Customer behavior analysis

September 5, 2023

1 Social media advertising

The goal of a business is to understand the needs of its customers and respond to them through products. Knowledge of customer needs allows the company to guide its strategy and development process. By observing certain characteristics of their customers such as age, gender and salary, companies study how customers interact with the products they offer. This technique is used more and more, it allows us to respond to the needs and desires of customers in order to retain them.

The study of consumer behavior on social networks in relation to the purchase of products online is becoming increasingly important. One of the main advantages of this type of advertising is that the company can exploit the demographic, behavioural and geographical information of users and target their advertising appropriately.b

1.0.1 *Objet de l'étude*

In this project, I study the dependence on the purchase of a product according to the sex, age and estimated salary of a person. Consequently, the study of the advertising strategy allows the company to know with which group of consumers it should make more advertising.

1.0.2 *Dataset*

The data contains 5 columns:

UserID: Identifier of each person who purchased the product or not.

Gender: The person can be male or female.

Age: Age of person

Estimated Salary: A person's salary

Purchased: This is a binary variable (0, 1). 0 means product not purchased and 1 means product purchased. This variable is our target variable.

The dataset comes from the Kaggle site which can be downloaded here

Libraries important

[85]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the dataset

[86]:
```python
df = pd.read_csv("Social_Network_Ads.csv")
```

[87]:
```python
# First 5 rows of the dataset
df.head()
```

[87]:

	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

[88]:
```python
# Dataset details
df.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
```

I find that there is no missing data in my dataset and that I have two types of data (int64 and object). And the size of my dataset is 400 observations with 5 columns.

[89]:
```python
# Dataset shape
df.shape
```

[89]: (400, 5)

Descriptive statistics

reference

[90]:
```python
df.iloc[:, 1:].describe()
```

[90]:

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000

50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

There are no large ranges in the data, but scaling will allow Machine Learning models to better adapt to the data.

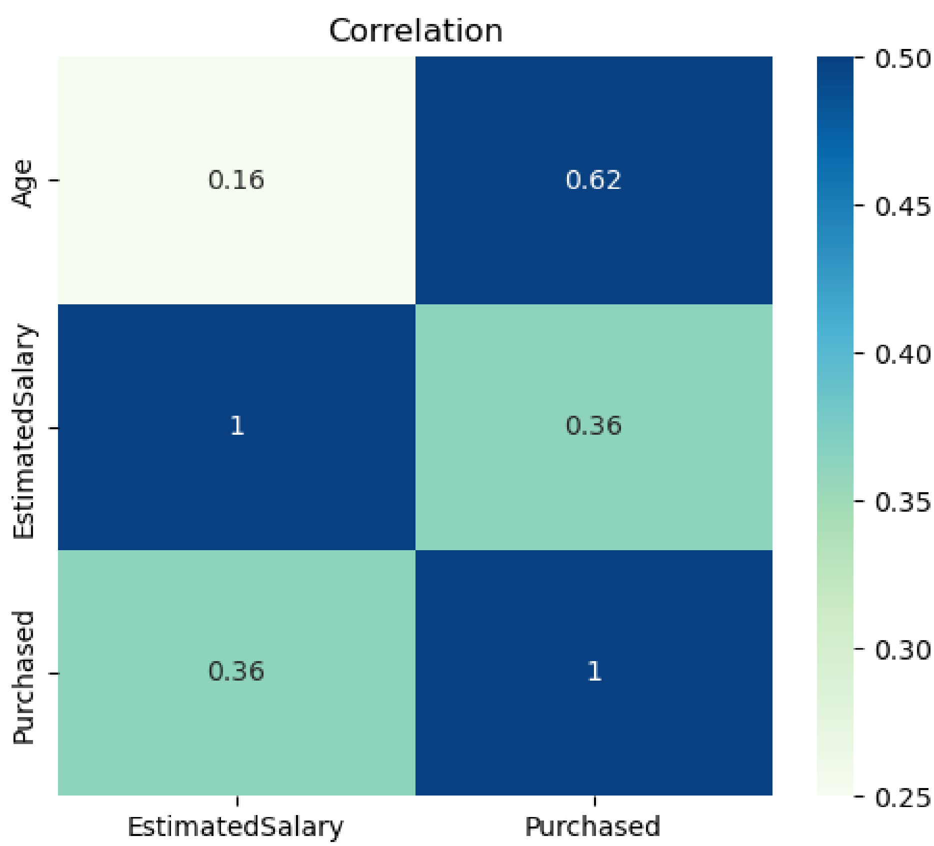
Correlation

reference

reference

[171]:
```
#Correlation between numerical features
plt.figure(figsize=(6, 5))
sns.heatmap(df.iloc[:, 1:].select_dtypes(include =np.number).corr().iloc[:, 1:
 ↪], cmap='GnBu', vmin=0.25, vmax=0.5, annot = True)
plt.title("Correlation")
```

[171]: Text(0.5, 1.0, 'Correlation')



The correlation between features is low. However, it is around 62% and 36% between the target variable and the variable "age" and "EtimatedSalary".

1.0.3 *Data visualization*

Count plot

The countplot function can be used to represent the number of observations of a categorical variable for each group with bars.

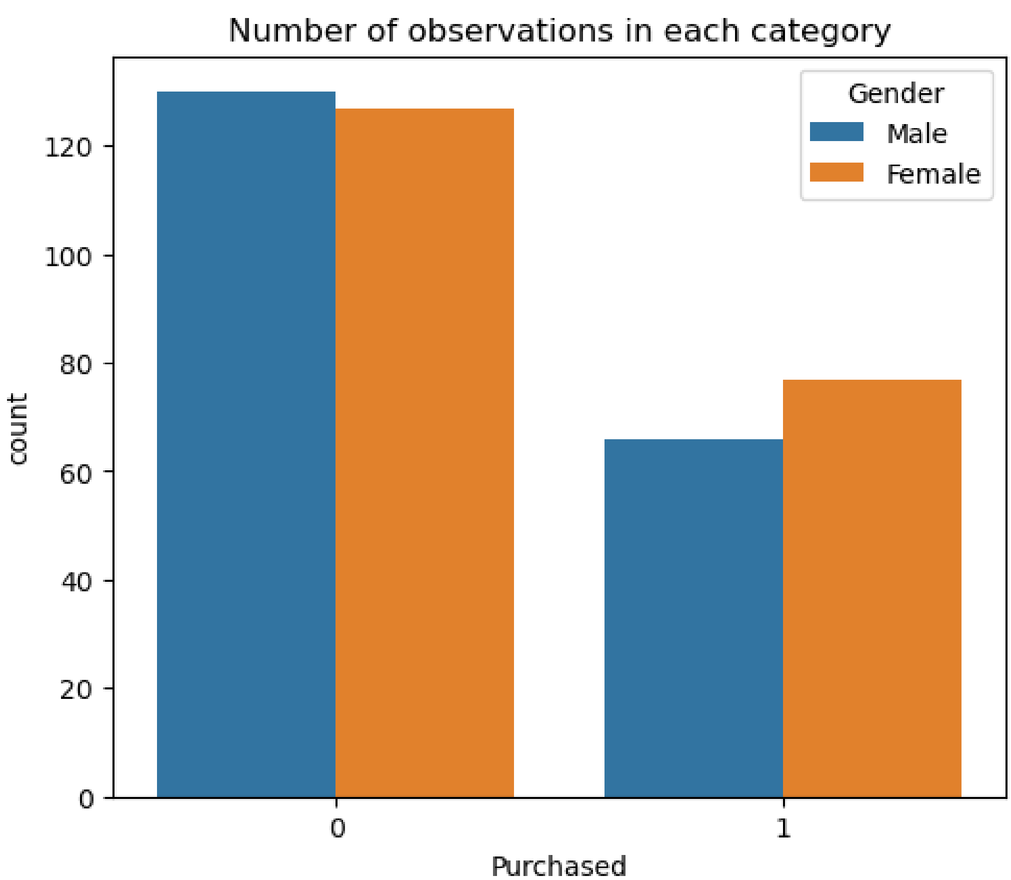
reference

reference

Purchased

[92]:
```
plt.figure(figsize=(6, 5))
sns.countplot(x="Purchased", hue='Gender', data = df)
plt.title("Number of observations in each category")
```

