Week2 titanic

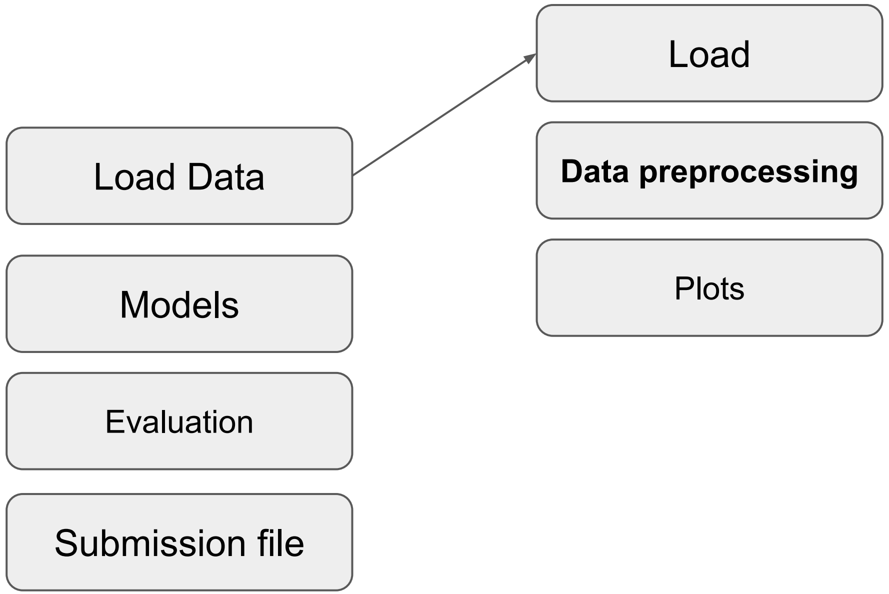
SangHyuk Lee

I introduce this notebook,

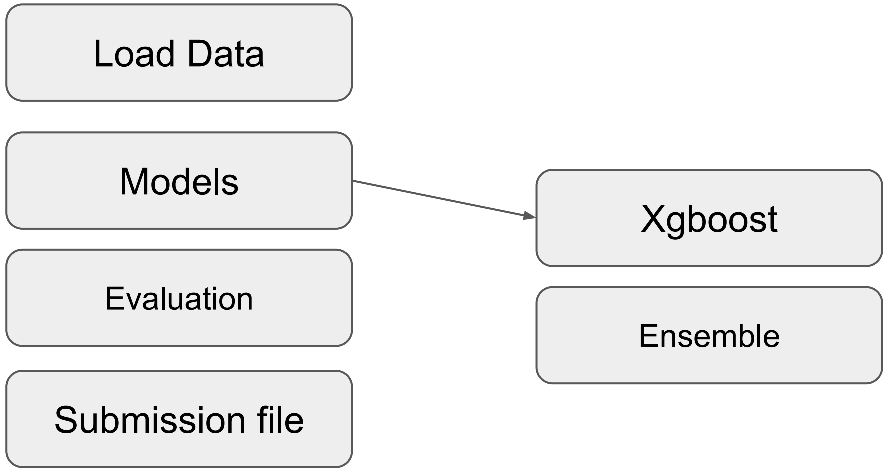
[Titanic - The Only Notebook You Need To See]

Contents

Procedure


Load Data
Models
Evaluation
Submission file
Load
Data preprocessing
Plots

Procedure


Load Data
Models
Evaluation
Submission file
Xgboost
Ensemble

A test score

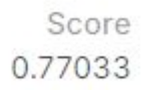
Given procedure in the code, I submitted a prediction for test data via xgboost.

