MANIPAL UNIVERSITY
JAIPUR

DATA SCIENCE PROJECT ON K-NN ALGORITHM

TANYA GUPTA
189302170

INTRODUCTION :

This project will be a walkthrough of a simple KNN model in an attempt to strategize a basic ad-targeting campaign for a social media network/website. One of our sponsor's commercials seems to be particularly successful among our older, wealthier users but seemingly less-so with our younger ones. We'd like to put in place an effective model so that we can figure out who our target demographic is for this particular event, thus maximizing our click-through rate. We'd like to show this ad to younger users with a lower probability than older/wealthier users, and use the time/space to expose them to content they're more likely to be interested in.

DATASET DESCRIPTION:

Our dataset contains some information about all of our users in the social network, including their User ID, Gender, Age, and Estimated Salary. The last column of the dataset is a vector of Booleans describing whether or not each individual ended up clicking on the advertisement (0 = False, 1 = True)

Age : Age of an individual in years.

User ID : Unique ID assigned to each individual.

Gender : Gender of each individual (Male/ Female).

Estimated Salary : Salary of an individual

Purchased : 0 =False , 1= True

We're just concerned right now about how the users' Age and Estimated Salary influence their decision to click or not click on the advertising. To do so, we'll use our dataset to derive the related vectors: the independent variables (X) and the dependent variable (Y).

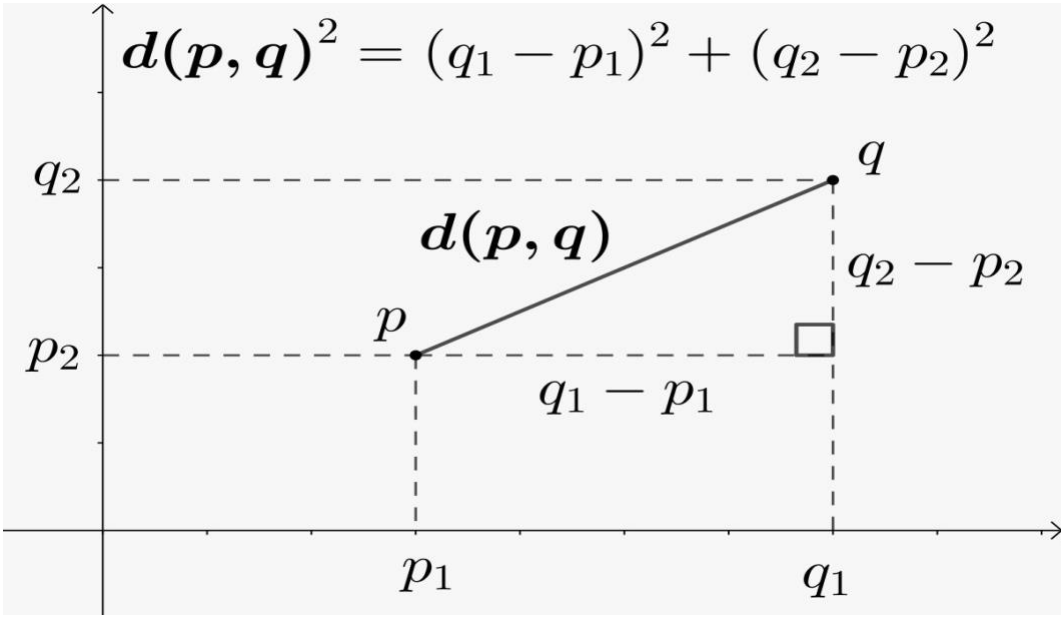
ALGORITHM USED :

The algorithm used for our project is a very simple & versatile and one of the topmost machine learning algorithms- K Nearest Neighbour(KNN).

KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution, i.e, the model structure is determined from the dataset. Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier.

The algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories and stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

It calculates the distance between the test data and the input in the graph and gives the prediction according. To calculate this distance in a given graph, it uses the Euclidean distance formula given below:



The K-NN working can be explained on the basis of below algorithm:

STEP 1 : Decide on the number of neighbors (K).
STEP 2 : Determine the Euclidean distance between neighbors.
SETP 3 : Using the measured Euclidean distance, find the K closest neighbors.
STEP 4 : Count the number of data points in each category among these k neighbors.
STEP 5 : Assign the new data points to the group with the greatest number of neighbors.
STEP 6 : We've completed the model.

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set(color_codes=True)
import matplotlib.pyplot as plt
```

Exploratory Data Analysis

In [2]:
```
data=pd.read_csv("Social_Network_Ads.csv")
```

In [159]:
```
data
```

Out[159]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [10]:
```
data.describe()
```

Out[10]:

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

In [11]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

In [12]:
```
data.isnull().any()
```

Out[12]:
```
User ID            False
Gender             False
Age                False
EstimatedSalary    False
Purchased          False
dtype: bool
```

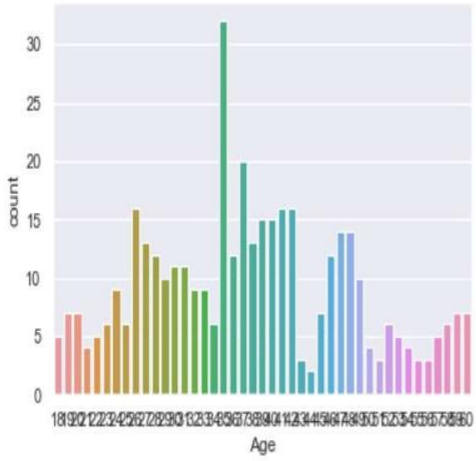
In [178]:
```
data.head()
```

Out[178]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

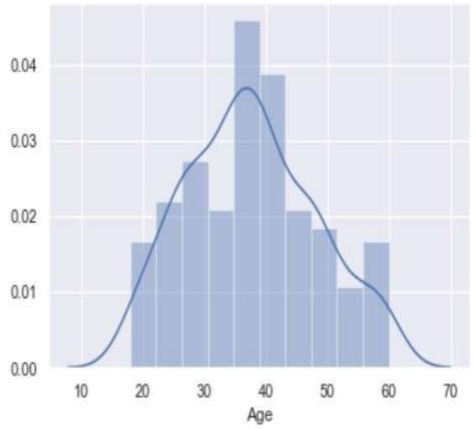
DATA VISUALISATION :

In [112]:
```
x= data['Age']
ax = sns.countplot(x=x, data=data)
```



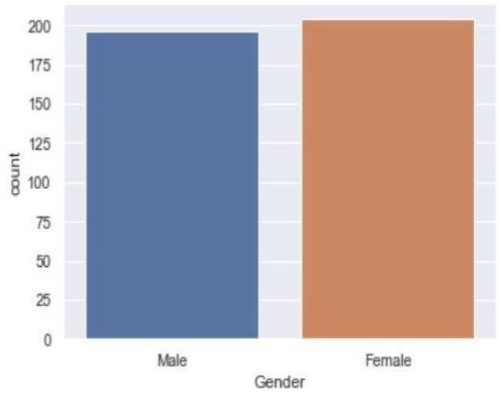
In [21]:
```
sns.distplot(data['Age'])
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x23a398467c8>