[92]:
```
Text(0.5, 1.0, 'Number of observations in each category')
```



[93]:
```
df.columns
```

[93]: Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'],
dtype='object')

[94]:
```
df['Purchased'].value_counts()
```

[94]:
```
0    257
1    143
Name: Purchased, dtype: int64
```

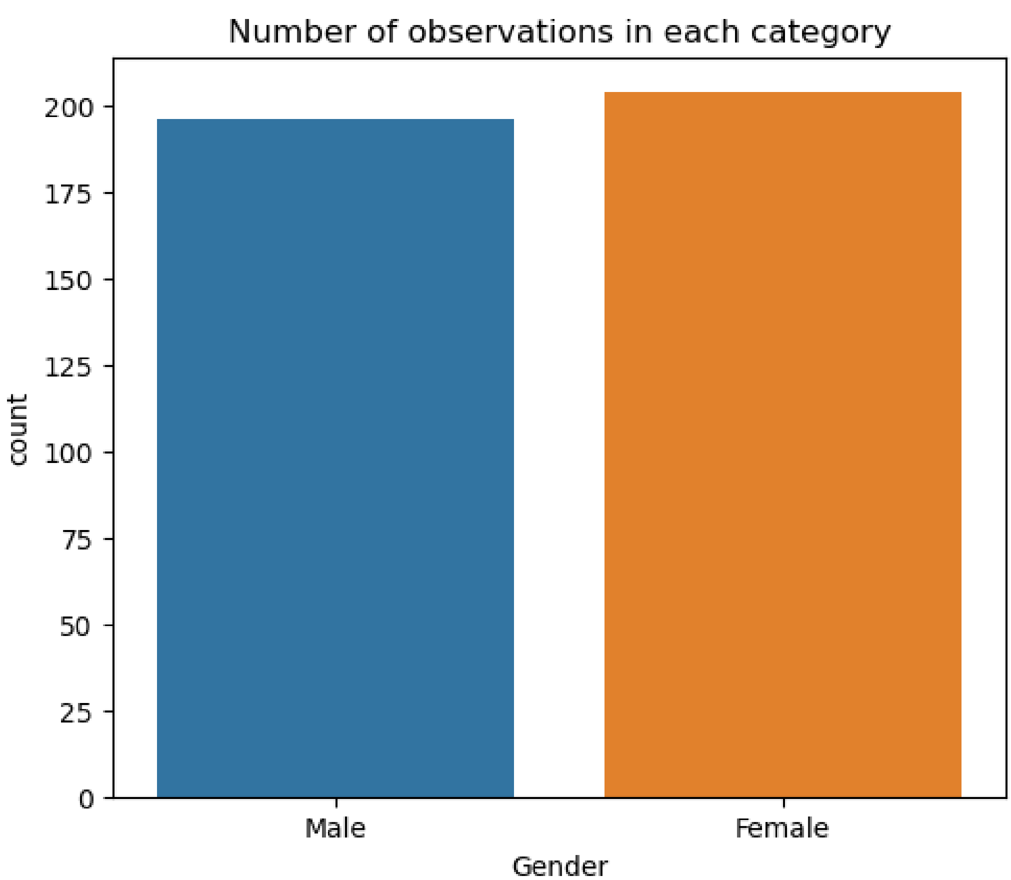
I see that class 0 is almost twice as big as class 1. I will try to keep my data as is, but for more details on unbalanced classes you can check this link

I also notice that women buy more than men. In this case, the company must continue to target more women in its advertising campaigns while keeping an eye on men.

Gender

[95]:
```
plt.figure(figsize=(6, 5))
sns.countplot(x="Gender", data = df)
plt.title("Number of observations in each category")
```

[95]: Text(0.5, 1.0, 'Number of observations in each category')



[96]:
```
df["Gender"].value_counts()
```

[96]:
```
Female    204
Male      196
Name: Gender, dtype: int64
```

We see that there are more women than men in the company's data, even if the gap is very small.

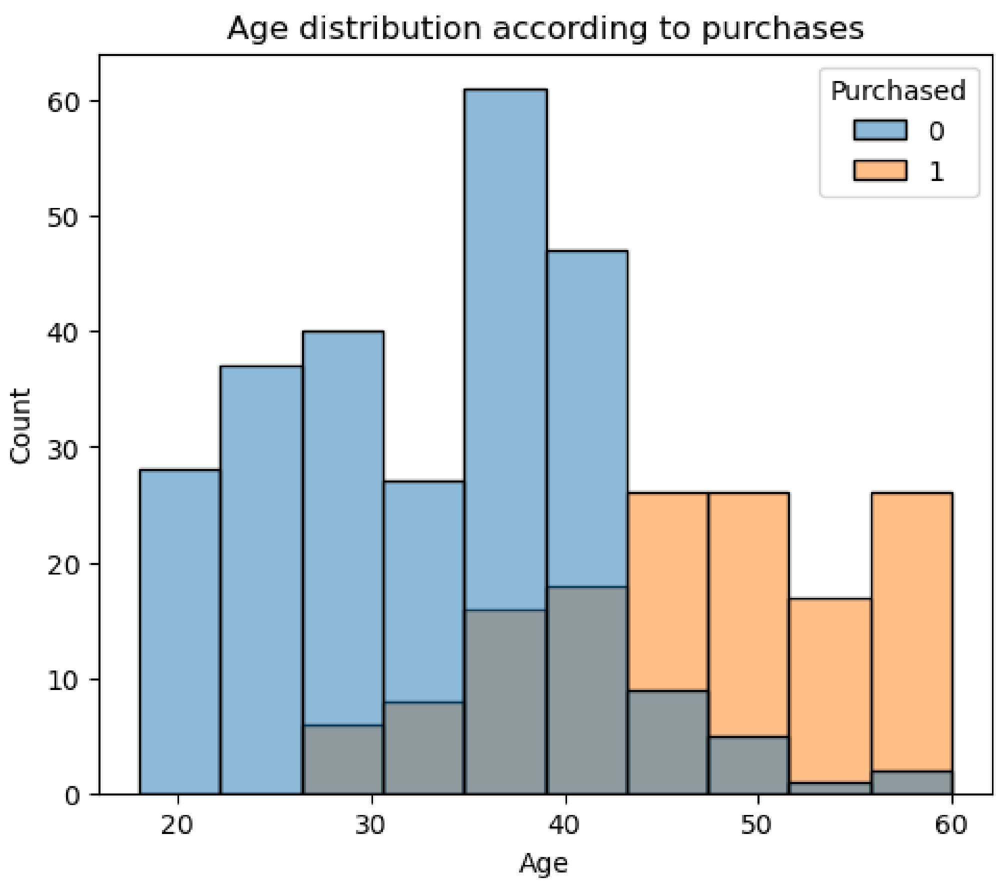
Age

reference

reference

[97]:
```
plt.figure(figsize=(6, 5))
plt.title("Age distribution according to purchases")
sns.histplot(x = "Age", hue = "Purchased", data = df)
```

