Table of Content:

0. Introduction

1. Business Problem

2. Importing libraries and loading the data

2.1 Importing and Interpreting the use of libraries

Pandas is flexible, fast and easy to use open source data analysis and manipulation tool.
Numpy is python library used to perform mathematical operations on array.
matplotlib and **seaborn** are python libraries used for data visualization. We can visualize data by charts and plots.
sklearn is used for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

In [1]:

```
import pandas as pd
import numpy as np

# Libraries for plotting and visualization
import matplotlib.pyplot as plt
import seaborn as sns

# libraries for modeling
from lightgbm import LGBMClassifier
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV,RandomizedS
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

import time
import warnings
warnings.filterwarnings(action='ignore')
```

2.2 Loading the data and looking at sample of 5 rows

Fields overview:

Static Informations:

Static Information is provided by the subject vessel's crew and is transmitted every 6 minutes regardless of the vessel's movement status

- 1 - mmsi : a unique identification number for each vessel station (the vessel's flag can also be deducted from it)
- 2 - shiptype : ship's type (tanker, cargo, fishing etc)
- 3 - width : width of vessel
- 4 - length : length of vessel
- 5 - draught : draught of vessel (the vertical distance between the waterline and the bottom of the hull (keel))

Dynamic Informations:

Dinamic Informations automatically transmitted every 2 to 10 seconds depending on the vessel's speed and course while underway and every 6 minutes while anchored from vessels

- 6 - navigationalstatus : instant situation (navigating, mooored etc)
- 7 - sog : speed of ground
- 8 - cog : course of ground (0 to 359 degrees)
- 9 - heading : fore side (heading and cog are not same because of the wind, current. But they are close to each other)

Data Source: https://www.kaggle.com/code/eminserkanerdonmez/ai-in-maritime-industsy/data

In [2]:
```
#loading the dataset
df = pd.read_csv('ais_data.csv').drop('Unnamed: 0', axis=1)

# check the size of the dataset using shape
print("size of the dataset:",df.shape)

print("\nLook at 1st 5 rows in the data:")
df.head()
```

```
size of the dataset: (358351, 9)

Look at 1st 5 rows in the data:
```

Out[2]:

	mmsi	navigationalstatus	sog	cog	heading	shiptype	width	length	draught
0	219019621	Unknown value	0.0	86.0	86.0	Fishing	4.0	9.0	NaN
1	265628170	Unknown value	0.0	334.5	NaN	Port tender	8.0	27.0	NaN
2	219005719	Unknown value	0.0	208.7	NaN	Fishing	4.0	11.0	NaN
3	219028066	Unknown value	0.0	NaN	NaN	Pleasure	3.0	12.0	NaN
4	212584000	Moored	0.0	153.0	106.0	Cargo	13.0	99.0	6.3

In [3]:
```
#Get information for the dataset such as datatype and non null counts
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 358351 entries, 0 to 358350
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   mmsi               358351 non-null  int64  
 1   navigationalstatus 358351 non-null  object 
 2   sog                357893 non-null  float64
 3   cog                355182 non-null  float64
 4   heading            337737 non-null  float64
 5   shiptype           358351 non-null  object 
 6   width              354640 non-null  float64
 7   length             354608 non-null  float64
 8   draught            332808 non-null  float64
dtypes: float64(6), int64(1), object(2)
memory usage: 24.6+ MB
```

Data has total 9 columns with 7 being numerical and categorical.

Numerical columns: mmsi, sog, cog, heading, width, length & draught

Categorical columns: navigationalstatus & shiptype

3. EDA - EXPLORATORY DATA ANALYSIS

3.1. Missing value information

In [4]:
```python
pd.options.display.float_format = "{:,.3f}".format

def missing_values_table(df):
    m=df.isnull().sum()
    print(pd.DataFrame({'n_miss' : m[m!=0],'% of total count' : m[m!=0]/len(df)}))

missing_values_table(df)
```

```
         n_miss  % of total count
sog         458             0.001
cog        3169             0.009
heading   20614             0.058
width      3711             0.010
length     3743             0.010
draught   25543             0.071
```

Column **draught** has ~7.1% and **heading** has ~5.7% missing values.

width and **length** each has around 1% missing values.

3.2. Numerical columns

1. Summary statistics for Numerical columns

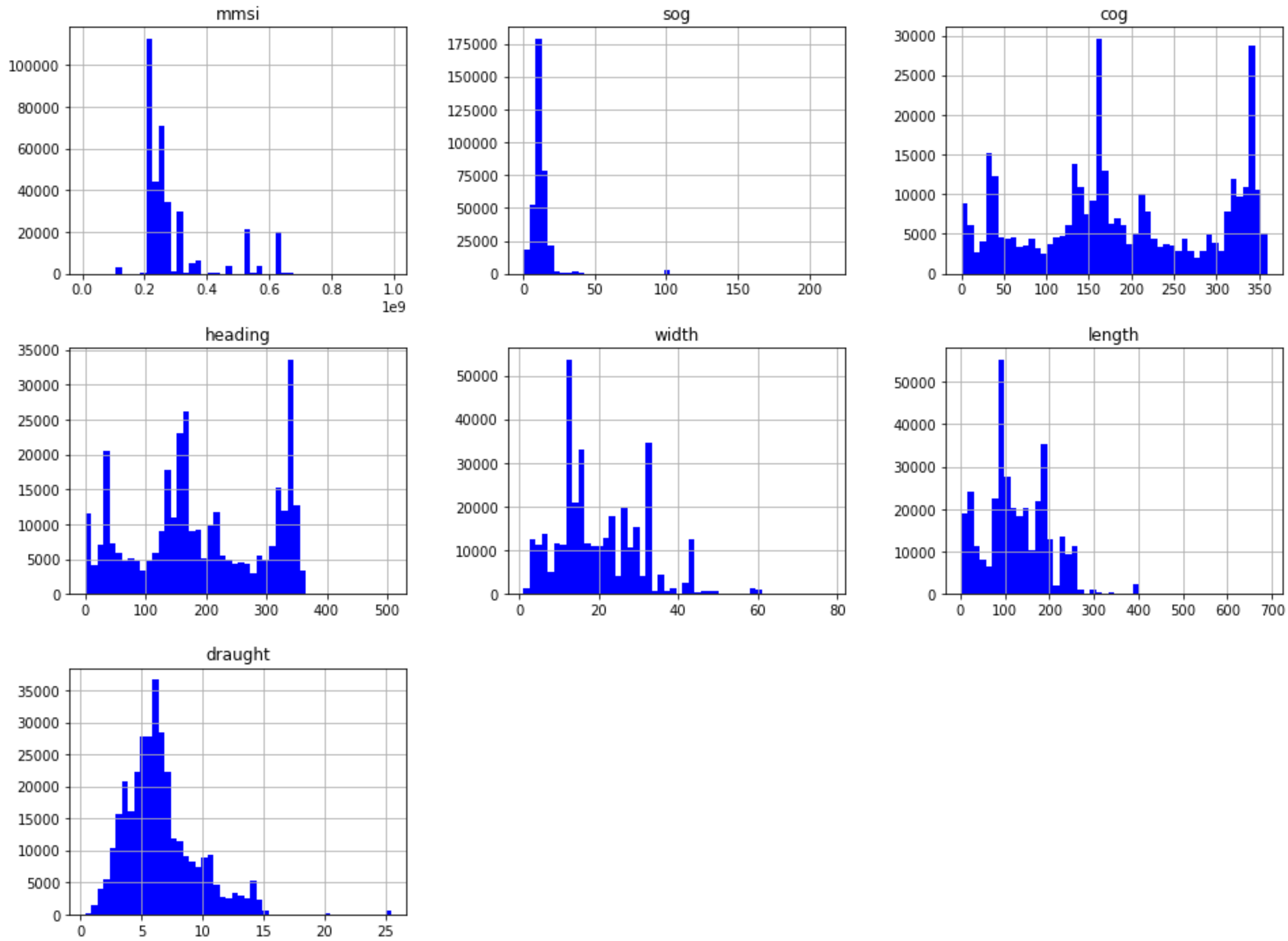
In [5]:
```python
pd.options.display.float_format = "{:,.2f}".format
num_cols = ['mmsi','sog','cog','heading','width','length','draught']

df[num_cols].describe()
```

Out[5]:

	mmsi	sog	cog	heading	width	length	draught
count	358,351.00	357,893.00	355,182.00	337,737.00	354,640.00	354,608.00	332,808.00
mean	293,967,827.62	12.12	189.06	190.08	19.95	124.97	6.57
std	121,386,631.12	9.36	107.59	107.11	10.81	71.27	2.93
min	9,112,856.00	0.00	0.00	0.00	1.00	2.00	0.40
25%	219,578,000.00	9.20	116.30	120.00	12.00	83.00	4.60
50%	248,659,000.00	11.30	168.70	170.00	17.00	115.00	6.10
75%	304,665,000.00	13.30	300.18	303.00	28.00	181.00	7.90
max	992,195,011.00	214.00	359.90	507.00	78.00	690.00	25.50

