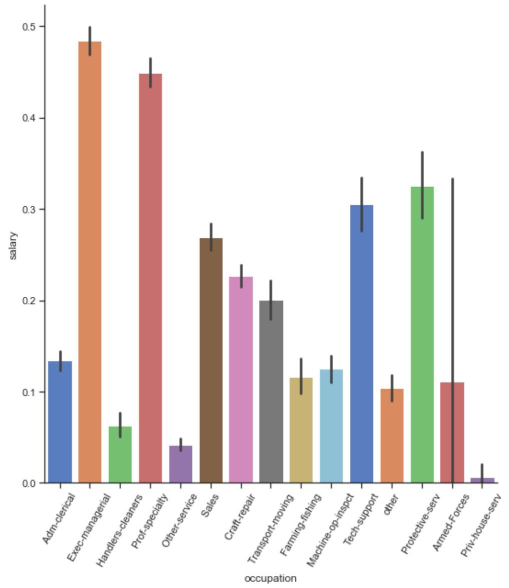
```
dftrain['maritalstatus'].replace(' Married-AF-spouse', ' Married-civ-spouse', inplace=True)
dftest['maritalstatus'].replace(' Married-AF-spouse', ' Married-civ-spouse', inplace=True)
```

```
sns.factorplot(x="maritalstatus",y="salary",data=dftrain,kind="bar", size = 6,
palette = "muted")
plt.xticks(rotation=60);
```




```
dftrain['occupation'].fillna(' 0', inplace=True)
dftest['occupation'].fillna(' 0', inplace=True)
```

```
sns.factorplot(x="occupation",y="salary",data=dftrain,kind="bar", size = 8,
palette = "muted")
plt.xticks(rotation=60);
```



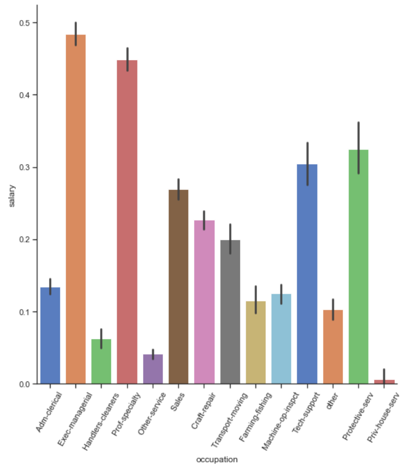

```
dftrain['occupation'].value_counts()
```

```
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3770
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
other                 1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: occupation, dtype: int64
```

Everything looks good, except Armed-Forces. They are similar to 0 and that's what we replace them with.

```
dftrain['occupation'].replace(' Armed-Forces', 'other', inplace=True)
dftest['occupation'].replace(' Armed-Forces', 'other', inplace=True)
```

```
sns.factorplot(x="occupation",y="salary",data=dftrain,kind="bar", size = 8,
palette = "muted")
plt.xticks(rotation=60);
```

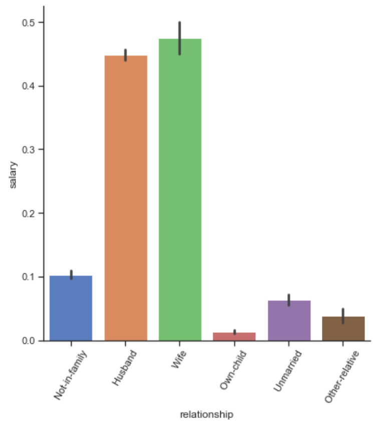


Relationship

```
dftrain.relationship.value_counts()
```

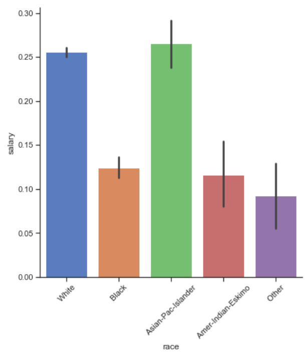
```
 Husband           13193
 Not-in-family      8305
 Own-child          5068
 Unmarried          3446
 Wife               1568
 Other-relative      981
Name: relationship, dtype: int64
```

```
sns.factorplot(x="relationship",y="salary",data=dftrain,kind="bar", size = 6,
palette = "muted")
plt.xticks(rotation=60);
```



Race

```
sns.factorplot(x="race",y="salary",data=dftrain,kind="bar", size = 6,
palette = "muted")
plt.xticks(rotation=45);
```




```
dftrain['race'].value_counts()
```

