Data Preprocessing

1. Removing the space in the names of the columns:

```
df.columns = df.columns.str.replace(" ", "")
df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'income'],
      dtype='object')
```

2. Checking for null and na values:

```
#print('Null values:\n', df.isnull().sum())
#print('\nNa values:\n', df.isna().sum())
for i in df.columns:
    print(i, '\t\t', df[i].dtype,'\t\t\t',df[i].isnull().any())
```

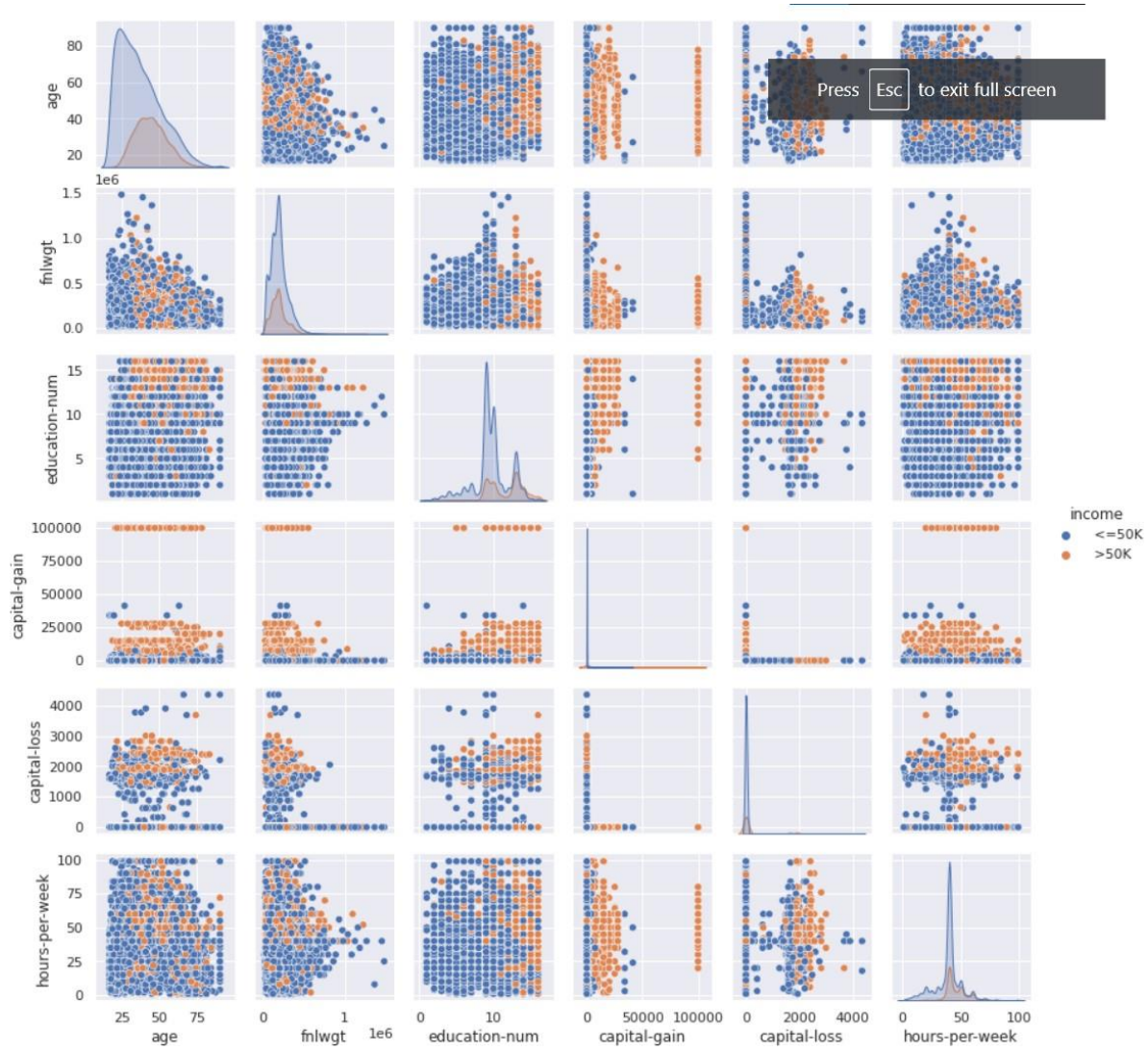
```
age             int64                   False
workclass               object                          False
fnlwgt          int64                   False
education               object                          False
education-num           int64                   False
marital-status          object                          False
occupation              object                          False
relationship            object                          False
race            object                          False
sex             object                          False
capital-gain            int64                   False
capital-loss            int64                   False
hours-per-week          int64                   False
native-country          object                          False
income          object                          False
```