Load Data

Xgboost

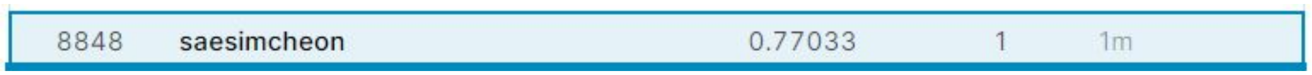
Evaluation

Submission file



Score
0.77033

In leaderboard,



```
8848    saesimcheon    0.77033    1    1m
```

Libraries

Data pre-process and structure

- pandas
- numpy
- re

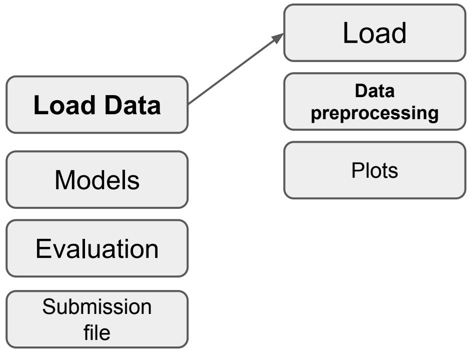
Modeling

- sklearn
- xgboost

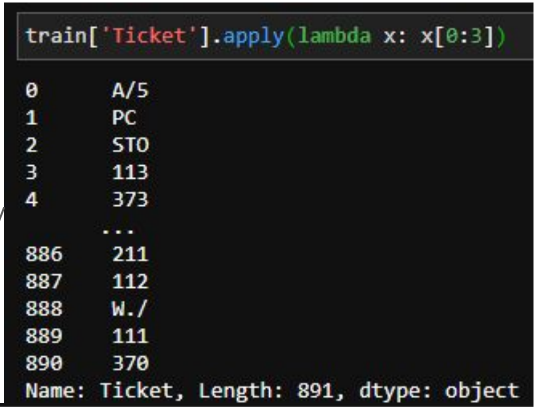
Visualization

- seaborn
- matplotlib.pyplot
- plotly

Load Data - Data preprocessing : Ticket -> Ticket_type

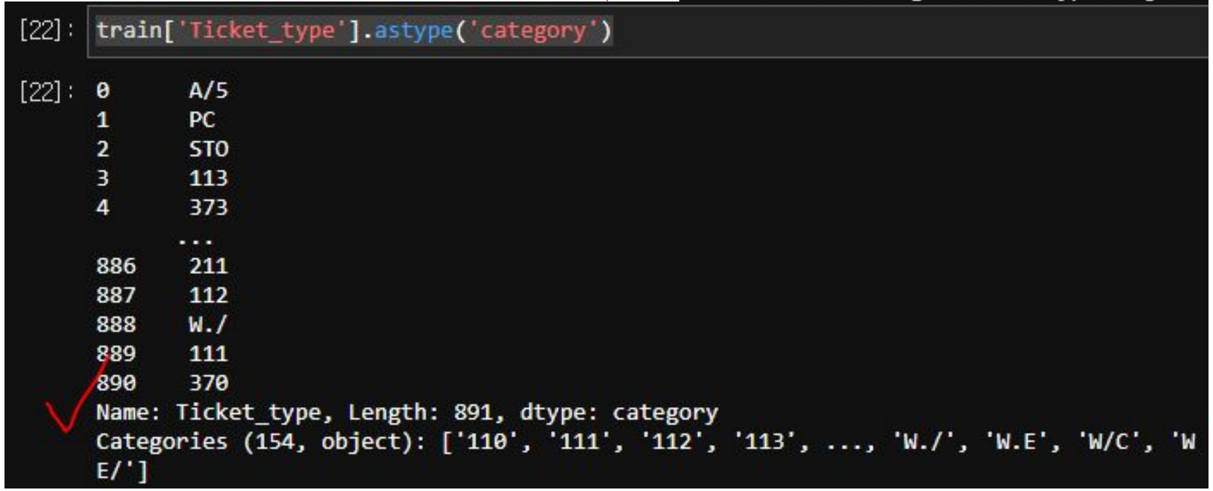


First of all, the author sliced **three characters of ticket** variable



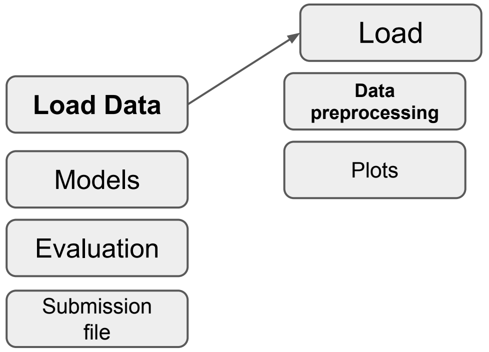
```
train['Ticket'].apply(lambda x: x[0:3])

0      A/5
1      PC
2      STO
3      113
4      373
      ...
886    211
887    112
888    W./
889    111
890    370
Name: Ticket, Length: 891, dtype: object
```