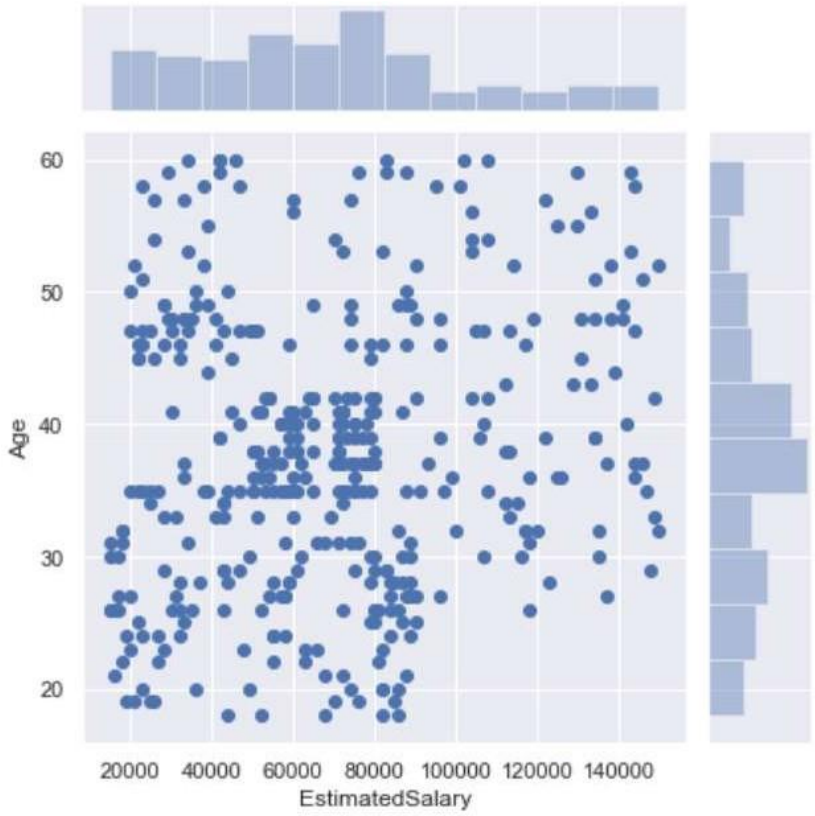
In [25]:
```
sns.countplot(data['Gender'])
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x23a3ab4dfc8>



In [23]:
```
sns.jointplot(data['EstimatedSalary'], data['Age'])
```

Out[23]:
```
<seaborn.axisgrid.JointGrid at 0x23a382a4fc8>
```



FOR 80:20 RATIO :

We split our data. 80% of our data will be train data and 20% of it will be test data.

FEATURE SCALING

In [125]:
```
#Importing the dataset
x=data.iloc[:, [2,3]].values
y=data.iloc[:,4].values
```

In [167]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In [168]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [169]:
```
# Fitting classifier to the Training set
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski',p=2)
classifier.fit(x_train, y_train)
```

Out[169]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [170]:
```
y_pred=classifier.predict(x_test)
```

In [171]:
```
from sklearn.metrics import accuracy_score
```

In [172]:
```
accuracy_score(y_test,y_pred)
```

Out[172]:
```
0.95
```

Confusion Matrix :

In [173]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
```

In [174]:
```
cm
```

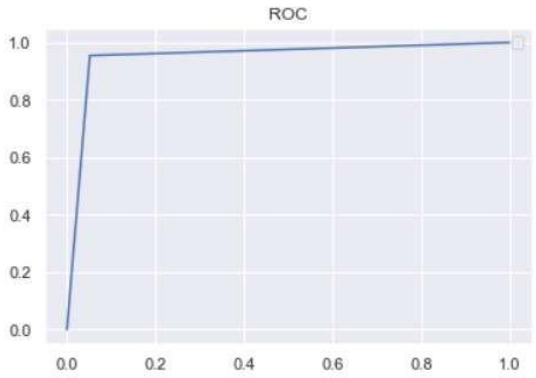
Out[174]:
```
array([[55,  3],
       [ 1, 21]], dtype=int64)
```

ROC :

In [175]:
```
import sklearn.metrics as metrics
fpr,tpr,threshold=metrics.roc_curve(y_test,y_pred)
roc_auc=metrics.auc(fpr,tpr)
```

In [176]:
```
roc_auc
```

Out[176]:
```
0.95141065830721
```



FOR 70:30 RATIO :