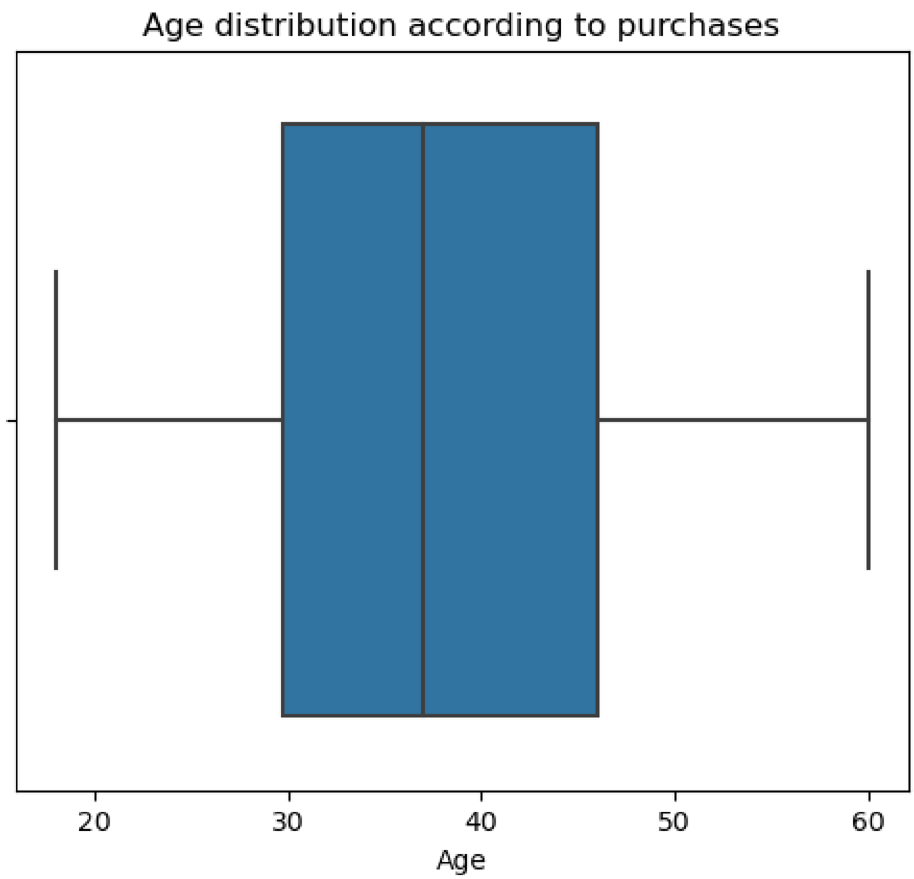
[97]:
```
<Axes: title={'center': 'Age distribution according to purchases'},
xlabel='Age', ylabel='Count'>
```



I see clearly that the company focuses on people over 44 (targeting), because they are the ones who bought the product.

[98]:
```
plt.figure(figsize=(6, 5))
plt.title("Age distribution according to purchases")
sns.boxplot(x = "Age", data = df)
```

[98]:
```
<Axes: title={'center': 'Age distribution according to purchases'},
xlabel='Age'>
```

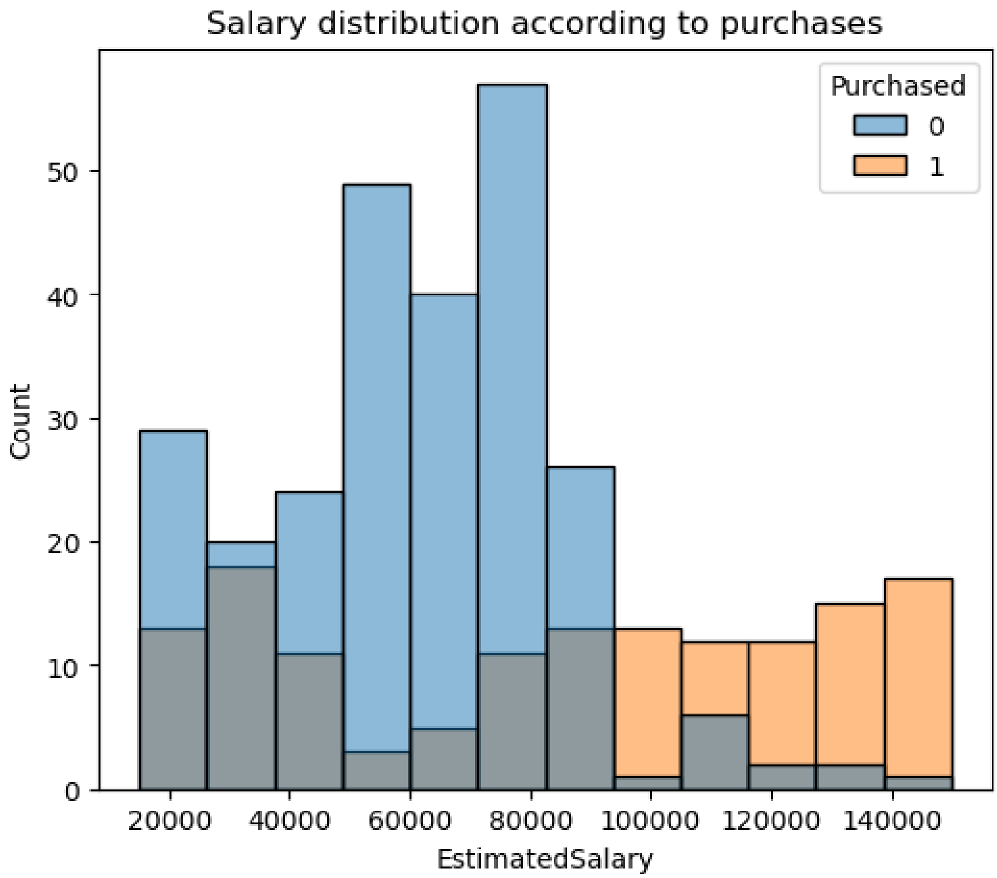


There are no outliers in this feature, but the data is a little spread out, simple scaling will be enough to contain the data variability.

EstimatedSalary

[99]:
```
plt.figure(figsize=(6,5))
plt.title("Salary distribution according to purchases")
sns.histplot(x = "EstimatedSalary", hue = "Purchased", data = df)
```

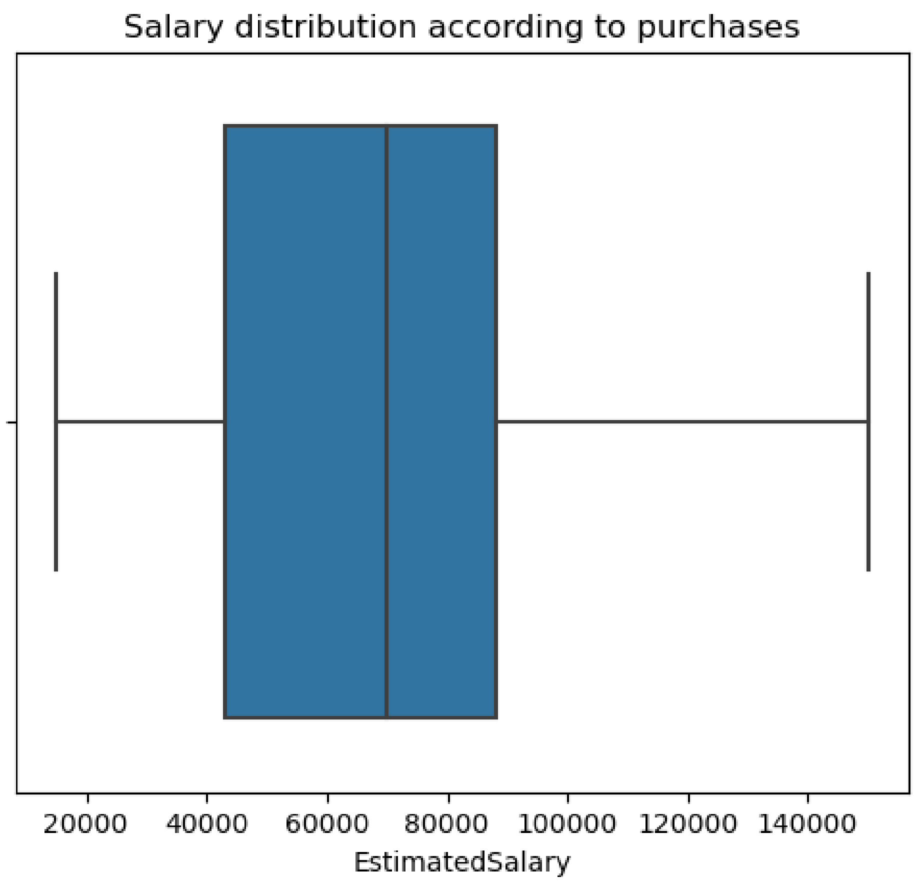
[99]: <Axes: title={'center': 'Salary distribution according to purchases'}, xlabel='EstimatedSalary', ylabel='Count'>



