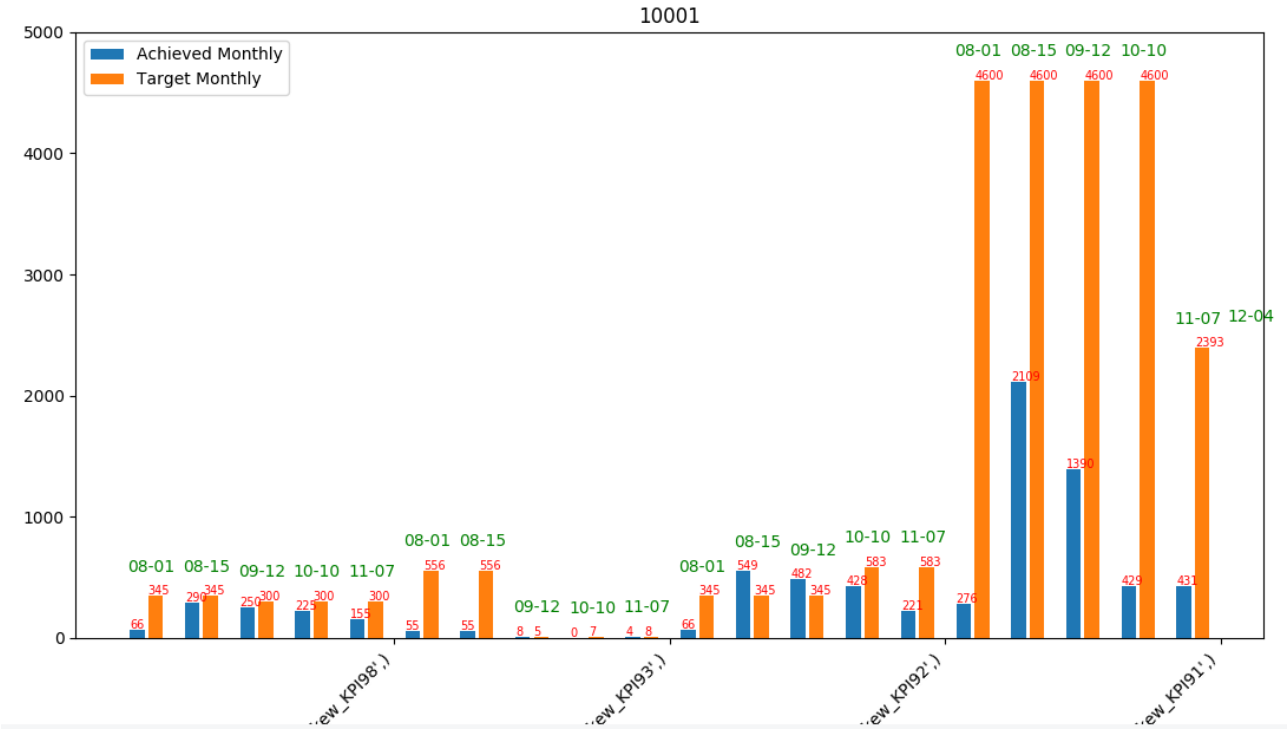
WEEK 1:

- Studied the given database and Visulaized the Employee data of the database.
- Visulaized the data using Matplotlib in python.

TASK:

To visualize the data of the first four business period of four active KPI's of an active employee and build a Machine Learning Model to predict the monthly target for each KPI.

Visualization Part:



Machine Learning Part:

Model Name: Linear Regression with Polynomial features.
Independent variables: KPI name, Current Month,Number of days, Monthly target, Last month achievement.
Dependent variables: Monthly Achievement
Data preprocessing: Train Test split, Label Encoding

Result:

Input: 0,9,27,4600,2109
Predicted Result: [[1390.00000001]]
Mean Absolute Error: 1.5565799760253185e-09
Mean Squared Error: 6.4302375599701375e-18
Root Mean Squared Error: 2.535791308442029e-09

