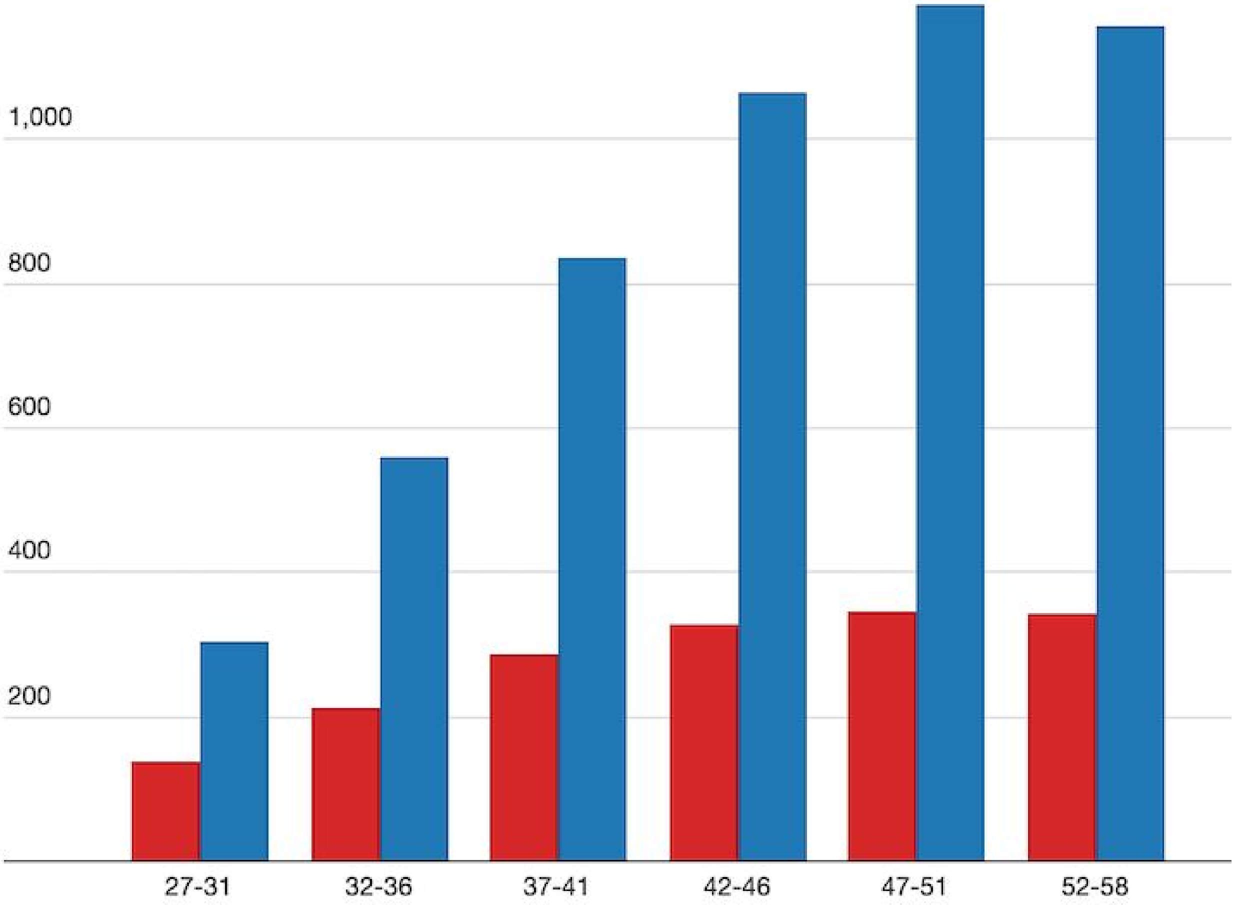
CAPSTONE2-MILESTONE 2



This report consists of data wrangling,exploratory data analysis and statistical inferences regarding adult data.

MEHMET KETENCI

PROBLEM: Our goal is to be able to classify given data with respect to certain income level.

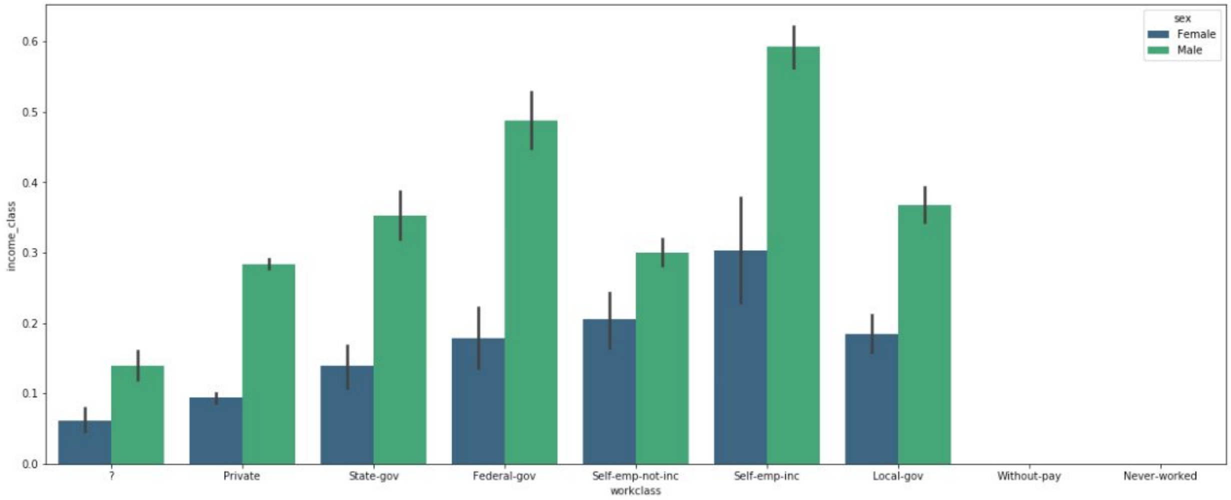
CLIENTS AND WHY DO THEY CARE: Result of this dataset has crucial importance for those industries below in determining tendency of group of people both high income level with low income level.

