Task-4A:

In [311]:
```python
import pandas as pd
counts = pd.read_csv('Fremont_Bridge_Bicycle_Counter.csv', index_col='Date', parse_dates=True)
weather = pd.read_csv('weather.csv', index_col='DATE', parse_dates=True)
```

In [312]:
```python
daily = counts.resample('d').sum()
daily['Total'] = daily.sum(axis=1)
daily = daily[['Total']]   # remove other columns
```

In [313]:
```python
days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
for i in range(7):
    daily[days[i]] = (daily.index.dayofweek == i).astype(float)
```

In [314]:
```python
from pandas.tseries.holiday import USFederalHolidayCalendar
cal = USFederalHolidayCalendar()
holidays = cal.holidays('2012', '2016')
daily = daily.join(pd.Series(1, index=holidays, name='holiday'))
daily['holiday'].fillna(0, inplace=True)
```

In [315]:
```python
def hours_of_daylight(date, axis=23.44, latitude=47.61):
    """Compute the hours of daylight for the given date"""
    days = (date - pd.datetime(2000, 12, 21)).days
    m = (1. - np.tan(np.radians(latitude))
         * np.tan(np.radians(axis) * np.cos(days * 2 * np.pi / 365.25)))
    return 24. * np.degrees(np.arccos(1 - np.clip(m, 0, 2))) / 180.

daily['daylight_hrs'] = list(map(hours_of_daylight, daily.index))
```

```
C:\Users\NERIMAN.GURSOY\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: The pandas.datetime class is deprec
ated and will be removed from pandas in a future version. Import from datetime module instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [316]:
```python
weather['TMIN'] /= 10
weather['TMAX'] /= 10
weather['Temp (C)'] = 0.5 * (weather['TMIN'] + weather['TMAX'])

daily = daily.join(weather[['Temp (C)']])
```

In [317]:
```python
daily['annual'] = (daily.index - daily.index[0]).days / 365.
daily.dropna(axis=0, how='any', inplace=True)
```


In [316]:
```python
weather['TMIN'] /= 10
weather['TMAX'] /= 10
weather['Temp (C)'] = 0.5 * (weather['TMIN'] + weather['TMAX'])

daily = daily.join(weather[['Temp (C)']])
```

In [317]:
```python
daily['annual'] = (daily.index - daily.index[0]).days / 365.
daily.dropna(axis=0, how='any', inplace=True)
```


In [318]:
```python
daily.head()
```

Out[318]:

Date	Total	Mon	Tue	Wed	Thu	Fri	Sat	Sun	holiday	daylight_hrs	Temp (C)	annual
2019-11-01	15048.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	9.714047	4.65	7.082192
2019-11-02	7856.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	9.663639	4.90	7.084932
2019-11-03	6940.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	9.613729	4.85	7.087671
2019-11-04	18384.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9.564338	4.95	7.090411
2019-11-05	18216.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	9.515483	4.85	7.093151

In [326]:
```python
X = daily.drop(["Total"], axis=1)
y = daily['Total']
```

In [326]:
```
X = daily.drop(["Total"], axis=1)
y = daily['Total']
```

In [327]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 5)  #5th degree
X_poly = poly_reg.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
```

Out[327]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [328]:
```
print("Model slope:     ", lin_reg.coef_[0])
print("Model intercept:", lin_reg.intercept_)
```

```
Model slope:      12560030810.384565
Model intercept: 241080865376.0259
```

In [330]:
```
y_pred = lin_reg.predict(X_poly)
```

In [331]:
```
df = pd.DataFrame({'True Values':y, 'Predicted Values':y_pred})
df.head()
```

Out[331]:

Date	True Values	Predicted Values
2019-11-01	15048.0	14847.838409
2019-11-02	7856.0	7576.518097
2019-11-03	6940.0	7178.846222
2019-11-04	18384.0	18556.127472
2019-11-05	18216.0	18037.182159

In [336]:
```
mse = np.mean((y - y_pred)**2)  #mean square error

mse
```

Out[336]:
```
1473987.3693042286
```

Ridge Regression

In [341]:
```
from sklearn.linear_model import Ridge

ridgeReg = Ridge(alpha=0.1, normalize=True)
ridgeReg.fit(X_poly, y)
pred = ridgeReg.predict(X_poly)
```

In [344]:
```
##calculating mse

mse = np.mean((y - pred)**2)
mse
```

Out[344]:
```
5780370.881733926
```

Lasso Regression

In [345]:
```
from sklearn.linear_model import Lasso

lassoReg = Lasso(alpha=0.1, normalize=True)
lassoReg.fit(X_poly, y)

pred = lassoReg.predict(X_poly)
```

```
C:\Users\NERIMAN.GURSOY\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: Object
ive did not converge. You might want to increase the number of iterations. Duality gap: 342098755.79741263, tolerance: 732914.9
16744856
  positive)
```

In [346]:
```
##calculating mse

mse = np.mean((y - pred)**2)
mse
```

Out[346]:
```
4272129.386137333
```

Regularization is to reduce the possibility of overfitting of the model by revising the coefficients of the developed model.

