Models design

Abstract

The challenge requests to create Machine Learning model according to dataset provided (file dataset.csv). The best option is a Regression Analysis Model (Linear Regression, Polynomial Regression, Random Forests, etc.) due to values in a continuous scale.

Below the steps:

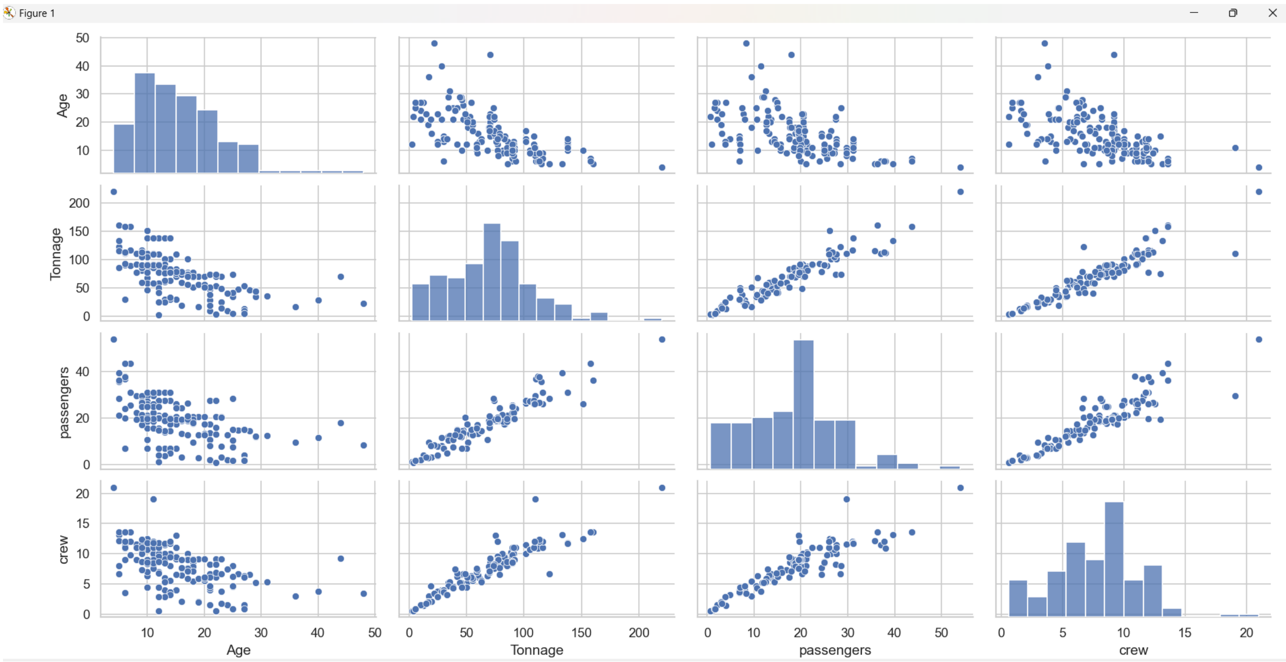
1. data analysis focusing on find features correlated to target variable
2. build the simplest model (es. Linear Regression) and evaluate it, if the result is not so good
3. build more complex model (es. Random Forests) and evaluate it

Data Analysis

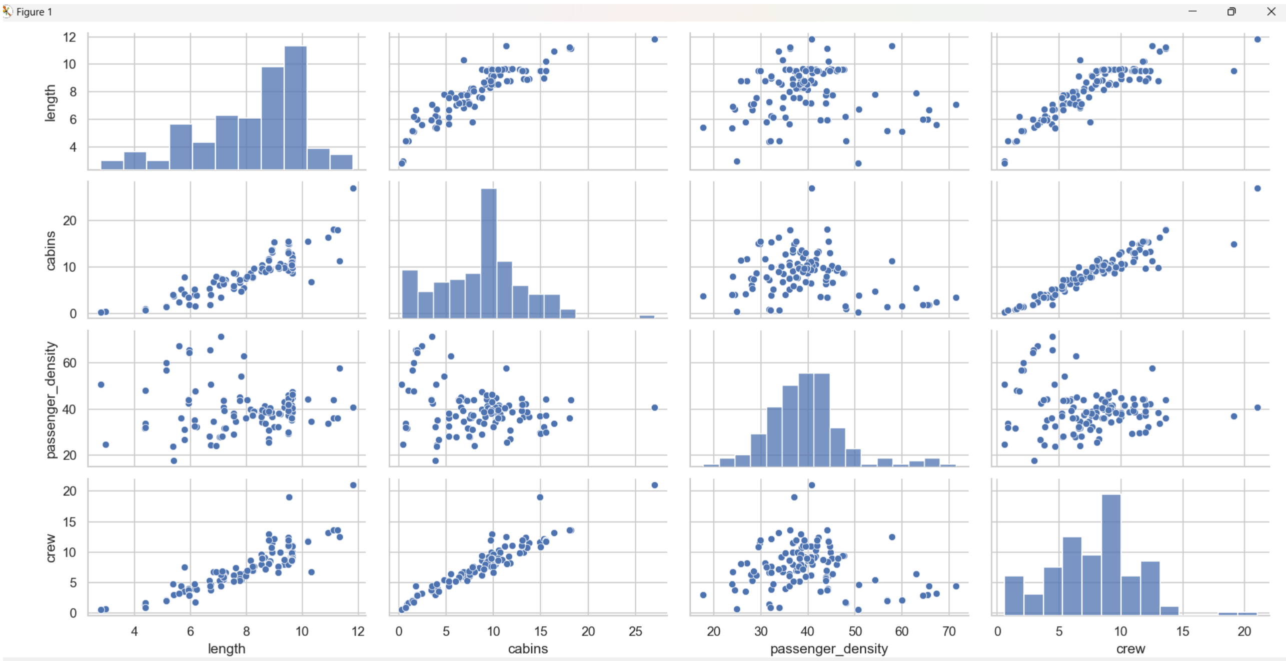
For this step we can use:

- scatterplot matrix, to select correlated variables
- covariance matrix, to evaluate the best correlated variable (for a first linear regression model)

Running *covariance_matrix.py* python code we can plot the two images below:

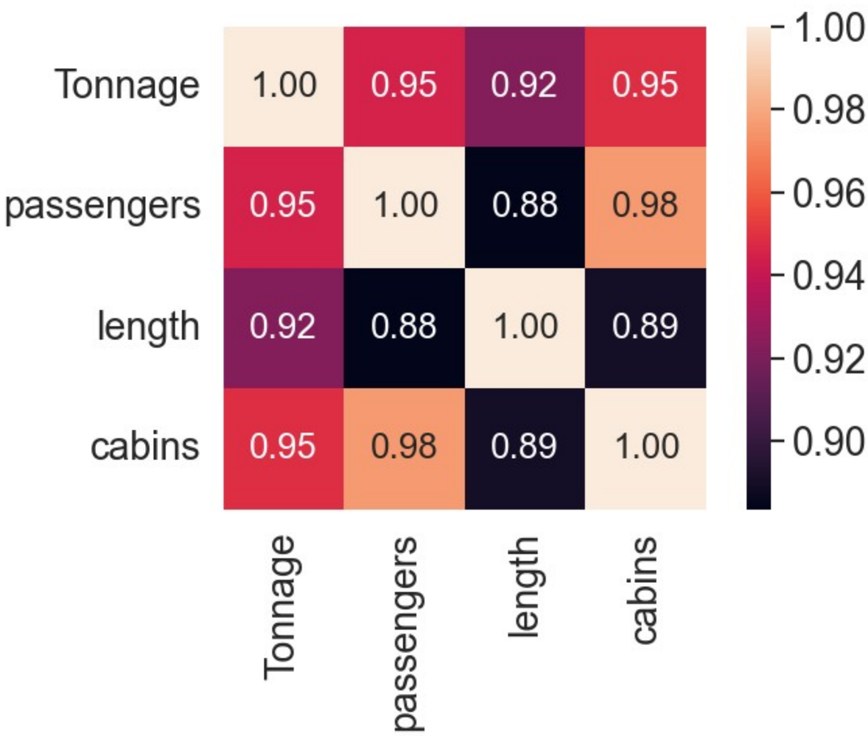


looking on the first image we see that we have to investigate “Tonnage” and “passengers” variables;



while looking on the second image we see that we have to investigate “cabins” and “length” variables too: the four variable seems to be correlated to crew (target variable).

Using covariance matrix:



we see that *passengers* variable is the candidate to evaluate a Linear Regression Model

Linear Regression Model

Linear Regression Model is the simplest one and it is very similar to Adaptive Linear Neuron Model. We can use the implementation in Scikit-Learn library which has an optimized implementation (file sklearn.linear.py):

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# get correlated columns
columns = ['Tonnage', 'passengers', 'length', 'cabins', 'crew']

df = pd.read_csv('./dataset.csv', sep=',', usecols=columns)

df.isnull().sum()

# remove rows that contain missing values
df = df.dropna(axis=0)
df.isnull().sum()

# dataset
X = df[columns]

target = 'crew'
features = df.columns[df.columns != target]

# input data
X = df[features].values
# output data
y = df[target].values

# spit dataset in train and test in order to evaluate the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)

# linear regression model
slr = LinearRegression()

# train the model
slr.fit(X_train, y_train)
y_train_pred = slr.predict(X_train)
y_test_pred = slr.predict(X_test)

# evaluate the model
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
print(f'MSE train: {mse_train:.2f}')
print(f'MSE test: {mse_test:.2f}')

r2_train = r2_score(y_train, y_train_pred)
r2_test =r2_score(y_test, y_test_pred)
print(f'R^2 train: {r2_train:.2f}')
print(f'R^2 test: {r2_test:.2f}')

# results: R2 train and test > 90%
```

- import library
- open file and read dataset
- remove missing values
- split dataset in train and test
- train the model
- evaluate the model

For evaluation we use two measures:

1. mean squared error
2. coefficient of determination

below the results:

```
MSE train: 1.04
MSE test: 0.68
R^2 train: 0.92
R^2 test: 0.93
```

0.93 for test dataset is already a good result.

Random Forsets

We investigated also Random Forests. This model is based on decision tree (random_forest.py):

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor


# get correlated columns
columns = ['Tonnage', 'passengers', 'length', 'cabins', 'crew']

df = pd.read_csv('./dataset.csv', sep=',', usecols=columns)

target = 'crew'
features = df.columns[df.columns != target]

X = df[features].values
y = df[target].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=123)
```

```python
forest = RandomForestRegressor(n_estimators=1000, criterion='squared_error', random_state=1, n_jobs=-1)
forest.fit(X_train, y_train)
y_train_pred = forest.predict(X_train)
y_test_pred = forest.predict(X_test)

# evaluate the model
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
print(f'MSE train: {mse_train:.2f}')
print(f'MSE test: {mse_test:.2f}')

r2_train = r2_score(y_train, y_train_pred)
r2_test =r2_score(y_test, y_test_pred)
print(f'R^2 train: {r2_train:.2f}')
print(f'R^2 test: {r2_test:.2f}')
```

below the resutls:

```
MSE train: 0.27
MSE test: 0.45
R^2 train: 0.98
R^2 test: 0.95
```

0.98 for train dataset and 0.95 for test dataset is the best results.