- **Marketing and e-marketing companies**:Companies may offer their different products with respect to different income levels.
- **Health, car, home and life insurance companies**: Data will be produced by statistical inferences and Eda with Machine Learning is beneficial for this industry due to mapping right customer and right insurance type and amount.
- **Investment industry**:This industry can aim right income class with respect to customers` different features.
- **Loan companies and banks**:Classified data can be used for this industry to match correct loan or credit amount with targeted people.
- **Travel agencies**: People have different vacation habits,some people may want to go to the seaside while some enjoys spending time mountains and some like museums or historical places.This industry may offer right travel options to correct people.

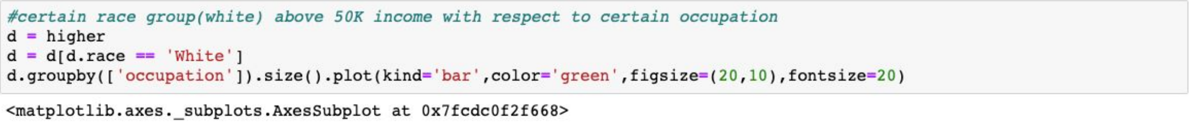
WHAT WILL CLIENTS DECIDE AFTER MY PROJECT: They can easily determine their target mass with simplified numbers or values and prominent and well-organized visuals.

HOW WILL I ACQUIRE THE DATA?

Data itself was obtained on http://archive.ics.uci.edu/ml/datasets/Adult

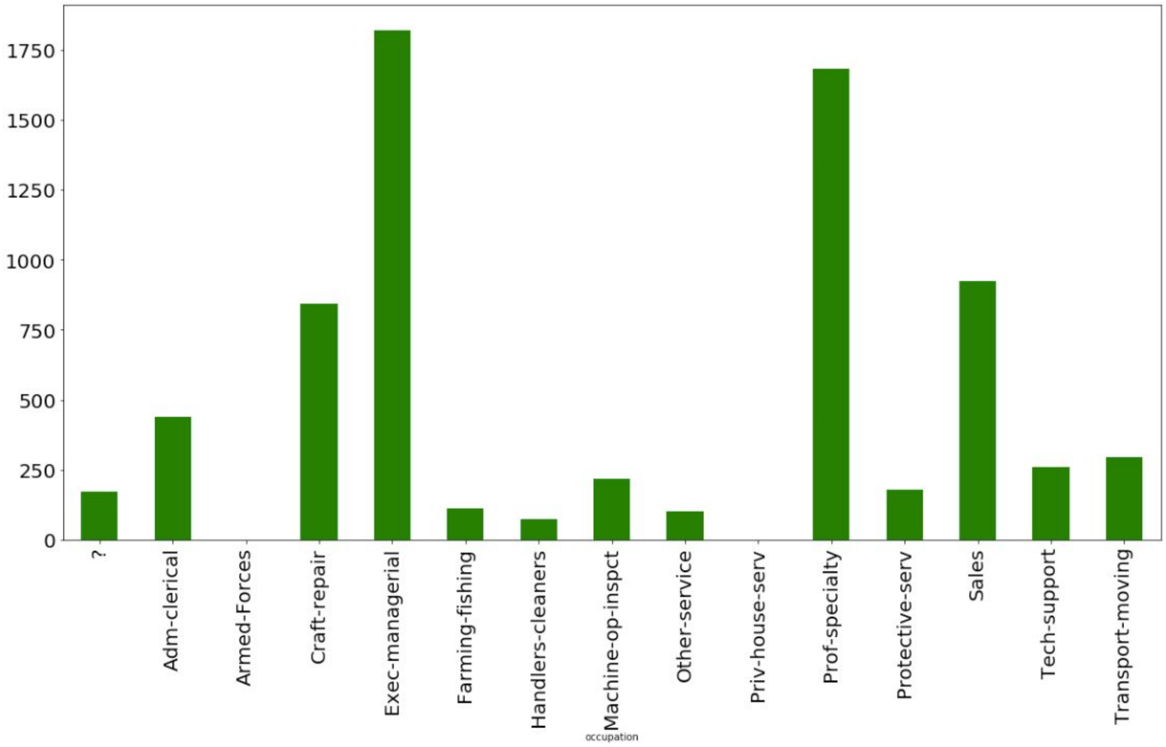


This graph shows the income distribution of groups that depend on a specific work class and gender.

