Table of Contents

Introduction

Road assistance companies need to efficiently plan ahead of time to allocate their resources to meet demand on an hourly basis. Without meeting the demand, this could cause unnecessary prolonged obstructions on roads, leading to traffic and in effect resulting increased fuel consumption and effect on quality of life and economic productivity.

This project will involve analysing the hourly number of accidents over the course of 2018 to find trends in number of accidents around a specific hour of the day; also check factors like urban or rural area and precipitation affect the number of accidents that occur.

Import the libraries required for analysis

In [1]:

```
import warnings
warnings.filterwarnings("ignore")

import time
#library for data manipulation and preprocessing
import pandas as pd

#library for number crunching
import numpy as np

#library for visualisation
import matplotlib.pyplot as plt
import statsmodels.api as sm
import scipy.stats
import seaborn as sns
import matplotlib.dates as mdates
plt.style.use('bmh')

from sklearn.metrics import mean_squared_error
#time series data
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV

def get_mda(y, yhat):
    """Mean Directional Accuracy, as per:
    https://www.wikiwand.com/en/Mean_Directional_Accuracy
    """
    a = np.sign(np.diff(y))
    b = np.sign(np.diff(yhat))
    return np.sum(a == b)/a.shape[0]
```

Data

Load the data

In [2]:

```
df = pd.read_csv("UK Road Accidents 2018.csv")
df.head(5)
```

Out[2]:

	datetime	Count	Day_of_Week	Precipitation	High_Winds	Road_Surface_Conditions	Light_
0	2018-01-01 00:00:00	18	Monday	Fine	False	Dry	Darkr
1	2018-01-01 01:00:00	9	Monday	Fine	False	Wet or damp	Darkr
2	2018-01-01 02:00:00	14	Monday	Fine	False	Wet or damp	Darkr
3	2018-01-01 03:00:00	10	Monday	Fine	False	Wet or damp	Darkr
4	2018-01-01 04:00:00	8	Monday	Fine	False	Dry	Darkr

In [3]:

```
df.shape
```

Out[3]:

```
(8540, 8)
```

There are 8 columns in total in the dataset with 8540 entries. Each row contains records on accident occurences in a year, month, day, time, day of the week, weather percipitations, high winds, road surface conditions, light conditions and whether accident occurred in an urban or rural area.

The count column is the target variable in the dataset that will be predicted in the predictive models.

Sampling: Train Test Split

A train-test split will be performed using both `random sampling` and `stratified sampling` methods; where most appropriate will be chosen to ensure that the critical attributes of a population are correctly represented.

Random Sampling

In [4]:

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(df, test_size=0.2, random_state=7, shuffl
e=False)
print(f"{train_set.shape[0]} train instances and {test_set.shape[0]} test instan
ces")
```

```
6832 train instances and 1708 test instances
```

Stratified Sampling

In [5]:

```
from sklearn.model_selection import StratifiedShuffleSplit

stratified_splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_s
tate=7)

train_index, test_index = list(stratified_splitter.split(df, df["Day_of_Week"]))
[0]
strat_train_set = df.loc[train_index]
strat_test_set = df.loc[test_index]
```

Random vs Stratified

In [6]:

```
def day_of_week_proportions(data):
    return data["Day_of_Week"].value_counts() / len(data)

# create a random split
rand_train_set, rand_test_set = train_test_split(df, test_size=0.2, random_state
=7)

# create a temporary dataframe for easy visualization
df_tmp = pd.DataFrame({
    "Overall": day_of_week_proportions(df),
    "Random test set": day_of_week_proportions(rand_test_set),
    "Stratified test set": day_of_week_proportions(strat_test_set),
}).sort_index()

# add two columns for the percent of the difference to the overall proportion
df_tmp["Rand. %error"] = 100 * df_tmp["Random test set"] / df_tmp["Overall"] - 1
00
df_tmp["Strat. %error"] = 100 * df_tmp["Stratified test set"] / df_tmp["Overall"
] - 100

df_tmp
```

Out[6]:

	Overall	Random test set	Stratified test set	Rand. %error	Strat. %error
Friday	0.142506	0.151639	0.142272	6.409203	-0.164339
Monday	0.144496	0.149297	0.144614	3.322528	0.081037
Saturday	0.145550	0.144614	0.145785	-0.643604	0.160901
Sunday	0.145667	0.144614	0.145785	-0.723473	0.080386
Thursday	0.140984	0.140515	0.141101	-0.332226	0.083056
Tuesday	0.139578	0.131148	0.139344	-6.040268	-0.167785
Wednesday	0.141218	0.138173	0.141101	-2.155887	-0.082919

As we can see in the above table, the random splitting produces a test set where Friday and Monday are over represented by 6% and 3% respectively. Tuesday and Wednesday are under-represented by 6% and 2% respectively.

Stratified sampling resulted in under- or over-representations of the days of the week by no more that 0.16%. Choose to use stratified sampling as it is more representative of the dataset.

Rename variables

In [7]:

```
trainset = strat_train_set
testset = strat_test_set
```

Exploratory Data Analysis

Data Types

In [8]:

```
trainset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6832 entries, 2092 to 426
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   datetime                 6832 non-null   object
 1   Count                    6832 non-null   int64 
 2   Day_of_Week              6832 non-null   object
 3   Precipitation            6832 non-null   object
 4   High_Winds               6832 non-null   bool  
 5   Road_Surface_Conditions  6832 non-null   object
 6   Light_Conditions         6832 non-null   object
 7   Urban_or_Rural_Area      6832 non-null   object
dtypes: bool(1), int64(1), object(6)
memory usage: 433.7+ KB
```

From an overview, we can see that the datetime column datatype needs to be converted from an object to datetime64[ns]. In the non-null column, it looks like there are no missing values; however this is will be checked and confirmed.

Convert datetime column

In [9]:

```
trainset['datetime'] = pd.to_datetime(trainset['datetime'])
trainset.head()
```

Out[9]:

	datetime	Count	Day_of_Week	Precipitation	High_Winds	Road_Surface_Conditions	Lig
2092	2018-04-01 05:00:00	2	Sunday	Fine	False	Wet or damp	D
6441	2018-10-03 18:00:00	18	Wednesday	Fine	False	Dry	
6172	2018-09-22 10:00:00	13	Saturday	Fine	False	Dry	
4187	2018-06-29 16:00:00	32	Friday	Fine	False	Dry	
2330	2018-04-11 10:00:00	12	Wednesday	Fine	False	Wet or damp	

In [10]:

```
#repeat for testset
testset['datetime'] = pd.to_datetime(testset['datetime'])
```

Value counts in each column (categorical)

In [11]:

```
trainset['Day_of_Week'].value_counts()
```

Out[11]:

```
Sunday       995
Saturday     994
Monday       987
Friday       974
Wednesday    965
Thursday     963
Tuesday      954
Name: Day_of_Week, dtype: int64
```

In [12]:

```
trainset['Precipitation'].value_counts()
```

Out[12]:

```
Fine                              5996
Raining                            645
Snowing                             92
Other                               44
Unknown                             35
Fog or mist                         19
Data missing or out of range         1
Name: Precipitation, dtype: int64
```

80 records are classed as 'other', 'unknown' or 'data missing out of range'. Missing values will be dealt with in next step

In [13]:

```
trainset['High_Winds'].value_counts()
```

Out[13]:

```
False    6776
True       56
Name: High_Winds, dtype: int64
```

In [14]:

```
trainset['Road_Surface_Conditions'].value_counts()
```

Out[14]:

```
Dry                               5056
Wet or damp                       1631
Snow                                90
Frost or ice                        42
Data missing or out of range         8
Flood over 3cm. deep                 5
Name: Road_Surface_Conditions, dtype: int64
```

8 records classed as 'data missing or out of range'

In [15]:

```
trainset['Light_Conditions'].value_counts()
```

Out[15]:

```
Daylight                         3859
Darkness - lights lit            2453
Darkness - no lighting            414
Darkness - lighting unknown        84
Darkness - lights unlit            22
Name: Light_Conditions, dtype: int64
```

In [16]:

```
trainset['Urban_or_Rural_Area'].value_counts()
```

Out[16]:

```
Urban    5525
Rural    1307
Name: Urban_or_Rural_Area, dtype: int64
```

Data Transformation

Missing Values

The `Precipitation` , `Road_Surface_Conditions` , columns have records where the records are `Unknown` , `Data missing or out of range` and `Other` . Here we are going to replace the missing values using median.

Precipitation column

In [17]:

```
#Filter for missing values
filter_list = ['Data missing or out of range', 'Unknown', 'Other']
trainset[trainset.Precipitation.isin(filter_list)]
```

Out[17]:

	datetime	Count	Day_of_Week	Precipitation	High_Winds	Road_Surface_Conditions	Lig
8451	2018-12-28 05:00:00	1	Friday	Other	False	Wet or damp	lig
1301	2018-02-26 02:00:00	1	Monday	Other	False	Dry	
5844	2018-09-08 08:00:00	5	Saturday	Unknown	False	Dry	
5399	2018-08-20 04:00:00	2	Monday	Unknown	False	Wet or damp	lig
7391	2018-11-13 01:00:00	2	Tuesday	Unknown	False	Wet or damp	
...	...	...	...	...	...	...	
27	2018-01-02 04:00:00	1	Tuesday	Other	False	Dry	D
8120	2018-12-14 04:00:00	1	Friday	Unknown	False	Data missing or out of range	D
2138	2018-04-03 04:00:00	1	Tuesday	Unknown	False	Data missing or out of range	
6047	2018-09-17 00:00:00	2	Monday	Unknown	False	Dry	
2368	2018-04-13 01:00:00	1	Friday	Other	False	Wet or damp	D

80 rows × 8 columns

We can see from the `trainset['Precipitation'].value_counts()` line, that all rows add up to 80 records; as evidenced in above table

In [18]:

```
# replace these rows with nan
trainset['Precipitation'] = trainset['Precipitation'].replace(['Other','Unknown'
,'Data missing or out of range'],np.NaN)
```

In [19]:

```
#check results
filter_list = ['Data missing or out of range', 'Unknown', 'Other']
trainset[trainset.Precipitation.isin(filter_list)]
```

Out[19]:

datetime	Count	Day_of_Week	Precipitation	High_Winds	Road_Surface_Conditions	Light_C

In [20]:

```
#repeat for testset
testset['Precipitation'] = testset['Precipitation'].replace(['Other','Unknown',
'Data missing or out of range'],np.NaN)
```

Road Surface Conditions column

In [21]:

```
#filter for missing values
filter_list = ['Data missing or out of range', 'Unknown', 'Other']
trainset[trainset.Road_Surface_Conditions.isin(filter_list)]
```

Out[21]:

	datetime	Count	Day_of_Week	Precipitation	High_Winds	Road_Surface_Conditions	Lig
1810	2018-03-20 03:00:00	1	Tuesday	NaN	False	Data missing or out of range	D
4973	2018-08-02 02:00:00	2	Thursday	NaN	False	Data missing or out of range	
4761	2018-07-24 02:00:00	1	Tuesday	Fine	False	Data missing or out of range	D
7553	2018-11-19 23:00:00	2	Monday	NaN	False	Data missing or out of range	
5753	2018-09-04 03:00:00	2	Tuesday	NaN	False	Data missing or out of range	
6687	2018-10-14 04:00:00	4	Sunday	Fine	False	Data missing or out of range	lig
8120	2018-12-14 04:00:00	1	Friday	NaN	False	Data missing or out of range	D
2138	2018-04-03 04:00:00	1	Tuesday	NaN	False	Data missing or out of range	

In [22]:

```
#replace these rows with nan
trainset['Road_Surface_Conditions'] = trainset['Road_Surface_Conditions'].replac
e(['Other','Unknown','Data missing or out of range'],np.NaN)
```

In [23]:

```
#check results
filter_list = ['Data missing or out of range', 'Unknown', 'Other']
trainset[trainset.Road_Surface_Conditions.isin(filter_list)]
```

Out[23]:

datetime	Count	Day_of_Week	Precipitation	High_Winds	Road_Surface_Conditions	Light_C

In [24]:

```
#repeat for testset
testset['Road_Surface_Conditions'] = testset['Road_Surface_Conditions'].replace
(['Other','Unknown','Data missing or out of range'],np.NaN)
```

Check for missing values

In [25]:

```
trainset.isna().sum()
```

Out[25]:

```
datetime                    0
Count                       0
Day_of_Week                 0
Precipitation              80
High_Winds                  0
Road_Surface_Conditions     8
Light_Conditions            0
Urban_or_Rural_Area         0
dtype: int64
```

- Replace the missing values with mode in the dataset

In [26]:

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="most_frequent")

trainset_categorical = trainset["Precipitation"].values
trainset_numerical = trainset.drop("Precipitation", axis=1)

imputer.fit(trainset_numerical)
```

Out[26]:

```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='most_frequent', verbose=
0)
```

In [27]:

```
transformed = imputer.transform(trainset_numerical)
```

In [28]:

```
trainset = pd.DataFrame(transformed, columns=trainset_numerical.columns)

# add the categorical variable back
trainset['Precipitation'] = trainset_categorical

# check if there are missing values again
trainset.isnull().sum()
```

Out[28]:

```
datetime                   0
Count                      0
Day_of_Week                0
High_Winds                 0
Road_Surface_Conditions    0
Light_Conditions           0
Urban_or_Rural_Area        0
Precipitation             80
dtype: int64
```

In [29]:

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="most_frequent")

trainset_categorical = trainset["Road_Surface_Conditions"].values
trainset_numerical = trainset.drop("Road_Surface_Conditions", axis=1)

imputer.fit(trainset_numerical)
```

Out[29]:

```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='most_frequent', verbose=
0)
```

In [30]:

```
transformed = imputer.transform(trainset_numerical)
```

In [31]:

```
trainset = pd.DataFrame(transformed, columns=trainset_numerical.columns)

# add the categorical variable back
trainset['Road_Surface_Conditions'] = trainset_categorical

# check if there are missing values again
trainset.isnull().sum()
```

Out[31]:

```
datetime                   0
Count                      0
Day_of_Week                0
High_Winds                 0
Light_Conditions           0
Urban_or_Rural_Area        0
Precipitation              0
Road_Surface_Conditions    0
dtype: int64
```

Transform the test dataset

In [32]:

```
testset.isnull().sum()
```

Out[32]:

```
datetime                    0
Count                       0
Day_of_Week                 0
Precipitation              20
High_Winds                  0
Road_Surface_Conditions     3
Light_Conditions            0
Urban_or_Rural_Area         0
dtype: int64
```

In [33]:

```
# precipitation
testset_categorical = testset["Precipitation"].values
testset_numerical = testset.drop("Precipitation", axis=1)

transformed = imputer.transform(testset_numerical)

testset = pd.DataFrame(transformed, columns=testset_numerical.columns)

# add the categorical variable back
testset['Precipitation'] = testset_categorical

# check if there are missing values again
testset.isnull().sum()
```

Out[33]:

```
datetime                  0
Count                     0
Day_of_Week               0
High_Winds                0
Road_Surface_Conditions   0
Light_Conditions          0
Urban_or_Rural_Area       0
Precipitation            20
dtype: int64
```

In [34]:

```
#road surface conditions
testset_categorical = testset["Road_Surface_Conditions"].values
testset_numerical = testset.drop("Road_Surface_Conditions", axis=1)

transformed = imputer.transform(testset_numerical)

testset = pd.DataFrame(transformed, columns=testset_numerical.columns)

# add the categorical variable back
testset['Road_Surface_Conditions'] = testset_categorical

# check if there are missing values again
testset.isnull().sum()
```

Out[34]:

```
datetime                  0
Count                     0
Day_of_Week               0
High_Winds                0
Light_Conditions          0
Urban_or_Rural_Area       0
Precipitation             0
Road_Surface_Conditions   0
dtype: int64
```

There are now no missing values in the train and test dataset

Feature Engineering

- Adding month,hour of the day, year, season as extra columns.
- Mapping the weekdays to a dict to use the actual weekdays in words like 'Monday', etc. Python usually indicates Monday as 0, Tuesday as 1 and so on.
- Adding season based on the UK calender

In [35]:

```
def season_calc(month):
    """adding season based on the UK weather """
    if month in [3,4,5]:
        return "spring"
    if month in [6,7,8]:
        return "summer"
    if month in [9,10,11]:
        return "autumn"
    else:
        return "winter"
```

In [36]:

```
trainset['year'] = trainset.datetime.dt.year
trainset['month'] = trainset.datetime.dt.month
trainset['day'] = trainset.datetime.dt.day
trainset['hour'] = trainset.datetime.dt.hour
trainset['season'] = trainset.datetime.dt.month.apply(season_calc)
```

In [37]:

```
#check the results
trainset.head()
```

Out[37]:

	datetime	Count	Day_of_Week	High_Winds	Light_Conditions	Urban_or_Rural_Area	Precipi
0	2018-04-01 05:00:00	2	Sunday	False	Darkness - lights lit	Urban	
1	2018-10-03 18:00:00	18	Wednesday	False	Daylight	Urban	
2	2018-09-22 10:00:00	13	Saturday	False	Daylight	Rural	
3	2018-06-29 16:00:00	32	Friday	False	Daylight	Urban	
4	2018-04-11 10:00:00	12	Wednesday	False	Daylight	Urban	

- Repeat steps for test data set

In [38]:

```
testset['year'] = testset.datetime.dt.year
testset['month'] = testset.datetime.dt.month
testset['day'] = testset.datetime.dt.day
testset['hour'] = testset.datetime.dt.hour
testset['season'] = testset.datetime.dt.month.apply(season_calc)
```

Create dummy variables

As predictive models will be build with the effect of area, dummy variables need to be created as these are categorical variables

In [39]:

```
from sklearn.preprocessing import OneHotEncoder

one_hot_encoder = OneHotEncoder(drop="first", sparse=False)

# the input to the encoder must be a 2-d numpy array,
# so we take the column, extract their values and reshape the array to be 2-d
cat_vals = trainset['Urban_or_Rural_Area'].values.reshape(-1,1)

transformed = one_hot_encoder.fit_transform(cat_vals)

# put the transformed data as columns in the dataframe
col_names = one_hot_encoder.categories_[0].tolist()[1:]
for i, col_name in enumerate(col_names):
    trainset[col_name] = transformed[:,i]

# check if the dummies are produced correctly
trainset.head(2)
```

Out[39]:

	datetime	Count	Day_of_Week	High_Winds	Light_Conditions	Urban_or_Rural_Area	Precipi
0	2018-04-01 05:00:00	2	Sunday	False	Darkness - lights lit	Urban	
1	2018-10-03 18:00:00	18	Wednesday	False	Daylight	Urban	

In [40]:

```
# delete the categorical column
del trainset['Urban_or_Rural_Area']
```

Create dummy variables in the test set

In [41]:

```
cat_vals = testset['Urban_or_Rural_Area'].values.reshape(-1,1)
transformed = one_hot_encoder.transform(cat_vals)

for i, col_name in enumerate(col_names):
    testset[col_name] = transformed[:,i]

# check if the dummies are produced correctly
testset.head()
```

Out[41]:

	datetime	Count	Day_of_Week	High_Winds	Light_Conditions	Urban_or_Rural_Area	Precipi
0	2018-03-06 16:00:00	26	Tuesday	False	Daylight	Urban	
1	2018-05-20 17:00:00	23	Sunday	False	Daylight	Urban	
2	2018-04-03 14:00:00	24	Tuesday	False	Daylight	Urban	
3	2018-01-31 18:00:00	32	Wednesday	False	Darkness - lights lit	Urban	
4	2018-04-26 11:00:00	19	Thursday	False	Daylight	Urban	

In [42]:

```
# delete the categorical column
del testset['Urban_or_Rural_Area']
```

Finalise trainset for analysis

In [43]:

```
trainset.head(2)
```

Out[43]:

	datetime	Count	Day_of_Week	High_Winds	Light_Conditions	Precipitation	Road_Surface_
0	2018-04-01 05:00:00	2	Sunday	False	Darkness - lights lit	Fine	W
1	2018-10-03 18:00:00	18	Wednesday	False	Daylight	Fine	

- Sort series data in line with date

In [44]:

```
trainset = trainset.sort_values(by=['datetime'])
trainset.head()
```

Out[44]:

	datetime	Count	Day_of_Week	High_Winds	Light_Conditions	Precipitation	Road_Surfa
824	2018-01-01 00:00:00	18	Monday	False	Darkness - lights lit	Fine	
263	2018-01-01 02:00:00	14	Monday	False	Darkness - lights lit	Fine	
3993	2018-01-01 03:00:00	10	Monday	False	Darkness - lights lit	Fine	
6325	2018-01-01 04:00:00	8	Monday	False	Darkness - lights lit	Fine	
3709	2018-01-01 05:00:00	9	Monday	False	Darkness - lights lit	Fine	

In [45]:

```
trainset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6832 entries, 824 to 2458
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   datetime                 6832 non-null   datetime64[ns]
 1   Count                    6832 non-null   object
 2   Day_of_Week              6832 non-null   object
 3   High_Winds               6832 non-null   object
 4   Light_Conditions         6832 non-null   object
 5   Precipitation            6832 non-null   object
 6   Road_Surface_Conditions  6832 non-null   object
 7   year                     6832 non-null   int64
 8   month                    6832 non-null   int64
 9   day                      6832 non-null   int64
 10  hour                     6832 non-null   int64
 11  season                   6832 non-null   object
 12  Urban                    6832 non-null   float64
dtypes: datetime64[ns](1), float64(1), int64(4), object(7)
memory usage: 747.2+ KB
```

- Reinstate int64 to Count column

In [46]:

```
trainset['Count'] = trainset['Count'].astype(str).astype(int)
```

In [47]:

```
trainset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6832 entries, 824 to 2458
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   datetime                 6832 non-null   datetime64[ns]
 1   Count                    6832 non-null   int64
 2   Day_of_Week              6832 non-null   object
 3   High_Winds               6832 non-null   object
 4   Light_Conditions         6832 non-null   object
 5   Precipitation            6832 non-null   object
 6   Road_Surface_Conditions  6832 non-null   object
 7   year                     6832 non-null   int64
 8   month                    6832 non-null   int64
 9   day                      6832 non-null   int64
 10  hour                     6832 non-null   int64
 11  season                   6832 non-null   object
 12  Urban                    6832 non-null   float64
dtypes: datetime64[ns](1), float64(1), int64(5), object(6)
memory usage: 747.2+ KB
```

- Convert Day_of_Week and season to category

In [48]:

```
for col in ['spring', 'summer', 'autumn', 'winter']:
    trainset['season'] = trainset['season'].astype('category')
```

In [49]:

```
for col in ['Monday', 'Tuesday', 'Wednesday','Thursday', 'Friday','Saturday', 'S
unday']:
    trainset['Day_of_Week'] = trainset['Day_of_Week'].astype('category')
```

In [50]:

```
trainset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6832 entries, 824 to 2458
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   datetime                 6832 non-null   datetime64[ns]
 1   Count                    6832 non-null   int64
 2   Day_of_Week              6832 non-null   category
 3   High_Winds               6832 non-null   object
 4   Light_Conditions         6832 non-null   object
 5   Precipitation            6832 non-null   object
 6   Road_Surface_Conditions  6832 non-null   object
 7   year                     6832 non-null   int64
 8   month                    6832 non-null   int64
 9   day                      6832 non-null   int64
 10  hour                     6832 non-null   int64
 11  season                   6832 non-null   category
 12  Urban                    6832 non-null   float64
dtypes: category(2), datetime64[ns](1), float64(1), int64(5), object
(4)
memory usage: 654.4+ KB
```

- Select columns for analysis

In [51]:

```
trainset = trainset.drop(columns=['High_Winds', 'Light_Conditions', 'Precipitati
on', 'Road_Surface_Conditions'])
trainset.head(2)
```

Out[51]:

	datetime	Count	Day_of_Week	year	month	day	hour	season	Urban
824	2018-01-01 00:00:00	18	Monday	2018	1	1	0	winter	1.0
263	2018-01-01 02:00:00	14	Monday	2018	1	1	2	winter	1.0

In [52]:

```
#set datetime as index
trainset = trainset.set_index('datetime')
trainset.index
```

Out[52]:

```
DatetimeIndex(['2018-01-01 00:00:00', '2018-01-01 02:00:00',
               '2018-01-01 03:00:00', '2018-01-01 04:00:00',
               '2018-01-01 05:00:00', '2018-01-01 06:00:00',
               '2018-01-01 07:00:00', '2018-01-01 09:00:00',
               '2018-01-01 10:00:00', '2018-01-01 11:00:00',
               ...
               '2018-12-31 08:00:00', '2018-12-31 09:00:00',
               '2018-12-31 11:00:00', '2018-12-31 15:00:00',
               '2018-12-31 16:00:00', '2018-12-31 17:00:00',
               '2018-12-31 18:00:00', '2018-12-31 20:00:00',
               '2018-12-31 22:00:00', '2018-12-31 23:00:00'],
              dtype='datetime64[ns]', name='datetime', length=6832,
freq=None)
```

In [53]:

```
trainset.head(3)
```

Out[53]:

	Count	Day_of_Week	year	month	day	hour	season	Urban
datetime								
2018-01-01 00:00:00	18	Monday	2018	1	1	0	winter	1.0
2018-01-01 02:00:00	14	Monday	2018	1	1	2	winter	1.0
2018-01-01 03:00:00	10	Monday	2018	1	1	3	winter	1.0

Repeat for test dataset

In [54]:

```
testset = testset.sort_values(by=['datetime'])
```

In [55]:

```
testset['Count'] = testset['Count'].astype(str).astype(int)
```

In [56]:

```
for col in ['spring', 'summer', 'autumn', 'winter']:
    trainset['season'] = trainset['season'].astype('category')
```

In [57]:

```
for col in ['Monday', 'Tuesday', 'Wednesday','Thursday', 'Friday','Saturday', 'S
unday']:
    trainset['Day_of_Week'] = trainset['Day_of_Week'].astype('category')
```

In [58]:

```
testset = testset.drop(columns=['High_Winds', 'Light_Conditions', 'Precipitatio
n', 'Road_Surface_Conditions'])
```

In [59]:

```
testset = testset.set_index('datetime')
```

Visualisations

Heatmap

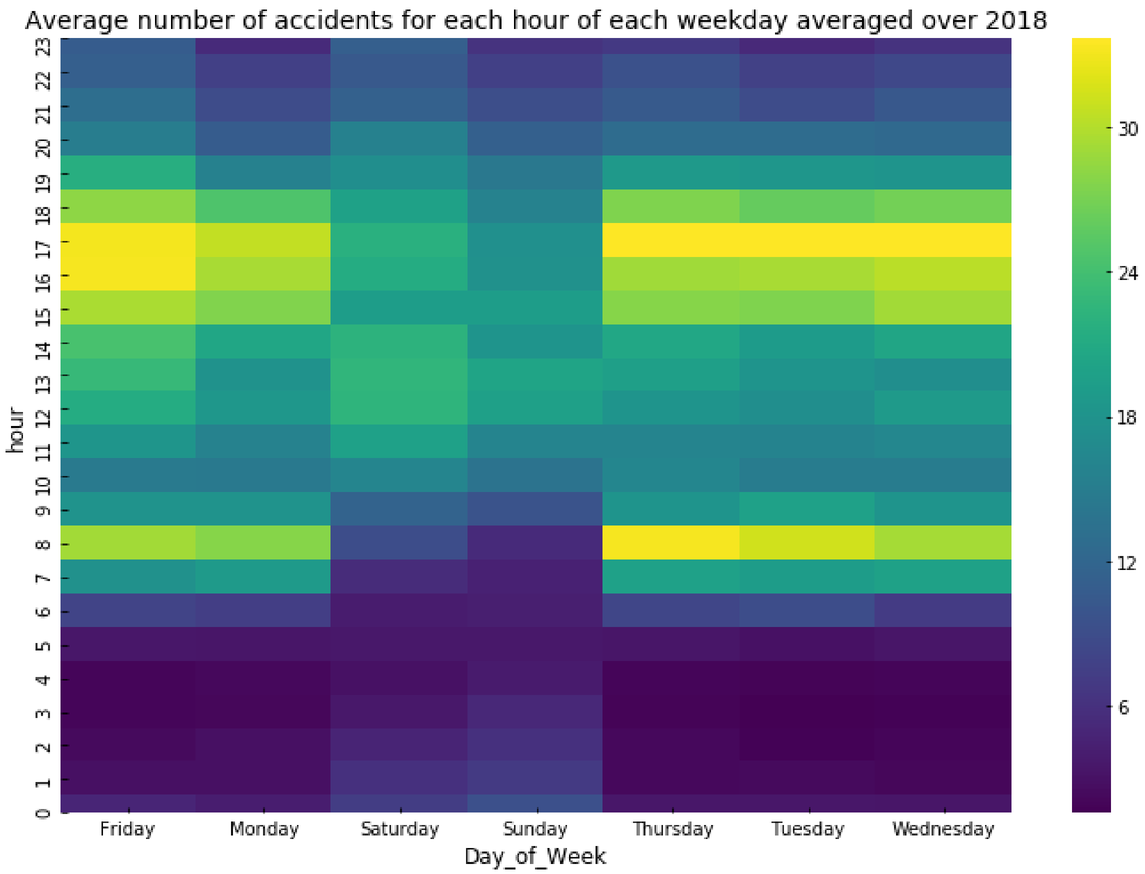
Plotting a map of hourly vs weekdays number of accidents to variation across any particular week

In [60]:

```
hour_weekday = trainset.pivot_table(values='Count', index='hour', columns = 'Day
_of_Week', aggfunc = 'mean')
```

In [61]:

```
#plotting a heatmap with a colorbar; the colorbar shows the number of accidents
_ = plt.figure(figsize=(12, 8))
ax = sns.heatmap(hour_weekday.sort_index(ascending = False), cmap='viridis')
#_ = plot title
_ = ax.set_title("Average number of accidents for each hour of each weekday aver
aged over 2018", fontsize = 14)
```



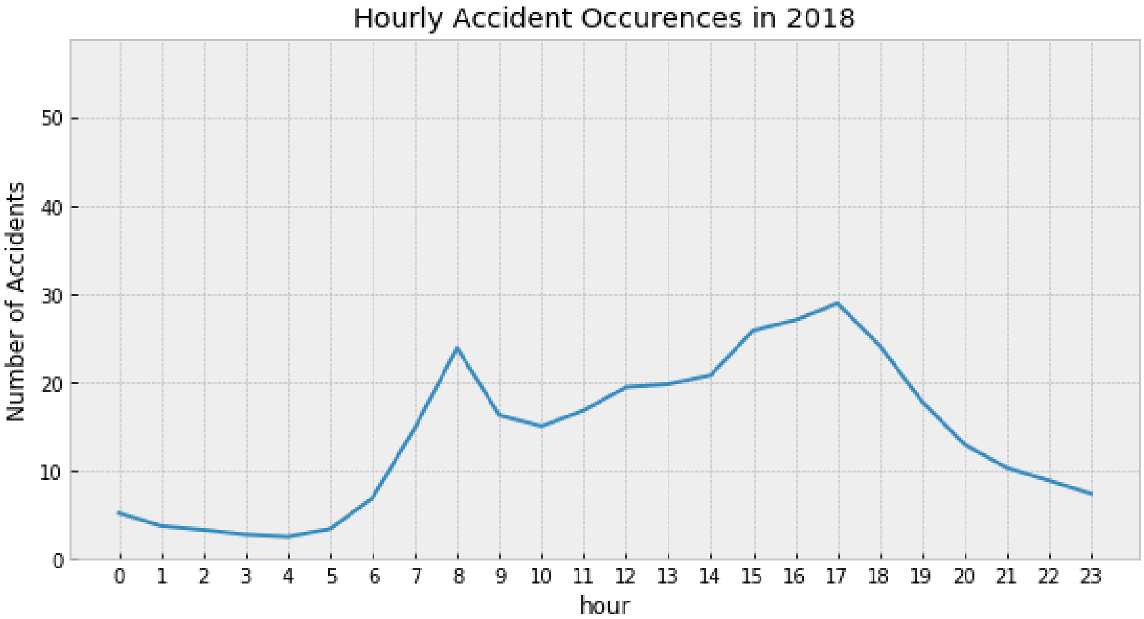
- The heatmap above shows that the average number of accidents from Monday to Friday is below 10 in the night. This increses between 7:30am and 8:30am, slows down slightly around midday and picks up again between 3pm and 6pm.
- Over the weekend, the average number of accidents rises above 12 in the night to early morning. Overall accidents occur less on the weekends.
- During the weekdays, the reason for the higher average number of accidents occuring between 7:30am -8:30am and 3pm - 6pm can be explained by peak times where individuals are rushing to work and school.

Average Hourly 2018

Plotting average hourly accidents over 2018

In [62]:

```
#Plotting average hourly number of accidents observed over the entire period
trainset.groupby('hour')['Count'].mean().plot(figsize = (10,5))
_ = plt.ylabel('Number of Accidents')
_ = plt.ylim([0, max(trainset.groupby('hour')['Count'].mean()) + 30])
_ = plt.xticks(trainset['hour'].unique())
_ = plt.title('Hourly Accident Occurences in 2018')
```



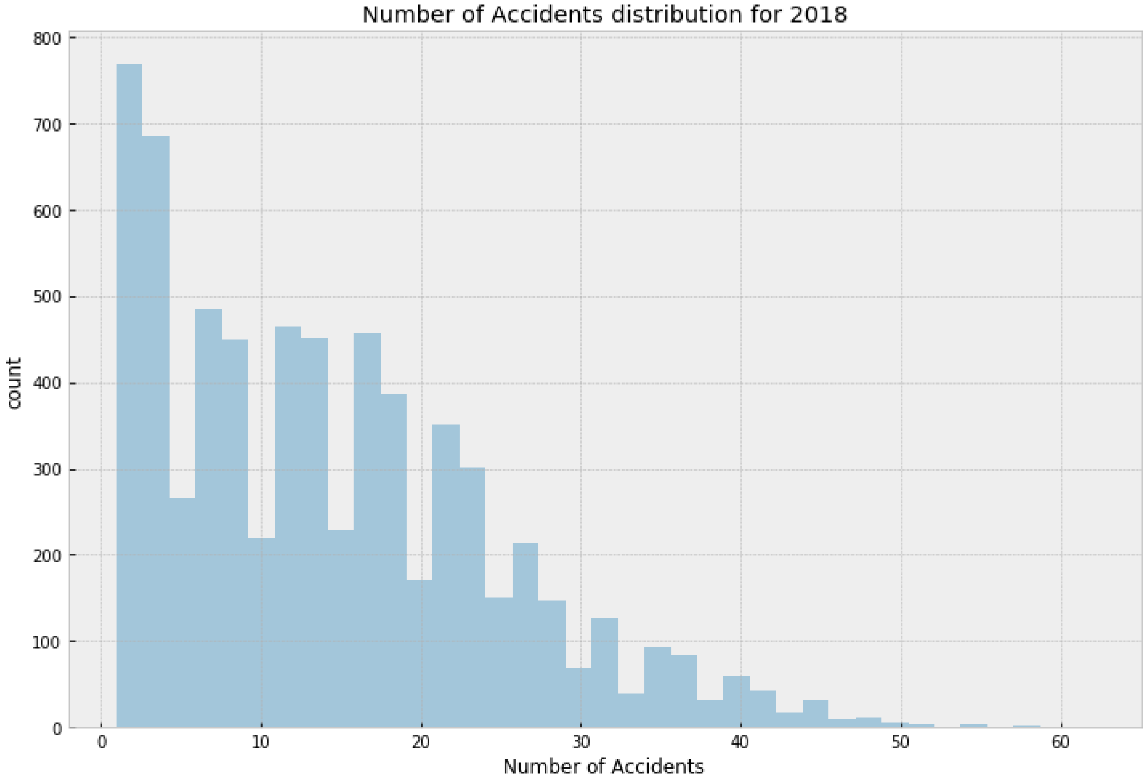
- As expected the hourly accidents peak in the early hours in the day and 3pm-5:30pm approximately due to rush hour/peak time. This was also shown in the heatmap plotted above

Plotting a histogram

Visualise the distribution of the number of accidents recorded

In [63]:

```
_ = plt.figure(figsize = (12,8))
_ = sns.distplot(trainset['Count'], kde=False)
_ = plt.title('Number of Accidents distribution for 2018')
_ = plt.xlabel('Number of Accidents')
_ = plt.ylabel('count')
```



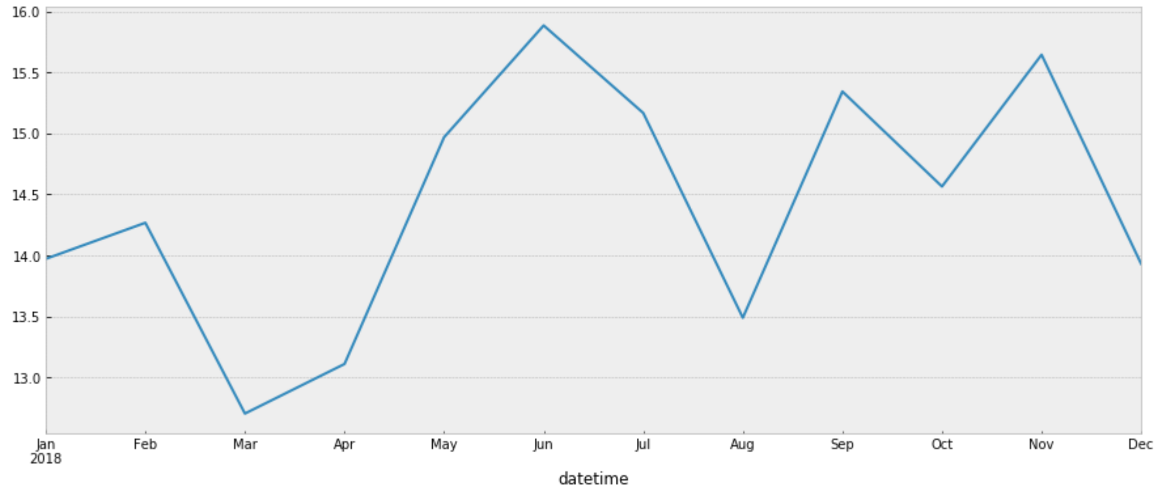
The histogram above shows:

- for most of the days in 2018, there were around 5 - 25 accidents a day
- highest number of accidents in a day was around 58 accidents
- There are no outliers/anomalies in the data to deal with

Average Monthly 2018

In [64]:

```
avm = trainset['Count'].resample('MS').mean()
avm.plot(figsize=(15, 6))
plt.show()
```



- The graph above shows the peaks in the number of accidents that occur during the summmer and autumn months

Time Series

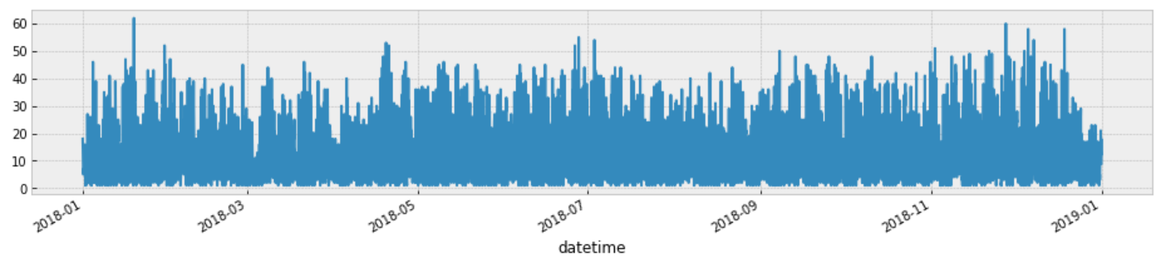
Plot time series dataset

In [65]:

```
tsp = trainset['Count']
tsp.plot(figsize=(16,3))
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f916a69e950>
```



Correlations

As a predictive model will be built with questions on if area has an effect on the number of accidents, would be helpful to calculate correlation coefficient

In [66]:

```
# Let's calculate the pearsonr coefficient between the number of accidents and a
rea being urban or rural
scipy.stats.pearsonr(trainset['Count'], trainset['Urban'])
```

Out[66]:

```
(0.3072922466819525, 2.4416961710152814e-149)
```

- There is a postive correlation between the number of accidents and the area in which the accidents occurred.

Time Series Model

To build time series model, additonal preprocessing and transformation steps are required

Additional Preprocessing and Transformation

In [67]:

```
#create a simple time series dataframe
tsd = pd.DataFrame(trainset["Count"], columns=['Count'])
```

In [68]:

```
tsd.head(3)
```

Out[68]:

	Count
datetime	
2018-01-01 00:00:00	18
2018-01-01 02:00:00	14
2018-01-01 03:00:00	10

In [69]:

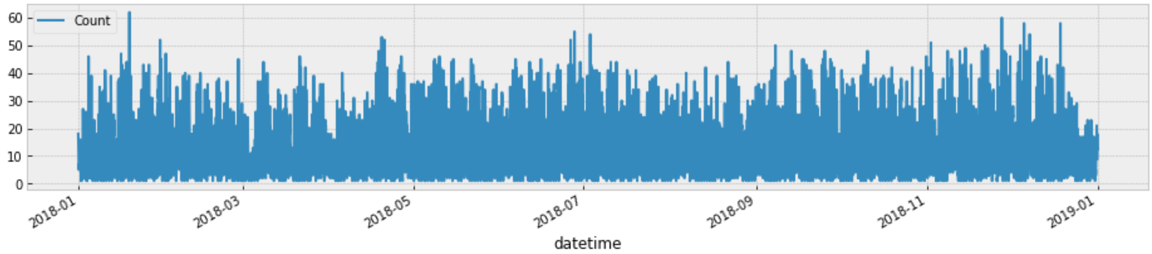
```
#repeat for test data
test2 = pd.DataFrame(testset["Count"], columns=['Count'])
```

In [70]:

```
# Plot the time series
tsd.plot(figsize=(16,3))
```

Out[70]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9169baa810>
```

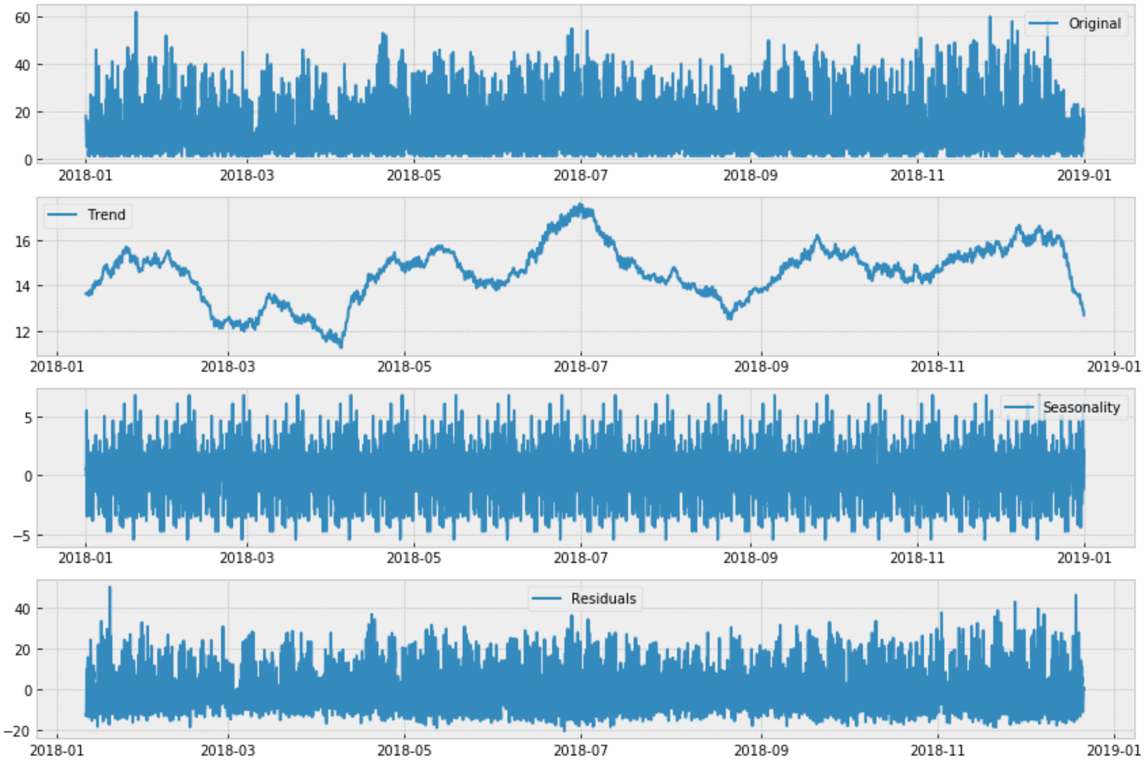


Decompose time series

Decompose the time series data by `trend` , `seasonality` and `residuals`

In [71]:

```
from statsmodels.tsa.seasonal import seasonal_decompose

# specify the number of observations in a cycle
decomposition = seasonal_decompose(tsd['Count'], freq=365)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.figure(figsize=(12, 8))
plt.subplot(411)
plt.plot(tsd['Count'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```



- There seems to be seasonality during with the year, with an upward trend in the number of accidents in the summer and autumn months.
- The hourly change in the number of accidents has no cyclic behaviour.
- There does not seem to be any random fluctuations in the series.

Testing for stationarity

Use ADF and KPSS test totest for stationarity

In [72]:

```
from statsmodels.tsa.stattools import adfuller, kpss

for x in ["Count"]:
    adf_pval = adfuller(tsd[x])[1]
    print(x)
    print(f"ADF, p-value: {adf_pval}")
    kpss_stat, kpss_pval, lags, crit_vals = kpss(trainset['Count'])
    print(f"KPSS, p-value: {kpss_pval}")
```

```
Count
ADF, p-value: 1.2211447254837683e-22
KPSS, p-value: 0.01811627239901272
```

- Based on the p-values of the above tests; the ADF test fails to reject the null of a unit root, and the KPSS test rejects the null of stationarity. The next steps will be done to stationarize the data.

Log Transformation

In [73]:

```
for x in ["Count"]:
    print(x)
    logs = np.log(tsd[x])
    adf_pval = adfuller(logs)[1]
    print(f"ADF, p-value: {adf_pval}")
    kpss_stat, kpss_pval, lags, crit_vals = kpss(logs)
    print(f"KPSS, p-value: {kpss_pval}")
```

```
Count
ADF, p-value: 2.465969207267234e-27
KPSS, p-value: 0.03486597650163498
```

- After log transformation, the ADF still fails to reject the null of a unit root and the KPSS test rejects the null of stationarity. Next step will be to perform differencing

Differencing

In [74]:

```
tsd_diff = tsd.diff()
tsd_diff.head(3)
```

Out[74]:

	Count
datetime	
2018-01-01 00:00:00	NaN
2018-01-01 02:00:00	-4.0
2018-01-01 03:00:00	-4.0

In [75]:

```
#drop the first missing value differencing created
tsd_diff.dropna(inplace=True)
```

In [76]:

```
#conduct adf and kpss test
for x in ["Count"]:
    print(x)
    adf_pval = adfuller(tsd_diff[x])[1]
    print("ADF, p-value:", adf_pval)
    kpss_stat, kpss_pval, lags, crit_vals = kpss(tsd_diff[x])
    print("KPSS, p-value:", kpss_pval)
```

```
Count
ADF, p-value: 0.0
KPSS, p-value: 0.1

/opt/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/stattool
s.py:1710: InterpolationWarning: p-value is greater than the indicat
ed p-value
  warn("p-value is greater than the indicated p-value", Interpolatio
nWarning)
```

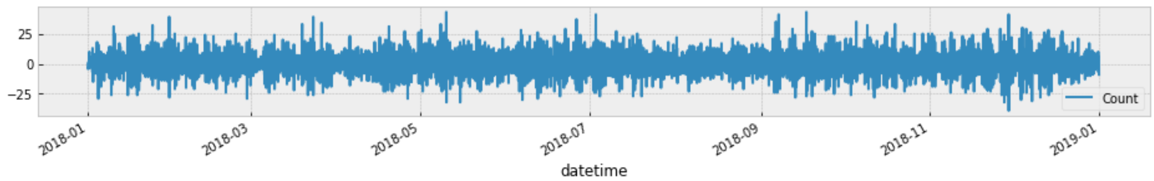
As a result of the first differencing, both tests indicate that the series has become stationary; the ADF rejects the null of unit root and the KPSS test fails to reject the null of stationarity at 0.05 significance level.

In [77]:

```
#plot the differenced data
plt.subplot(311)
tsd_diff['Count'].plot(figsize=(16, 6), legend=True)
```

Out[77]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f916a09bdd0>
```



In [78]:

```
train_diff = tsd_diff
```

- Apply differencing to test data

In [79]:

```
test_diff = test2.diff().dropna()
```

In [80]:

```
testdiff = testset[['Count', 'Urban']].shift().replace(np.NaN, 0.0)
```

Baseline

Will use a persistence baseline for this analysis as it is most common baseline method used for supervised machine learning; and can also be used on time series data.

In [81]:

```
test_diff['Count'].head()
```

Out[81]:

```
datetime
2018-01-01 08:00:00    -1.0
2018-01-01 15:00:00     9.0
2018-01-01 22:00:00   -12.0
2018-01-01 23:00:00     3.0
2018-01-02 05:00:00    -5.0
Name: Count, dtype: float64
```

In [82]:

```
test_diff['Count'].shift().head()
```

Out[82]:

```
datetime
2018-01-01 08:00:00     NaN
2018-01-01 15:00:00    -1.0
2018-01-01 22:00:00     9.0
2018-01-01 23:00:00   -12.0
2018-01-02 05:00:00     3.0
Name: Count, dtype: float64
```

In [83]:

```
mse = mean_squared_error(test_diff['Count'][1:], test_diff['Count'].shift()[1:])
np.sqrt(mse)
```

Out[83]:

```
20.72135517709077
```

In [84]:

```
mda = get_mda(test_diff['Count'][1:], test_diff['Count'].shift()[1:])
mda
```

Out[84]:

```
0.318475073313783
```

ARIMA

Determine the parameters

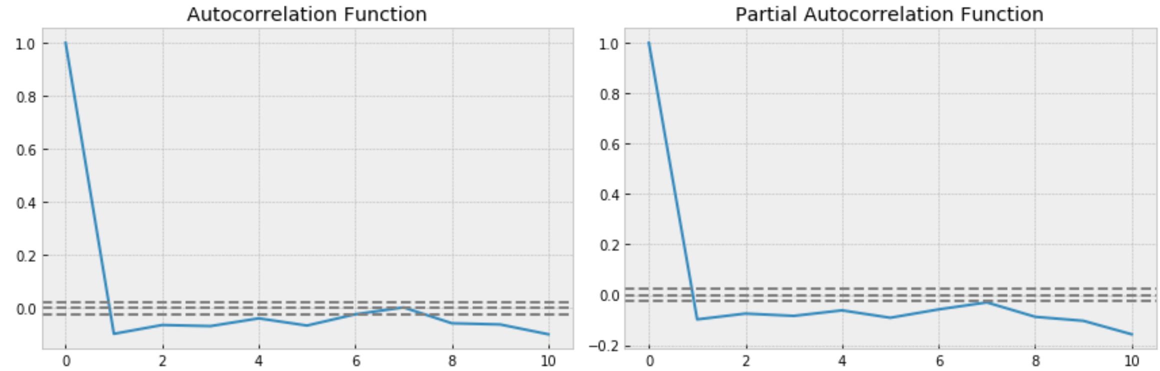
The most common way to determine the p and q parameters is by using ACF and PACF plots.