3. Finding correlation of the dataset:

```
#print('Null values:\n', df.isnull().sum())
#print('\nNa values:\n', df.isna().sum())
for i in df.columns:
    print(i, '\t\t', df[i].dtype,'\t\t\t',df[i].isnull().any())
```

```
age             int64                   False
workclass               object                          False
fnlwgt          int64                   False
education               object                          False
education-num           int64                   False
marital-status          object                          False
occupation              object                          False
relationship            object                          False
race            object                          False
sex             object                          False
capital-gain            int64                   False
capital-loss            int64                   False
hours-per-week          int64                   False
native-country          object                          False
income          object                          False
```

4. Finding the correlation of the dataset:



5. Normalizing the dataset:

```
#Normalization

numeric = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
c_num = df.select_dtypes(include=numeric).columns.tolist()
c_num
```

```
['age',
 'fnlwgt',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week']
```

6. Replacing the ? with unknown in the dataset:

```
df=df.replace([' ?'], 'unknown')
display(HTML(df.head(5).to_html()))
```

Data after preprocessing:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
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

Classification models:

1. KNN:

Converting data into numeric form:

```
#convert to numeric

df_dummies = pd.get_dummies(df[['workclass','education','marital-status','occupation','relationship','race','sex','native-country']])
df_dummies
df_dummies.dtypes

dfknn = df.join(df_dummies)
dfknn = dfknn.drop(['workclass','education','marital-status','occupation','relationship','race','sex','native-country'], axis=1)
dfknn = dfknn.drop(['workclass_ Federal-gov','education_ 10th','marital-status_ Divorced','occupation_ Adm-clerical','relationship_ Husband','ra

display(HTML(dfknn.head(5).to_html()))
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	income	workclass_ Local-gov	workclass_ Never-worked	workclass_ Private	workclass_ Self-emp-inc	workclass_ Self-emp-not-inc	workclass_ State-gov	workclass_ Without-pay
0	39	77516	13	2174	0	40	<=50K	0	0	0	0	0	1	0
1	50	83311	13	0	0	13	<=50K	0	0	0	0	1	0	0
2	38	215646	9	0	0	40	<=50K	0	0	1	0	0	0	0
3	53	234721	7	0	0	40	<=50K	0	0	1	0	0	0	0
4	28	338409	13	0	0	40	<=50K	0	0	1	0	0	0	0

Normalizing the data:

```
#select numeric cols
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
cols_numeric = dfknn.select_dtypes(include=numerics).columns.tolist()
print('Selected numerical columns:\n',cols_numeric)

#normalization
for col in cols_numeric:
  dfknn[col]=(dfknn[col]-dfknn[col].min())/(dfknn[col].max()-dfknn[col].min())

