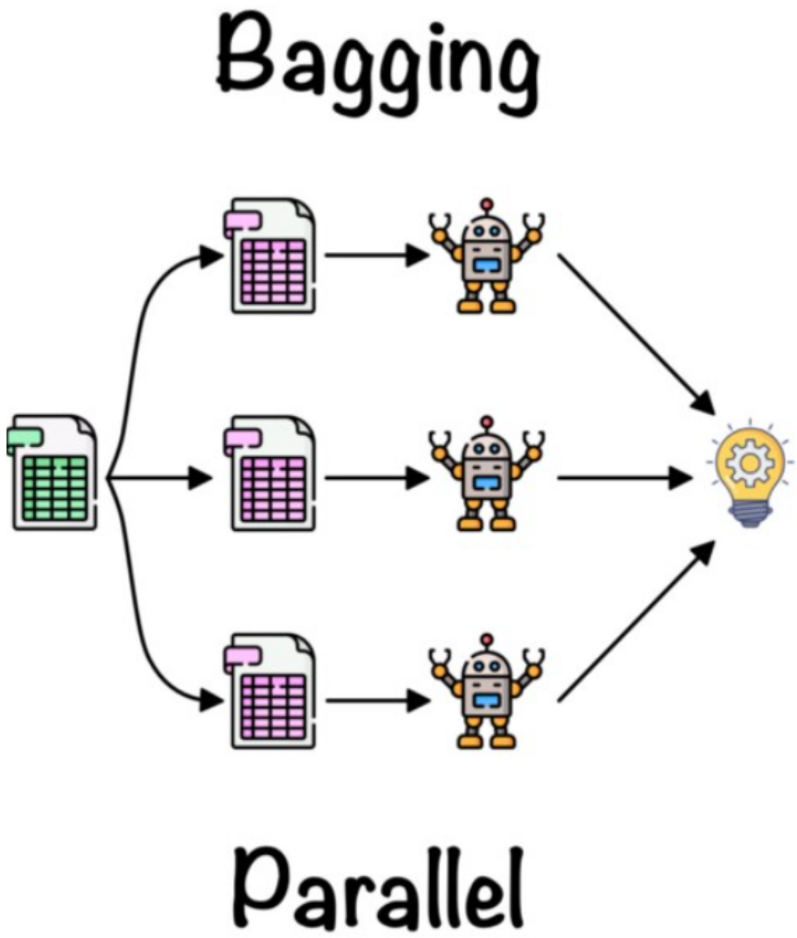
Bagging

Bagging, also known as Bootstrap aggregating, is an ensemble learning technique that helps to improve the performance and accuracy of machine learning algorithms. It is used to deal with bias-variance trade-offs and reduces the variance of a prediction model. Bagging avoids overfitting of data and is used for both regression and classification models, specifically for decision tree algorithms.



Steps to Perform Bagging

- Consider there are n observations and m features in the training set. You need to select a random sample from the training dataset without replacement
- A subset of m features is chosen randomly to create a model using sample observations
- The feature offering the best split out of the lot is used to split the nodes
- The tree is grown, so you have the best root nodes
- The above steps are repeated n times. It aggregates the output of individual decision trees to give the best prediction

Advantages of Bagging in Machine Learning

- Bagging minimizes the overfitting of data
- It improves the model's accuracy
- It deals with higher dimensional data efficiently

Steps involved

Data Injection

- Data Profiling
- Basic Operations
- Data Cleaning
- Analysis of features and Statistical Analysis

EDA

- Univariate Analysis
- Bivariate Analysis
- Multivariate Analysis
- Pre-processing
- Handling DUplicate values
- Handling null values

Mapping

- Feature Encoding
- Spliting of categorical and numerical variable
- Train-Test split

Model Creation

- Decision Tree Classifier
- HyperParameter Tuning : Decision Tree Classifier
- Bagging Classifier
- Hyperparameter tuning : Bagging Classifier
- Random Forest Classifier
- Hyperparameter tuning : Random Forest Classifier
- Extra Trees Classifier
- HyperParameter Tuning : Extra Tree Classifier
- Voting Classifier
- hard_voting
- soft_voting

Evaluation

- Accuracy Score
- Roc-auc score
- Precision
- Recall
- F1_Score

Attribute Information:

Listing of attributes:

50K, <=50K.

- age: continuous
- workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, 3. State-gov, Without-pay, Never-worked.
- fnlwgt: continuous.
- education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, 6. 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.