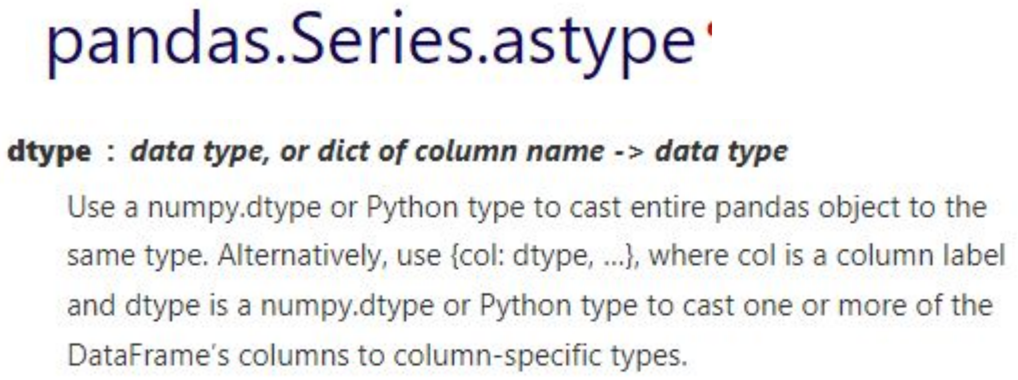
To convert type of ticket into **category**, the author used **astype method.**



```
[22]: train['Ticket_type'].astype('category')

[22]: 0      A/5
      1      PC
      2      STO
      3      113
      4      373
            ...
      886    211
      887    112
      888    W./
      889    111
      890    370
      Name: Ticket_type, Length: 891, dtype: category
      Categories (154, object): ['110', '111', '112', '113', ..., 'W./', 'W.E', 'W/C', 'W
      E/']
```

Load Data - Data preprocessing : Ticket -> Ticket_type

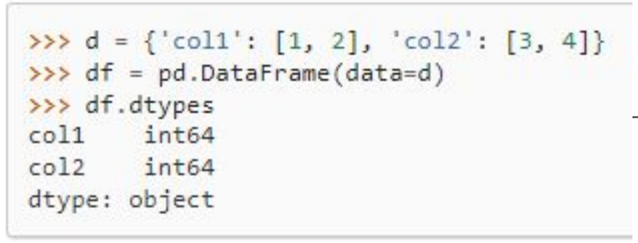




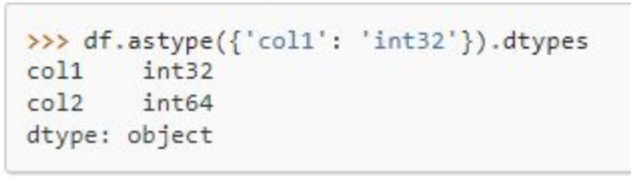
pandas.Series.astype

dtype : ***data type, or dict of column name -> data type***

Use a numpy.dtype or Python type to cast entire pandas object to the same type. Alternatively, use {col: dtype, ...}, where col is a column label and dtype is a numpy.dtype or Python type to cast one or more of the DataFrame's columns to column-specific types.

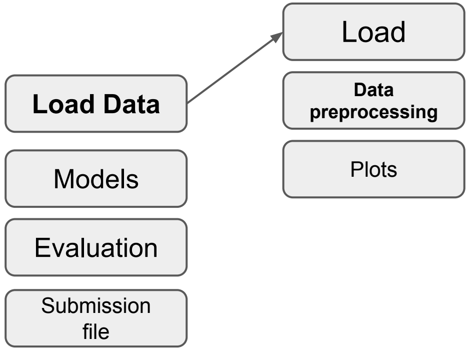


```
>>> d = {'col1': [1, 2], 'col2': [3, 4]}
>>> df = pd.DataFrame(data=d)
>>> df.dtypes
col1    int64
col2    int64
dtype: object
```

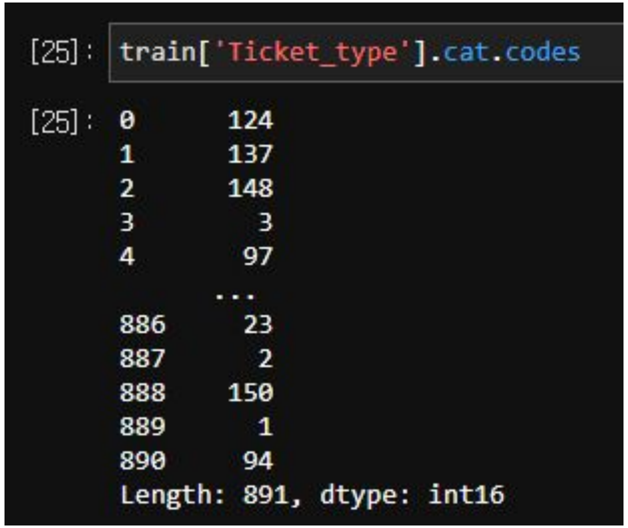


```
>>> df.astype({'col1': 'int32'}).dtypes
col1    int32
col2    int64
dtype: object
```