People who have made purchases have more than 90,000 salaries per month.

[100]:
```
plt.figure(figsize=(6, 5))
plt.title("Salary distribution according to purchases")
sns.boxplot(x = "EstimatedSalary", data = df)
```

[100]: <Axes: title={'center': 'Salary distribution according to purchases'}, xlabel='EstimatedSalary'>



There are no outliers in this feature, but the data is a little spread out, simple scaling will be enough to contain the data variability.

1.0.4 *Data preprocessing*

Delete User ID

This feature is not important for Machine Learning models

[101]:
```
data = df.drop("User ID", axis = 1)
```

[102]:
```
data.head()
```

[102]:
```
   Gender  Age  EstimatedSalary  Purchased
0    Male   19            19000          0
1    Male   35            20000          0
2  Female   26            43000          0
3  Female   27            57000          0
4    Male   19            76000          0
```

One hot encoding

[103]:
```python
def func(Gender):
    if Gender == "Female":
        Gender =1
    else :
        Gender = 0
    return Gender

data["Gender"] = data["Gender"].apply(func)
```

[104]:
```python
data["Gender"].value_counts()
```

[104]:
```
1    204
0    196
Name: Gender, dtype: int64
```

another way of doing

other ways to do

Or simply

df["Gender"] = df["Gender"].apply(lambda x : 1 if x == 'Female' else 0)

df["Gender"] = df.apply(lambda x : 1 if x["Gender"]== 'Female' else 0, axis =1)

df["Gender"] = df["Gender"].map({"Female": 1, "Male" :0})

Data set split

reference

reference

[105]:
```python
X = data.drop("Purchased", axis = 1)
y  = data["Purchased"]
```

[106]:
```python
from sklearn.model_selection import train_test_split
```

[107]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=101)
```

[108]:
```python
X_train.shape
```

[108]:
```
(280, 3)
```

[109]:
```python
X_test.shape
```

[109]:
```
(120, 3)
```

Data Scaling

reference

reference

[110]:
```
from sklearn.preprocessing import MinMaxScaler
```

[111]:
```
scaler = MinMaxScaler()
```

[112]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

1.0.5 *Create Models*

[113]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

Logistic Regression reference

reference

[114]:
```
# Create model
lr_model = LogisticRegression(random_state=0)
```

[115]:
```
# Fit the model
lr_model.fit(X_train, y_train.values)
```

[115]:
```
LogisticRegression(random_state=0)
```

[116]:
```
predict_prob = lr_model.predict_log_proba(X_test)
```

Evaluate model

1. ***Prediction***

You can get the actual predictions, based on the **probability matrix**, with **.predict()**

[117]:
```
lr_predictions = lr_model.predict(X_test)
```

2. ***Accuracy (fraction of correct predictions)***

Correct predictions / total number of data points

[118]:
```
# Accuracy
round(lr_model.score(X_test, y_test), 2)
```

[118]:
```
0.84
```

3. ***Classification report***

The classification_report() function allows to print the classification metrics of our model

reference

[119]:
```
print(classification_report(y_test, lr_predictions))
```

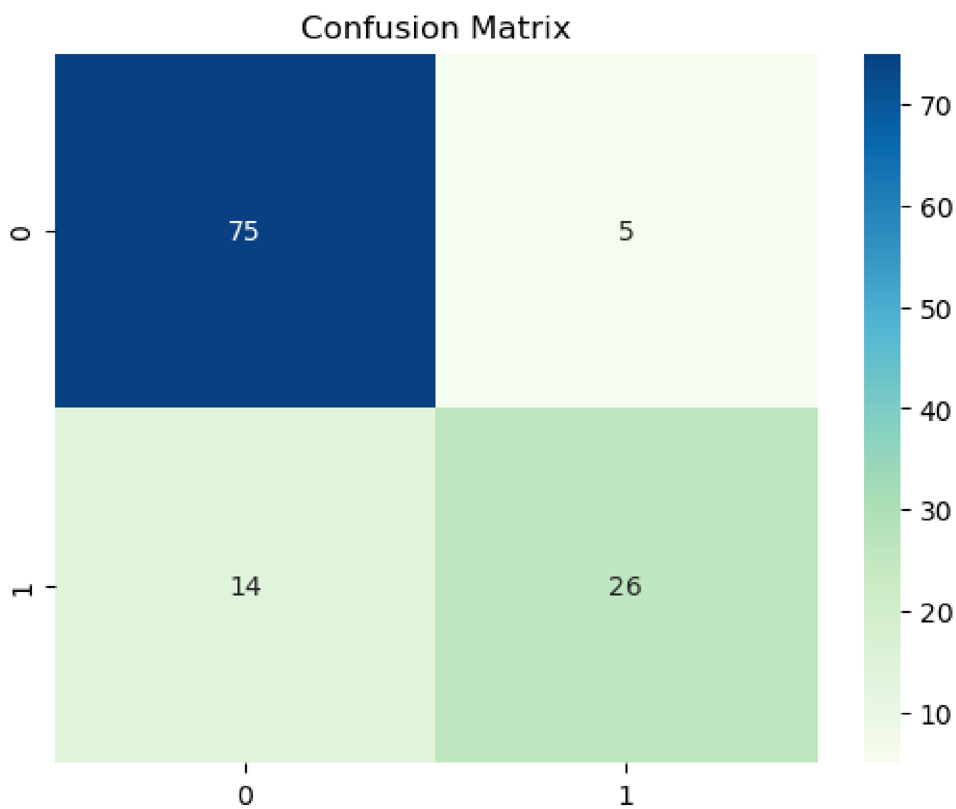
```
              precision    recall  f1-score   support