In [30]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')

# Visualization
import matplotlib.pyplot as plt
%matplotlib inline
import missingno
import seaborn as sns
from pandas.plotting import scatter_matrix
from mpl_toolkits.mplot3d import Axes3D
import math

from sklearn.preprocessing import OneHotEncoder, LabelEncoder, label_binarize
from sklearn.model_selection import train_test_split
#Model
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
# Evaluation Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
# Plots
from sklearn import tree
```

Data Ingestion

In [2]:

```
df = pd.read_excel('Income_data.xlsx')
len(df)
```

Out[2]:

```
48842
```

Numerical Columns

In [3]:

```
cat_col=[fea for fea in df.columns if df[fea].dtype == 'O']
cat_col
```

Out[3]:

```
['worktype',
 'education',
 'maritial_status',
 'occupation',
 'relationship',
 'race',
 'gender',
 'native_country',
 'salary']
```

Categorical Columns

In [4]:

```
num_col=[fea for fea in df.columns if df[fea].dtype != 'O']
num_col
```

Out[4]:

```
['age',
 'fnlwgt',
 'education_num',
 'capital_gain',
 'capital_loss',
 'hours_per_week']
```

Univariate Analysis

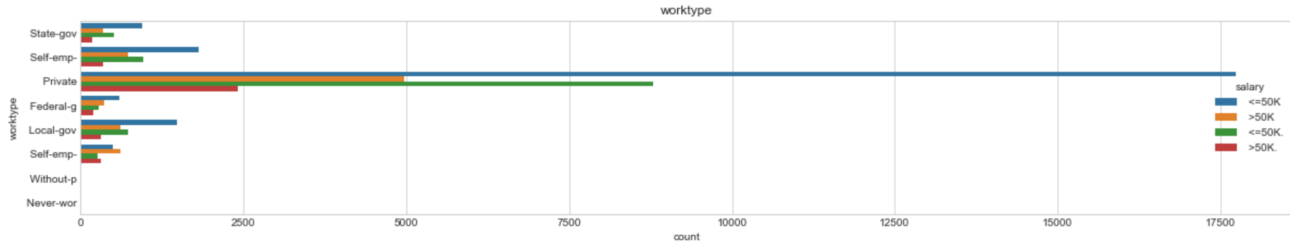
In [5]:

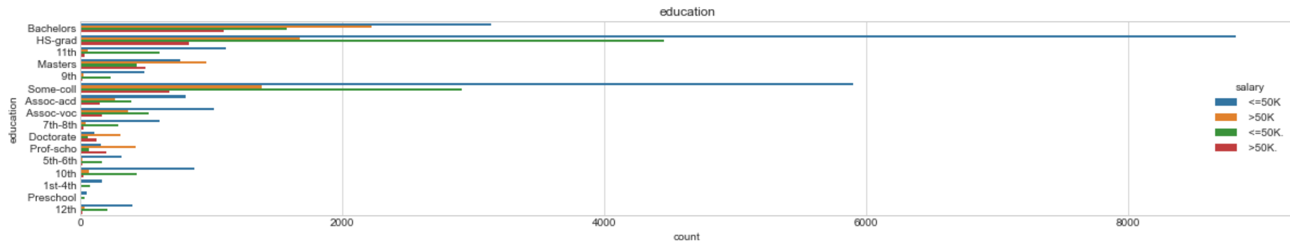
```
# Let’s plot the distribution of each feature
def plot_distribution(df, cols=5, width=20, height=15, hspace=0.2, wspace=0.5):
    plt.style.use('seaborn-whitegrid')
    fig = plt.figure(figsize=(width,height))
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=wspace, hspace=hspace)
    rows = math.ceil(float(df.shape[1]) / cols)
    for i, column in enumerate(df.columns):
        ax = fig.add_subplot(rows, cols, i + 1)
        ax.set_title(column)
        if df.dtypes[column] == np.object:
            g = sns.countplot(y=column, data=df)
            substrings = [s.get_text()[:18] for s in g.get_yticklabels()]
            g.set(yticklabels=substrings)
            plt.xticks(rotation=25)
        else:
            g = sns.distplot(df[column])
            plt.xticks(rotation=25)

