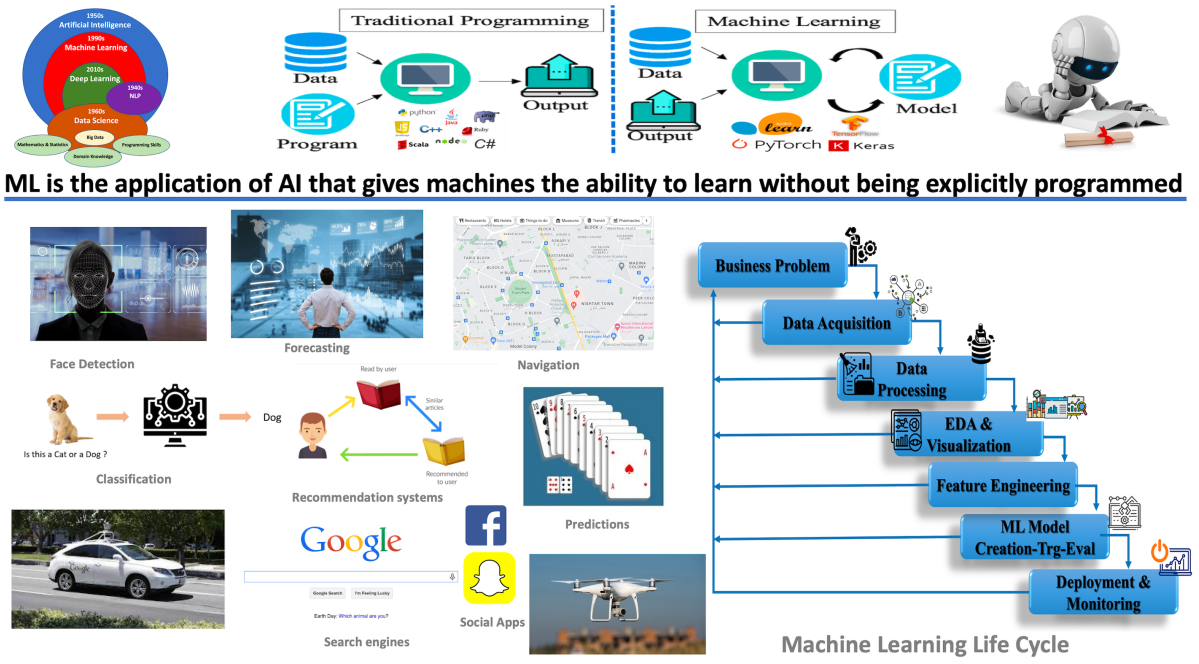
Department of Data Science

Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.24 (K-Nearest Neighbors)

Open in Colab (https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)



Learning agenda of this notebook

- Theory and Intuition Behind K-Nearest Neighbors Algorithm
  - The Big Picture
  - Geometric Intuition and Working of KNN Algorithm for Classification
  - Geometric Intuition and Working of KNN Algorithm for Regression
  - KNN Algorithm and it's Complexity
  - Distance/Similarity Measures for KNN Algorithm
    - Manhatten Distance

    - Euclidean Distance
    - Chebyshev Distance
    - Minkowski distance
    - Properties/Constraints on Distance Metrics
    - Computing Distance Metrics from Scratch using `numPy` library
    - Computing Distance Metrics using `SciPy` library

- **Example 1:** Hands on Example (Binary Classification `without using Scikit-Learn`)

- **Example 2:** Hands on Example (Binary Classification `using Scikit-Learn`)
    - Load dataset
    - Perform EDA
    - Train-Test split
    - Data Preprocessing and Feature Engineering
    - Model Training, Evaluation and Prediction
    - Hyperparameter Tuning and Choosing Best Value of `K`
        - Mannually
        - Using GridSearchCV
        - Using RandomizedSearchCV
    - Decision Boundry and Impact of Different Values of K (Underfitting and Overfitting)

- Limitations of KNN
- Enhancements of KNN
- Tasks To Do

1. Theory and Intuition Behind K-Nearest Neighbors Algorithm

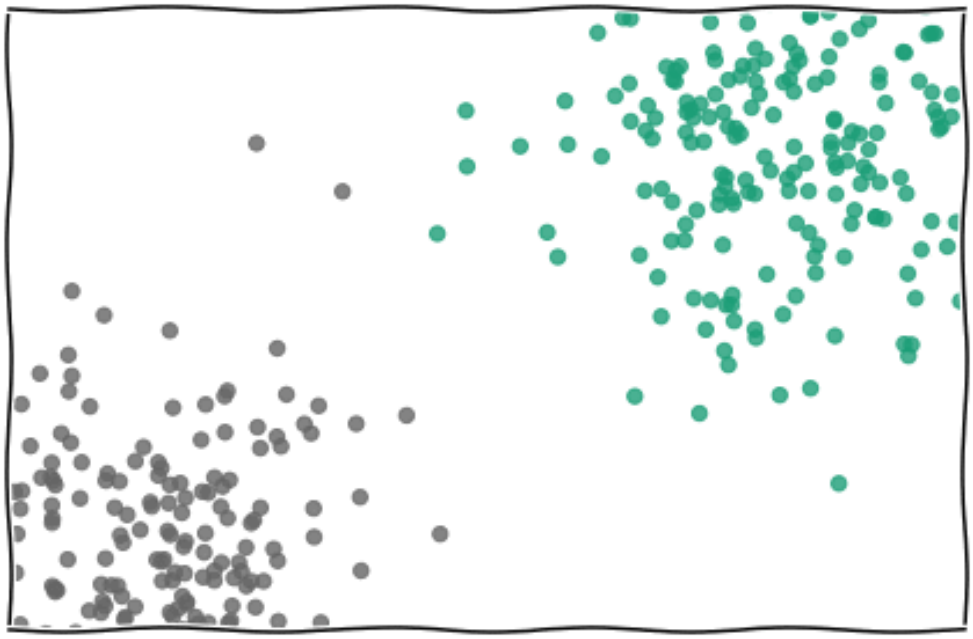
a. The Big Picture

> "You are the average of the five people you spend the most time with"
>
> **-Jim Rohn-**

> In **KNN** predictions are made for a new instance (x) by searching through the entire training set for **K** most similar cases (neighbors) and summarizing the output variable for those K cases.

- **Applications of KNN Algorithm**
    - `Recommendation systems:` KNN can be used to make personalized recommendations to users, based on their similarity to other users or items.
    - `Image recognition:` The algorithm compares a new image to the closest training examples in a feature space, based on their pixel values.
    - `Medical diagnosis:` KNN can be used to diagnose medical conditions based on the symptoms and test results of patients, by comparing them to the closest examples

in a



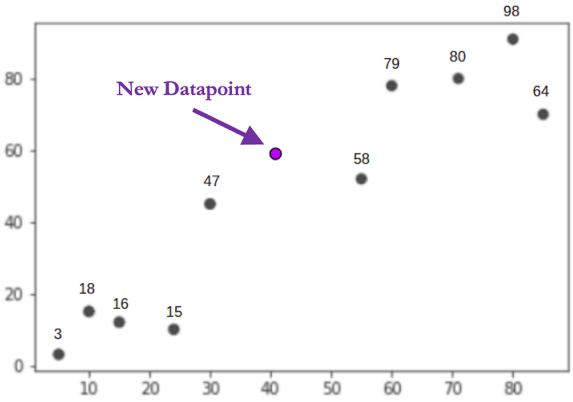
database of known cases.

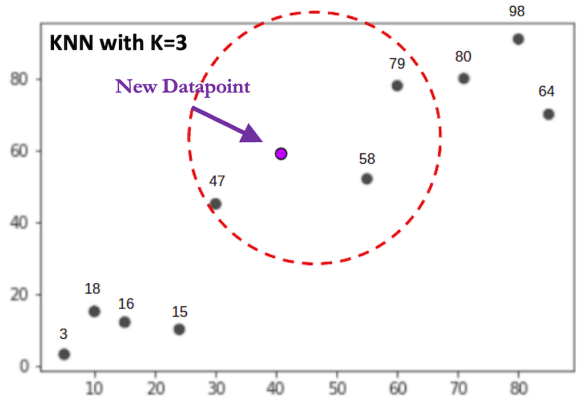
- `Customer Segmentation:` KNN can be used to identify customer segments by grouping them based on their similarity in a feature space defined by their demographic, purchase history or behavior data.
- `Anomaly detection:` KNN can be used to identify outliers or anomalies in a dataset, by identifying the examples that are far away from their nearest neighbors.
- `Fraud detection:` KNN can be used to identify fraudulent transactions by identifying the examples that are far away from their nearest neighbors in a feature space defined by the transaction details.
- `Quality Control:` KNN can be used to identify defective items in a manufacturing process by comparing new items to the closest examples in a feature space defined by the measurements.

b. Geometric Intuition and Working of KNN Algorithm for

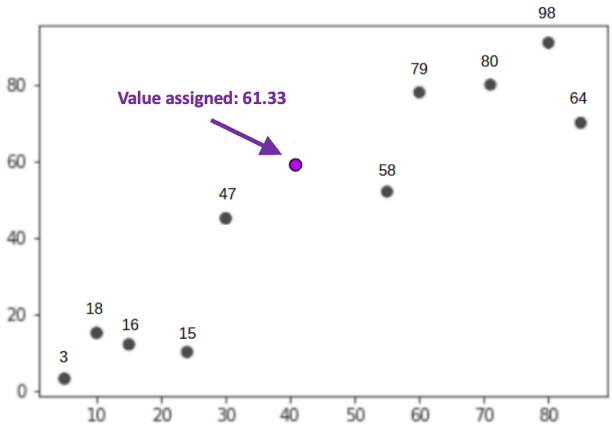
Classification

c. Geometric Intuition and Working of KNN Algorithm for Regression





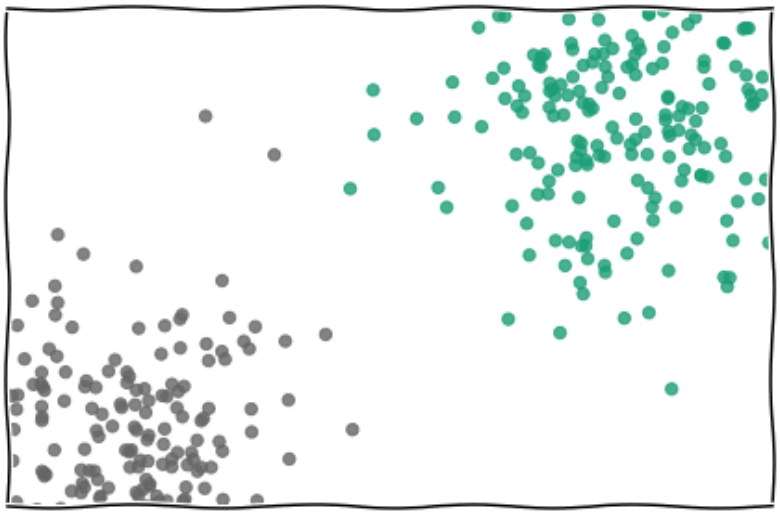
NewPoint = (47*weight +58*weight +79*weight)/3
NewPoint = (47*1+58*1+79*1)/3
NewPoint = 61.3



The weight can be $\frac{1}{d}$, where `d` is the distance from the query point

d. KNN Algorithm and it's Complexity

- **Input:** Training data, Query/Test point, Value of K, Distance metric
- **Algorithm:**
  - **Step 1:** Calculate the distance between test point and each row of training dataset (For `n` data points there will be `n` distances) $O(nm)$



  - **Step 2:** Choose `K` observations from the training data set, which are nearest to the query/test point $O(nk)$

- **Step 3:** Assign label to query/test point. For classification, take the `majority vote` of the output label of the `K` data points. For regression, take mean or `weighted average` of the `K` datapoints. $O(k)$

`Total Time Complexity:`
$O(nm + nk + k) = O(nm)$

Points to Ponder

- **KNN is Lazy Learning Algorithm:** Do not learn from the training set (no training phase), rather stores the data and at the time of classification/prediction, it performs an action on the dataset.
- **KNN is Non-Parametric Algorithm:** There are no model parameters to be learned, and the algorithm makes no assumptions on the underlying data distribution.
- **Distance/Similarity Metrics:** For the algorithm to work best on a particular dataset we need to choose the most appropriate distance metric accordingly. (`Manhatten` , `Euclidean` , `Chebyshev`)
- **Value of `K` :** The only learning required for KNN is to find out the optimal `K` value using Cross-Validation or GridSearch.
- **KNN is Memory Hungry:** KNN is memory hungry as it tries to keep all the training dataset during the testing phase. This is unlike Linear and Logistic Regression Models, which learn model parameters during the training phase and use them during the testing/prediction phase. So KNN needs no training time, however, whenever, the algorithm is put into action, it needs to search the entire dataset to find K nearest neighbours. So the space complexity of this algorithm is also `O(nm)`

e. Distance/Similarity Measures for KNN Algorithm

- There exist a variety of distance/similarity measures that can be used to check the similarity between two data points. The selection of distance measure for KNN algorithm depends on the problem at hand and how much noisy your data is. The better that metric computes similarity, the better is the result of the classifier.
- [Effects of Distance Measure Choice on KNN Classifier Performance - A Review (https://arxiv.org/pdf/1708.04321.pdf)]

