Prakriti Shetty

Sep 3 · 8 min read · Listen

Save

A Comprehensive Guide on How To Get Started with Machine Learning

In this article, I'll take you through all the steps required for you to place a confident * foot forward into the world of Machine Learning, starting from the very basics of data science and data engineering to full- fledged ML.

At the very basis of any ML project are 6 very broad and simple steps:
1. Data Preprocessing: Data Wrangling and EDA

2. Train and test Split

3. Algorithm Setup

4. Model fitting

5. Model predictions

6. Model evaluation

I'll be using the Titanic Survival Prediction dataset from Kaggle to give examples along the way. The basic goal is to setup a classification model to predict whether a particular individual will survive or not based on certain given attributes.

So let's get started, shall we?

The very first thing you'd want to do is import all the libraries you'd require in your project. Now, it's quite possible that you won't be able to identify all of them at the start, but I'd suggest to keep this block variable and keep adding import statements

```
import matplotlib.pyplot as plt
import seaborn as sns
from typing import Text, List, Dict, Set, Tuple, Optional, Union
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import (StandardScaler, OneHotEncoder)
from sklearn.base import BaseEstimator
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay, f1_score
```

Let's now load our test and train datasets

```
train_data = pd.read_csv("/kaggle/input/titanic/train.csv")
test_data = pd.read_csv("/kaggle/input/titanic/test.csv")
```

Step 1 : Data Wrangling, Exploratory Data Analysis and Data Engineering

Let's first have a look at our dataset

```
train_data.head()
```

	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
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Feature Engineering

1.) Let's first handle all the missing values in the entire dataset

```
for i in train_data.columns:
    print(i)
    print(train_data[i].isnull().value_counts())
```

We notice that there are missing values in 'Age', 'Cabin' and 'Embarked'

```
#store top 10 ages and their frequencies
top_10_ages= train_data['Age'].value_counts().head(10).index.values
freq_of_top_10_ages=
train_data['Age'].value_counts().head(10).values

age_probabilities= list(map(lambda value:
(1/sum(freq_of_top_10_ages))*value ,freq_of_top_10_ages))

# np.random.choice(a, size=None, replace=True, p=None)
#p is the probabilities associated with each entry in an array
#Generates a random sample from a given 1-D array

train_data['Age']= train_data['Age'].apply( lambda value: value if
not np.isnan(value) else np.random.choice(top_10_ages, p=
age_probabilities))
```

What we did here was find the 10 most frequent ages and their corresponding frequencies. We then found out the probabilities associated with each age and updated the missing values of the 'Age' column with random filling from these 10 most frequent values, considering their probabilities.

Note that we didn't use the most-frequent-value imputation method since there's a good percentage of missing values in the 'Age' column and we'd like to preserve the randomness of the dataset as far as possible.



```
mode = train_data['Embarked'].value_counts().values[0]
train_data['Embarked'].fillna(mode, inplace=True)
```

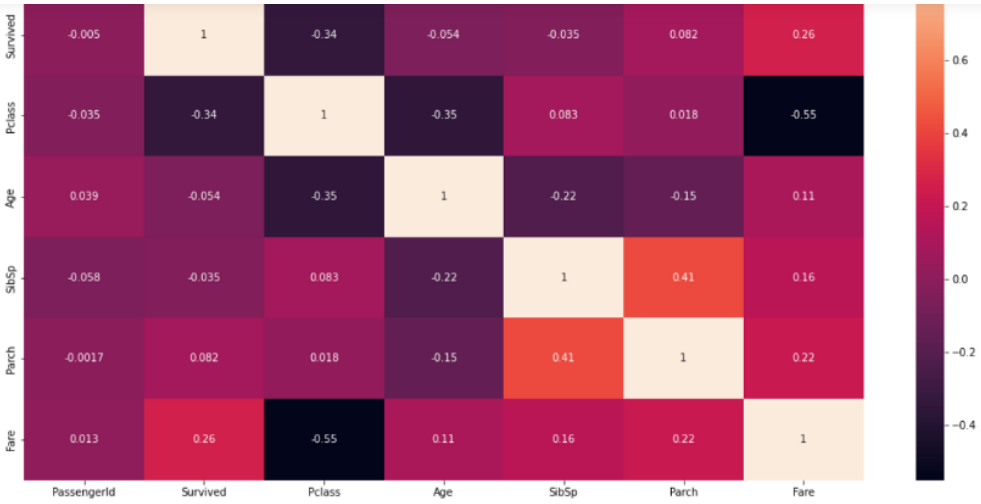
2.) What we do next is check all the unique categories of all the attributes to detect any unwanted datatypes. We do find some in the 'Embarked' column and we embark upon it's timely elimination (pardon the poor joke :D)

```
for i in train_data.columns[0:10]:
    print(i)
    print(train_data[i].unique())

train_data['Embarked'].replace(to_replace= 644, value=
train_data['Embarked'].value_counts().index.values[0], inplace=
True)
```

