## Download Data from Kaggle

```
!pip install -q kaggle && mkdir ~/.kaggle && echo '{"username":"YOUR_KAGGLE_USERNAME","key":"YOUR_KAGGLE_KEY"}' > ~/.kaggle/kaggle.json && chmod 600 ~/.kaggle/kaggle.json && kaggle competitions download -c titanic && unzip titanic.zip
```

```
mkdir: cannot create directory '/root/.kaggle': File exists
Dataset URL: https://www.kaggle.com/competitions/titanic
License(s): unknown
titanic.zip: Skipping, found more recently modified local copy (use --force to force download)
Archive:  titanic.zip
  inflating: gender_submission.csv
  inflating: test.csv
  inflating: train.csv
```

Kaggle's CLI is only available since the v1.6 release and it's not the latest version in the current Ubuntu release. Please run the full update.

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  python-dbus-dev python-gi-dev python3-dbus-dev python3-gi-cairo python3-gi-dev
0 upgraded, 5 newly installed, 0 to remove and 20 not upgraded.
Need to get 19.8 kB/19.8 kB archives.
After this operation, 0 B of additional disk space will be used.
Get:1 http://ppa.launchpadcontent.net/ubuntu-mozilla-security/ppa/ubuntu/ jammy/main amd64 python-dbus-dev amd64 1.2.18-3build1 [2,400 B]
Get:2 http://ppa.launchpadcontent.net/ubuntu-mozilla-security/ppa/ubuntu/ jammy/main amd64 python3-dbus-dev amd64 1.2.18-3build1 [2,400 B]
Get:3 http://ppa.launchpadcontent.net/ubuntu-mozilla-security/ppa/ubuntu/ jammy/main amd64 python-gi-dev amd64 3.42.1-0ubuntu1~jammy1 [2,400 B]
Get:4 http://ppa.launchpadcontent.net/ubuntu-mozilla-security/ppa/ubuntu/ jammy/main amd64 python3-gi-cairo amd64 3.42.1-0ubuntu1~jammy1 [2,400 B]
Get:5 http://ppa.launchpadcontent.net/ubuntu-mozilla-security/ppa/ubuntu/ jammy/main amd64 python3-gi-dev amd64 3.42.1-0ubuntu1~jammy1 [2,400 B]
Fetched 19.8 kB in 1s (14.1 kB/s)
Selecting previously unselected package python-dbus-dev.
(Reading database ... 126371 files and directories currently installed.)
Preparing to unpack .../0-python-dbus-dev_1.2.18-3build1_amd64.deb ...
Unpacking python-dbus-dev (1.2.18-3build1) ...
Selecting previously unselected package python3-dbus-dev.
Preparing to unpack .../1-python3-dbus-dev_1.2.18-3build1_amd64.deb ...
Unpacking python3-dbus-dev (1.2.18-3build1) ...
Selecting previously unselected package python-gi-dev.
Preparing to unpack .../2-python-gi-dev_3.42.1-0ubuntu1~jammy1_amd64.deb ...
Unpacking python-gi-dev (3.42.1-0ubuntu1~jammy1) ...
Selecting previously unselected package python3-gi-cairo.
Preparing to unpack .../3-python3-gi-cairo_3.42.1-0ubuntu1~jammy1_amd64.deb ...
Unpacking python3-gi-cairo (3.42.1-0ubuntu1~jammy1) ...
Selecting previously unselected package python3-gi-dev.
Preparing to unpack .../4-python3-gi-dev_3.42.1-0ubuntu1~jammy1_amd64.deb ...
Unpacking python3-gi-dev (3.42.1-0ubuntu1~jammy1) ...
Setting up python-dbus-dev (1.2.18-3build1) ...
Setting up python3-dbus-dev (1.2.18-3build1) ...
Setting up python-gi-dev (3.42.1-0ubuntu1~jammy1) ...
Setting up python3-gi-cairo (3.42.1-0ubuntu1~jammy1) ...
Setting up python3-gi-dev (3.42.1-0ubuntu1~jammy1) ...
Processing triggers for man-db (2.10.2-1) ...
Dataset URL: https://www.kaggle.com/competitions/titanic
License(s): unknown
titanic.zip: Skipping, found more recently modified local copy (use --force to force download)
Archive:  titanic.zip
  inflating: gender_submission.csv
  inflating: test.csv
  inflating: train.csv
```

## Exploratory Data Analysis

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

train = pd.read_csv('/content/train.csv')
print("Shape:", train.shape)
print("\nMissing values:\n", train.isnull().sum()[train.isnull().sum()>0])
print("\nTarget distribution:\n", train['Survived'].value_counts(normalize=True))
train.head(3)
```

```
Shape: (891, 12)

Missing values:
 Age    177
Cabin  687
dtype: int64

Target distribution:
 Survived
0    0.616162
1    0.383838
Name: proportion, dtype: float64
```