Load Data - Data preprocessing : Ticket -> Ticket_type

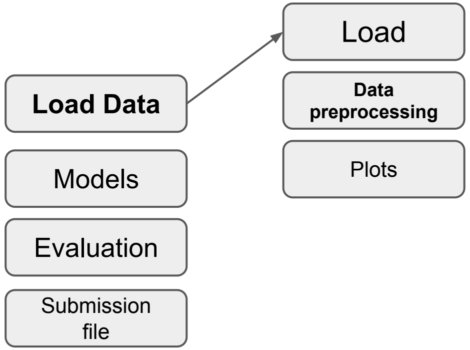


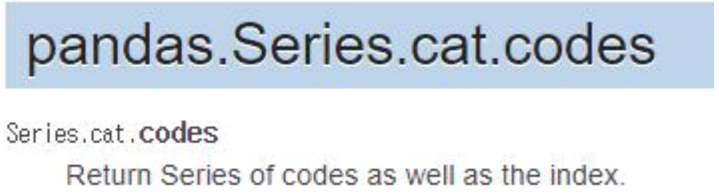
Through using **.cat.codes attribute, the ticket type is converted into int16**



```
[25]: train['Ticket_type'].cat.codes

[25]: 0      124
      1      137
      2      148
      3        3
      4       97
            ...
      886     23
      887      2
      888    150
      889      1
      890     94
      Length: 891, dtype: int16
```

Load Data - Data preprocessing : Ticket -> Ticket_type

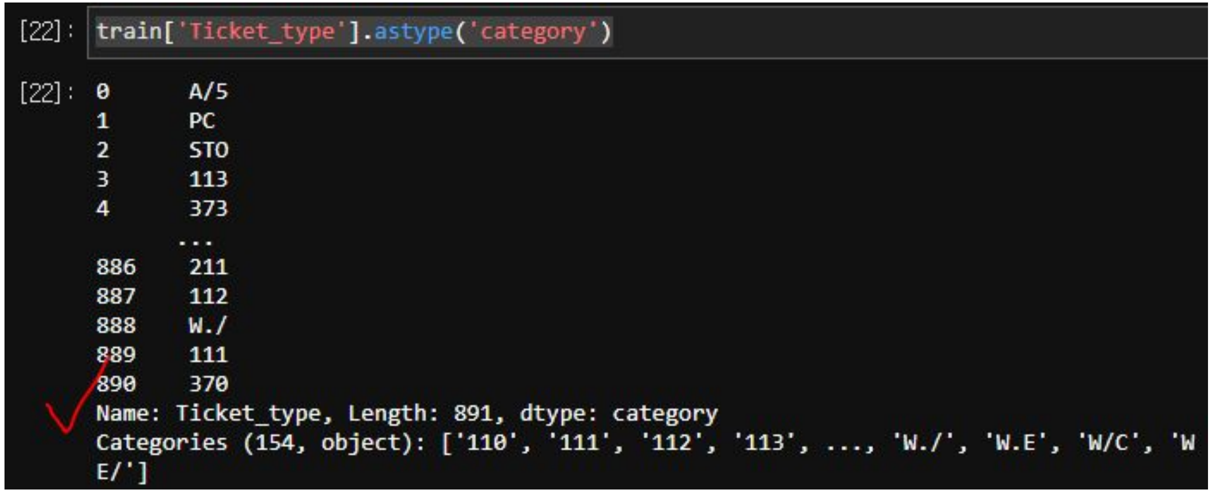




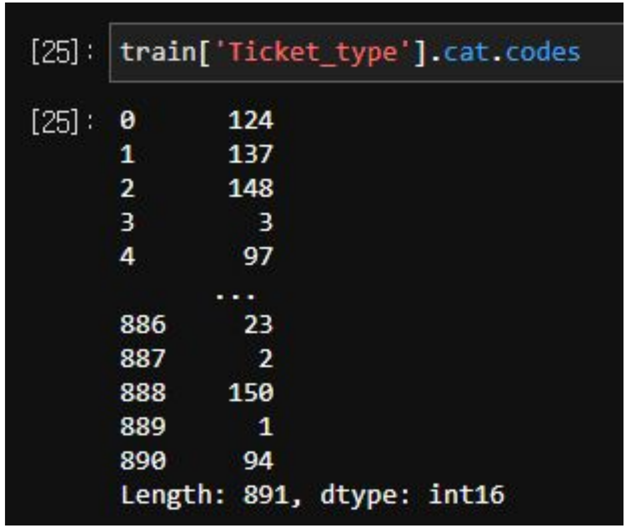
pandas.Series.cat.codes

`Series.cat.codes`

Return Series of codes as well as the index.