display(HTML(dfknn.head(10).to_html()))
```

```
Selected numerical columns:
 ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	income	workclass_ Local-gov	workclass_ Never-worked	workclass_ Private	workclass_ Self-emp-inc	workclass_ Self-emp-not-inc	workclass_ State-gov	wo
0	0.301370	0.044302	0.800000	0.021740	0.0	0.397959	<=50K	0	0	0	0	0	1	
1	0.452055	0.048238	0.800000	0.000000	0.0	0.122449	<=50K	0	0	0	0	1	0	
2	0.287671	0.138113	0.533333	0.000000	0.0	0.397959	<=50K	0	0	1	0	0	0	
3	0.493151	0.151068	0.400000	0.000000	0.0	0.397959	<=50K	0	0	1	0	0	0	
4	0.150685	0.221488	0.800000	0.000000	0.0	0.397959	<=50K	0	0	1	0	0	0	
5	0.273973	0.184932	0.866667	0.000000	0.0	0.397959	<=50K	0	0	1	0	0	0	
6	0.438356	0.100448	0.266667	0.000000	0.0	0.153061	<=50K	0	0	1	0	0	0	
7	0.479452	0.134036	0.533333	0.000000	0.0	0.448980	>50K	0	0	0	0	1	0	
8	0.191781	0.022749	0.866667	0.140841	0.0	0.500000	>50K	0	0	1	0	0	0	
9	0.342466	0.099947	0.800000	0.051781	0.0	0.397959	>50K	0	0	1	0	0	0	

Encoding the y variable:

```
#Encoding the y variable
y = dfknn['income']
le = preprocessing.LabelEncoder()
le.fit(y)
y_encoded = le.transform(y)
print('encode y :\n', y_encoded)
```

Accuracy: We get the highest accuracy for k = 3 is 98%

2. Naive Baye:

Converting the numerical columns into categorical:

```
#converting numerical to categorical data
df_nb = df
df_nb['age'] = pd.cut(df_nb['age'],3)
df_nb['fnlwgt'] = pd.cut(df_nb['fnlwgt'],3)
df_nb['education-num'] = pd.cut(df_nb['education-num'],3)
df_nb['capital-gain'] = pd.cut(df_nb['capital-gain'],2)
df_nb['capital-loss'] = pd.cut(df_nb['capital-loss'],2)
df_nb['hours-per-week'] = pd.cut(df_nb['hours-per-week'],3)
```

Performing one hot encoding on the data

```
# transform categorical data to numerical data, i.e., one-hot encoding
print(df_nb.dtypes)
df_nb=pd.get_dummies(df_nb.drop('income',axis=1))
df_nb['income']=y_encoded
display(HTML(df_nb.head(5).to_html()))
```

Accuracy of the model by n fold is: 58.4%

3. Decision Tree and Random Forest:

```
# by hold-out evaluation
x_train, x_test, y_train, y_test = train_test_split(df_nb, y_encoded, test_size=0.2)
clf=DecisionTreeClassifier() # note: there are many parameters in API
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
acc=accuracy_score(y_pred, y_test)
print('Tree Accuracy by hold-out evaluation: ',acc)

# by N-fold cross validation
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Tree Accuracy by N-fold Cross Validation:",acc)

# Example of randomForest = bagging method of decision trees
tree = DecisionTreeClassifier()
# Note: you can use tree only or the random forest for the purpose of evaluations
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)
acc=cross_val_score(bag, x, y, cv=5, scoring='accuracy').mean()
print("RandomForest Accuracy by N-fold Cross Validation:",acc)
```

Accuracy of the decision tree is 81.16 and for random forest is: 85.53

4. SVM:

```
# by hold-out evaluation
x_train, x_test, y_train, y_test = train_test_split(dfknn, y_encoded, test_size=0.2)
clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
acc=accuracy_score(y_pred, y_test)
print('Accuracy by hold-out evaluation: ',acc)

x=dfknn.drop('income',axis=1)
y=dfknn['income']
# by N-fold cross validation
clf=SVC(kernel='linear', C=1E10)
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Accuracy by N-fold Cross Validation:",acc)
clf=SVC(kernel='rbf', C=1E10)
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Accuracy by N-fold Cross Validation:",acc)
```

Accuracy of the model is: 84.99%

5. Logistic Regression:

```
# by hold-out evaluation
x_train, x_test, y_train, y_test = train_test_split(dfknn, y_encoded, test_size=0.2)
clf=LogisticRegression()
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
acc=accuracy_score(y_pred, y_test)
print('Accuracy by hold-out evaluation: ',acc)

x=dfknn.drop('income',axis=1)
y=dfknn['income']
# by N-fold cross validation
clf=LogisticRegression()
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Accuracy by N-fold Cross Validation:",acc)
```