plot_distribution(df, cols=3, width=20, height=20, hspace=0.45, wspace=0.5)
```

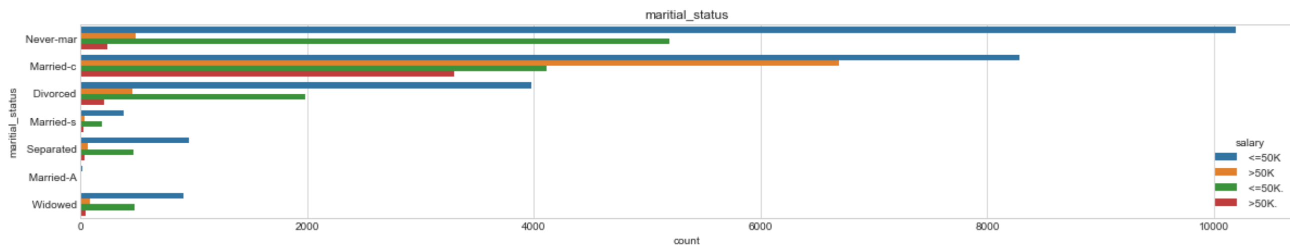
Bivariate Analysis

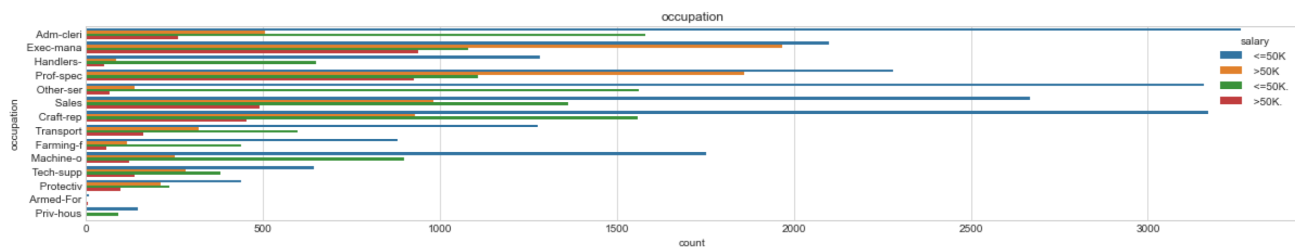
In [6]:

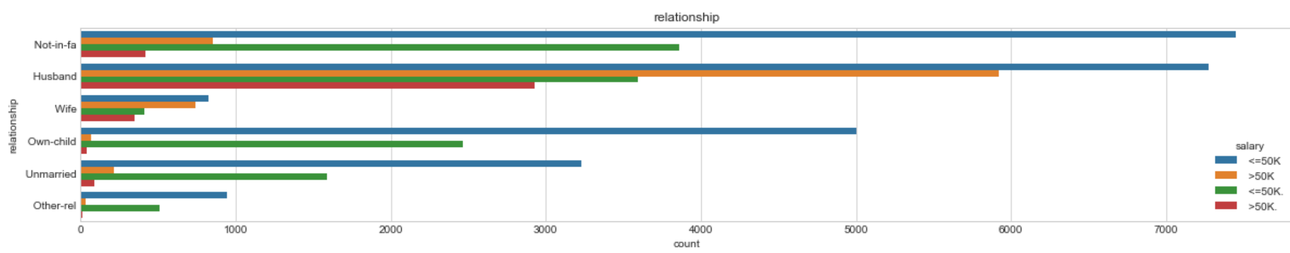
```
# Plot a count of the categories from each categorical feature split by our prediction class: salary - predclass.
def plot_bivariate_bar(dataset, hue, cols=5, width=20, height=15, hspace=0.2, wspace=0.5):
    dataset = dataset.select_dtypes(include=[np.object])
    plt.style.use('seaborn-whitegrid')
    fig = plt.figure(figsize=(width,height))
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=wspace, hspace=hspace)
    rows = math.ceil(float(dataset.shape[1]) / cols)
    for i, column in enumerate(dataset.columns):
        ax = fig.add_subplot(rows, cols, i + 1)
        ax.set_title(column)
        if dataset.dtypes[column] == np.object:
            g = sns.countplot(y=column, hue=hue, data=dataset)
            substrings = [s.get_text()[:10] for s in g.get_yticklabels()]
            g.set(yticklabels=substrings)