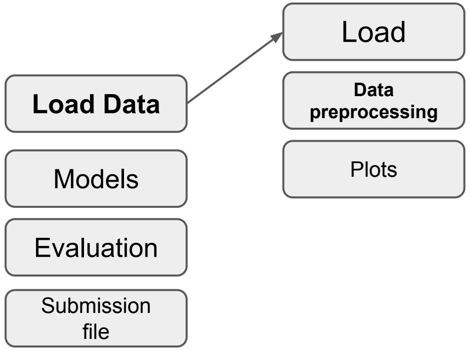


```
[22]: train['Ticket_type'].astype('category')

[22]: 0      A/5
      1      PC
      2      STO
      3      113
      4      373
            ...
      886    211
      887    112
      888    W./
      889    111
      890    370
      Name: Ticket_type, Length: 891, dtype: category
      Categories (154, object): ['110', '111', '112', '113', ..., 'W./', 'W.E', 'W/C', 'W
      E/']
```

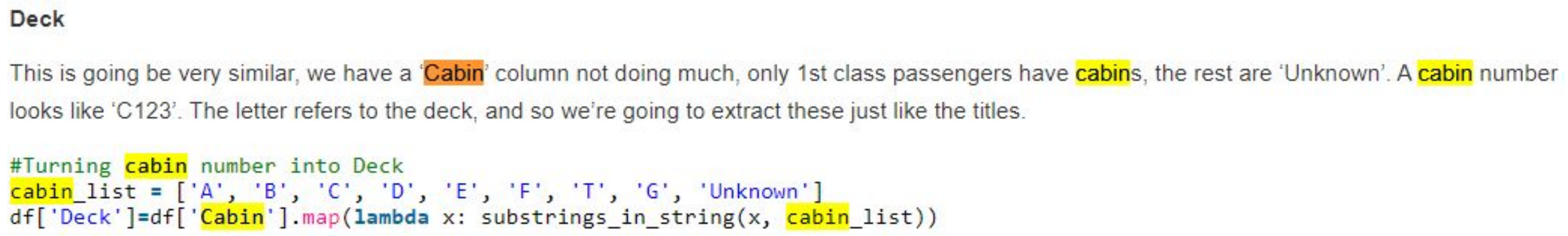


```
[25]: train['Ticket_type'].cat.codes

[25]: 0      124
      1      137
      2      148
      3        3
      4       97
            ...
      886     23
      887      2
      888    150
      889      1
      890     94
      Length: 891, dtype: int16
```

Load Data - Data preprocessing : List of variables added



- Length of name
- **Whether a passenger had a cabin on the Titanic**
- Family Size
- Boarding alone
- Categorical Age
- **Title**
- Categorical Fare