Preprocessing Code:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder = LabelEncoder()
k2 = labelencoder.fit_transform(k2)
```

Model Code:

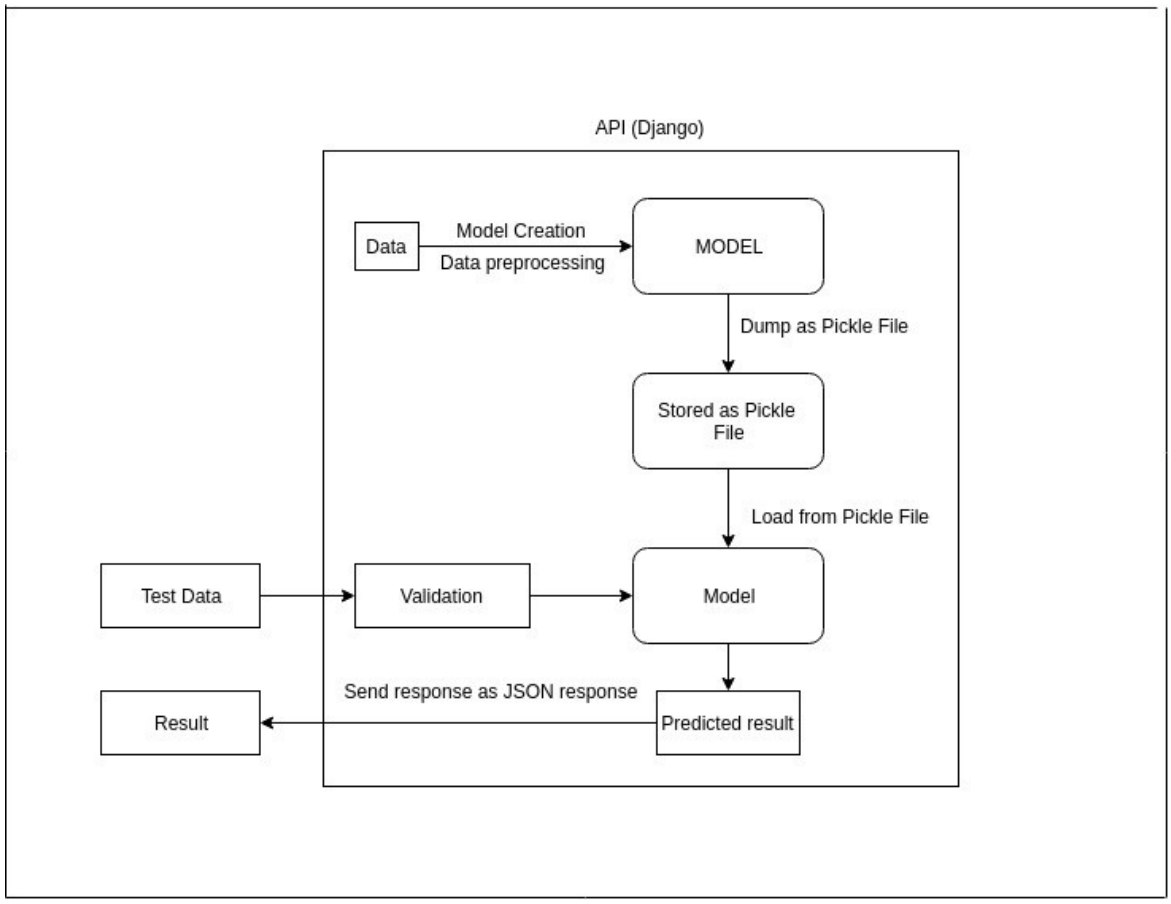
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(res)
poly_reg.fit(X_poly,c1)
regressor = LinearRegression()
regressor.fit(X_poly, c1)
```

WEEK 2:

Task:

- Create an API in django .
- Create a machine learning model to predict the salary of an employee based on his experience.
- Dump the model in the pickle format and load the model again to predict the results in form of API.
- Get the response in JSON format.