2. Distribution of values for numerical columns

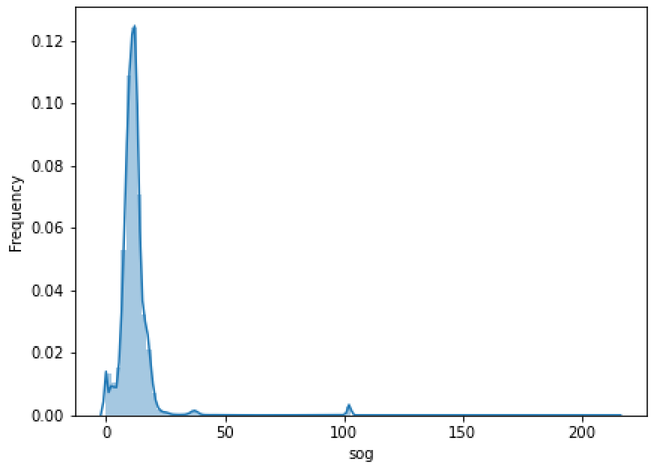
In [6]:
```
dummy = df[num_cols].hist(bins=50, figsize=(16,12), color = "blue")
```

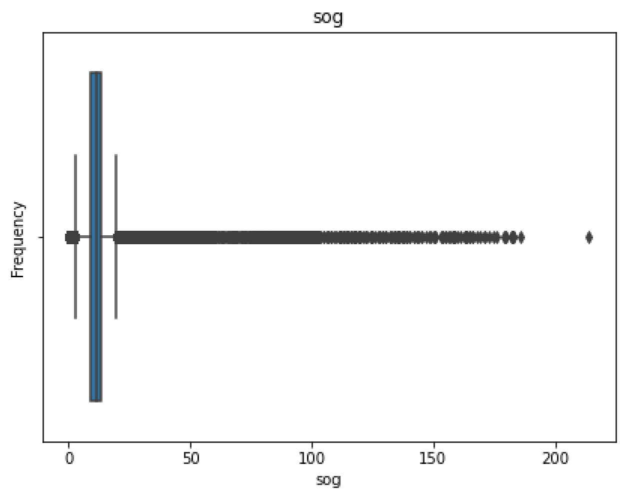


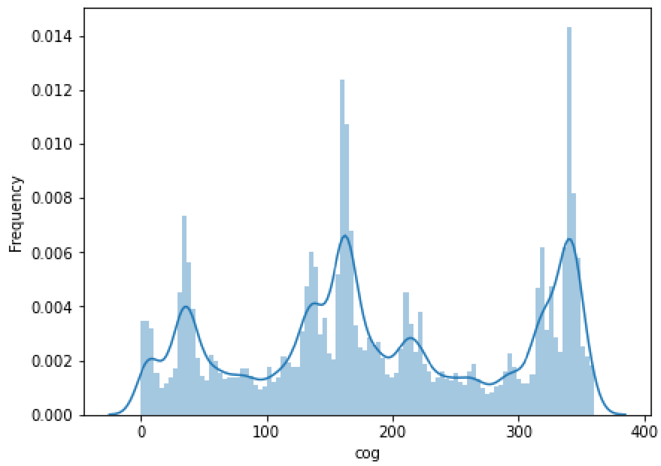
3. Frequency distribution and Box-Plot

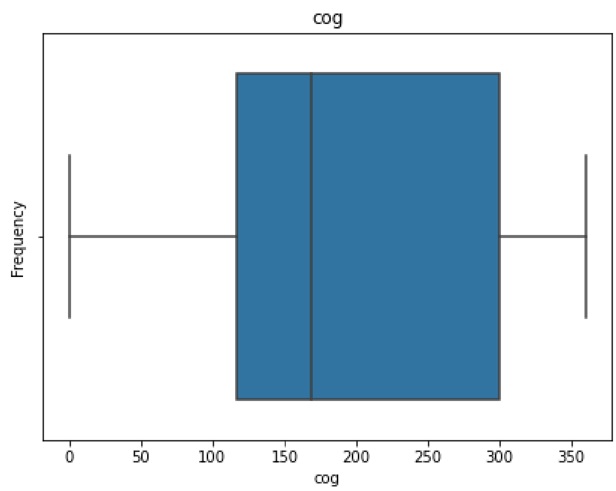
In [7]:
```
col_list = ['sog', 'cog', 'heading','width', 'length', 'draught']

for col in col_list:
    fig = plt.figure(figsize = (15,5))
    #Histogram
    plt.subplot(1,2,1)
    #Define plot object
    hist = sns.distplot(df.loc[:,col].astype(float), bins = 100)
    #Setting graph title
    #hist.set_title(col)
    hist.set(xlabel = col, ylabel = 'Frequency')
    #Boxplot
```

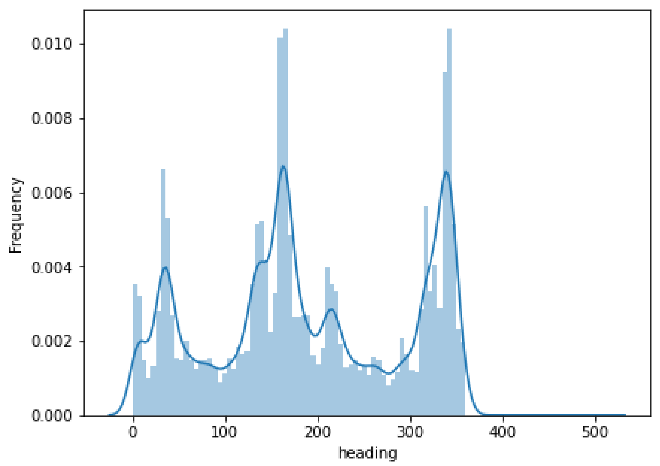
```
    plt.subplot(1,2,2)
    #Define plot object
    box = sns.boxplot(df.loc[:,col].astype(float))
    #Setting graph title
    box.set_title(col)
    box.set(xlabel = col, ylabel = 'Frequency')
    #Showing the plot
    plt.show()
```