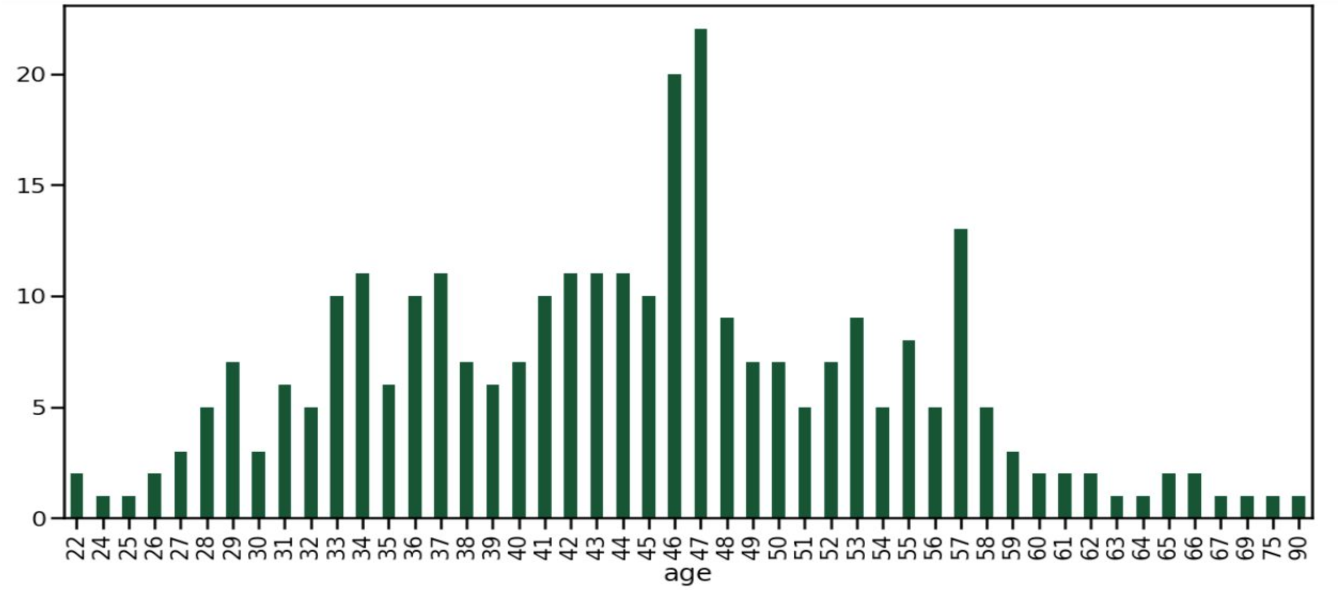


```
#certain race group(white) above 50K income with respect to certain occupation
d = higher
d = d[d.race == 'White']
d.groupby(['occupation']).size().plot(kind='bar',color='green',figsize=(20,10),fontsize=20)
```