           0       0.84      0.94      0.89        80
           1       0.84      0.65      0.73        40

    accuracy                           0.84       120
   macro avg       0.84      0.79      0.81       120
weighted avg       0.84      0.84      0.84       120
```

3. ***Confusion Matrix***

reference

[120]:
```
sns.heatmap(confusion_matrix(y_test, lr_predictions), cmap='GnBu', annot = True)
plt.title("Confusion Matrix")
```

[120]:
```
Text(0.5, 1.0, 'Confusion Matrix')
```



1.0.6 *Support Vector Machine*

reference

reference

[121]:
```
svm_model = SVC(random_state=0)
```

[122]:
```
svm_model.fit(X_train, y_train.values)
```

[122]:
```
SVC(random_state=0)
```

Evaluate Model

1. ***Prediction***

[123]:
```
svm_prediction = svm_model.predict(X_test)
```

2. ***Classification report***

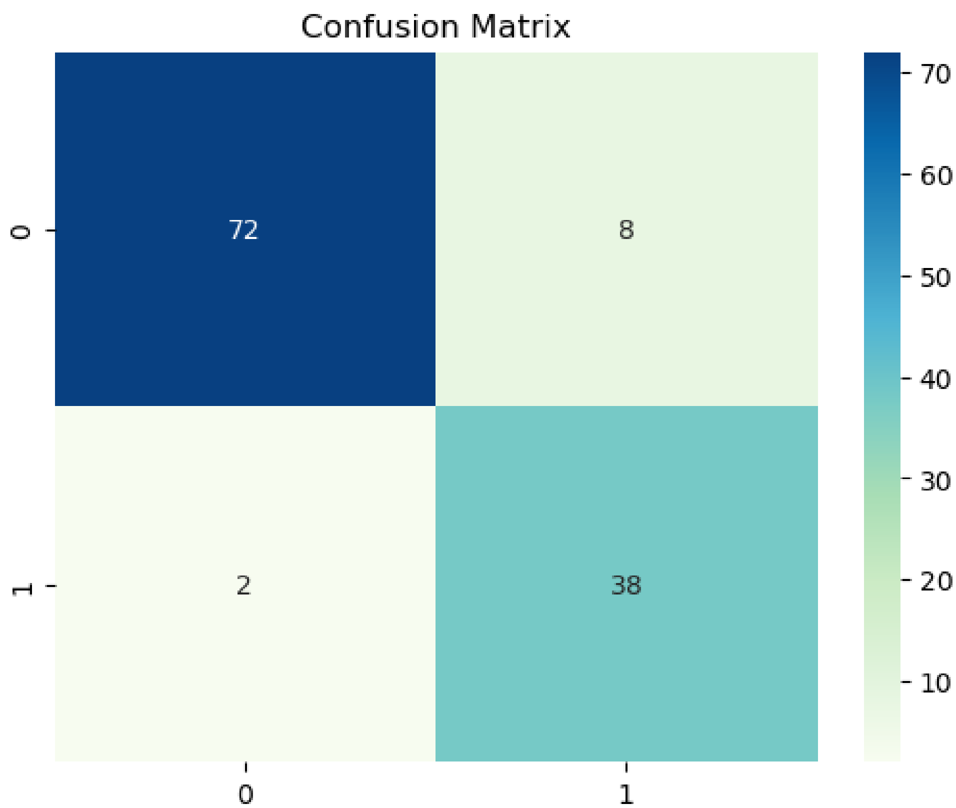
[124]:
```
print(classification_report(y_test, svm_prediction))
```

```
              precision    recall  f1-score   support

           0       0.97      0.90      0.94        80
           1       0.83      0.95      0.88        40

    accuracy                           0.92       120
   macro avg       0.90      0.93      0.91       120
weighted avg       0.92      0.92      0.92       120
```

3. Confusion Matrix

[125]:
```
sns.heatmap(confusion_matrix(y_test, svm_prediction),cmap='GnBu', annot = True)
plt.title("Confusion Matrix")
```

[125]:
```
Text(0.5, 1.0, 'Confusion Matrix')
```



1.0.7 *k-Nearest Neighbours Classifier (KNN)*

reference

reference

[126]:
```
knn_model = KNeighborsClassifier()
```

[127]:
```
knn_model.fit(X_train, y_train.values)
```

[127]:
```
KNeighborsClassifier()
```

1. ***Prediction***

[128]:
```
knn_predictions = knn_model.predict(X_test)
```

2. ***Accuracy***

[129]:
```
round(knn_model.score(X_test, y_test), 2)
```

[129]:
```
0.94
```

3. ***Classification report***

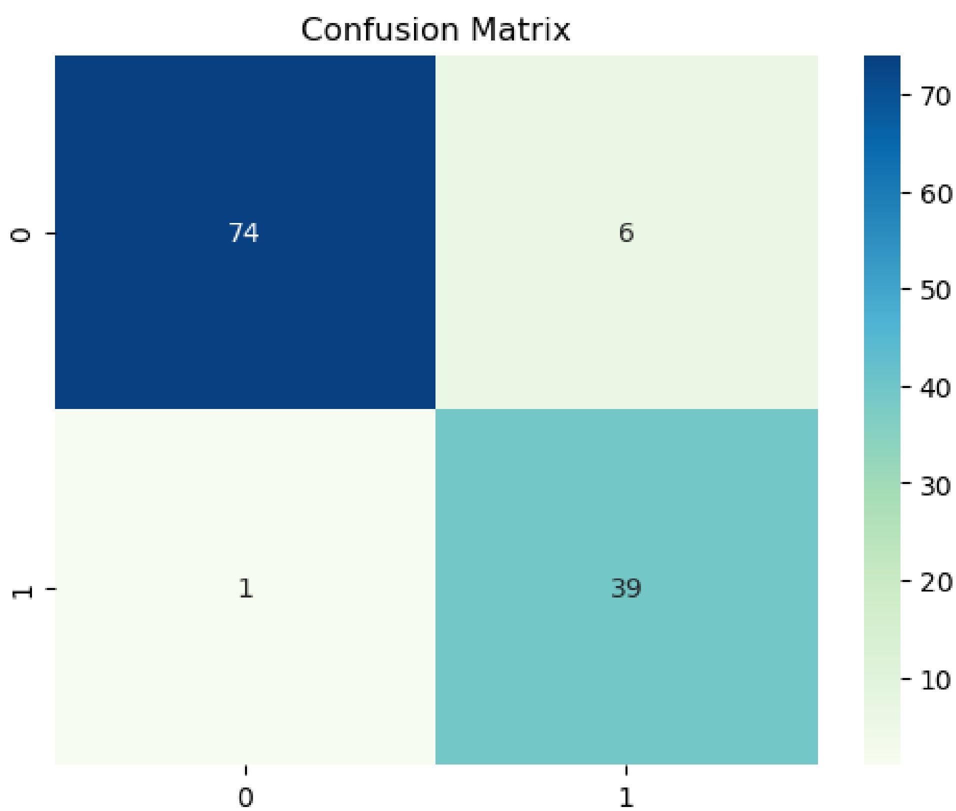
[130]:
```
print(classification_report(y_test, knn_predictions))
```

```
              precision    recall  f1-score   support

           0       0.99      0.93      0.95        80
           1       0.87      0.97      0.92        40