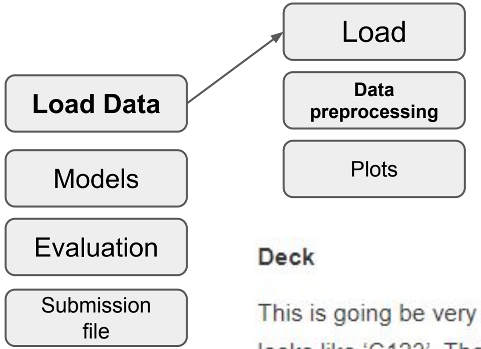


Deck

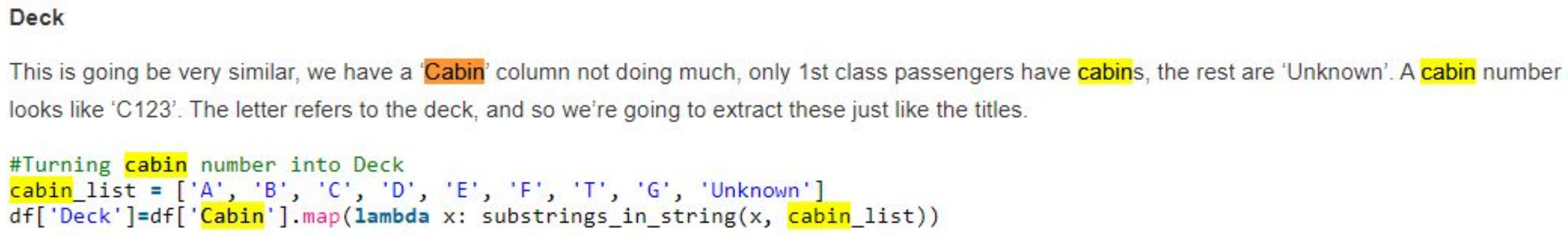
This is going be very similar, we have a 'Cabin' column not doing much, only 1st class passengers have cabins, the rest are 'Unknown'. A cabin number looks like 'C123'. The letter refers to the deck, and so we're going to extract these just like the titles.

```
#Turning cabin number into Deck
cabin_list = ['A', 'B', 'C', 'D', 'E', 'F', 'T', 'G', 'Unknown']
df['Deck']=df['Cabin'].map(lambda x: substrings_in_string(x, cabin_list))
```

Load Data - Data preprocessing : List of variables added



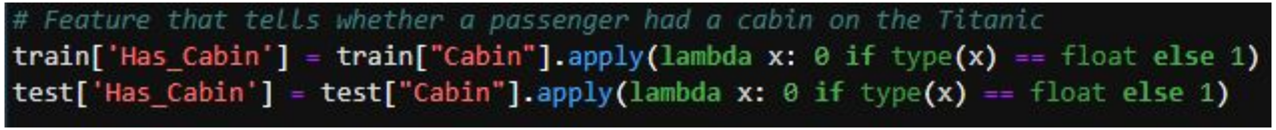
- Whether a passenger had a cabin on the Titanic



Deck

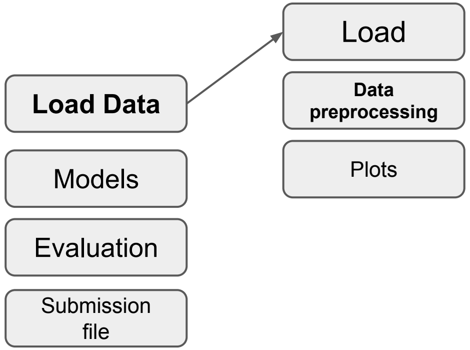
This is going be very similar, we have a 'Cabin' column not doing much, only 1st class passengers have cabins, the rest are 'Unknown'. A cabin number looks like 'C123'. The letter refers to the deck, and so we're going to extract these just like the titles.

```
#Turning cabin number into Deck
cabin_list = ['A', 'B', 'C', 'D', 'E', 'F', 'T', 'G', 'Unknown']
df['Deck']=df['Cabin'].map(lambda x: substrings_in_string(x, cabin_list))
```

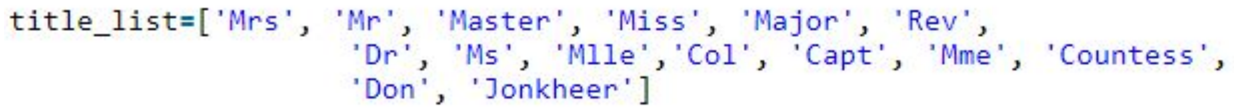


```
# Feature that tells whether a passenger had a cabin on the Titanic
train['Has_Cabin'] = train["Cabin"].apply(lambda x: 0 if type(x) == float else 1)
test['Has_Cabin'] = test["Cabin"].apply(lambda x: 0 if type(x) == float else 1)
```

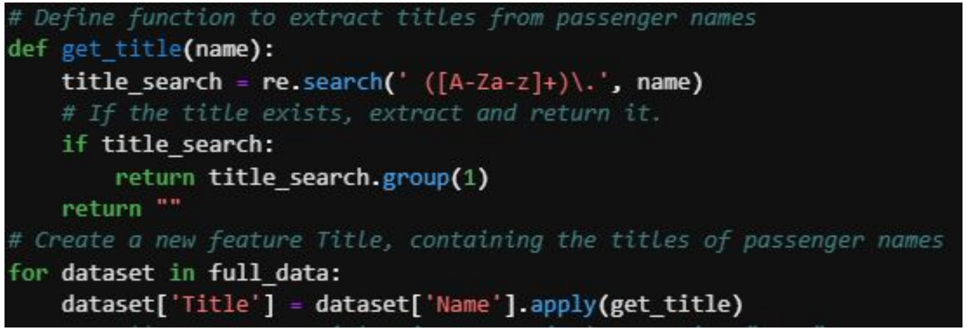
Load Data - Data preprocessing : List of variables added