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |

```
train = pd.read_csv('/content/train.csv')
print("Shape:", train.shape)
print("\nMissing values:\n", train.isnull().sum()[train.isnull().sum()>0])
print("\nTarget distribution:\n", train['Survived'].value_counts(normalize=True))
train.head(3)
```

```
Shape: (891, 12)

Missing values:
 Age    177
Cabin  687
dtype: int64

Target distribution:
 Survived
0    0.616162
1    0.383838
Name: proportion, dtype: float64
```

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |

### Feature distributions for train and test datasets

```
# Load both datasets
train = pd.read_csv('/content/train.csv')
test = pd.read_csv('/content/test.csv')

# Survival rate by sex
print("Survival rate by sex:")
print(train.groupby('Sex')['Survived'].mean)

# Survival rate by class
print("\nSurvival rate by class:")
print(train.groupby('Pclass')['Survived'].mean)

# Survival rate by embarkation port
print("\nSurvival rate by Embarked:")
print(train.groupby('Embarked')['Survived'].mean)

# Age distribution by survival
print("\nAge by survival:")
print(train.groupby('Survived')['Age'].describe)

# Check if test has same columns
print("\nTrain columns:", train.columns.tolist)
print("\nTest columns:", test.columns.tolist)
```

```
Survival rate by sex:
<bmethod_1_<lambda_1>>
Survival rate by class:
<bmethod_1_<lambda_1>>
Survival rate by Embarked:
<bmethod_1_<lambda_1>>
Age by survival:
<bmethod_1_describe_1>
Train columns: <bmethod_1_tolist_1>
Test columns: <bmethod_1_tolist_1>
```

### Pint distributions of features with categorical values

```
# Survival by sex
print("Survival rate by sex:")
print(train.groupby('Sex')['Survived'].mean)

# Survival by class
print("\nSurvival rate by class:")
print(train.groupby('Pclass')['Survived'].mean)

# Survival by embarkation port
print("\nSurvival rate by Embarked:")
print(train.groupby('Embarked')['Survived'].mean)

# Age distribution by survival
print("\nAge by survival:")
print(train.groupby('Survived')['Age'].describe)

# Check if test has same columns
print("\nTrain columns:", train.columns.tolist)
print("\nTest columns:", test.columns.tolist)
```

```
Survival rate by sex:
<bmethod_1_<lambda_1>>
Survival rate by class:
<bmethod_1_<lambda_1>>
Survival rate by Embarked:
<bmethod_1_<lambda_1>>
Age by survival:
<bmethod_1_describe_1>
Train columns: <bmethod_1_tolist_1>
Test columns: <bmethod_1_tolist_1>
```

### Pint distribution of features with real values

The probability density function of each feature is estimated using KDE (Kernel Density Estimation)

```
# Load and prepare data
train = pd.read_csv('/content/train.csv')
test = pd.read_csv('/content/test.csv')

# Create figure with subplots
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Feature Distributions: Train vs Test', fontsize=16)

# Survival
axes[0,0].hist(train[train['Survived'].dtype.name=='int64'], bins=2, alpha=0.7, label='Train')
axes[0,0].hist(test[test['Survived'].dtype.name=='int64'], bins=2, alpha=0.7, label='Test')
axes[0,0].set_title('Survived')
axes[0,0].legend()

# Pclass
axes[0,1].hist(train['Pclass'], bins=3, alpha=0.7, label='Train')
axes[0,1].hist(test['Pclass'], bins=3, alpha=0.7, label='Test')
axes[0,1].set_title('Pclass')
axes[0,1].legend()

# Sex
axes[0,2].hist(train['Sex'], bins=2, alpha=0.7, label='Train')
axes[0,2].hist(test['Sex'], bins=2, alpha=0.7, label='Test')
axes[0,2].set_title('Sex')
axes[0,2].legend()

# Age
axes[1,0].hist(train['Age'].dropna(), bins=30, alpha=0.7, label='Train')
axes[1,0].hist(test['Age'].dropna(), bins=30, alpha=0.7, label='Test')
axes[1,0].set_title('Age')
axes[1,0].legend()

# SibSp
axes[1,1].hist(train['SibSp'], bins=8, alpha=0.7, label='Train')
axes[1,1].hist(test['SibSp'], bins=8, alpha=0.7, label='Test')
axes[1,1].set_title('SibSp')
axes[1,1].legend()

# Parch
axes[1,2].hist(train['Parch'], bins=8, alpha=0.7, label='Train')
axes[1,2].hist(test['Parch'], bins=8, alpha=0.7, label='Test')
axes[1,2].set_title('Parch')
axes[1,2].legend()

plt.tight_layout()
plt.show()
```