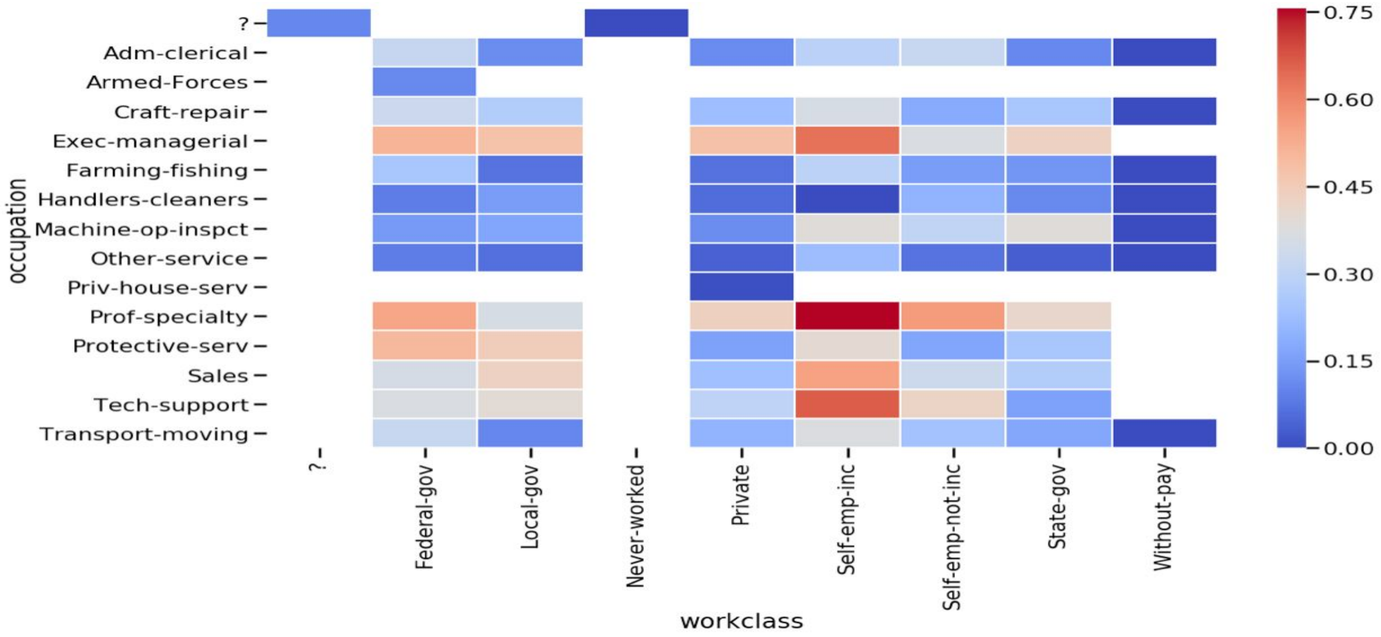
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcdc0f2f668>
```



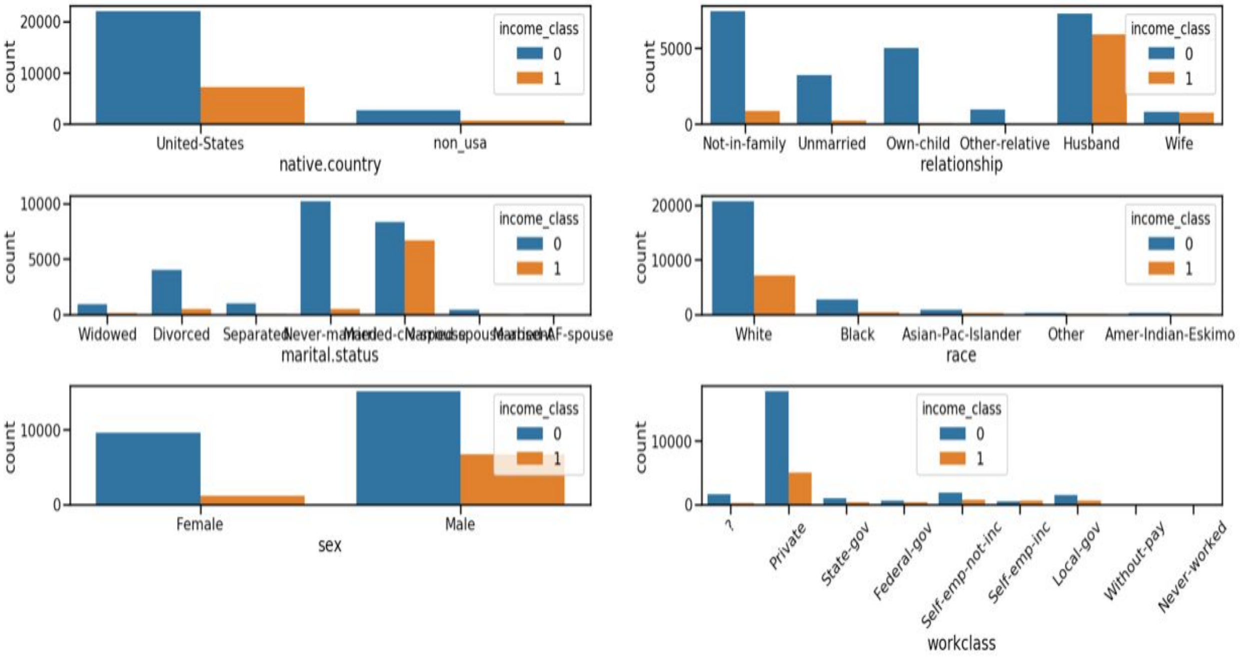
This graph shows the professional distribution of the white population with more than 50 thousand income. The top 5 consists of Exec-managerial,Prof-speciality,Sales,Craft-repair and Adm-clerical.



This graph shows the age distribution of the high-income group of black men. According to the graph, the surplus of people aged 46 and 47 is in the foreground.



Here we see the heatmap graph according to the income averages of the people working in a certain occupational group and sector and according to this graph (Prof-specialty, Self-emp-inc), (Exec-managerial, Self-emp-inc), (Tech-support, Self-emp) -inc), (Sales, Self-emp-inc) and () combinations have a high average income.



Here we also see various comparisons based on income classes.

CHI SQUARE TEST

In [ ]:
```
# H0:'Race' and 'Income' variables are independent.
# Ha:'Race' and 'Income' variables are not independent.
```

In [68]:
```
pd.crosstab(data['income'],data['race'])
```

Out[68]:

race	Amer-Indian-Eskimo	Asian-Pac-Islander	Black	Other	White
income					
<=50K	275	763	2737	246	20699
>50K	36	276	387	25	7117

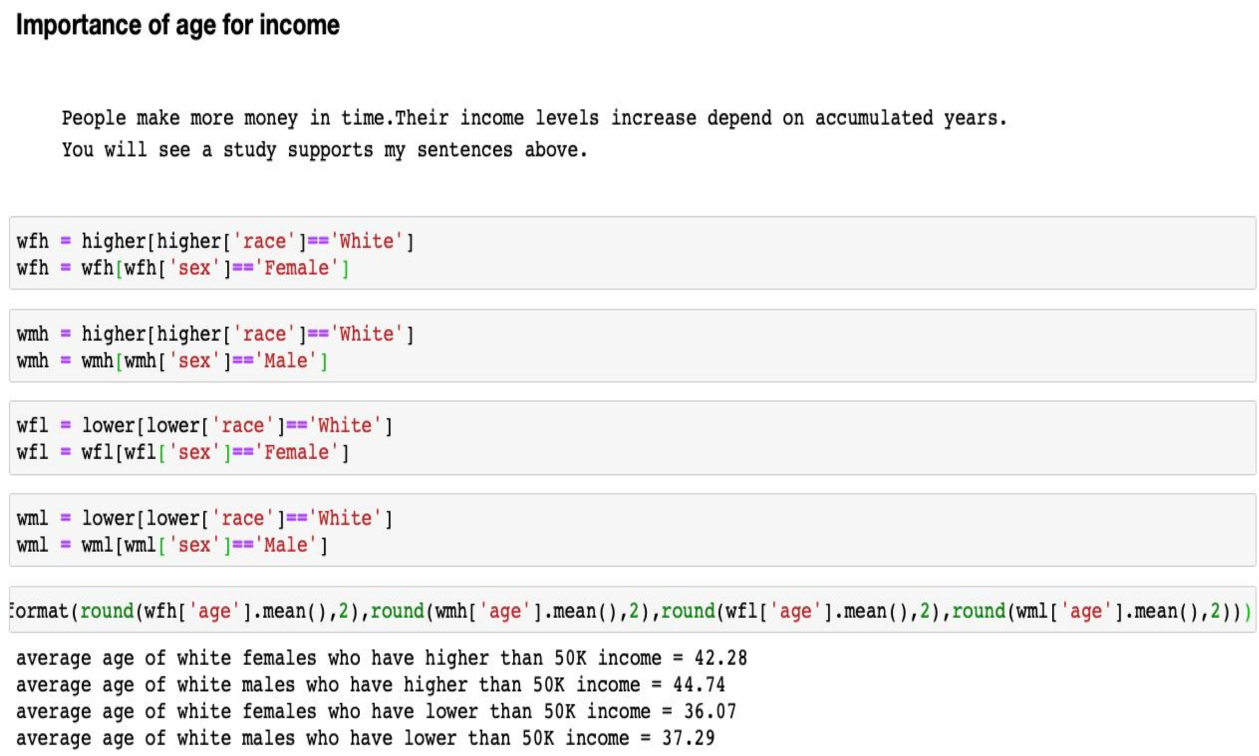
The null hypothesis states that knowing the race variables doesnt help us predict the variables of income. Alternative hypothesis is that knowing race variables might help us to predict income values.

In [72]:
```
from scipy.stats import chi2_contingency
chi2_contingency(pd.crosstab(data['race'],data['income']))
```

Out[72]:
```
(330.9204310085741,
 2.305960610160958e-70,
 4,
 array([[  236.10822763,    74.89177237],
        [  788.79886981,   250.20113019],
        [ 2371.71094254,   752.28905746],
        [  205.74060993,    65.25939007],
        [21117.64135008,  6698.35864992]]))
```

P-value less than 0.05(significance level), We then reject Null Hypothesis and accept alternative Hypothesis.

It means race is not an independent column.Income is correlated with race.



Importance of age for income