Block Diagram:



Machine Learning Part:

Model Name: Linear Regression,Polynomial Regression.
Independent variables: Experience.
Dependent variables: Salary
Data preprocessing: Train Test split

Result:

Code:
Training and Dumping Model:

Polynomial Regression:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
poly_reg.fit(X_poly, y)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

Linear Regression:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Dump Model:

```
modulePath = os.path.dirname(__file__)
filePath = os.path.join(modulePath,'linear_regression/model_{0}.sav'.format(datetime_str))
pickle.dump(regressor, open(filePath, 'wb'))
```

Predicting and Loading Model:

```
modulePath = os.path.dirname(__file__)
filePath = os.path.join(modulePath, '{1}/{0}'.format(last_line[:-1],l1))
model = pickle.load(open(filePath, 'rb'))
 if(l1=='poly_regression'):
        ans=(model[1].fit_transform(np.array([[float(i)]])))
        flt=float(model[0].predict(ans))
    else:
        flt=float(model.predict(np.array([[float(i)]])))
```

WEEK 3 &4:

Problem Statement:

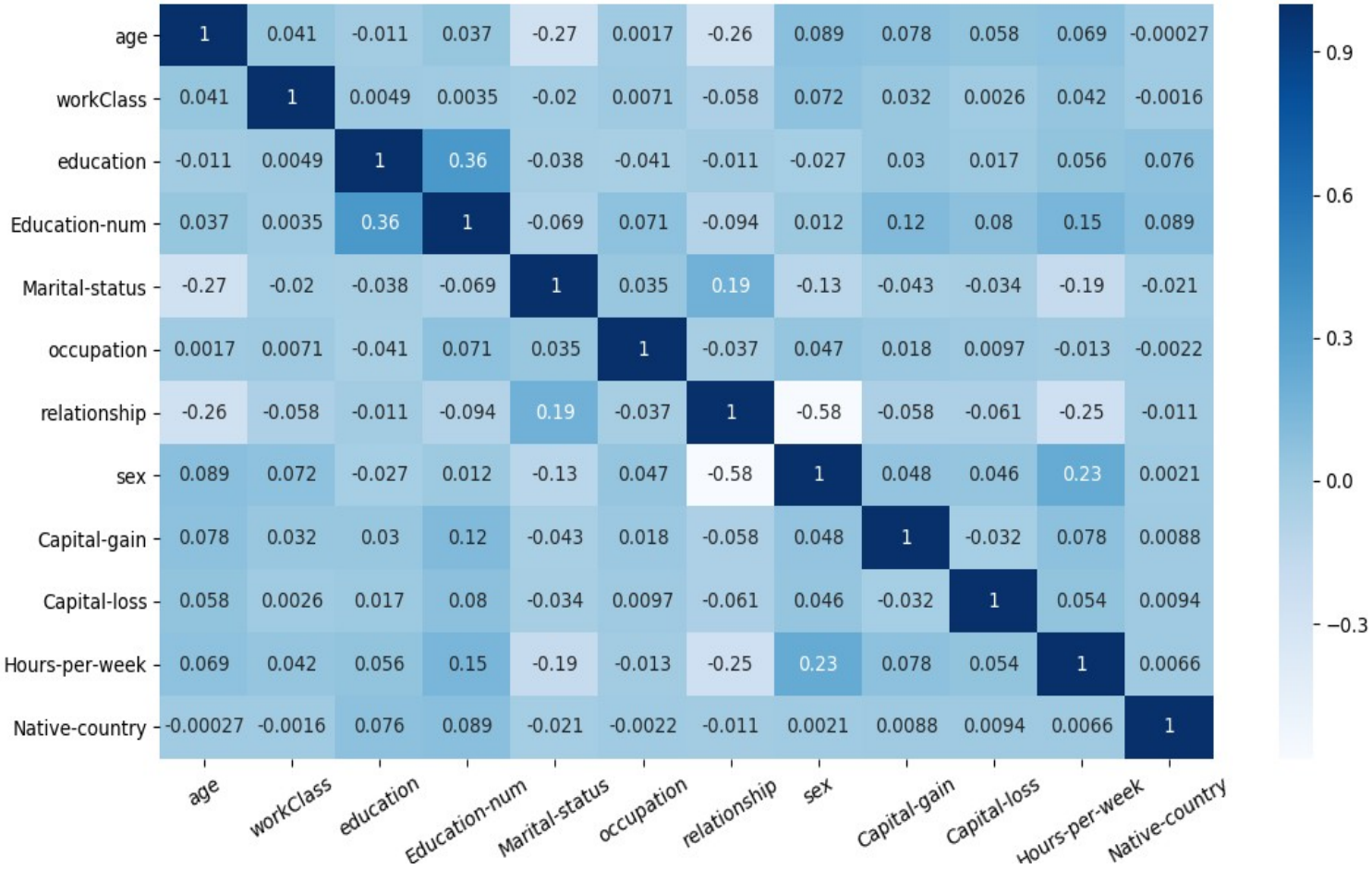
- Create an API in django to predict the Salaray of an adult using the given data.
- Algorithms to be used:

Logistic Regression
KNN
Naive Bayes
SVM
Random Forest
XG Boost

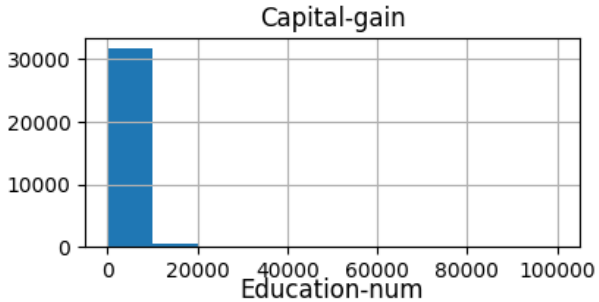
Machine Learning Part:

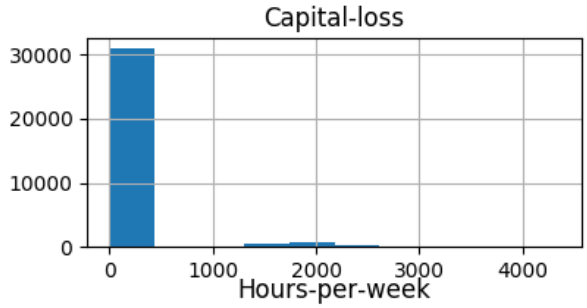
Model Name: Logistic Regression, KNN,SVM,Naive Bayes,Random Forest,XG Boost.
Independent variables: Age,Workclass,Education,Education-num,Marital Status, Occupation,Relationship,Sex,Capital-gain,Capital-loss,Hours per week,Native country.
Dependent variables: Salary.
Data preprocessing: Label Encoding, Test Train Split, Standard scaling.

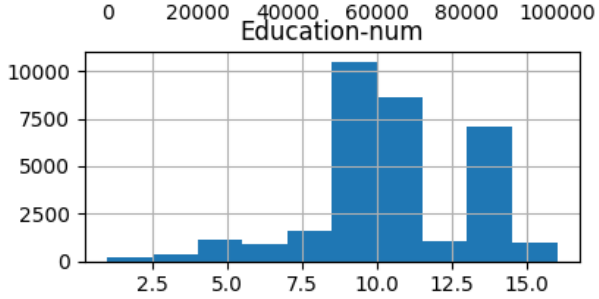
Correlation:

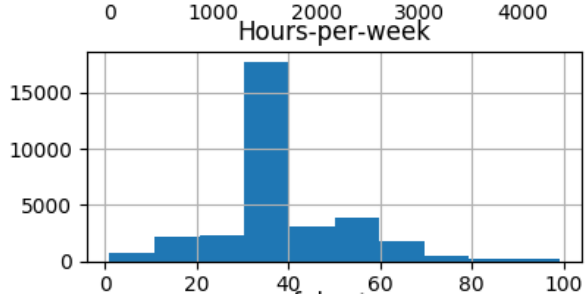


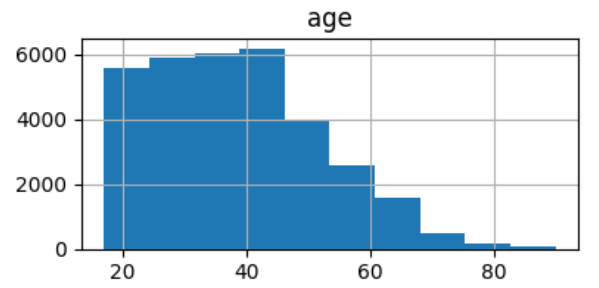
Univariate Analysis:

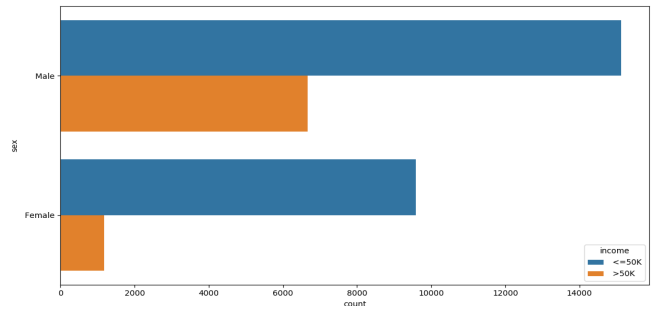




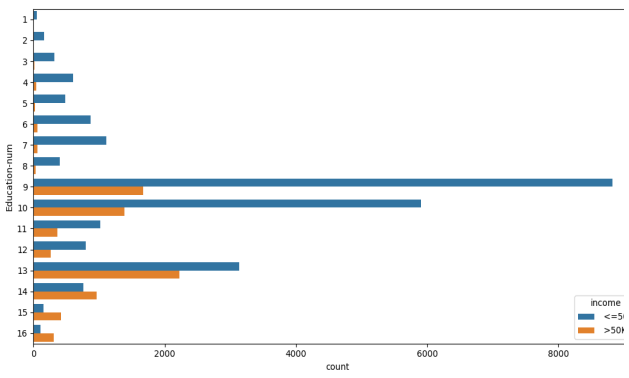




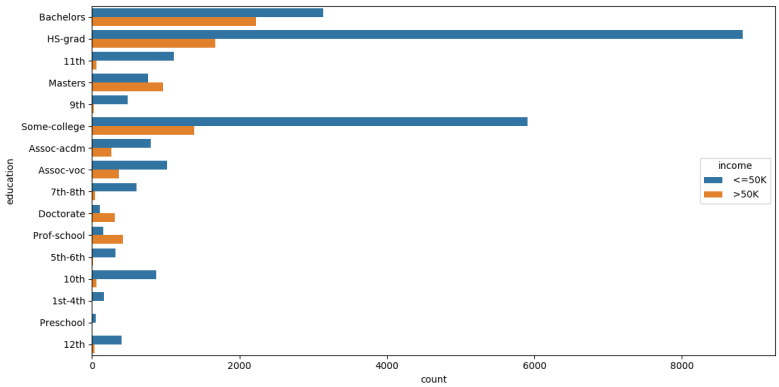




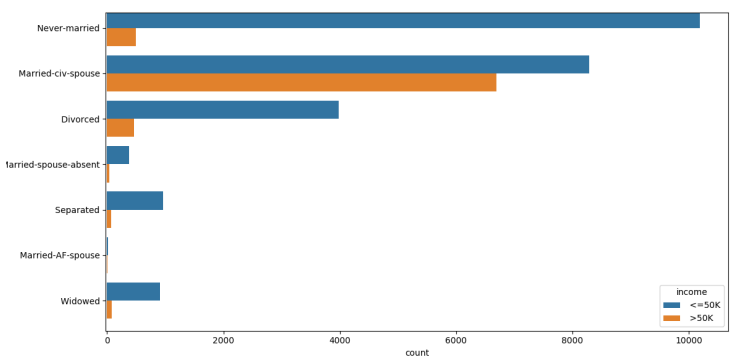
Education -num



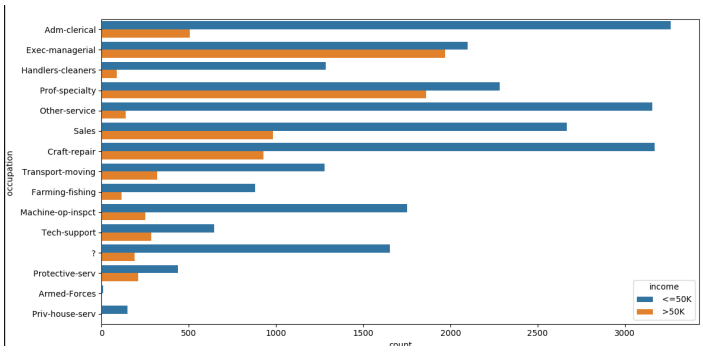
Education



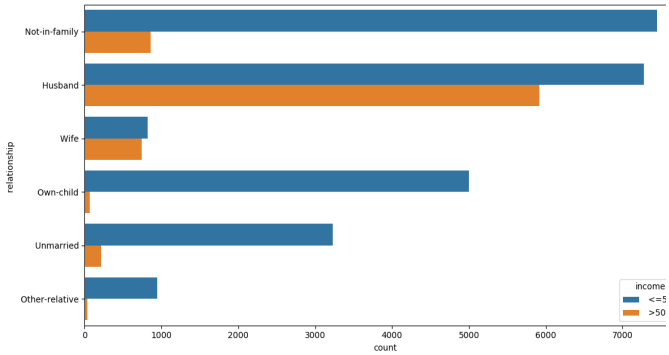
Marital Status



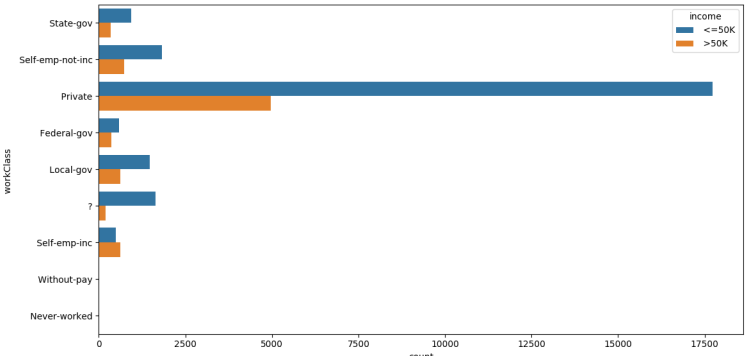
Occupation



Relationship



Education



Result:

```
// 20191219192338
// http://127.0.0.1:8000/train/?l1=Logistic_Regression

{
  "Status ": "Model created sucessfully",
  " Model Type": "Logistic_Regression",
  " Model Name": "model_Logistic_Regression_2019-12-19-13-53-36.pickle",
  " Accuracy": 0.8205380174425746,
  " Recall": 0.8419213973799127,
  " Precision": 0.939113492450073
}
```

```
// 20191219192249
// http://127.0.0.1:8000/train/?l1=KNN

{
  "Status ": "Model created sucessfully",
  " Model Type": "KNN",
  " Model Name": "model_KNN_2019-12-19-13-52-38.pickle",
  " Accuracy": 0.8637759489006265,
  " Recall": 0.8814199395770392,
  " Precision": 0.9473940574768631
}
```

```
// 20191219192431
// http://127.0.0.1:8000/train/?l1=Naive_bayes

{
  "Status ": "Model created sucessfully",
  " Model Type": "Naive_bayes",
  " Model Name": "model_Naive_bayes_2019-12-19-13-54-29.pickle",