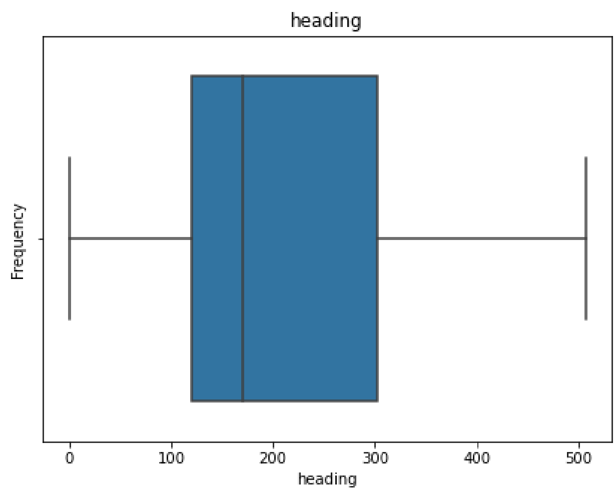


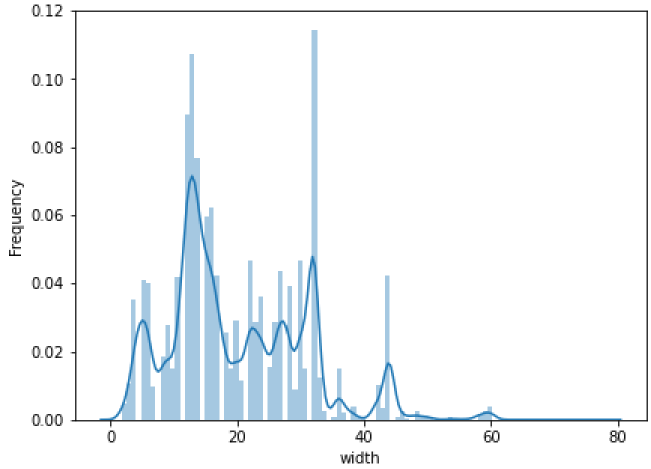


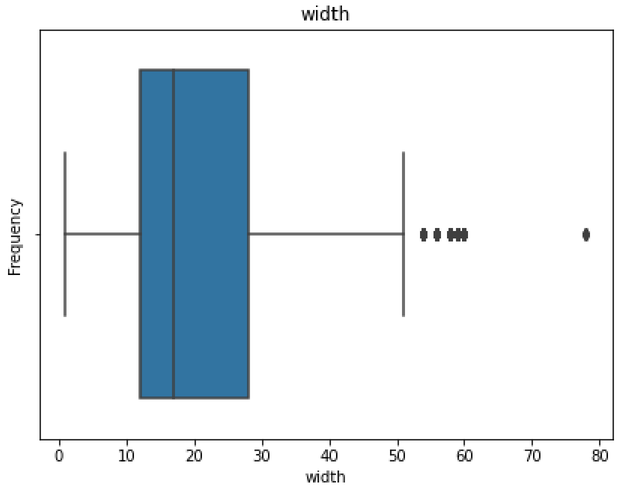


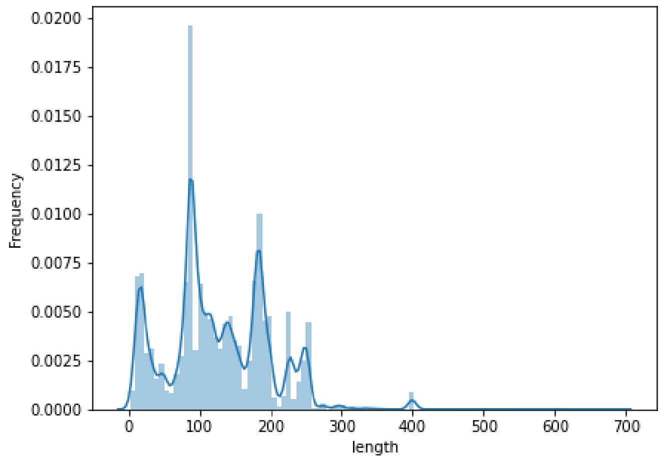


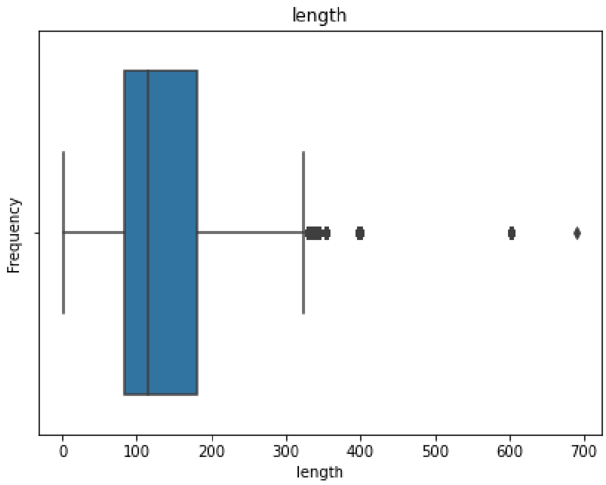


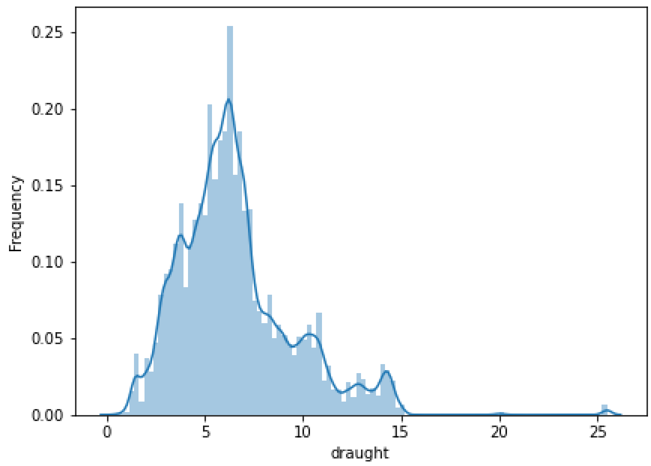


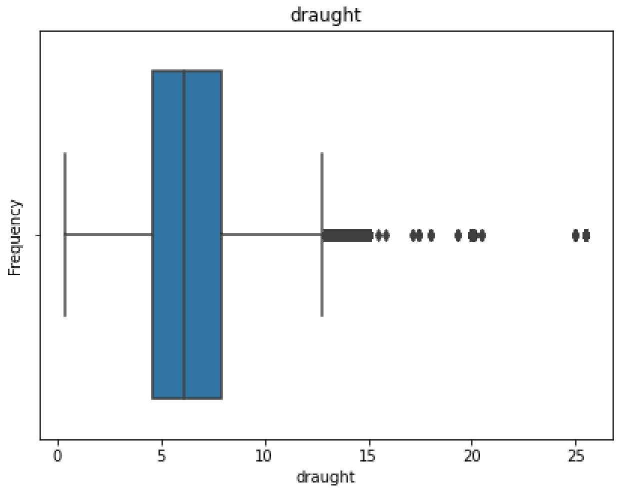












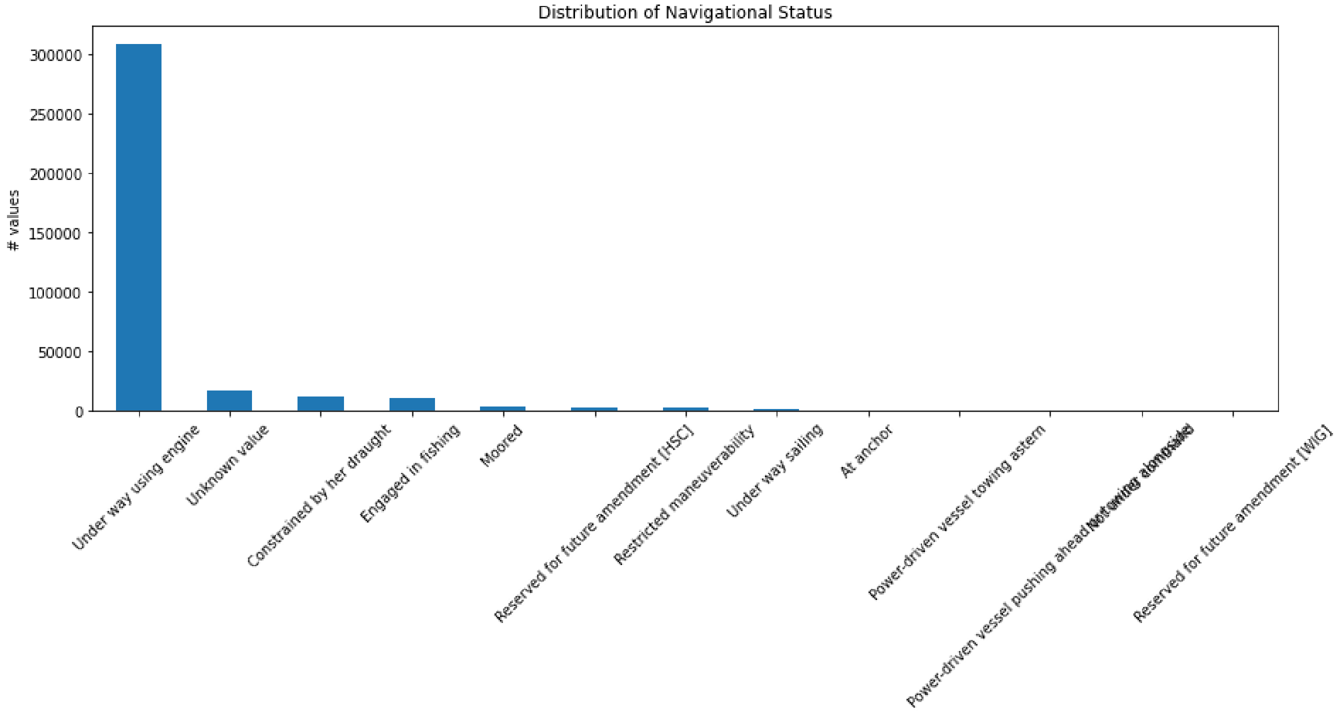
As per the box-plot method, there are a bunch of outliers in "sog" and few in "width", "length" and "draught".
"cog" and "heading" does not seem to have any outliers.

3.3. Distribution of values for categorical columns

In [8]:

```
# Distribution of column "navigationalstatus"

print('% Distribution of Navigational Status:\n')
print(df['navigationalstatus'].value_counts(1))

print('\n\n')
plt.figure(figsize=(15,5))
df['navigationalstatus'].value_counts().plot.bar()
plt.title('Distribution of Navigational Status')
plt.ylabel('# values')
plt.xticks(rotation = 45)
plt.show()
```

```
% Distribution of Navigational Status:

Under way using engine                              0.86
Unknown value                                       0.05
Constrained by her draught                          0.03
Engaged in fishing                                  0.03
Moored                                              0.01
Reserved for future amendment [HSC]                 0.01
Restricted maneuverability                          0.01
Under way sailing                                   0.00
At anchor                                           0.00
Power-driven vessel towing astern                   0.00
Power-driven vessel pushing ahead or towing alongside   0.00
Not under command                                   0.00
Reserved for future amendment [WIG]                 0.00
Name: navigationalstatus, dtype: float64
```

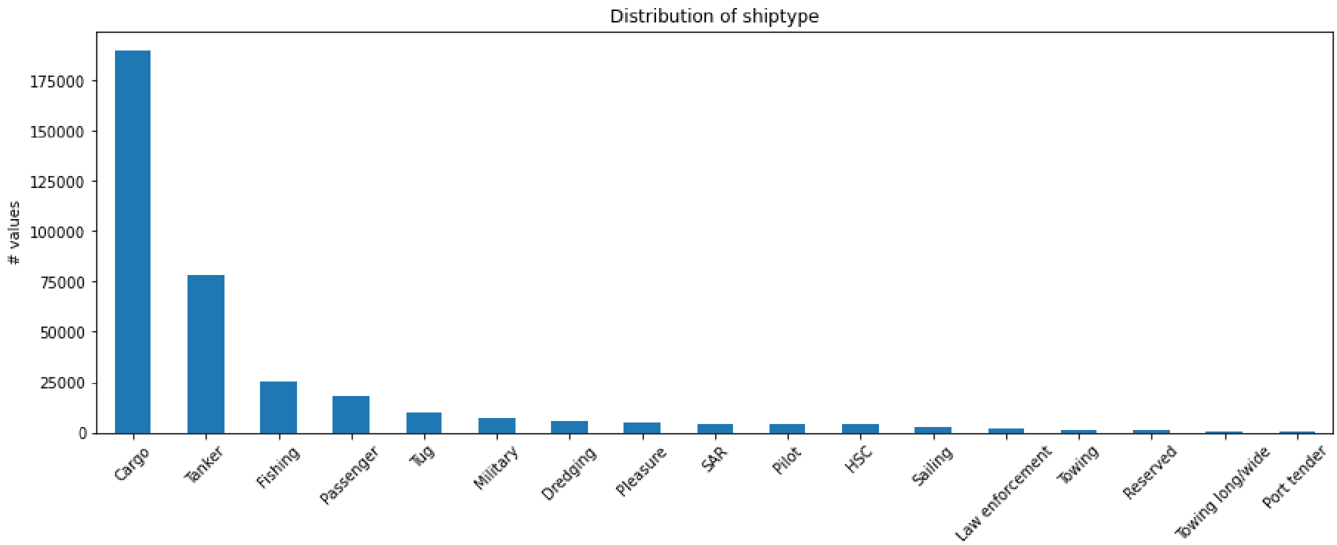


The value **Under way using engine** contributes to 86% of the total data in column "navigationalstatus". While **Unknown values** makes up 5%.

