Python初級數據分析員證書

（六）數據分析 及可視化專案

13. 數據分析專案 Demo 17 – Cancer Type Prediction

Review

- Statistics
- Hypothesis testing
- Algebra
- Linear regression
- Propositional logic
- Python
- R
- SQL

- Pandas, NumPy, SciPy
- Data Visualization, Matplotlib, Seaborn, Plotly
- Dashboard Visualization, Business Intelligence
- Storytelling

Chapter Summary

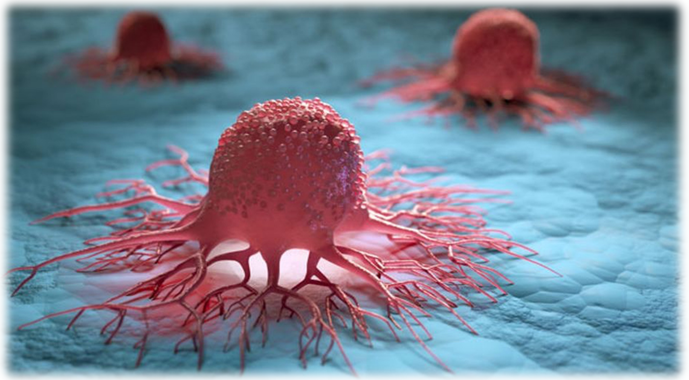
- Scenario
- Data Import
- Data Wrangling
- EDA
- Model Train
- Type Prediction

Scenario

In the dataset, there are 570 cancer cells and 30 features to determine whether the cancer cells in our data are benign or malignant.

Our cancer data contains 2 types of cancers:

1. **Benign 良性 cancer (B) and**
2. **Malignant 惡性 cancer (M).**



Data import

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.model_selection import train_test_split
6 from sklearn import preprocessing,metrics
```

```
1 pd.set_option('display.max_columns',None)
2 df = pd.read_csv("Cancer_Data.csv")
3 df.sample(3)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
69	859487	B	12.78	16.49	81.37	502.5	0.09831
5	843786	M	12.45	15.70	82.57	477.1	0.12780
225	88143502	B	14.34	13.47	92.51	641.2	0.09906

Overview all columns

The column ‘diagnosis’ is the target to analyse and predict.

```
1 df.shape
```

```
(569, 33)
```

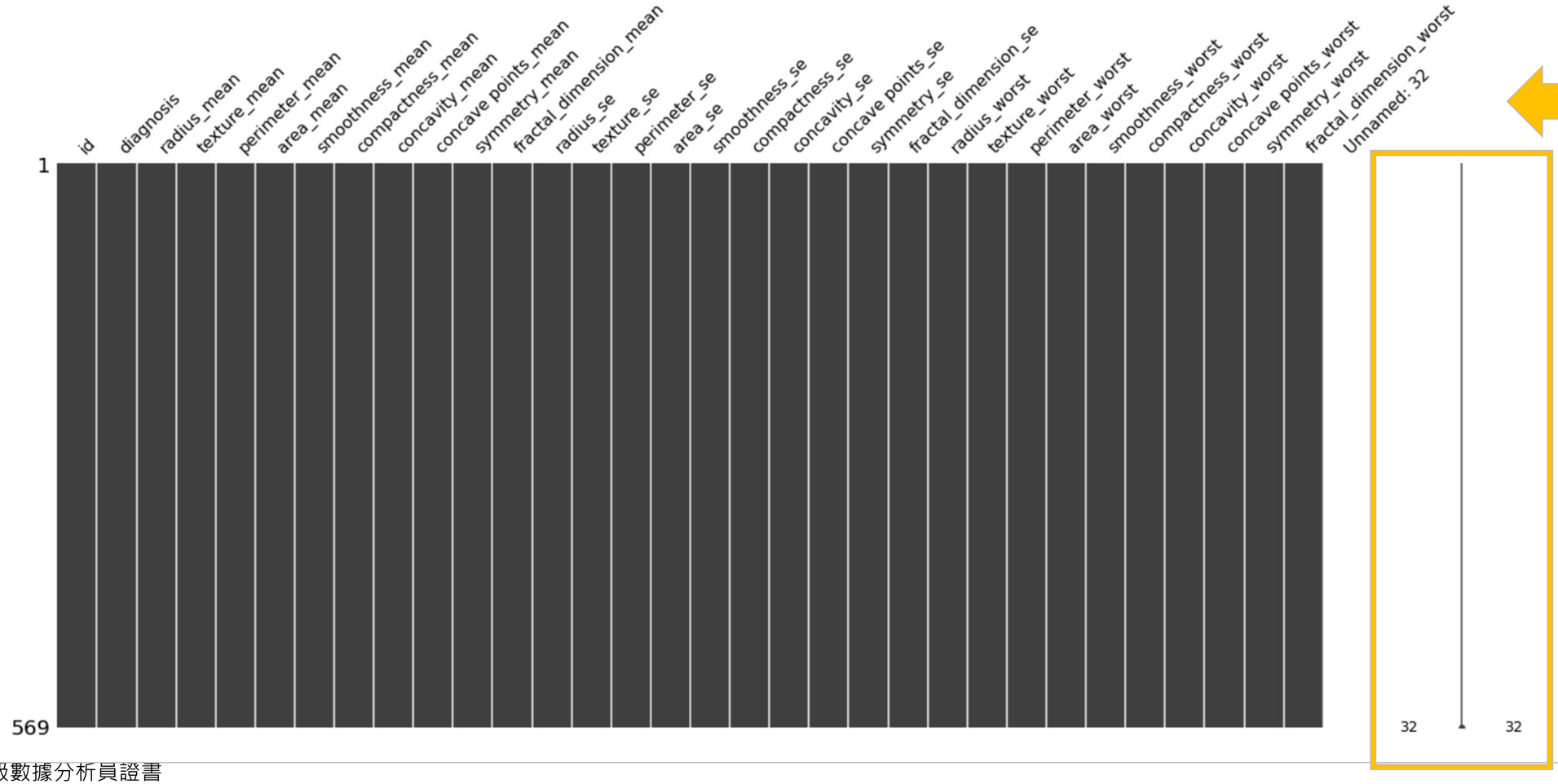
```
1 df.columns
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

Check NaN and Null with missingno