plot_bivariate_bar(df, hue='salary', cols=1, width=20, height=40, hspace=0.4, wspace=0.5)
```

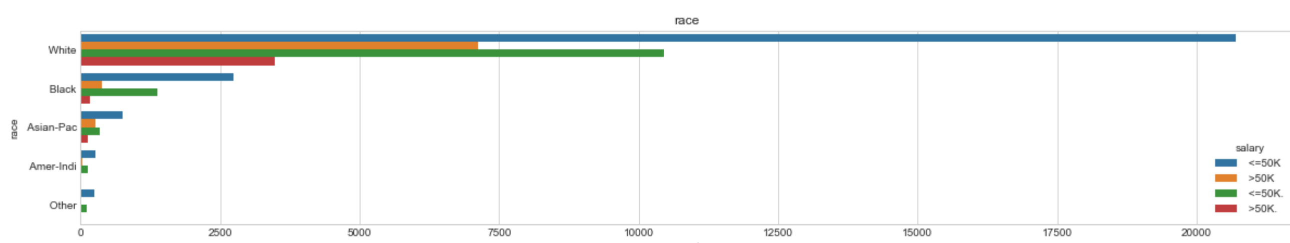

worktype
State-gov
Self-emp-
Private
Federal-g
Local-gov
Self-emp-
Without-p
Never-wor
worktype
0 2500 5000 7500 10000 12500 15000 17500
count
salary
<=50K
>50K
<=50K.
>50K.

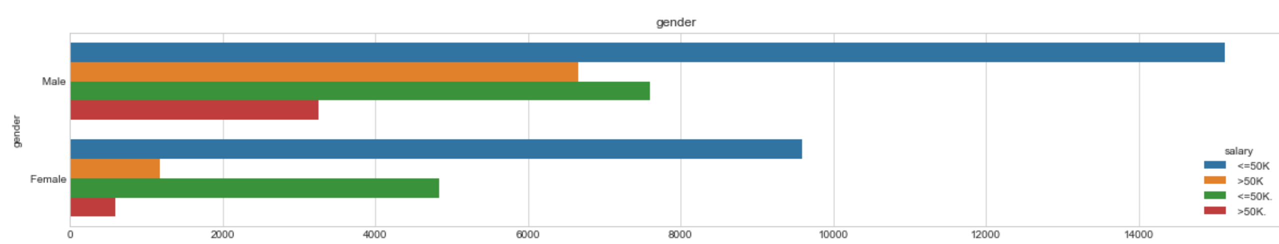

education
Bachelors
HS-grad
11th
Masters
9th
Some-coll
Assoc-acd
Assoc-voc
7th-8th
Doctorate
Prof-scho
5th-6th
10th
1st-4th
Preschool
12th
education
0 2000 4000 6000 8000
count
salary
<=50K
>50K
<=50K.
>50K.

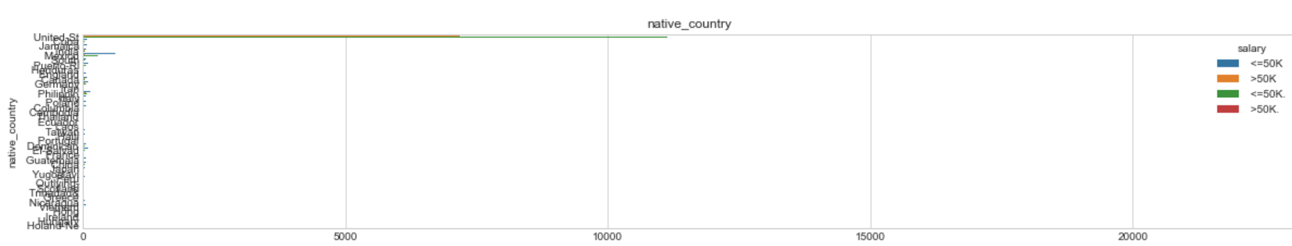

maritial_status
Never-mar
Married-c
Divorced
Married-s
Separated
Married-A
Widowed
maritial_status
0 2000 4000 6000 8000 10000
count
salary
<=50K
>50K
<=50K.
>50K.

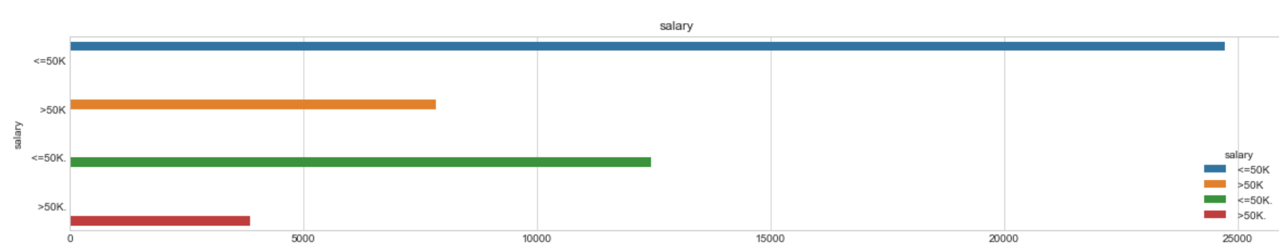

occupation
Adm-cleri
Exec-mana
Handlers-
Prof-spec