We now take a slight detour to understand the dataset before going deeper into feature engineering.

```
train_corr= train_data.corr()

plt.figure(figsize=(18,10))
sns.heatmap(train_corr, annot=True)
plt.show()
```





Whoa! Needs a lot of work, back to feature engineering guys :)

3.) Recategorization

a. Let's now use the 'Name' column to recategorise and extract Titles into another column

```
TITLES = ('Mrs', 'Mr', 'Master', 'Miss', 'Major', 'Rev',
          'Dr', 'Ms', 'Mlle','Col', 'Capt', 'Mme', 'Countess',
          'Don', 'Jonkheer')

def patterns(name: Text, titles: Set) -> Optional[Text]:
    for title in titles:
        if title in name:
            return title
    return "Untitled"

train_data['Title']= train_data['Name'].apply(lambda name:
patterns(name, TITLES))
```

Subsequently recategorise 'Titles' to three broad categories using the 'Sex' column as well

```
def squeeze_title( dataframe:pd.DataFrame )-> Text:
    title= dataframe['Title']
    if title in ('Don', 'Major', 'Capt', 'Jonkheer', 'Rev', 'Col'):
        return 'Mr'
```

```
            if dataframe['Sex'] == 'male':
                return 'Mr'
            else:
                return 'Mrs'
        else:
            return title

train_data['Title']= train_data.apply(squeeze_title, axis=1)
```

b. Recategorise the 'Cabin' column to broad categories as well

```
cabin_list = ['A', 'B', 'C', 'D', 'E', 'F', 'T', 'G', 'Unknown']

train_data['Deck'] = train_data['Cabin'].apply(lambda letter:
patterns(str(letter), cabin_list))
train_data.drop(columns='Cabin', inplace=True)
```