```
1 import missingno as msno
2 msno.matrix(df)
```

```
<AxesSubplot:>
```



Unnamed: 32

Data cleaning

```
1 df['Unnamed: 32']
```

```
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
       ..
564    NaN
565    NaN
566    NaN
567    NaN
568    NaN
Name: Unnamed: 32, Length: 569, dtype: float64
```

This column was filled with all NaN

```
1 df['Unnamed: 32'].unique()
```

```
array([nan])
```

Drop the unnecessary columns

```
1 df['Unnamed: 32'].unique()
```

```
array([nan])
```

```
1 # the id column is irrelevant to our analysis
2 df['id'].nunique()
```

```
569
```

```
1 # Drop the unnecessary columns
2 df = df.drop(["id","Unnamed: 32"],axis = 'columns')
3 df.shape
```

```
(569, 31)
```

Label encoding

```
1 encoder = preprocessing.LabelEncoder()
2 df['diagnosis'] = encoder.fit_transform(df['diagnosis'])
3 df.head(3)
```

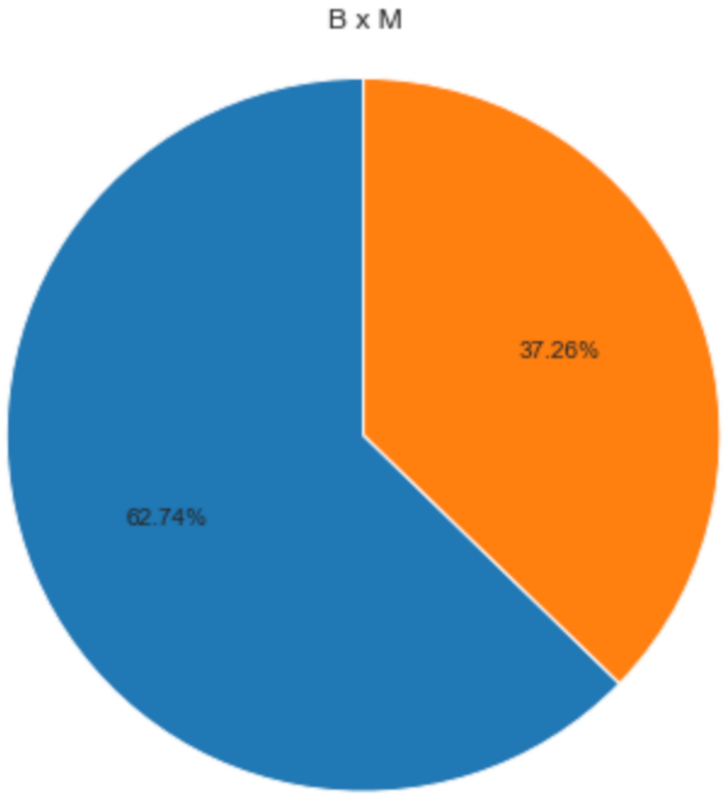
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
0	1	17.99	10.38	122.8	1001.0	0.11840	0.27760
1	1	20.57	17.77	132.9	1326.0	0.08474	0.07864
2	1	19.69	21.25	130.0	1203.0	0.10960	0.15990