Other-ser
Sales
Craft-rep
Transport
Farming-f
Machine-o
Tech-supp
Protectiv
Armed-For
Priv-hous
occupation
0 500 1000 1500 2000 2500 3000
count
salary
<=50K
>50K
<=50K.
>50K.


relationship
Not-in-fa
Husband
Wife
Own-child
Unmarried
Other-rel
relationship
0 1000 2000 3000 4000 5000 6000 7000
count
salary
<=50K
>50K
<=50K.
>50K.


race
White
Black
Asian-Pac
Amer-Indi
Other
race
0 2500 5000 7500 10000 12500 15000 17500 20000
count
salary
<=50K
>50K
<=50K.
>50K.


gender
Male
Female
gender
0 2000 4000 6000 8000 10000 12000 14000
count
salary
<=50K
>50K
<=50K.
>50K.


native_country
United-St
Holand-Ne
native_country
0 5000 10000 15000 20000
count
salary
<=50K
>50K
<=50K.
>50K.


salary
<=50K
>50K
<=50K.
>50K.
salary
0 5000 10000 15000 20000 25000
count
salary
<=50K
>50K
<=50K.
>50K.

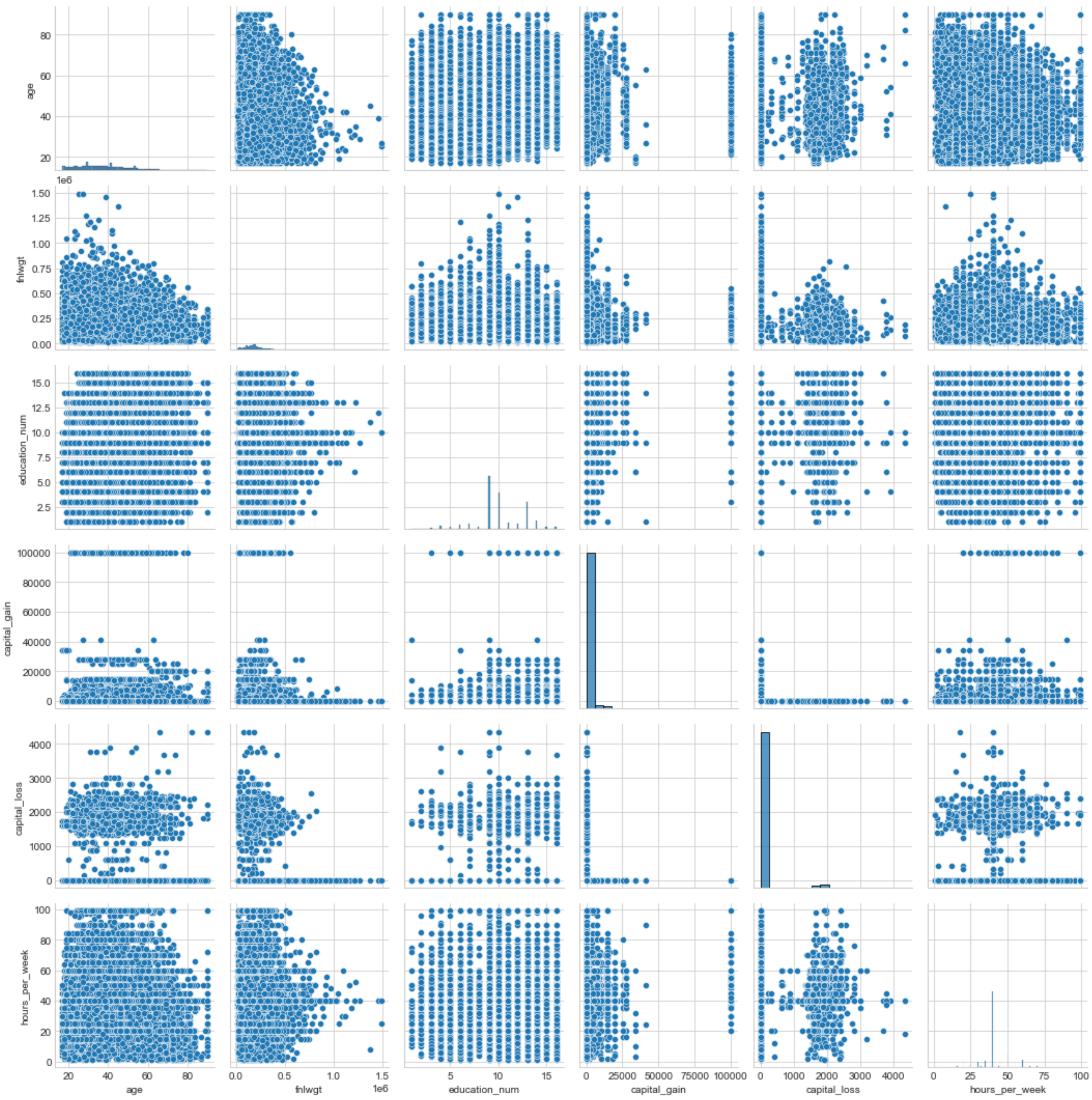
Multivariate Analysis

In [7]:

```
sns.pairplot(df)
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x27987240e80>
```



Handling Duplicates

In [8]:

```
df.duplicated().sum()
```

Out[8]:

```
29
```

In [9]:

```
df = df.drop_duplicates()
```

In [10]:

```
df.duplicated().sum()
```

Out[10]:

```
0
```

Handling Null Values

In [11]:

```
df.isnull().sum()
```

Out[11]:

```
age                 0
worktype         2799
fnlwgt              0
education           0
education_num       0
maritial_status     0
occupation       2809
relationship        0
race                0
gender              0
capital_gain        0
capital_loss        0
hours_per_week      0
native_country    856
salary              0
dtype: int64
```

In [12]:

```
null_values = df.isnull().sum().sum()
if null_values == 0:
    print('No null values exist')
else:
    from sklearn.impute import SimpleImputer
    #imputing with most frequent values
    imputer = SimpleImputer(strategy='most_frequent', missing_values=np.nan)
    imputer = imputer.fit(df)
    df = pd.DataFrame(imputer.transform(df.loc[:,:]), columns = df.columns)
```

In [13]:

```
df.isnull().sum()
```

Out[13]:

```
age               0
worktype          0
fnlwgt            0
education         0
education_num     0
maritial_status   0
occupation        0
relationship      0
race              0
gender            0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
salary            0
dtype: int64
```

Feature Encoding

Remember that Machine Learning algorithms perform Linear Algebra on Matrices, which means all features need have numeric values. The process of converting Categorical Features into values is called Encoding. Let's perform both One-Hot encoding.

Additional Resources: [http://pbpython.com/categorical-encoding.html (http://pbpython.com/categorical-encoding.html)](http://pbpython.com/categorical-encoding.html)

In [14]:

```

# One Hot Encodes all labels before Machine Learning
one_hot_cols = df.columns.tolist()
#one_hot_cols.remove('predclass')
dataset_bin_enc = pd.get_dummies(df, columns=one_hot_cols)

dataset_bin_enc.head()
```

Out[14]:

ıge_21	age_22	age_23	age_24	age_25	age_26	...	native_country_ Taiwan	native_country_ Thailand	native_country_ Trinadad&Tobago	native_country_ United-States	native_country_ Vietnam	native_countr Yugoslav
0	0	0	0	0	0	...	0	0	0	1	0	
0	0	0	0	0	0	...	0	0	0	1	0	
0	0	0	0	0	0	...	0	0	0	1	0	
0	0	0	0	0	0	...	0	0	0	1	0	
0	0	0	0	0	0	...	0	0	0	0	0	

In [19]:

```
# Label Encode all labels
df1 = df.apply(LabelEncoder().fit_transform)

df1.head()
```

Out[19]:

	age	worktype	fnlwgt	education	education_num	maritial_status	occupation	relationship	race	gender	capital_gain	capital_loss	hours_per_week
0	22	6	3461	9	12	4	0	1	4	1	27	0	39
1	33	5	3788	9	12	2	3	0	4	1	0	0	12
2	21	3	18342	11	8	0	5	1	4	1	0	0	39
3	36	3	19995	1	6	2	5	0	2	1	0	0	39
4	11	3	25405	9	12	2	9	5	2	0	0	0	39

In [20]:

```
X = df1.drop('salary', axis = 1)
y = df1['salary']
```

In [23]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=7,test_size=0.2)
```