    accuracy                           0.94       120
   macro avg       0.93      0.95      0.94       120
weighted avg       0.95      0.94      0.94       120
```

4. ***Confusion Matrix***

[131]:
```
sns.heatmap(confusion_matrix(y_test, knn_predictions), cmap='GnBu', annot =␣
 ↪True)
plt.title("Confusion Matrix")
```

[131]: Text(0.5, 1.0, 'Confusion Matrix')



1.0.8 *Random Forest Classifier*

reference

reference

[132]:
```
rfc_model = RandomForestClassifier()
```

[133]:
```
rfc_model.fit(X_train, y_train.values)
```

[133]:
```
RandomForestClassifier()
```

1. ***Prediction***

[134]:
```
rfc_prediction = rfc_model.predict(X_test)
```

2. ***Acurracy***

[135]:
```
round(rfc_model.score(X_test, y_test), 2)
```

[135]:
```
0.92
```

3. ***Classification report***

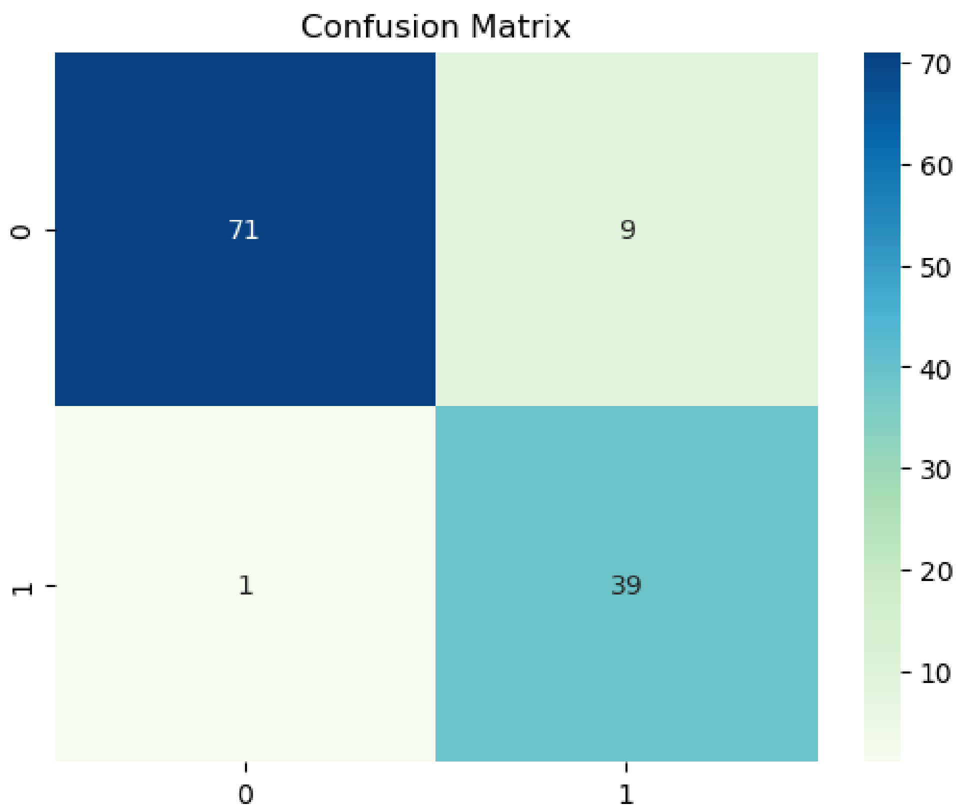
[136]:
```
print(classification_report(y_test, rfc_prediction))
```

```
              precision    recall  f1-score   support

           0       0.99      0.89      0.93        80
           1       0.81      0.97      0.89        40

    accuracy                           0.92       120
   macro avg       0.90      0.93      0.91       120
weighted avg       0.93      0.92      0.92       120
```

4. ***Confusion Matrix***

[137]:
```
sns.heatmap(confusion_matrix(y_test, rfc_prediction), cmap='GnBu', annot =True)
plt.title("Confusion Matrix")
```

[137]:
```
Text(0.5, 1.0, 'Confusion Matrix')
```



1.0.9 *AdaBoost Classifier*

reference

reference

[138]:
```
abc_model = AdaBoostClassifier()
```

[139]:
```
abc_model.fit(X_train, y_train.values)
```

[139]: AdaBoostClassifier()

1. ***Prediction***

[140]:
```
abc_predictions = abc_model.predict(X_test)
```

2. ***Accuracy***

[141]:
```
abc_model.score(X_test, y_test.values)
```

[141]: 0.925

2. ***Classification report***

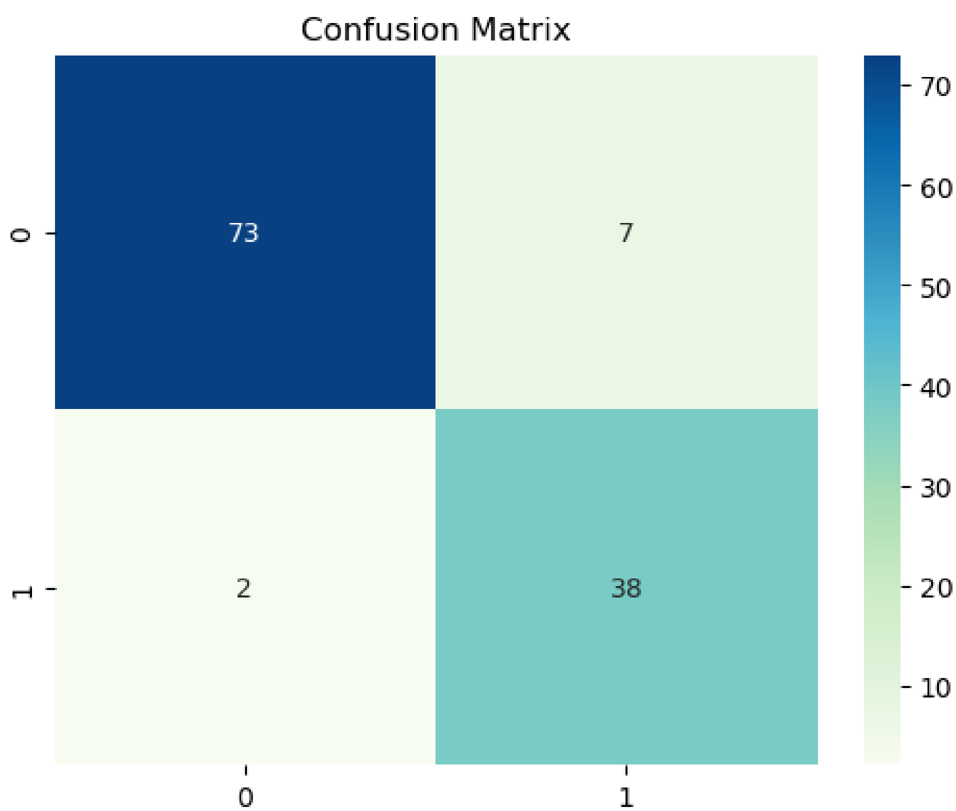
[142]:
```
print(classification_report(y_test, abc_predictions))
```

```
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        80
           1       0.84      0.95      0.89        40

    accuracy                           0.93       120
   macro avg       0.91      0.93      0.92       120
weighted avg       0.93      0.93      0.93       120
```

3. ***Confusion Matrix***

[143]:
```
sns.heatmap(confusion_matrix(y_test, abc_predictions), cmap='GnBu', annot =␣
↪True)
plt.title("Confusion Matrix")
```

[143]: Text(0.5, 1.0, 'Confusion Matrix')



1.0.10 *Neural network artificial*

reference

reference

1. ***Create model***

[144]:
```
nna_model = Sequential()
```

Dense layers

Dropout Layers

[145]:
```
nna_model.add(Dense(units=30,activation='relu'))
nna_model.add(Dropout(0.5))
nna_model.add(Dense(units=15,activation='relu'))
nna_model.add(Dropout(0.5))
nna_model.add(Dense(units=1,activation='sigmoid'))
```

losses

adam

optimizer

[146]:
```
nna_model.compile(loss='binary_crossentropy', optimizer='adam')
```

2. ***Early Stoping***

reference

[147]:
```
early_stop = EarlyStopping(monitor='val_loss', patience = 2)
```

3. ***Fit Model***

[148]:
```
nna_model.fit(x=X_train,
          y=y_train,
          epochs=200,
          validation_data=(X_test, y_test), verbose=1,
          callbacks=[early_stop]
          )
```

```
Epoch 1/200
9/9 [==============================] - 2s 53ms/step - loss: 0.7315 - val_loss:
0.6900
Epoch 2/200
9/9 [==============================] - 0s 10ms/step - loss: 0.7132 - val_loss:
0.6860
Epoch 3/200
9/9 [==============================] - 0s 11ms/step - loss: 0.6911 - val_loss:
0.6836
Epoch 4/200
9/9 [==============================] - 0s 12ms/step - loss: 0.6918 - val_loss:
```