```
1 df['diagnosis'].value_counts()
```

```
0    357
1    212
Name: diagnosis, dtype: int64
```

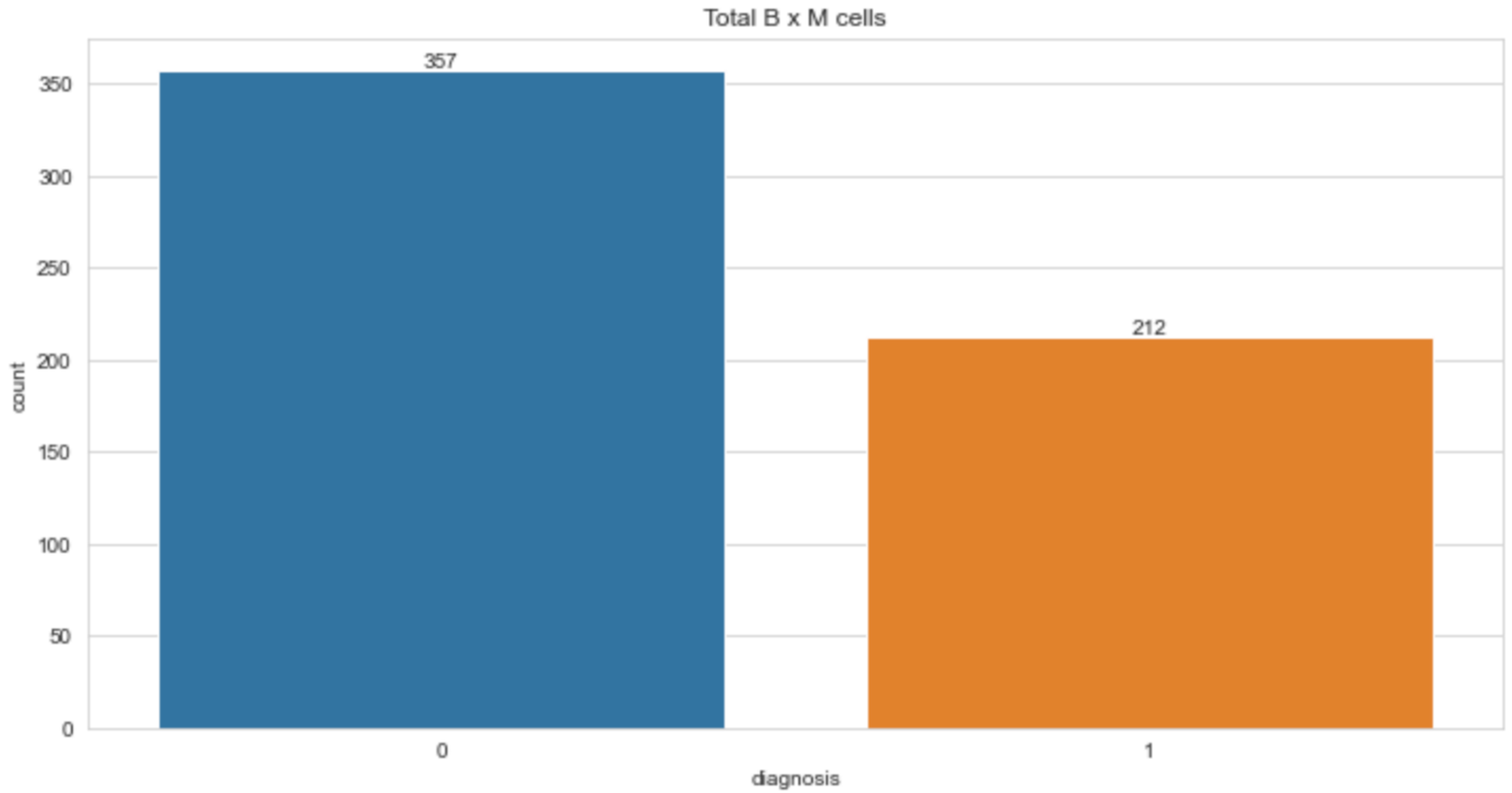
EDA

```
plt.figure(figsize=(10,6))
sns.set_style("whitegrid")
plt.pie(df['diagnosis'].value_counts(),autopct='%1.2f%%', startangle=90)
plt.axis('equal')
plt.title("B x M")
plt.show()
```



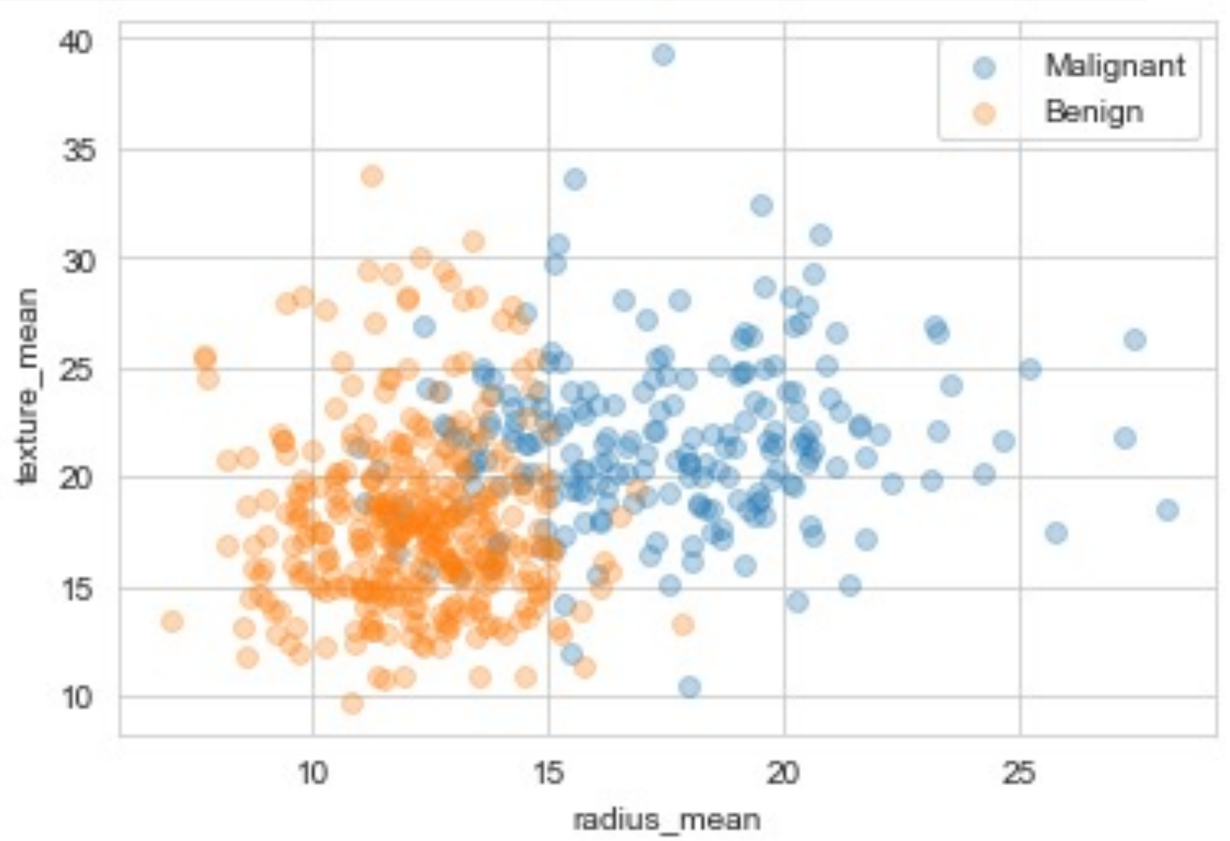
Benign vs Malignant

```
1  plt.figure(figsize=(12,6))
2  ax = sns.countplot(data=df, x='diagnosis')
3  plt.title('Total B x M cells')
4
5  ax.bar_label(ax.containers[0], label_type='edge')
6  plt.show()
```



radius_mean and texture_mean

```
1  M = df[df.diagnosis == 1] #Diagnosis transfers all values of M to M data
2  B = df[df.diagnosis == 0] #Diagnosis transfers all values of B to B data
3
4  plt.scatter(M.radius_mean,M.texture_mean, label = "Malignant", alpha = 0.3)
5  plt.scatter(B.radius_mean,B.texture_mean,label = "Benign", alpha = 0.3)
6
7  plt.xlabel("radius_mean")
8  plt.ylabel("texture_mean")
9
10 plt.legend()
11 plt.show()
```

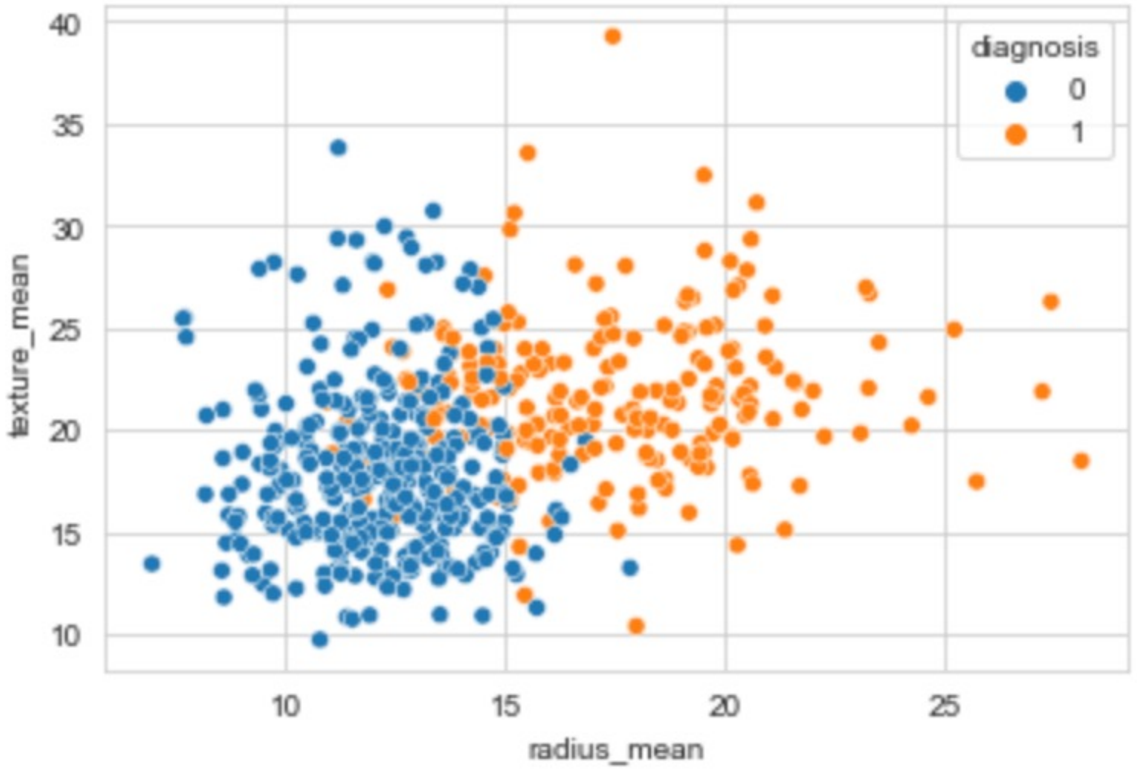


radius_mean and texture_mean

We can also plot with seaborn

```
1 import seaborn as sns
2 sns.scatterplot(data=df, x="radius_mean",
3                 y="texture_mean", hue="diagnosis")
```

```
<AxesSubplot:xlabel='radius_mean', ylabel='texture_mean'>
```



radius_mean and texture_mean

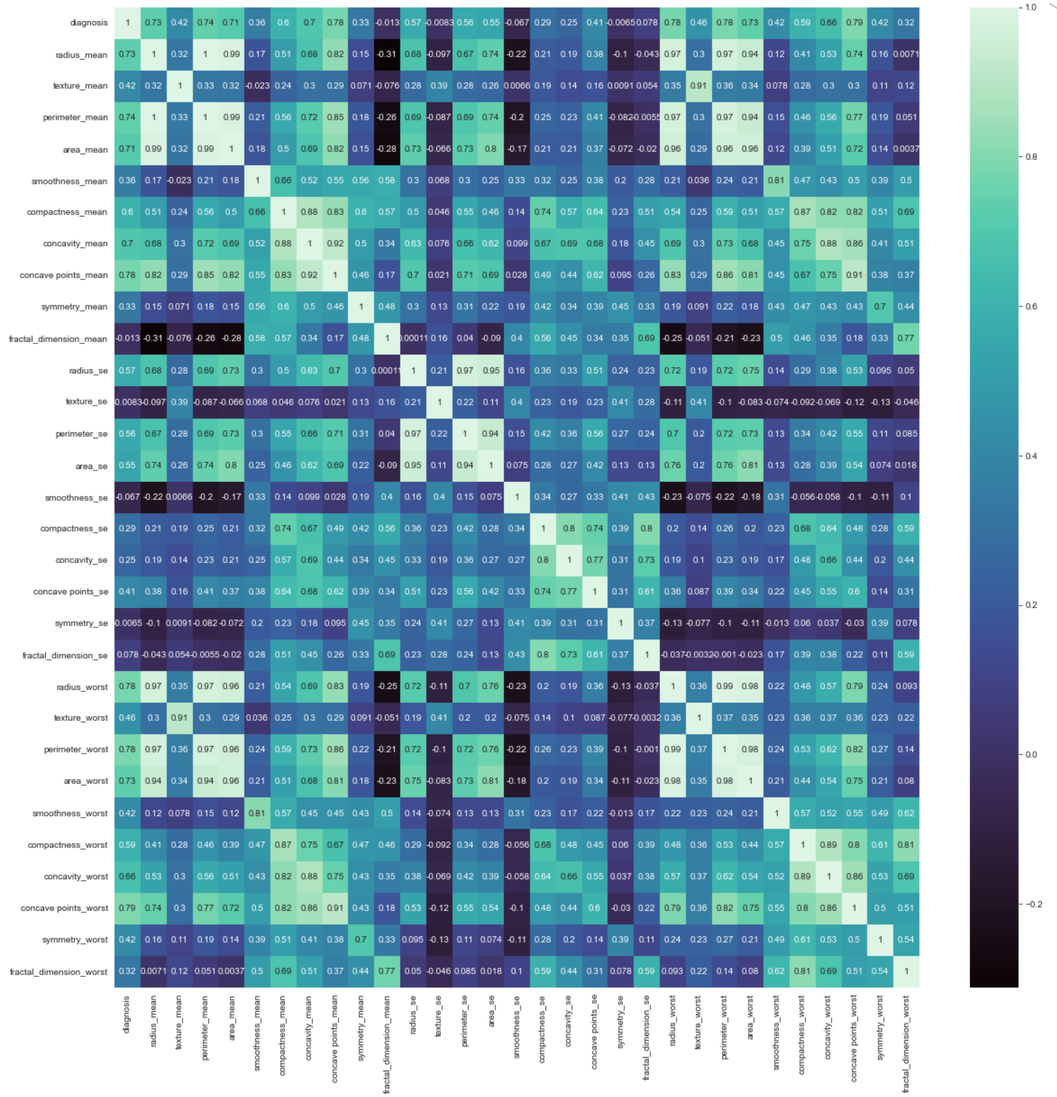
```
1 df.groupby('diagnosis')[['radius_mean','texture_mean']].mean()
```

diagnosis	radius_mean	texture_mean
0	12.146524	17.914762
1	17.462830	21.604906

Malignant has higher value in both radius_mean and texture_mean

Heatmap

```
1  plt.figure(figsize=(20,20))
2  sns.heatmap(df.corr(),cbar=True,annot=True,cmap='mako')
3  plt.show()
```



df.corrwith()

```
1 df.corrwith(df.diagnosis).sort_values(ascending=False)
```

```
diagnosis                 1.000000
concave points_worst      0.793566
perimeter_worst           0.782914
concave points_mean       0.776614
radius_worst              0.776454
perimeter_mean            0.742636
area_worst                0.733825
radius_mean               0.730029
area_mean                 0.708984
concavity_mean            0.696360
concavity_worst           0.659610
compactness_mean          0.596534
compactness_worst         0.590998
radius_se                 0.567134
perimeter_se              0.556141
area_se                   0.548236
texture_worst             0.456903
smoothness_worst          0.421465
symmetry_worst            0.416294
texture_mean              0.415185
concave points_se         0.408042
smoothness_mean           0.358560
symmetry_mean             0.330499
fractal_dimension_worst   0.323872
compactness_se            0.292999
concavity_se              0.253730
fractal_dimension_se      0.077972
symmetry_se              -0.006522
texture_se               -0.008303
fractal_dimension_mean   -0.012838
smoothness_se            -0.067016
dtype: float64
```

If there was significant negative relation, we should keep an eye on.

Building model

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, plot_confusion_matrix

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
X = df.drop("diagnosis", axis=1)
y = df['diagnosis']
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.33, random_state=43, stratify=y)
```

```
model_dict = {}
```

Linear Discriminant Analysis

Linear Discriminant Analysis

```
1 classifer = LinearDiscriminantAnalysis()
2 predictor = classifer.fit(X_train, y_train)
3 y_pred = predictor.predict(X_val)
4 accuracy_lda = accuracy_score(y_val, y_pred)
5 model_dict['linear_discriminant_analysis'] = accuracy_lda
6 print(accuracy_lda)
```

```
0.9680851063829787
```

Random Forest Classifier

Random Forest Classifier

```
1 classifier = RandomForestClassifier(random_state=42)
2 predictor = classifier.fit(X_train, y_train)
3 y_pred = predictor.predict(X_val)
4 accuracy_rfc = accuracy_score(y_val, y_pred)
5 model_dict['random_forest_classifier'] = accuracy_rfc
6 print(accuracy_rfc)
```

```
0.9627659574468085
```

XGboost

```
1 classifier = XGBClassifier(random_state=42, eval_metric='logloss')
2 predictor_xgb = classifier.fit(X_train, y_train)
3 y_pred = predictor_xgb.predict(X_val)
4 accuracy_xgb = accuracy_score(y_val, y_pred)
5 model_dict['xgboost_classifier'] = accuracy_xgb
6 print(accuracy_xgb)
```

```
0.973404255319149
```

Accuracy Summary

Model Accuracy Summary

```
1 model_accuracies_df = pd.DataFrame(columns=['Model', 'Accuracy'])
2 model_accuracies_df['Model'] = model_dict.keys()
3 model_accuracies_df['Accuracy'] = model_dict.values()
4 model_accuracies_df
```

	Model	Accuracy
0	linear_discriminant_analysis	0.968085
1	random_forest_classifier	0.962766
2	xgboost_classifier	0.973404

Multiple Model Testing

To run model one by one, alternatively we can create a loop to run all.

```
1 from sklearn.model_selection import cross_val_score
2 from sklearn.ensemble import BaggingClassifier
3 from sklearn.tree import DecisionTreeClassifier
4 from sklearn.svm import SVC
5 from sklearn.linear_model import LogisticRegression
6 from tqdm.notebook import tqdm
```

```
1 X = df.drop("diagnosis", axis=1)
2 y = df['diagnosis']
```

```
1 X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
2 X_train.shape,X_test.shape
```

```
((455, 30), (114, 30))
```

Multiple Model Testing

Write a for loop and set up each model's parameter.

```
algos = [RandomForestClassifier(random_state=42),
         BaggingClassifier(random_state=42),
         DecisionTreeClassifier(random_state=42),
         SVC(random_state=42),
         LogisticRegression(),
         LinearDiscriminantAnalysis()]
for algo in tqdm(algos):
    print(str(algo))
    cs = cross_val_score(algo,X,y)
    print("cross_val_score ",cs)
    print("average ",sum(cs)/len(cs))
    print('')

```

Multiple Model Testing

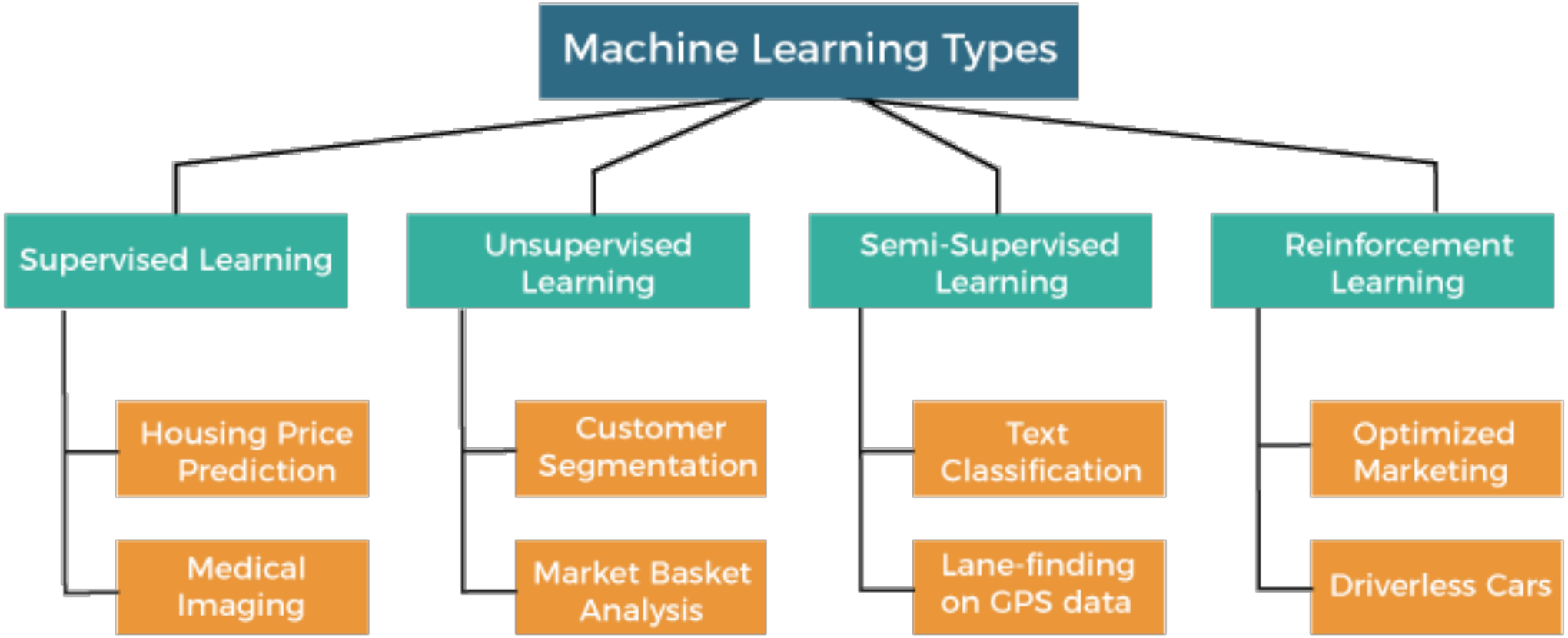

```
RandomForestClassifier(random_state=42)
cross_val_score  [0.92105263 0.93859649 0.98245614 0.96491228 0.97345133]
average  0.9560937742586555

BaggingClassifier(random_state=42)
cross_val_score  [0.92982456 0.92982456 0.97368421 0.95614035 0.97345133]
average  0.952585002328831

DecisionTreeClassifier(random_state=42)
cross_val_score  [0.9122807  0.90350877 0.92982456 0.95614035 0.88495575]
average  0.9173420276354604

SVC(random_state=42)
cross_val_score  [0.85087719 0.89473684 0.92982456 0.94736842 0.9380531 ]
average  0.9121720229777983

LogisticRegression()
cross_val_score  [0.93859649 0.93859649 0.96491228 0.94736842 0.95575221]
average  0.9490451793199813

LinearDiscriminantAnalysis()
cross_val_score  [0.95614035 0.96491228 0.94736842 0.96491228 0.96460177]
average  0.9595870206489675
```

Which model is the best?

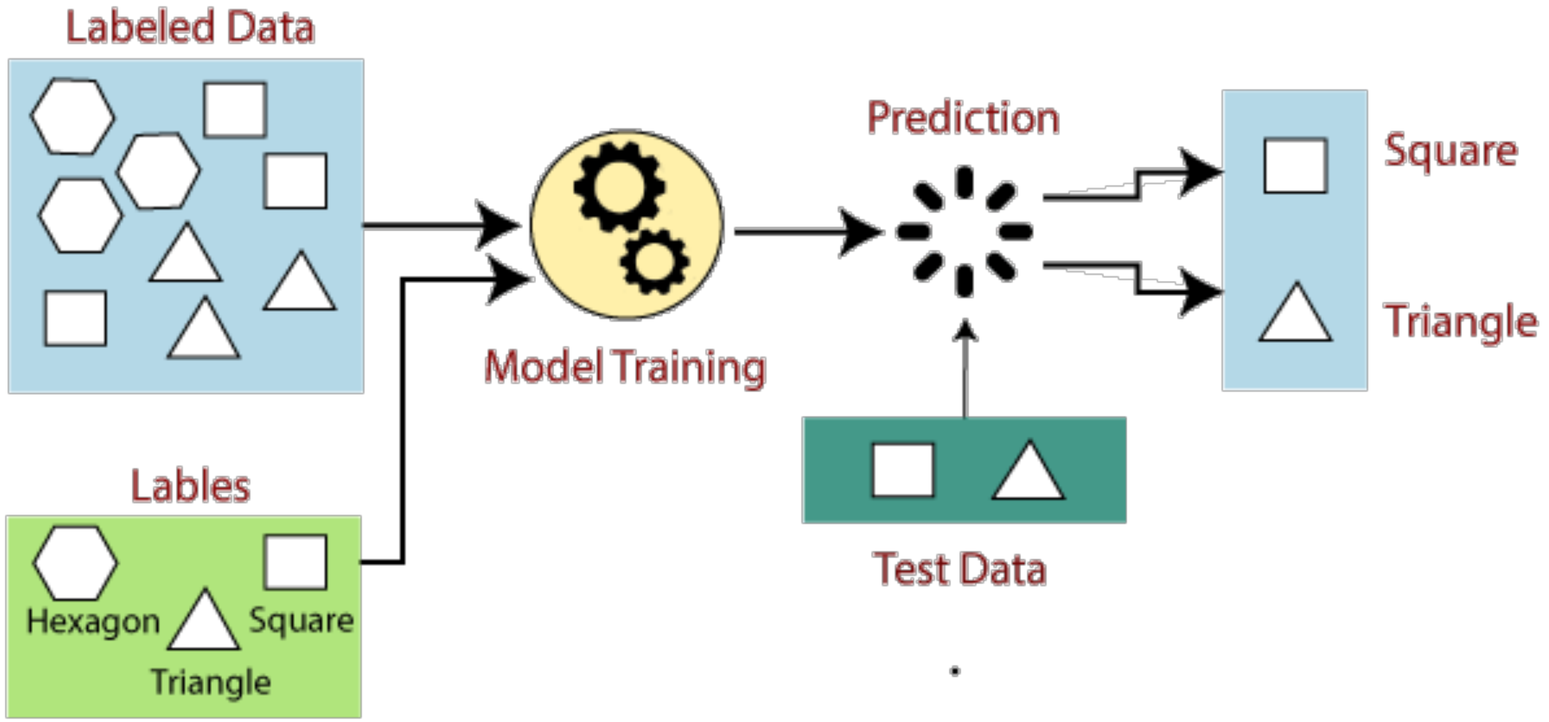
There is no the best. Each model has its unique characteristic.

Also, there are lots of parameter you can set. All come with different outcome. The proper way is test the possible kind of model and find a set of arguments.

Machine Learning Types

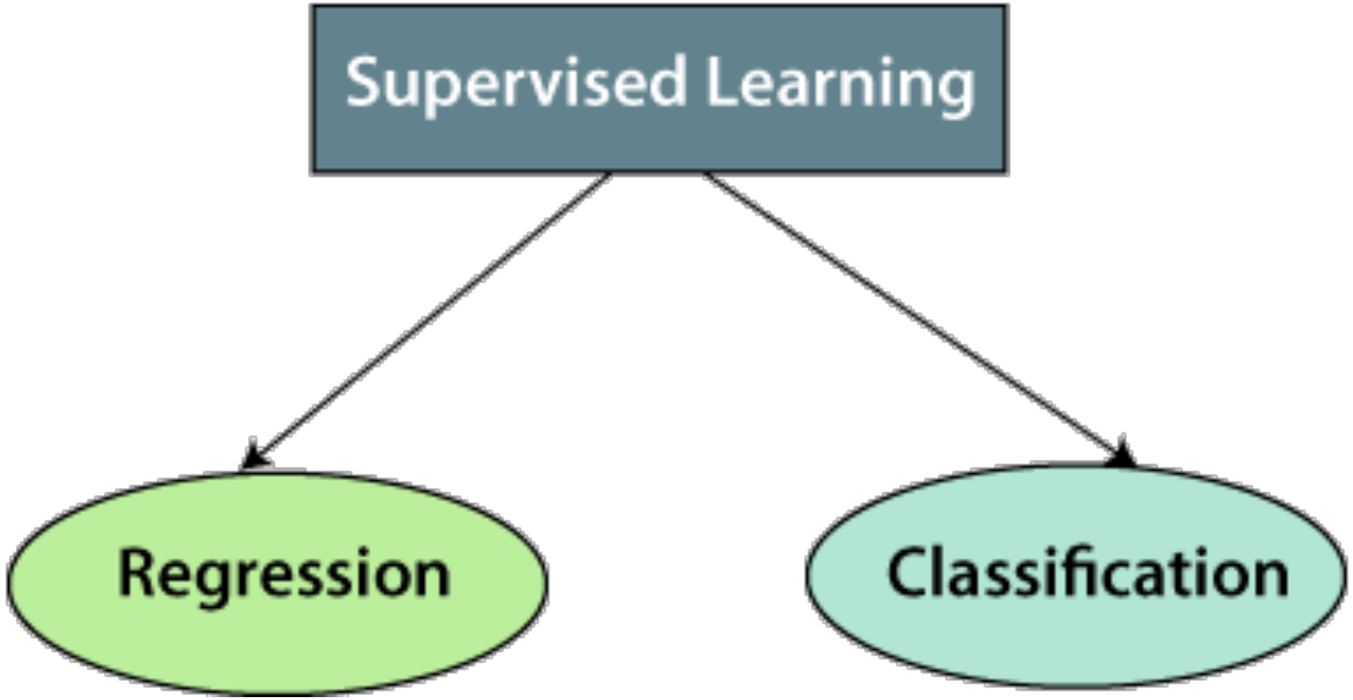


Supervised Learning



Supervised Learning

Supervised learning can be further divided into two types of problems:



Regression

1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

Classification

2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, Spam Filtering, True-false, etc.

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Advantages of Supervised learning

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as **fraud detection, spam filtering**, etc.

Disadvantages of supervised learning

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

Unsupervised Machine Learning

Unsupervised Machine Learning

Unsupervised learning is different from the Supervised learning technique; as its name suggests, there is no need for supervision. It means, in unsupervised machine learning, the machine is trained using the unlabeled dataset, and the machine predicts the output without any supervision.

In unsupervised learning, the models are trained with the data that is neither classified nor labelled, and the model acts on that data without any supervision.

Applications of Unsupervised Learning

Applications of Unsupervised Learning

•**Network Analysis:** Unsupervised learning is used for identifying plagiarism and copyright in document network analysis of text data for scholarly articles.

•**Recommendation Systems:** Recommendation systems widely use unsupervised learning techniques for building recommendation applications for different web applications and e-commerce websites.

•**Anomaly Detection:** Anomaly detection is a popular application of unsupervised learning, which can identify unusual data points within the dataset. It is used to discover fraudulent transactions.

•**Singular Value Decomposition:** Singular Value Decomposition or SVD is used to extract particular information from the database. For example, extracting information of each user located at a particular location.

Reinforcement Learning

Reinforcement Learning

Reinforcement learning works on a feedback-based process, in which an AI agent (A software component) automatically explore its surrounding by hitting & trail, taking action, learning from experiences, and improving its performance. Agent gets rewarded for each good action and get punished for each bad action; hence the goal of reinforcement learning agent is to maximize the rewards.

Reference & Resources

Sklearn Website:

https://scikit-learn.org/

Plotly Graph Objects:

https://plotly.com/python/graph-objects/

Seaborn:

https://seaborn.pydata.org/examples/index.html

Matplotlib:

https://matplotlib.org/