```
People make more money in time.Their income levels increase depend on accumulated years.
You will see a study supports my sentences above.
```

In [142]:
```
wfh = higher[higher['race']=='White']
wfh = wfh[wfh['sex']=='Female']
```

In [143]:
```
wmh = higher[higher['race']=='White']
wmh = wmh[wmh['sex']=='Male']
```

In [144]:
```
wfl = lower[lower['race']=='White']
wfl = wfl[wfl['sex']=='Female']
```

In [145]:
```
wml = lower[lower['race']=='White']
wml = wml[wml['sex']=='Male']
```

In [150]:
```
format(round(wfh['age'].mean(),2),round(wmh['age'].mean(),2),round(wfl['age'].mean(),2),round(wml['age'].mean(),2)))
```

```
average age of white females who have higher than 50K income = 42.28
average age of white males who have higher than 50K income = 44.74
average age of white females who have lower than 50K income = 36.07
average age of white males who have lower than 50K income = 37.29
```

T TEST

We are curious as to whether education level(number) differ between genders in a group that below 50K income. We take out samples from both male and female.Here is the result below:(alpha=0.05)

```
#We take samples that have below 30 for the sake of being independent rule
#from both group in order to apply T test.
a = np.random.choice(lf['education.num'],24)
b = np.random.choice(lm['education.num'],20)
print(a.mean(),b.mean(),a.std(),b.std())
```

```
9.875 9.95 2.8182810955143087 2.108909670896314
```

```
pd.DataFrame({'Mean':[10.5,9.45],
              'Std':[2.33,2.65],
             'Samples':[24,20]},
             index=['Female<50K','Male<50K'])
```

	Mean	Std	Samples
Female<50K	10.50	2.33	24
Male<50K	9.45	2.65	20

```
t=(10.50-9.45)/((2.33**2/24)+(2.65**2/20))**0.5
t


#Small t value indicates that education levels for male and female are similar.
```

```
1.3819029685511863
```

```
stats.ttest_ind(a,b)
```

```
Ttest_indResult(statistic=-0.09601321720529037, pvalue=0.9239667474636487)
```

We fail to reject Null hypothesis since p value is greater than threshold(significance level). As a result,there is no significance difference between being opposite genders of lower income as to education levels.

Z TEST

2) Comparing average age of two different groups

Average age of black people with above 50K and below with 50K in the dataset are respectively 43 and 36. standard deviations of both group are respectively 9 and 12. Test if mean of both group in real world is equal each other?(alpha=0.01)

```
#   H0: Ma-Mb = 0
#   Ha: Ma-Mb ≠ 0
```

```
Ma = 43
Mb = 36
σ1 = 9
σ2 = 12
z = (Ma-Mb-0)/((σ1**2/387)+(σ2**2/2737))**0.5
z
```

```
13.677859045246745
```

```
our z value=13.7 is way bigger than z critical value=2.575
so we reject Null hypothesis and accept alternative hypothesis
As a result mean age of black group with above 50K and below 50K are not going to be equal e
ach other.
```

DATA PREPROCESSING

I will first convert '?' into 'Not available'.

```
data[data=='?'] = np.nan
data = data.fillna('Not available')
data.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	ca
0	90	Not available	77053	HS-grad	9	Widowed	Not available	Not-in-family	White	Female	0	43
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	43
2	66	Not available	186061	Some-college	10	Widowed	Not available	Unmarried	Black	Female	0	43
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	39
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	39

```
cat_col = data[['workclass','education','marital.status',
                              'occupation','relationship','race','sex','native.country']]

num_cal = data[['age','fnlwgt','education.num','capital.gain','capital.loss','hours.per.week']]


target_col = data[['income']]
```

I used dummy variables on only categorical columns in order to label columns and convert strings into numeric variables.

```
cat_col = pd.get_dummies(cat_col,drop_first=True)
cat_col.head()
```

	workclass_Local-gov	workclass_Never-worked	workclass_Not available	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-not-inc	workclass_Sta gov
0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0
4	0	0	0	1	0	0	0