In [85]:

```
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(train_diff['Count'], nlags=10)
lag_pacf = pacf(train_diff['Count'], nlags=10, method='ols')
```

In [86]:

```
plt.figure(figsize=(12, 4))

plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(train_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(train_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')

plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(train_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(train_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```



From results the ACF and PACF graphs above, set *q* as 1 and *p* as 1. As the lagged correlation is not negative, can conclude the series has not been overdifferenced.

Hyperparameter Tuning

Here the ARIMA hyperparameters will be tuned using a grid search

In [87]:

```
#confirm using code to find best parameters
resDiff = sm.tsa.arma_order_select_ic(train_diff, max_ar=1, max_ma=1, ic='aic',
trend='c')
print('ARMA(p,q) =',resDiff['aic_min_order'],'is the best.')
```

```
ARMA(p,q) = (1, 1) is the best.
```

Therefore ARIMA model order would be (1,0,1) as we are using differenced series

Build ARIMA Model

In [88]:

```
endog = train_diff['Count']
endog.head(3)
```

Out[88]:

```
datetime
2018-01-01 02:00:00   -4.0
2018-01-01 03:00:00   -4.0
2018-01-01 04:00:00   -2.0
Name: Count, dtype: float64
```

In [89]:

```
from statsmodels.tsa.arima_model import ARIMA
```

In [90]:

```
arima = ARIMA(endog, order=(0, 0, 1)).fit(solver="bfgs", disp=0)

# print the significance of the variables
print(arima.summary().tables[1])
```

```
====================================================================
===========
                  coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------
-----------
const          -0.0011      0.087     -0.013      0.990      -0.171
0.169
ma.L1.Count    -0.1170      0.013     -8.779      0.000      -0.143
-0.091
====================================================================
===========

/opt/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_
model.py:219: ValueWarning: A date index has been provided, but it h
as no associated frequency information and so will be ignored when
e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

Evaluate ARIMA on test data

- Retrain new model for each instance of the test data

In [91]:

```
# buffers keeping previously seen endogenous variables
history_endog = [x for x in endog]
predictions = []


# for each test observation, take the first 300 for convenience
for i, test_obs in enumerate(test_diff['Count'][:200]):

    # build a model using the current buffers for endogenous variables
    model = ARIMA(history_endog, order=(0, 0, 1)).fit(solver="bfgs", disp=0)

    # forecast the value for the test instance, supplying corresponding exogenou
s variables
    yhat = model.forecast()[0][0]

    # remember the forecasted value
    predictions.append(yhat)

    # update the buffers for the endogenous variables
    history_endog.append(test_obs)

    print('predicted=%f, expected=%f' % (yhat, test_obs))
```

```
predicted=0.974270, expected=-1.000000
predicted=0.229527, expected=9.000000
predicted=-1.026657, expected=-12.000000
predicted=1.285117, expected=3.000000
predicted=-0.202581, expected=-5.000000
predicted=0.560882, expected=6.000000
predicted=-0.639906, expected=4.000000
predicted=-0.545189, expected=-4.000000
predicted=0.404364, expected=9.000000
predicted=-1.009711, expected=-1.000000
predicted=-0.001218, expected=1.000000
predicted=-0.117548, expected=3.000000
predicted=-0.365711, expected=-7.000000
predicted=0.779157, expected=-9.000000
predicted=1.145724, expected=-2.000000
predicted=0.366670, expected=5.000000
predicted=-0.544973, expected=-7.000000
predicted=0.755350, expected=28.000000
predicted=-3.208414, expected=-10.000000
predicted=0.802281, expected=-6.000000
predicted=0.803712, expected=-3.000000
predicted=0.448362, expected=2.000000
predicted=-0.184201, expected=-7.000000
predicted=0.803924, expected=17.000000
predicted=-1.918527, expected=-2.000000
predicted=0.010010, expected=-3.000000
predicted=0.356694, expected=-8.000000
predicted=0.988712, expected=0.000000
predicted=0.115837, expected=-1.000000
predicted=0.130766, expected=7.000000
predicted=-0.814102, expected=9.000000
predicted=-1.159763, expected=-6.000000
predicted=0.573087, expected=3.000000
predicted=-0.286938, expected=1.000000
predicted=-0.151768, expected=-10.000000
predicted=1.165995, expected=-3.000000
predicted=0.491766, expected=-8.000000
predicted=1.001662, expected=1.000000
predicted=-0.002063, expected=-1.000000
predicted=0.115596, expected=1.000000
predicted=-0.106838, expected=0.000000
predicted=-0.014892, expected=17.000000
predicted=-2.011617, expected=-8.000000
predicted=0.708203, expected=22.000000
predicted=-2.525838, expected=-17.000000
predicted=1.728126, expected=-15.000000
predicted=1.987659, expected=15.000000
predicted=-1.555695, expected=27.000000
predicted=-3.388006, expected=-10.000000
predicted=0.789454, expected=4.000000
predicted=-0.380408, expected=-31.000000
predicted=3.660418, expected=10.000000
predicted=-0.759836, expected=1.000000
predicted=-0.211434, expected=-5.000000
predicted=0.574442, expected=10.000000
predicted=-1.132660, expected=-6.000000
predicted=0.585490, expected=17.000000
predicted=-1.976066, expected=2.000000
predicted=-0.475685, expected=-20.000000
predicted=2.355323, expected=5.000000
predicted=-0.318829, expected=-4.000000
```

```
predicted=0.444474, expected=0.000000
predicted=0.053470, expected=34.000000
predicted=-4.098659, expected=-41.000000
predicted=4.568136, expected=-3.000000
predicted=0.930727, expected=-1.000000
predicted=0.236528, expected=6.000000
predicted=-0.712949, expected=-3.000000
predicted=0.281222, expected=-4.000000
predicted=0.527006, expected=4.000000
predicted=-0.430605, expected=-3.000000
predicted=0.315802, expected=6.000000
predicted=-0.703781, expected=13.000000
predicted=-1.690700, expected=-6.000000
predicted=0.532485, expected=6.000000
predicted=-0.675110, expected=-8.000000
predicted=0.906206, expected=11.000000
predicted=-1.249439, expected=-16.000000
predicted=1.832086, expected=-8.000000
predicted=1.215910, expected=20.000000
predicted=-2.337307, expected=-11.000000
predicted=1.080442, expected=26.000000
predicted=-3.120186, expected=-29.000000
predicted=3.280231, expected=-4.000000
predicted=0.918267, expected=1.000000
predicted=-0.012003, expected=18.000000
predicted=-2.279598, expected=-19.000000
predicted=2.126596, expected=-1.000000
predicted=0.395795, expected=-3.000000
predicted=0.429660, expected=5.000000
predicted=-0.583369, expected=-3.000000
predicted=0.305557, expected=1.000000
predicted=-0.090358, expected=-2.000000
predicted=0.240892, expected=8.000000
predicted=-0.989236, expected=7.000000
predicted=-1.016128, expected=-1.000000
predicted=-0.002265, expected=2.000000
predicted=-0.254594, expected=3.000000
predicted=-0.413392, expected=-14.000000
predicted=1.727986, expected=-6.000000
predicted=0.978865, expected=17.000000
predicted=-2.040987, expected=9.000000
predicted=-1.399984, expected=-22.000000
predicted=2.627645, expected=0.000000
predicted=0.333084, expected=-4.000000
predicted=0.550098, expected=21.000000
predicted=-2.612239, expected=7.000000
predicted=-1.220486, expected=-21.000000
predicted=2.527647, expected=-7.000000
predicted=1.210870, expected=2.000000
predicted=-0.102646, expected=29.000000
predicted=-3.709415, expected=-7.000000
predicted=0.420631, expected=7.000000
predicted=-0.838606, expected=-20.000000
predicted=2.453730, expected=-8.000000
predicted=1.330847, expected=-1.000000
predicted=0.295036, expected=15.000000
predicted=-1.877405, expected=4.000000
predicted=-0.748098, expected=15.000000
predicted=-1.999142, expected=-26.000000
predicted=3.073732, expected=23.000000
predicted=-2.571752, expected=-31.000000
```

```
predicted=3.708271, expected=2.000000
predicted=0.220734, expected=9.000000
predicted=-1.147138, expected=5.000000
predicted=-0.801762, expected=0.000000
predicted=-0.104378, expected=0.000000
predicted=-0.013399, expected=-1.000000
predicted=0.128797, expected=1.000000
predicted=-0.113448, expected=-15.000000
predicted=1.940798, expected=13.000000
predicted=-1.447229, expected=-2.000000
predicted=0.072013, expected=-9.000000
predicted=1.186411, expected=1.000000
predicted=0.022902, expected=-6.000000
predicted=0.786230, expected=26.000000
predicted=-3.309916, expected=-24.000000
predicted=2.738379, expected=26.000000
predicted=-3.106250, expected=11.000000
predicted=-1.869987, expected=-26.000000
predicted=3.230644, expected=-2.000000
predicted=0.697725, expected=8.000000
predicted=-0.976971, expected=12.000000
predicted=-1.732146, expected=-22.000000
predicted=2.721197, expected=-8.000000
predicted=1.431739, expected=13.000000
predicted=-1.552861, expected=15.000000
predicted=-2.213116, expected=0.000000
predicted=-0.293936, expected=-2.000000
predicted=0.229700, expected=-11.000000
predicted=1.501649, expected=-12.000000
predicted=1.797742, expected=27.000000
predicted=-3.380953, expected=-6.000000
predicted=0.352923, expected=13.000000
predicted=-1.697389, expected=-35.000000
predicted=4.510581, expected=8.000000
predicted=-0.473575, expected=-6.000000
predicted=0.749022, expected=-1.000000
predicted=0.235726, expected=-2.000000
predicted=0.301520, expected=9.000000
predicted=-1.182441, expected=-8.000000
predicted=0.925074, expected=6.000000
predicted=-0.691560, expected=-9.000000
predicted=1.128903, expected=2.000000
predicted=-0.120677, expected=5.000000
predicted=-0.698687, expected=14.000000
predicted=-1.998092, expected=12.000000
predicted=-1.892271, expected=-27.000000
predicted=3.422865, expected=-6.000000
predicted=1.276661, expected=0.000000
predicted=0.171050, expected=5.000000
predicted=-0.657949, expected=9.000000
predicted=-1.311879, expected=-2.000000
predicted=0.092800, expected=12.000000
predicted=-1.615931, expected=-24.000000
predicted=3.051851, expected=8.000000
predicted=-0.677168, expected=37.000000
predicted=-5.140886, expected=-35.000000
predicted=4.148415, expected=-5.000000
predicted=1.267164, expected=-3.000000
predicted=0.589842, expected=8.000000
predicted=-1.029960, expected=2.000000
predicted=-0.421251, expected=6.000000
```