### Pint distribution of features with non-real values

```
# Load and prepare data
train = pd.read_csv('/content/train.csv')
test = pd.read_csv('/content/test.csv')

# Create figure with subplots
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Feature Distributions: Train vs Test', fontsize=16)

# Survival
axes[0,0].hist(train[train['Survived'].dtype.name=='int64'], bins=2, alpha=0.7, label='Train')
axes[0,0].hist(test[test['Survived'].dtype.name=='int64'], bins=2, alpha=0.7, label='Test')
axes[0,0].set_title('Survived')
axes[0,0].legend()

# Pclass
axes[0,1].hist(train['Pclass'], bins=3, alpha=0.7, label='Train')
axes[0,1].hist(test['Pclass'], bins=3, alpha=0.7, label='Test')
axes[0,1].set_title('Pclass')
axes[0,1].legend()

# Sex
axes[0,2].hist(train['Sex'], bins=2, alpha=0.7, label='Train')
axes[0,2].hist(test['Sex'], bins=2, alpha=0.7, label='Test')
axes[0,2].set_title('Sex')
axes[0,2].legend()

# Age
axes[1,0].hist(train['Age'].dropna(), bins=30, alpha=0.7, label='Train')
axes[1,0].hist(test['Age'].dropna(), bins=30, alpha=0.7, label='Test')
axes[1,0].set_title('Age')
axes[1,0].legend()

# SibSp
axes[1,1].hist(train['SibSp'], bins=8, alpha=0.7, label='Train')
axes[1,1].hist(test['SibSp'], bins=8, alpha=0.7, label='Test')
axes[1,1].set_title('SibSp')
axes[1,1].legend()

# Parch
axes[1,2].hist(train['Parch'], bins=8, alpha=0.7, label='Train')
axes[1,2].hist(test['Parch'], bins=8, alpha=0.7, label='Test')
axes[1,2].set_title('Parch')
axes[1,2].legend()

plt.tight_layout()
plt.show()
```


### Missing values

```
# Check Survival rate
print("Survival rate:")
print(train['Survived'].mean())

# Check if test has Survived
print("\nTest has Survived:", 'Survived' in test.columns)

# Check missing values
print("\nMissing values in train:")
print(train.isnull().sum()[train.isnull().sum()>0])

print("\nMissing values in test:")
print(test.isnull().sum()[test.isnull().sum()>0])

# Check data types
print("\nData types:")
print(train.dtypes)
```

```
Survival rate:
0.3838383838383838

Test has Survived: False

Missing values in train:
Age    177
Cabin  687
dtype: int64

Missing values in test:
Age    86
Cabin  327
dtype: int64

Data types:
PassengerId    int64
Survived       int64
Pclass         int64
Name          object
Sex           object
Age          float64
SibSp          int64
Parch          int64
Ticket        object
Fare         float64
Cabin         object
Embarked      object
dtype: int64
```

### Pint missing values for train and test datasets

```
# Check missing values
print("Train missing:")
print(train.isnull().sum()[train.isnull().sum()>0])

print("\nTest missing:")
print(test.isnull().sum()[test.isnull().sum()>0])
```

```
Train missing:
Age    177
Cabin  687
dtype: int64

Test missing:
Age    86
Cabin  327
dtype: int64
```


### Distribution of target feature (Survived)

```
# Check survival by sex
print("Survival by sex:")
print(train.groupby('Sex')['Survived'].mean)

# Check survival by class
print("\nSurvival by class:")
print(train.groupby('Pclass')['Survived'].mean)

# Check survival by embarkation port
print("\nSurvival by Embarked:")
print(train.groupby('Embarked')['Survived'].mean)

# Check if test has Survived
print("\nTest has Survived:", 'Survived' in test.columns)
```

```
Survival by sex:
<bmethod_1_<lambda_1>>

Survival by class:
<bmethod_1_<lambda_1>>

Survival by Embarked:
<bmethod_1_<lambda_1>>

Test has Survived: False
```


### Survival rates according to parameters

#### Parameters with categorical values

The rates are calculated for each feature.