5 rows × 94 columns

I also standardized numerical columns.Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bias.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
num_cal = pd.DataFrame(data=sc.fit_transform(num_cal),columns=['age','fnlwgt','education.num',
                                                            'capital.gain','capital.loss','hour
s.per.week'])
num_cal.head()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
0	3.769612	-1.067997	-0.420060	-0.14592	10.593507	-0.035429
1	3.183112	-0.539169	-0.420060	-0.14592	10.593507	-1.817204
2	2.010110	-0.035220	-0.031360	-0.14592	10.593507	-0.035429
3	1.130359	-0.468215	-2.363558	-0.14592	9.461864	-0.035429
4	0.177296	0.709482	-0.031360	-0.14592	9.461864	-0.035429

```
target_col = target_col['income'].map({'<=50K': 0, '>50K': 1})
```

I merged then three data and got a dataset that is ready to be worked for further ML algorithms.

```
data = pd.concat([num_cal,target_col,cat_col],axis=1)
data.head()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	income	workclass_Local-gov	workclass_Nev worked
0	3.769612	-1.067997	-0.420060	-0.14592	10.593507	-0.035429	0	0	0
1	3.183112	-0.539169	-0.420060	-0.14592	10.593507	-1.817204	0	0	0
2	2.010110	-0.035220	-0.031360	-0.14592	10.593507	-0.035429	0	0	0
3	1.130359	-0.468215	-2.363558	-0.14592	9.461864	-0.035429	0	0	0
4	0.177296	0.709482	-0.031360	-0.14592	9.461864	-0.035429	0	0	0

5 rows × 101 columns

```
#Seting feature and target columns

x = data.drop(['income'],axis=1).values
y = data['income'].values
```

```
#Split data as train and test set

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 38)
```

Now,let`s see some Ml algorithms that will give us higher accuracy.

DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(criterion='entropy',random_state=38)
dtc.fit(X_train,y_train)

y_pred = dtc.predict(X_test)

cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[6515,  929],
       [ 868, 1457]])
```

```
round(accuracy_score(y_test,y_pred),4)
```

```
0.8161
```

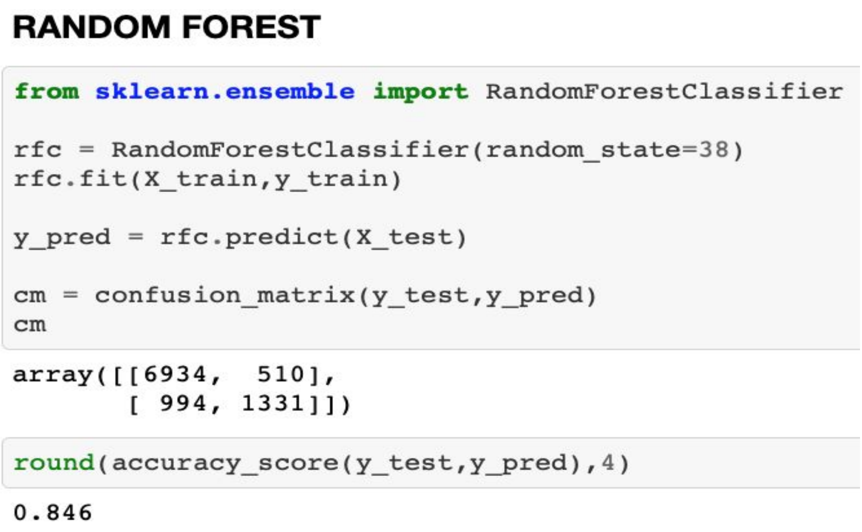
KNN

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5,metric='minkowski')
knn.fit(X_train,y_train)

y_pred = knn.predict(X_test)

cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[6729,  715],
       [ 903, 1422]])
```

```
round(accuracy_score(y_test,y_pred),4)
```

```
0.8344
```



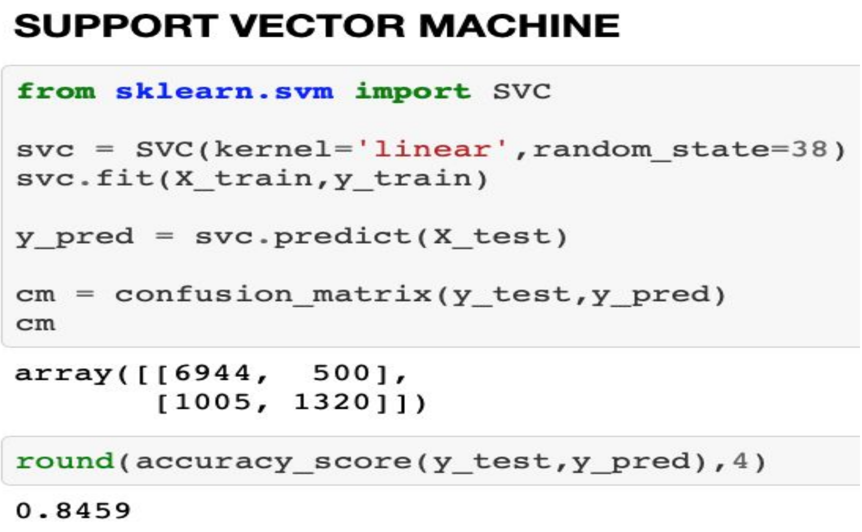
RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random_state=38)
rfc.fit(X_train,y_train)

y_pred = rfc.predict(X_test)

cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[6934,  510],
       [ 994, 1331]])
```

```
round(accuracy_score(y_test,y_pred),4)
```

```
0.846
```