Now that we've completed recategorization, let's have a look at our dataset

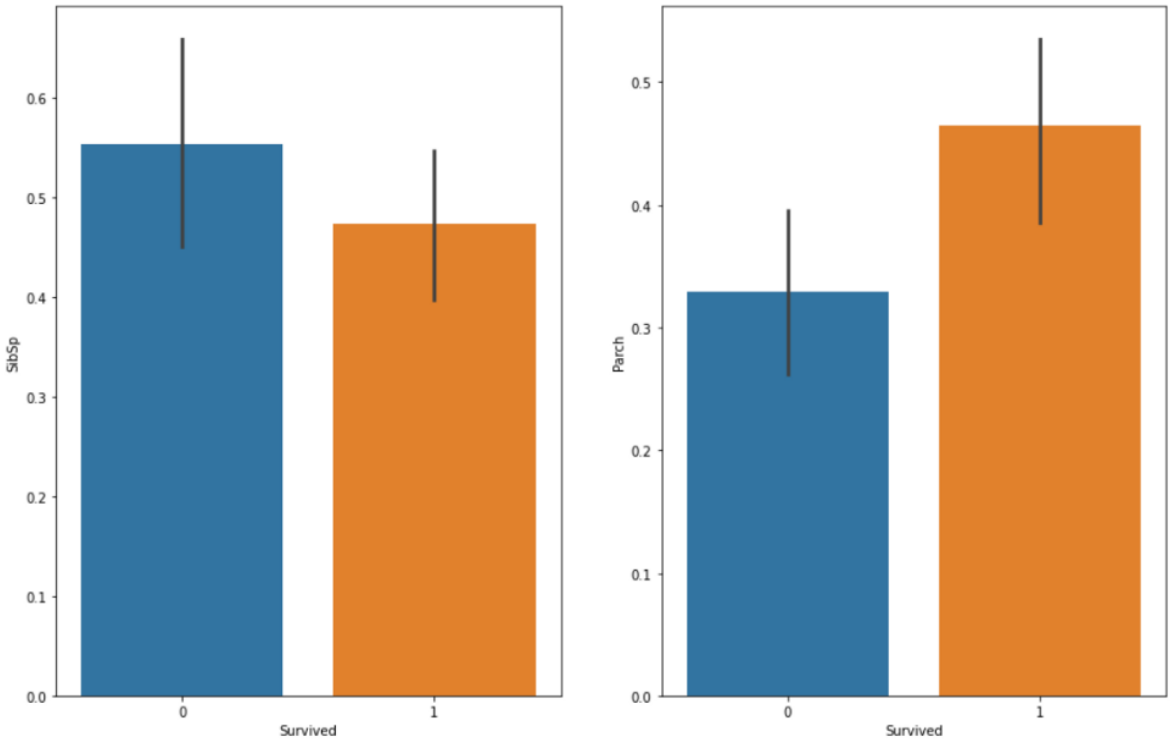
Before:

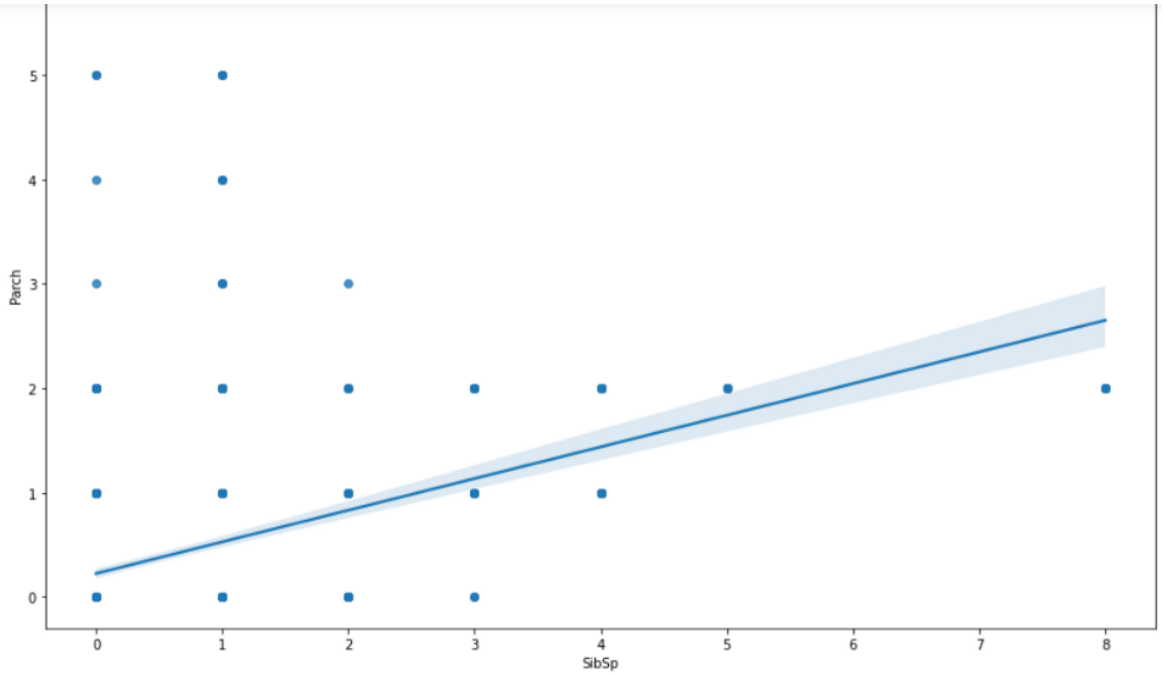
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

After:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Title	Deck
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	Mr	Untitled
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	Mrs	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	Miss	Untitled
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	Mrs	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	Mr	Untitled
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	S	Mr	Untitled
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	S	Miss	B
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	18.0	1	2	W./C. 6607	23.4500	S	Miss	Untitled
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C	Mr	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	Q	Mr	Untitled

```
fig= plt.figure(figsize=(15,21))
ss= fig.add_subplot(221)
pc= fig.add_subplot(222)
ss_pc= fig.add_subplot(2,2,(3,4))

sns.barplot(x='Survived', y='SibSp', data=train_data, ax= ss)
sns.barplot(x='Survived', y='Parch', data=train_data, ax= pc)
ss_pc.title.set_text("Correlation between Sibsp and Parch")
sns.regplot(x='SibSp', y='Parch', data=train_data, ax= ss_pc)

plt.show()
```





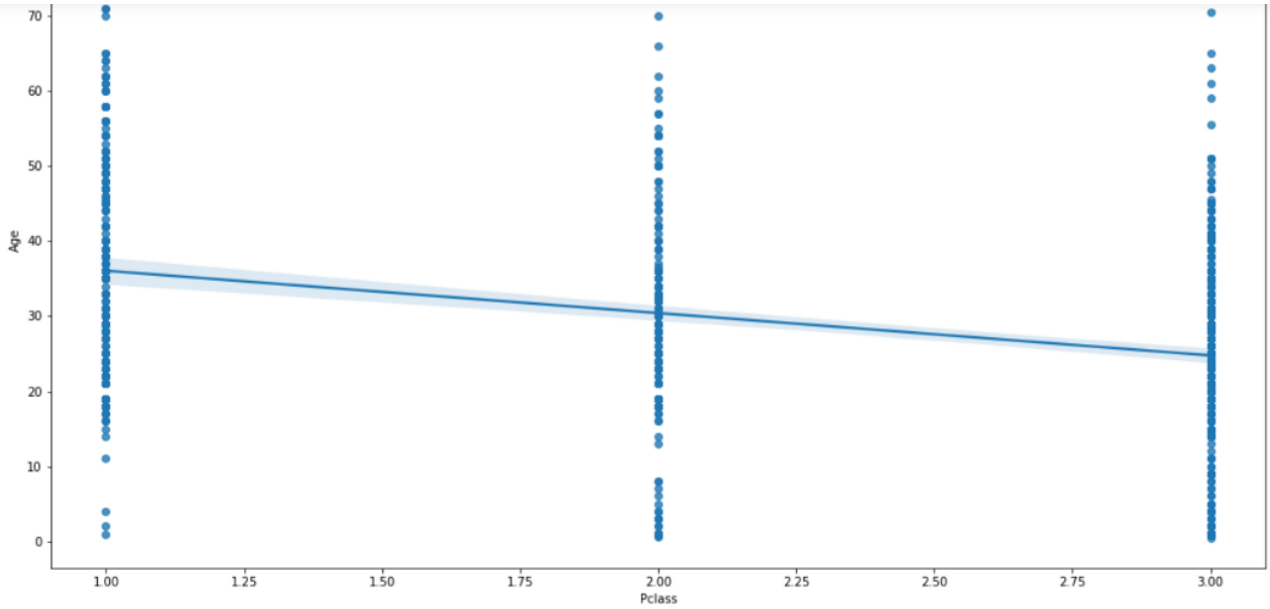
There's a positive correlation between Sibsp and Parch. That is, people with family relations most likely didn't survive themselves but they saved their families first. Even out of these, more so the ones with a sibling/ spouse rather than a kid/ parent.