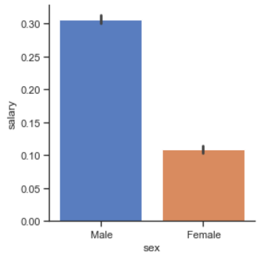
```
 White                 27816
 Black                  3124
 Asian-Pac-Islander     1039
 Amer-Indian-Eskimo      311
 Other                   271
Name: race, dtype: int64
```

```
dftrain['race'].value_counts()
```

```
White                 27816
Black                  3124
Asian-Pac-Islander     1039
Amer-Indian-Eskimo      311
Other                   271
Name: race, dtype: int64
```

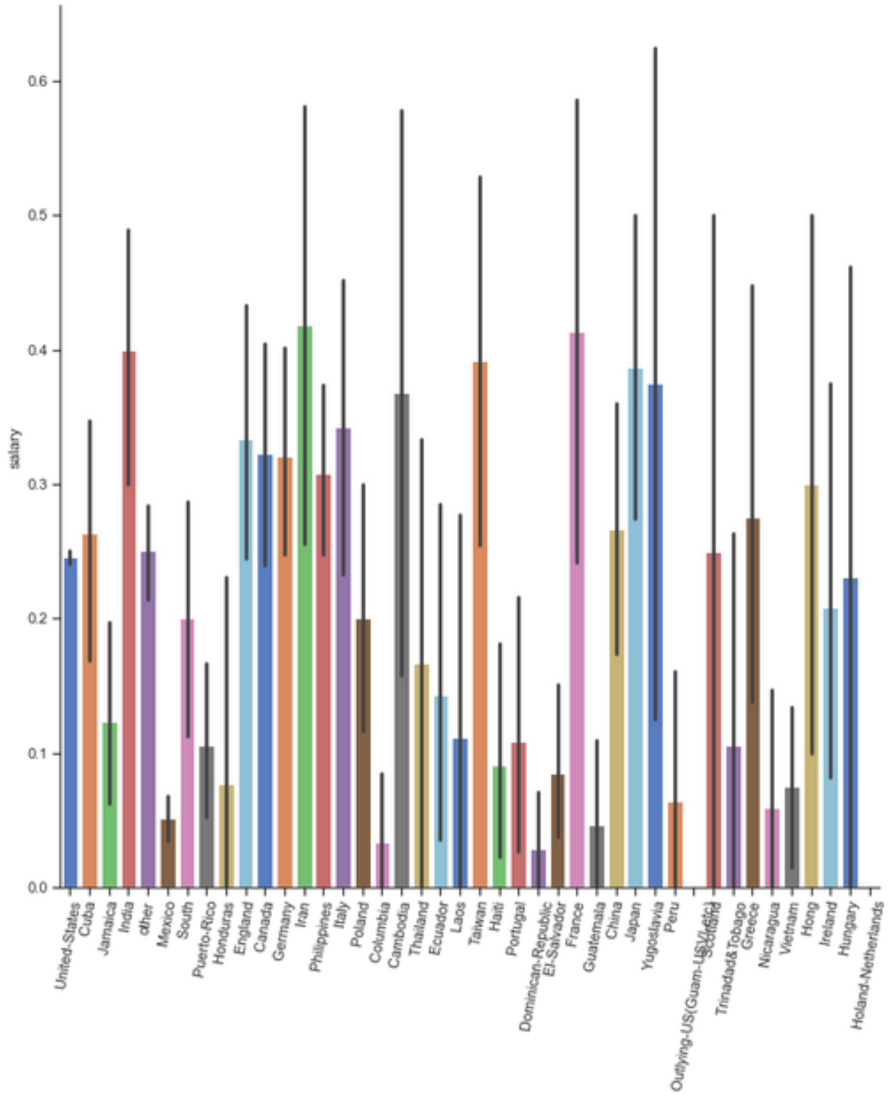
Sex

```
sns.factorplot(x="sex",y="salary",data=dftrain,kind="bar", size = 4,
palette = "muted");
```




```
dftrain['native-country'].fillna(' 0', inplace=True)
dftest['native-country'].fillna(' 0', inplace=True)
```

```
sns.factorplot(x="native-country",y="salary",data=dftrain,kind="bar", size = 10,
palette = "muted")
plt.xticks(rotation=80);
```