```
0.6801
Epoch 5/200
9/9 [==============================] - 0s 12ms/step - loss: 0.6937 - val_loss:
0.6777
Epoch 6/200
9/9 [==============================] - 0s 32ms/step - loss: 0.6927 - val_loss:
0.6750
Epoch 7/200
9/9 [==============================] - 0s 20ms/step - loss: 0.6778 - val_loss:
0.6719
Epoch 8/200
9/9 [==============================] - 0s 16ms/step - loss: 0.6906 - val_loss:
0.6679
Epoch 9/200
9/9 [==============================] - 0s 14ms/step - loss: 0.6688 - val_loss:
0.6621
Epoch 10/200
9/9 [==============================] - 0s 10ms/step - loss: 0.6780 - val_loss:
0.6559
Epoch 11/200
9/9 [==============================] - 0s 10ms/step - loss: 0.6724 - val_loss:
0.6492
Epoch 12/200
9/9 [==============================] - 0s 10ms/step - loss: 0.6610 - val_loss:
0.6417
Epoch 13/200
9/9 [==============================] - 0s 10ms/step - loss: 0.6619 - val_loss:
0.6361
Epoch 14/200
9/9 [==============================] - 0s 10ms/step - loss: 0.6571 - val_loss:
0.6302
Epoch 15/200
9/9 [==============================] - 0s 10ms/step - loss: 0.6541 - val_loss:
0.6233
Epoch 16/200
9/9 [==============================] - 0s 9ms/step - loss: 0.6491 - val_loss:
0.6131
Epoch 17/200
9/9 [==============================] - 0s 10ms/step - loss: 0.6534 - val_loss:
0.6034
Epoch 18/200
9/9 [==============================] - 0s 9ms/step - loss: 0.6359 - val_loss:
0.5928
Epoch 19/200
9/9 [==============================] - 0s 25ms/step - loss: 0.6223 - val_loss:
0.5827
Epoch 20/200
9/9 [==============================] - 0s 17ms/step - loss: 0.6351 - val_loss:
```

```
0.5735
Epoch 21/200
9/9 [==============================] - 0s 16ms/step - loss: 0.6187 - val_loss:
0.5638
Epoch 22/200
9/9 [==============================] - 0s 12ms/step - loss: 0.6077 - val_loss:
0.5574
Epoch 23/200
9/9 [==============================] - 0s 11ms/step - loss: 0.6179 - val_loss:
0.5493
Epoch 24/200
9/9 [==============================] - 0s 10ms/step - loss: 0.5909 - val_loss:
0.5413
Epoch 25/200
9/9 [==============================] - 0s 10ms/step - loss: 0.5797 - val_loss:
0.5296
Epoch 26/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5640 - val_loss:
0.5166
Epoch 27/200
9/9 [==============================] - 0s 10ms/step - loss: 0.5971 - val_loss:
0.5078
Epoch 28/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5633 - val_loss:
0.5021
Epoch 29/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5712 - val_loss:
0.4957
Epoch 30/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5604 - val_loss:
0.4913
Epoch 31/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5464 - val_loss:
0.4855
Epoch 32/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5456 - val_loss:
0.4797
Epoch 33/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5261 - val_loss:
0.4702
Epoch 34/200
9/9 [==============================] - 0s 10ms/step - loss: 0.5233 - val_loss:
0.4606
Epoch 35/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5416 - val_loss:
0.4556
Epoch 36/200
9/9 [==============================] - 0s 10ms/step - loss: 0.5229 - val_loss:
```

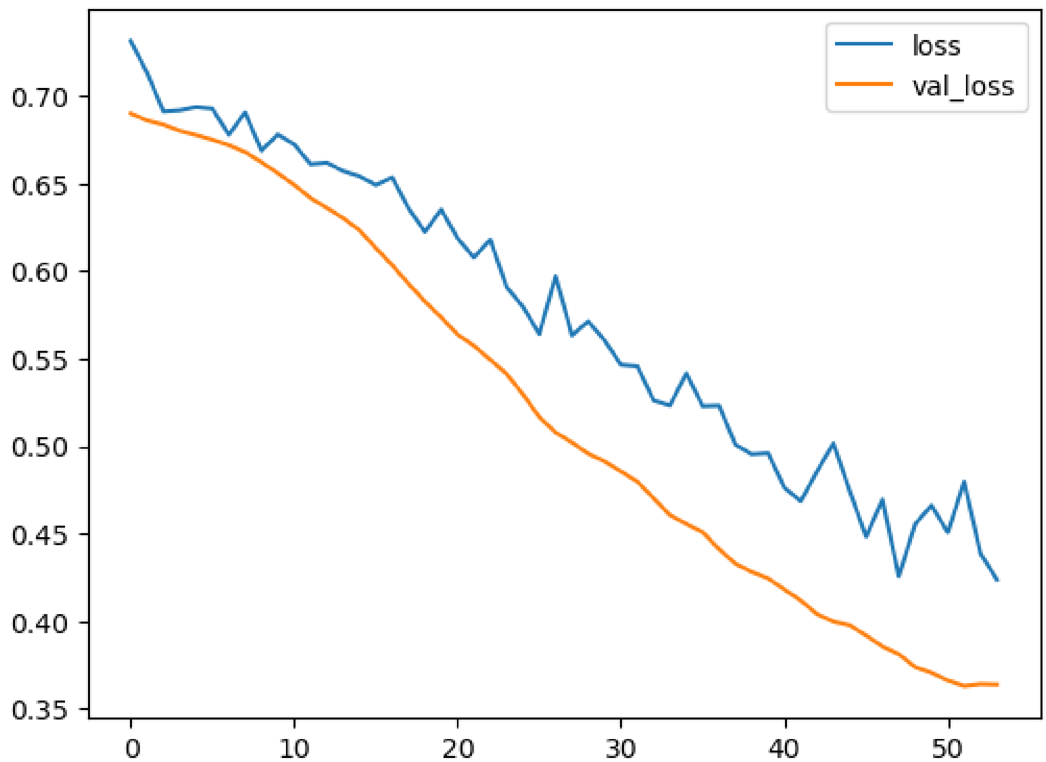
```
0.4509
Epoch 37/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5232 - val_loss:
0.4412
Epoch 38/200
9/9 [==============================] - 0s 10ms/step - loss: 0.5009 - val_loss:
0.4329
Epoch 39/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4954 - val_loss:
0.4282
Epoch 40/200
9/9 [==============================] - 0s 10ms/step - loss: 0.4960 - val_loss:
0.4245
Epoch 41/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4763 - val_loss:
0.4182
Epoch 42/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4686 - val_loss:
0.4119
Epoch 43/200
9/9 [==============================] - 0s 10ms/step - loss: 0.4856 - val_loss:
0.4041
Epoch 44/200
9/9 [==============================] - 0s 9ms/step - loss: 0.5017 - val_loss:
0.3998
Epoch 45/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4743 - val_loss:
0.3978
Epoch 46/200
9/9 [==============================] - 0s 10ms/step - loss: 0.4483 - val_loss:
0.3919
Epoch 47/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4696 - val_loss:
0.3855
Epoch 48/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4257 - val_loss:
0.3812
Epoch 49/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4555 - val_loss:
0.3738
Epoch 50/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4662 - val_loss:
0.3706
Epoch 51/200
9/9 [==============================] - 0s 10ms/step - loss: 0.4507 - val_loss:
0.3662
Epoch 52/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4797 - val_loss:
```