In [31]:

```
'''Hyperparameters of Decision Tree Classifier'''
DTC_parameters = {
 'criterion' : ['gini', 'entropy', 'log_loss'],
 'splitter' : ['best', 'random'],
 'max_depth' : range(1,10,1),
 'min_samples_split' : range(2,10,2),
 'min_samples_leaf' : range(1,5,1),
 'max_features' : ['auto', 'sqrt', 'log2']
}

'''Hyperparameters of Bagging Classifier'''
Bagging_parameters = {
 'n_estimators' : [5, 10, 15],
 'max_samples' : range(2, 10, 1),
 'max_features' : range(2, 10, 3)
}

'''Hyperparameters of Random Forest Classifier'''
RFC_parameters = {
  'criterion' : ['gini', 'entropy', 'log_loss'],
 'max_depth' : range(1, 10, 1),
 'min_samples_split' : range(2, 10, 2),
 'min_samples_leaf' : range(1, 10, 1),
}

'''Hyperparameters of Random Forest Classifier'''
ETC_parameters = {
 'n_estimators' : [10,20,30],
 'criterion' : ['gini', 'entropy', 'log_loss'],
 'max_depth' : range(2,10,1),
 'min_samples_split' : range(2,10,2),
 'min_samples_leaf' : range(1,5,1),
 'max_features' : ['sqrt', 'log2']
}

'''Hard and Soft Voting Classifier'''
lr = LogisticRegression(multi_class='multinomial', random_state=7)
rfc = RandomForestClassifier(n_estimators=50, random_state=7)
svc = SVC(probability=True, random_state=7)

'''All Models'''
models = {
1 : DecisionTreeClassifier(),
2 : GridSearchCV(estimator = DecisionTreeClassifier(), param_grid = DTC_parameters, verbose=2, n_jobs = -1, cv=3),
3 : BaggingClassifier(),
4 : GridSearchCV(estimator = BaggingClassifier(), param_grid = Bagging_parameters, verbose=2, n_jobs = -1, cv=3),
5 : RandomForestClassifier(),
6 : GridSearchCV(estimator = RandomForestClassifier(), param_grid = RFC_parameters, verbose=2, n_jobs = -1, cv=3),
7 : ExtraTreesClassifier(),
8 : GridSearchCV(estimator = ExtraTreesClassifier(), param_grid = ETC_parameters, verbose=2, n_jobs = -1, cv=3),
9 : VotingClassifier(estimators = [('lr', lr), ('rfc', rfc),('svc', svc)], voting='hard'),
10 : VotingClassifier(estimators = [('lr', lr), ('rfc', rfc),('svc', svc)], voting='soft')
}
```

In [32]:

```
map_keys = list(models.keys())
```

In [33]:

```
# Get model name using id from linear_model_collection
def get_model_building_technique_name(num):
    if num == 1:
        return 'DecisionTreeClassifier'
    if num == 2:
        return 'GridSearchCV_DecisionTreeClassifier'
    if num ==3:
        return 'BaggingClassifier'
    if num == 4:
        return 'GridSearchCV_BaggingClassifier'
    if num == 5:
        return 'RandomForestClassifier'
    if num == 6:
        return 'GridSearchCV_RandomForestClassifier'
    if num == 7:
        return 'ExtraTreesClassifier'
    if num == 8:
        return 'GridSearchCV_ExtraTreesClassifier'
    if num == 9:
        return 'VotingClassifier_Hard'
    if num ==10:
        return 'VotingClassifier_Soft'
    return ''
```