Accuracy of the model is 85%

6. Neural Network:

```
from sklearn.neural_network import MLPClassifier

# by hold-out evaluation
x_train, x_test, y_train, y_test = train_test_split(dfknn, y_encoded, test_size=0.2)
clf=MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=1)
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
acc=accuracy_score(y_pred, y_test)
print('Accuracy by hold-out evaluation: ',acc)

x=dfknn.drop('income',axis=1)
y=dfknn['income']
# by N-fold cross validation
clf=MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(10,8), random_state=1)
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Accuracy by N-fold Cross Validation:",acc)
```

Accuracy of the model is: 85.3%

```
# check degree of imbalance in labels
cf=df[' income'].value_counts()
crf=df[' income'].value_counts()/df.shape[0]
print("\nClass frequency:\n", cf, "\n\nClass relative frequency:\n", crf)


# get features and labels
x=df.drop(' income',axis=1)
y=df[' income']


from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

ros = RandomOverSampler(random_state=10)
ros.fit(x, y)
print('Original dataset shape {}'.format(Counter(y)))
x_resampled, y_resampled = ros.fit_resample(x, y)
print('After oversampling dataset shape {}'.format(Counter(y_resampled)))

print('Original dataset shape {}'.format(Counter(y)))
ros = RandomUnderSampler(random_state=30)
x_resampled, y_resampled = ros.fit_resample(x, y)
print('After undersampling dataset shape {}'.format(Counter(y_resampled)))

# get features and labels, SMOTE can only be applied on numerical features
x=dfknn.drop('income',axis=1)
y=dfknn['income']

ros = SMOTE(random_state=10, k_neighbors=2)
print('Original dataset shape {}'.format(Counter(y)))
x_resampled, y_resampled = ros.fit_resample(x, y)
print('After oversampling by SMOTE dataset shape {}'.format(Counter(y_resampled)))
```

Comparing the Accuracies of each model:

1. KNN: 98%
2. NAIVE BAYES: 58.4%
3. Decision Tree: 81.16% and Random Forest: 85.53%
4. SVM: 85%
5. LOGISTIC REGRESSION: 85.03%
6. NEURAL NETWORKS: 85.40%

Best parameter performance:

The best results are got with with the following changes in the parameters KNN: weight is uniform Decision Tree: criterion is the gini index Random forest tree number is 50

The best performance is of knn

Comparing the Performances of each model:

1. KNN: 98%
2. Knn parameter change : 98.3
3. NAIVE BAYES: 58.4%
4. Decision Tree: 81.16% and Random Forest: 85.53%
5. Decision tree and random forest after parameters: 81.3 and 82.41%
6. SVM: 85%
7. LOGISTIC REGRESSION: 85.03%
8. NEURAL NETWORKS: 85.40%

Performing the feature selection using the filter method and feature reduction using the PCA algorithm:

1. Finding the correlation of the variables and selecting the variables with the variables whose values have the correlation more than 0.2

```
#Correlation with output variable
cor = dfknn.corr()
cor_target = abs(cor["income"])
#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.2]
print('\nSelected features by Filter model:\n',relevant_features)


dfd = dfknn[['age','education-num','capital-gain','hours-per-week','marital-status_ Married-civ-spouse','marital-status_ Never-married','occupat
#print(dfknn_1)
```

2. Creating the data frame with only those parameter
3. Training the model:
    1. Decision tree and random forest

```
# by hold-out evaluation
x_train, x_test, y_train, y_test = train_test_split(dfd, y_encoded, test_size=0.2)
clf=DecisionTreeClassifier() # note: there are many parameters in API
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
acc=accuracy_score(y_pred, y_test)
print('Tree Accuracy by hold-out evaluation: ',acc)

# by N-fold cross validation
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Tree Accuracy by N-fold Cross Validation:",acc)

# Example of randomForest = bagging method of decision trees
tree = DecisionTreeClassifier()
# Note: you can use tree only or the random forest for the purpose of evaluations
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)
acc=cross_val_score(bag, x, y, cv=5, scoring='accuracy').mean()
print("RandomForest Accuracy by N-fold Cross Validation:",acc)
```