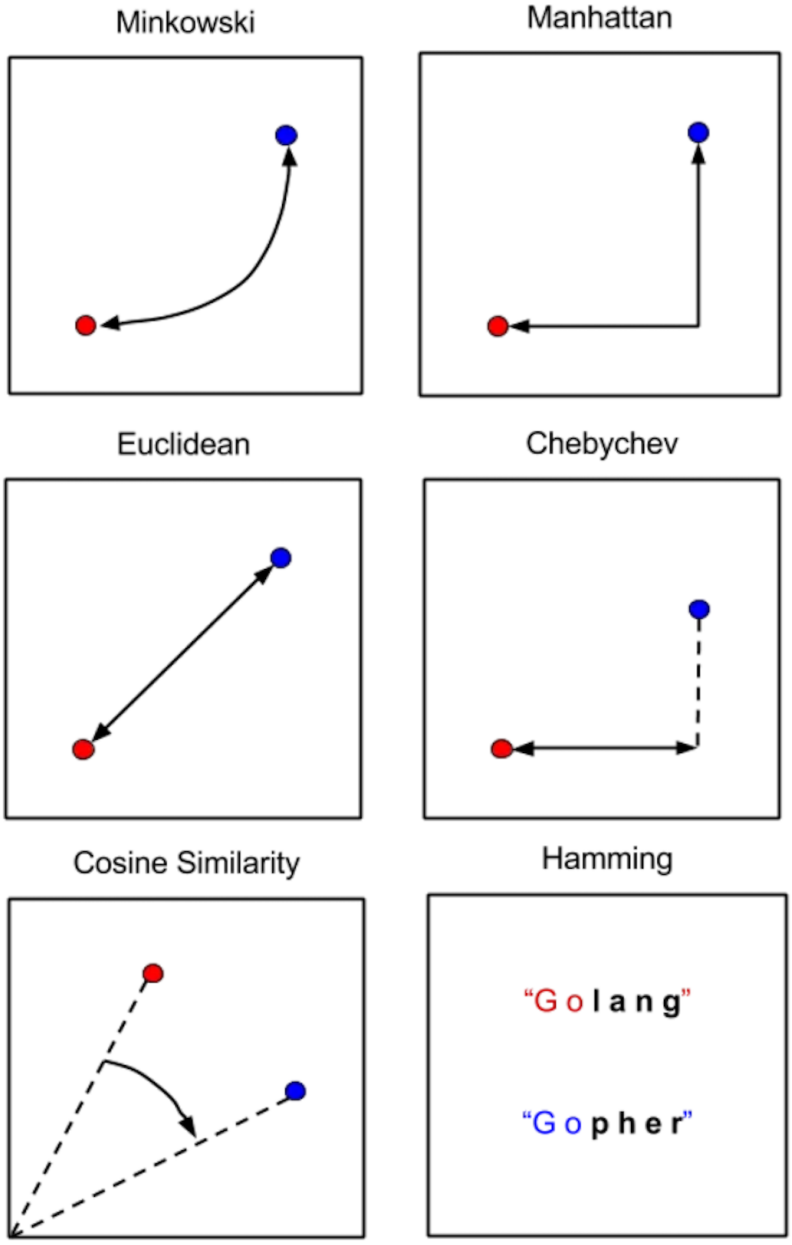
- **Minkowski Distance** is the generalized form of `Manhattan` , `Euclidean` and `Chebyshev` distances. The Minkowski distance between two points having `m` dimensions $P =< p_1, p_2, p_3, \cdots, p_m >$ and $Q =< q_1, q_2, q_3, \cdots, q_m >$, is defined as:

$$d_{minkowski}(p, q) = \left(\sum_{i=1}^{m} |p_i - q_i|^a\right)^{\frac{1}{a}}, a \geq 1$$

- **Manhattan Distance (L1/CityBlock/TaxiCab):**

$$d_{manhattan}(p, q) = \left(\sum_{i=1}^{m} |p_i - q_i|\right)$$

- **Euclidean Distance (L2):**



$$d_{euclidean}(p,q) = \left(\sum_{i=1}^{m} |p_i - q_i|^2\right)^{\frac{1}{2}}$$

- **Chebyshev Distance (L∞/Chessboard/Maximum):**

$$d_{chebyshev}(p,q) = \lim_{a\to\infty} \left(\sum_{i=1}^{m} |p_i - q_i|^a\right)^{\frac{1}{a}}$$
$$= max_i |p_i - q_i|$$

> Chebyshev is actually the maximum of the absolute distances between the corresponding elements of the two vectors. We use Chebyshev distance, when we are interested to know that which feature value in the two vectors is actually most affecting the distance.

- **Properties/Constraints on Distance Metrics**
  - **Non-Negativity:** $d(p, q) \geq 0$
  - **Identity of indiscernible vectors:** $d(p, q) = 0, \text{ iff } p = q$

Computing Distance Metrics from Scratch and using `SciPy` library

$$d_{manhattan}(p, q) = \left(\sum_{i=1}^{m} |p_i - q_i|\right)$$

$$d_{euclidean}(p, q) = \left(\sum_{i=1}^{m} |p_i - q_i|^2\right)^{\frac{1}{2}}$$

$$d_{chebyshev}(p, q) = \lim_{a \to \infty} \left(\sum_{i=1}^{m} |p_i - q_i|^a\right)^{\frac{1}{a}} = max_i |p_i - q_i|$$

In [1]:
```
from scipy.spatial.distance import minkowski, cityblock, euclidean, cheb

u = [5, 3, 9]
v = [4, 6, 1]
print("u: ", u)
print("v: ", v)

# Calculate City Block (Manhattan) distance using formula, cityblock() a
print('\nManhattan Distance:',sum(abs(i - j) for i, j in zip(u, v)))
print('Manhattan Distance:', cityblock(u, v, w=None))
print('Manhattan Distance:', minkowski(u, v, w=None, p=1))# p is the ord

# Calculate Euclidean distance using formula, euclidean() and minkowski(
print('\nEuclidean Distance:', (sum((i - j)**2 for i, j in zip(u, v)))**
print('Euclidean Distance:', euclidean(u, v, w=None))
print('Euclidean Distance:', minkowski(u, v, w=None, p=2))# p is the ord

# Calculate Chebyshev distance using formula, chebyshev() and minkowski(
print('\nChebyshev Distance: ', (sum((i - j)**100 for i, j in zip(u, v))
print('Chebyshev Distance: ', chebyshev(u, v, w=None))
print('Chebyshev Distance: ', minkowski(u, v, w=None, p=100))# p is the
```

```
u:  [5, 3, 9]
v:  [4, 6, 1]

Manhattan Distance: 12
Manhattan Distance: 12
Manhattan Distance: 12.0

Euclidean Distance: 8.602325267042627
Euclidean Distance: 8.602325267042627
Euclidean Distance: 8.602325267042627

Chebyshev Distance:  8.0
Chebyshev Distance:  8
Chebyshev Distance:  8.0
```

2. Hands on Example (Binary Classification w/o

Load Dataset:

In [2]:
```
import numpy as np
import pandas as pd
df = pd.read_csv("datasets/fruits_classification.csv")
df
```

Out[2]:

	mass	width	height	color_score	fruit_name	fruit_label
0	176	7.4	7.2	0.60	apple	0
1	178	7.1	7.8	0.92	apple	0
2	156	7.4	7.4	0.84	apple	0
3	140	7.3	7.1	0.87	apple	0
4	154	7.1	7.5	0.78	orange	1
5	180	7.6	8.2	0.79	orange	1
6	154	7.2	7.2	0.82	orange	1
7	118	5.9	8.0	0.72	lemon	2
8	120	6.0	8.4	0.74	lemon	2
9	118	6.1	8.1	0.70	lemon	2

New unknown fruit `having mass, width, height and color_score`

In [3]:
```
new_fruit = pd.DataFrame([[130, 7.5, 9.2, 0.5]], columns=['mass', 'width
new_fruit
```

Out[3]:

	mass	width	height	color_score
0	130	7.5	9.2	0.5

Calculate Euclidean Distance between the query fruit and all the fruits in the dataset:

In [4]:
```
def euclid_distance(x):
    a = x.to_numpy()          # first vector, which is the row of the dat
    b = new_fruit.to_numpy() # second vector, which is the query/test ve
    distance = (sum((i - j)**2 for i, j in zip(a, b[0])))**0.5
    return distance
```

In [5]:
```
df.apply(euclid_distance, axis=1)# axis=1 means apply function to each r
```

Out[5]:
```
0     46.043675
1     48.023915
2     26.064643
3     10.226774
4     24.065087
5     50.010940
6     24.087183
7     12.167514
8     10.146310
9     12.133013
dtype: float64
```

In [6]:
```
df['distance'] = df.apply(euclid_distance, axis=1)# axis=1 means apply f
df
```

Out[6]:

	mass	width	height	color_score	fruit_name	fruit_label	distance
0	176	7.4	7.2	0.60	apple	0	46.043675
1	178	7.1	7.8	0.92	apple	0	48.023915
2	156	7.4	7.4	0.84	apple	0	26.064643
3	140	7.3	7.1	0.87	apple	0	10.226774
4	154	7.1	7.5	0.78	orange	1	24.065087
5	180	7.6	8.2	0.79	orange	1	50.010940
6	154	7.2	7.2	0.82	orange	1	24.087183
7	118	5.9	8.0	0.72	lemon	2	12.167514
8	120	6.0	8.4	0.74	lemon	2	10.146310
9	118	6.1	8.1	0.70	lemon	2	12.133013

Sort the dataframe by distance column:

In [7]:
```
# sort() method is used to sort by the values along either axis.

df_sort = df.sort_values(by='distance', ascending=True, axis=0)
df_sort
```

Out[7]:

	mass	width	height	color_score	fruit_name	fruit_label	distance
8	120	6.0	8.4	0.74	lemon	2	10.146310
3	140	7.3	7.1	0.87	apple	0	10.226774
9	118	6.1	8.1	0.70	lemon	2	12.133013
7	118	5.9	8.0	0.72	lemon	2	12.167514
4	154	7.1	7.5	0.78	orange	1	24.065087
6	154	7.2	7.2	0.82	orange	1	24.087183
2	156	7.4	7.4	0.84	apple	0	26.064643
0	176	7.4	7.2	0.60	apple	0	46.043675
1	178	7.1	7.8	0.92	apple	0	48.023915
5	180	7.6	8.2	0.79	orange	1	50.010940

Assign Class Label of `K=3` nearest neighbors by majority vote:

> So, the new observation having mass=130, width=7.5, height=9.2, and color_score=0.5, is **Lemon**

3. Hands on Example (Binary Classification using Scikit-Learn)

a. Load Dataset

In [8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
df=pd.read_csv('datasets/User_Data.csv')
df
```

Out[8]:

	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
...	...	...	...	...
395	Female	46	41000	1
396	Male	51	23000	1
397	Female	50	20000	1
398	Male	36	33000	0
399	Female	49	36000	1

400 rows × 4 columns

b. Exploratory Data Analysis (EDA)

Since EDA has no real set methodolgy, the following is a short check list you might want to walk through:

1. What question(s) are you trying to solve (or prove wrong)?
2. What's missing from the data and how do you deal with it?
3. Are there any outliers and how to treat them?
4. What kind of data do you have and how do you treat different types?
5. How can you add, change or remove features to get more out of your data?
6. How your data is distributed and the correlation between different variables?

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           400 non-null    object
 1   Age              400 non-null    int64
 2   EstimatedSalary  400 non-null    int64
 3   Purchased        400 non-null    int64
dtypes: int64(3), object(1)
memory usage: 12.6+ KB
```

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

Out[10]:

	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

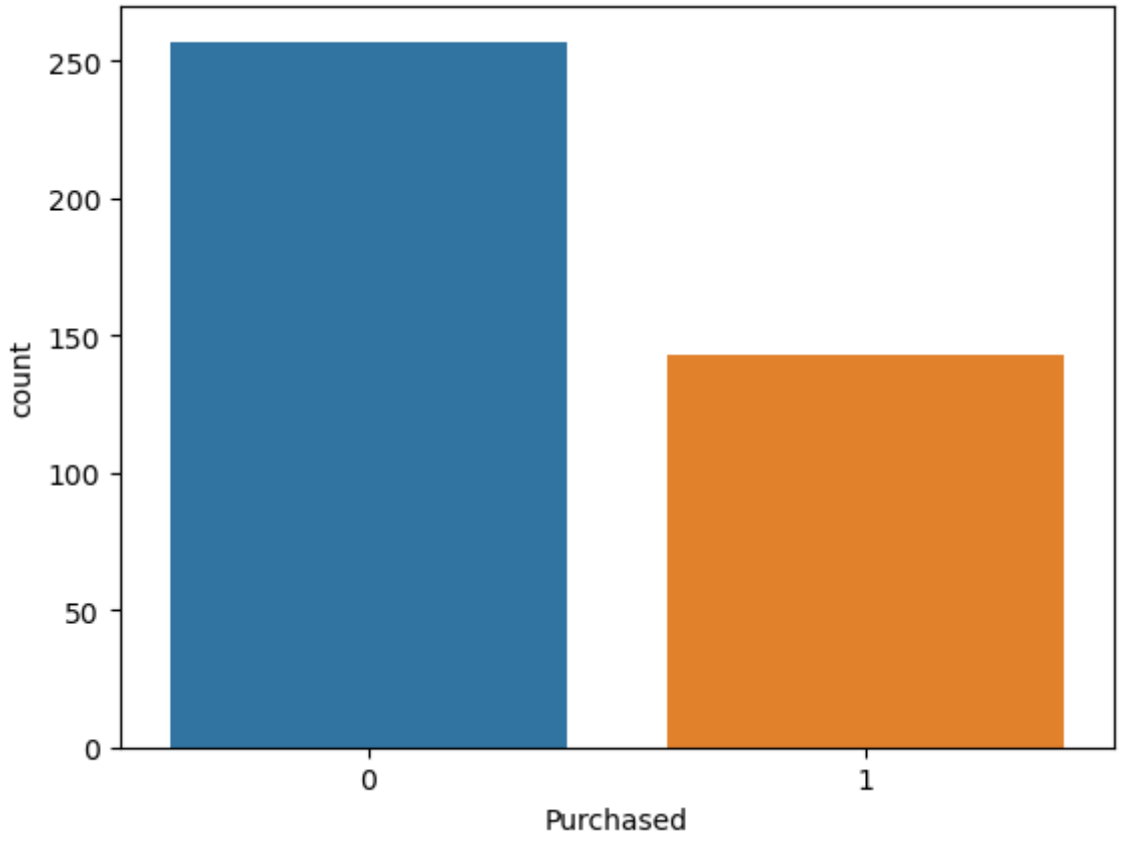
In [11]:
```
df.Gender.value_counts()
```