- **Title**

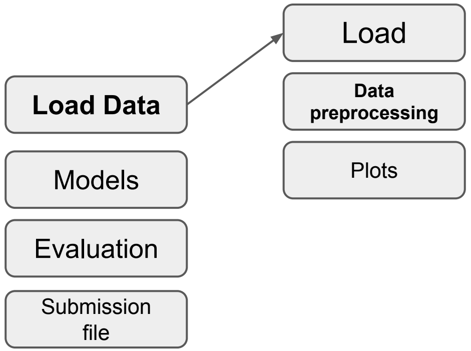


```
title_list=['Mrs', 'Mr', 'Master', 'Miss', 'Major', 'Rev',
                        'Dr', 'Ms', 'Mlle','Col', 'Capt', 'Mme', 'Countess',
                        'Don', 'Jonkheer']
```

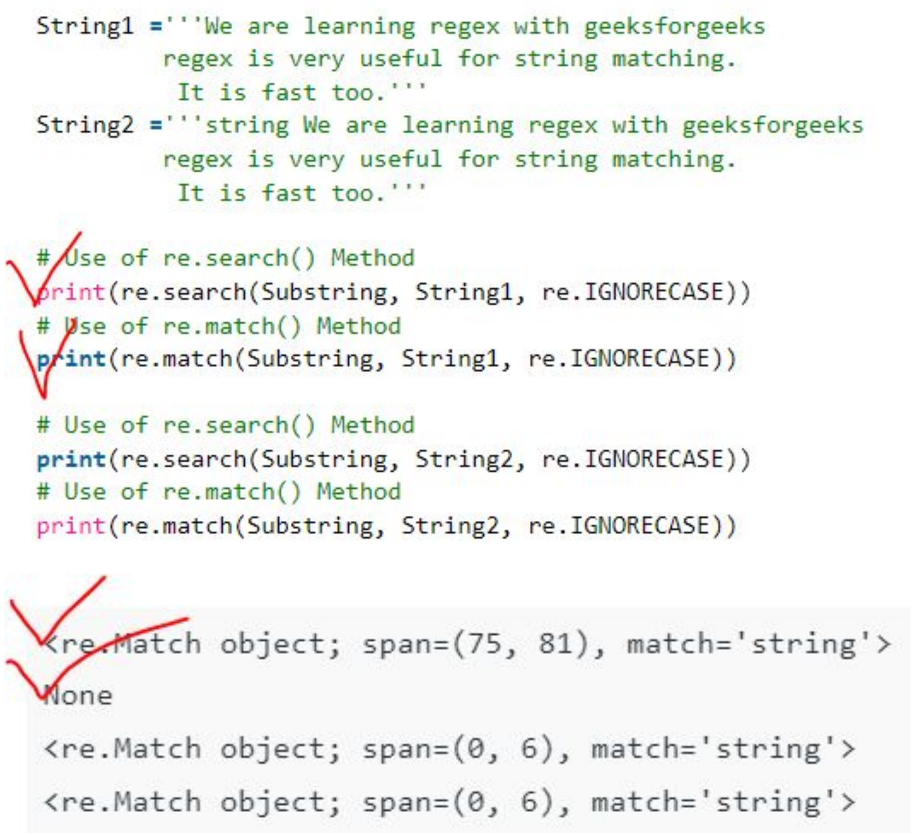


```
# Define function to extract titles from passenger names
def get_title(name):
    title_search = re.search(' ([A-Za-z]+)\.', name)
    # If the title exists, extract and return it.
    if title_search:
        return title_search.group(1)
    return ""
# Create a new feature Title, containing the titles of passenger names
for dataset in full_data:
    dataset['Title'] = dataset['Name'].apply(get_title)
```