Also, it'd be a logical idea to combine both columns since they represent the same idea

```
train_data['Family_size']= train_data['SibSp']+ train_data['Parch']
```

(ii) Between 'Age' and 'Pclass'

```
plt.figure(figsize=(18,10))
sns.regplot(x='Pclass', y='Age', data= train_data)
plt.show()
```



Weak correlation, hence not useful

(iii) Between 'Age' and 'Survival Rate'

```
tester= train_data.copy()
tester[['Survived','Died']]= pd.get_dummies(tester['Survived'])
new= tester[['Age','Survived','Died']].groupby('Age').sum()
new.reset_index(inplace=True)

#sorting values by two columns means that if the first column has
same values, then sort according to the second column
new.sort_values(by=['Died','Survived'], ascending=False,
inplace=True)
new
```

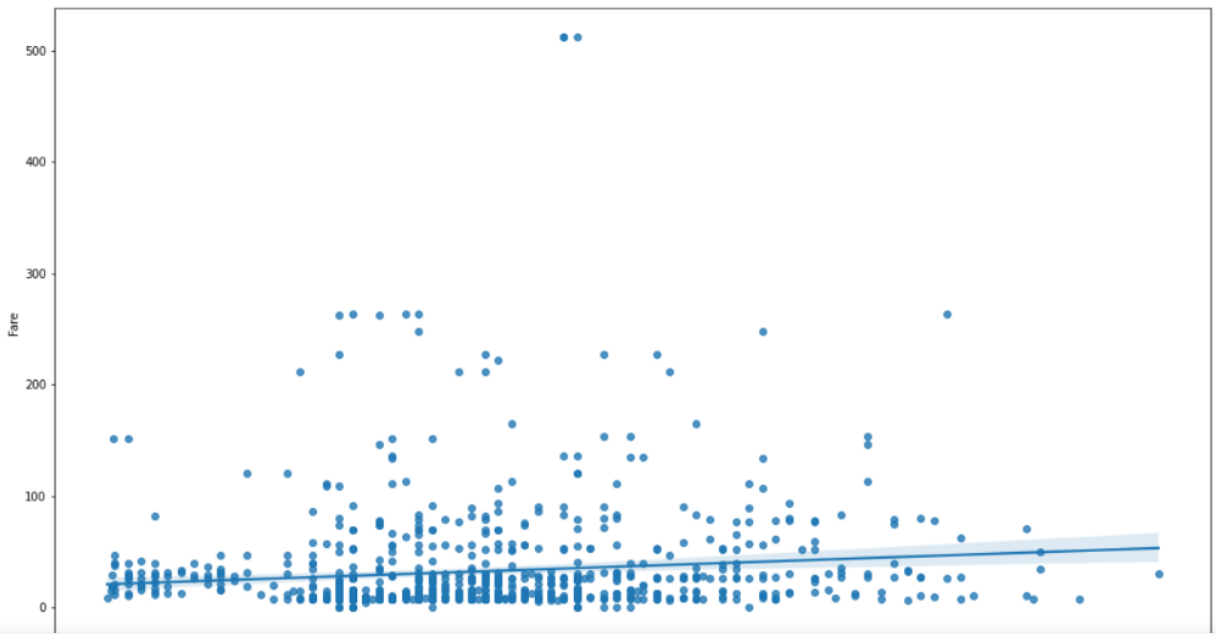
31	24.0	27	19
24	19.0	35	17
28	22.0	30	16
23	18.0	31	15
48	36.0	19	15
...	...	...	...
49	36.5	1	0
71	55.5	1	0
82	66.0	1	0
84	70.5	1	0
86	74.0	1	0

88 rows × 3 columns

Most young people of the age of 24 died

(iv) Between 'Age' and 'Fare'

```
plt.figure(figsize=(18,10))
sns.regplot(x='Age',y='Fare',data=train_data)
plt.show()
```