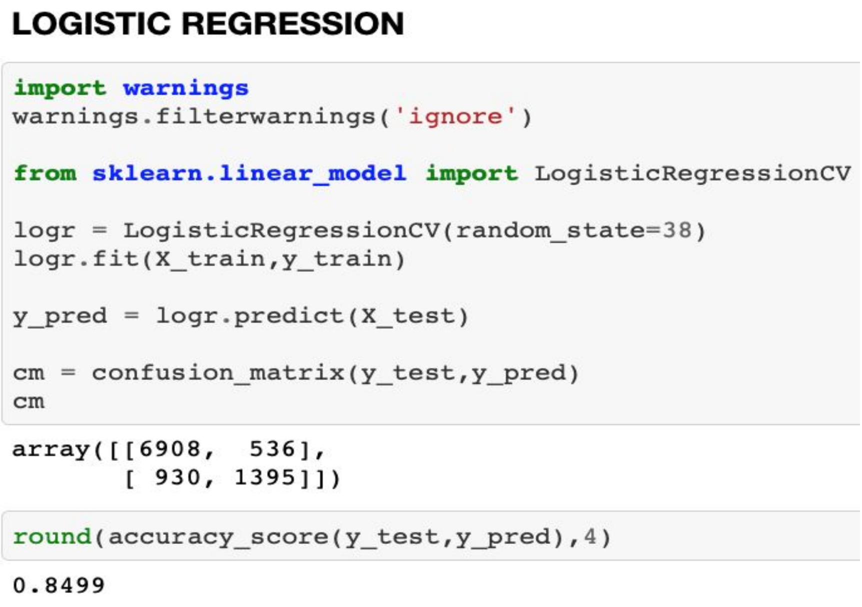
SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC

svc = SVC(kernel='linear',random_state=38)
svc.fit(X_train,y_train)

y_pred = svc.predict(X_test)

cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[6944,  500],
       [1005, 1320]])
```

```
round(accuracy_score(y_test,y_pred),4)
```

```
0.8459
```



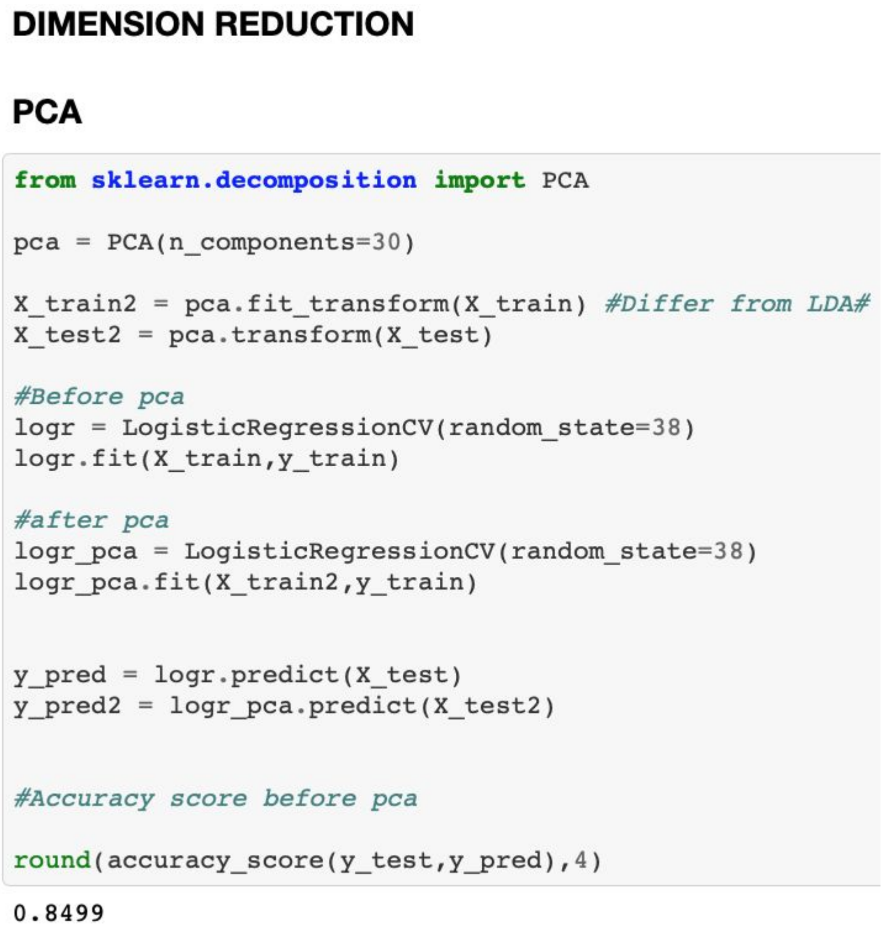
LOGISTIC REGRESSION

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

from sklearn.linear_model import LogisticRegressionCV

logr = LogisticRegressionCV(random_state=38)
logr.fit(X_train,y_train)

y_pred = logr.predict(X_test)

cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[6908,  536],
       [ 930, 1395]])
```

```
round(accuracy_score(y_test,y_pred),4)
```

```
0.8499
```

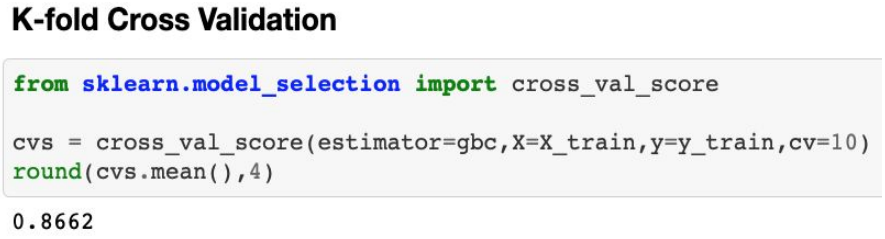


DIMENSION REDUCTION

PCA

```
from sklearn.decomposition import PCA

pca = PCA(n_components=30)

X_train2 = pca.fit_transform(X_train) #Differ from LDA#
X_test2 = pca.transform(X_test)

#Before pca
logr = LogisticRegressionCV(random_state=38)
logr.fit(X_train,y_train)

#after pca
logr_pca = LogisticRegressionCV(random_state=38)
logr_pca.fit(X_train2,y_train)


y_pred = logr.predict(X_test)
y_pred2 = logr_pca.predict(X_test2)


#Accuracy score before pca

round(accuracy_score(y_test,y_pred),4)
```