In [9]:

```
# Distribution of column "shiptype"

print('% Distribution of Ship Type:\n')
print(df['shiptype'].value_counts(1))

print('\n\n')
plt.figure(figsize=(15,5))
df['shiptype'].value_counts().plot.bar()
plt.title('Distribution of shiptype')
plt.ylabel('# values')
plt.xticks(rotation = 45)
plt.show()
```

```
% Distribution of Ship Type:

Cargo              0.53
Tanker             0.22
Fishing            0.07
Passenger          0.05
Tug                0.03
Military           0.02
Dredging           0.02
Pleasure           0.01
SAR                0.01
Pilot              0.01
HSC                0.01
Sailing            0.01
Law enforcement    0.00
Towing             0.00
Reserved           0.00
Towing long/wide   0.00
Port tender        0.00
Name: shiptype, dtype: float64
```



Cargo contributes to 53% of the total data in column "shiptype". **Tanker** makes up 2%. **Fishing** 7% and **Passenger** 3%

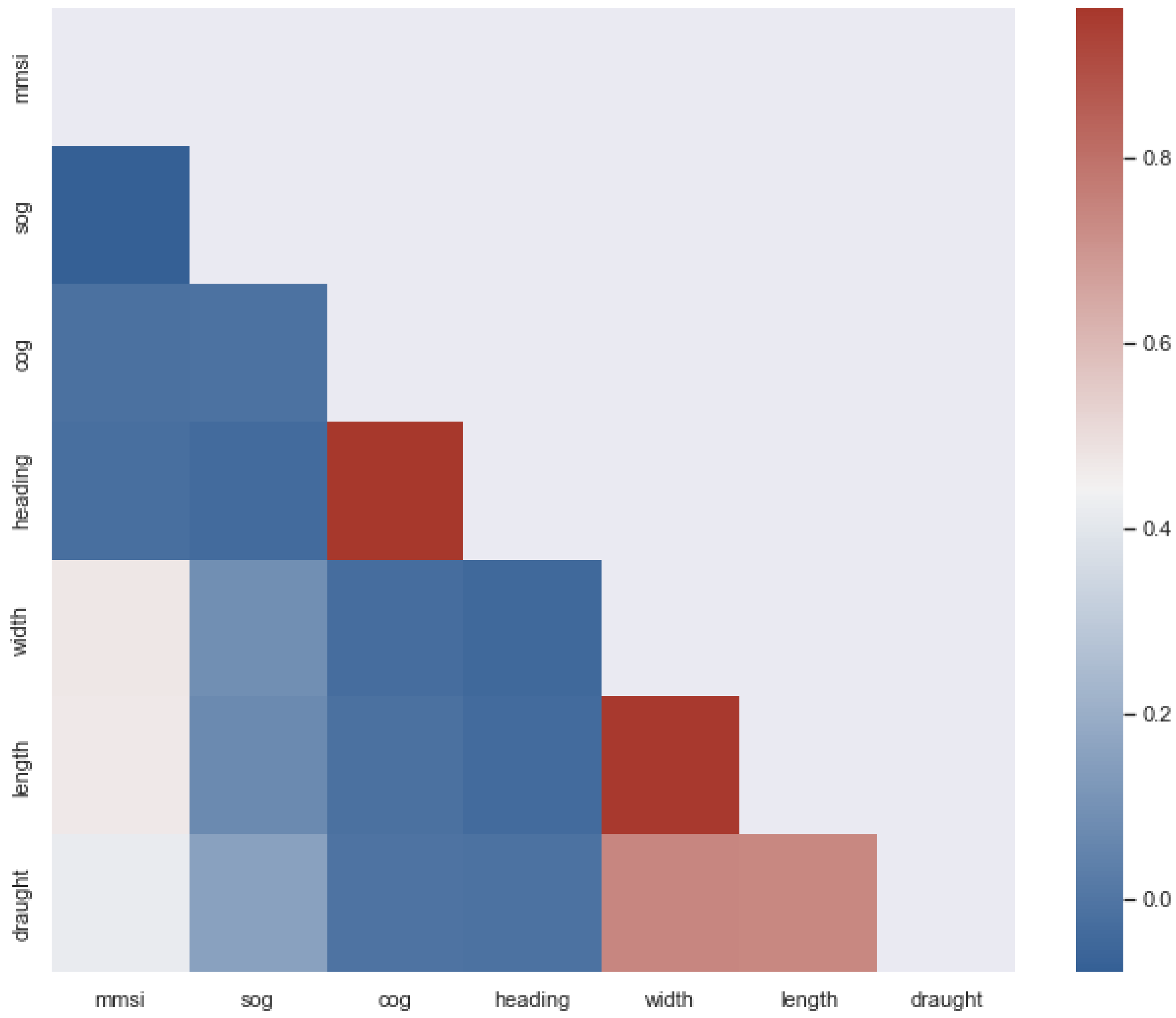
3.4. Outlier detection using Box-Plot method

In [10]:
```
def thresholds(col, data, d, u):
    q3=data[col].quantile(u)
    q1=data[col].quantile(d)
    down=q1-(q3-q1)*1.5
    up=q1+(q3-q1)*1.5
    return down, up

def check_outliers(col, data, d=0.25, u=0.75, plot=False):
    down, up = thresholds(col, data, d, u)
    ind = data[(data[col] < down) | (data[col] > up)].index
    if plot:
        sns.boxplot(x=col, data=data)
        plt.show()
    if len(ind)!= 0:
        print(f"\n Number of outliers for '{col}' : {len(ind)}")
        return col

for col in num_cols:
    check_outliers(col, df, 0.01, 0.99) # we set thresholds at 0.01 and 0.99
```

```
Number of outliers for 'mmsi' : 24

Number of outliers for 'sog' : 2910

Number of outliers for 'width' : 9

Number of outliers for 'length' : 47

Number of outliers for 'draught' : 567
```

3.5. Correlation Matrix between numerical columns

In [11]:
```
# correlation matrix dataframe
df_corr = df[num_cols].corr()

### Heat map of correlation matrix
sns.set_theme(style="darkgrid")

mask = np.triu(np.ones_like(df_corr))
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(250, 15, s=75, l=40, center="light", as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(df_corr, mask=mask, cmap=cmap, square=True)
plt.show()
```



heading is highly related to **cog** (96% correlation).
Similary, **width** also has a 96% correlation with **length**
draught has 74% correlation with both **lenght** and **width**

3.6 Average length and width by shiptype

In [12]:
```
df[['mmsi','shiptype','length','width']].drop_duplicates(subset='mmsi')\
                                        .groupby(['shiptype'])\
                                        .agg({'width':'mean','length':'mean'})\
                                        .reset_index()\
                                        .sort_values('width')
```

Out[12]:

	shiptype	width	length
11	SAR	2.88	9.20
8	Pleasure	3.75	11.30
12	Sailing	4.06	13.66
9	Port tender	4.78	14.00
7	Pilot	5.10	16.79
4	Law enforcement	5.94	24.22
2	Fishing	6.03	22.37
14	Towing	7.45	22.82
15	Towing long/wide	9.00	28.33
5	Military	9.29	50.75
16	Tug	10.58	35.23
10	Reserved	11.92	44.08
1	Dredging	12.38	61.28
3	HSC	13.20	45.68
6	Passenger	16.75	90.51
0	Cargo	21.57	144.02
13	Tanker	28.75	174.91

There seems to have a clear distinction between dimensions of the ship (length & widht) for different ship types.
These might be strong predictors in the model for "shiptype".