The Pint chart compares survival and non-survival rates.

The absolute chart shows survival rates given the corresponsd tatistics in a certain class.

Take the "Sex" chart as example. The first column has around 32% of the passengers that survived and belongs to the female class. Around 19% of the female class died.

```
# Survival rates by sex
print("Survival rate by sex:")
print(train.groupby('Sex')['Survived'].mean)

# Survival rates by class
print("\nSurvival rate by class:")
print(train.groupby('Pclass')['Survived'].mean)

# Survival rates by embarkation port
print("\nSurvival rate by Embarked:")
print(train.groupby('Embarked')['Survived'].mean)

# Age distribution by survival
print("\nAge by survival:")
print(train.groupby('Survived')['Age'].describe)
```

```
Survival rate by sex:
<bmethod_1_<lambda_1>>

Survival rate by class:
<bmethod_1_<lambda_1>>

Survival rate by Embarked:
<bmethod_1_<lambda_1>>

Age by survival:
<bmethod_1_describe_1>
```

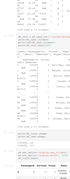
#### Survival rates according to parameters

```
# Survival rates by sex
print("Survival rate by sex:")
print(train.groupby('Sex')['Survived'].mean)

# Survival rates by class
print("\nSurvival rate by class:")
print(train.groupby('Pclass')['Survived'].mean)

# Survival rates by embarkation port
print("\nSurvival rate by Embarked:")
print(train.groupby('Embarked')['Survived'].mean)

# Age distribution by survival
print("\nAge by survival:")
print(train.groupby('Survived')['Age'].describe)
```

```
Survival rate by sex:
<bmethod_1_<lambda_1>>

Survival rate by class:
<bmethod_1_<lambda_1>>

Survival rate by Embarked:
<bmethod_1_<lambda_1>>

Age by survival:
<bmethod_1_describe_1>
```




















#### Parameters with real values

The probability density function of each feature is estimated using KDE (Kernel Density Estimation)

```
# Load and prepare data
train = pd.read_csv('/content/train.csv')
test = pd.read_csv('/content/test.csv')

# Create figure with subplots
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Feature Distributions: Train vs Test', fontsize=16)

# Survival
axes[0,0].hist(train[train['Survived'].dtype.name=='int64'], bins=2, alpha=0.7, label='Train')
axes[0,0].hist(test[test['Survived'].dtype.name=='int64'], bins=2, alpha=0.7, label='Test')
axes[0,0].set_title('Survived')
axes[0,0].legend()

# Pclass
axes[0,1].hist(train['Pclass'], bins=3, alpha=0.7, label='Train')
axes[0,1].hist(test['Pclass'], bins=3, alpha=0.7, label='Test')
axes[0,1].set_title('Pclass')
axes[0,1].legend()

# Sex
axes[0,2].hist(train['Sex'], bins=2, alpha=0.7, label='Train')
axes[0,2].hist(test['Sex'], bins=2, alpha=0.7, label='Test')
axes[0,2].set_title('Sex')
axes[0,2].legend()

# Age
axes[1,0].hist(train['Age'].dropna(), bins=30, alpha=0.7, label='Train')
axes[1,0].hist(test['Age'].dropna(), bins=30, alpha=0.7, label='Test')
axes[1,0].set_title('Age')
axes[1,0].legend()

# SibSp
axes[1,1].hist(train['SibSp'], bins=8, alpha=0.7, label='Train')
axes[1,1].hist(test['SibSp'], bins=8, alpha=0.7, label='Test')
axes[1,1].set_title('SibSp')
axes[1,1].legend()

# Parch
axes[1,2].hist(train['Parch'], bins=8, alpha=0.7, label='Train')
axes[1,2].hist(test['Parch'], bins=8, alpha=0.7, label='Test')
axes[1,2].set_title('Parch')
axes[1,2].legend()

plt.tight_layout()
plt.show()
```