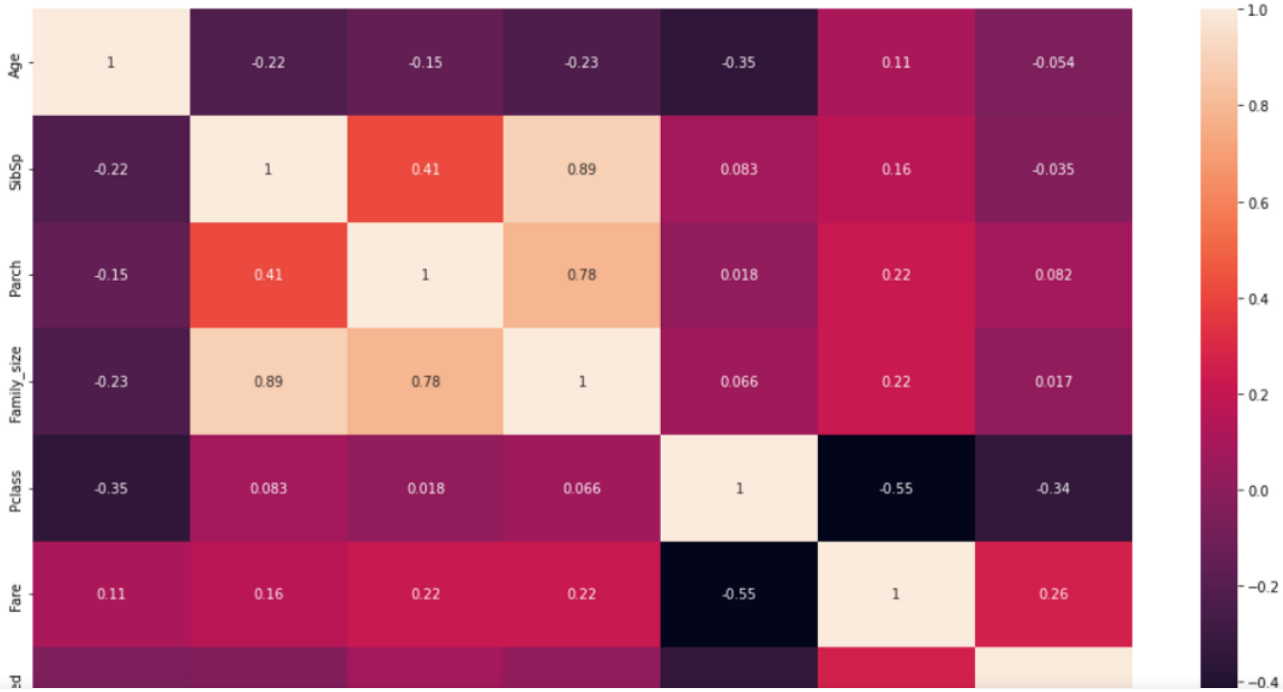
(v) Between 'Pclass' and 'Fare'

```
train_data[['Pclass','Fare']].loc[train_data['Fare']==0,:]
```

We find values of Pclass for which the fare becomes zero, hence we can't relate Pclass to Fare. Failed Hypothesis again :/

Let's now drop unwanted columns, check if the correlation map looks good and then proceed to the next steps after data engineering

```
train_data.drop(columns=['Name','Ticket','PassengerId'],
inplace=True)

COLUMNS = ['Title', 'Sex', 'Age', 'SibSp', 'Parch', 'Family_size',
'Pclass',
           'Fare', 'Deck', 'Embarked', 'Survived']
train_data = train_data[COLUMNS]

plt.figure(figsize=(18,10))
sns.heatmap(train_data.corr(), annot=True)
plt.show()
```



Step 2 : Train Test Split

Before moving into the splitting of the dataset, just keep 2 things in mind: One, don't forget to make sure all your categorical variables are encoded into numerics. Second, before splitting, specify your X and y

```
train_data = pd.get_dummies(train_data, columns =
['Title','Sex','Deck','Embarked'])

y_label= train_data['Survived']
X_features= train_data.iloc[:,:-1]

folds=4 #75, 25 split
# random state- shuffling applied to the data before the split
# stratify- When it splits the label, it will make sure that the
proportion of the training label is similar to the proportion of the
testing label, for good accuracy

X_train, X_test, Y_train, Y_test= train_test_split(X_features,
y_label, test_size=(1/folds), random_state= 42, stratify= y_label)
```

Step 3: Algorithm Setup

Before fitting the model, we want to make sure that our datasets have transformed & normalized, correctly. So, we're going to build a transformed pipeline to add our features into it.