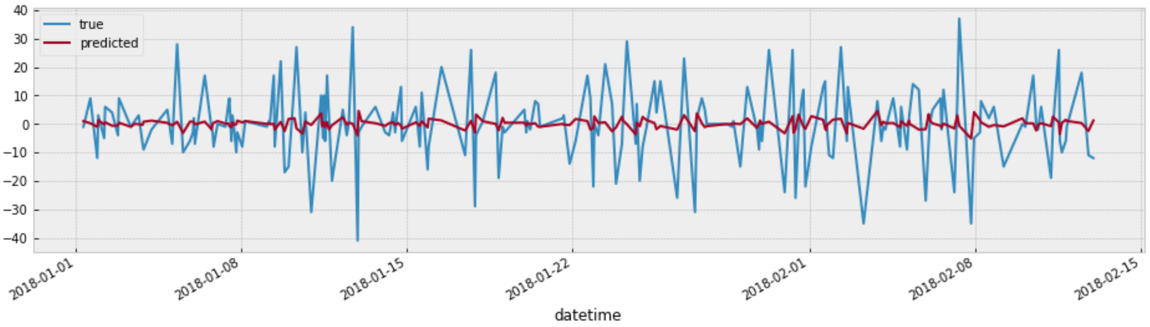
```
predicted=-0.890786, expected=-15.000000
predicted=1.959286, expected=0.000000
predicted=0.270211, expected=-1.000000
predicted=0.174417, expected=17.000000
predicted=-2.338674, expected=0.000000
predicted=-0.324279, expected=-2.000000
predicted=0.232889, expected=6.000000
predicted=-0.800424, expected=-19.000000
predicted=2.531722, expected=-1.000000
predicted=0.488819, expected=26.000000
predicted=-3.555450, expected=-6.000000
predicted=0.341675, expected=-10.000000
predicted=1.443553, expected=-6.000000
predicted=1.036288, expected=-1.000000
predicted=0.282299, expected=18.000000
predicted=-2.471198, expected=-11.000000
predicted=1.191950, expected=-12.000000
```

In [92]:

```
#plot predictions on a graph to compare with observed values
pd.DataFrame({"true": test_diff['Count'][:200], "predicted": predictions}).plot(
figsize=(16, 4))
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f91518dc6d0>
```



- Above is a plot of the predicted values vs the observed models using ARIMA. The predicted values seems to match the peaks and troughs.

In [93]:

```
mse = mean_squared_error(test_diff['Count'][:200], predictions)
np.sqrt(mse)
```

Out[93]:

```
12.520498573775068
```

In [94]:

```
mda = get_mda(test_diff['Count'][:200], predictions)
mda
```

Out[94]:

```
0.7085427135678392
```

Here the ARIMA hyperparameters will be tuned using a grid search

In [95]:

```
from statsmodels.tsa.arima_model import ARIMA
```

Further Models

ARIMAX

Add exogenous variables

In [96]:

```
exog = trainset[['Count', 'Urban']].shift().replace(np.NaN, 0.0)
exog.head(3)
```

Out[96]:

	Count	Urban
datetime		
2018-01-01 00:00:00	0.0	0.0
2018-01-01 02:00:00	18.0	1.0
2018-01-01 03:00:00	14.0	1.0

In [97]:

```
exogd = exog.diff().dropna()
exogd
```

Out[97]:

	Count	Urban
datetime		
2018-01-01 02:00:00	18.0	1.0
2018-01-01 03:00:00	-4.0	0.0
2018-01-01 04:00:00	-4.0	0.0
2018-01-01 05:00:00	-2.0	0.0
2018-01-01 06:00:00	1.0	0.0
...	...	...
2018-12-31 17:00:00	10.0	0.0
2018-12-31 18:00:00	1.0	0.0
2018-12-31 20:00:00	-7.0	0.0
2018-12-31 22:00:00	-2.0	0.0
2018-12-31 23:00:00	6.0	0.0

6831 rows × 2 columns

In [98]:

```
#rename exog
exog = exogd
```

In [99]:

```
endog = train_diff['Count']
endog
```

Out[99]:

```
datetime
2018-01-01 02:00:00   -4.0
2018-01-01 03:00:00   -4.0
2018-01-01 04:00:00   -2.0
2018-01-01 05:00:00    1.0
2018-01-01 06:00:00   -4.0
                       ...
2018-12-31 17:00:00    1.0
2018-12-31 18:00:00   -7.0
2018-12-31 20:00:00   -2.0
2018-12-31 22:00:00    6.0
2018-12-31 23:00:00   -9.0
Name: Count, Length: 6831, dtype: float64
```

Build ARIMAX model

In [100]:

```
from statsmodels.tsa.arima_model import ARIMA

arima = ARIMA(endog, exog=exog, order=(0, 0, 1)).fit(solver="bfgs", disp=0)

# print the significance of the variables
print(arima.summary().tables[1])
```

```
/opt/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_
model.py:219: ValueWarning: A date index has been provided, but it h
as no associated frequency information and so will be ignored when
e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)

====================================================================
===========
                  coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------
-----------
const        5.996e-05   4.61e-05      1.301      0.193   -3.04e-05
0.000
Count           0.7021      0.009     77.038      0.000       0.684
0.720
Urban          -0.4617      0.243     -1.902      0.057      -0.937
0.014
ma.L1.Count    -1.0000      0.000  -2149.492      0.000      -1.001
-0.999
====================================================================
===========
```

In [ ]:

```

```

Evaluate model on test data

In [101]:

```
# buffers keeping previously seen endogenous and exogenous variables
history_endog = [x for x in endog]
history_exog = [x for x in exog.values]
predictions = []

# create exogenous variables also for the test data
test_exog = testdiff[['Count', 'Urban']].shift().replace(np.NaN, 0.0)

# for each test observation, take the next 300 for convenience
for i, test_obs in enumerate(test_diff['Count'][200:400]):

    # build a model using the current buffers for endogenous and exogenous varia
bles
    model = ARIMA(history_endog, exog=history_exog, order=(0, 0, 1)).fit(solver=
"bfgs", disp=0)

    # forecast the value for the test instance, supplying corresponding exogenou
s variables
    yhat = model.forecast(exog=test_exog.iloc[i])[0][0]

    # remember the forecasted value
    predictions.append(yhat)

    # update the buffers for the endogenous and exogenous variables
    history_endog.append(test_obs)
    history_exog.append(test_exog.iloc[i])

    print('predicted=%f, expected=%f' % (yhat, test_obs))
```

```
predicted=8.049571, expected=1.000000
predicted=7.045170, expected=14.000000
predicted=-1.095469, expected=2.000000
predicted=2.527754, expected=-16.000000
predicted=29.947982, expected=-2.000000
predicted=34.744400, expected=27.000000
predicted=12.712235, expected=-16.000000
predicted=29.674745, expected=23.000000
predicted=12.118860, expected=8.000000
predicted=12.369147, expected=-29.000000
predicted=44.930130, expected=-10.000000
predicted=63.297150, expected=11.000000
predicted=59.220923, expected=6.000000
predicted=59.278726, expected=-1.000000
predicted=67.233368, expected=-16.000000
predicted=82.276874, expected=0.000000
predicted=74.371741, expected=5.000000
predicted=60.779669, expected=26.000000
predicted=32.726741, expected=-10.000000
predicted=39.015032, expected=-15.000000
predicted=63.922744, expected=0.000000
predicted=65.686789, expected=-2.000000
predicted=64.652376, expected=-6.000000
predicted=64.983977, expected=5.000000
predicted=56.406325, expected=23.000000
predicted=27.717255, expected=-12.000000
predicted=44.362293, expected=-5.000000
predicted=24.860091, expected=0.000000
predicted=20.205686, expected=-2.000000
predicted=14.317995, expected=8.000000
predicted=6.451927, expected=-5.000000
predicted=9.048683, expected=-10.000000
predicted=14.851674, expected=4.000000
predicted=13.449953, expected=6.000000
predicted=9.409727, expected=5.000000
predicted=8.971273, expected=-7.000000
predicted=14.045981, expected=-6.000000
predicted=11.582802, expected=-2.000000
predicted=7.628140, expected=1.000000
predicted=2.850881, expected=12.000000
predicted=-3.178514, expected=1.000000
predicted=-1.437983, expected=4.000000
predicted=-1.673059, expected=-20.000000
predicted=8.167300, expected=-1.000000
predicted=8.840056, expected=3.000000
predicted=5.188164, expected=30.000000
predicted=-1.120441, expected=-24.000000
predicted=13.837212, expected=21.000000
predicted=-2.671490, expected=-18.000000
predicted=10.856836, expected=19.000000
predicted=8.675168, expected=-30.000000
predicted=22.276658, expected=9.000000
predicted=11.951569, expected=4.000000
predicted=3.570007, expected=17.000000
predicted=-1.132803, expected=-10.000000
predicted=6.780990, expected=2.000000
predicted=4.116245, expected=-7.000000
predicted=8.282302, expected=1.000000
predicted=5.588207, expected=0.000000
predicted=8.211904, expected=-7.000000
predicted=10.851715, expected=1.000000
```

```
predicted=5.326752, expected=-4.000000
predicted=5.900269, expected=-7.000000
predicted=6.011125, expected=17.000000
predicted=-0.497053, expected=17.000000
predicted=2.901083, expected=-28.000000
predicted=9.744776, expected=-1.000000
predicted=3.651410, expected=14.000000
predicted=-1.896352, expected=-15.000000
predicted=5.188745, expected=-4.000000
predicted=3.627529, expected=35.000000
predicted=-7.994481, expected=-20.000000
predicted=4.572872, expected=2.000000
predicted=1.629642, expected=-15.000000
predicted=6.459738, expected=29.000000
predicted=-2.723809, expected=-14.000000
predicted=6.255069, expected=-5.000000
predicted=6.934033, expected=-5.000000
predicted=5.650863, expected=-7.000000
predicted=7.090275, expected=27.000000
predicted=-3.779565, expected=-5.000000
predicted=0.932110, expected=-2.000000
predicted=4.379474, expected=-9.000000
predicted=5.611193, expected=-6.000000
predicted=8.522218, expected=-1.000000
predicted=3.742516, expected=9.000000
predicted=-0.774481, expected=1.000000
predicted=0.281647, expected=1.000000
predicted=3.111240, expected=-10.000000
predicted=4.262043, expected=-3.000000
predicted=2.552074, expected=8.000000
predicted=-1.341889, expected=8.000000
predicted=-1.572691, expected=-2.000000
predicted=0.646024, expected=-12.000000
predicted=4.008323, expected=-4.000000
predicted=2.324386, expected=0.000000
predicted=1.818305, expected=5.000000
predicted=1.416685, expected=6.000000
predicted=0.936692, expected=-3.000000
predicted=3.255455, expected=-1.000000
predicted=3.831058, expected=0.000000
predicted=1.977145, expected=3.000000
predicted=-0.000152, expected=1.000000
predicted=2.070849, expected=-8.000000
predicted=5.953171, expected=24.000000
predicted=-3.981675, expected=-16.000000
predicted=4.027551, expected=0.000000
predicted=1.187689, expected=-11.000000
predicted=6.215612, expected=3.000000
predicted=4.640704, expected=-2.000000
predicted=2.817852, expected=24.000000
predicted=-5.588319, expected=-25.000000
predicted=5.838106, expected=3.000000
predicted=5.144255, expected=11.000000
predicted=1.902667, expected=-5.000000
predicted=6.264333, expected=10.000000
predicted=0.714582, expected=-2.000000
predicted=1.387055, expected=-17.000000
predicted=5.535654, expected=1.000000
predicted=3.695698, expected=5.000000
predicted=2.646542, expected=-6.000000
predicted=7.221542, expected=3.000000
```

```
predicted=2.591847, expected=22.000000
predicted=-0.671676, expected=-22.000000
predicted=6.111219, expected=5.000000
predicted=1.025851, expected=-4.000000
predicted=3.229125, expected=6.000000
predicted=1.777474, expected=-4.000000
predicted=4.051500, expected=0.000000
predicted=3.554556, expected=10.000000
predicted=0.658965, expected=-14.000000
predicted=6.376260, expected=17.000000
predicted=-2.261599, expected=-5.000000
predicted=2.913733, expected=-1.000000
predicted=3.064173, expected=-7.000000
predicted=3.528791, expected=-6.000000
predicted=3.475846, expected=7.000000
predicted=-0.851155, expected=13.000000
predicted=-0.251717, expected=23.000000
predicted=-5.800708, expected=-29.000000
predicted=10.301352, expected=2.000000
predicted=7.459642, expected=13.000000
predicted=0.426673, expected=-3.000000
predicted=2.571195, expected=-19.000000
predicted=8.446308, expected=10.000000
predicted=3.755651, expected=4.000000
predicted=1.263788, expected=2.000000
predicted=0.020201, expected=-2.000000
predicted=2.526459, expected=-19.000000
predicted=9.598170, expected=-1.000000
predicted=6.374635, expected=29.000000
predicted=-2.355041, expected=-10.000000
predicted=4.330081, expected=-11.000000
predicted=4.716641, expected=-8.000000
predicted=7.361235, expected=3.000000
predicted=4.330854, expected=4.000000
predicted=4.868279, expected=3.000000
predicted=1.001749, expected=2.000000
predicted=1.225399, expected=-10.000000
predicted=3.695615, expected=4.000000
predicted=0.549966, expected=-4.000000
predicted=1.579874, expected=7.000000
predicted=0.047254, expected=-10.000000
predicted=3.139583, expected=0.000000
predicted=2.037813, expected=0.000000
predicted=0.672286, expected=6.000000
predicted=-1.013910, expected=18.000000
predicted=-3.967804, expected=9.000000
predicted=-0.713985, expected=-21.000000
predicted=9.253132, expected=-5.000000
predicted=4.331296, expected=7.000000
predicted=-0.598177, expected=17.000000
predicted=-4.295485, expected=-30.000000
predicted=7.295338, expected=9.000000
predicted=1.328814, expected=5.000000
predicted=0.589366, expected=-6.000000
predicted=4.640042, expected=-8.000000
predicted=3.283968, expected=-1.000000
predicted=2.125759, expected=15.000000
predicted=1.955681, expected=2.000000
predicted=1.269640, expected=11.000000
predicted=-1.788685, expected=-14.000000
predicted=3.532327, expected=-4.000000
```

```
predicted=3.224179, expected=-8.000000
predicted=4.355228, expected=2.000000
predicted=2.753483, expected=1.000000
predicted=0.908368, expected=22.000000
predicted=-5.008030, expected=-21.000000
predicted=4.510716, expected=-4.000000
predicted=4.513868, expected=0.000000
predicted=3.447257, expected=16.000000
predicted=-1.142146, expected=-1.000000
predicted=2.778545, expected=-1.000000
predicted=1.515870, expected=-13.000000
predicted=4.161721, expected=8.000000
predicted=2.474644, expected=-2.000000
predicted=3.901164, expected=15.000000
predicted=-1.231352, expected=-9.000000
predicted=2.963568, expected=0.000000
predicted=1.587609, expected=-11.000000
```

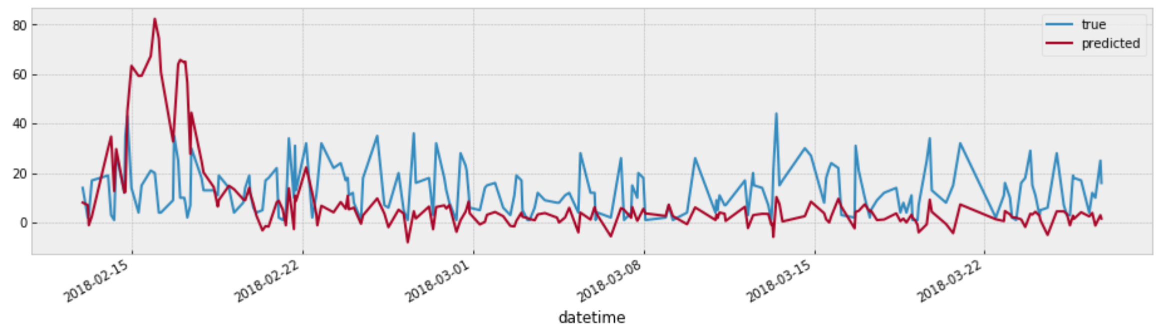
- Plot predictions on a graph to compare with the observed values

In [102]:

```
pd.DataFrame({"true": testdiff['Count'][200:400], "predicted": predictions}).plo
t(figsize=(16, 4))
```

Out[102]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9151988310>
```



- The predicted values from the ARIMAX model matches the peaks and troughs more closely than the previous ARIMA model created. We can see that the sudden spikes are also predicted by the model.

In [103]:

```
mse = mean_squared_error(testdiff['Count'][200:400], predictions)
np.sqrt(mse)
```

Out[103]:

```
18.10303545197831
```

In [104]:

```
mda = get_mda(testdiff['Count'][200:400], predictions)
mda
```

Out[104]:

```
0.6733668341708543
```

The numerical measures of accuracy, RMSE score and the MDA score

Model Evaluation

RMSE Results

In [105]:

```
rmse_table = pd.DataFrame({
    'method': ['Persistence baseline', 'ARIMA', 'ARIMAX'],
    'RMSE': [20.72135517709077, 12.520498573775068, 18.10303545197831],
})

rmse_table.set_index("method", inplace=True)

# add columns with percent changes on the baselines
rmse_table['% Change on Pers. baseline'] = 100 * rmse_table['RMSE'] / 20.7213551
7709077 - 100

rmse_table
```

Out[105]:

method	RMSE	% Change on Pers. baseline
Persistence baseline	20.721355	0.000000
ARIMA	12.520499	-39.576835
ARIMAX	18.103035	-12.635852

The table above shows the root mean square error for the previously calculated baseline, ARIMA model and ARIMAX model. It shows that the ARIMA model produced the best rates: 39% reduction on the persistence baseline.

MDA Results

In [106]:

```
mda_table = pd.DataFrame({
    'method': ['Persistence baseline', 'ARIMA', 'ARIMAX'],
    'MDA': [0.318475073313783, 0.7085427135678392, 0.6733668341708543 ]
})

mda_table.set_index("method", inplace=True)

# add columns with percent changes on the baselines
mda_table['% Change on Pers. baseline, MDA'] = 100 * mda_table['MDA'] / 0.318475
073313783 - 100
mda_table
```

Out[106]:

	MDA	% Change on Pers. baseline, MDA
method		
Persistence baseline	0.318475	0.000000
ARIMA	0.708543	122.479802
ARIMAX	0.673367	111.434706

In terms of MDA, ARIMA model also produced the better results with the MDA increase of 122% on the persistence baseline

Conclusion

Time series models; ARIMA and ARIMAX were used to forecast the number of accidents to help a road assistance company effectively allocate their resources to meet demand. From the EDA, it was shown that the number of accidents were higher between 7:30am - 8:30 and 3:30pm-6:30pm from Monday to Friday. It was said that this coult be explained by the rush hour/peak time travel.

The ARIMAX model built included the exogenous variable `Urban` determining the effect the type of area had on the number of accidents that occurred.

Both models were used to predict instances of the test data where the ARIMAX model was able to match the peaks and troughs in the data much closer than the ARIMA model.

It can be said that as the predicted values matched the observed values, it can be said that the road assistance would be recommended to:

- Increase the number of resources allocated during the hours of rush hour/peak times
- Have a certain number of resources on standby for the remaining quieter hours of the day

Possible further improvements

- Though understanding that it is difficult to predict human error, would try more methods where time series models and other traditional ML models could be used simultaneously for better performance
- Using new data from most recent years, using it as new test set
- Take into consideration the effect of other variables on the number of accidents
- Determine whether holiday periods such as Christmas have an effect on the number of accidents

In [ ]:

```

```