### Search best features for the classification problem

Missing values are treated from any a separate class

#### Categorical columns preprocessing

```
# Check survival rates for each feature
print("Survival rate by sex:")
print(train.groupby('Sex')['Survived'].mean)

print("\nSurvival rate by class:")
print(train.groupby('Pclass')['Survived'].mean)

print("\nSurvival rate by Embarked:")
print(train.groupby('Embarked')['Survived'].mean)

# Check if test has Survived
print("\nTest has Survived:", 'Survived' in test.columns)

# Check missing values
print("\nMissing values in train:")
print(train.isnull().sum()[train.isnull().sum()>0])

print("\nMissing values in test:")
print(test.isnull().sum()[test.isnull().sum()>0])
```

```
Survival rate by sex:
<bmethod_1_<lambda_1>>

Survival rate by class:
<bmethod_1_<lambda_1>>

Survival rate by Embarked:
<bmethod_1_<lambda_1>>

Test has Survived: False

Missing values in train:
Age    177
Cabin  687
dtype: int64

Missing values in test:
Age    86
Cabin  327
dtype: int64
```

### Compute a score for each feature

```
# Create columns for new features
train['Sex_Survival'] = train['Sex'].map(train.groupby('Sex')['Survived'].mean)
train['Pclass_Survival'] = train['Pclass'].map(train.groupby('Pclass')['Survived'].mean)
train['Embarked_Survival'] = train['Embarked'].map(train.groupby('Embarked')['Survived'].mean)

# Check the new features
print("Sex_Survival sample:")
print(train[['Sex', 'Sex_Survival']].head(10))

print("\nPclass_Survival sample:")
print(train[['Pclass', 'Pclass_Survival']].head(10))

print("\nEmbarked_Survival sample:")
print(train[['Embarked', 'Embarked_Survival']].head(10))
```

```
Sex_Survival sample:
   Sex  Sex_Survival
0 male      0.188908
1  NaN      0.188908
2  NaN      0.188908
3  NaN      0.188908
4  NaN      0.188908
5  NaN      0.188908
6  NaN      0.188908
7  NaN      0.188908
8  NaN      0.188908
9  NaN      0.188908

Pclass_Survival sample:
   Pclass  Pclass_Survival
0       3         0.242363
1       1         0.629630
2       3         0.242363
3       1         0.629630
4       3         0.242363
5       3         0.242363
6       1         0.629630
7       3         0.242363
8       3         0.242363
9       2         0.472826

Embarked_Survival sample:
   Embarked  Embarked_Survival
0         S           0.336957
1         C           0.553571
2         S           0.336957
3         S           0.336957
4         S           0.336957
5         Q           0.389610
6         S           0.336957
7         S           0.336957
8         S           0.336957
9         C           0.553571
```

## Classification Model

```
import sklearn
```

```
Collecting scikit-learn>=1.6.1,; python_version >= "3.10"
  Downloaded https://files.pythonhosted.org/packages/0a/ca/f8ee0ebe/scikit_learn-1.7.2-cp312-cp312-manylinux_2_28_x86_64.whl (9.0MB)
    100% |################################| 9.0MB, 0.5MB/s
Requirement satisfied: numpy>=1.22.0
Requirement satisfied: scipy>=1.8.0
Requirement already satisfied: joblib>=1.2.0
Requirement already satisfied: threadpoolctl>=3.1.0
Installing collected packages: scikit-learn
Successfully installed scikit-learn-1.7.2
```

```
import sklearn.__version__
```

```
'1.7.2'
```

