Introduction to Machine Learning

- what is machine learning
    - Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy
    - Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed
    - Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.
- Types of machine learning algorithms
    - **Supervised machine learning algorithms**
        - Can apply what has been learned in the past to new data using labeled examples to predict future events
        - Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values
        - The system is able to provide targets for any new input after sufficient training
        - The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly
        - Type of algorithms
            - **Regression**
                - Population growth prediction
                - Expecting life expectancy
                - Market forecasting/prediction
                - Advertising Popularity prediction
                - Stock prediction
                - E.g.
                    - Linear and multi-linear regression
                    - Logistic regression
                    - Naïve Bayes
                    - Support Vector Machine
            - **Classification**
                - Find whether an email received is a spam or ham
                - Identify customer segments
                - Find if a bank loan is granted
                - Identify if a kid will pass or fail in an examination
                - E.g.

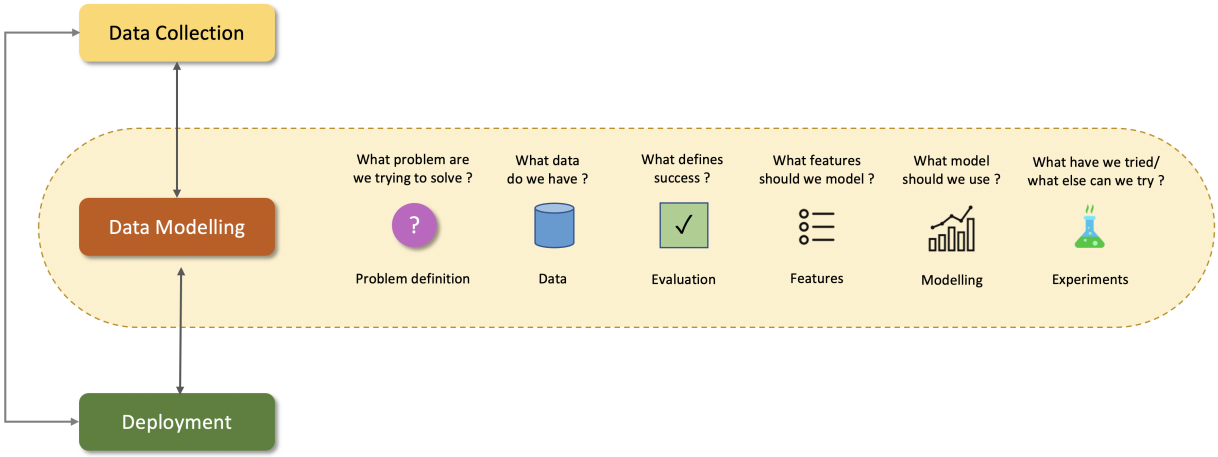
                - Logistic Regression
                - Decision Tree
                - Random Forest
                - Support Vector Machine
                - K-nearest neighbor

- **Unsupervised machine learning algorithms**
    - Are used when the information used to train is neither classified nor labeled
    - Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data
    - The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data
    - Type of algorithms
        - **Clustering**
            - Batsman vs bowler
            - Customer spending more money vs less money
            - E.g.
                - K-means clustering
                - Hierarchical clustering
        - **Association Rules Mining**
            - Market basket analysis
            - E.g.
                - Apriori
                - Eclat

- **Reinforcement machine learning algorithms**
    - Is a learning method that interacts with its environment by producing actions and discovers errors or rewards
    - Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning
    - This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance
    - Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal
    - E.g.
        - Resources management in computer clusters
        - Traffic Light Control
        - Robotics
        - Web system configuration
        - Chemistry
    - Types of algorithms

    - Q-Learning
    - Deep Q-Learning

- creating machine learning pipeling



Prepare/Transform your dataset

- **Rescale Data**
    - When your data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale
    - Often this is referred to
      as normalization and attributes are often rescaled into the range between 0 and 1
    - https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing
    - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

```
  # Rescale data (between 0 and 1)

  from pandas import read_csv
  from numpy import set_printoptions
  from sklearn.preprocessing import MinMaxScaler

  filename = 'pima-indians-diabetes.data.csv'
  names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class'] dataframe = read_csv(filename, names=names)
  array = dataframe.values

  # separate array into input and output components
  X = array[:,0:8]
  Y = array[:,8]
  scaler = MinMaxScaler(feature_range=(0, 1))
  rescaledX = scaler.fit_transform(X)
```

- **Standardize Data**

- Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1
- http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler. html

```
  # Standardize data (0 mean, 1 stdev)
  from sklearn.preprocessing import StandardScaler from pandas import
read_csv
  from numpy import set_printoptions

  filename = 'pima-indians-diabetes.data.csv'
  names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']

  dataframe = read_csv(filename, names=names)
  array = dataframe.values

  # separate array into input and output components
  X = array[:,0:8]
  Y = array[:,8]

  # summarize
  scaler = StandardScaler().fit(X)
  rescaledX = scaler.transform(X)
```

Feature selection

- Feature selection is a process where you automatically select those features in your data that contribute most to the prediction variable or output in which you are interested
- Having irrelevant features in your data can decrease the accuracy of many models, especially linear algorithms like linear and logistic regression
- Three benefits of performing feature selection before modeling your data are
  - **Reduces Overfitting**
    - Less redundant data means less opportunity to make decisions based on noise
  - **Improves Accuracy**
    - Less misleading data means modeling accuracy improves
  - **Reduces Training Time**
    - Less data means that algorithms train faster
- **PCA**
  - Principal Component Analysis (or PCA) uses linear algebra to transform the dataset into a compressed form
  - Generally this is called a data reduction technique

- A property of PCA is that you can choose the number of dimensions or principal components in the transformed result

```
  # import required packages
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns

  # load the data
  df = pd.read_csv('heart_disease.csv')
  print(df.head())

  # decide the x and y
  x = df.drop('target', axis=1)
  y = df['target']

  # split the data into train and test datasets
  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  from sklearn.decomposition import PCA

  # create pca object
  pca = PCA(n_components=2)

  # extract 2 features from 13 independent variables
  # principal_components = pca.fit_transform(x_test, y_test)
  principal_components = pca.fit_transform(x_test)
  # print(principal_components[:, 0])

  plt.scatter(principal_components[:, 0][y_prediction == 0],
principal_components[:, 1][y_prediction == 0], color="red")
  plt.scatter(principal_components[:, 0][y_prediction == 1],
principal_components[:, 1][y_prediction == 1], color="green")
  plt.xlabel('principal component 1')
  plt.ylabel('principal component 2')
```

Regression

- **Simple Linear Regression**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('Salary_Data.csv')

# check if salary and experience are related
# get the correlation coeffiecient from [0, 1] or [1, 0] position
corr = np.corrcoef(df['Salary'], df['YearsExperience'])
print(corr[0, 1])

# exploratory data analysis
plt.scatter(df['YearsExperience'], df['Salary'])
plt.xlabel('Years of experience')
plt.ylabel('Salary')
plt.title('Experience vs Salary')

# since salary is the dependent variable, removing it from
x = df.drop('Salary', axis=1)

# dependent variable
y = df['Salary']

x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=12345)

# create a model
model = LinearRegression()

# fit the training data into the model
# asking model to find the insights/patterns of the data
model.fit(x_train, y_train)

# predict the salaries for all the employees in the test data set
y_predictions = model.predict(x_test)
```

- **Multiple Linear Regression**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('Advertising.csv')
df.head()

# find relationship
corr = df.corr()
```

```
  corr.style.background_gradient(cmap = 'coolwarm')

  # visualization
  sns.heatmap(corr,cmap='coolwarm')
  sns.pairplot(df)

  #decide x and y
  x = df.drop('sales',axis =1)
  y = df['sales']

  #split data into training and testing data
  x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.8)

  # create a model
  model = LinearRegression()

  # train the model using train data set
  model.fit(x_train, y_train)

  # predict the salaries for all the employees in the test data set
  y_predictions = model.predict(x_test)
```

- **Polynomial Regression**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('Position_Salaries.csv')

# remove Position and Salary from data as x
x = df.drop(['Salary', 'Position'], axis=1)

# get Salary from data as y
y = df['Salary']

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=4)

# process all the x data values
x_polynomial = polynomial_features.fit_transform(x)

from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x_polynomial, y)
```

```
y_prediction = model.predict(x_polynomial)
```

- **Elastic Net Regression**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# read the data
df = pd.read_csv('Salary_Data.csv')

x = df.drop('Salary', axis=1)
y = df['Salary']

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

from sklearn.linear_model import ElasticNet

model = ElasticNet()
model.fit(x_train, y_train)

y_prediction = model.predict(x_test)
```

- **Ridge Regression**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('50_Startups.csv')

# decide x and y
x = df.drop(['Profit', 'State'], axis=1)
y = df['Profit']

# split the data into train and test
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)
```

```
from sklearn.linear_model import Ridge

# create the model
model = Ridge()

# train the model
model.fit(x_train, y_train)

y_prediction = model.predict(x_test)
```

- **Lasso Regression**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('50_Startups.csv')

# decide x and y
x = df.drop(['Profit', 'State'], axis=1)
y = df['Profit']

# split the data into train and test
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

from sklearn.linear_model import Lasso

# create the model
model = Lasso(alpha=2)

# train the model
model.fit(x_train, y_train)

y_prediction = model.predict(x_test)
```

- evaluating models using
  - **MAE**

```
from sklearn.metrics import mean_absolute_error
```

```
MAE = mean_absolute_error(y_test, y_predictions)
print(f"MAE = {MAE}")
```

- **MSE**

```
from sklearn.metrics import mean_squared_error

MSE = mean_squared_error(y_test, y_predictions)
print(f"MSE = {MSE}")
```

- **RMSE**

```
from sklearn.metrics import mean_squared_error

MSE = mean_squared_error(y_test, y_predictions)
RMSE = np.sqrt(MSE)
print(f"RMSE = {RMSE}")
```

- **R2 score**

```
from sklearn.metrics import r2_score

R2 = r2_score(y_test, y_predictions)
print(f"R2 = {R2}")
```

Classification

- **Logistic Regression**

```
  # import required packages
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns

  df = pd.read_csv('hearing_test.csv')

  x = df.drop('test_result', axis=1)
  y = df['test_result']

  # split the data into train and test
  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  from sklearn.linear_model import LogisticRegressionCV

  # create the model
  model = LogisticRegressionCV()

  # train the model
  model.fit(x_train, y_train)

  y_prediction = model.predict(x_test)
```

- **Support Vector Matrix (SVM)**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns

  df = pd.read_csv('hearing_test.csv')
  print(df.head())

  # decide the x and y
  x = df.drop(['test_result'], axis=1)
  y = df['test_result']

  # split the data into train and test datasets
```

```
  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  from sklearn.svm import SVC

  # create the model
  model = SVC(C=2.0)

  # train the model
  model.fit(x_train, y_train)

  y_prediction = model.predict(x_test)
```

- **Naive Bayes**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns

  df = pd.read_csv('hearing_test.csv')
  print(df.head())

  # decide the x and y
  x = df.drop(['test_result'], axis=1)
  y = df['test_result']

  # split the data into train and test datasets
  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  from sklearn.naive_bayes import GaussianNB

  # create the model
  model = GaussianNB()

  # train the model
  model.fit(x_train, y_train)

  y_prediction = model.predict(x_test)
```

- **KNN**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns

  df = pd.read_csv('hearing_test.csv')
  print(df.head())

  # decide the x and y
  x = df.drop(['test_result'], axis=1)
  y = df['test_result']

  # split the data into train and test datasets
  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  from sklearn.neighbors import KNeighborsClassifier

  # create the model
  model = KNeighborsClassifier()

  # train the model
  model.fit(x_train, y_train)

  y_prediction = model.predict(x_test)
```

- **Decision Tree**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns

  df = pd.read_csv('hearing_test.csv')
  print(df.head())

  # decide the x and y
  x = df.drop(['test_result'], axis=1)
  y = df['test_result']

  # split the data into train and test datasets
  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)
```

```
from sklearn.tree import DecisionTreeClassifier

# create the model
model = DecisionTreeClassifier()

# train the model
model.fit(x_train, y_train)

y_prediction = model.predict(x_test)
```

- Evaluation model
  - **Confusion Matrix**

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_prediction)
print(cm)
```

  - **Accuracy Score**

```
accuracy = (cm[0, 0] + cm [1, 1]) / (cm[0, 0] + cm[0, 1] + cm [1, 1] + cm[1, 0])
print(f"accuracy = {accuracy * 100: 0.2f}%")

from sklearn.metrics import accuracy_score
print(f"accuracy = {accuracy_score(y_test, y_prediction) * 100: 0.2f}%")
```

  - **Precision Score**

```
from sklearn.metrics import precision_score

precision = precision_score(y_test, y_prediction)
print(f"precision = {precision}")
```

  - **Recall Score**

```
from sklearn.metrics import recall_score

recall = recall_score(y_test, y_prediction)
print(f"recall = {recall}")
```

- **F1 Score**

```
from sklearn.metrics import f1_score

f1 = f1_score(y_test, y_prediction)
print(f"f1 score = {f1}")
```

- **Classification Report**

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_prediction))
```

- **RoC Score**

```
# RoC score
from sklearn.metrics import roc_auc_score
print(f"roc auc score = {roc_auc_score(y_test, y_prediction)}")
```

- **AuC**

```
# AuC
from sklearn.metrics import plot_roc_curve

plot_roc_curve(model, x_test, y_test)
plt.plot([0, 1], [0, 1], linestyle="--")
```

Ensemble Learning

- **bagging**
  - **Random Forest**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns

  df = pd.read_csv('hearing_test.csv')
  print(df.head())

  # decide the x and y
  x = df.drop(['test_result'], axis=1)
  y = df['test_result']

  # split the data into train and test datasets
  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  from sklearn.ensemble import RandomForestClassifier

  # create the model
  model = RandomForestClassifier(n_estimators=100)

  # train the model
  model.fit(x_train, y_train)

  y_prediction = model.predict(x_test)
```

- **boosting**
  - **AdaBoost**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt

  df = pd.read_csv('heart_disease.csv')
  df.head()

  x = df.drop(['trestbps', 'chol', 'restecg', 'fbs', 'target'],
axis=1)
```

```
  y = df['target']

  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  from sklearn.ensemble import AdaBoostClassifier

  model = AdaBoostClassifier(n_estimators=10)
  model.fit(x_train, y_train)

  y_prediction = model.predict(x_test)
```

- **Cat Boost**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  from catboost.datasets import titanic

  train_df, test_df = titanic()
  train_df.head()

  # find the NA records
  null_values = train_df.isnull().sum()

  # remove the null values with some high value
  train_df.fillna(-999, inplace=True)
  test_df.fillna(-999, inplace=True)

  x = train_df.drop('Survived', axis=1)
  y = train_df['Survived']

  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  # find the categorical features
  # print(train_df)
  categorical_features_index = np.where(x.dtypes != np.float)[0]

  from catboost import CatBoostClassifier, metrics
  model = CatBoostClassifier(logging_level='Silent')
  model.fit(x_train, y_train,
cat_features=categorical_features_index)

  y_prediction = model.predict(x_test)
```

- **XGBoost**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt

  df = pd.read_csv('heart_disease.csv')
  df.head()

  x = df.drop(['trestbps', 'chol', 'restecg', 'fbs', 'target'],
axis=1)
  y = df['target']

  from sklearn.model_selection import train_test_split

  x_train, x_test, y_train, y_test = train_test_split(x, y,
train_size=0.8, random_state=123456)

  from xgboost import XGBClassifier

  model = XGBClassifier(n_estimators=500)
  model.fit(x_train, y_train)

  y_prediction = model.predict(x_test)
```

Clustering

- **KMeans Clustering**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns

  # read the data
  df = pd.read_csv('Mall_Customers.csv')
  x = df.drop(['CustomerID', 'Gender', 'Age'], axis=1)

  from sklearn.cluster import KMeans

  # used to collect the wss values for different values of k
  wss = []

  for k in range(1, 11):
      # create kmeans object for n_cluster = k
      clusters = KMeans(n_clusters=k)
      clusters.fit(x)
      # collect the wss for every cluster
      wss.append(clusters.inertia_)

  print(wss)


  k_values = np.arange(1, 11)
  plt.plot(k_values, wss, label="Elbow line")
  plt.scatter(k_values, wss)
  plt.xlabel('K value')
  plt.ylabel('WSS')
  plt.title("Elbow Method")
  plt.legend()

  # visualization
  clusters = KMeans(n_clusters=5)
  clusters.fit(x)
  print(clusters.labels_)
  plt.scatter(df['Annual Income (k$)'][clusters.labels_ == 0],
df['Spending Score (1-100)'][clusters.labels_ == 0], color="red")
  plt.scatter(df['Annual Income (k$)'][clusters.labels_ == 1],
df['Spending Score (1-100)'][clusters.labels_ == 1], color="green")
  plt.scatter(df['Annual Income (k$)'][clusters.labels_ == 2],
df['Spending Score (1-100)'][clusters.labels_ == 2], color="blue")
  plt.scatter(df['Annual Income (k$)'][clusters.labels_ == 3],
df['Spending Score (1-100)'][clusters.labels_ == 3], color="magenta")
  plt.scatter(df['Annual Income (k$)'][clusters.labels_ == 4],
df['Spending Score (1-100)'][clusters.labels_ == 4], color="black")
```

```
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Clustering')
```

- **Hierarchical Clustering**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


df = pd.read_csv('Mall_Customers.csv')

# clean the data
df = df.drop(['CustomerID', 'Age', 'Gender'], axis=1)

### create the clusters
from sklearn.cluster import AgglomerativeClustering

# create the hiearchical model
hiearchical = AgglomerativeClustering(n_clusters=5)

# fit the data into the clusters
clusters = hiearchical.fit_predict(df)
print(clusters)

plt.figure(figsize=(15, 8))
colors = ['red', 'green', 'blue', 'brown', 'magenta']
markers = ['.', '^', '1', 's', 'd']

# create clusters
for k in range(5):
  plt.scatter(
  df['Annual Income (k$)'][clusters == k],
  df['Spending Score (1-100)'][clusters == k],
  label=f"cluster {k}",
  color=colors[k],
  marker=markers[k])

plt.legend()

# visualization

from scipy.cluster.hierarchy import dendrogram
from scipy.cluster import hierarchy

linkage_matrix = hierarchy.linkage(hiearchical.children_)

plt.figure(figsize=(20, 10))
```

```
    dn = dendrogram(linkage_matrix)
```

Association Rule Mining

- **apriori**

```
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  import seaborn as sns


  # load the data
  # the transactions file does not have any header
  # the first row is not the header but rather it is a transaction
having number of product
  df = pd.read_csv('Market_Basket_Optimisation.csv', header=None)
  # print(df.columns)

  # collect all transactions
  transactions = []

  # number of transactions
  rows = df.shape[0]

  # fun a loop over the transactions and collect every transaction
  for row in range(rows):
      # get all the columns from the row
      items = df.iloc[row, :]

      # collect all the items in a transaction
      transaction = []
      for item in items:
          if not pd.isna(item):
              transaction.append(item)

      # append the transaction to transactions
      transactions.append(transaction)


  from apyori import apriori

  rules = list(apriori(transactions, min_support=0.15))
  for rule in rules:
      print(rule.items)
```