We split our data. 70% of our data will be train data and 30% of it will be test data.

```
In [179]: # Splitting the dataset into the Training set and Test set
          from sklearn.model_selection import train_test_split
          x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)

In [183]: from sklearn.metrics import accuracy_score

In [184]: accuracy_score(y_test,y_pred)
Out[184]: 0.9166666666666666
```

Confusion Matrix :

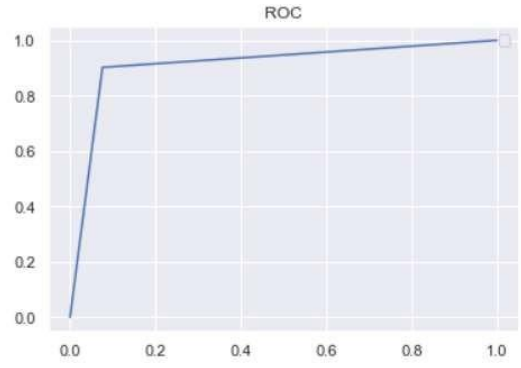
```
In [185]: from sklearn.metrics import confusion_matrix
          cm=confusion_matrix(y_test,y_pred)

In [186]: cm
Out[186]: array([[73,  6],
                 [ 4, 37]], dtype=int64)
```

ROC :

```
In [187]: import sklearn.metrics as metrics
          fpr,tpr,threshold=metrics.roc_curve(y_test,y_pred)
          roc_auc=metrics.auc(fpr,tpr)

In [188]: roc_auc
Out[188]: 0.9132448286508181
```



FOR 60:40 RATIO :

We split our data. 60% of our data will be train data and 40% of it will be test data.

In [190]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4,random_state=0)
```

In [194]:
```
from sklearn.metrics import accuracy_score
```

In [195]:
```
accuracy_score(y_test,y_pred)
```

Out[195]: 0.89375

Confusion Matrix :

In [196]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
```

In [197]:
```
cm
```

Out[197]:
```
array([[93,  8],
       [ 9, 50]], dtype=int64)
```

In [198]:
```
import sklearn.metrics as metrics
fpr,tpr,threshold=metrics.roc_curve(y_test,y_pred)
roc_auc=metrics.auc(fpr,tpr)
```

In [199]:
```
roc_auc
```

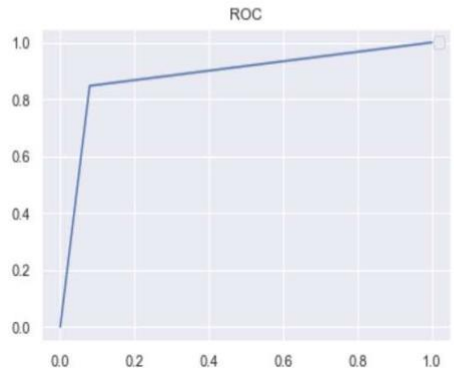
Out[199]: 0.8841248531632824

ROC :

In [200]:
```
plt.title('ROC')
plt.plot(fpr,tpr,color='b')
plt.legend()
plt.plot
```

```
No handles with labels found to put in legend.
```

Out[200]: <function matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)>



EVALUATION PARAMETERS FOR K-NN :

RATIO/MEASURES	Train Test Ratio 1 (80:20)	Train Test Ratio 2 (70:30)	Train Test Ratio 3 (60:40)
Specificity	0.982	0.9481	0.911
Sensitivity	0.875	0.8605	0.862
Accuracy	0.95	0.91667	0.893
Precision	0.875	0.8605	0.862
FPR	0.053	0.07792	0.078
FNR	0.041	0.09302	0.155
NPV	0.982	0.94805	0.911
FDR	0.125	0.1395	0.137
F1-Score	0.9119	0.88095	0.854
MCC	0.8796	0.8175	0.771