```
0.3631
Epoch 53/200
9/9 [==============================] - 0s 10ms/step - loss: 0.4388 - val_loss:
0.3641
Epoch 54/200
9/9 [==============================] - 0s 9ms/step - loss: 0.4238 - val_loss:
0.3639
```

[148]: `<keras.callbacks.History at 0x1b070a32a60>`

4. ***Loss and validation functions***

[149]:
```
model_loss = pd.DataFrame(nna_model.history.history)
model_loss.plot();
```



Evaluate model

[150]:
```
nna_predictions = (nna_model.predict(X_test) > 0.5).astype("int32")
```

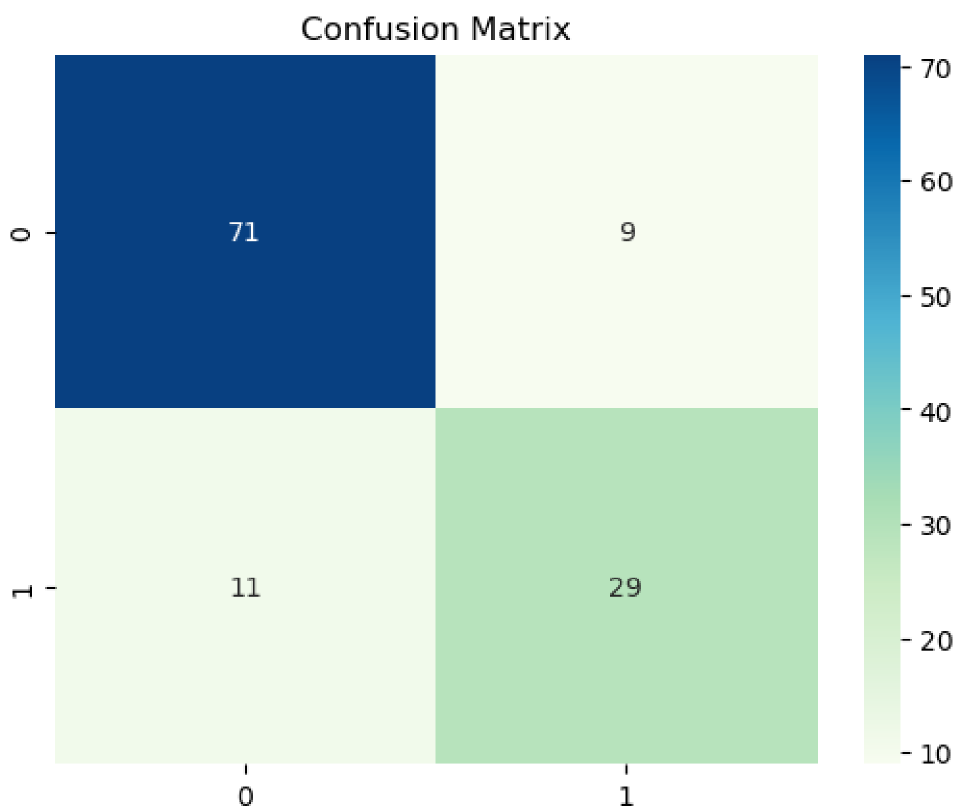
```
4/4 [==============================] - 0s 3ms/step
```

[151]:
```
# https://en.wikipedia.org/wiki/Precision_and_recall
print(classification_report(y_test,nna_predictions))
```

```
              precision    recall  f1-score   support

           0       0.87      0.89      0.88        80
           1       0.76      0.72      0.74        40

    accuracy                           0.83       120
   macro avg       0.81      0.81      0.81       120
weighted avg       0.83      0.83      0.83       120
```

[152]:
```
sns.heatmap(confusion_matrix(y_test, nna_predictions), cmap='GnBu', annot =
 ↪True)
plt.title("Confusion Matrix")
```

[152]:
```
Text(0.5, 1.0, 'Confusion Matrix')
```



1.0.11 *Model performance comparison*

[153]:
```
dict = {"Logistic Regression" : [0.84, 0.65, 0.73, 0.84], "Support Vector␣
↪Machines" : [0.83, 0.95, 0.88, 0.92], "K Neighbors Classifier" : [0.87, 0.
↪97, 0.92, 0.94 ], "Random Forest Classifier" : [0.81, 0.97, 0.89, 0.92],␣
↪"AdaBoost Classifier": [0.84, 0.95, 0.89, 0.93], "Neural network artificial"␣
↪: [0.82, 0.93, 0.87, 0.91]}
```

[154]:
```
pd.DataFrame(dict, index = ["precision", "recall", "f1-score", "accuracy"])
```

[154]:
```
           Logistic Regression  Support Vector Machines  \
precision                 0.84                     0.83
recall                    0.65                     0.95
f1-score                  0.73                     0.88
accuracy                  0.84                     0.92

           K Neighbors Classifier  Random Forest Classifier  \
precision                    0.87                      0.81
recall                       0.97                      0.97
f1-score                     0.92                      0.89
accuracy                     0.94                      0.92

           AdaBoost Classifier  Neural network artificial
precision                 0.84                       0.82
recall                    0.95                       0.93
f1-score                  0.89                       0.87
accuracy                  0.93                       0.91
```

We note that the knn model is the most efficient model. We will therefore save this model for future predictions on new data.

In the meantime, we will retrieve a random client from our data to test the prediction of the chosen model.

Save knn_model

reference

[159]:
```
import pickle

filename = open('knn_model_save.pkl', 'wb')
pickle.dump(knn_model, filename)

filename.close()
```

1.0.12 *Predict whether a randomly selected person in the dataset buys the product or not*

We use the model that has the best performance to make this prediction (knn_model)

Load knn_model

[172]:
```
import pprint, pickle

knn_model_file = open('knn_model_save.pkl', 'rb')

loaded_knn_model = pickle.load(knn_model_file)
pprint.pprint(loaded_knn_model)

knn_model_file.close()
```

```
KNeighborsClassifier()
```

Looking for a customer in the dataset

[174]:
```
import random
random.seed(101)
random_ind = random.randint(0,len(data))

single_customer = data.drop('Purchased',axis=1).iloc[random_ind]
single_customer
```

[174]:
```
Gender                  1
Age                    43
EstimatedSalary    112000
Name: 297, dtype: int64
```

Scaling row

[175]:
```
single_customer = scaler.transform(single_customer.values.reshape(-1, 3))
```

```
C:\Users\HP\anaconda3\envs\DeepLearning\lib\site-packages\sklearn\base.py:439:
UserWarning: X does not have valid feature names, but MinMaxScaler was fitted
with feature names
  warnings.warn(
```

Predict which customer it is

[176]:
```
loaded_knn_model.predict(single_customer)
```

[176]:
```
array([1], dtype=int64)
```

This is a customer who purchased the product

Check prediction

[177]:
```
data.iloc[random_ind]['Purchased']
```

[177]:
```
1
```

Our model indeed predicts that this chosen person buys the product, which also corresponds to reality.

1.1 *Conclusion*

We found the most efficient model (KNN) that will allow us to make future predictions on new data that our model did not know. The above prediction on a sample of customers gave conclusive results. This KNN model is ready for deployment. It was saved, it was enough to download it to make new predictions.

1.2 *References*

https://www.kaggle.com/datasets/rakeshrau/social-network-ads

https://copyassignment.com/customer-behaviour-analysis-machine-learning-and-python/

https://rstudio-pubs-static.s3.amazonaws.com/346115_dfb9c60faa934625938d4fe0d2ac1364.html

https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/

https://towardsdatascience.com/understanding-random-forest-58381e0602d2

https://aws.amazon.com/fr/what-is/neural-network/

https://realpython.com/python-ai-neural-network/