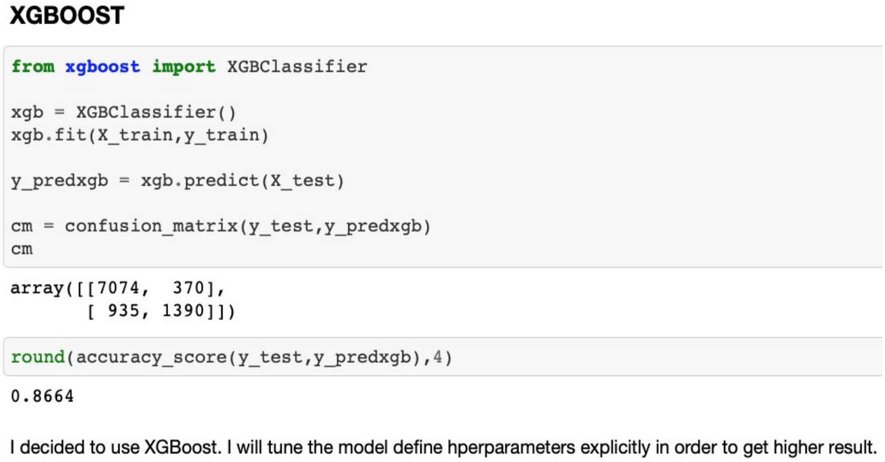
```
0.8499
```



K-fold Cross Validation

```
from sklearn.model_selection import cross_val_score

cvs = cross_val_score(estimator=gbc,X=X_train,y=y_train,cv=10)
round(cvs.mean(),4)
```

```
0.8662
```



XGBOOST

```
from xgboost import XGBClassifier

xgb = XGBClassifier()
xgb.fit(X_train,y_train)

y_predxgb = xgb.predict(X_test)

cm = confusion_matrix(y_test,y_predxgb)
cm
```

```
array([[7074,  370],
       [ 935, 1390]])
```

```
round(accuracy_score(y_test,y_predxgb),4)
```

```
0.8664
```

I decided to use XGBoost. I will tune the model define hperparameters explicitly in order to get higher result.

MODEL TUNING

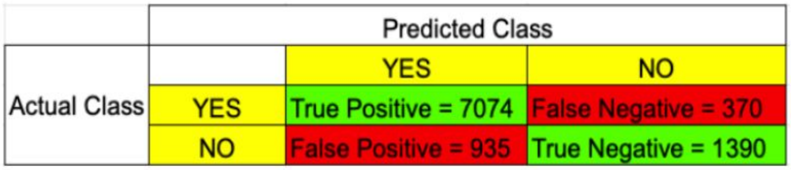
```
#Import necessary modules
from sklearn.model_selection import RandomizedSearchCV

# Setup the hyperparameter grid

p = {'n_estimators':[50,100,150],
          'min_child_weight': [1, 5, 10],
          'learning_rate':[0.05,0.1,0.15,0.2],
          'gamma': [0.5, 1, 1.5, 2, 5],
          'max_depth': [3, 4, 5]
        }

# Instantiate a XGBoosting classifier
xgb = XGBClassifier(random_state=38)

# Instantiate the RandomizedSearchCV object
gs = RandomizedSearchCV(estimator = xgb,param_distributions= p,scoring='accuracy',cv=5,random_state=38)

# Fit it to the data
gs.fit(X_train,y_train)

# Print the tuned parameters and score
print('Tuned XGboosting Parameters: {}'.format(gs.best_params_))
print("Best score is {}".format(gs.best_score_))
```

```
Tuned XGboosting Parameters: {'n_estimators': 100, 'min_child_weight': 1, 'max_depth': 4, 'learning_rate': 0.15, 'gamma': 1}
Best score is 0.8715777465777466
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predxgb))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.92      7444
           1       0.79      0.60      0.68      2325

    accuracy                           0.87      9769
   macro avg       0.84      0.77      0.80      9769
weighted avg       0.86      0.87      0.86      9769
```

ANALYSIS OF TABLE ABOVE



		Predicted Class	
Actual Class		YES	NO
	YES	True Positive = 7074	False Negative = 370
	NO	False Positive = 935	True Negative = 1390

The first metric we look as a datascientist is accuracy = 87%. It looks awesome but it is insufficent if we check merely accuracy.There are other concepts to analyze our dataset being health enough as well. Let`s dive into it.

Precision: Precision is a good measure to determine, when the costs of False Positive is high. Since our false positive value,precision can be good interpreter in here.It is the ratio of correctly predicted positive observations to the total predicted positive observations . The question that this metric answer is of people that labeled as below 50K, how many of them actually below 50K? or of labeled as above 50K, how many of them actually above 50K?

In here almost 90% of correct prediction on <=50K and almost 80% correct prediction on >50K.These high precisions make our model more reliable.

Recall: Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.Especially predicting 95% correct on people below 50K is incredible estimation while other group needs to be worked much on it.

F1-Score: It is required when we seek a balance between Precision and Recall.It is a harmonic mean of both precision and recall.F1-score for the both groups especially people who have income below 50K is very high.

Overall with %87 accuracy, we built very good model by ML algorithms.