4. FEATURE ENGINEERING

4.1. Creating new features

1. ‘cog’ and ‘heading’ variables are very similar. And they also have a high correlation of 96% between them. So we can combine these two as one.
2. Dividing the 360-degree route into 8 regions.
3. the ships with less than 5.5kts speed and no route information were tagged as 'FIX'.
4. Vessels’ speed depends on ship type mostly. We fill the missings according to 'sog' and 'route' variables. Then assigned as a new variable "speed"
5. new variables dimension = widht*length

In [13]:
```
# First, the filling was made according to those in the 'heading' but not in the 'cog'.
df['cog'] = np.where(df['cog'].isnull(), df['heading'], df['cog'])

# Secondly, we divided the 360-degree route into 8 regions.
rot= [-1, 45, 90, 135, 180, 225, 270, 315, 360]
```

```
df['waypoint'] = pd.cut(df['cog'], rot, labels=['NNE','ENE','ESE','SSE','SSW','WSW','WNW','NN

# Finally, the ships with less than 5.5kts speed and no route information were tagged as 'FIX
df['waypoint'] = np.where((df['sog']<5.5) & (df['waypoint'].isnull()), 'FIX', df['waypoint'])

# Filling the missings according to 'sog' and 'cog' variables. Then assigned as a new variabl
df['speed'] = df["sog"].fillna(df.groupby(['shiptype', 'waypoint'])['sog'].transform('mean'))

# dimension = lenght*width
df['dimension'] = df['width'] * df['length']
```

4.2. Dropping features not required for modeling and deduping

In [14]:
```
# columns to drop
drop_cols = ['mmsi','heading']

# new df_m for modeling data only
df_m = df.drop(drop_cols, axis=1).drop_duplicates()

print('shape modeling data after dropping columns and deduping', df_m.shape)
```

```
shape modeling data after dropping columns and deduping (348379, 10)
```

4.3. Filling missing values with median for numerical features and mode for categorical

In [15]:
```
# filling waypoint with mode
df_m['waypoint'] = df_m['waypoint'].fillna(df_m['waypoint'].mode()[0])

# filling numerical with median
df_m = df_m.fillna(df_m.median(numeric_only=True))
```

5. OHE - one-hot encoding for categorical features

Only "navigationalstatus" and "waypoint" will be encoded and shiptype since shiptype is the model target

In [16]:
```
# One hot encoder function;
def one_hot_encoder(df, cat_cols, drop_first=True):
    dataframe = pd.get_dummies(df, columns=cat_cols, drop_first=drop_first)
    return dataframe
```

In [17]:
```
df_m = one_hot_encoder(df_m, cat_cols=['waypoint','navigationalstatus'], drop_first=True)
df_m
```

Out[17]:

	sog	cog	shiptype	width	length	draught	speed	dimension	waypoint_ESE	waypoint_FIX	...
0	0.00	86.00	Fishing	4.00	9.00	6.10	0.00	36.00	0	0	...
1	0.00	334.50	Port tender	8.00	27.00	6.10	0.00	216.00	0	0	...
2	0.00	208.70	Fishing	4.00	11.00	6.10	0.00	44.00	0	0	...
3	0.00	169.00	Pleasure	3.00	12.00	6.10	0.00	36.00	0	1	...
4	0.00	153.00	Cargo	13.00	99.00	6.30	0.00	1,287.00	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
358346	11.00	171.90	Cargo	12.00	82.00	4.20	11.00	984.00	0	0	...
358347	16.60	341.60	Cargo	27.00	170.00	8.90	16.60	4,590.00	0	0	...
358348	20.60	340.70	Passenger	36.00	224.00	6.90	20.60	8,064.00	0	0	...
358349	34.90	96.20	Pilot	3.00	7.00	6.10	34.90	21.00	1	0	...
358350	11.50	315.00	Fishing	8.00	32.00	6.00	11.50	256.00	0	0	...

348379 rows × 28 columns

In [18]:
```
df_m.rename(columns={"navigationalstatus_Reserved for future amendment [HSC]":"navigationalst
df_m.rename(columns={"navigationalstatus_Reserved for future amendment [WIG]":"navigationalst
```

In [ ]:

6. Data-Splitting

Splitting the data into train and test for modelling

Train 80% and Test 20%

6.1 Feature and Target split

In [19]:
```
target = 'shiptype'

X = df_m.drop(target, axis=1)
y = df_m[target]
```

6.2 Train-Test split

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=7)
```

6.3 Scaling the features - Robust Scaler

'RobustScaler' is used because it is robust to outliers

In [21]:
```
# RobustScaler object - fitting on train
scaler = RobustScaler()
scaler.fit(X_train)
```

```
# transforming X_train
X_train = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns)

# transforming X_test
X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
```

In [ ]:

7. Modeling and evaluation

1) First, a baseline model will be built where the prediction will be the most common class in the target.
2) Then some multi-class classifier models will be built with default parameters and without any hyper-parameter tuning.
3) Best performing model from step 2 will be picked and further fine-tuned.

In [22]:

```
## Function to plot feature importance
def show_feat_imp(model, X, n_feats=15):
    """
    model: model object
    X: feature dataset on which prediction is to be done. Ex: X_test or X_train
    n_feats: number of top features to display
    """
    feat_imp = pd.Series(model.feature_importances_, index = X.columns).sort_values(ascending
    display(feat_imp.head(n_feats))
    feat_imp.head(n_feats).plot.bar(x='features', y='feat_imp', figsize=(15,8), align="center
```

7.1 Baseline Model

Here, we use a majority class classifier as a baseline i.e we find the most common class amongst the data set where the output is always a prediction

In [23]:

```
# majority class - mode of shiptype
y_mode = y_train.mode()[0]
print('majority category for shiptype:', y_mode)
```

```
majority category for shiptype: Cargo
```

The baseline method will present "Cargo" as output for all predictions. We can use macro-averaging in this project (precision, recall and F-score are evaluated in each class separately and then averaged across classes).

So if we apply the baseline classifier to all of the training set for the "Cargo" label, the accuracy measures will be:

In [24]:

```
# prediction using mode
prediction = np.repeat(y_mode, y_test.shape[0])
```

In [25]:

```
# performance of baseline model on test
print(classification_report(y_test,prediction))
```

```
                  precision    recall  f1-score   support

           Cargo       0.54      1.00      0.70     37524
        Dredging       0.00      0.00      0.00      1062
         Fishing       0.00      0.00      0.00      4986
             HSC       0.00      0.00      0.00       709
 Law enforcement       0.00      0.00      0.00       320
        Military       0.00      0.00      0.00      1416
       Passenger       0.00      0.00      0.00      3298
           Pilot       0.00      0.00      0.00       789
        Pleasure       0.00      0.00      0.00       625
     Port tender       0.00      0.00      0.00        57
        Reserved       0.00      0.00      0.00       144
             SAR       0.00      0.00      0.00       709
         Sailing       0.00      0.00      0.00       370
          Tanker       0.00      0.00      0.00     15452
          Towing       0.00      0.00      0.00       208
Towing long/wide       0.00      0.00      0.00       122
             Tug       0.00      0.00      0.00      1885

        accuracy                           0.54     69676
       macro avg       0.03      0.06      0.04     69676
    weighted avg       0.29      0.54      0.38     69676
```

The accuracy of baseline model is at 54%. It can definitely be improved.

7.2 Decision Tree Classifier

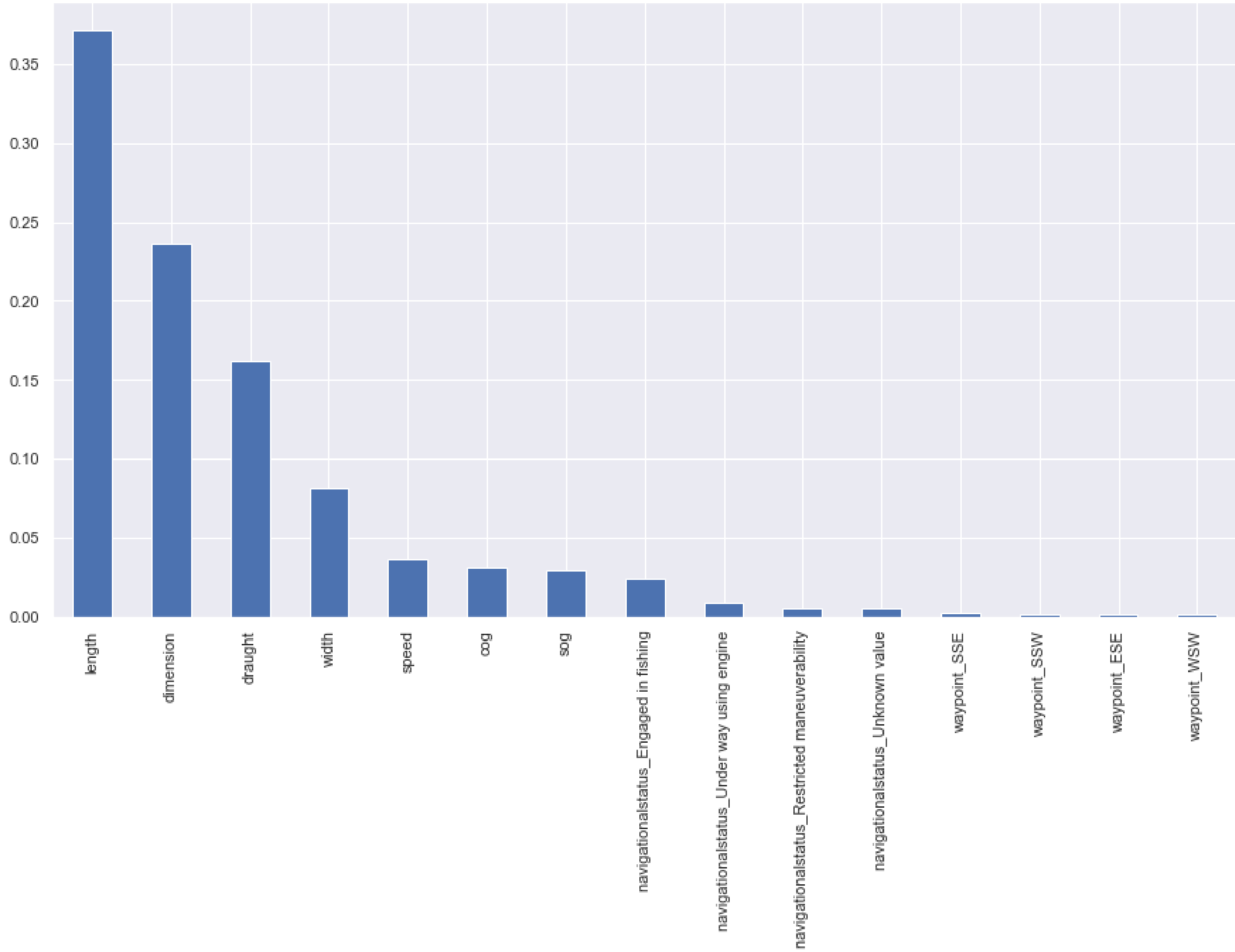
In [26]:
```
# decision tree classifier object
dt = DecisionTreeClassifier(random_state=7)
dt.fit(X_train, y_train)