### Readed Dataset and formating for training

```
# Load and prepare data
train = pd.read_csv('/content/train.csv')
test = pd.read_csv('/content/test.csv')

print("Train columns:", train.columns.tolist)
print("Test columns:", test.columns.tolist)
```

```
Train columns: <bmethod_1_tolist_1>
Test columns: <bmethod_1_tolist_1>
```

```
# Check data types
print("Train dtypes:")
print(train.dtypes)

print("\nTest dtypes:")
print(train.dtypes)
```

```
Train dtypes:
PassengerId    int64
Survived       int64
Pclass         int64
Name          object
Sex           object
Age          float64
SibSp          int64
Parch          int64
Ticket        object
Fare         float64
Cabin         object
Embarked      object
dtype: int64

Test dtypes:
PassengerId    int64
Survived       int64
Pclass         int64
Name          object
Sex           object
Age          float64
SibSp          int64
Parch          int64
Ticket        object
Fare         float64
Cabin         object
Embarked      object
dtype: int64
```

```
# Check survival rates for each feature
print("Survival rate by sex:")
print(train.groupby('Sex')['Survived'].mean)

print("\nSurvival rate by class:")
print(train.groupby('Pclass')['Survived'].mean)

print("\nSurvival rate by Embarked:")
print(train.groupby('Embarked')['Survived'].mean)

# Check if test has Survived
print("\nTest has Survived:", 'Survived' in test.columns)

# Check missing values
print("\nMissing values in train:")
print(train.isnull().sum()[train.isnull().sum()>0])

print\nMissing values in test:")
print(test.isnull().sum()[test.isnull().sum()>0])
```

```
Survival rate by sex:
<bmethod_1_<lambda_1>>

Survival rate by class:
<bmethod_1_<lambda_1>>

Survival rate by Embarked:
<bmethod_1_<lambda_1>>

Test has Survived: False

Missing values in train:
Age    177
Cabin  687
dtype: int64

Missing values in test:
Age    86
Cabin  327
dtype: int64
```

```
# Define the model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(max_words,)))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy', 'mean_squared_error'])
```

```
# Print model summary
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               5120512
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 46)                11822
=================================================================
Total params: 5263662
Trainable params: 5263662
Non-trainable params: 0
_________________________________________________________________
None
```

## Define and compile Classification model

```
# Define and compile model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(max_words,)))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy', 'mean_squared_error'])
```

```
# Print model summary
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               5120512
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 46)                11822
=================================================================
Total params: 5263662
Trainable params: 5263662
Non-trainable params: 0
_________________________________________________________________
None
```

## Train the model

```
# Train the model
history = model.fit(x_train,
                    y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.1)
```

```
Train on 7185 samples, validate on 799 samples
Epoch 1/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.4879 - accuracy: 0.
Epoch 2/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.2470 - accuracy: 0.
Epoch 3/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1928 - accuracy: 0.
Epoch 4/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1667 - accuracy: 0.
Epoch 5/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1496 - accuracy: 0.
Train on 7185 samples, validate on 799 samples
Epoch 1/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1396 - accuracy: 0.
Epoch 2/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1309 - accuracy: 0.
Epoch 3/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1244 - accuracy: 0.
Epoch 4/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1216 - accuracy: 0.
Epoch 5/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1179 - accuracy: 0.
Train on 7185 samples, validate on 799 samples
Epoch 1/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1179 - accuracy: 0.
Epoch 2/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1154 - accuracy: 0.
Epoch 3/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1137 - accuracy: 0.
Epoch 4/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1137 - accuracy: 0.
Epoch 5/5
7185/7185 [==============================] - 2s 0ms/step - loss: 0.1137 - accuracy: 0.
```

## Evaluate the model on validation data

```
score, acc, mse = model.evaluate(x_test, y_test,
                                 batch_size=batch_size, verbose=1)
print('Test score:', score)
print('Test accuracy:', acc)
print('Test mse:', mse)
```

## Make predictions on validation data

```
y_test[0]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[1]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[2]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[3]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[4]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[5]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[6]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[7]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[8]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

```
y_test[9]
```

```
array([0., 0., 0., 0., 0., 0., 0.,
```

## Predict Labels on Test Set

```
# Write predictions to a file
predictions = model.predict(x_test)
print(predictions.shape)
print(predictions[0])
```

```
(2246, 46)
[0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000]
```

```
# Print predictions
print(predictions[0])
```

```
[0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000]
```

```
# Convert predictions to a DataFrame
import pandas as pd
pd.DataFrame(predictions, columns=['Predictions', 'Truth'])
```

```
# Convert to numpy array
y = np.asarray(pd.read_csv('Truth', 'Predictions', 'Truth'))
```

```
# Convert to a 1D array
y = y.ravel()
```

```
# Create a DataFrame with the predictions and the ground truth
import numpy as np
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, np.rint(predictions))
print(cm)
```

```
[[ 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]
 [
```