Out[11]:
```
Female    204
Male      196
Name: Gender, dtype: int64
```

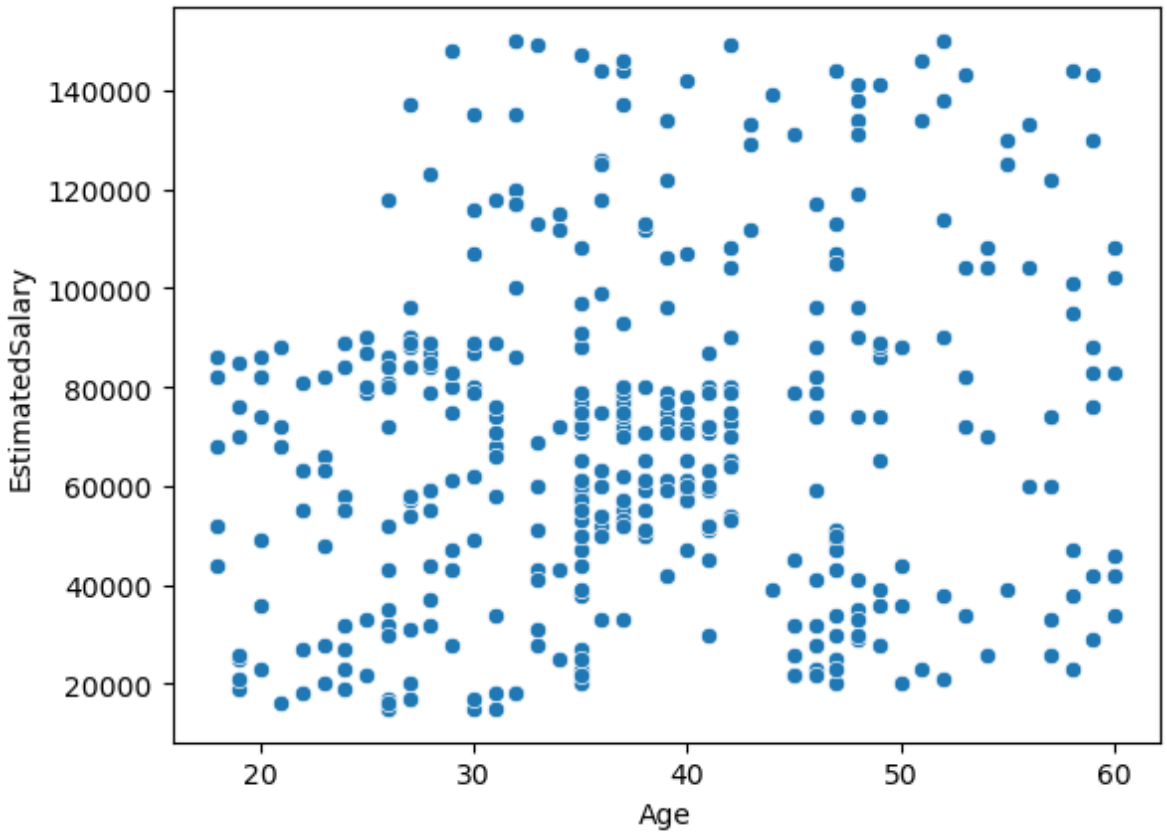
In [12]:
```
df.Purchased.value_counts()
```

Out[12]:
```
0    257
1    143
Name: Purchased, dtype: int64
```

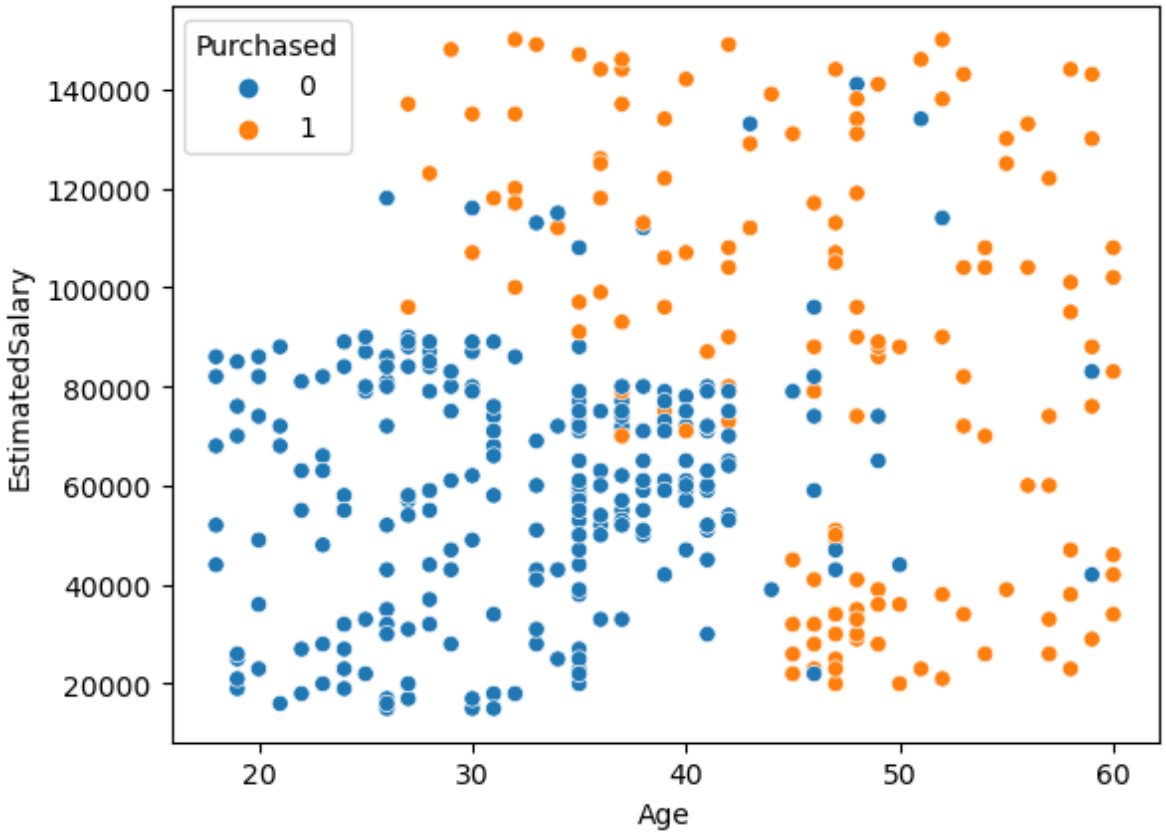
In [13]:
```
sns.countplot(x=df.Purchased);
```



In [14]:
```
sns.scatterplot(x='Age', y='EstimatedSalary', data=df);
```



In [15]:
```
sns.scatterplot(x='Age', y='EstimatedSalary', data=df, hue='Purchased');
```

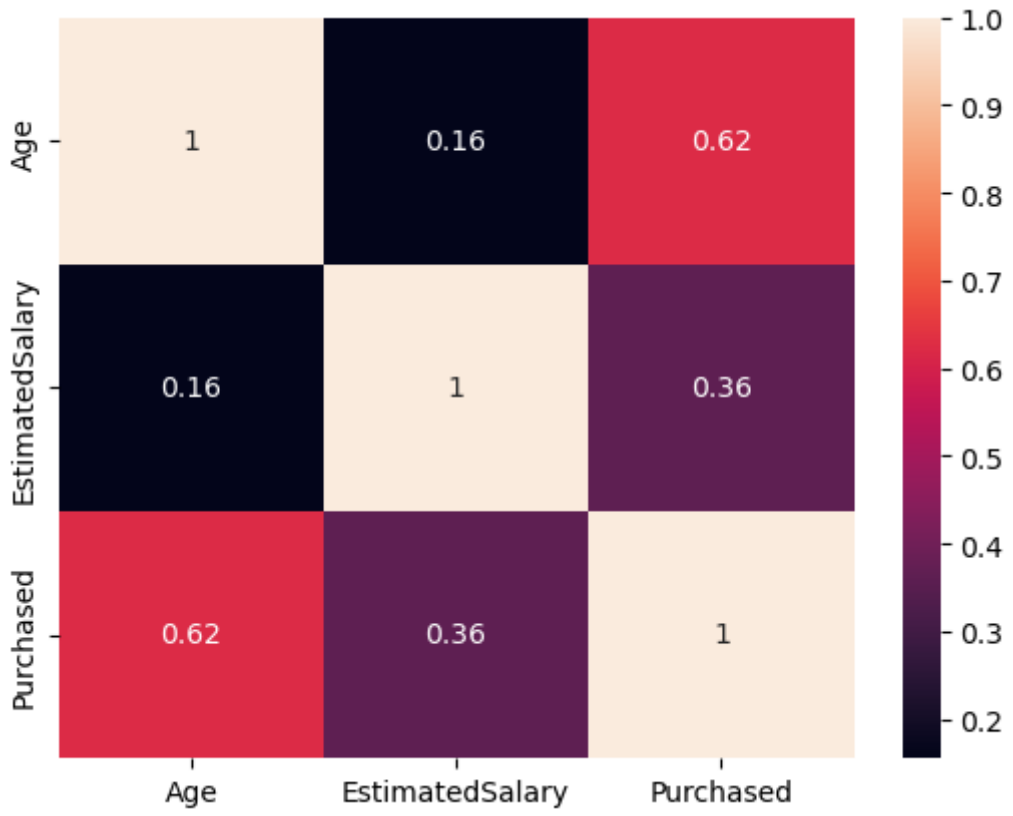


In [16]:
```
df.corr(numeric_only=True)
```

Out[16]:

	Age	EstimatedSalary	Purchased
Age	1.000000	0.155238	0.622454
EstimatedSalary	0.155238	1.000000	0.362083
Purchased	0.622454	0.362083	1.000000

In [17]:
```
sns.heatmap(df.corr(numeric_only=True), annot=True);
```



c. Train-Test Split

In [18]:
```
df
```

Out[18]:

	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
...	...	...	...	...
395	Female	46	41000	1
396	Male	51	23000	1
397	Female	50	20000	1
398	Male	36	33000	0
399	Female	49	36000	1

400 rows × 4 columns

In [19]:
```
from sklearn.model_selection import train_test_split
X = df.drop('Purchased', axis=1) # df.iloc[:,0:2]
y = df['Purchased']              # df.iloc[:,-1]

X = df.drop('Purchased', axis=1)
y = df['Purchased']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
len(X_train), len(y_train), len(X_test), len(y_test)
```

Out[19]:
```
(320, 320, 80, 80)
```

In [20]:
```
X_train
```

Out[20]:

	Gender	Age	EstimatedSalary
333	Male	40	65000
273	Male	39	106000
307	Female	47	113000
4	Male	19	76000
292	Male	55	39000
...	...	...	...
23	Female	45	22000
271	Female	59	76000
386	Female	49	39000
325	Female	41	60000
111	Female	37	71000

320 rows × 3 columns

d. Data Preprocessing and Feature Engineering

- Missing values Imputation
- Encoding Categorical Features
- Detecting and handling outliers
- Feature Scaling
- Extracting Information
- Combining Information

(i) Missing Value Imputation

In [21]:
```
df.isnull().sum()
```

Out[21]:
```
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

(ii) Encode the `Gender` Column using sklearn's `OneHotEncoder`

In [22]:
```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(drop='first',sparse=False,dtype=np.int32)

ohe.fit(X_train[['Gender']])

X_train['Gender'] = ohe.transform(X_train[['Gender']])
X_test['Gender'] = ohe.transform(X_test[['Gender']])
X_train
```

Out[22]:

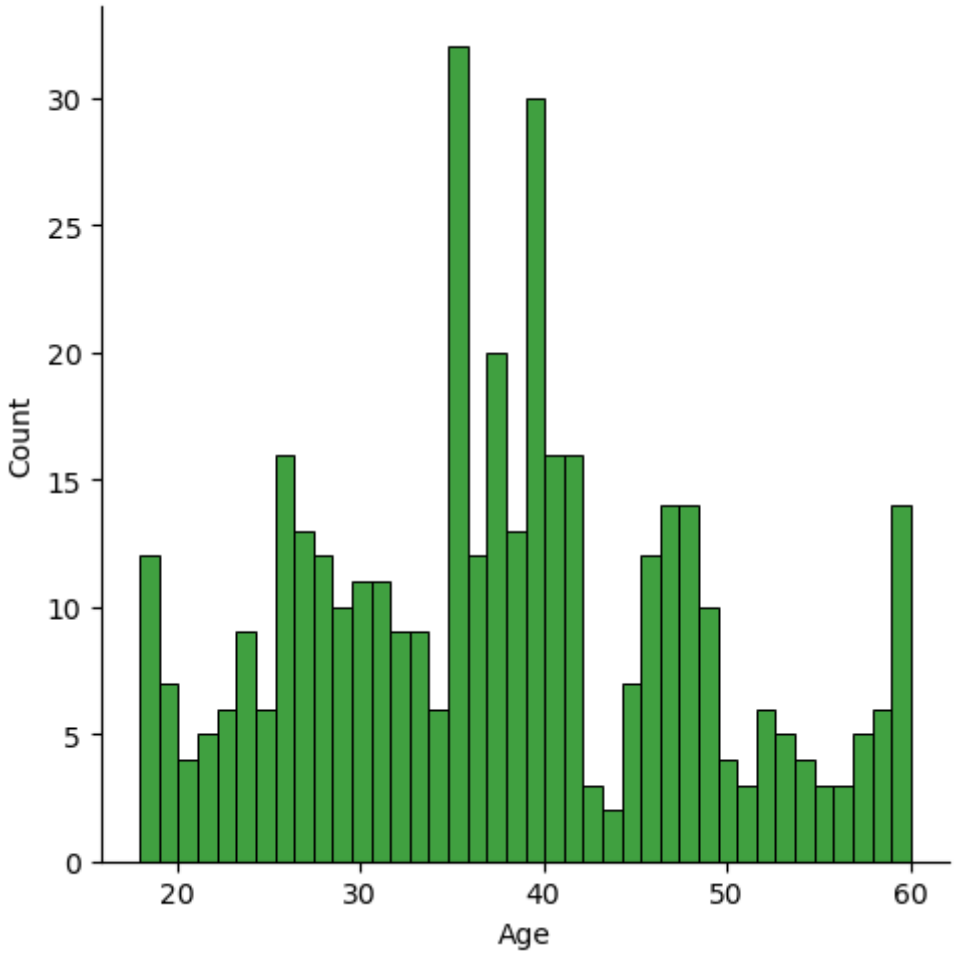
	Gender	Age	EstimatedSalary
333	1	40	65000
273	1	39	106000
307	0	47	113000
4	1	19	76000
292	1	55	39000
...	...	...	...
23	0	45	22000
271	0	59	76000
386	0	49	39000
325	0	41	60000
111	0	37	71000

320 rows × 3 columns

(iii) Detect and Handle Outliers

Check the distribution of Age Column and Detect outliers (if any):

In [23]:
```
# Check the distribution of data under the Age column
sns.displot(x='Age',data=df, kind='hist', bins=40, color='green');
```

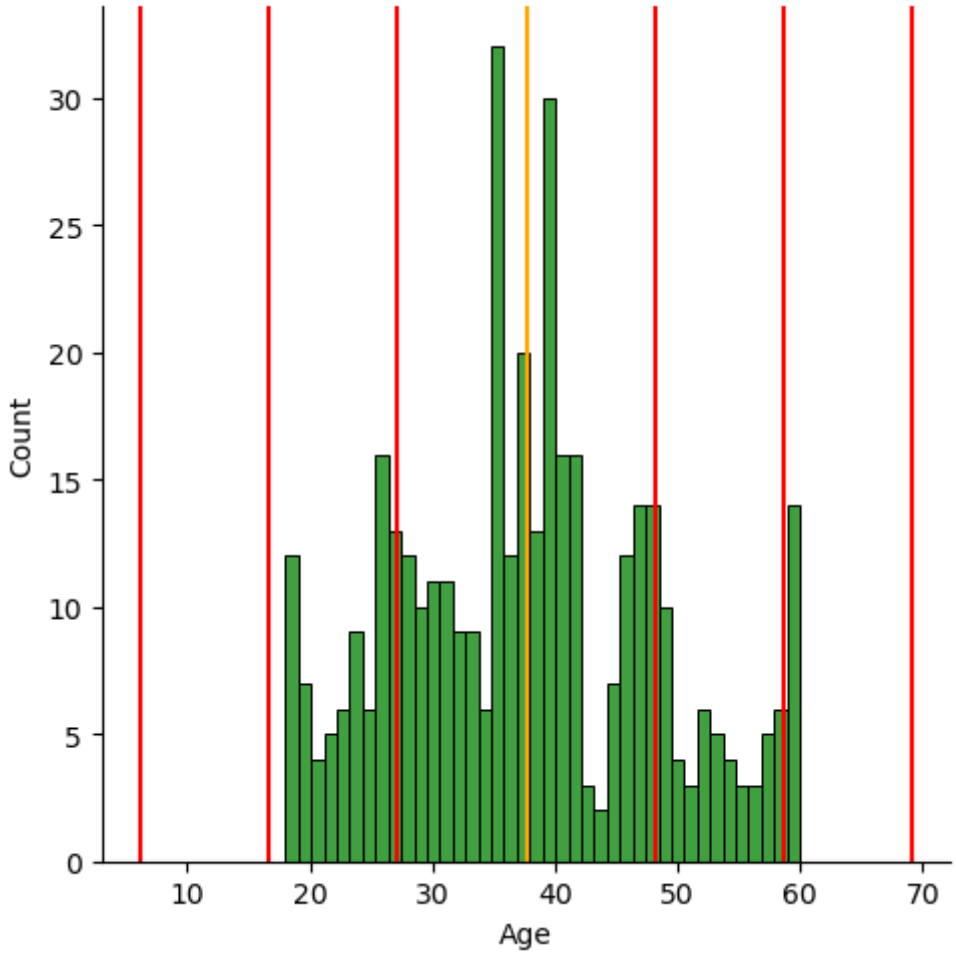


- Since the data is almost normally distributed, so we can use Z-Score method to treat outliers.

In [24]:
```
# Check out outliers, i.e., any datapoint before $\mu-3\sigma$ and beyon
mu = df.Age.mean()
sigma = df.Age.std()
minimum = df.Age.min()
maximum = df.Age.max()
print("Mean: ", mu)
print("Std Dev: ", sigma)
print("Min: ", minimum)
print("Max: ", maximum)

sns.displot(x='Age',data=df, kind='hist', bins=40, color='green')
plt.axvline(df.Age.mean(), color='orange')
for i in [-3, -2, -1, 1, 2, 3]:
    plt.axvline(mu+i*sigma, color='red')
plt.show()
```

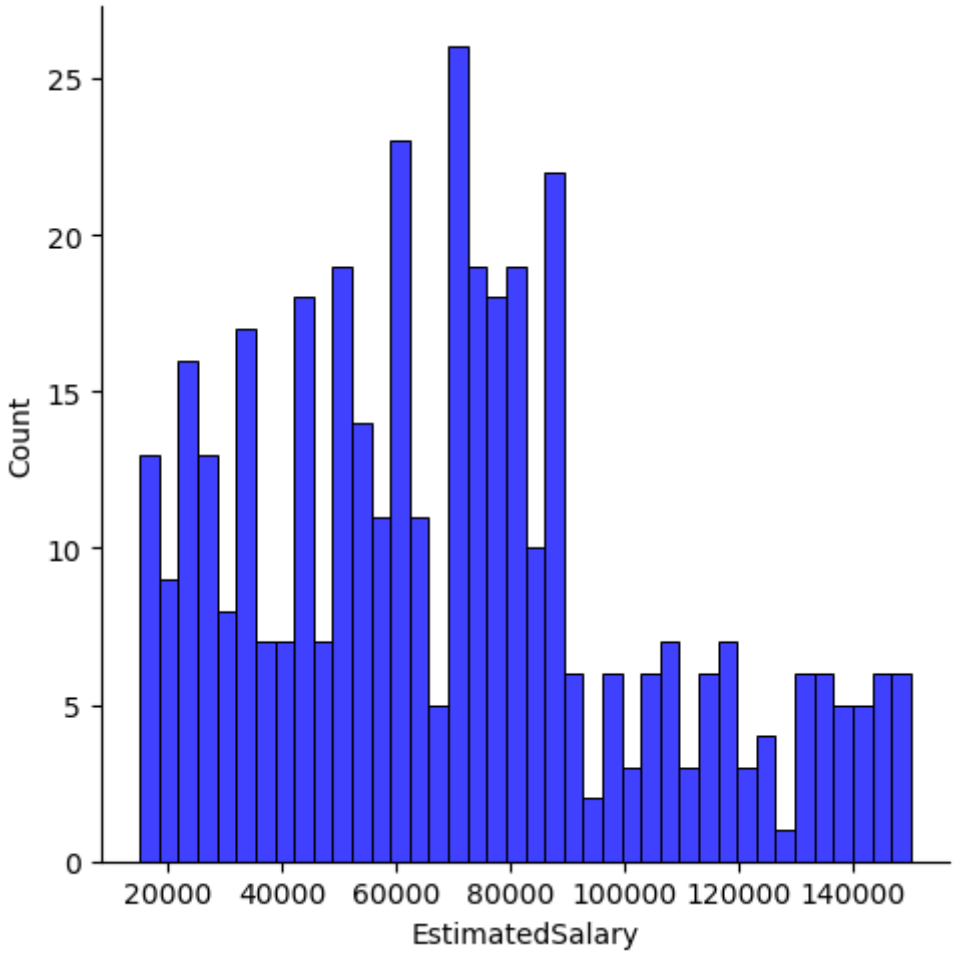
```
Mean:  37.655
Std Dev:  10.48287659730792
Min:  18
Max:  60
```



- So we can see that there are no outliers under the Age column

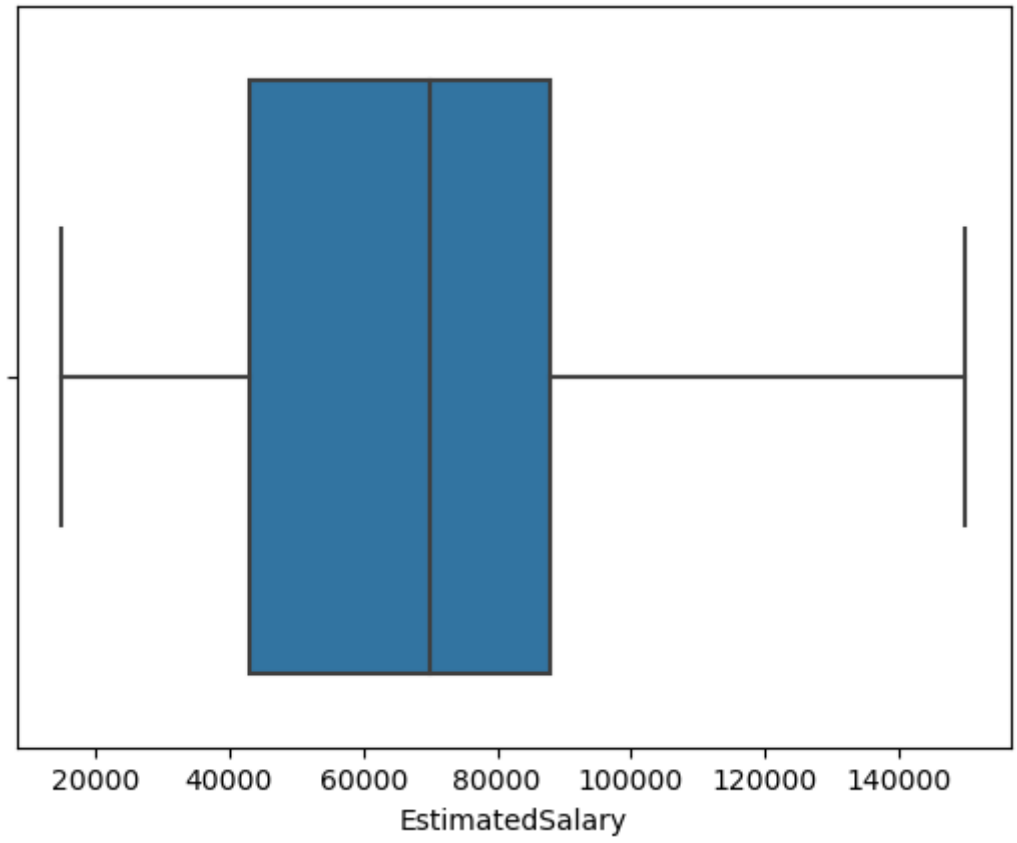
Check the distribution of EstimatedSalary Column and Detect outliers (if any):

In [25]:
```
# Check the distribution of data under the EstimatedSalary column
sns.displot(x='EstimatedSalary',data=df, kind='hist', bins=40, color='bl
```



- Since the data is NOT normally distributed, so we can use BoxPlot method to treat outliers.

In [26]:
```
# Check out outliers, i.e., any datapoint before Q1-1.5*IQR and beyond Q
sns.boxplot(x=df['EstimatedSalary'], data=df);
```



- So we can see that there are no outliers under the EstimatedSalary column

(iv) Feature Scaling

In [59]:
```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(X_train)

X_train = ss.transform(X_train)
X_test = ss.transform(X_test)
X_train
```

e. Model Training

(i) Instentiate and Train the `KNeighborsClassifier()` Model

- Now we will fit the K-NN classifier to the training data. To do this we will import the KNeighborsClassifier class of Sklearn Neighbors library. After importing the class, we will create the Classifier object of the class. The Parameter of this class will be
  - `n_neighbors=5` : To define the required neighbors of the algorithm. Usually, it takes 5.

- `metric='minkowski'` : This is the default parameter and it decides the distance between the points.
- `p={2,1}` : It is equivalent to the standard Euclidean metric.
- `weights={'uniform', 'distance'}`

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.get_params()
```

Out[28]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

In [60]:
```
#help(KNeighborsClassifier)
```

In [30]:
```
# Train our model
model.fit(X_train, y_train)
```

Out[30]:
```
KNeighborsClassifier()
```

> **Note:** KNN is a lazy learning algorithm, i.e., it do not learn in the training phase, rather, rather just stores the training data in appropriate data structures (depending on the `algorithm` hyperparameter) and use it later in the prediction phase

(ii) Carry out the Prediction on `X_test`

In [31]:
```
y_pred = model.predict(X_test)
y_pred
```

Out[31]:
```
array([1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
1,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
1,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0])
```

> During the prediction step, KNN calculate the Euclidean distance of each X_test data point from the saved X_train data points. It use majority vote from the K nearest neighbors and assign appropriate label to each X_test data point

(iii) Evaluate the Model

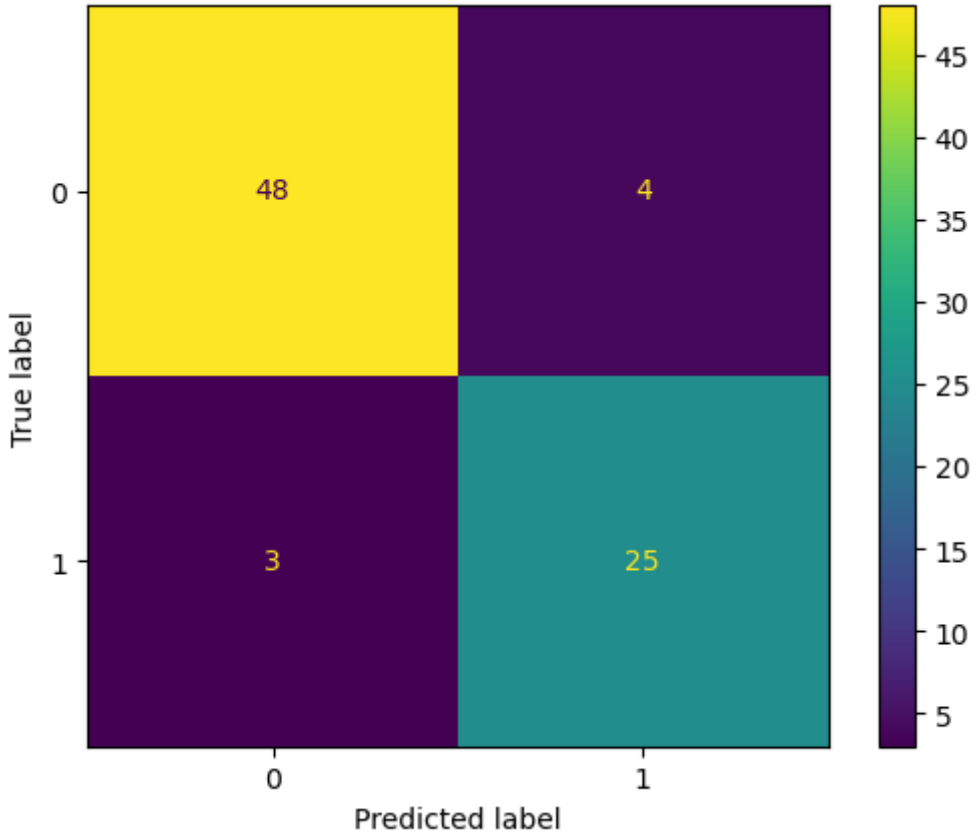
In [32]:
```
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test) # accuracy_score(y_test,y_pred)
print('Accuracy: {:.4f}'.format(accuracy_score(y_test, y_pred)))
```

```
Accuracy: 0.9125
```

In [33]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

Out[33]:
```
array([[48,  4],
       [ 3, 25]])
```

In [34]:
```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
#ConfusionMatrixDisplay.from_estimator(model, X_test, y_test);
```

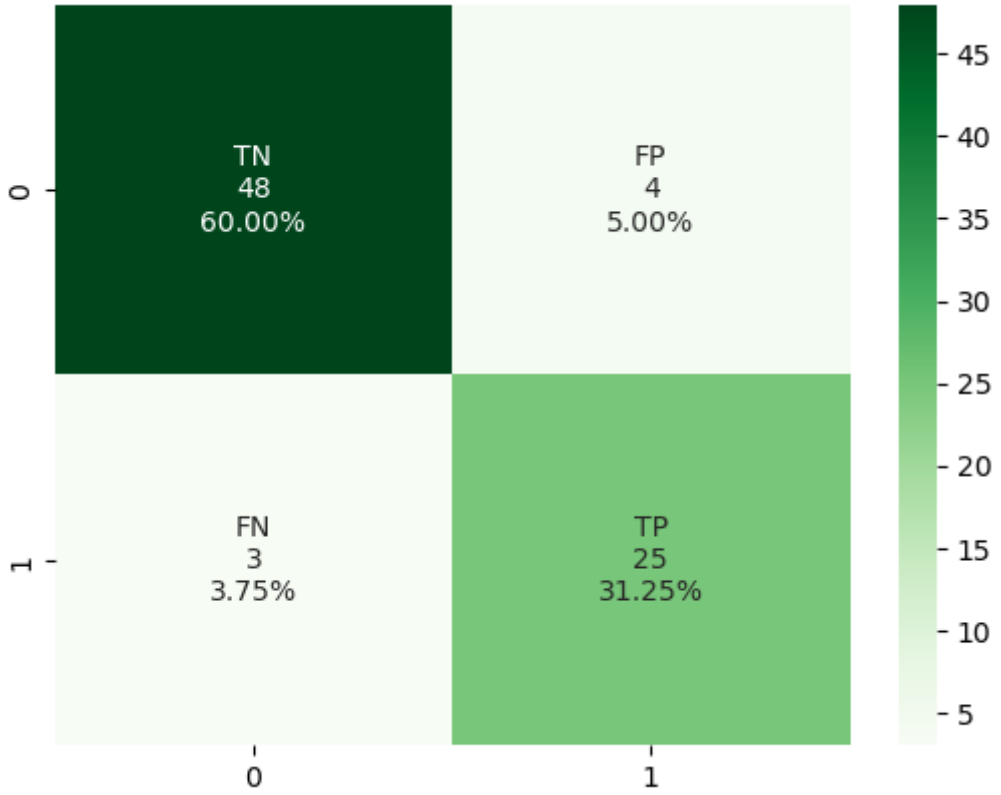


In [35]:
```
cm = confusion_matrix(y_test, y_pred)
TP = cm[1, 1]
TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]

names = ['TN','FP','FN','TP']
counts = [value for value in cm.flatten()]
percentages = ["{0:.2%}".format(value) for value in cm.flatten()/np.sum(

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(names, counts, perce
labels = np.asarray(labels).reshape(2,2)

sns.heatmap(confusion_matrix(y_test,y_pred), annot=labels, fmt='', cmap=
plt.show();
```



In [36]:
```
df.Purchased.value_counts()
```

Out[36]:
```
0    257
1    143
Name: Purchased, dtype: int64
```

In [37]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_scor

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average=None)
recall = recall_score(y_test, y_pred, average=None)
f1 = f1_score(y_test, y_pred, average=None)

print("Accuracy score: ", accuracy)
print("Precision score: ", precision)
print("Recall score: ", recall)
print("F1 score: ", f1)
print("\n Classification Report:\n", classification_report(y_test, y_pre
```

```
Accuracy score:  0.9125
Precision score:  [0.94117647 0.86206897]
Recall score:  [0.92307692 0.89285714]
F1 score:  [0.93203883 0.87719298]

 Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        52
           1       0.86      0.89      0.88        28

    accuracy                           0.91        80
   macro avg       0.90      0.91      0.90        80
weighted avg       0.91      0.91      0.91        80
```

f. Hyperparameter Tuning (Choosing Best Value of K)

(i) Finding Best Value of κ Mannually

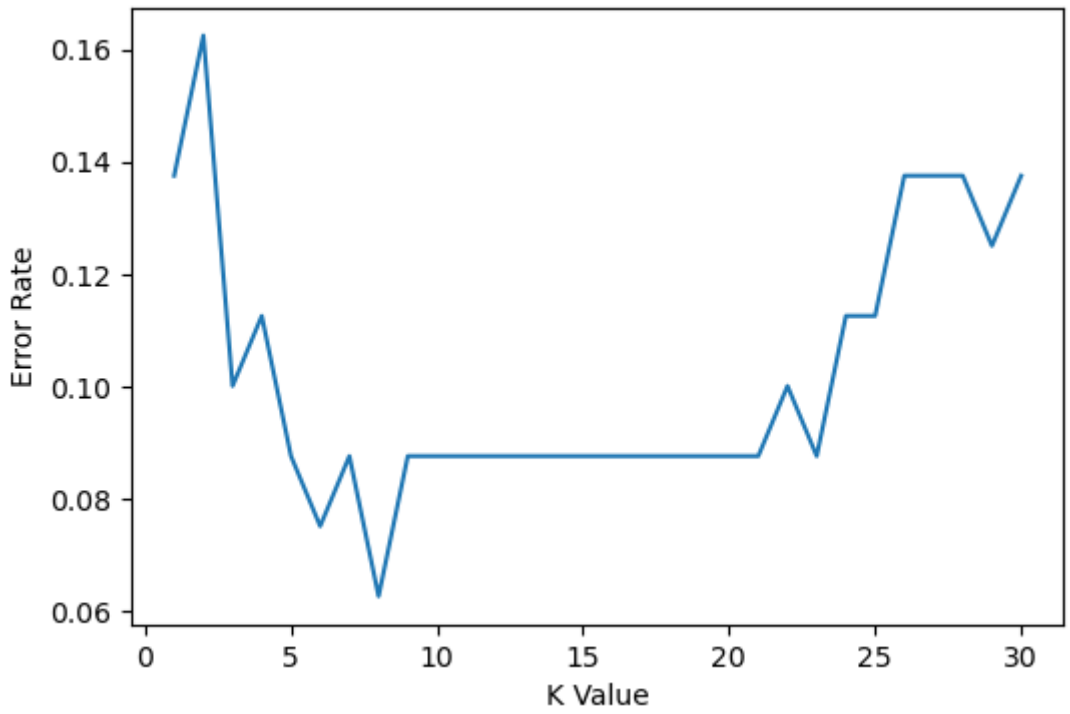
In [38]:
```
df=pd.read_csv('datasets/User_Data.csv')
X = df.drop('Purchased', axis=1) # df.iloc[:,0:2]
y = df['Purchased']              # df.iloc[:,-1]
# Do a train-test-split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
# One hot encoding
ohe = OneHotEncoder(drop='first',sparse=False,dtype=np.int32)
ohe.fit(X_train[['Gender']])
X_train['Gender'] = ohe.transform(X_train[['Gender']])
X_test['Gender'] = ohe.transform(X_test[['Gender']])
# SCALE DATA
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)

# Calculate errors
test_error_list = []
for k in range(1,31):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train,y_train)
    y_pred_test = knn_model.predict(X_test)

    test_error = 1 - accuracy_score(y_test, y_pred_test)
    test_error_list.append(test_error)
    print('For k={},\tError Rate={:.4f}'.format(k, test_error))
```

```
For k=1,        Error Rate=0.1375
For k=2,        Error Rate=0.1625
For k=3,        Error Rate=0.1000
For k=4,        Error Rate=0.1125
For k=5,        Error Rate=0.0875
For k=6,        Error Rate=0.0750
For k=7,        Error Rate=0.0875
For k=8,        Error Rate=0.0625
For k=9,        Error Rate=0.0875
For k=10,       Error Rate=0.0875
For k=11,       Error Rate=0.0875
For k=12,       Error Rate=0.0875
For k=13,       Error Rate=0.0875
For k=14,       Error Rate=0.0875
For k=15,       Error Rate=0.0875
For k=16,       Error Rate=0.0875
For k=17,       Error Rate=0.0875
For k=18,       Error Rate=0.0875
For k=19,       Error Rate=0.0875
For k=20,       Error Rate=0.0875
For k=21,       Error Rate=0.0875
For k=22,       Error Rate=0.1000
For k=23,       Error Rate=0.0875
For k=24,       Error Rate=0.1125
For k=25,       Error Rate=0.1125
For k=26,       Error Rate=0.1375
For k=27,       Error Rate=0.1375
For k=28,       Error Rate=0.1375
For k=29,       Error Rate=0.1250
For k=30,       Error Rate=0.1375
```

In [39]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(6,4),dpi=100)
plt.plot(range(1,31), test_error_list)
plt.ylabel('Error Rate')
plt.xlabel("K Value")
plt.show()
```



(ii) Finding Best Value of `K` using `GridSearchCV()` Method

In [40]:
```
from sklearn.model_selection import GridSearchCV

df=pd.read_csv('datasets/User_Data.csv')
X = df.drop('Purchased', axis=1) # df.iloc[:,0:2]
y = df['Purchased']              # df.iloc[:,-1]
# Do a train-test-split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
# One hot encoding
ohe = OneHotEncoder(drop='first',sparse=False,dtype=np.int32)
ohe.fit(X_train[['Gender']])
X_train['Gender'] = ohe.transform(X_train[['Gender']])
X_test['Gender'] = ohe.transform(X_test[['Gender']])
# SCALE DATA
ss=StandardScaler()
X_train=ss.fit_transform(X_train)
X_test=ss.transform(X_test)


#Dictionary with parameters names (`str`) as keys and lists of parameter
params = {'n_neighbors':list(range(1,20))}
# Find the best parameters out of the above dictionary using GridSearchC
gs = GridSearchCV(estimator=KNeighborsClassifier(),
                  param_grid=params,
                  scoring='accuracy',
                  cv=5)
gs.fit(X_train, y_train)

#Attributes of gs object
print("Best Score: ", gs.best_score_)
print("Best Value of K: ", gs.best_params_)
```

```
Best Score:  0.909375
Best Value of K:  {'n_neighbors': 9}
```

g. Decision Boundry and Impact of Different Values of K (Underfitting and Overfitting)

How does **K** affect the classifier? What happens if k=n What if k=1?

In a classification problem with two or more classes, a decision boundary or decision surface is a hypersurface that partitions the underlying vector space into two or more sets, one for each class. The classifier will classify all the points on one side of the decision boundary as belonging to one class and all those on the other side as belonging to the other class.

1. For smaller values of K(=1) the model generally overfits and shows High Variance.
2. For higher values of K(=n, where n is the no. of rows in training set), the model underfits and shows High Bias
3. As we increase the value of K the smoothness of Decision Boundary/Surface increases

In [41]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.plotting import plot_decision_regions
import warnings
warnings.filterwarnings("ignore")

#Load dataset
df=pd.read_csv('datasets/User_Data.csv')

#Drop Gender column to have just two input features for easy visualizati
X = df.drop(['Gender','Purchased'], axis=1)
y = df['Purchased']

# Do a train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Do scaling
ss = StandardScaler()
ss.fit(X_train)
X_train = ss.transform(X_train)
X_test = ss.transform(X_test)
```

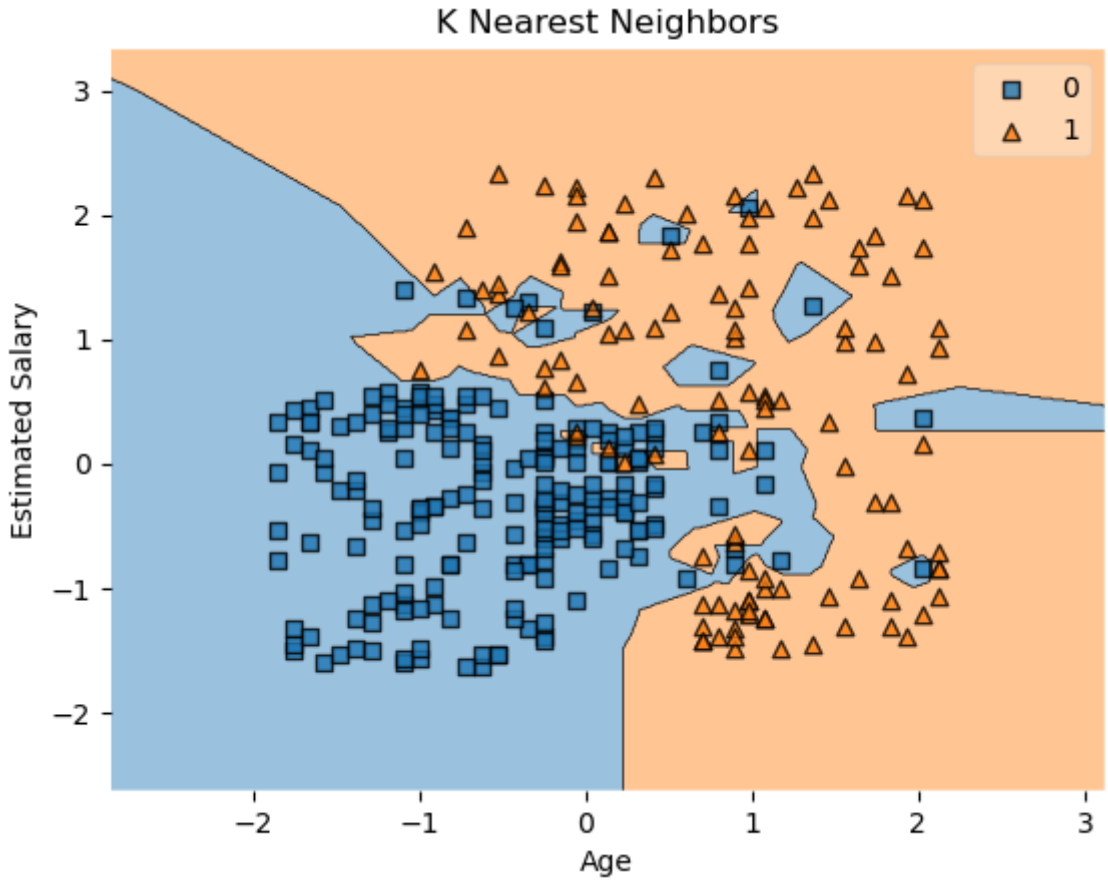
What happens if K=1

```
https://rasbt.github.io/mlxtend/
import sys
!{sys.executable} -m pip install mlxtend --upgrade --no-deps
```

In [42]:
```
from mlxtend.plotting import plot_decision_regions

model=KNeighborsClassifier(n_neighbors=1)
model.fit(X_train, y_train)
# Plot decision boundry
plot_decision_regions(X=X_train, y=y_train.values, clf=model);

plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.title('K Nearest Neighbors')
plt.show();
```



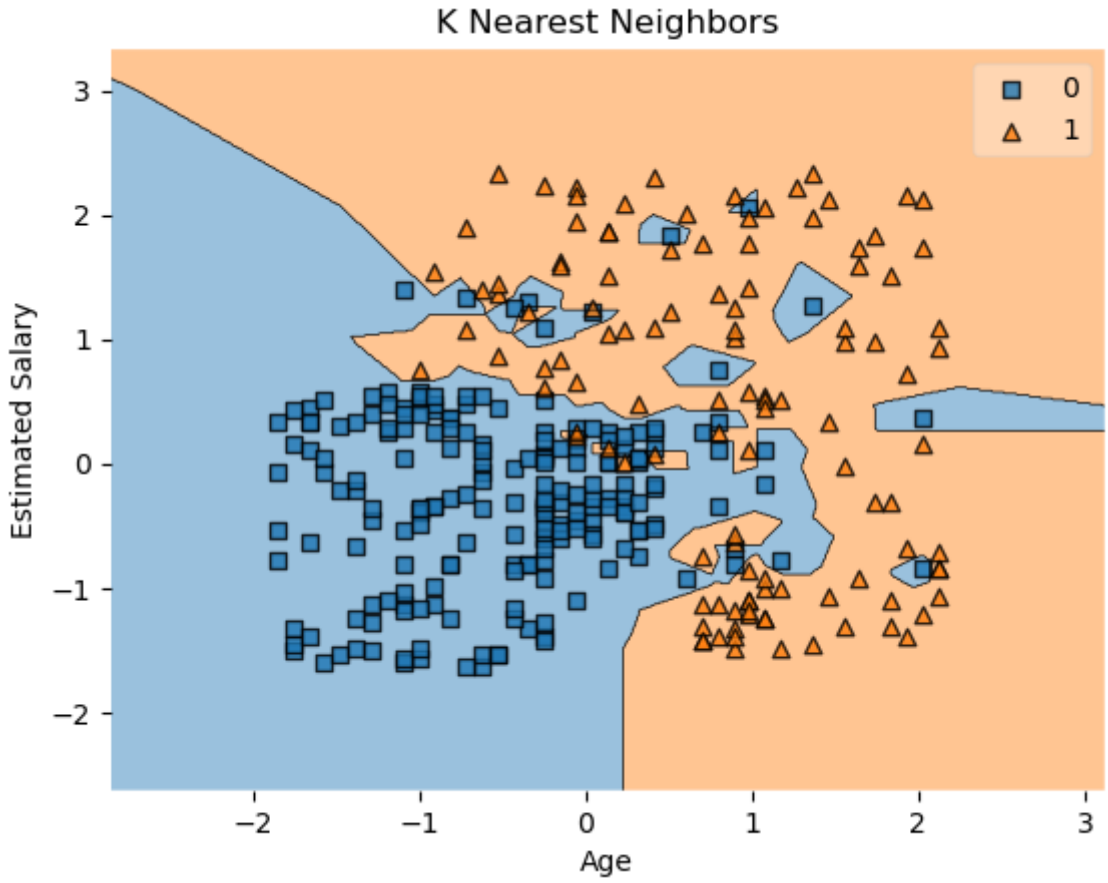
For k=1, the model is generally overfits and show high variance and is sensitive to noise

What happens if we increase the value of k ?

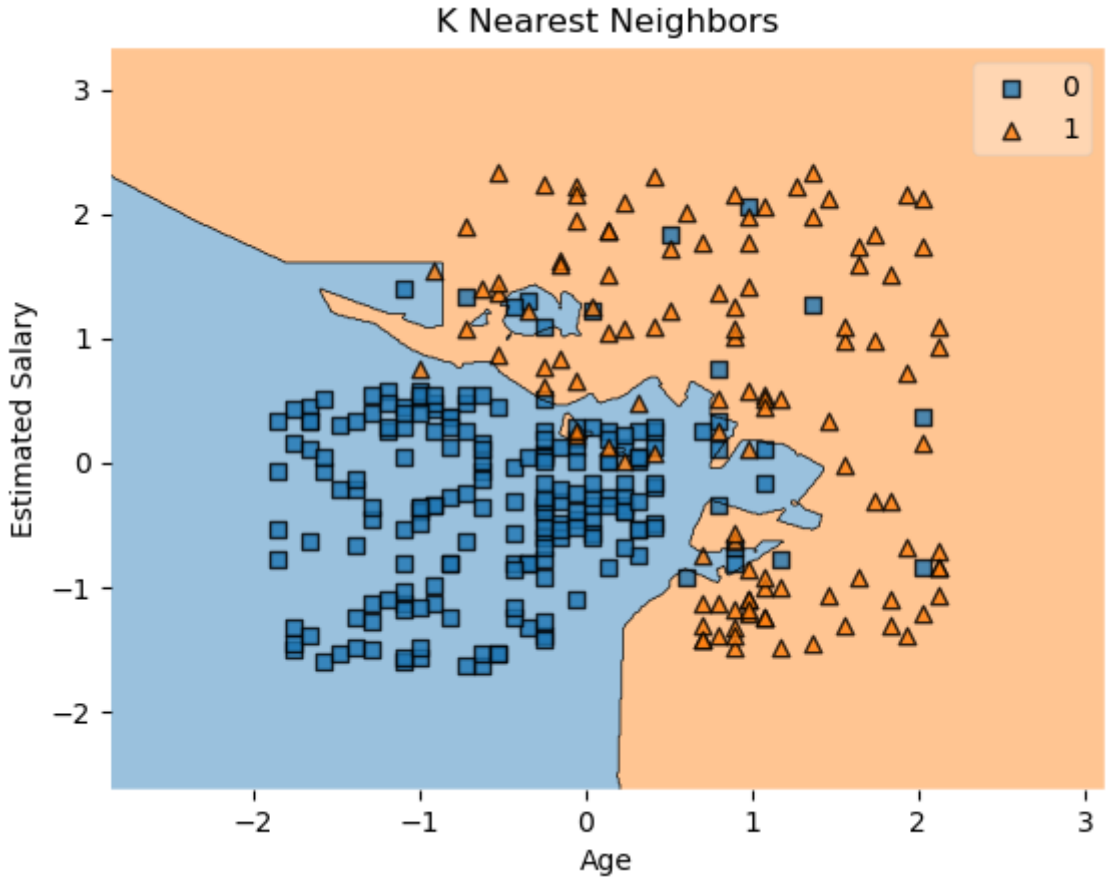
In [43]:
```
for k in range(1,10,2):
    model=KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)

# Plot decision boundry
    print("k=", k)
    plot_decision_regions(X=X_train, y=y_train.values, clf=model)
    plt.xlabel('Age')
    plt.ylabel('Estimated Salary')
    plt.title('K Nearest Neighbors')
    plt.show();
```

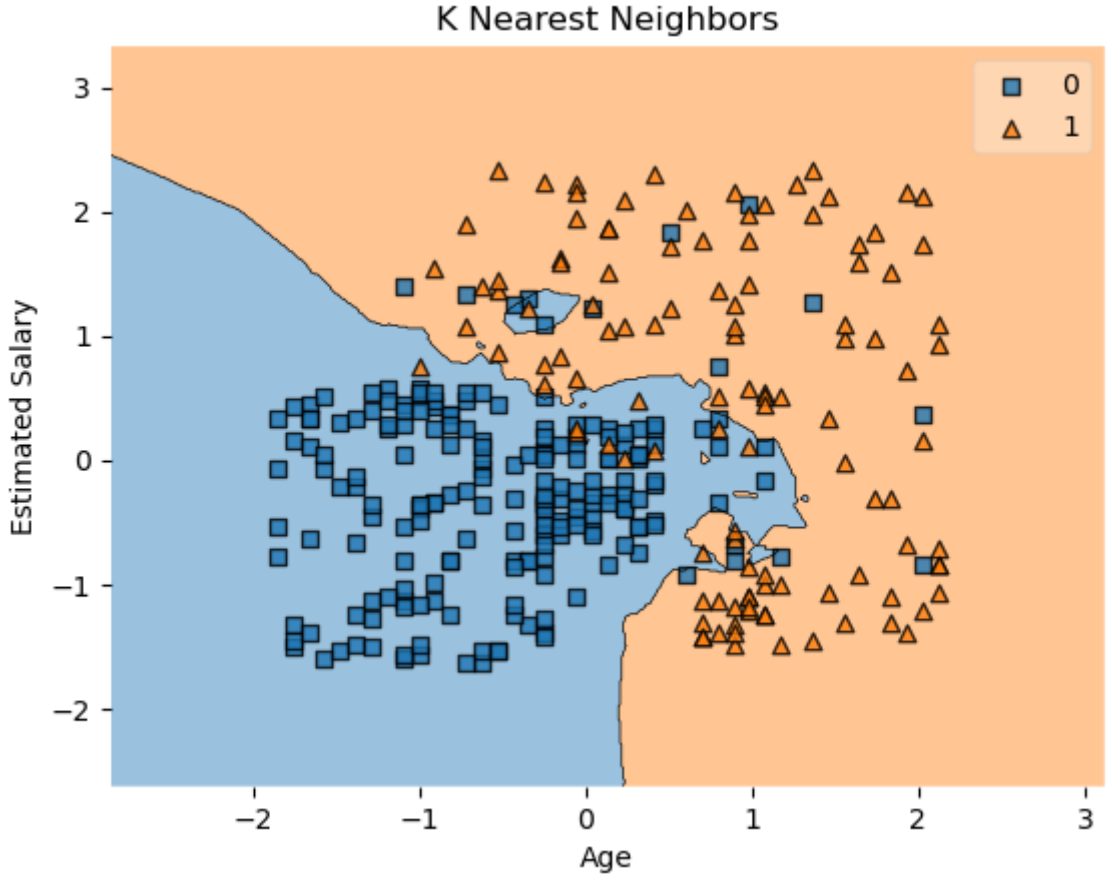
```
k= 1
```



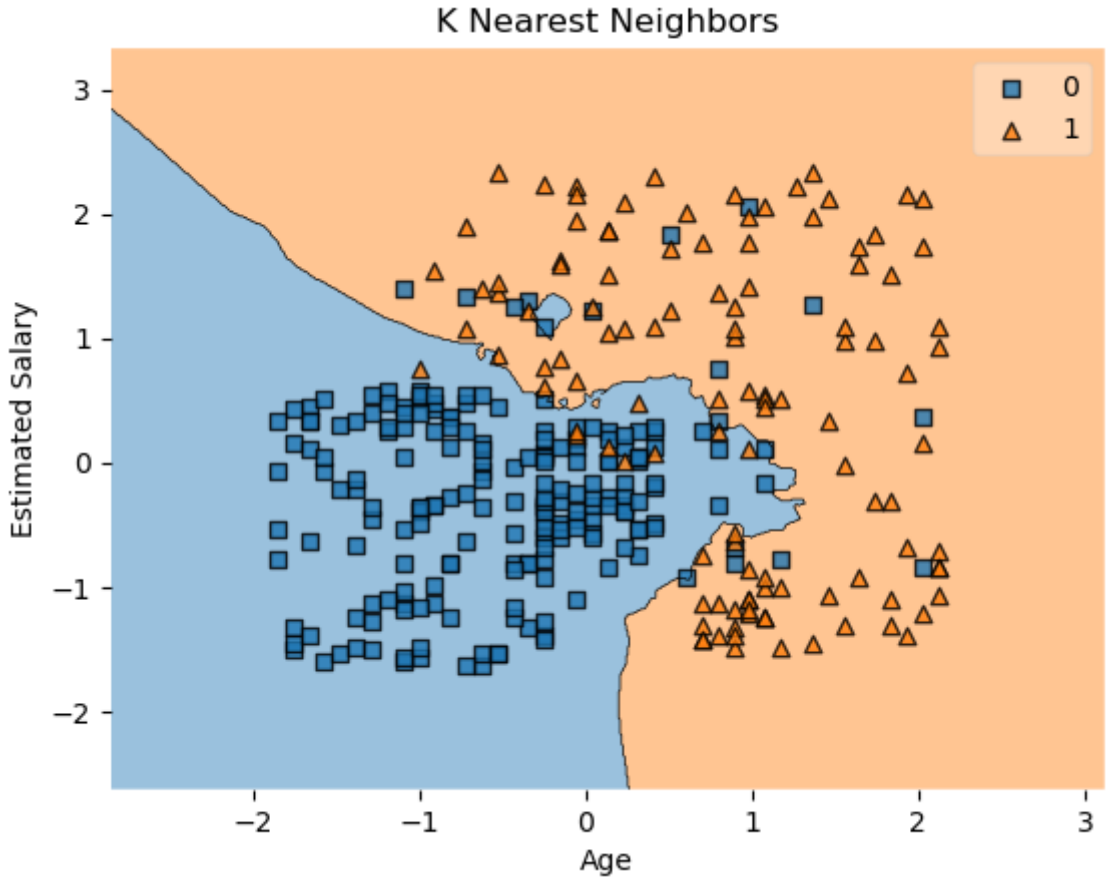

```
k= 3
```


K Nearest Neighbors
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

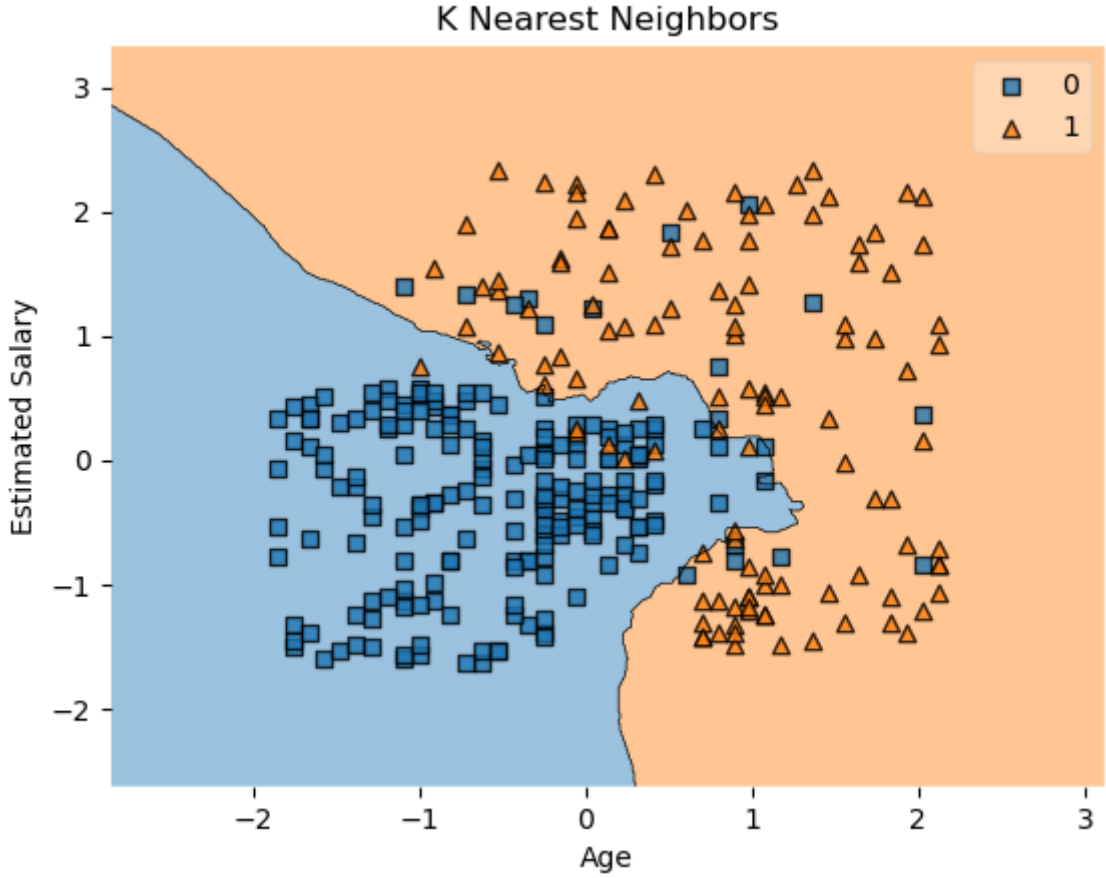
k= 5


K Nearest Neighbors
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

k= 7


K Nearest Neighbors
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

k= 9


K Nearest Neighbors
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

> If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority. The training error rate and the validation error rate are two parameters we need to access different K-value.

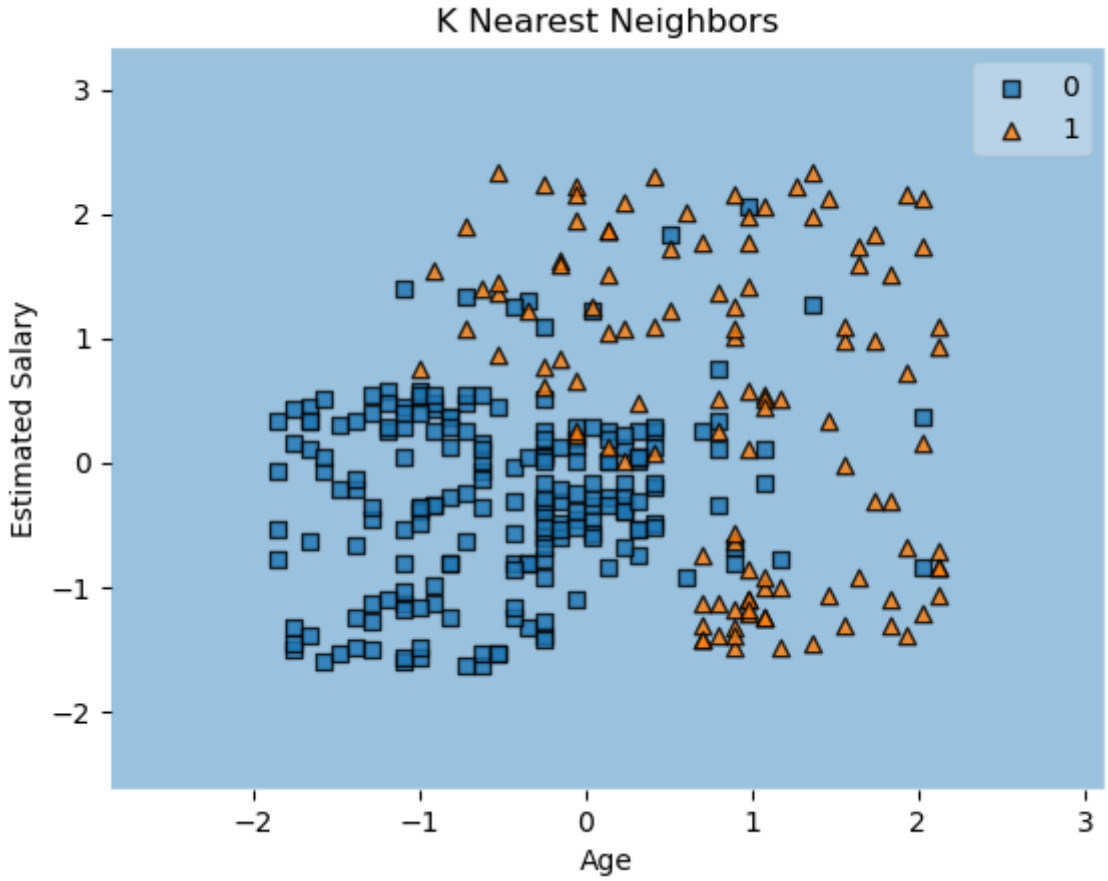
What happens if we set `K=n`?

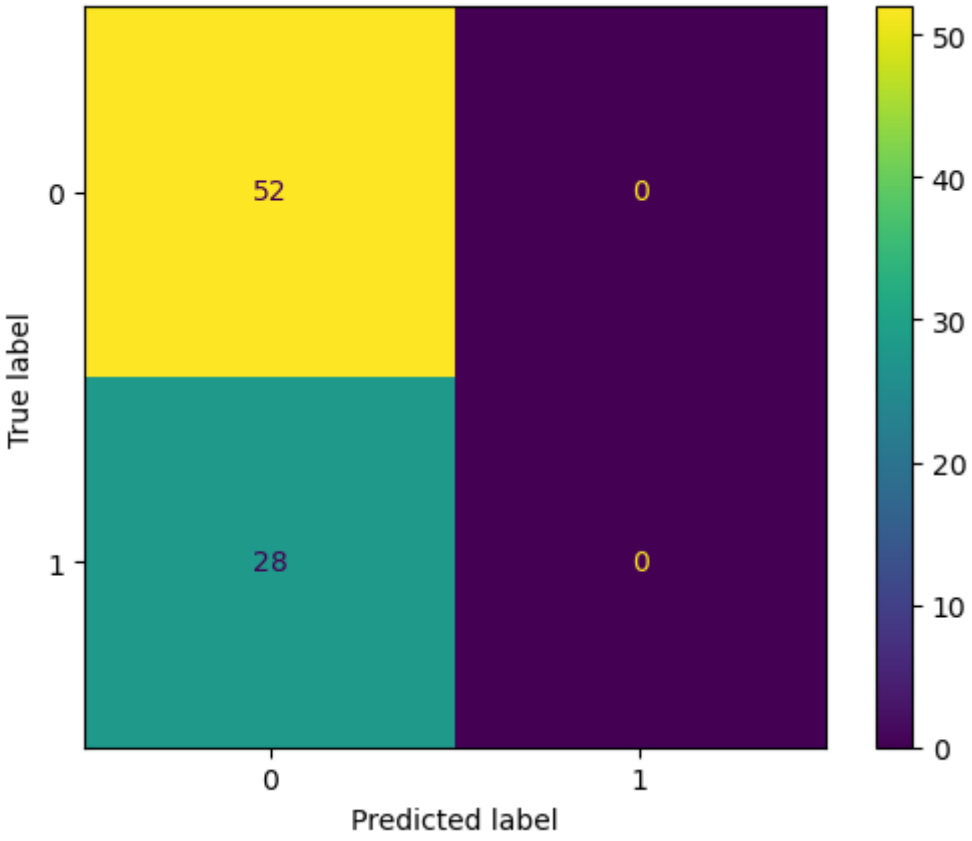
In [44]:
```
X_train.shape
```

Out[44]:
```
(320, 2)
```

In [45]:
```
model=KNeighborsClassifier(n_neighbors=320)
model.fit(X_train, y_train)

# Plot decision boundry
plot_decision_regions(X=X_train, y=y_train.values, clf=model)
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.title('K Nearest Neighbors')
plt.show();
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test);
```





For k=n (where n is the no. of rows in training set), all the data points are classified as belonging to Class 0, which is the majority class (shows extreme high Bias)

Multi-Class Classification using KNN

Load iris Dataset

In [46]:
```
from sklearn import datasets
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np

iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
df.sample(10, random_state=54)
```

Out[46]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
45	4.8	3.0	1.4	0.3	0
91	6.1	3.0	4.6	1.4	1
103	6.3	2.9	5.6	1.8	2
94	5.6	2.7	4.2	1.3	1
96	5.7	2.9	4.2	1.3	1
42	4.4	3.2	1.3	0.2	0
79	5.7	2.6	3.5	1.0	1
116	6.5	3.0	5.5	1.8	2
4	5.0	3.6	1.4	0.2	0

In [47]:
```
df.rename(columns={'sepal length (cm)':'sepal_length',
                   'sepal width (cm)':'sepal_width',
                   'petal length (cm)':'petal_length',
                   'petal width (cm)': 'petal_width'},
                    inplace=True)
df.sample(10, random_state=54)
```

Out[47]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
45	4.8	3.0	1.4	0.3	0
91	6.1	3.0	4.6	1.4	1
103	6.3	2.9	5.6	1.8	2
94	5.6	2.7	4.2	1.3	1
96	5.7	2.9	4.2	1.3	1
42	4.4	3.2	1.3	0.2	0
79	5.7	2.6	3.5	1.0	1
116	6.5	3.0	5.5	1.8	2
4	5.0	3.6	1.4	0.2	0

Do a Train-Test-Split

In [48]:
```
from sklearn.model_selection import train_test_split

X = df.drop('species', axis=1) # df.iloc[:,0:2]
y = df['species']              # df.iloc[:,-1]

X = df.drop('species', axis=1)
y = df['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
len(X_train), len(y_train), len(X_test), len(y_test)
```

Out[48]:
```
(120, 120, 30, 30)
```

Instentiate and Train the `KNN()` Model

In [49]:
```
from sklearn.neighbors import KNeighborsClassifier
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)


model = KNeighborsClassifier(n_neighbors=5)# change values of n_neighbor
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred
```

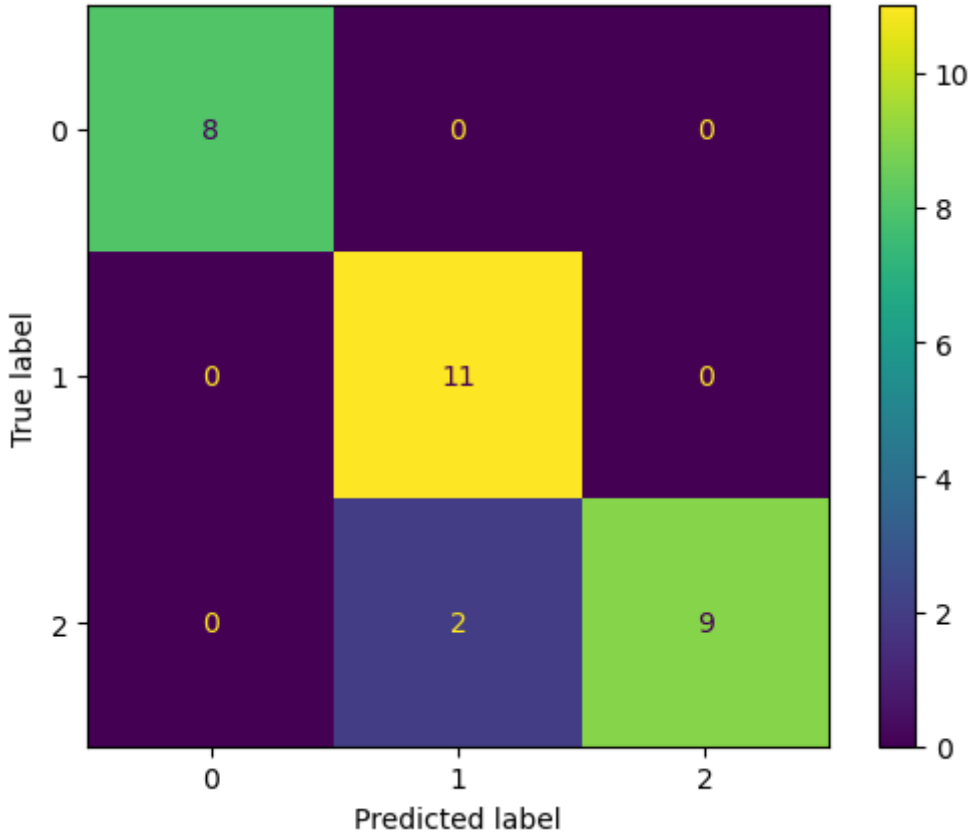
Out[49]:
```
array([0, 0, 1, 2, 1, 1, 0, 1, 2, 0, 0, 2, 2, 2, 1, 1, 2, 2, 0, 0, 1,
2,
       1, 1, 2, 1, 1, 0, 1, 1])
```

Confusion Matrix

In [50]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[50]:
```
array([[ 8,  0,  0],
       [ 0, 11,  0],
       [ 0,  2,  9]])
```

In [51]:
```
from sklearn.metrics import ConfusionMatrixDisplay
#disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred))
#disp.plot();
ConfusionMatrixDisplay.from_predictions(y_test, y_pred);
```



In [52]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       0.85      1.00      0.92        11
           2       1.00      0.82      0.90        11

    accuracy                           0.93        30
   macro avg       0.95      0.94      0.94        30
weighted avg       0.94      0.93      0.93        30
```

4. Limitations of KNN

(i) Doesn't work well with large datasets

(ii) Doesn’t work well on high dimensional data

(iii) Doesn’t work well with dataset having outliers

(iv) Doesn’t work well with dataset having non-homogenious scales

(v) Doesn’t work well with imbalanced datasets

(v) Doesn't tell you how much each i/p feature is contributing in deciding, to which class the new query point belong to

5. Enhancements of KNN

- **Performance of KNN:**
  - **Training:** $O(1)$, but testing: $O(nm)$
    - **Reduce `m` :** Dimensionality reduction
    - **Reduce `n` :** Don't compare to all training examples, rather identify $n << N$ potential near neighbors and compare only to those
- There are different Approximate Nearest Neighbor techniques that speed up the search by preprocessing the data into an efficient index. Amongst these, there are data structure-based techniques like:

(i) K-D Trees

(ii) Locality-Sensitive Hashing

(iii) Inverted Lists

- An Advanced KNN Classification Algorithm Based on KD-tree: https://ieeexplore.ieee.org/abstract/document/8690508 (https://ieeexplore.ieee.org/abstract/document/8690508)
- An Enhanced KNN Algorithm Using Information Gain and Clustering: https://ieeexplore.ieee.org/abstract/document/6783471 (https://ieeexplore.ieee.org/abstract/document/6783471)
- Combining Fuzzy Logic and KNN Algorithm for Recommendation Systems: https://www.mecs-press.org/ijitcs/ijitcs-v13-n4/IJITCS-V13-N4-1.pdf (https://www.mecs-press.org/ijitcs/ijitcs-v13-n4/IJITCS-V13-N4-1.pdf)
- A new variant of Fuzzy K-Nearest Neighbor using Interval Type-2 Fuzzy Logic: https://ieeexplore.ieee.org/document/8491472 (https://ieeexplore.ieee.org/document/8491472)
- A new fuzzy k-nearest neighbor classifier based on the Bonferroni mean: https://www.sciencedirect.com/science/article/abs/pii/S0167865520303792 (https://www.sciencedirect.com/science/article/abs/pii/S0167865520303792)
- Comparison of KNN Algorithm and Fuzzy Tsukamoto Logic: https://iopscience.iop.org/article/10.1088/1742-6596/1175/1/012068/pdf (https://iopscience.iop.org/article/10.1088/1742-6596/1175/1/012068/pdf)

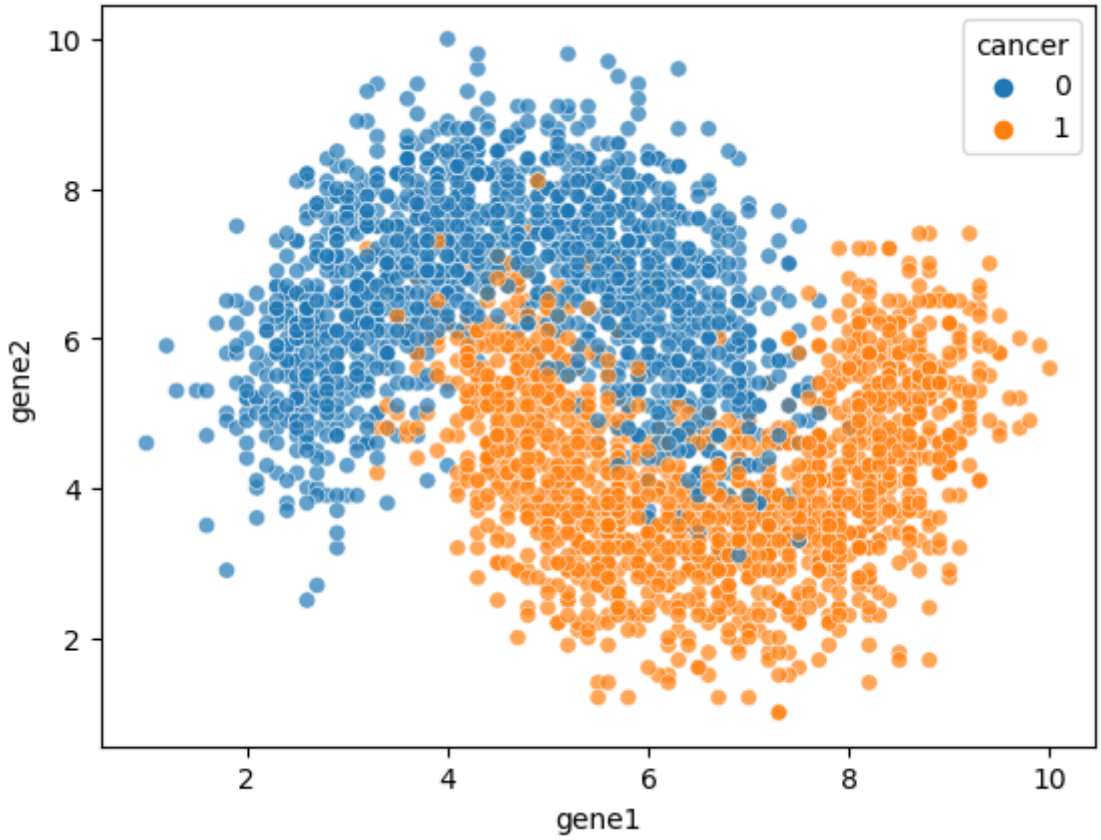
Tasks To Do

In [53]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn import datasets
```

a. Cancer-Genes (Binary Classification)

In [54]:
```
df = pd.read_csv('datasets/gene_expression.csv')
print(df.head())
sns.scatterplot(x='gene1',y='gene2',hue='cancer',data=df,alpha=0.7);
```

```
   gene1  gene2  cancer
0    4.3    3.9       1
1    2.5    6.3       0
2    5.7    3.9       1
3    6.1    6.2       0
4    7.4    3.4       1
```



b. Breast-Cancer (Binary Classification)

In [55]:
```
cancer = datasets.load_breast_cancer()
df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
df['target'] = cancer.target
df.head()
```

Out[55]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

c. Titanic Dataset (Binary Classification)

In [56]:
```
titanic = datasets.fetch_openml(name='titanic', version=1)
df = pd.DataFrame(titanic.data, columns=titanic.feature_names)
df['target'] = titanic.target
df
```

Out[56]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	b
0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	
1	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	
2	1.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
3	1.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
4	1.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
...	...	...	...	...	...	...	...	...	...	...	
1304	3.0	Zabour, Miss. Hileni	female	14.5000	1.0	0.0	2665	14.4542	None	C	N
1305	3.0	Zabour, Miss. Thamine	female	NaN	1.0	0.0	2665	14.4542	None	C	N
1306	3.0	Zakarian, Mr. Mapriededer	male	26.5000	0.0	0.0	2656	7.2250	None	C	N
1307	3.0	Zakarian, Mr. Ortin	male	27.0000	0.0	0.0	2670	7.2250	None	C	N
1308	3.0	Zimmerman, Mr. Leo	male	29.0000	0.0	0.0	315082	7.8750	None	S	N

1309 rows × 14 columns

d. iris Dataset (Multi-Class Classification)

In [57]:
```
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
df.sample(10, random_state=54)
```

Out[57]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
45	4.8	3.0	1.4	0.3	0
91	6.1	3.0	4.6	1.4	1
103	6.3	2.9	5.6	1.8	2
94	5.6	2.7	4.2	1.3	1
96	5.7	2.9	4.2	1.3	1
42	4.4	3.2	1.3	0.2	0
79	5.7	2.6	3.5	1.0	1
116	6.5	3.0	5.5	1.8	2
4	5.0	3.6	1.4	0.2	0

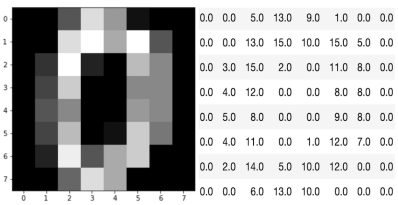
e. Digit Classification

In [58]:
```
digits = datasets.load_digits()
df = pd.DataFrame(digits.data, columns=digits.feature_names)
df['target'] = digits.target
df.head()
```

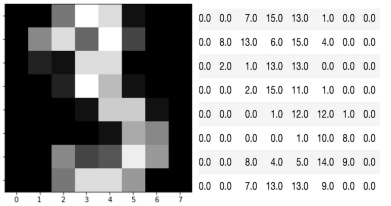
Out[58]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0

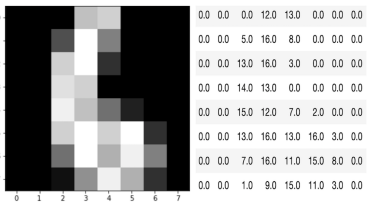
5 rows × 65 columns



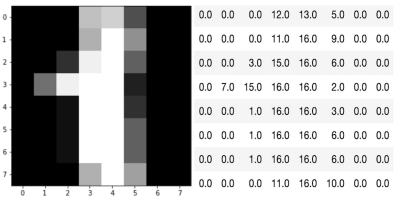
0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0
0.0	0.0	13.0	15.0	10.0	15.0	5.0	0.0
0.0	3.0	15.0	2.0	0.0	11.0	8.0	0.0
0.0	4.0	12.0	0.0	0.0	8.0	8.0	0.0
0.0	5.0	8.0	0.0	0.0	9.0	8.0	0.0
0.0	4.0	11.0	0.0	1.0	12.0	7.0	0.0
0.0	2.0	14.0	5.0	10.0	12.0	0.0	0.0
0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0



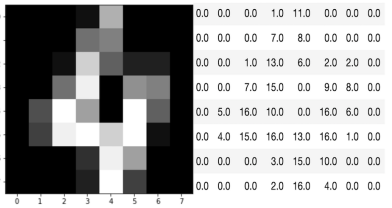
0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0
0.0	8.0	13.0	6.0	15.0	4.0	0.0	0.0
0.0	2.0	1.0	13.0	13.0	0.0	0.0	0.0
0.0	0.0	2.0	15.0	11.0	1.0	0.0	0.0
0.0	0.0	0.0	1.0	12.0	12.0	1.0	0.0
0.0	0.0	0.0	0.0	1.0	10.0	8.0	0.0
0.0	0.0	8.0	4.0	5.0	14.0	9.0	0.0
0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0



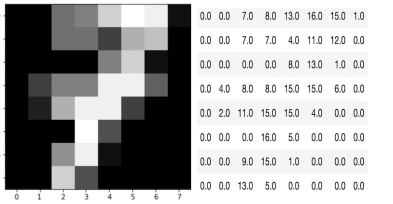
0.0	0.0	0.0	12.0	13.0	0.0	0.0	0.0
0.0	0.0	5.0	16.0	8.0	0.0	0.0	0.0
0.0	0.0	13.0	16.0	3.0	0.0	0.0	0.0
0.0	0.0	14.0	13.0	0.0	0.0	0.0	0.0
0.0	0.0	15.0	12.0	7.0	2.0	0.0	0.0
0.0	0.0	13.0	16.0	13.0	16.0	3.0	0.0
0.0	0.0	7.0	16.0	11.0	15.0	8.0	0.0
0.0	0.0	1.0	9.0	15.0	11.0	3.0	0.0



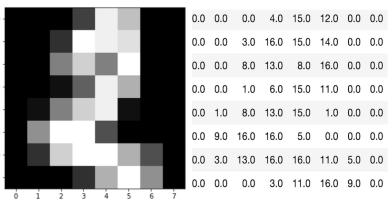
0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0
0.0	0.0	0.0	11.0	16.0	9.0	0.0	0.0
0.0	0.0	3.0	15.0	16.0	6.0	0.0	0.0
0.0	7.0	15.0	16.0	16.0	2.0	0.0	0.0
0.0	0.0	1.0	16.0	16.0	3.0	0.0	0.0
0.0	0.0	1.0	16.0	16.0	6.0	0.0	0.0
0.0	0.0	1.0	16.0	16.0	6.0	0.0	0.0
0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0



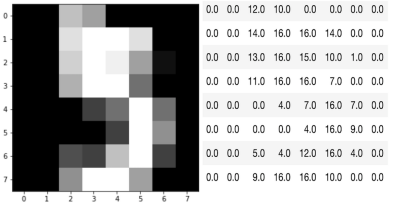
0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0
0.0	0.0	0.0	7.0	8.0	0.0	0.0	0.0
0.0	0.0	1.0	13.0	6.0	2.0	2.0	0.0
0.0	0.0	7.0	15.0	0.0	9.0	8.0	0.0
0.0	5.0	16.0	10.0	0.0	16.0	6.0	0.0
0.0	4.0	15.0	16.0	13.0	16.0	1.0	0.0
0.0	0.0	0.0	3.0	15.0	10.0	0.0	0.0
0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0



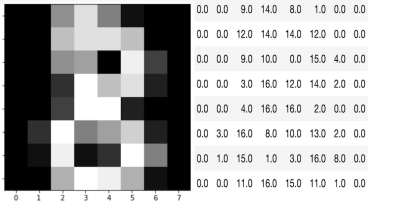
0.0	0.0	7.0	8.0	13.0	16.0	15.0	1.0
0.0	0.0	7.0	7.0	4.0	11.0	12.0	0.0
0.0	0.0	0.0	0.0	8.0	13.0	1.0	0.0
0.0	4.0	8.0	8.0	15.0	15.0	6.0	0.0
0.0	2.0	11.0	15.0	15.0	4.0	0.0	0.0
0.0	0.0	0.0	16.0	5.0	0.0	0.0	0.0
0.0	0.0	9.0	15.0	1.0	0.0	0.0	0.0
0.0	0.0	13.0	5.0	0.0	0.0	0.0	0.0



0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0
0.0	0.0	3.0	16.0	15.0	14.0	0.0	0.0
0.0	0.0	8.0	13.0	8.0	16.0	0.0	0.0
0.0	0.0	1.0	6.0	15.0	11.0	0.0	0.0
0.0	1.0	8.0	13.0	15.0	1.0	0.0	0.0
0.0	9.0	16.0	16.0	5.0	0.0	0.0	0.0
0.0	3.0	13.0	16.0	16.0	11.0	5.0	0.0
0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0



0.0	0.0	12.0	10.0	0.0	0.0	0.0	0.0
0.0	0.0	14.0	16.0	16.0	14.0	0.0	0.0
0.0	0.0	13.0	16.0	15.0	10.0	1.0	0.0
0.0	0.0	11.0	16.0	16.0	7.0	0.0	0.0
0.0	0.0	0.0	4.0	7.0	16.0	7.0	0.0
0.0	0.0	0.0	0.0	4.0	16.0	9.0	0.0
0.0	0.0	5.0	4.0	12.0	16.0	4.0	0.0
0.0	0.0	9.0	16.0	16.0	10.0	0.0	0.0



0.0	0.0	9.0	14.0	8.0	1.0	0.0	0.0
0.0	0.0	12.0	14.0	14.0	12.0	0.0	0.0
0.0	0.0	9.0	10.0	0.0	15.0	4.0	0.0
0.0	0.0	3.0	16.0	12.0	14.0	2.0	0.0
0.0	0.0	4.0	16.0	16.0	2.0	0.0	0.0
0.0	3.0	16.0	8.0	10.0	13.0	2.0	0.0
0.0	1.0	15.0	1.0	3.0	16.0	8.0	0.0
0.0	0.0	11.0	16.0	15.0	11.0	1.0	0.0