Load Data - Data preprocessing : List of variables added



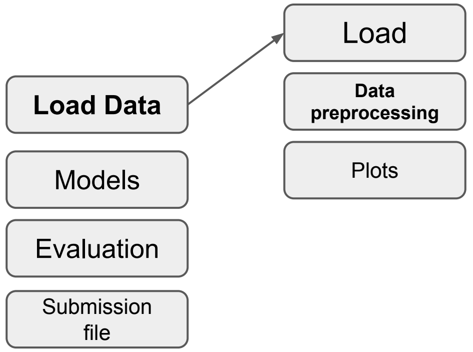
- **Title**



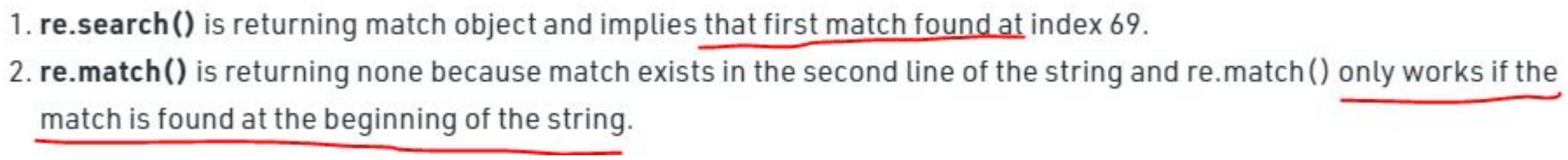
```
String1 ='''We are learning regex with geeksforgeeks
         regex is very useful for string matching.
          It is fast too.'''
String2 ='''string We are learning regex with geeksforgeeks
         regex is very useful for string matching.
          It is fast too.'''

# Use of re.search() Method
print(re.search(Substring, String1, re.IGNORECASE))
# Use of re.match() Method
print(re.match(Substring, String1, re.IGNORECASE))

# Use of re.search() Method
print(re.search(Substring, String2, re.IGNORECASE))
# Use of re.match() Method
print(re.match(Substring, String2, re.IGNORECASE))
```

```
<re.Match object; span=(75, 81), match='string'>
None
<re.Match object; span=(0, 6), match='string'>
<re.Match object; span=(0, 6), match='string'>
```

Load Data - Data preprocessing : List of variables added

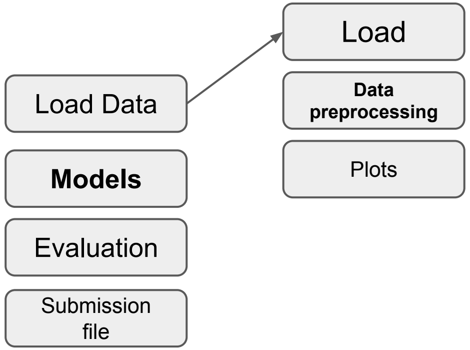


- **Title**

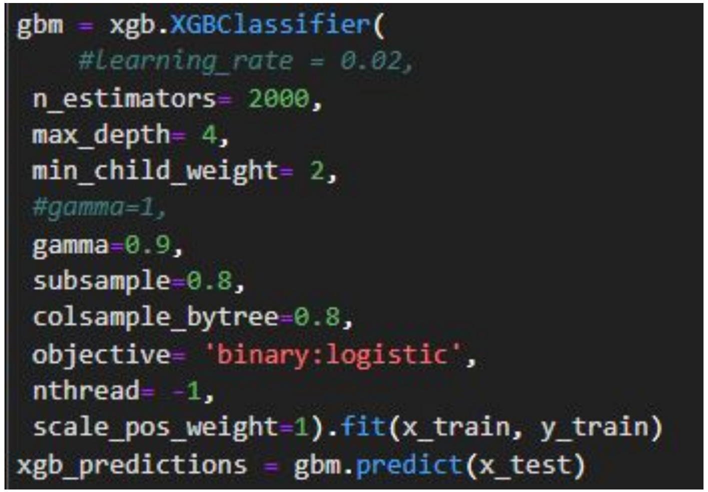


1. **re.search()** is returning match object and implies that first match found at index 69.
2. **re.match()** is returning none because match exists in the second line of the string and re.match() only works if the match is found at the beginning of the string.

Models - XGboost



Hyperparameters for XGboost



```
gbm = xgb.XGBClassifier(
    #learning_rate = 0.02,
 n_estimators= 2000,
 max_depth= 4,
 min_child_weight= 2,
 #gamma=1,
 gamma=0.9,
 subsample=0.8,
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread= -1,
 scale_pos_weight=1).fit(x_train, y_train)
xgb_predictions = gbm.predict(x_test)
```

For me,

For all data analysis,
variables given but not available must be transformed into **usable variables** to improve my accuracy.

Once some information is given, Just make it usable and then put in my model.

References

https://triangleinequality.wordpress.com/2013/09/08/basic-feature-engineerin
g-with-the-titanic-data/