# Prediction on test
dt_pred = dt.predict(X_test)

# performance of decision tree classifier on test
print(classification_report(y_test,dt_pred))
```

```
                  precision    recall  f1-score   support

           Cargo       0.99      0.99      0.99     37524
        Dredging       0.97      0.97      0.97      1062
         Fishing       0.96      0.96      0.96      4986
             HSC       0.99      0.98      0.99       709
 Law enforcement       0.98      0.98      0.98       320
        Military       0.99      0.99      0.99      1416
       Passenger       0.99      0.99      0.99      3298
           Pilot       0.96      0.98      0.97       789
        Pleasure       0.65      0.62      0.64       625
     Port tender       0.66      0.72      0.69        57
        Reserved       0.92      0.90      0.91       144
             SAR       0.89      0.90      0.89       709
         Sailing       0.59      0.61      0.60       370
          Tanker       0.98      0.98      0.98     15452
          Towing       0.90      0.91      0.91       208
Towing long/wide       0.96      0.95      0.95       122
             Tug       0.99      0.99      0.99      1885

        accuracy                           0.98     69676
       macro avg       0.90      0.91      0.91     69676
    weighted avg       0.98      0.98      0.98     69676
```

Accuracy of the model is **98%** which is quite good for a simple decision tree.

In [27]:
```python
# feature importance
show_feat_imp(model=dt, X=X_test)
```

```
length                                       0.37
dimension                                    0.24
draught                                      0.16
width                                        0.08
speed                                        0.04
cog                                          0.03
sog                                          0.03
navigationalstatus_Engaged in fishing        0.02
navigationalstatus_Under way using engine    0.01
navigationalstatus_Restricted maneuverability  0.00
navigationalstatus_Unknown value             0.00
waypoint_SSE                                 0.00
waypoint_SSW                                 0.00
waypoint_ESE                                 0.00
waypoint_WSW                                 0.00
dtype: float64
```



Top features are "length" and "dimension" contributing almost 60%. This indicates that different ship types have significantly varying sizes/dimensions.

7.3 Random Forest Classifier

In [28]:
```python
# random forest classifier object
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# Prediction on test
rf_pred = rf.predict(X_test)

# performance of random forest classifier on test
print(classification_report(y_test, rf_pred))
```

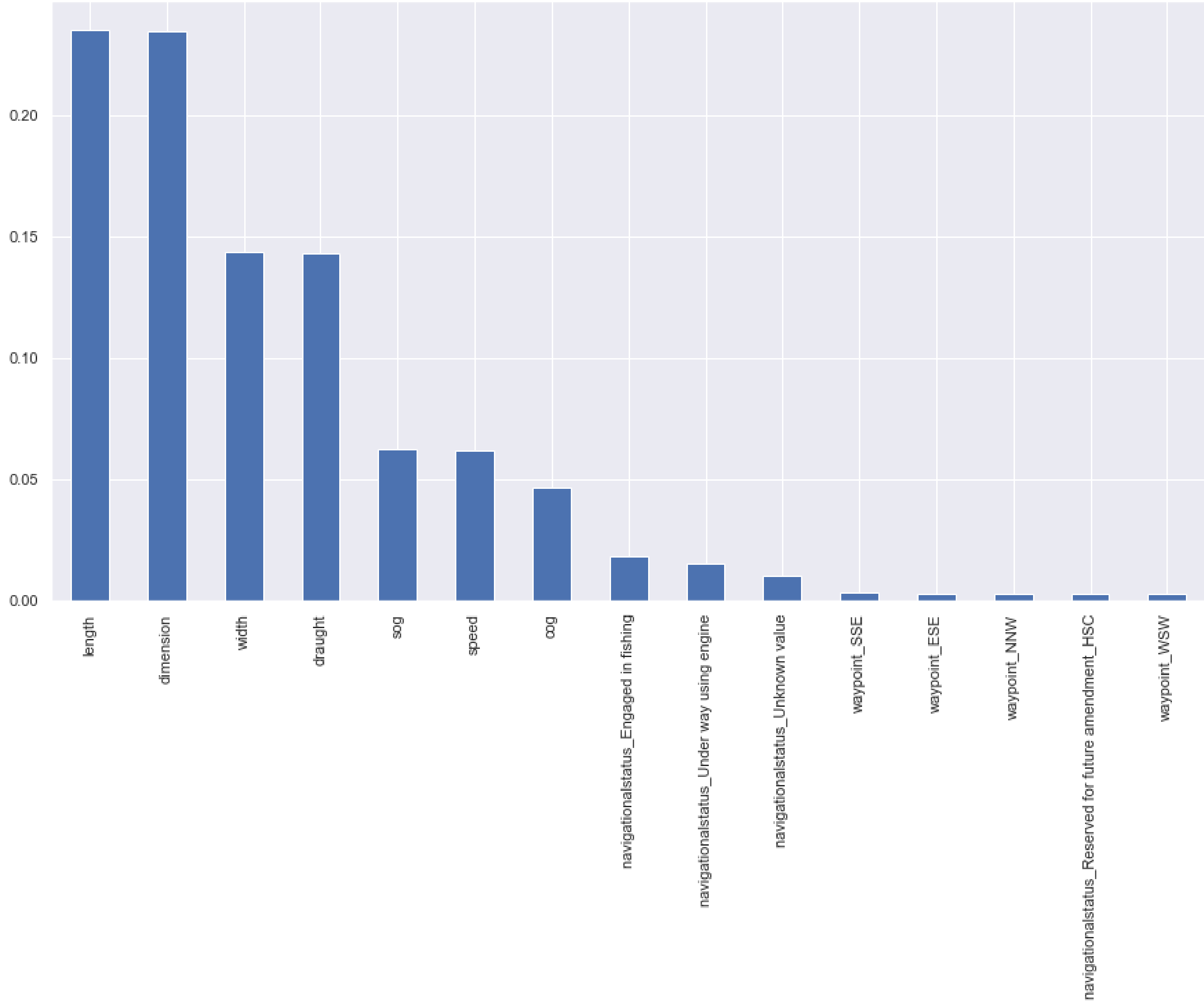
```
                   precision    recall  f1-score   support

            Cargo       0.99      0.99      0.99     37524
         Dredging       0.98      0.96      0.97      1062
          Fishing       0.96      0.97      0.97      4986
              HSC       1.00      0.99      0.99       709
  Law enforcement       0.99      0.98      0.99       320
         Military       0.99      0.98      0.98      1416
        Passenger       0.99      0.99      0.99      3298
            Pilot       0.97      0.98      0.97       789
         Pleasure       0.68      0.65      0.66       625
      Port tender       0.76      0.74      0.75        57
         Reserved       0.90      0.90      0.90       144
              SAR       0.92      0.90      0.91       709
          Sailing       0.62      0.64      0.62       370
           Tanker       0.99      0.97      0.98     15452
           Towing       0.91      0.91      0.91       208
Towing long/wide       0.97      0.91      0.94       122
              Tug       0.99      0.99      0.99      1885

         accuracy                           0.98     69676
        macro avg       0.92      0.91      0.91     69676
     weighted avg       0.98      0.98      0.98     69676
```

Random forest has similar performance like decision tree with **98%** accuracy on test set.

In [29]:

```
# feature importance
show_feat_imp(model=rf, X=X_test)
```

```
length                                                 0.24
dimension                                              0.23
width                                                  0.14
draught                                                0.14
sog                                                    0.06
speed                                                  0.06
cog                                                    0.05
navigationalstatus_Engaged in fishing                  0.02
navigationalstatus_Under way using engine              0.02
navigationalstatus_Unknown value                       0.01
waypoint_SSE                                           0.00
waypoint_ESE                                           0.00
waypoint_NNW                                           0.00
navigationalstatus_Reserved for future amendment_HSC   0.00
waypoint_WSW                                           0.00
dtype: float64
```



Feature importance is also similar to decision tree with top features being lenght, dimension, widht and draught.

7.4 LightGBM classifier

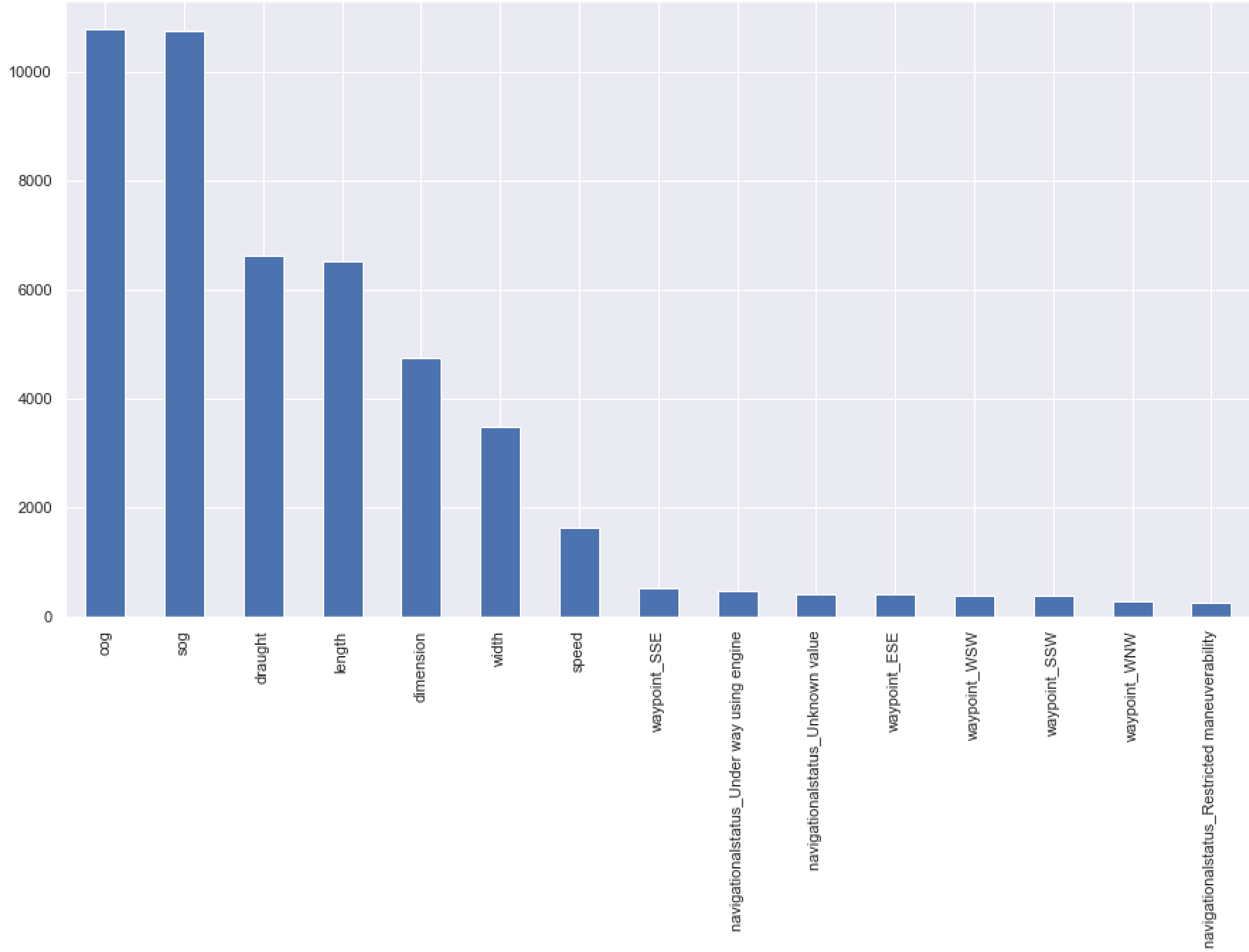
In [30]:
```
# random forest classifier object
lgbm = LGBMClassifier(random_state=7)
lgbm.fit(X_train, y_train)

# Prediction on test
lgbm_pred = lgbm.predict(X_test)

# performance of lgbm classifier on test
print(classification_report(y_test, lgbm_pred))
```

```
                   precision    recall  f1-score   support

           Cargo       0.83      0.83      0.83     37524
        Dredging       0.11      0.05      0.07      1062
         Fishing       0.58      0.69      0.63      4986
             HSC       0.00      0.00      0.00       709
 Law enforcement       0.18      0.25      0.21       320
        Military       0.36      0.34      0.35      1416
       Passenger       0.57      0.68      0.62      3298
           Pilot       0.28      0.40      0.33       789
        Pleasure       0.13      0.21      0.16       625
     Port tender       0.00      0.00      0.00        57
        Reserved       0.00      0.00      0.00       144
             SAR       0.07      0.13      0.09       709
         Sailing       0.09      0.09      0.09       370
          Tanker       0.78      0.60      0.68     15452
          Towing       0.03      0.16      0.05       208
Towing long/wide       0.01      0.10      0.02       122
             Tug       0.17      0.12      0.14      1885

        accuracy                           0.68     69676
       macro avg       0.25      0.27      0.25     69676
    weighted avg       0.71      0.68      0.69     69676
```

LightGBM has an accuracy of 68% only and is not performing as well as random forest.

In [31]:
```
# feature importance
show_feat_imp(model=lgbm, X=X_test)
```

```
cog                                             10759
sog                                             10730
draught                                          6614
length                                           6500
dimension                                        4754
width                                            3477
speed                                            1614
waypoint_SSE                                      508
navigationalstatus_Under way using engine         450
navigationalstatus_Unknown value                  407
waypoint_ESE                                      397
waypoint_WSW                                      370
waypoint_SSW                                      367
waypoint_WNW                                      263
navigationalstatus_Restricted maneuverability     252
dtype: int32
```



7.5 Linear SVM classifier

In [32]:
```
from sklearn.svm import LinearSVC
lsvm = LinearSVC(random_state=7)
lsvm.fit(X_train, y_train)

# Prediction on test
lsvm_pred = lsvm.predict(X_test)

# performance of lgbm classifier on test
print(classification_report(y_test, lsvm_pred))
```

```
                  precision    recall  f1-score   support

           Cargo       0.67      0.94      0.78     37524
        Dredging       0.28      0.01      0.02      1062
         Fishing       0.74      0.73      0.73      4986
             HSC       0.76      0.74      0.75       709
 Law enforcement       0.00      0.00      0.00       320
        Military       0.17      0.00      0.00      1416
       Passenger       0.63      0.18      0.27      3298
           Pilot       0.67      0.73      0.70       789
        Pleasure       0.51      0.05      0.09       625
     Port tender       0.00      0.00      0.00        57
        Reserved       0.00      0.00      0.00       144
             SAR       0.72      0.60      0.66       709
         Sailing       0.00      0.00      0.00       370
          Tanker       0.68      0.31      0.43     15452
          Towing       0.50      0.02      0.04       208
Towing long/wide       0.00      0.00      0.00       122
             Tug       0.67      0.73      0.70      1885

        accuracy                           0.68     69676
       macro avg       0.41      0.30      0.30     69676
    weighted avg       0.65      0.68      0.62     69676
```

Linear SVM also has an accuracy of 68% similar to LightGBM and is not performing as well as random forest.

7.6 Fine tuning Random Forest Classifier

Fine tuning hyperparameters of Random Forest Classifier since it is the best performing base model

7.6.1 Tuning Random Forest hyperparameters using Grid search CV

In [33]:

```
rf = RandomForestClassifier(random_state=7)

# specify the hyperparameters and their grid values
# 4 x 3 x 2 = 24 combinations in the grid

param_grid = {
    'n_estimators': [50, 100, 200, 400],
    'max_depth': [6, 10, None],
    'min_samples_split': [2, 5]
}

# using 5-fold cross-validation
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='f1_macro', return_train_score=True,

start = time.time()
grid_search.fit(X_train, y_train)
end = time.time() - start
print(f"Took {end} seconds")
```

```
Took 2315.9793124198914 seconds
```

In [34]:

```
print('best parameters from grid search:')
print(grid_search.best_estimator_)

print("best f1_macro score:", grid_search.best_score_)
```

```
best parameters from grid search:
RandomForestClassifier(min_samples_split=5, n_estimators=200, random_state=7)
best f1_macro score: 0.9100639583811295
```

Best hyperparameter values are:

max_depth: None

min_samples_split: 5

n_estimators: 200

Let's review the scores achieved by all the models in the search grid

In [35]:
```
# reviewing cv results for different hyperparameter sets
cv_results = pd.DataFrame(grid_search.cv_results_)[['params', 'mean_train_score', 'mean_test_
cv_results["diff, %"] = 100*(cv_results["mean_train_score"]-cv_results["mean_test_score"])/cv
pd.set_option('display.max_colwidth', 100)
cv_results.sort_values('mean_test_score', ascending=False)
```

Out[35]:

	params	mean_train_score	mean_test_score	diff, %
22	{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}	0.97	0.91	6.60
21	{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 100}	0.97	0.91	6.57
23	{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 400}	0.97	0.91	6.65
20	{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 50}	0.97	0.91	6.59
18	{'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}	1.00	0.91	9.15
19	{'max_depth': None, 'min_samples_split': 2, 'n_estimators': 400}	1.00	0.91	9.19
17	{'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}	1.00	0.91	9.23
16	{'max_depth': None, 'min_samples_split': 2, 'n_estimators': 50}	1.00	0.91	9.28
11	{'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 400}	0.71	0.70	2.21
10	{'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}	0.71	0.69	2.34
15	{'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 400}	0.71	0.69	2.40
14	{'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 200}	0.71	0.69	2.40
13	{'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 100}	0.70	0.69	2.21
9	{'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}	0.70	0.68	2.74
12	{'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 50}	0.70	0.68	1.90
8	{'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 50}	0.70	0.68	2.68
2	{'max_depth': 6, 'min_samples_split': 2, 'n_estimators': 200}	0.46	0.46	0.27
6	{'max_depth': 6, 'min_samples_split': 5, 'n_estimators': 200}	0.46	0.46	0.32
3	{'max_depth': 6, 'min_samples_split': 2, 'n_estimators': 400}	0.46	0.46	0.22
7	{'max_depth': 6, 'min_samples_split': 5, 'n_estimators': 400}	0.46	0.46	0.25
1	{'max_depth': 6, 'min_samples_split': 2, 'n_estimators': 100}	0.46	0.46	0.21
5	{'max_depth': 6, 'min_samples_split': 5, 'n_estimators': 100}	0.46	0.46	0.20
0	{'max_depth': 6, 'min_samples_split': 2, 'n_estimators': 50}	0.45	0.45	0.05
4	{'max_depth': 6, 'min_samples_split': 5, 'n_estimators': 50}	0.45	0.45	0.05

Random Forest classifier performance varies greatly across runs, ranging from a mean test f1_macro score of 0.45 to 0.91.
We notice that increasing the value of max depth results in better performance.
Decreasing min_samples_split also improves the performance but too little value leads to overfitting.

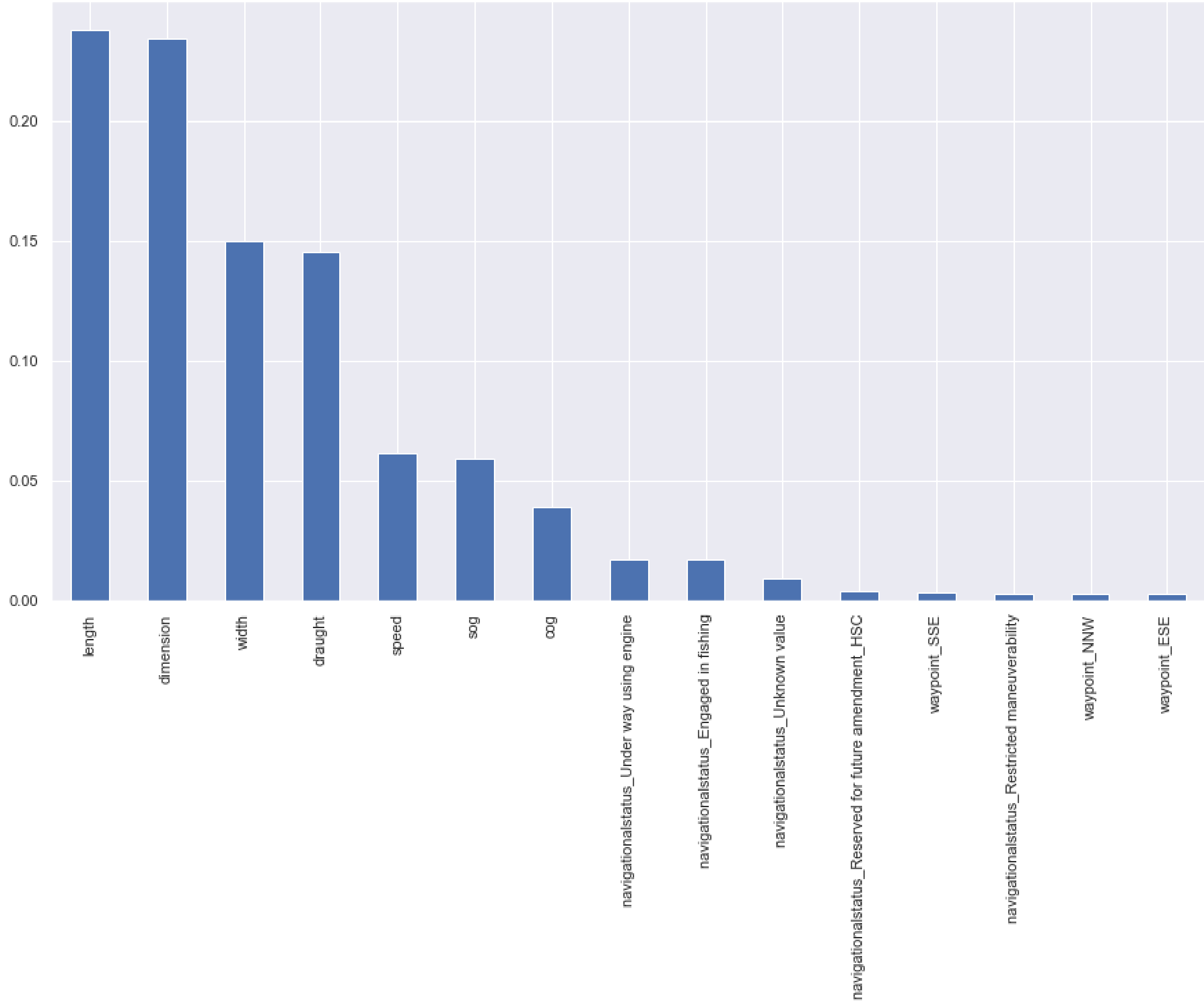
7.6.2 training the RF classifier using best hyperparameters

In [36]:
```
%%time

# random forest classifier object
rf = RandomForestClassifier(n_estimators=200,\
                            min_samples_split=5,
                            max_depth=None,
                            n_jobs=-1
                           )
rf.fit(X_train, y_train)

# Prediction on test
rf_pred = rf.predict(X_test)

# performance of random forest classifier on test
print(classification_report(y_test, rf_pred))
```

```
                  precision    recall  f1-score   support

           Cargo       0.99      0.99      0.99     37524
        Dredging       0.97      0.96      0.97      1062
         Fishing       0.96      0.97      0.97      4986
             HSC       1.00      0.98      0.99       709
 Law enforcement       1.00      0.98      0.99       320
        Military       0.99      0.98      0.98      1416
       Passenger       0.99      0.98      0.99      3298
           Pilot       0.97      0.98      0.98       789
        Pleasure       0.68      0.68      0.68       625
     Port tender       0.78      0.70      0.74        57
        Reserved       0.92      0.90      0.91       144
             SAR       0.93      0.90      0.92       709
         Sailing       0.63      0.63      0.63       370
          Tanker       0.99      0.97      0.98     15452
          Towing       0.90      0.91      0.91       208
Towing long/wide       0.97      0.90      0.94       122
             Tug       0.99      0.99      0.99      1885

        accuracy                           0.98     69676
       macro avg       0.92      0.91      0.91     69676
    weighted avg       0.98      0.98      0.98     69676

CPU times: total: 3min 45s
Wall time: 32.5 s
```

In [37]:
```
# feature importance
show_feat_imp(model=rf, X=X_test)
```

```
length                                                 0.24
dimension                                              0.23
width                                                  0.15
draught                                                0.15
speed                                                  0.06
sog                                                    0.06
cog                                                    0.04
navigationalstatus_Under way using engine              0.02
navigationalstatus_Engaged in fishing                  0.02
navigationalstatus_Unknown value                       0.01
navigationalstatus_Reserved for future amendment_HSC   0.00
waypoint_SSE                                           0.00
navigationalstatus_Restricted maneuverability          0.00
waypoint_NNW                                           0.00
waypoint_ESE                                           0.00
dtype: float64
```



Performance of the tuned random forest model:

1. accuracy: 98%
2. macro avg: 91%
3. weighted avg: 98%

Top features and their importance:

1. length: 24%
2. dimension: 23%
3. widht: 15%
4. draught: 15%
5. speed: 6%
6. cog: 6%
7. sog: 4%

8. Conclusion

In this assignment we were sucessfully loaded tha data into dataframe and performed exploratory data analysis. Mutiple libraries including pandas, numpy and matplotlib were used.
Insights were drawn about the different fields in the data and their relationship with each other.

Missing values were handled and outliers were detected.

In the next section, we performed predictive modeling with the objective of identifying the type of vessel(ship) given it's characteristic features.

For modeling exercise, data was split into train and test sets. Robust-scaling was applied to numerical fields and one hot encoding was done for categorical ones. Different ML algorithms were tested and Random Forest was choosen as the final model based on it's superior accuracy over other models.

By predicitng ship type it will be easier to provide.........

```
In [ ]:
```