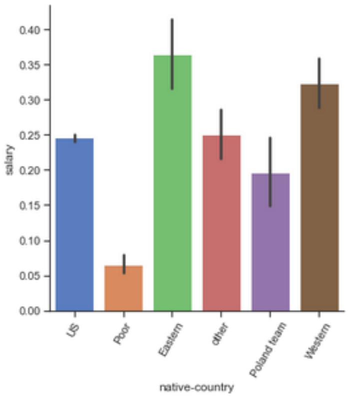
```
def native(country):
    if country in [' United-States', ' Cuba', ' 0']:
        return 'US'
    elif country in [' England', ' Germany', ' Canada', ' Italy', ' France', ' Greece', ' Philippines']:
        return 'Western'
    elif country in [' Mexico', ' Puerto-Rico', ' Honduras', ' Jamaica', ' Columbia', ' Laos', ' Portugal', ' Haiti',
                     ' Dominican-Republic', ' El-Salvador', ' Guatemala', ' Peru',
                     ' Trinadad&Tobago', ' Outlying-US(Guam-USVI-etc)', ' Nicaragua', ' Vietnam', ' Holand-Netherlands' ]:
        return 'Poor' # no offence
    elif country in [' India', ' Iran', ' Cambodia', ' Taiwan', ' Japan', ' Yugoslavia', ' China', ' Hong']:
        return 'Eastern'
    elif country in [' South', ' Poland', ' Ireland', ' Hungary', ' Scotland', ' Thailand', ' Ecuador']:
        return 'Poland team'

    else:
        return country
```

```
dftrain['native-country'] = dftrain['native-country'].apply(native)
dftest['native-country'] = dftest['native-country'].apply(native)
```

```
dftrain['native-country'].value_counts()
```

```
US             29265
Poor            1415
Western          677
other            583
Eastern          386
Poland team      235
Name: native-country, dtype: int64
```

```
sns.factorplot(x="native-country",y="salary",data=dftrain,kind="bar", size = 5,
palette = "muted")
plt.xticks(rotation=60);
```




```
print(dftest.isnull().sum())
```

```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
maritalstatus     0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
salary            0
dtype: int64
```

```
print(dftrain.isnull().sum())
```

```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
maritalstatus     0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
salary            0
dtype: int64
```

One-hot encoding

```
#merge datasets
joint = pd.concat([dftrain, dftest], axis=0)
```

```
joint.dtypes
```

```
age                 int64
workclass          object
fnlwgt            float64
education          object
education-num       int64
maritalstatus      object
occupation         object
relationship       object
race               object
sex                object
capital-gain        int64
capital-loss        int64
hours-per-week      int64
native-country     object
salary              int64
dtype: object
```

```
joint.head()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	salary	workclass: Federal-gov	workclass: Local-gov	workclass: Never-worked	...	race: Other	race: White	sex: Female	sex: Male	native-country:Eastern	native-country:Poland team	native-country:Poor	native-country:US	native-country:Western	native-country:other
0	39	11.258253	13	2174	0	40	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
1	50	11.330348	13	0	0	13	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
2	38	12.281398	9	0	0	40	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
3	53	12.366157	7	0	0	40	0	0	0	0	...	0	0	0	1	0	0	0	1	0	0
4	28	12.732013	13	0	0	40	0	0	0	0	...	0	0	1	0	0	0	0	1	0	0

5 rows × 64 columns

```
train = joint.head(dftrain.shape[0])
test = joint.tail(dftest.shape[0])
```

```
train
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	salary	workclass: Federal-gov	workclass: Local-gov	workclass: Never-worked	...	race: Other	race: White	sex: Female	sex: Male	native-country:Eastern	native-country:Poland team	native-country:Poor	native-country:US	native-country:Western	native-country:other
0	39	11.258253	13	2174	0	40	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
1	50	11.330348	13	0	0	13	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
2	38	12.281398	9	0	0	40	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
3	53	12.366157	7	0	0	40	0	0	0	0	...	0	0	0	1	0	0	0	1	0	0
4	28	12.732013	13	0	0	40	0	0	0	0	...	0	0	1	0	0	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	12.458010	12	0	0	38	0	0	0	0	...	0	1	1	0	0	0	0	1	0	0
32557	40	11.947140	9	0	0	40	1	0	0	0	...	0	1	0	1	0	0	0	1	0	0
32558	58	11.931050	9	0	0	40	0	0	0	0	...	0	1	1	0	0	0	0	1	0	0
32559	22	12.213500	9	0	0	20	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
32560	52	12.570466	9	15024	0	40	1	0	0	0	...	0	1	1	0	0	0	0	1	0	0