  " Accuracy": 0.8043237931458052,
  " Recall": 0.8183212493028444,
  " Precision": 0.9529144341613899
}
```

```
// 20191219192151
// http://127.0.0.1:8000/train/?l1=SVM

{
  "Status ": "Model created sucessfully",
  " Model Type": "SVM",
  " Model Name": "model_SVM_2019-12-19-13-51-11.pickle",
  " Accuracy": 0.8127994103918438,
  " Recall": 0.8149205055034652,
  " Precision": 0.9736970287384316
}
```

```
// 20191230130204
// http://127.0.0.1:8000/train/?l1=XG_Boost

{
  "Status ": "Model created sucessfully",
  " Model Type": "XG_Boost",
  " Model Name": "model_XG_Boost.pickle",
  " Accuracy": 0.8599680628915367,
  " Recall": 0.8784497059267079,
  " Precision": 0.9457704172755318
}
```

```
// 20191230130044
// http://127.0.0.1:8000/train/?l1=Random_Forest

{
  "Status ": "Model created sucessfully",
  " Model Type": "Random_Forest",
  " Model Name": "model_Random_Forest.pickle",
  " Accuracy": 0.8432624984645621,
  " Recall": 0.8763681208570988,
  " Precision": 0.9230394544568924
}
```

```
// 20191219192054
// http://127.0.0.1:8000/result/?l3=37,Private,Some-college,10,Married-civ-spouse,Exec-managerial,Husband,Male,0,0,80,United-States&l1=KNN

{
  "Result": ">50k"
}
```

Code:

Training:

```
    if l1=="Logistic_Regression":
        from sklearn.linear_model import LogisticRegression
        classifier = LogisticRegression(random_state=0)
        classifier.fit(X_train,y_train)
    if l1=="SVM":
        from sklearn.svm import SVC
        classifier = SVC(kernel = 'linear', random_state = 0)
        classifier.fit(X_train,y_train)
    if l1=="KNN":
        from sklearn.neighbors import KNeighborsClassifier
        classifier = KNeighborsClassifier(n_neighbors = 10, metric = 'minkowski', p = 2)
        classifier.fit(X_train,y_train)
    if l1=="Naive_bayes":
        from sklearn.naive_bayes import GaussianNB
        classifier.fit(X_train,y_train)
        classifier.fit(X_train,y_train)
    if l1=="Random_Forest":
        from sklearn.ensemble import RandomForestClassifier
        classifier = RandomForestClassifier()
        grid_select.grid_RFA(classifier)
        classifier.fit(X_train,y_train)
    if l1=="XG_Boost":
       from xgboost import XGBClassifier
       classifier = XGBClassifier()
       grid_select.grid_XG(classifier)
       classifier.fit(X_train, y_train)
```

Prediction:

```
  labelencoder = LabelEncoder()
  l=[1,2,4,5,6,7,11]
  for i in l:
     X[:,i]= labelencoder.fit_transform(X[:,i])
     temp[:,i]= labelencoder.transform((temp[:,i]))
  temp = np.asarray(temp, dtype=np.int32, order='C')
  from sklearn.preprocessing import StandardScaler
  sc = StandardScaler()
  X = sc.fit_transform(X)
  temp=sc.transform(temp)

  result=model.predict(temp)
  if result==[0]:
     result="<=50k"
  else:
     result=">50k"
  responseData = {
     "Result":result
  }
  return  JsonResponse(responseData,safe=False)
```

Grid Search:

```
def grid_SVM(classifier):
    kernel=['linear','rbf','poly','sigmoid']
    GridSearchCV(estimator=classifier,cv=3,param_grid=dict(kernel=kernel,random_state=[0]))

def grid_KNN(classifier):
    neighbor=[10]
    metric=['minkowski']
    p=[2]
    grid=GridSearchCV(estimator=classifier,cv=3,param_grid=dict(n_neighbors =neighbor, metric =
metric, p = p))

def grid_RFA(classifier):
    est=[10,20,30]
    cri=['entropy','gini']
    grid=GridSearchCV(estimator=classifier,cv=3,param_grid=dict(n_estimators = est, criterion =
cri, random_state = [0]))

def grid_XG(classifier):
    boost=['gbtree','gblinear','dart']
    grid=GridSearchCV(estimator=classifier,cv=3,param_grid=dict(booster=boost))
```

Result (Grid search):

```
// 20191230130344
// http://127.0.0.1:8000/train/?l1=SVM

{
  "Status ": "Model created sucessfully",
  " Model Type": "SVM",
  " Model Name": "model_SVM.pickle",
  " Accuracy": 0.8455963640830365,
  " Recall": 0.8681285671372785,
  " Precision": 0.9384640363695405
}
```

```
// 20191230130344
// http://127.0.0.1:8000/train/?l1=SVM

{
  "Status ": "Model created sucessfully",
  " Model Type": "SVM",
  " Model Name": "model_SVM.pickle",
  " Accuracy": 0.8455963640830365,
  " Recall": 0.8681285671372785,
  " Precision": 0.9384640363695405
}
```

```
// 20191230130351
// http://127.0.0.1:8000/train/?l1=Random_Fores

{
  "Status ": "Model created sucessfully",
  " Model Type": "Random_Forest",
  " Model Name": "model_Random_Forest.pickle",
  " Accuracy": 0.8454735290504852,
  " Recall": 0.8818762661679913,
  " Precision": 0.9188179899334308
}
```

```
// 20191230130911
// http://127.0.0.1:8000/train/?l1=XG_Boost

{
  "Status ": "Model created sucessfully",
  " Model Type": "XG_Boost",
  " Model Name": "model_XG_Boost.pickle",
  " Accuracy": 0.8599680628915367,
  " Recall": 0.8784497059267079,
  " Precision": 0.9457704172755318
}
```

React:

Build a google authentication API module using express and mongo db using Oauth.