Accuracy: 81.42 and 85.55

2. Svm

```
# by hold-out evaluation
# by hold-out evaluation
x_train, x_test, y_train, y_test = train_test_split(dfsvm, y_encoded, test_size=0.2)
clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
acc=accuracy_score(y_pred, y_test)
print('Accuracy by hold-out evaluation: ',acc)

x=dfknn.drop('income',axis=1)
y=dfknn['income']
# by N-fold cross validation
clf=SVC(kernel='linear', C=1E10)
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Accuracy by N-fold Cross Validation:",acc)
clf=SVC(kernel='rbf', C=1E10)
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Accuracy by N-fold Cross Validation:",acc)
```

```
print("Accuracy by N-fold Cross Validation:",acc)
```

```
Selected features by Filter model:
 age                                0.234037
education-num                       0.335154
capital-gain                        0.223329
hours-per-week                      0.229689
income                              1.000000
marital-status_ Married-civ-spouse  0.444696
marital-status_ Never-married       0.318440
occupation_ Exec-managerial         0.214861
relationship_ Own-child             0.228532
sex_ Male                           0.215980
Name: income, dtype: float64
Accuracy by hold-out evaluation:  1.0
Accuracy by N-fold Cross Validation: 0.8499126487150439
```

Accuracy is 85%

Performing PCA:

1. SVM

```
from sklearn.decomposition import PCA
pca = PCA(n_components=5)
fit = pca.fit(x)
#fit.components_

PCAs = fit.transform(x)
PCA_select = PCAs[:,:3]
df_PCA = pd.DataFrame(data=PCA_select, columns=['PCA1','PCA2','PCA3'])
df_PCA['income'] = y
display('df_PCA:',HTML(df_PCA.head(10).to_html()))
#Building the model

y_encoded = df_PCA['income']
x_train, x_test, y_train, y_test = train_test_split(df_PCA, y_encoded, test_size = 0.2 )

clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
acc=accuracy_score(y_pred, y_test)
print('Accuracy by hold-out evaluation: ',acc)

x=dfknn.drop('income',axis=1)
y=dfknn['income']
# by N-fold cross validation
clf=SVC(kernel='linear', C=1E10)
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Accuracy by N-fold Cross Validation:",acc)
clf=SVC(kernel='rbf', C=1E10)
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Accuracy by N-fold Cross Validation:",acc)
```

   Reducing the dimension sof the features and then building the model :

2. Decision Tree and Random forest:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=5)
fit = pca.fit(x)
#fit.components_

PCAs = fit.transform(x)
PCA_select = PCAs[:,:3]
df_PCA = pd.DataFrame(data=PCA_select, columns=['PCA1','PCA2','PCA3'])
df_PCA['income'] = y
display('df_PCA:',HTML(df_PCA.head(10).to_html()))
#Building the model

y_encoded = df_PCA['income']
x_train, x_test, y_train, y_test = train_test_split(df_PCA, y_encoded, test_size = 0.2 )

clf=DecisionTreeClassifier() # note: there are many parameters in API
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
acc=accuracy_score(y_pred, y_test)
print('Tree Accuracy by hold-out evaluation: ',acc)

# by N-fold cross validation
acc=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
print("Tree Accuracy by N-fold Cross Validation:",acc)

# Example of randomForest = bagging method of decision trees
tree = DecisionTreeClassifier()
# Note: you can use tree only or the random forest for the purpose of evaluations
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)
acc=cross_val_score(bag, x, y, cv=5, scoring='accuracy').mean()
print("RandomForest Accuracy by N-fold Cross Validation:",acc)
```

Accuracy: 81.29 and 85.55%