32561 rows × 64 columns

```
test
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	salary	workclass: Federal-gov	workclass: Local-gov	workclass: Never-worked	...	race: Other	race: White	sex: Female	sex: Male	native-country:Eastern	native-country:Poland team	native-country:Poor	native-country:US	native-country:Western	native-country:other
0	25	12.331837	7	0	0	40	0	0	0	0	...	0	0	0	1	0	0	0	1	0	0
1	38	11.405507	9	0	0	50	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
2	28	12.727696	12	0	0	40	1	0	1	0	...	0	1	0	1	0	0	0	1	0	0
3	44	11.984952	10	7688	0	40	1	0	0	0	...	0	0	0	1	0	0	0	1	0	0
4	18	11.547308	10	0	0	30	0	0	0	0	...	0	1	1	0	0	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
16276	39	12.280345	13	0	0	36	0	0	0	0	...	0	1	1	0	0	0	0	1	0	0
16277	64	12.680454	9	0	0	40	0	0	0	0	...	0	0	0	1	0	0	0	1	0	0
16278	38	12.834639	13	0	0	50	0	0	0	0	...	0	1	0	1	0	0	0	1	0	0
16279	44	11.337286	13	5455	0	40	0	0	0	0	...	0	0	0	1	0	0	0	1	0	0
16280	35	12.112580	13	0	0	60	1	0	0	0	...	0	1	0	1	0	0	0	1	0	0

16281 rows × 64 columns

```
Xtrain = train.drop('salary', axis=1)
ytrain = train['salary']

Xtest = test.drop('salary', axis=1)
ytest = test['salary']

scaler = StandardScaler()
scaler.fit(Xtrain)
Xtrain = scaler.transform(Xtrain)
Xtest = scaler.transform(Xtest)
```

Logistic regression data prediction

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
param_grid = {'C': [0.1, 0.4, 0.7, 1, 4, 7, 10]}
grid1 = GridSearchCV(lr, param_grid).fit(Xtrain, ytrain)
print("Grid Logistic Regression: ", grid1.best_score_, grid1.best_params_)
```

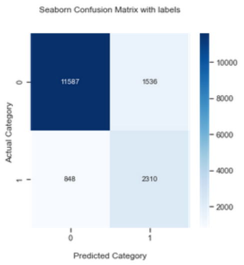
```
Grid Logistic Regression:  0.8525845304288417 {'C': 0.1}
```

```
ypred=grid1.predict(Xtest)
```

```
print(classification_report(ypred, ytest))
```

```
              precision    recall  f1-score   support

           0       0.93      0.88      0.91     13123
           1       0.60      0.73      0.66      3158

    accuracy                           0.85     16281
   macro avg       0.77      0.81      0.78     16281
weighted avg       0.87      0.85      0.86     16281
```

```
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

#Generate the confusion matrix
cf_matrix = confusion_matrix(ypred, ytest)

import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(5, 5))
ax = sns.heatmap(cf_matrix, annot=True, cmap='Blues',fmt="d")

ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Category')
ax.set_ylabel('Actual Category ');


## Display the visualization of the Confusion Matrix.
plt.show()
```




```python
from sklearn.svm import SVC
svc = SVC()
svc.fit(Xtrain, ytrain)
```

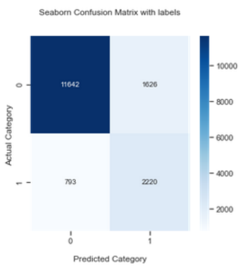
```
SVC()
```

```python
ypred=svc.predict(Xtest)
```

```python
print(classification_report(ypred, ytest))
```

```
              precision    recall  f1-score   support

           0       0.94      0.88      0.91     13268
           1       0.58      0.74      0.65      3013

    accuracy                           0.85     16281
   macro avg       0.76      0.81      0.78     16281
weighted avg       0.87      0.85      0.86     16281
```

```python
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

#Generate the confusion matrix
cf_matrix = confusion_matrix(ypred, ytest)

import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(5, 5))
ax = sns.heatmap(cf_matrix, annot=True, cmap='Blues',fmt="d")

ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Category')
ax.set_ylabel('Actual Category ');


## Display the visualization of the Confusion Matrix.
plt.show()
```




```python
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
param_grid = {'max_depth': [10, 40, 70, 100, 400, 700, None],
              'criterion': ['gini','entropy']}
grid1 = GridSearchCV(dtc, param_grid).fit(Xtrain, ytrain)
print("Grid DTC: ", grid1.best_score_, grid1.best_params_)
```

```
Grid DTC:  0.8558092989979217 {'criterion': 'entropy', 'max_depth': 10}
```

```python
ypred=grid1.predict(Xtest)
```

```python
print(classification_report(ypred, ytest))
```

```
              precision    recall  f1-score   support

           0       0.95      0.88      0.91     13482
           1       0.57      0.78      0.66      2799

    accuracy                           0.86     16281
   macro avg       0.76      0.83      0.78     16281
weighted avg       0.88      0.86      0.87     16281
```

```python
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

#Generate the confusion matrix
cf_matrix = confusion_matrix(ypred, ytest)

import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(5, 5))
ax = sns.heatmap(cf_matrix, annot=True, cmap='Blues',fmt="d")

ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Category')
ax.set_ylabel('Actual Category ');


## Display the visualization of the Confusion Matrix.
plt.show()
```