In [36]:

```
results = [];
for key_index in range(len(map_keys)):
    key = map_keys[key_index]
    if key in [1,2,3,4,5,6,7,8]:
        model = models[key]
        print(key)
        model.fit(X_train, y_train)

        '''Test Accuracy'''
        y_pred = model.predict(X_test)

        Accuracy_Test = accuracy_score(y_test, y_pred)
        conf_mat_Test = confusion_matrix(y_test, y_pred)
        true_positive_Test = conf_mat_Test[0][0]
        false_positive_Test = conf_mat_Test[0][1]
        false_negative_Test = conf_mat_Test[1][0]
        true_negative__Test = conf_mat_Test[1][1]
        Precision_Test = true_positive_Test /(true_positive_Test + false_positive_Test)
        Recall_Test = true_positive_Test/(true_positive_Test + false_negative_Test)
        F1_Score_Test = 2*(Recall_Test * Precision_Test) / (Recall_Test + Precision_Test)
        #AUC_Test = roc_auc_score(y_test, y_pred)


        '''Train Accuracy'''
        y_pred_train = model.predict(X_train)

        Accuracy_Train = accuracy_score(y_train, y_pred_train)
        conf_mat_Train = confusion_matrix(y_train, y_pred_train)
        true_positive_Train = conf_mat_Train[0][0]
        false_positive_Train = conf_mat_Train[0][1]
        false_negative_Train = conf_mat_Train[1][0]
        true_negative__Train = conf_mat_Train[1][1]
        Precision_Train = true_positive_Train /(true_positive_Train + false_positive_Train)
        Recall_Train = true_positive_Train/(true_positive_Train + false_negative_Train)
        F1_Score_Train = 2*(Recall_Train * Precision_Train) / (Recall_Train + Precision_Train)
        #AUC_Train = roc_auc_score(y_train, y_pred_train)

        results.append({
            'Model Name' : get_model_building_technique_name(key),
            'Trained Model' : model,
            'Accuracy_Test' : Accuracy_Test,
            'Precision_Test' : Precision_Test,
            'Recall_Test' : Recall_Test,
            'F1_Score_Test' : F1_Score_Test,
            #'AUC_Test' : AUC_Test,
            'Accuracy_Train' : Accuracy_Train,
            'Precision_Train' : Precision_Train,
            'Recall_Train' : Recall_Train,
            'F1_Score_Train' : F1_Score_Train,
            #'AUC_Train' : AUC_Train
            })
    else:
        key = map_keys[key_index]
        model = models[key]
        print(key)
        model.fit(X_train, y_train)


        '''Test Accuracy'''
        y_pred = model.predict(X_test)

        Accuracy_Test = accuracy_score(y_test, y_pred)
        conf_mat_Test = confusion_matrix(y_test, y_pred)
        true_positive_Test = conf_mat_Test[0][0]
        false_positive_Test = conf_mat_Test[0][1]
        false_negative_Test = conf_mat_Test[1][0]
        true_negative__Test = conf_mat_Test[1][1]
        Precision_Test = true_positive_Test /(true_positive_Test + false_positive_Test)
        Recall_Test = true_positive_Test/(true_positive_Test + false_negative_Test)
        F1_Score_Test = 2*(Recall_Test * Precision_Test) / (Recall_Test + Precision_Test)
        #AUC_Test = roc_auc_score(y_test, y_pred)

        '''Train Accuracy'''
        y_pred_train = model.predict(X_train)

        Accuracy_Train = accuracy_score(y_train, y_pred_train)
        conf_mat_Train = confusion_matrix(y_train, y_pred_train)
        true_positive_Train = conf_mat_Train[0][0]
        false_positive_Train = conf_mat_Train[0][1]
        false_negative_Train = conf_mat_Train[1][0]
        true_negative__Train = conf_mat_Train[1][1]
        Precision_Train = true_positive_Train /(true_positive_Train + false_positive_Train)
        Recall_Train = true_positive_Train/(true_positive_Train + false_negative_Train)
        F1_Score_Train = 2*(Recall_Train * Precision_Train) / (Recall_Train + Precision_Train)
        #AUC_Train = roc_auc_score(y_train, y_pred_train)

        results.append({
            'Model Name' : get_model_building_technique_name(key),
            'Trained Model' : model,
            'Accuracy_Test' : Accuracy_Test,
            'Precision_Test' : Precision_Test,
            'Recall_Test' : Recall_Test,
            'F1_Score_Test' : F1_Score_Test,
            #'AUC_Test' : AUC_Test,
            'Accuracy_Train' : Accuracy_Train,
            'Precision_Train' : Precision_Train,
            'Recall_Train' : Recall_Train,
            'F1_Score_Train' : F1_Score_Train,
            #'AUC_Train' : AUC_Train
            })
```

```
1
2
Fitting 3 folds for each of 2592 candidates, totalling 7776 fits
3
4
Fitting 3 folds for each of 72 candidates, totalling 216 fits
5
6
Fitting 3 folds for each of 972 candidates, totalling 2916 fits
7
8
Fitting 3 folds for each of 2304 candidates, totalling 6912 fits
9
10
```

In [37]:

```
result_df = pd.DataFrame(results)
result_df
```

Out[37]:

	Model Name	Trained Model	Accuracy_Test	Precision_Test	Recall_Test	F1_Score_Test	Accu
0	DecisionTreeClassifier	DecisionTreeClassifier()	0.449862	0.647032	0.660094	0.653498	
1	GridSearchCV_DecisionTreeClassifier	GridSearchCV(cv=3, estimator=DecisionTreeClass...	0.562737	0.999149	0.662904	0.797014	
2	BaggingClassifier	(DecisionTreeClassifier(random_state=164281282...	0.512650	0.827063	0.661563	0.735113	
3	GridSearchCV_BaggingClassifier	GridSearchCV(cv=3, estimator=BaggingClassifier...	0.488989	1.000000	0.665820	0.799390	
4	RandomForestClassifier	(DecisionTreeClassifier(max_features='auto', r...	0.534160	0.884201	0.663588	0.758172	
5	GridSearchCV_RandomForestClassifier	GridSearchCV(cv=3, estimator=RandomForestClass...	0.570112	0.999788	0.662540	0.796954	
6	ExtraTreesClassifier	(ExtraTreeClassifier(random_state=1336411775),...	0.512855	0.828571	0.662358	0.736200	
7	GridSearchCV_ExtraTreesClassifier	GridSearchCV(cv=3, estimator=ExtraTreesClassif...	0.566322	1.000000	0.663687	0.797851	
8	VotingClassifier_Hard	VotingClassifier(estimators=[('lr',\n ...	0.544812	1.000000	0.664411	0.798374	
9	VotingClassifier_Soft	VotingClassifier(estimators=[('lr',\n ...	0.545631	0.995447	0.665100	0.797414	

In [ ]:

In [ ]: