Study Unit 3

Training and Testing Machine Learning Models

Training and Testing Machine Learning Models Outline

- Scikit-learn module
- Data preprocessing
- Training and models evaluation
- Competing on Kaggle

Study Unit Duration

This Study Session requires a minimum of 3 hours' formal study time.

You may spend an additional 2-3 hours on revision.

Preamble

Scikit-learn is a library in Python that provides many supervised learning and unsupervised algorithms. It is built upon some of the packages you already familiar with, like NumPy, Pandas, and Matplotlib. With Scikit-learn module, you can train different machine learning models such as regression and classification and check their performance using any of the metrics discussed in unit 2.

Learning Outcomes of Study Unit 3

Upon completion of this study unit, you should be able to:

3.1 Train and evaluate classification models for predicting unknown categorical label and Solving problem using exploratory data analysis techniques.

3.2 Train and evaluate Logistic regression models for predicting unknown continuous label

3.3 Train and evaluate Logistic regression models for predicting unknown continuous label and how to select best model among the trained models

3.4 Compete on Kaggle for machine learning and data science competition

Terminologies, Acronyms and their Meaning

AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
DL	Deep learning
EDA	Exploratory Data Analysis
np	NumPy
sns	Seaborn

pd	Pandas
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative
RMSE	Root Mean Squared Error

3.1 Introduction to machine learning module: The Scikit-learn

The functionality that scikit-learn provides include:

- Regression
- Classification
- Clustering
- Model selection
- Preprocessing

Installation

The easiest way to install scikit-learn is by running the following on your terminal:

```
pip install -U scikit-learn
```

or

```
conda install -c conda-forge scikit-learn
```

Importing Scikit-learn module for classification models

Some of the classification models that can be imported from sklearn library includes:

- **Logistic Regression**: from sklearn.linear_model import LogisticRegression
- **K Nearest Neighbor**: from sklearn.neighbors import KNeighborsClassifier
- **Support Vector Machine**: from sklearn.svm import SVC
- **Decision Trees Classifier**: from sklearn.tree import DecisionTreeRegressor
- **Random Forest Classifier**: from sklearn.ensemble import RandomForestClassifier
- **Gradient Boost Classifier**: from sklearn.ensemble import GradientBoostingClassifier

Importing Scikit-learn module for regression models

Some of the regression models that can be imported from sklearn library includes:

- **Linear Regression**: from sklearn.linear_model import LinearRegression
- **K Nearest Neighbor Regressor**: from sklearn.neighbors import KNeighborsRegressor
- **Support Vector Machine**: from sklearn.svm import SVR
- **Decision Trees Regressor**: from sklearn.tree import DecisionTreeRegressor
- **Random Forest Regressor**: from sklearn.ensemble import RandomForestRegressor
- **Gradient Boost Regressor**: from sklearn.ensemble import GradientBoostingRegressor

3.1.1 Classification Machine Learning Model with Health Provider dataset in Ethiopia

Problem statement

You work as an analyst in the marketing department of a company that provides various medical insurance in Ethiopia. Your manager is unhappy with the low sales volume of a specific kind of insurance. The data engineer provides you with a sample dataset for those that visit the company website for medical insurance.

The dataset contains the following columns:

- User ID
- Gender
- Age

- Salary
- Purchase: An indicator of whether the users purchased (1) or not-purchased (0) a particular product.

We plan to use the following classifier to predict whether a person that visits the insurance company will buy or not.

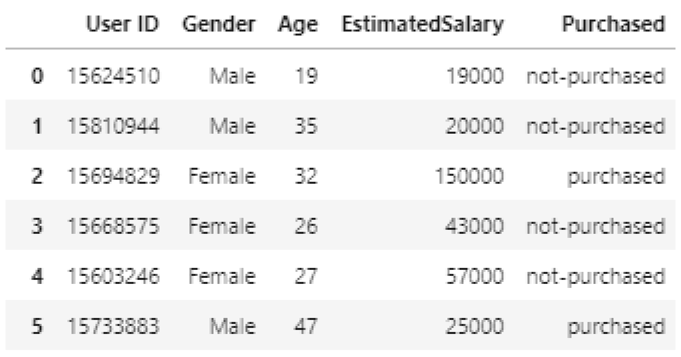
- Logistic regression
- Random forest
- Naive Bayes
- XGBoost
- Support Vector Machine (SVM)

Import Python modules

We need to import some packages that will enable us to explore the data and build machine learning models

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

```
insurance = pd.read_csv("datasets/Medical_insurance_dataset.csv")
insurance.head(5)
```



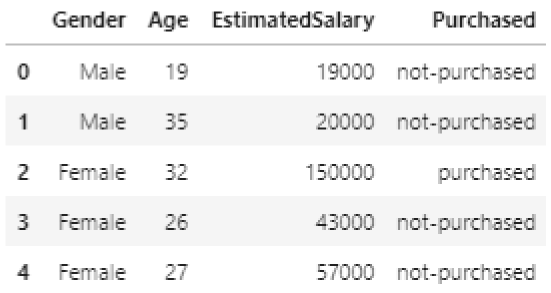
	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	not-purchased
1	15810944	Male	35	20000	not-purchased
2	15694829	Female	32	150000	purchased
3	15668575	Female	26	43000	not-purchased
4	15603246	Female	27	57000	not-purchased
5	15733883	Male	47	25000	purchased

```
insurance.shape
```

(400, 5)

We have 5 variables and 400 instances of those that want to buy medical insurance or not in this data. The User ID is a random number generated for every customer to comes to the company for medical insurance. Therefore, it is not useful in predicting whether the person will buy medical insurance or not. We will therefore, remove that variable from the data.

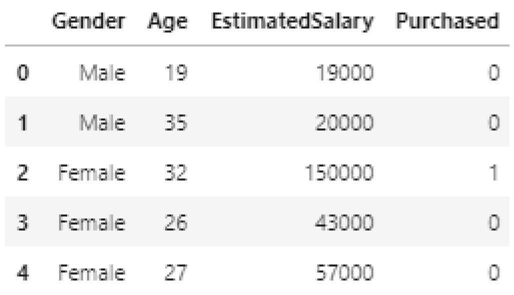
```
insurance.drop(["User ID"], axis= "columns", inplace = True)

insurance.head()
```



	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	not-purchased
1	Male	35	20000	not-purchased
2	Female	32	150000	purchased
3	Female	26	43000	not-purchased
4	Female	27	57000	not-purchased

We want to transform or recode the label Purchased to have 1 for those that bought the insurance and 0 for those that did not purchased the insurance. This will transform the output variable (label) to be numeric.

```
insurance["Purchased"] = insurance["Purchased"].apply(lambda x: 1 if x == "pu
rchased" else 0)

insurance.head()
```



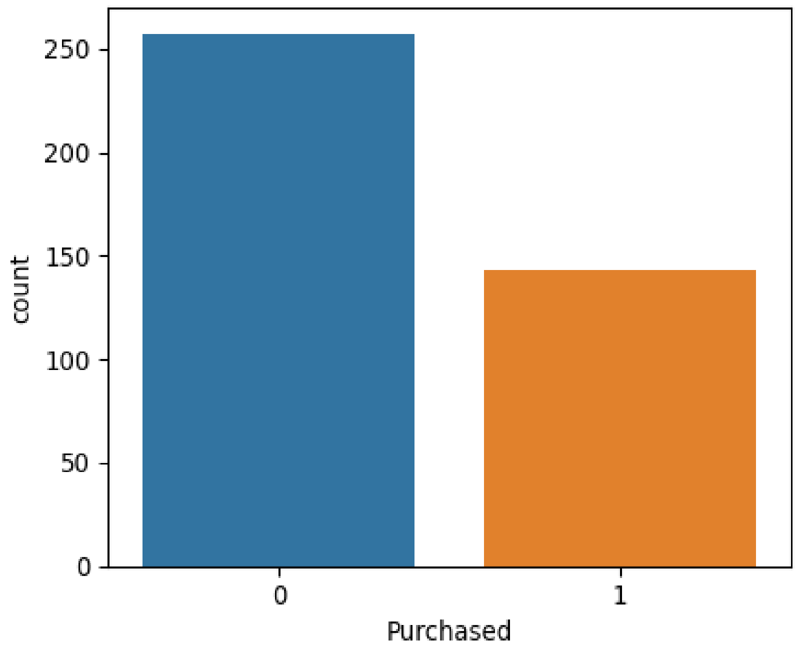
	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	32	150000	1
3	Female	26	43000	0
4	Female	27	57000	0

Now we have 3 features that include **gender**, **age**, and **estimated salary** while **purchased** is the label in this data. Since the label has just two classes or categories (purchased (1) and not-purchased (0)), this is a binary classification problem.

3.1.2 Exploratory Data Analysis

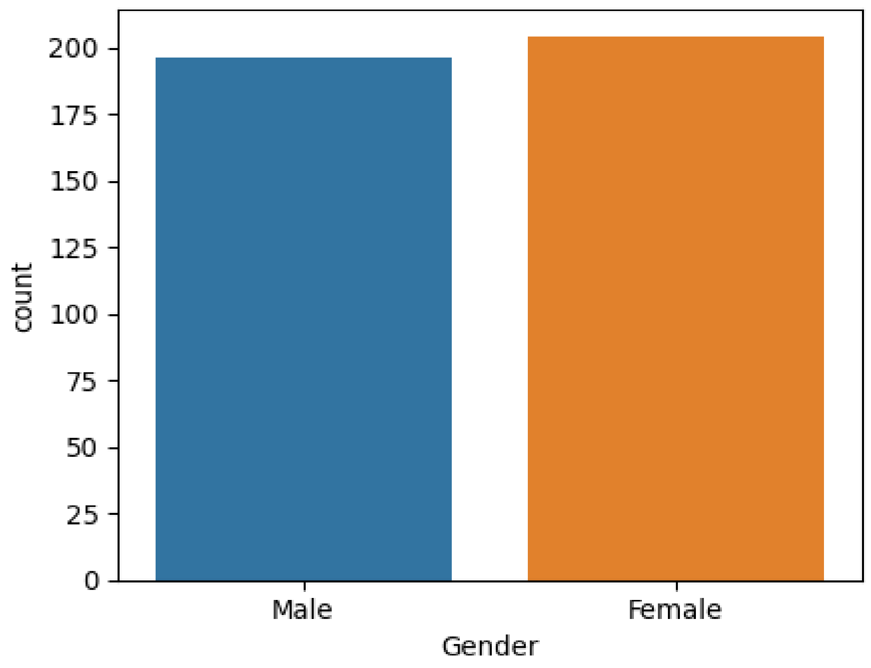
Fact generated by data exploratory will help us to know those features that can predict whether a person will purchase medical insurance or not. Let us start by visualizing the proportion of those that want to buy medical insurance or not.

```
sns.countplot(x = "Purchased", data = insurance);
```



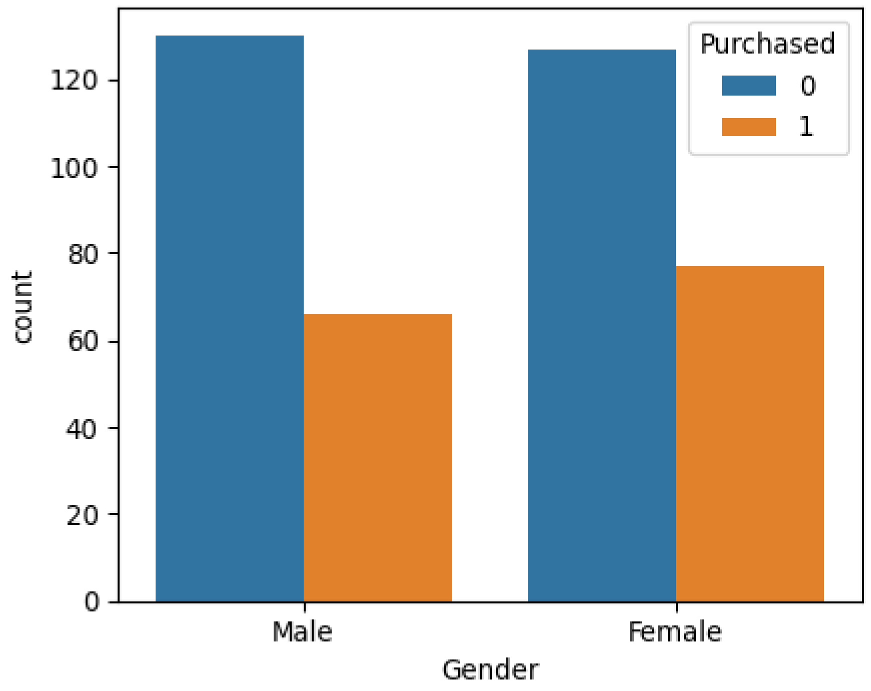
As you can see, majority of those that visit the medical insurance company did not want to buy the insurance. This is an example of class imbalanced. That is, there is no equal proportion of those that will buy or not.

```
sns.countplot(x = "Gender", data = insurance);
```



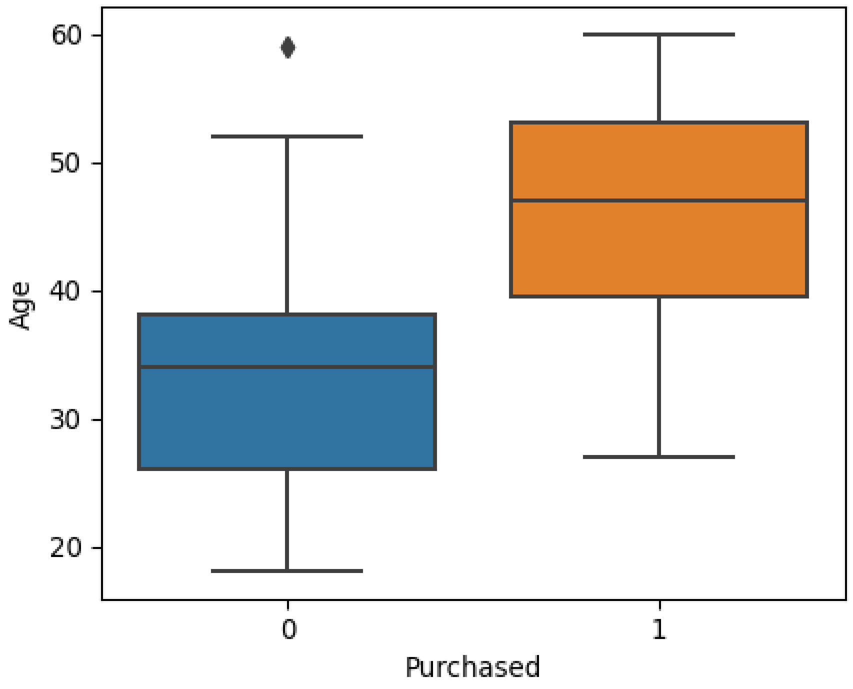
The proportion of males are almost the same as females.

```
sns.countplot(x = "Gender", hue = "Purchased", data = insurance)
```



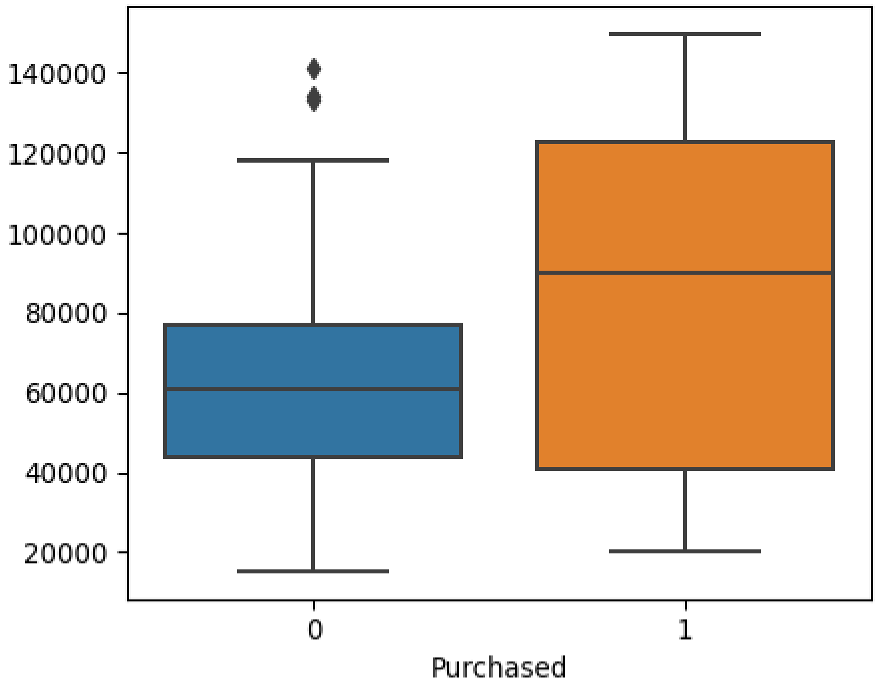
It seems that more females would purchase the insurance when compare with males.

```
sns.boxplot(x = "Purchased", y = "Age", data = insurance);
```



From the look of things, other people purchased the insurance compared with the younger people.

```
sns.boxplot(x = "Purchased", y = "EstimatedSalary", data = insurance);
```



People that earned higher salary purchased the insurance while those that earned low did not purchase the insurance. Of course, it is expected you purchase a medical insurance when you have money.

Importing machine learning models

```
from sklearn import metrics # For model evaluation
from sklearn.model_selection import train_test_split # To divide the data int
o training and test set
```

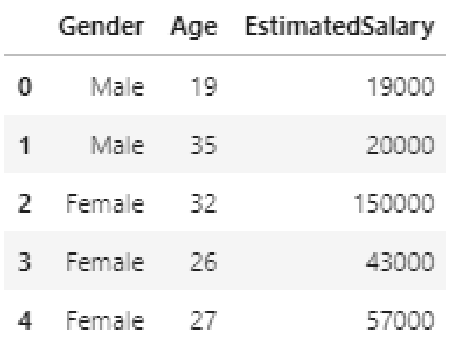
3.1.3 Data Preprocessing

Separating features and the label from the data

Now is the time to build machine learning models for the task of predicting whether the customers will buy medical insurance or not. Therefore, we shall separate the set of features (X) from the label (Y).

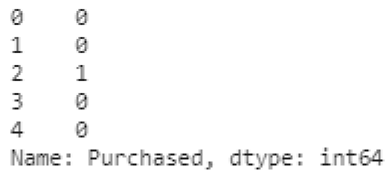
```
# split data into features and target
X = insurance.drop(["Purchased"], axis= "columns") # dropping the label varia
ble (Purchased) from the data
```

```python
y = insurance["Purchased"]

X.head()
```



	Gender	Age	EstimatedSalary
0	Male	19	19000
1	Male	35	20000
2	Female	32	150000
3	Female	26	43000
4	Female	27	57000

```python
y.head()
```

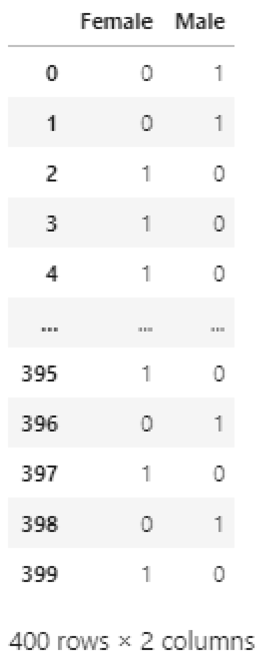


```
0    0
1    0
2    1
3    0
4    0
Name: Purchased, dtype: int64
```

One-hot encoding

As discussed in data preprocessing, we need to create a one-hot encoding for all the categorical features in the data because some algorithms cannot work with categorical data directly. They require all input variables and output variables to be numeric. In this case, we will create a one-hot encoding for the gender feature by using **pd.get_dummies()**.

```python
pd.get_dummies(insurance["Gender"])
```

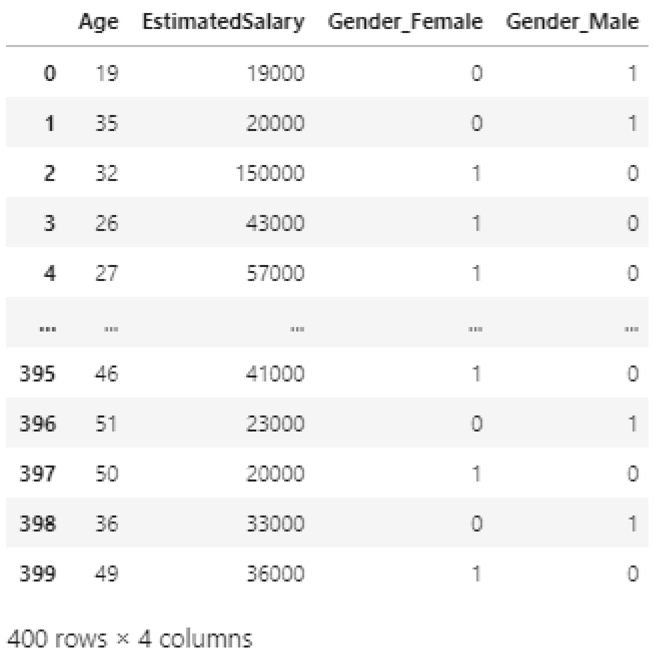


	Female	Male
0	0	1
1	0	1
2	1	0
3	1	0
4	1	0
...	...	...
395	1	0
396	0	1
397	1	0
398	0	1
399	1	0

400 rows × 2 columns

In fact, **pd.get_dummies( )** is very powerful to actually locate the categorical features and create a one-hot encoding for them. For example:

```
pd.get_dummies(X)
```

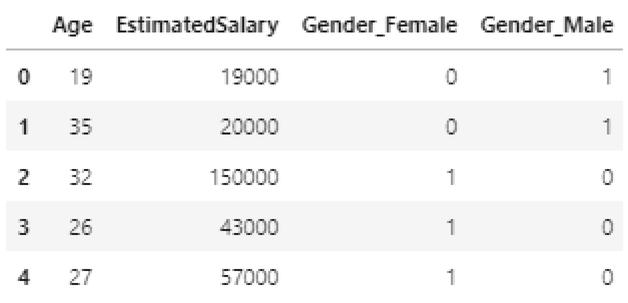


	Age	EstimatedSalary	Gender_Female	Gender_Male
0	19	19000	0	1
1	35	20000	0	1
2	32	150000	1	0
3	26	43000	1	0
4	27	57000	1	0
...	...	...	...	...
395	46	41000	1	0
396	51	23000	0	1
397	50	20000	1	0
398	36	33000	0	1
399	49	36000	1	0

400 rows × 4 columns

We now save this one-hot encoding result into X.

```
X = pd.get_dummies(X)

X.head()
```



	Age	EstimatedSalary	Gender_Female	Gender_Male
0	19	19000	0	1
1	35	20000	0	1
2	32	150000	1	0
3	26	43000	1	0
4	27	57000	1	0

Split the data into training and test set

As discussed in A, We will split our dataset (Features (X) and Label (Y)) into training and test data by using **train_test_split( )** function from the sklearn. The training set will be 80% while the test set will be 20%. The **random_state** that is set to 1234 is for all of us to have the same set of data. It can be set to any number of choices.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ra
ndom_state= 1234)
```

We now have the pair of training data **(X_train, y_train)** and test data **(X_test, y_test)**

3.2 Model training and evaluation

We will use the training data to build the model and then use test data to make prediction and evaluation respectively.

3.2.1 Logistic regression

Let's train a Logistic regression model with our training data. We need to import the Logistic regression from the sklearn model

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
```

We now create an object of class **LogisticRegression()** to train the model on

```
logisticmodel = LogisticRegression()
logisticmodel.fit(X_train, y_train)
```

LogisticRegression()

logisticmodel.fit trained the Logistic regression model. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
logisticmodel.predict(X_test)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0])
```

Let’s save the prediction result into **logistic_prediction**. This is what the model predicted for us.

```
logistic_prediction = logisticmodel.predict(X_test)
```

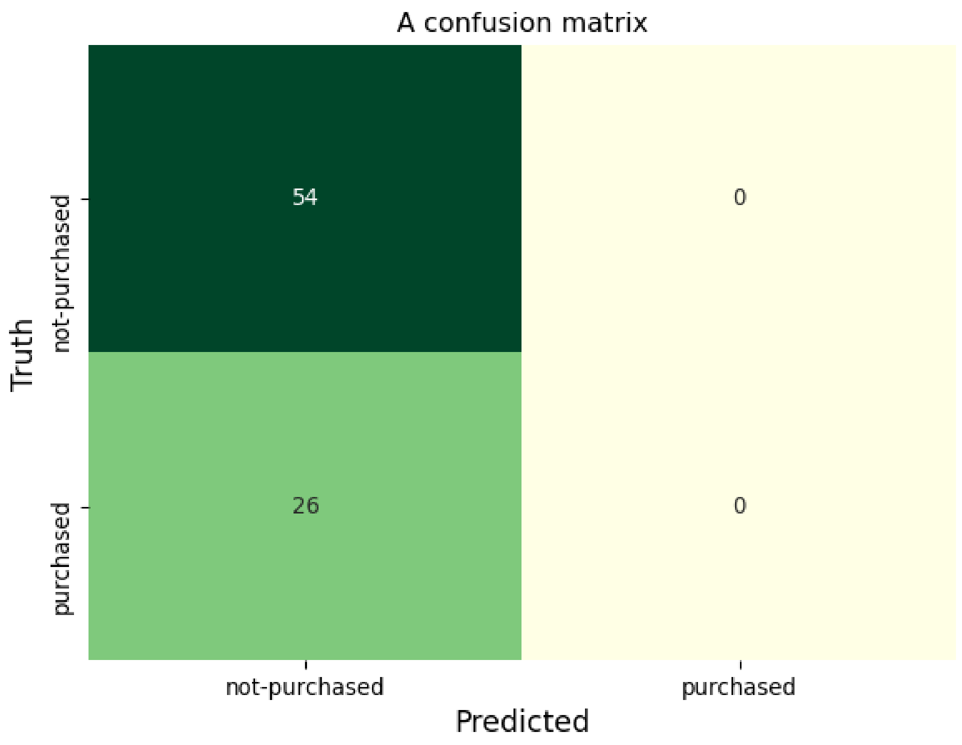
Model evaluation

Since we know the true label in the test set (i.e. **y_test**), we can compare this prediction with it, hence evaluate the logistic model. I have created a function that will help you visualize a confusion matrix for the logistic model and you can call on it henceforth to check the performance of any model.

```
def ConfusionMatrix(ytest, ypred, label = ["Negative", "Positive"]):
    "A beautiful confusion matrix function to check the model performance"
    from sklearn.metrics import confusion_matrix
    import seaborn as sns
    cm = confusion_matrix(ytest, ypred)
    plt.figure(figsize=(7, 5))
    sns.heatmap(cm, annot = True, cbar = False, fmt = 'd', cmap = 'YlGn')
    plt.xlabel('Predicted', fontsize = 13)
    plt.xticks([0.5, 1.5], label)
    plt.yticks([0.5, 1.5], label)
    plt.ylabel('Truth', fontsize = 13)
    plt.title('A confusion matrix');
```

By using the **ConfusionMatrix( )** function, we have:

```
ConfusionMatrix(y_test, logistic_prediction, label= ["not-purchased", "purcha
sed"])
```



Interpretation of the logistic regression model evaluation performance

- There are 54 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.
- There are 26 False Negative (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.

Evaluation metric

We will use some functions such accuracy and F1-score from metrics module.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, logistic_prediction)
```

```
0.675
```

The accuracy of the model is 66.25%. We cannot trust this accuracy since the data is class imbalanced. Therefore, we are going to use F1 score instead.

```
metrics.f1_score(y_test, logistic_prediction)
```

```
0.0
```

As you can see from the confusion matrix and the result of F1 score, this model is not efficient to predict whether or not a customer will buy the insurance.

Naive Bayes

Let's train a Naive Bayes classifier with our training data. We need to import the model from the sklearn model

```
from sklearn.naive_bayes import GaussianNB
naivemodel = GaussianNB()
naivemodel.fit(X_train, y_train)
```

```
GaussianNB()
```

naivemodel.fit() trained the Naive Bayes model. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
naivemodel_prediction = naivemodel.predict(X_test)
```

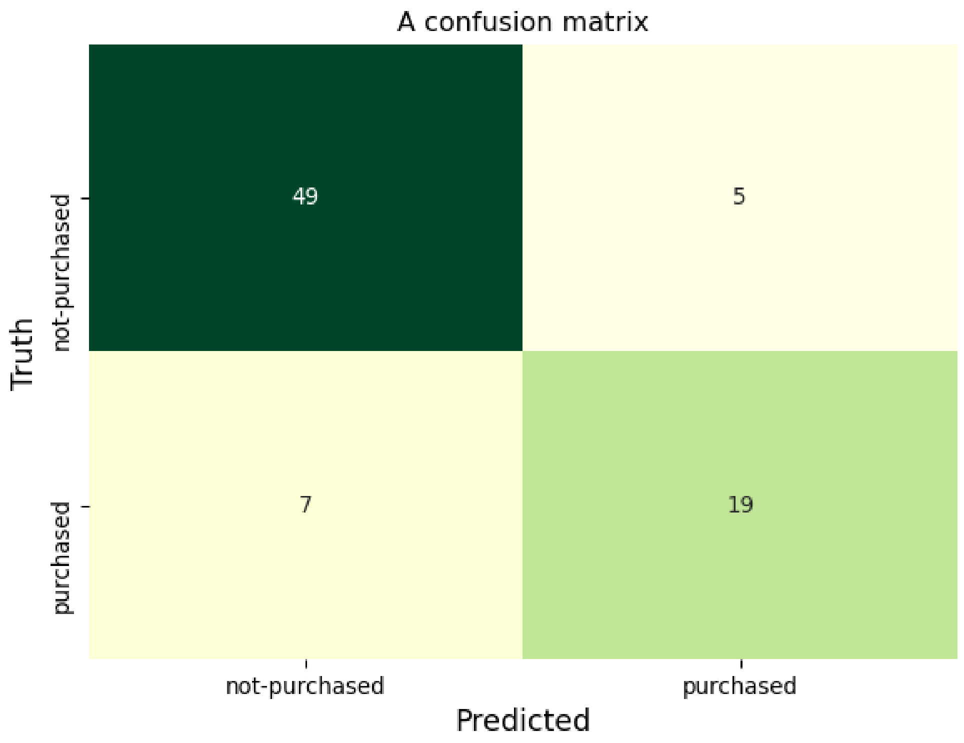
You can call one **naivemodel_prediction**to see the prediction

```
naivemodel_prediction
```

```
array([0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0])
```

By using the **ConfusionMatrix()** function, we can see how the model performed:

```
ConfusionMatrix(y_test, naivemodel_prediction, label= ["not-purchased", "purc
hased"])
```



Interpretation of the Naive model evaluation performance

- There are 49 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.
- There are 19 True Positives (TP): predicting that the customer will buy the insurance and truly the customer did buy the insurance.
- There are 7 False Negatives (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.
- There are 5 False Positives (FN): predicting that the customer will buy the insurance and the customer did not buy the insurance.

Evaluation metrics

We are going to check the **accuracy** and **F1** score of them model.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, naivemodel_prediction)
```

0.85

The accuracy of the model is 85%

We can check the F1 score by using:

```
metrics.f1_score(y_test, naivemodel_prediction)
```

0.76

The F1 score of the model is 76%

As you can see, this model seems good in predicting whether a patient will buy insurance or not.

3.2.2 Random Forest Model

Let's train a Random Forest model with our training data. We need to import the Random Forest model from the sklearn module

```
from sklearn.ensemble import RandomForestClassifier
randomforestmodel = RandomForestClassifier()
randomforestmodel.fit(X_train, y_train)
```

RandomForestClassifier()

randomforestmodel.fit() trained the Random Forest model on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
randomforestmodel_prediction = randomforestmodel.predict(X_test)
```

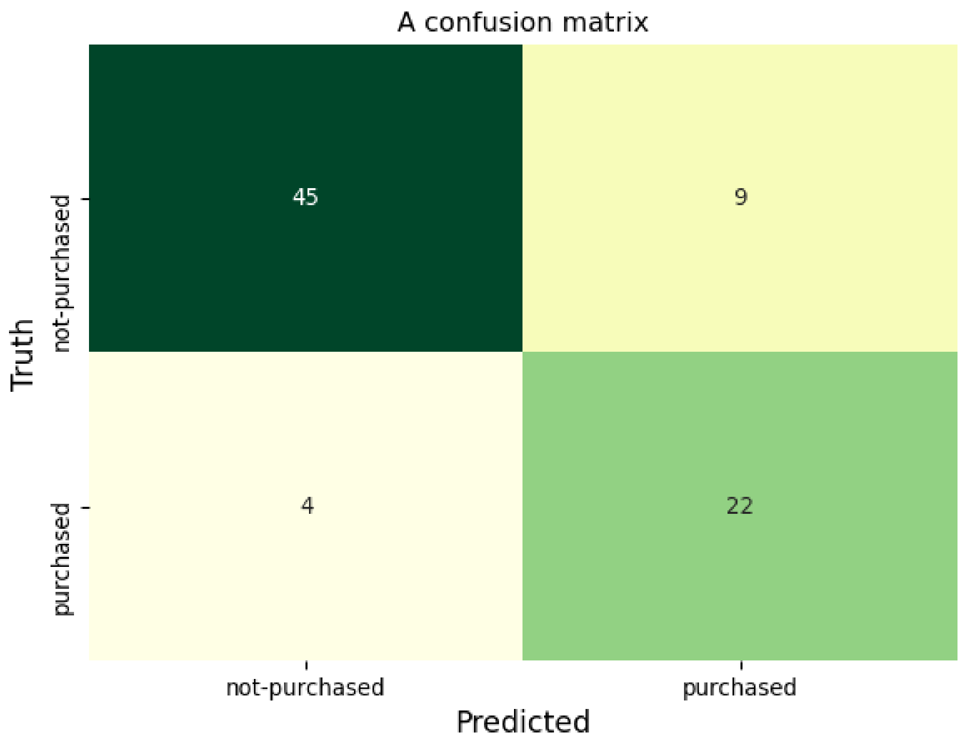
You can call one **randomforestmodel_prediction** to see the prediction

```
randomforestmodel_prediction
```

```
array([0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0])
```

By using the **ConfusionMatrix()** function, we can see how the model performed:

```
ConfusionMatrix(y_test, randomforestmodel_prediction, label= ["not-purchased"
, "purchased"])
```



Interpretation of the Random Forest model evaluation performance

- There are 45 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.
- There are 22 True Positives (TP): predicting that the customer will buy the insurance and truly the customer did buy the insurance.
- There are 4 False Negatives (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.

- There are 9 False Positives (FN): predicting that the customer will buy the insurance and the customer did not buy the insurance.

Evaluation metrics

We are going to check the **accuracy** and **F1** score of them model.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, randomforestmodel_prediction)
```

0.8375

The accuracy of the model is 83.75%

We can check the F1 score by using:

```
metrics.f1_score(y_test, randomforestmodel_prediction)
```

0.7719298245614036

The F1 score of the model is 77.19%

As you can see, this model seems good in predicting whether a patient will buy insurance or not.

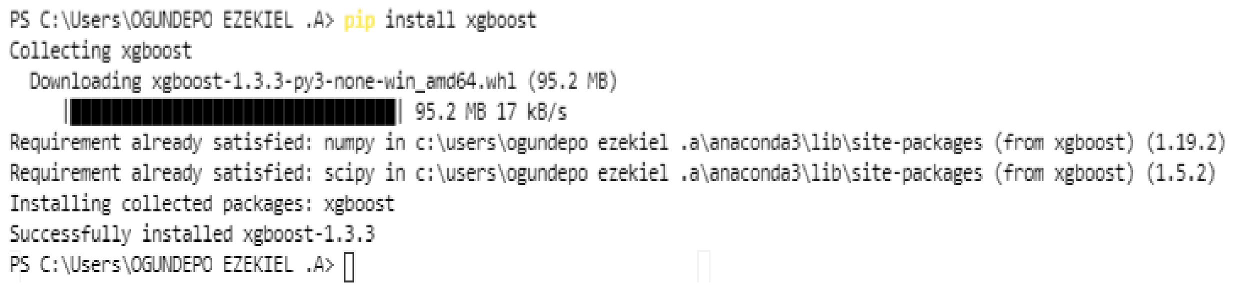
3.2.3 Extreme Gradient Boost (XGBoost) Model

Let's train an XGBoost model with our training data. We need to import the XGBoost model from the sklearn module but before we do that, we need to install the module because it is not available in the sklearn.

How to install XGBoost

Go to your termina and type **pip install xgboost**

pip install xgboost



```
PS C:\Users\OGUNDEPO EZEKIEL .A> pip install xgboost
Collecting xgboost
  Downloading xgboost-1.3.3-py3-none-win_amd64.whl (95.2 MB)
     |████████████████████████████████| 95.2 MB 17 kB/s
Requirement already satisfied: numpy in c:\users\ogundepo ezekiel .a\anaconda3\lib\site-packages (from xgboost) (1.19.2)
Requirement already satisfied: scipy in c:\users\ogundepo ezekiel .a\anaconda3\lib\site-packages (from xgboost) (1.5.2)
Installing collected packages: xgboost
Successfully installed xgboost-1.3.3
PS C:\Users\OGUNDEPO EZEKIEL .A>
```

After installation, you can now import it as follows:

```
from xgboost import XGBClassifier
xgboostmodel = XGBClassifier(use_label_encoder=False)
xgbboostmodel = xgboostmodel.fit(X_train, y_train)
```

xgboostmodel.fit() trained the XGBoost model on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
xgbboostmodel_prediction = xgboostmodel.predict(X_test)
```

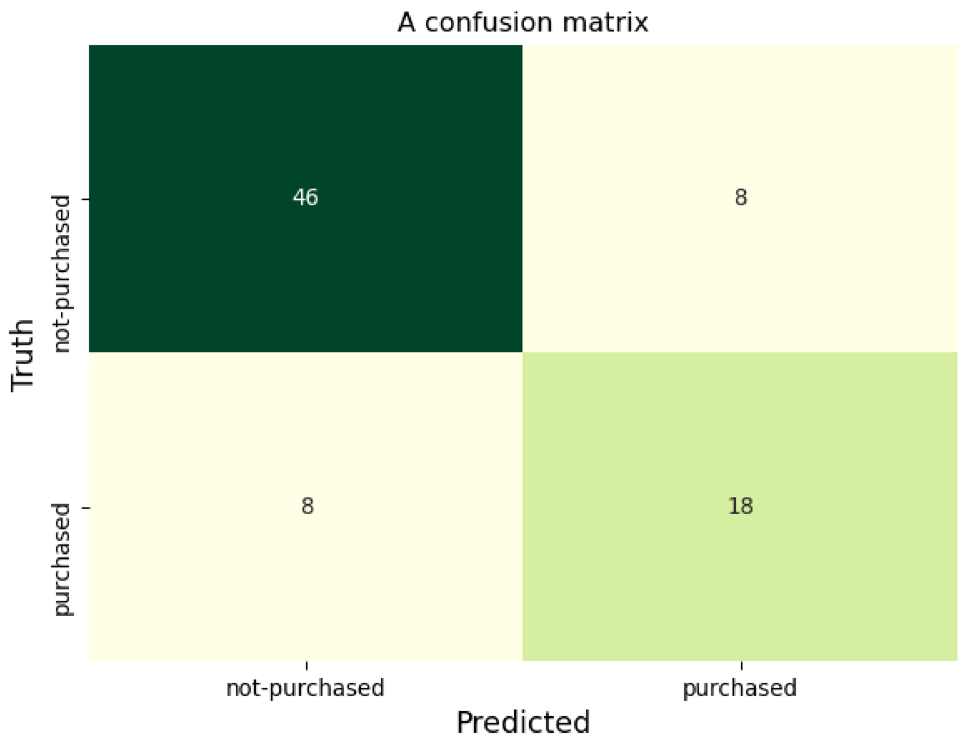
You can call on **xgbboostmodel_prediction** to see the prediction

```
xgbboostmodel_prediction
```

array([0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

By using the **ConfusionMatrix( )** function, we can see how the model performed:

```
ConfusionMatrix(y_test, xgbboostmodel_prediction, label= ["not-purchased", "p
urchased"])
```



Interpretation of the XGBoost model evaluation performance

- There are 46 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.
- There are 18 True Positives (TP): predicting that the customer will buy the insurance and truly the customer did buy the insurance.
- There are 8 False Negatives (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.
- There are 8 False Positives (FN): predicting that the customer will buy the insurance and the customer did not buy the insurance.

Evaluation metrics

We are going to check the **accuracy** and **F1** score of the model.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, xgbboostmodel_prediction)
```

0.8

The accuracy of the model is 80%

We can check the F1 score by using:

```
metrics.f1_score(y_test, xgbboostmodel_prediction)
```

0.6923076923076923

The F1 score of the model is 69.23%

As you can see, this model seems good in predicting whether a patient will buy insurance or not.

3.2.4 Support Vector Machine (SVM)

Let's train a Support Vector Machine model with our training data. We need to import the Support Vector Machine model from the sklearn module

```
from sklearn.svm import SVC
SVMmodel = SVC()
SVMmodel.fit(X_train, y_train)
```

SVC()

SVMmodel.fit() trained the Support Vector Machine on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
SVMmodel_prediction = SVMmodel.predict(X_test)
```

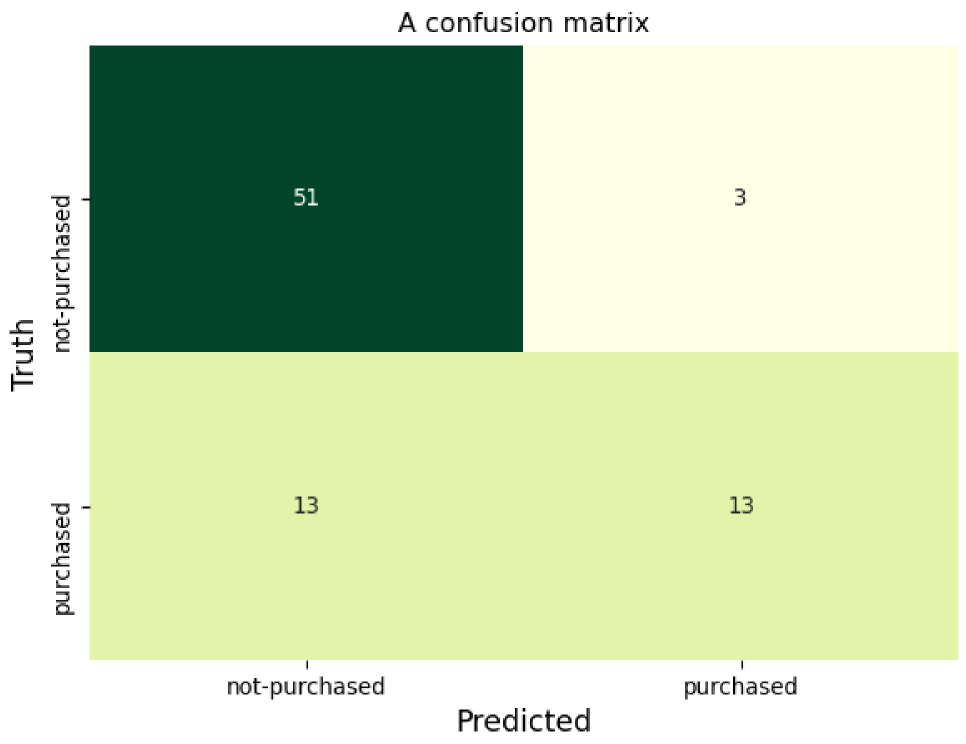
You can call on **SVMmodel_prediction** to see what has been predicted.

```
SVMmodel_prediction
```

array([0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0])

By using the **ConfusionMatrix( )** function, we can see how the model performed:

```
ConfusionMatrix(y_test, SVMmodel_prediction, label= ["not-purchased", "purcha
sed"])
```



Interpretation of the Random Forest model evaluation performance

- There are 51 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.
- There are 13 True Positives (TP): predicting that the customer will buy the insurance and truly the customer did buy the insurance.
- There are 13 False Negatives (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.
- There are 3 False Positives (FN): predicting that the customer will buy the insurance and the customer did not buy the insurance.

Evaluation metrics

We are going to check the **accuracy** and **F1** score of the model.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, SVMmodel_prediction)
```

```
0.8
```

The accuracy of the model is 80%

We can check the F1 score by using:

```
metrics.f1_score(y_test, SVMmodel_prediction)
```

```
0.6190476190476191
```

The F1 score of the model is 61.9%

As you can see, this model seems good in predicting whether a patient will buy insurance or not.

Models Summary

Model (s)	Accuracy (%)	F1-score (%)
Logistic regression	67.5	0
Naive Bayes	85	76
Random Forest	83.75	77.19
XGBoost	80	69.23
SVM	80	61.9

Having train all the five (5) models, we can see that the best model that can accurately predict whether a customer will buy the insurance or not is the Random Forest Model.

Class activity 1 (Peer to peer review activity)

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Use the following models to predict whether a customer will buy insurance or not. Your teacher has also included how to import those models for you.

- **K Nearest Neighbor**: from sklearn.neighbors import KNeighborsClassifier
- **Decision Trees Classifier**: from sklearn.tree import DecisionTreeClassifier
- **Gradient Boost Classifier**: from sklearn.ensemble import GradientBoostingClassifier

Which of the three (3) model is the best in term of the F1 score?

Class activity 2

The employee retention dataset (**HR_comma_sep.csv**) from https://www.kaggle.com/giripujar/hr-analytics can be seen in the activity directory. The dataset is from Human resources department of one big company in Somalia. The HR want to determine what is making the staff to leave the company and they have tasked you, a data scientist, to build a model to predict who is like to leave the company. The label in the dataset is **left (retention)**:

1. Do some exploratory data analysis to figure out which variables have direct and clear impact on employee retention (i.e. whether they leave the company or continue to work)
2. Plot bar charts showing impact of employee salaries on retention
3. Plot bar charts showing correlation between department and employee retention
4. Build at least three classification models for the dataset
5. Measure the accuracy of those models

6. Which model is the best among the models trained?

3.3 Regression Machine Learning Model with Kenya restaurant dataset

The objective of the regression task is to predict the amount of tip (gratuity in Kenya Shilling) given to a food server based on total_bill, gender, smoker (whether they smoke in the party or not), day(day of the week for the party), time(time of the day whether for lunch or dinner), and size(size of the party).

Label: The label for this problem is tip.

Features: There are 6 features and they include total bill, gender, smoker, day, time, and size.

We plan to use the following regression models (regressor) to predict the amount of tips that will be given during a particular party in the restaurant:

- Ordinary Least Square (OLS)
- Support Vector Machine (SVM)
- Extreme Gradient Boosting (XGBoost)
- Decision Tree
- Random Forest

3.3.1 Import Python modules

We need to import some packages that will enable us to explore the data and build machine learning models

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

```
tip = pd.read_csv("datasets/tips.csv")
tip.head(10)
```

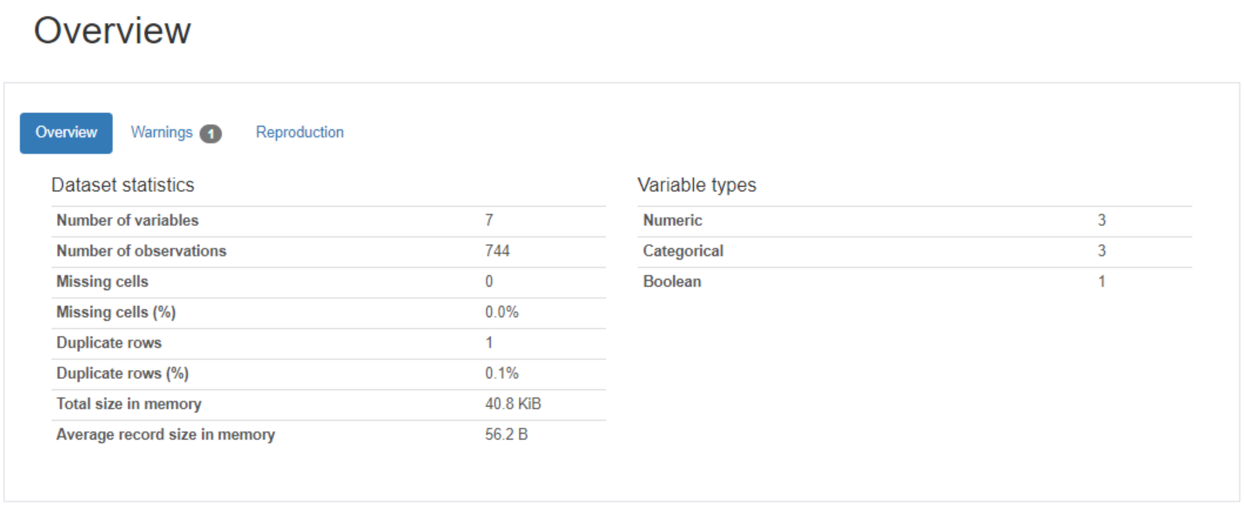
	total_bill	tip	gender	smoker	day	time	size
0	2125.50	360.79	Male	No	Thur	Lunch	1
1	2727.18	259.42	Female	No	Sun	Dinner	5
2	1066.02	274.68	Female	Yes	Thur	Dinner	4
3	3493.45	337.90	Female	No	Sun	Dinner	1
4	3470.56	567.89	Male	Yes	Sun	Lunch	6
5	2411.08	296.48	Female	Yes	Thur	Lunch	2
6	4607.43	374.96	Female	No	Thur	Dinner	4
7	1165.21	700.87	Female	No	Mon	Dinner	2
8	2895.04	347.71	Male	No	Sat	Dinner	5
9	2622.54	253.97	Male	Yes	Thur	Lunch	6

```
tip.shape
```

(744, 7)

We can use pandas_profiling to do some data exploration before training our models

```
tip.profile_report()
```



Overview

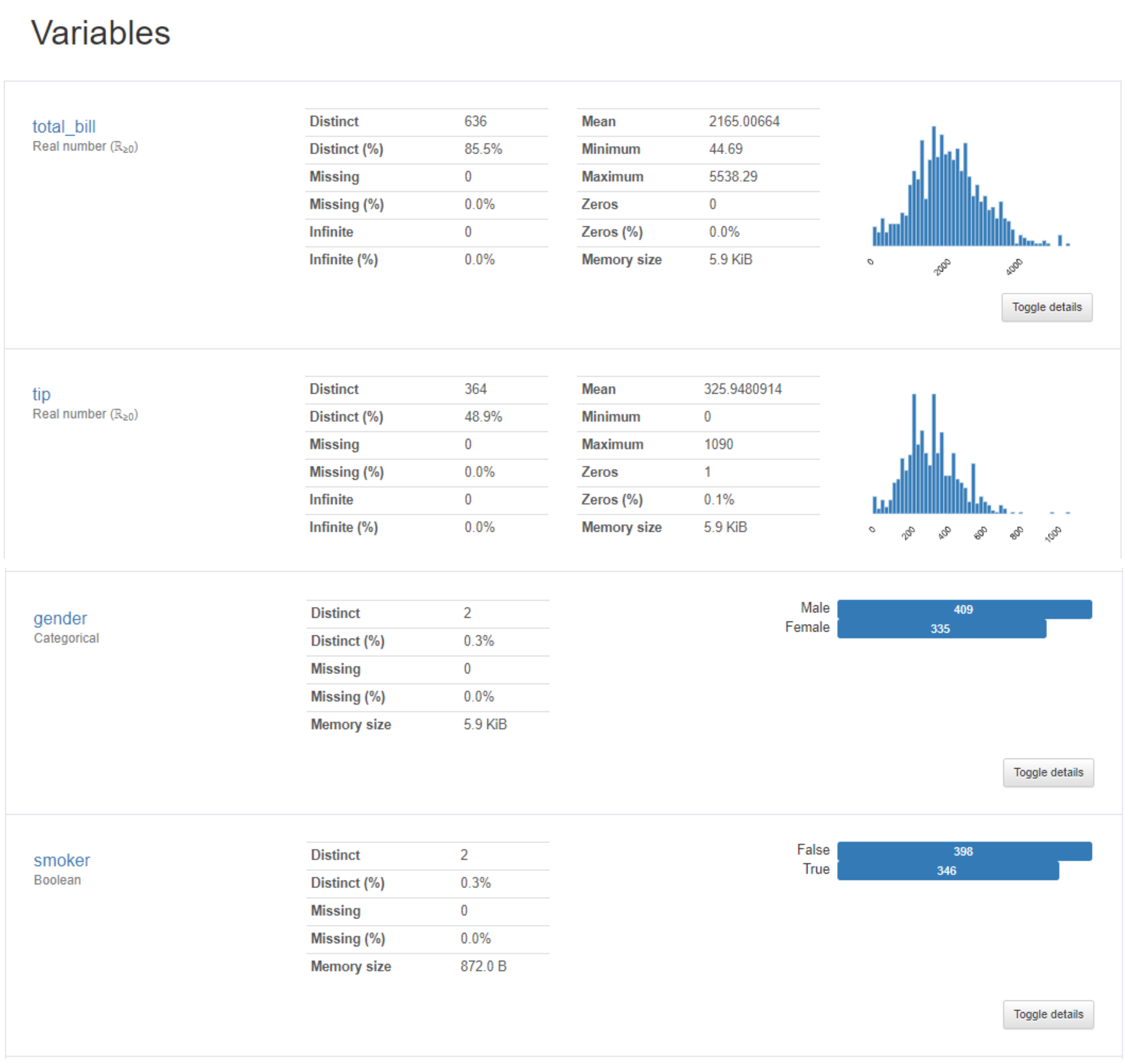
Overview | Warnings 1 | Reproduction

Dataset statistics

Number of variables	7
Number of observations	744
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	1
Duplicate rows (%)	0.1%
Total size in memory	40.8 KiB
Average record size in memory	56.2 B

Variable types

Numeric	3
Categorical	3
Boolean	1



Variables

total_bill
Real number (ℝ≥0)

Distinct	636	Mean	2165.00664
Distinct (%)	85.5%	Minimum	44.69
Missing	0	Maximum	5538.29
Missing (%)	0.0%	Zeros	0
Infinite	0	Zeros (%)	0.0%
Infinite (%)	0.0%	Memory size	5.9 KiB

tip
Real number (ℝ≥0)

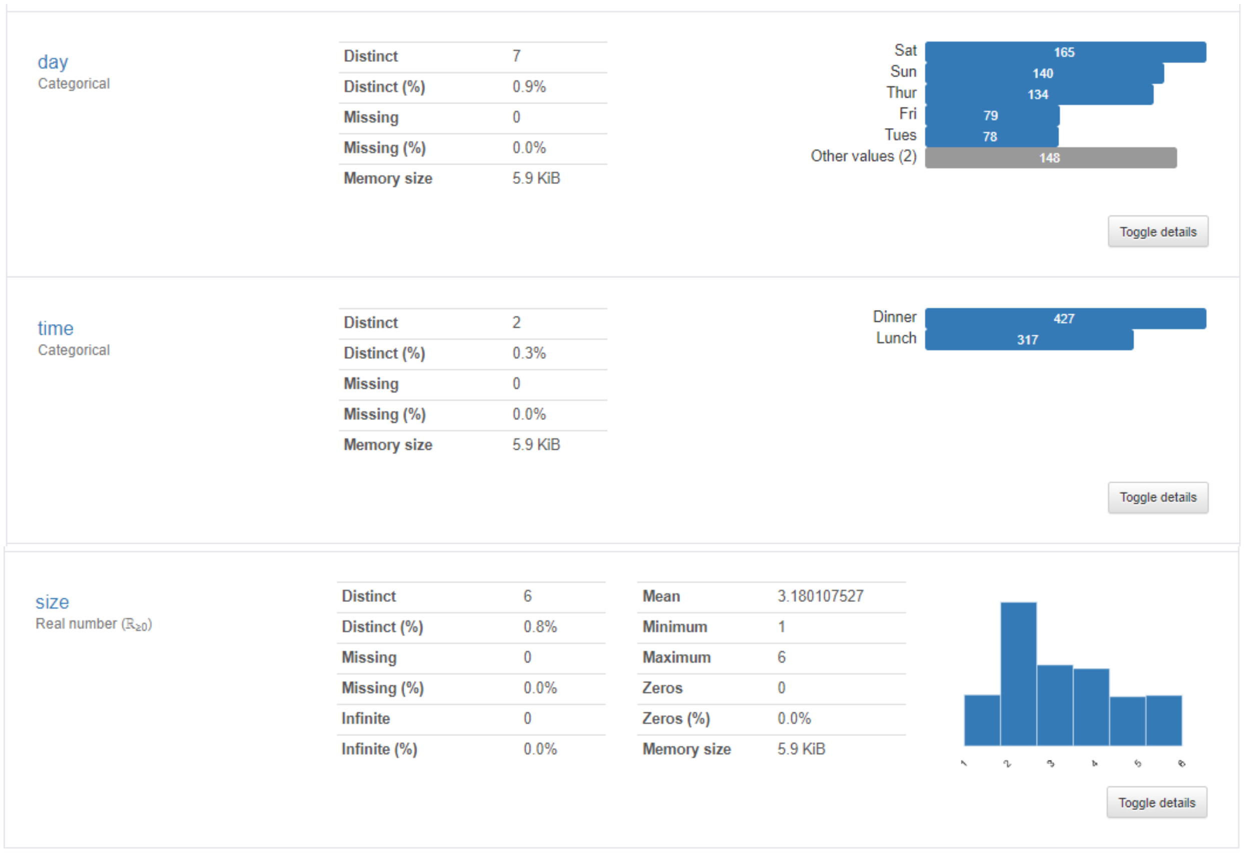
Distinct	364	Mean	325.9480914
Distinct (%)	48.9%	Minimum	0
Missing	0	Maximum	1090
Missing (%)	0.0%	Zeros	1
Infinite	0	Zeros (%)	0.1%
Infinite (%)	0.0%	Memory size	5.9 KiB

gender
Categorical

Distinct	2
Distinct (%)	0.3%
Missing	0
Missing (%)	0.0%
Memory size	5.9 KiB

smoker
Boolean

Distinct	2
Distinct (%)	0.3%
Missing	0
Missing (%)	0.0%
Memory size	872.0 B



day
Categorical

Distinct	7
Distinct (%)	0.9%
Missing	0
Missing (%)	0.0%
Memory size	5.9 KiB

Sat	165
Sun	140
Thur	134
Fri	79
Tues	78
Other values (2)	148

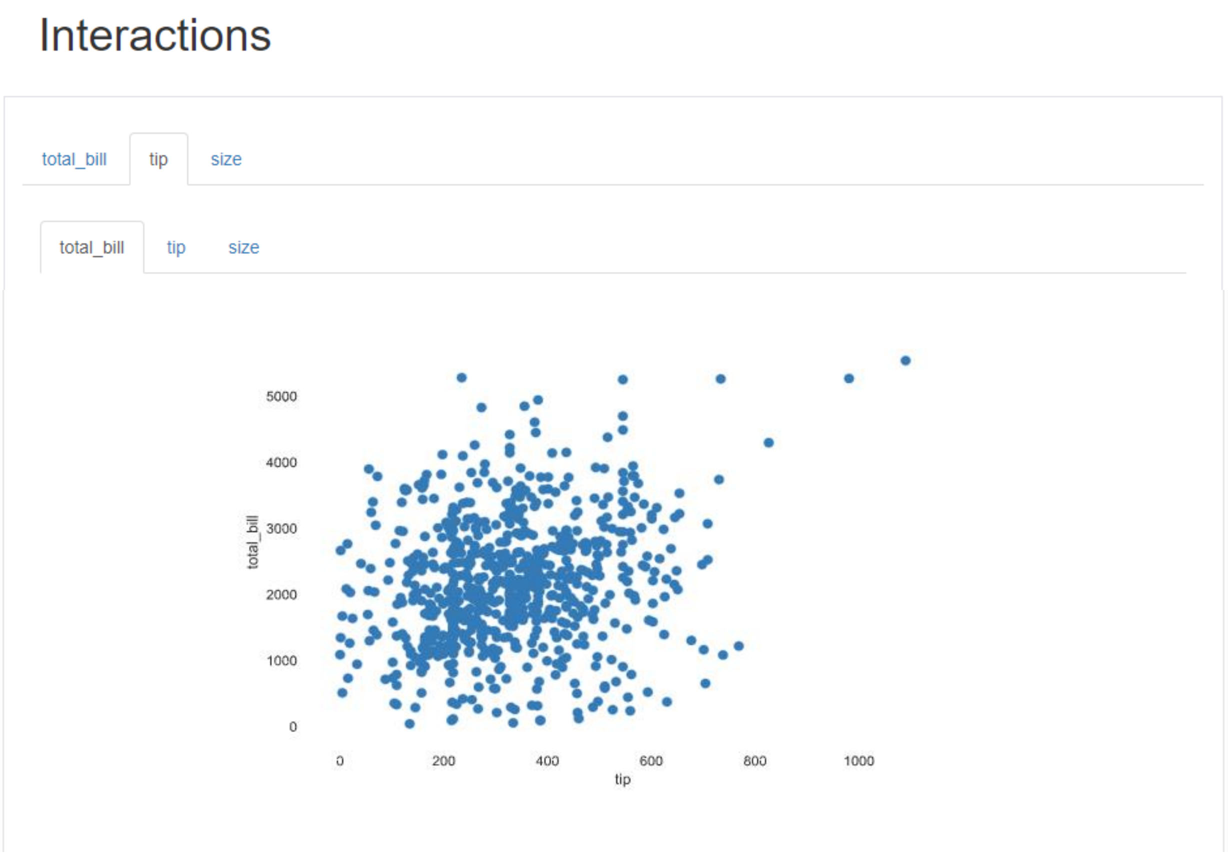
time
Categorical

Distinct	2
Distinct (%)	0.3%
Missing	0
Missing (%)	0.0%
Memory size	5.9 KiB

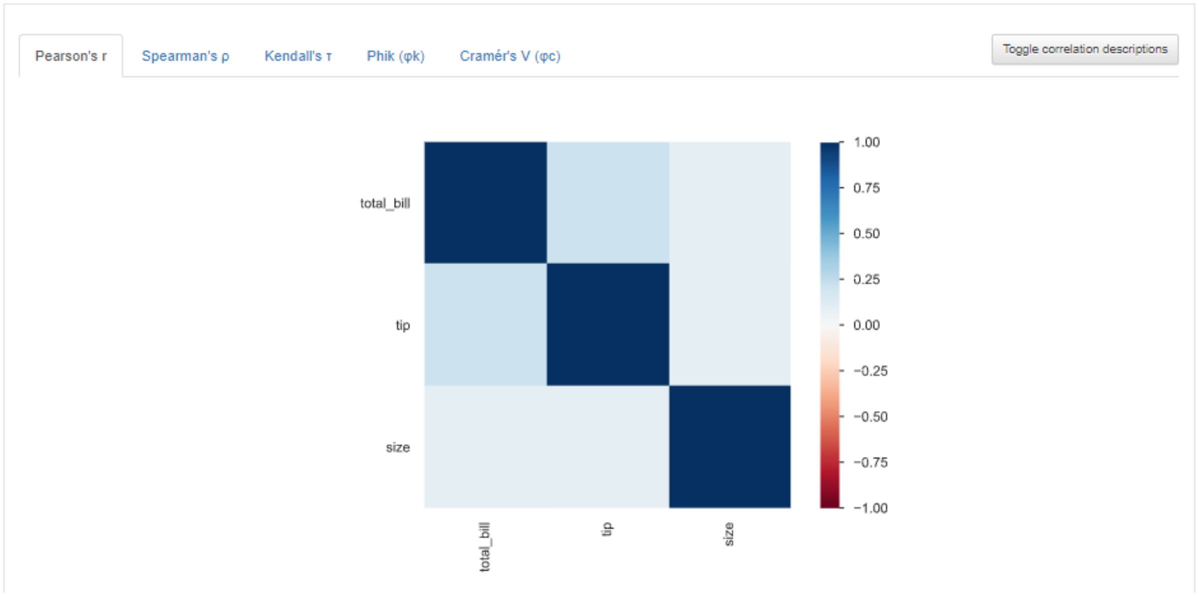
Dinner	427
Lunch	317

size
Real number ($\mathbb{R}_{\geq 0}$)

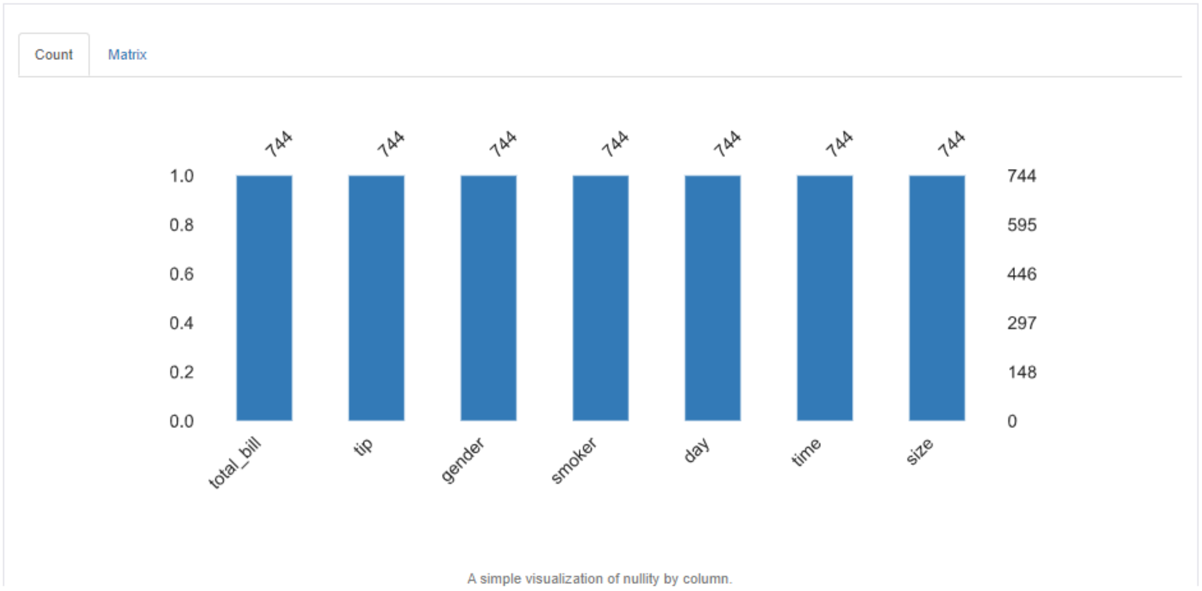
Distinct	6	Mean	3.180107527
Distinct (%)	0.8%	Minimum	1
Missing	0	Maximum	6
Missing (%)	0.0%	Zeros	0
Infinite	0	Zeros (%)	0.0%
Infinite (%)	0.0%	Memory size	5.9 KiB


Interactions
total_bill
tip
size
total_bill
tip
size
5000
4000
3000
2000
1000
0
total_bill
0
200
400
600
800
1000
tip

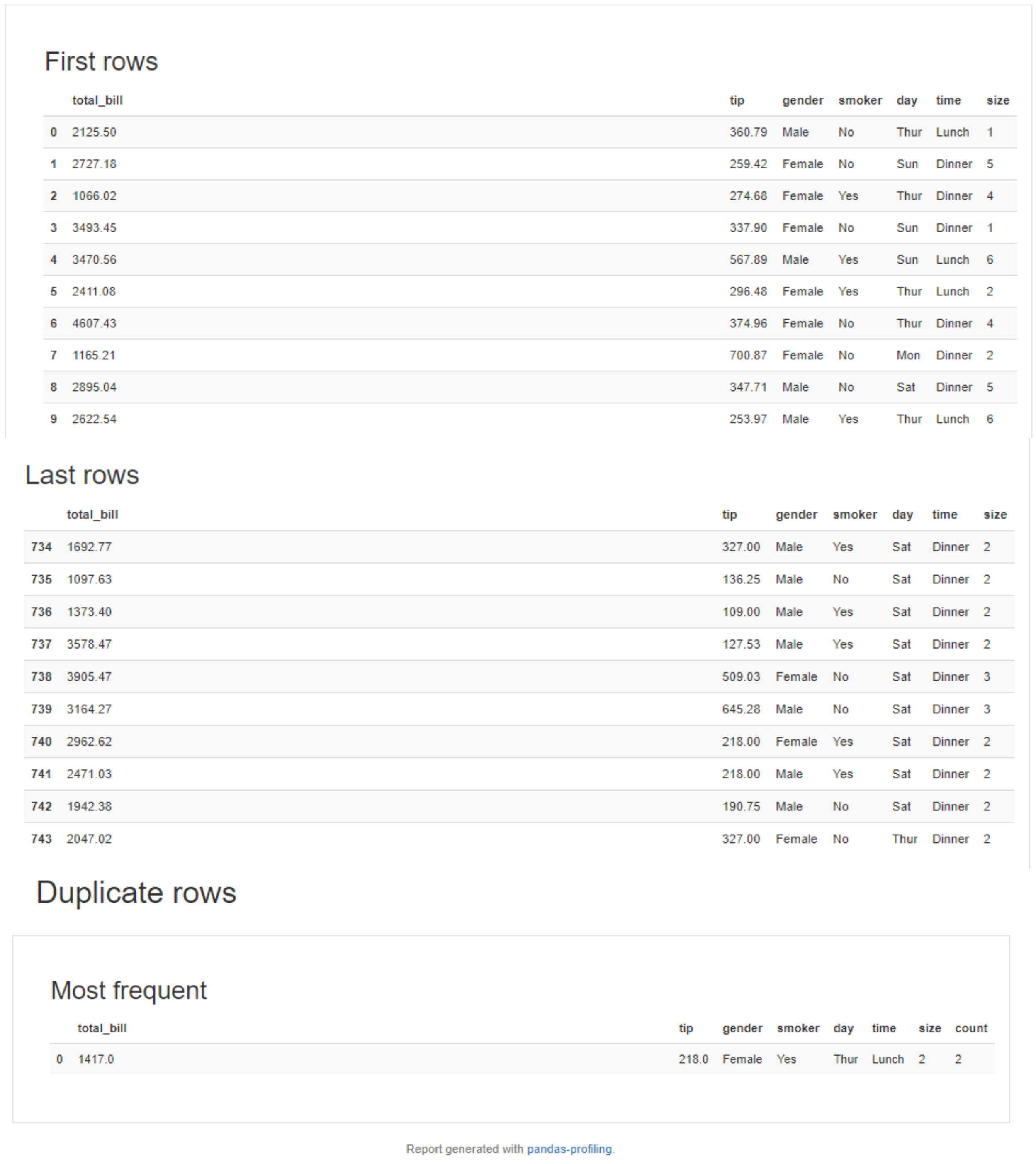
Correlations



Missing values



A simple visualization of nullity by column.



First rows

	total_bill	tip	gender	smoker	day	time	size
0	2125.50	360.79	Male	No	Thur	Lunch	1
1	2727.18	259.42	Female	No	Sun	Dinner	5
2	1066.02	274.68	Female	Yes	Thur	Dinner	4
3	3493.45	337.90	Female	No	Sun	Dinner	1
4	3470.56	567.89	Male	Yes	Sun	Lunch	6
5	2411.08	296.48	Female	Yes	Thur	Lunch	2
6	4607.43	374.96	Female	No	Thur	Dinner	4
7	1165.21	700.87	Female	No	Mon	Dinner	2
8	2895.04	347.71	Male	No	Sat	Dinner	5
9	2622.54	253.97	Male	Yes	Thur	Lunch	6

Last rows

	total_bill	tip	gender	smoker	day	time	size
734	1692.77	327.00	Male	Yes	Sat	Dinner	2
735	1097.63	136.25	Male	No	Sat	Dinner	2
736	1373.40	109.00	Male	Yes	Sat	Dinner	2
737	3578.47	127.53	Male	Yes	Sat	Dinner	2
738	3905.47	509.03	Female	No	Sat	Dinner	3
739	3164.27	645.28	Male	No	Sat	Dinner	3
740	2962.62	218.00	Female	Yes	Sat	Dinner	2
741	2471.03	218.00	Male	Yes	Sat	Dinner	2
742	1942.38	190.75	Male	No	Sat	Dinner	2
743	2047.02	327.00	Female	No	Thur	Dinner	2

Duplicate rows

Most frequent

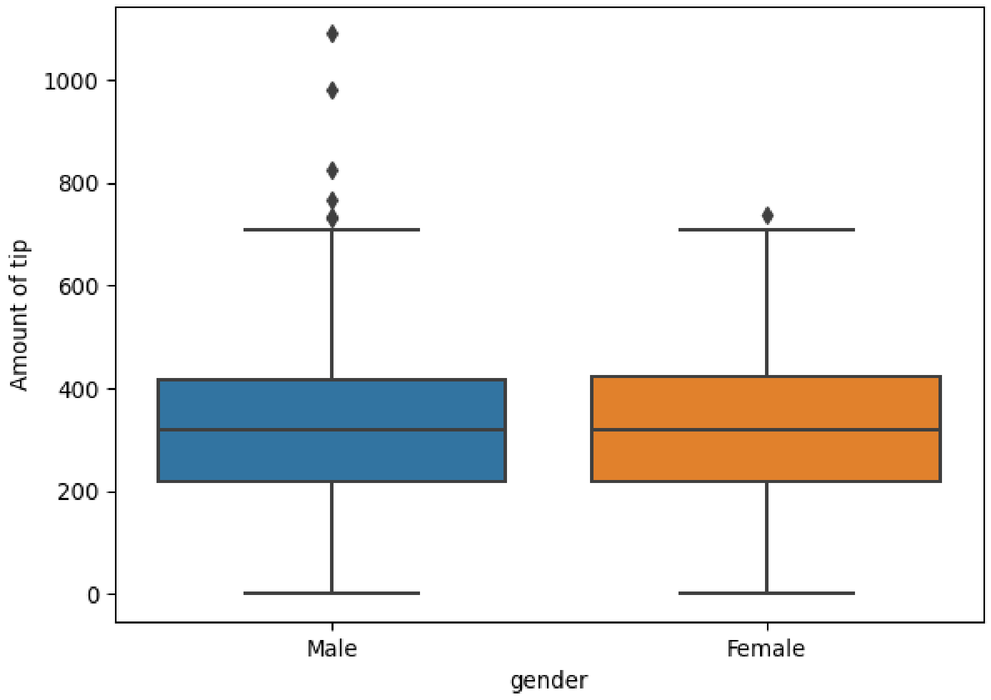
	total_bill	tip	gender	smoker	day	time	size	count
0	1417.0	218.0	Female	Yes	Thur	Lunch	2	2

Report generated with pandas-profiling.

We can also explore the relationship between the amount of tip and categorical variables

tip vs. gender

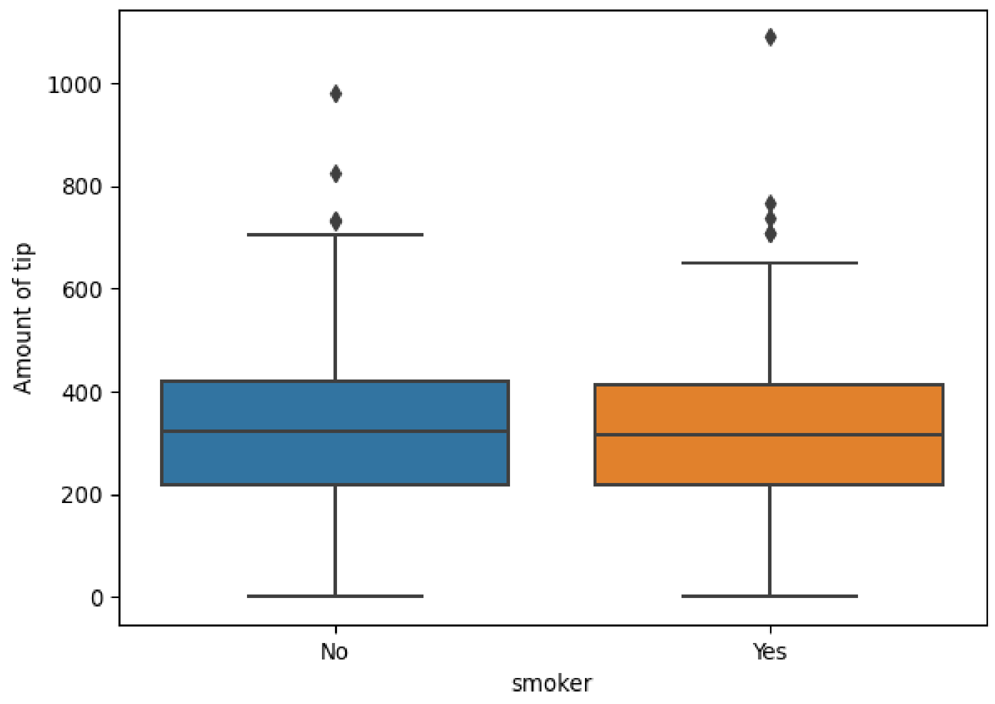
```
sns.boxplot(x = "gender", y = "tip", data = tip)
plt.ylabel("Amount of tip");
```



The amount of tips given by both gender is almost the same although there was an extreme amount of tip given by some men.

tip vs. smoker

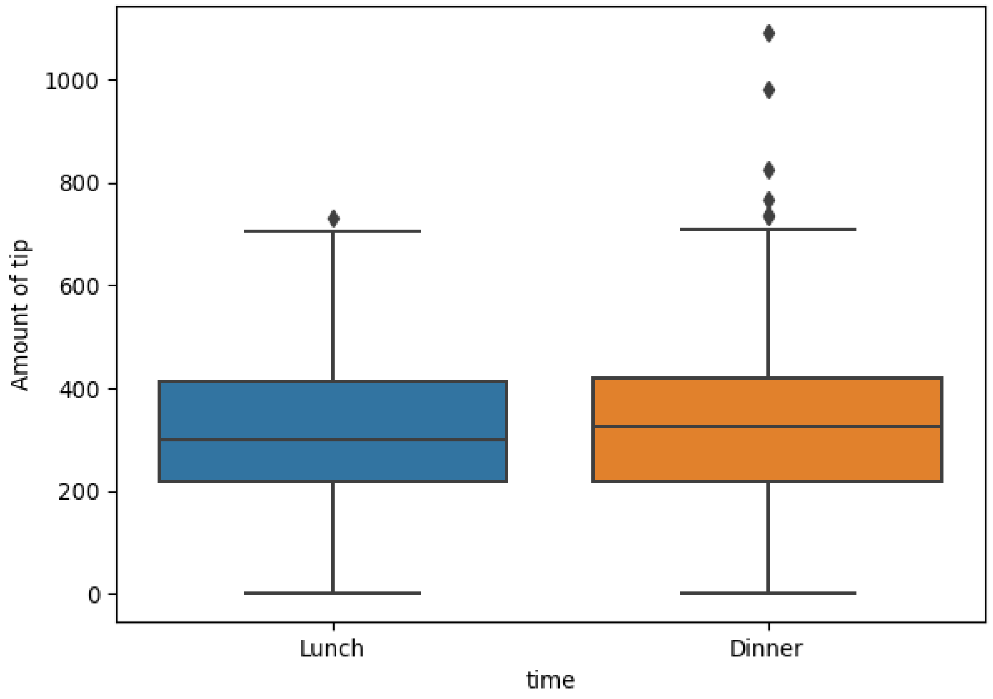
```
sns.boxplot(x = "smoker", y = "tip", data = tip)

plt.ylabel("Amount of tip");
```



Smoker and non-smoker gave almost amount of tip.

tip vs. time

```
sns.boxplot(x = "time", y = "tip", data = tip)

plt.ylabel("Amount of tip");
```



Smoker and non-smoker gave almost amount of tip.

3.3.2 Model building

After getting some insight about the data, we can now prepare the data for machine learning modelling

Importing machine learning models

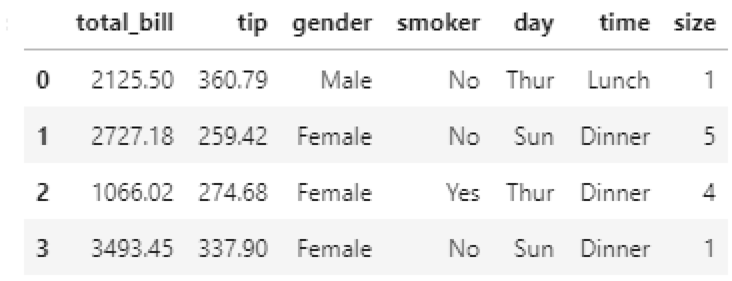
```
from sklearn import metrics # For model evaluation
from sklearn.model_selection import train_test_split # To divide the data int
o training and test set
```

Data Preprocessing

Separating features and the label from the data

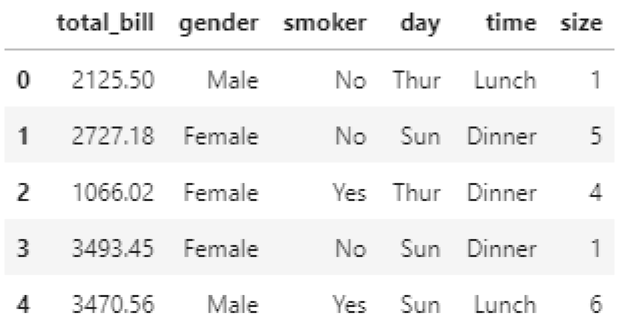
Now is the time to build machine learning models for the task of predicting the amount of tip that would be given for any party in the restaurant. Therefore, we shall separate the set of features (X) from the label (Y).

```
tip.head(4)
```



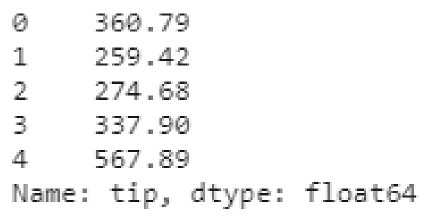
	total_bill	tip	gender	smoker	day	time	size
0	2125.50	360.79	Male	No	Thur	Lunch	1
1	2727.18	259.42	Female	No	Sun	Dinner	5
2	1066.02	274.68	Female	Yes	Thur	Dinner	4
3	3493.45	337.90	Female	No	Sun	Dinner	1

```
# split data into features and target
X = tip.drop(["tip"], axis= "columns") # droping the label variable (tip) fro
m the data
y = tip["tip"]

X.head()
```



	total_bill	gender	smoker	day	time	size
0	2125.50	Male	No	Thur	Lunch	1
1	2727.18	Female	No	Sun	Dinner	5
2	1066.02	Female	Yes	Thur	Dinner	4
3	3493.45	Female	No	Sun	Dinner	1
4	3470.56	Male	Yes	Sun	Lunch	6

```
y.head()
```



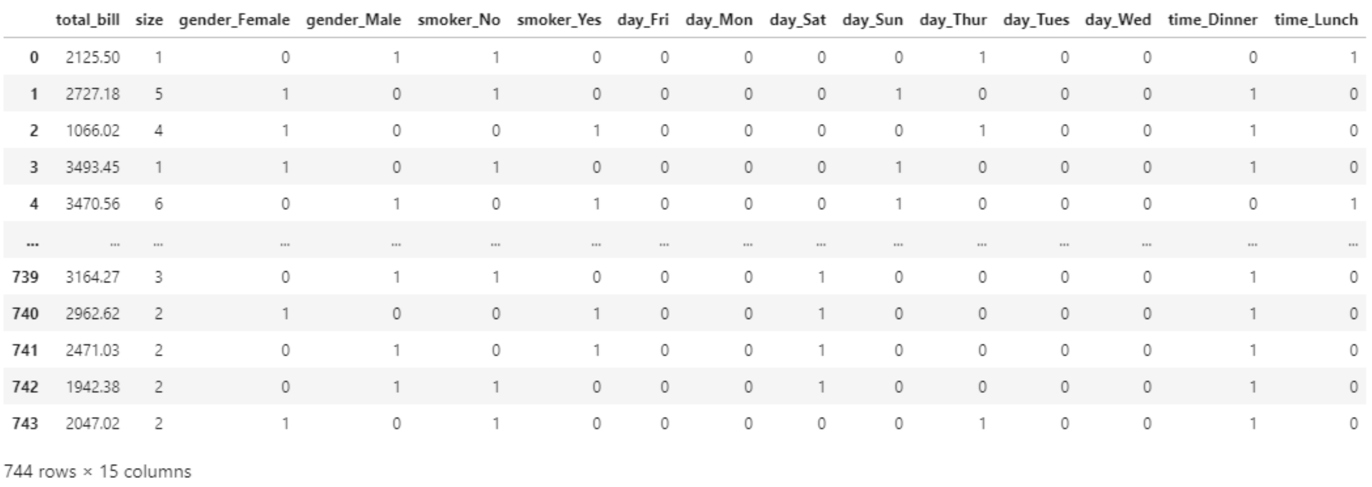
```
0    360.79
1    259.42
2    274.68
3    337.90
4    567.89
Name: tip, dtype: float64
```

Since the label is continuous, this is a regression task.

One-hot encoding

As discussed in data preprocessing,, we need to create a one-hot encoding for all the categorical features in the data because some algorithms cannot work with categorical data directly. They require all input variables and output variables to be numeric. In this case, we will create a one-hot encoding for gender, smoker, day and time by using **pd.get_dummies()**.

```
pd.get_dummies(X)
```

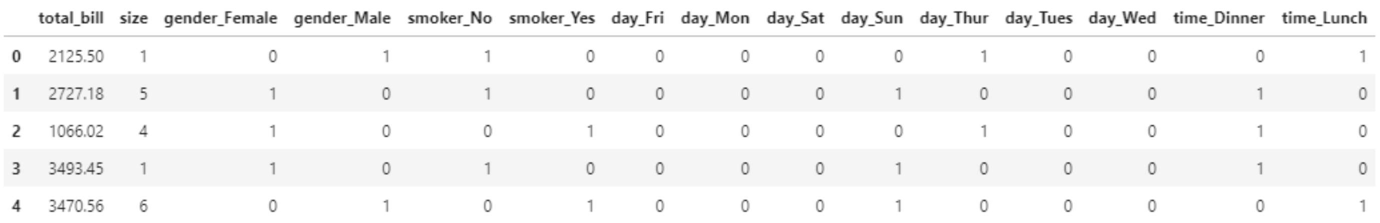


	total_bill	size	gender_Female	gender_Male	smoker_No	smoker_Yes	day_Fri	day_Mon	day_Sat	day_Sun	day_Thur	day_Tues	day_Wed	time_Dinner	time_Lunch
0	2125.50	1	0	1	1	0	0	0	0	0	1	0	0	0	1
1	2727.18	5	1	0	1	0	0	0	0	1	0	0	0	1	0
2	1066.02	4	1	0	0	1	0	0	0	0	1	0	0	1	0
3	3493.45	1	1	0	1	0	0	0	0	1	0	0	0	1	0
4	3470.56	6	0	1	0	1	0	0	0	1	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
739	3164.27	3	0	1	1	0	0	0	1	0	0	0	0	1	0
740	2962.62	2	1	0	0	1	0	0	1	0	0	0	0	1	0
741	2471.03	2	0	1	0	1	0	0	1	0	0	0	0	1	0
742	1942.38	2	0	1	1	0	0	0	1	0	0	0	0	1	0
743	2047.02	2	1	0	1	0	0	0	0	0	1	0	0	1	0

744 rows × 15 columns

We now save this result of one-hot encoding into X.

```
X = pd.get_dummies(X)

X.head()
```



	total_bill	size	gender_Female	gender_Male	smoker_No	smoker_Yes	day_Fri	day_Mon	day_Sat	day_Sun	day_Thur	day_Tues	day_Wed	time_Dinner	time_Lunch
0	2125.50	1	0	1	1	0	0	0	0	0	1	0	0	0	1
1	2727.18	5	1	0	1	0	0	0	0	1	0	0	0	1	0
2	1066.02	4	1	0	0	1	0	0	0	0	1	0	0	1	0
3	3493.45	1	1	0	1	0	0	0	0	1	0	0	0	1	0
4	3470.56	6	0	1	0	1	0	0	0	1	0	0	0	0	1

Split the data into training and test set

We will split our dataset (Features (X) and Label (Y)) into training and test data by using **train_test_split()** function from the sklearn. The training set will be 80% while the test set will be 20%. The **random_state** that is set to 1234 is for all of us to have the same set of data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ra
ndom_state= 1234)
```

We now have the pair of training data **(X_train, y_train)** and test data **(X_test, y_test)**

3.3.3 Model Training and evaluation

We will use the training data to build the model and then use test data to make prediction and evaluation respectively.

Linear Regression

Let's train a linear regression model with our training data. We need to import the Linear regression from the sklearn model

```
# Fitting Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
```

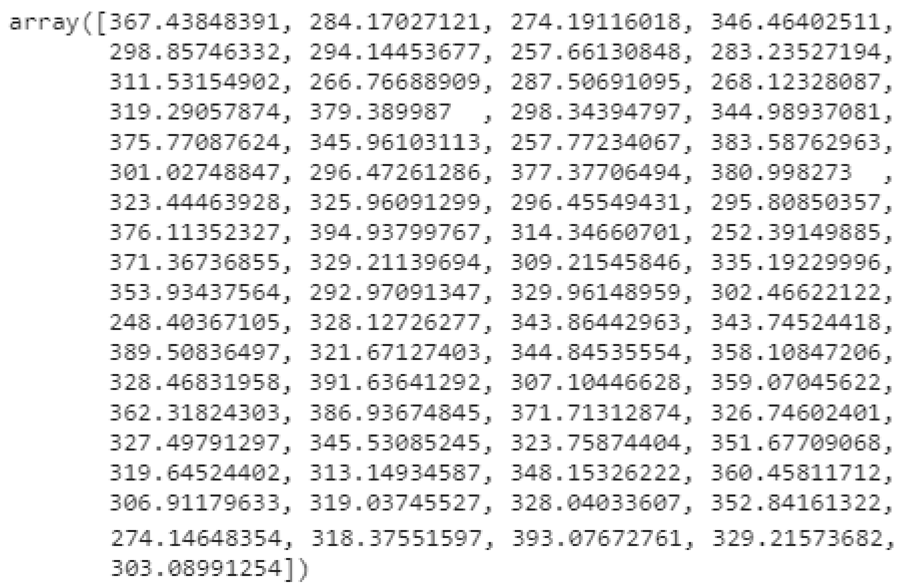
We now create an object of class **LinearRegression** to train the model on

```
linearmodel = LinearRegression()
linearmodel.fit(X_train, y_train)
```

LinearRegression()

linearmodel.fit trained the Linear regression model. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
linearmodel.predict(X_test)
```



```
array([367.43848391, 284.17027121, 274.19116018, 346.46402511,
       298.85746332, 294.14453677, 257.66130848, 283.23527194,
       311.53154902, 266.76688909, 287.50691095, 268.12328087,
       319.29057874, 379.389987  , 298.34394797, 344.98937081,
       375.77087624, 345.96103113, 257.77234067, 383.58762963,
       301.02748847, 296.47261286, 377.37706494, 380.998273  ,
       323.44463928, 325.96091299, 296.45549431, 295.80850357,
       376.11352327, 394.93799767, 314.34660701, 252.39149885,
       371.36736855, 329.21139694, 309.21545846, 335.19229996,
       353.93437564, 292.97091347, 329.96148959, 302.46622122,
       248.40367105, 328.12726277, 343.86442963, 343.74524418,
       389.50836497, 321.67127403, 344.84535554, 358.10847206,
       328.46831958, 391.63641292, 307.10446628, 359.07045622,
       362.31824303, 386.93674845, 371.71312874, 326.74602401,
       327.49791297, 345.53085245, 323.75874404, 351.67709068,
       319.64524402, 313.14934587, 348.15326222, 360.45811712,
       306.91179633, 319.03745527, 328.04033607, 352.84161322,
       274.14648354, 318.37551597, 393.07672761, 329.21573682,
       303.08991254])
```

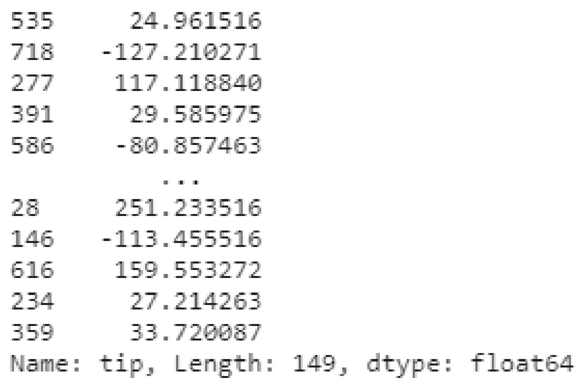
Let's save the prediction result into **linearmodel_prediction**. This is what the model predicted for us.

```
linearmodel_prediction = linearmodel.predict(X_test)
```

Model evaluation

Since the prediction is continuous, we can only measure how far the prediction is from the actual values. Let’s check the error for each prediction.

```
y_test - linearmodel_prediction
```



```
535     24.961516
718   -127.210271
277    117.118840
391     29.585975
586    -80.857463
          ...
28     251.233516
146   -113.455516
616    159.553272
234     27.214263
359     33.720087
Name: tip, Length: 149, dtype: float64
```

The positive ones show that the prediction is higher than the actual values while the negative ones are below the actual values. Let’s now measure this error by using the Root Mean Squared Error (RMSE).

```
MSE = metrics.mean_squared_error(y_test, linearmodel_prediction)

MSE
```

20201.415276948974

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)
```

142.13168287524417

Therefore, the RMSE for the linear regression is 142.1316828752442.

Random Forest Model

Let’s train a Random Forest model with our training data. We need to import the model from the sklearn module

```
from sklearn.ensemble import RandomForestRegressor
randomforestmodel = RandomForestRegressor()
randomforestmodel.fit(X_train, y_train)
```

RandomForestRegressor()

randomforestmodel.fit() trained the Random Forest model on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
randomforestmodel_prediction = randomforestmodel.predict(X_test)

MSE = metrics.mean_squared_error(y_test, randomforestmodel_prediction)

MSE
```

25061.411952729868

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)
```

158.30796553783978

Therefore, the RMSE of Random Forest is 160.3155113080993.

Extreme Gradient Boost (XGBoost) Model

Let’s train an XGBoost model with our training data. We need to import the XGBoost model from the xgboost module.

```
from xgboost import XGBRegressor
xgboostmodel = XGBRegressor(use_label_encoder=False)
xgbboostmodel = xgboostmodel.fit(X_train, y_train)
```

xgboostmodel.fit() trained the XGBoost model on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
xgbboostmodel_prediction = xgboostmodel.predict(X_test)
```

You can call on **xgbboostmodel_prediction** to see the prediction

```
MSE = metrics.mean_squared_error(y_test, xgbboostmodel_prediction)

MSE
```

29250.892630941566

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)
```

171.0289233753799

Therefore, the RMSE for the xgbboost model is 171.0289233753799

Support Vector Machine (SVM)

Let's train a Support Vector Machine model with our training data. We need to import the Support Vector Machine model from the sklearn module

```
from sklearn.svm import SVR
SVMmodel = SVR()
SVMmodel.fit(X_train, y_train)
```

SVR()

SVMmodel.fit() trained the Support Vector Machine on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
SVMmodel_prediction = SVMmodel.predict(X_test)
```

You can call on **SVMmodel_prediction** to see what has been predicted.

```
MSE = metrics.mean_squared_error(y_test, SVMmodel_prediction)

MSE
```

```
19853.340298954365
```

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)
```

```
140.90188181480886
```

Therefore, the RMSE for the Support Vector Machine is 140.90188181480886

Decision Tree

Let’s train a Decision Tree model with our training data. We need to import the Decision Tree model from the sklearn module

```
from sklearn.tree import DecisionTreeRegressor
decisiontree =  DecisionTreeRegressor()
decisiontree.fit(X_train, y_train)
```

```
DecisionTreeRegressor()
```

decisiontree.fit() trained the Decision Tree on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (**X_test**).

```
decisiontree_prediction = decisiontree.predict(X_test)
```

You can call on **decisiontree_prediction** to see what has been predicted.

```
MSE = metrics.mean_squared_error(y_test, decisiontree_prediction)

MSE
```

```
49999.11530536912
```

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)
```

223.60481950389425

Therefore, the RMSE for the Decision Tree is 223.60481950389425

Models Summary

Model (s)	**RMSE**
Linear regression	142.13
Random Forest	158.31
XGBoost	171.03
SVM	140.90
Decision Tree	223.60

Having train all the five (5) models, we can see that the best model that can accurately predict the amount of tips that would be given for a given party in the restaurant is the model with the lowest RMSE and that is Support Vector Machine.

Class activity 3 (Peer to peer review activity)

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Use the following models to predict the amount of tips that would be given for a given party in the restaurant. Your teacher has also included how to import those models for you.

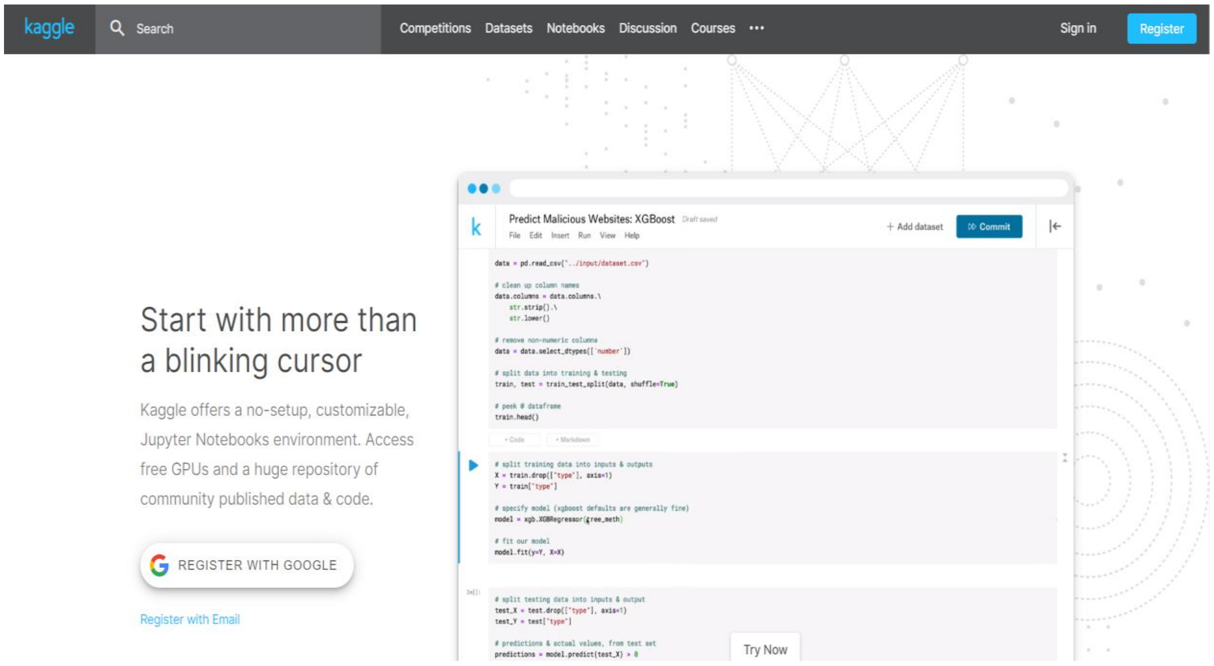
- **K Nearest Neighbor**: from sklearn.neighbors import KNeighborsRegressor
- **Ridge Regression**: from sklearn.linear_model import Ridge
- **Gradient Boost Classifier**: from sklearn.ensemble import GradientBoostingRegressor

Which of the three (3) model is the best in term of RMSE?

3.4 Machine Learning Competition Platform

3.4.1 Kaggle: Your Machine Learning and Data Science Community

Kaggle is the world's largest data science community with powerful tools and resources to help you achieve your data science goals. You can access Kaggle via www.kaggle.com.



Start with more than a blinking cursor

Kaggle offers a no-setup, customizable, Jupyter Notebooks environment. Access free GPUs and a huge repository of community published data & code.

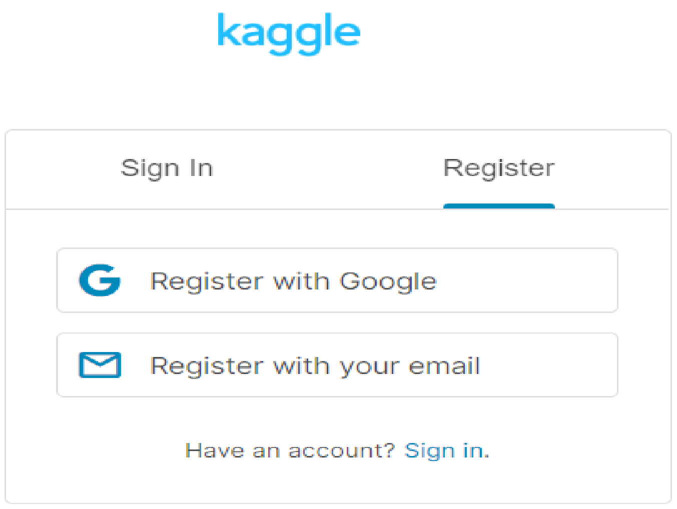
```
data = pd.read_csv("../input/dataset.csv")

# clean up column names
data.columns = data.columns.\
    str.strip().\
    str.lower()

# remove non-numeric columns
data = data.select_dtypes(['number'])

# split data into training & testing
train, test = train_test_split(data, shuffle=True)

# peek @ dataframe
train.head()
```

```
# split training data into inputs & outputs
X = train.drop(["type"], axis=1)
Y = train["type"]

# specify model (xgboost defaults are generally fine)
model = xgb.XGBRegressor(tree_meth)

# fit our model
model.fit(y=Y, X=X)
```

```
In[]:
# split testing data into inputs & output
test_X = test.drop(["type"], axis=1)
test_Y = test["type"]

# predictions & actual values, from test set
predictions = model.predict(test_X) > 0
```

Kaggle enables data scientists and other developers to engage in running machine learning contests, write and share code, and to host datasets. The types of data science problems posted on Kaggle can be anything from attempting to predict cancer occurrence by examining patient records to analyzing sentiment to evoke by movie reviews and how this affects audience reaction.

Kaggle Registration

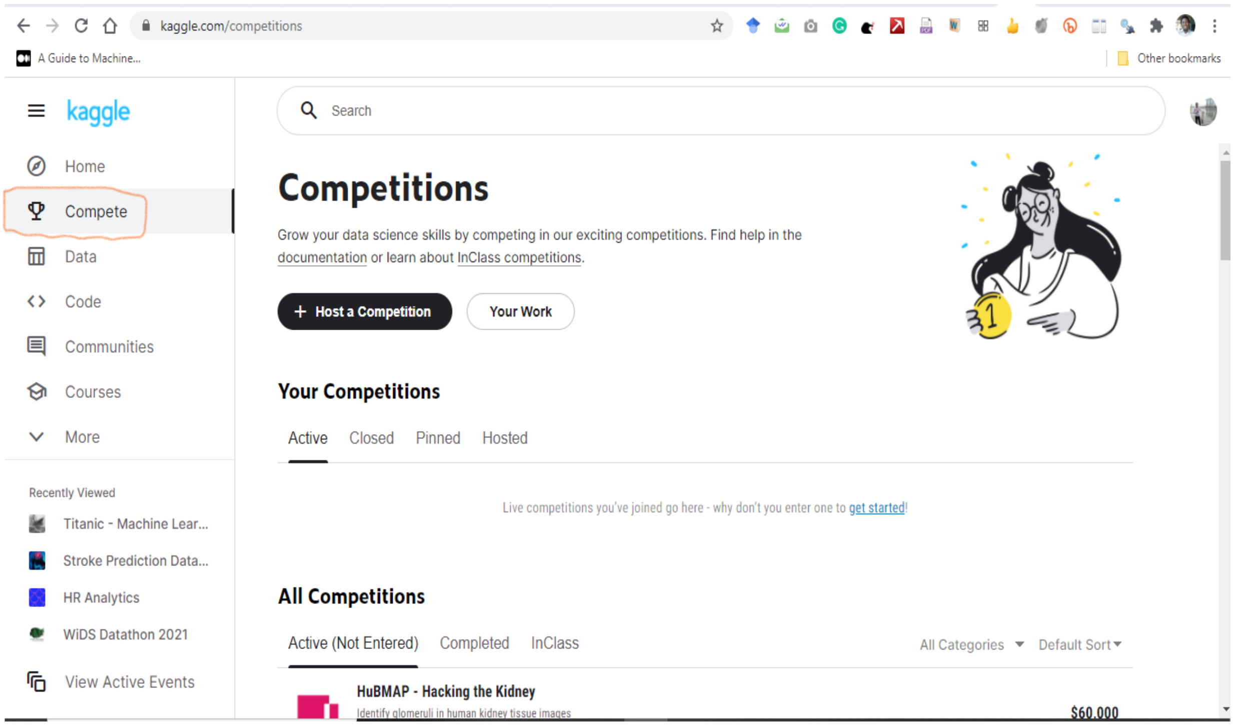

kaggle

Sign In
Register

Register with Google
Register with your email

Have an account? Sign in.

After a successful registration, you can now compete on different competitions on Kaggle.


kaggle.com/competitions

Competitions

Grow your data science skills by competing in our exciting competitions. Find help in the documentation or learn about InClass competitions.

Your Competitions

Live competitions you've joined go here - why don't you enter one to get started!

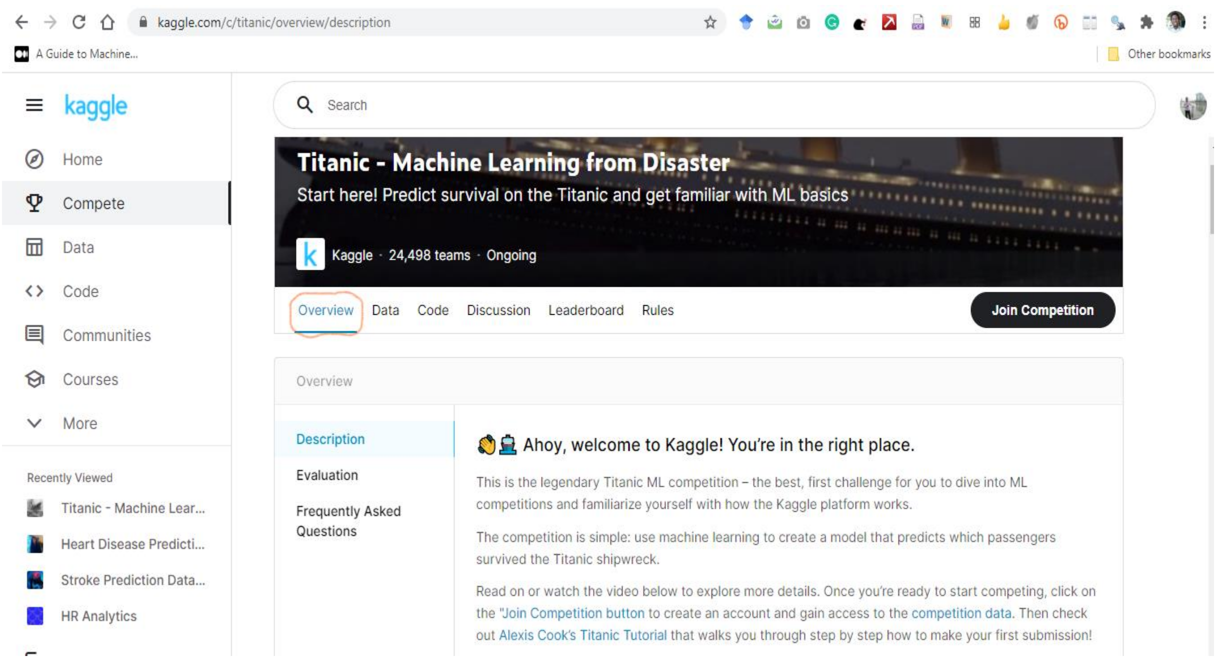
All Competitions

HuBMAP - Hacking the Kidney
Identify glomeruli in human kidney tissue images
$60,000

3.4.2 Titanic Competition

Overview tab

This contains a general introduction to the competition and in some cases, a case study to the problem you are trying to solve is described



Titanic - Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

Kaggle · 24,498 teams · Ongoing

Ahoy, welcome to Kaggle! You're in the right place.

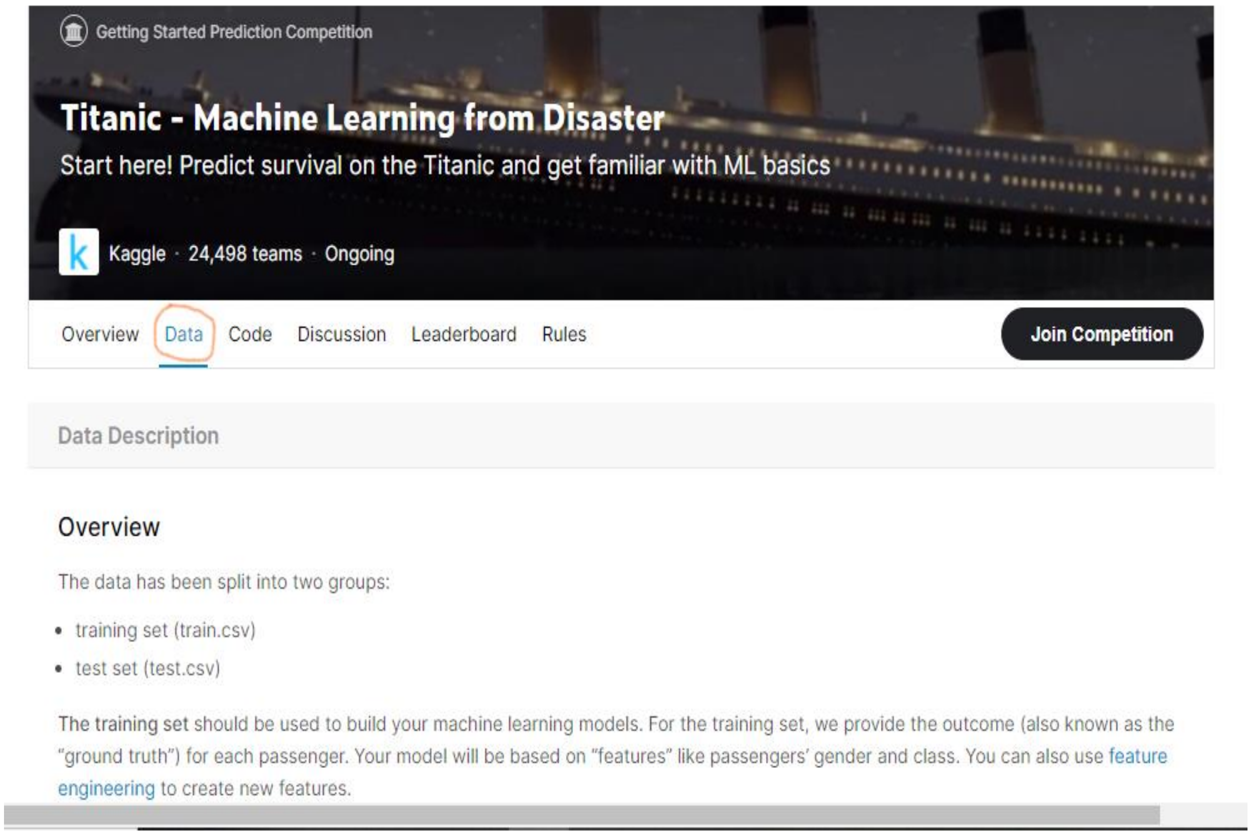
This is the legendary Titanic ML competition – the best, first challenge for you to dive into ML competitions and familiarize yourself with how the Kaggle platform works.

The competition is simple: use machine learning to create a model that predicts which passengers survived the Titanic shipwreck.

Read on or watch the video below to explore more details. Once you're ready to start competing, click on the "Join Competition button to create an account and gain access to the competition data. Then check out Alexis Cook's Titanic Tutorial that walks you through step by step how to make your first submission!

Data

It contains information about the dataset and a detailed explanation of each column in the dataset


Getting Started Prediction Competition
Kaggle · 24,498 teams · Ongoing
Overview
Data
Code
Discussion
Leaderboard
Rules
Join Competition

Titanic - Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

Data Description

Overview

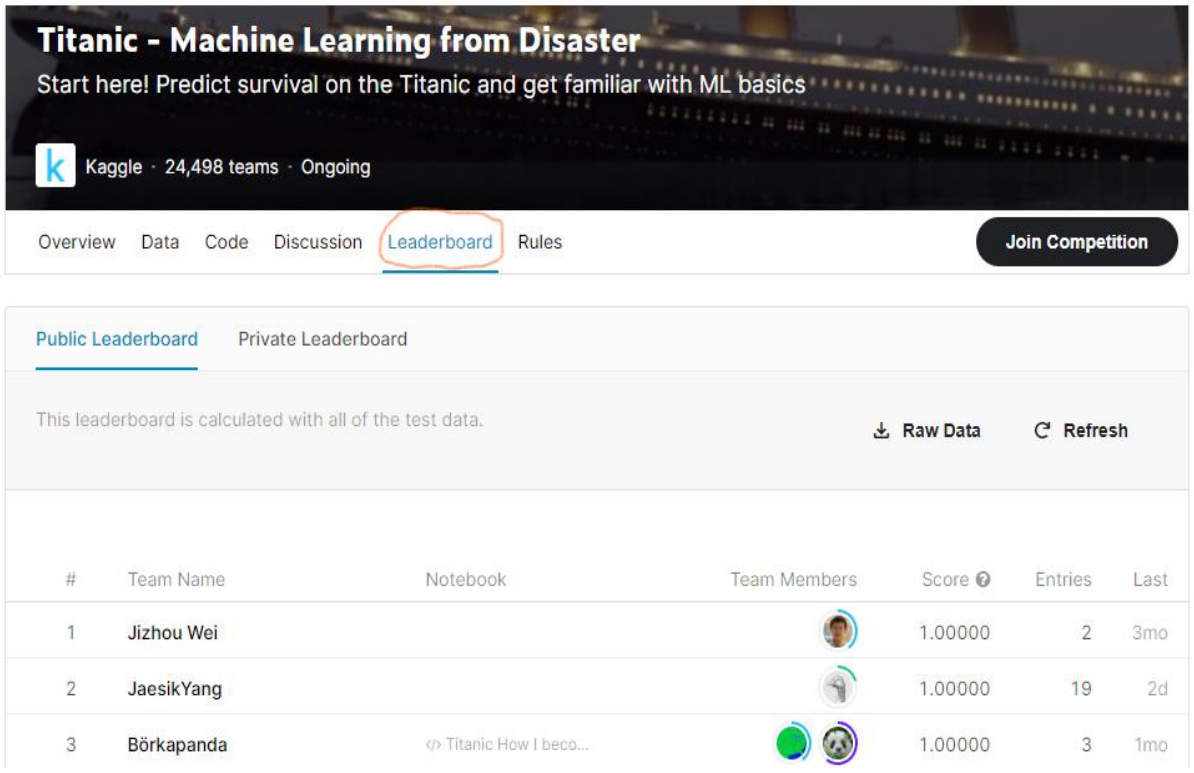
The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)

The training set should be used to build your machine learning models. For the training set, we provide the outcome (also known as the "ground truth") for each passenger. Your model will be based on "features" like passengers' gender and class. You can also use feature engineering to create new features.

Leaderboard

This display the position of each participant relative to each other based on what they submitted

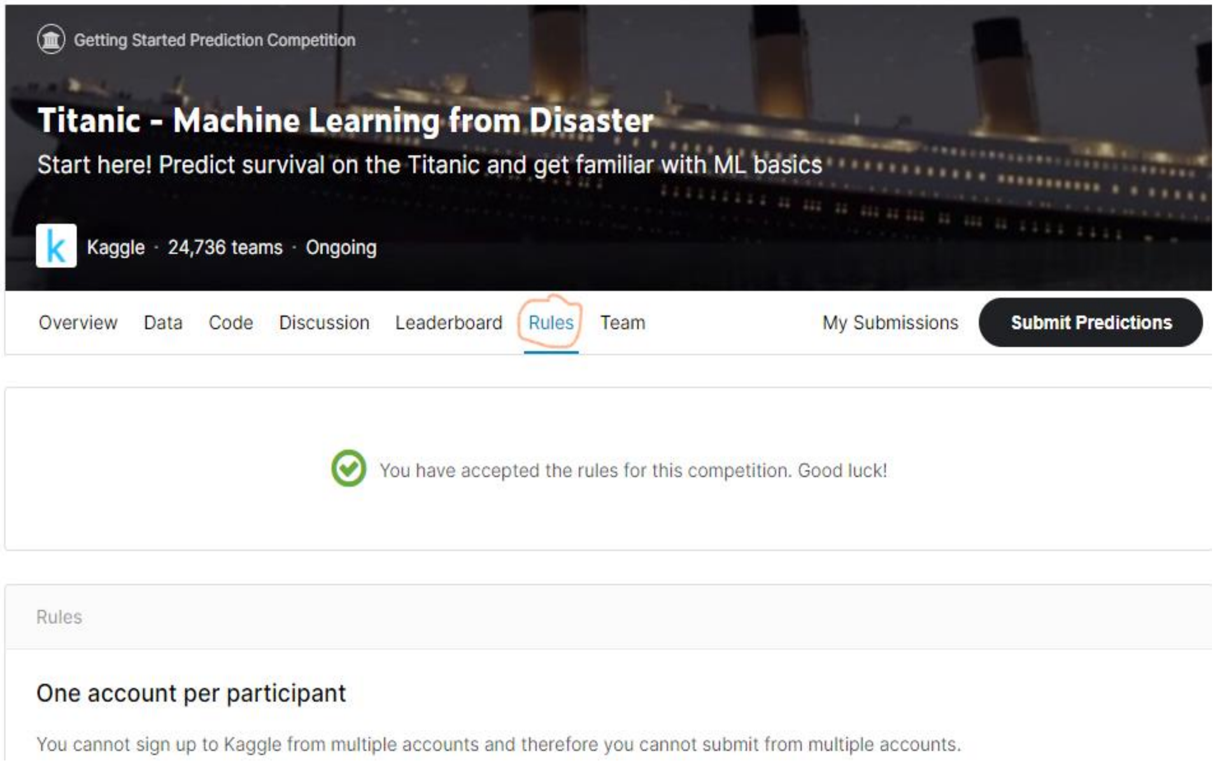

Titanic - Machine Learning from Disaster
Start here! Predict survival on the Titanic and get familiar with ML basics
Kaggle · 24,498 teams · Ongoing
Overview
Data
Code
Discussion
Leaderboard
Rules
Join Competition
Public Leaderboard
Private Leaderboard
Raw Data
Refresh

This leaderboard is calculated with all of the test data.

#	Team Name	Notebook	Team Members	Score	Entries	Last
1	Jizhou Wei			1.00000	2	3mo
2	JaesikYang			1.00000	19	2d
3	Börkapanda	</> Titanic How I beco...		1.00000	3	1mo

Rules

This tab explains the rules of the competition. Always remember to read this section


Getting Started Prediction Competition
Titanic - Machine Learning from Disaster
Start here! Predict survival on the Titanic and get familiar with ML basics
Kaggle · 24,736 teams · Ongoing
Overview
Data
Code
Discussion
Leaderboard
Rules
Team
My Submissions
Submit Predictions
Rules

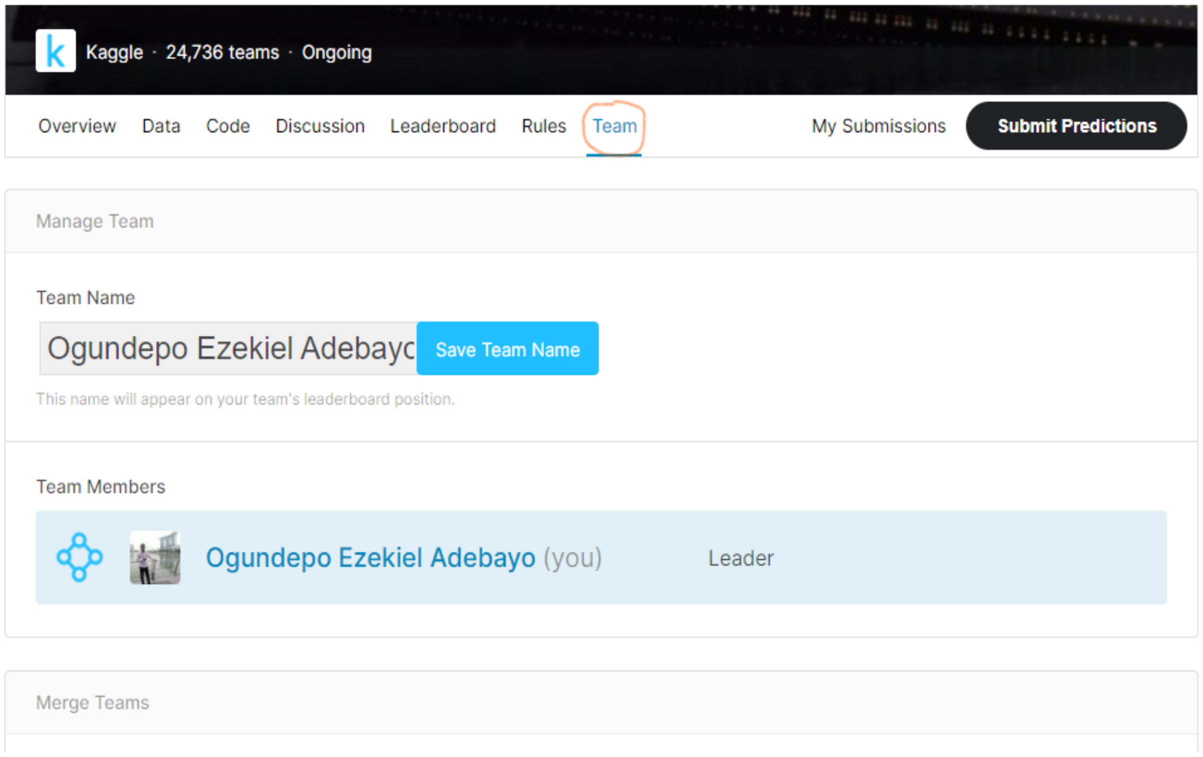
You have accepted the rules for this competition. Good luck!

One account per participant

You cannot sign up to Kaggle from multiple accounts and therefore you cannot submit from multiple accounts.

Team

Some competitions allow for team submission. This means that you can form a team to work with in that competition



Manage Team

Team Name

Ogundepo Ezekiel Adebayo

This name will appear on your team's leaderboard position.

Team Members

Ogundepo Ezekiel Adebayo (you) Leader

3.4.3 Other Machine Learning Competition Platforms

Other machine learning competitions are also available and that include:

Zindi

Zindi Africa connects organisations with thriving African data science community to solve the world's most pressing challenges using machine learning and AI. Visit https://zindi.africa/competitions for competition on Zindi.

DRIVEN DATA

DrivenData brings cutting-edge practices in data science and crowdsourcing to some of the world's biggest social challenges and the organizations taking them on. They host online challenges, usually lasting 2-3 months, where a global community of data scientists competes to come up with the best statistical model for difficult predictive problems that make a difference. Visit https://www.drivendata.org/competitions for competition on DRIVENDATA.

Class activity 4

You will compete with others on the Titanic challenge via https://www.kaggle.com/c/titanic.

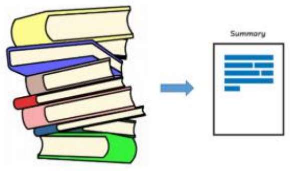
- Read about the competition instructions and download the data
- Build a predictive model using the train data and predict those that are likely to survive or not using the test data.
- Submit your prediction and note your accuracy on Kaggle

Additional resources

For more resources in this section please consider the following:

- https://datatofish.com/multiple-linear-regression-python/
- https://www.w3schools.com/python/python_ml_getting_started.asp
- https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/

Summary of Study Unit 3



In this study unit, you have learnt that:

1. Scikit-learn module can be used to train both regression and classification models
2. Before training a model, you to carry out some data processing such as separating features from the label, create one-hot encoding for categorical features, and split the data to training and test set

3. A training data will contain at most 80% of the original data while the test set will take the remainder
4. You will train the models with your training data and evaluate the performance of those models with the test set.
5. You can access the performance of your trained models with different evaluation metrics
6. You can compete on different machine learning competitions on Kaggle or Zindi