We will build two pipelines- one is a transformers pipeline for scaling of the numerical columns and encoding of the categorical columns

Transformers Pipeline-

```
NUMERICAL_COLUMNS = ["Age", "SibSp", "Parch", "Family_size", "Fare"]
CATEGORICAL_COLUMNS = ["Title", "Sex", "Pclass", "Deck", "Embarked"]
```

```
CATEGORICAL_COLUMNS))
```

Models Pipeline-

First we'll build a custom estimator

```
class ClfSwitcher(BaseEstimator):

    def __init__(self,
                 estimator = DecisionTreeClassifier()):
        """
        Custom Estimator is a custom class that helps you to switch
        between classifiers.

        Args:
            estimator: sklearn object - classifier
        """
        self.estimator = estimator

    def fit(self, X, y=None, **kwargs):
        self.estimator.fit(X, y)
        return self
    def predict(self, X, y=None):
        return self.estimator.predict(X)
    def predict_proba(self, X):
        return self.estimator.predict_proba(X)
    def score(self, X, y):
        return self.estimator.score(X, y)
```

We finally compile the entire pipeline

```
pipeline = Pipeline([
    ('transformer', transformers),
    ('clf', ClfSwitcher())
])

parameters = [
    {
        'clf__estimator': [DecisionTreeClassifier()],
        'clf__estimator__criterion': ['gini', 'entropy'],
```

```
    },
    {
        'clf__estimator': [RandomForestClassifier()],
        'clf__estimator__n_estimators': [100, 250],
        'clf__estimator__criterion': ['gini', 'entropy'],
    },
    {
        'clf__estimator': [SVC()],
        'clf__estimator__kernel': ['rbf', 'sigmoid'],
        'clf__estimator__C': [1e-3, 1e-2, 1e-1, 1.0, 10., 1e+2,
1e+3],
        'clf__estimator__degree': [3, 4, 5, 6]
    },
    {
        'clf__estimator': [LogisticRegression()],
        'clf__estimator__penalty': ['l1', 'l2'],
        'clf__estimator__tol': [1e-4, 1e-3, 1e-2],
        'clf__estimator__C': [1e-3, 1e-2, 1e-1, 1.0, 10., 1e+2,
1e+3],
        'clf__estimator__solver': ['lbfgs', 'liblinear']
    }
]

cv = KFold(n_splits=(folds - 1))
```

Step 4: Model Fitting

We fine tune the model to gauge the best one, and the most optimal hyperparameters

```
gscv = GridSearchCV(pipeline,
                    parameters,
                    cv=cv,
                    scoring='r2',
                    n_jobs=12,
                    verbose=3)
gscv.fit(X_train, y_train)
gscv.best_params_
model = pipeline.set_params(**gscv.best_params_)
model.fit(X_train, y_train)
```



```
Predictions = model.predict(X_test)
```

Step 6: Model Evaluation

Let's evaluate our model using single and multi evaluation metrics

a) Single Evaluation Metric

```
model.score(X_test, y_test)

f1_score(y_test, Predictions)
```

b) Multi Evaluation Metric

The confusion matrix gives us a quantitative metric on the number of true/ false positives or negatives

```
cm= confusion_matrix(Y_test, Predictions, labels=[1,0])

confusion_plot = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=['Survived', 'Died'])  fig, ax =
plt.subplots(1,1,figsize=(12, 8)) ax.grid(False)
confusion_plot.plot(cmap='Blues', ax=ax) ax.set_title('Confusion
Matrix Between True Label & Predicted Label') plt.show()
```

The classification matrix helps us analyse the precision, recall and f1 score

```
classification_report(Y_test, Predictions)
```



0	1.00	1.00	1.00	137
1	1.00	1.00	1.00	86
accuracy			1.00	223
macro avg	1.00	1.00	1.00	223
weighted avg	1.00	1.00	1.00	223

Phew!

If you've made it till here, pat yourself on the back, you've done really good work. We've looked at data processing from the very basics of missing value imputation right till correlations between variables. Then, we moved onto the ML models and there again I stretched you a bit by not taking the beaten track of trying multiple models one by one and going for a pipeline instead. I'd be honest, keeping up with this must not have been exactly cakewalk and I sincerely hope you learnt something of value.

Until next time!