The regression models used to revise the coefficients of the model are ridge and lasso regression.

Ridge regression tries to reduce the coefficient values by taking the sum of squares and imposes various penalties on the coefficients of the model and aims to reduce the values in this way. It tries to draw the coefficient values to a certain range. We can say that it is closer to zero.

```
In [347]: print("Model slope:    ", ridgeReg.coef_[0])
          print("Model intercept:", ridgeReg.intercept_)

          Model slope:     0.0
          Model intercept: 25206.922974283538
```

Lasso regression performs regularization over absolute values. The main purpose here is to reset some values directly and exclude them from the model, thus reducing the possibility of the model's multicollinearity.

```
In [348]: print("Model slope:    ", lassoReg.coef_[0])
          print("Model intercept:", lassoReg.intercept_)

          Model slope:     0.0
          Model intercept: 114148.51568024362
```

Task-4B:

Using breast cancer dataset, a simple Gaussian Naive Bayes model was built, and it was tried to predict whether patients are cancer or not. According to the outputs of the model, precision, recall, accuracy and f1-score values appear in the classification report.

```
In [73]: from sklearn.datasets import load_breast_cancer
         cancer = load_breast_cancer()
```

```
In [74]: X = cancer.data
         X.shape

Out[74]: (569, 30)
```

```
In [75]: y = cancer.target
         y.shape

Out[75]: (569,)
```

```
In [76]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
```

```
In [85]: from sklearn.naive_bayes import GaussianNB
         model = GaussianNB()
         model.fit(X_train, y_train)
         predict = model.predict(X_test)
```

```
In [87]: from sklearn.metrics import classification_report, accuracy_score
         report = classification_report(y_test, predict)
         score = accuracy_score(y_true= y_test, y_pred=predict)
         print(report)
         print("{} {:0.2f}%".format("Accuracy Score : ", score*100))

              precision    recall  f1-score   support

           0       0.96      0.92      0.94        48
           1       0.94      0.97      0.96        66

    accuracy                           0.95       114
   macro avg       0.95      0.94      0.95       114
weighted avg       0.95      0.95      0.95       114

Accuracy Score :  94.74%
```

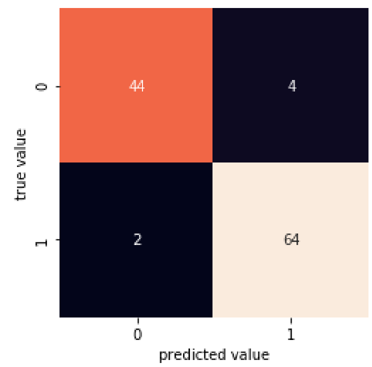
Accuracy: According to the above report, the accuracy value seems to be about 95 percent. The Accuracy value is the ratio of correct predictions to the total number of predictions. for this example, the ratio of correctly predicted cancer patients to the total number of patients.

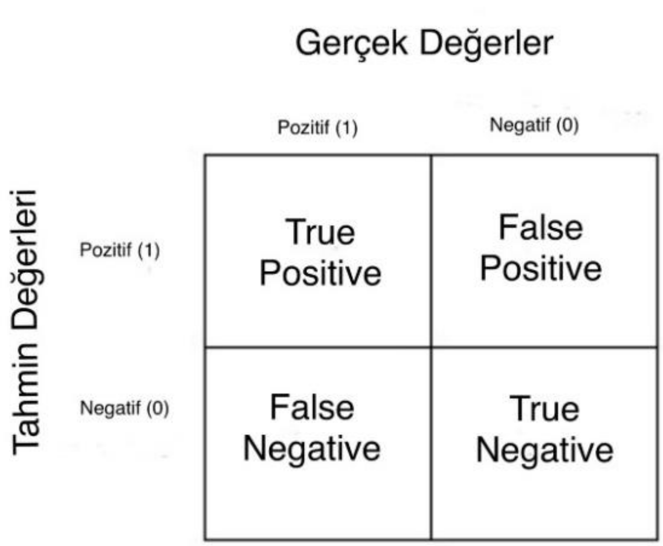
$(44 + 64) / (44 + 64 + 4 + 2) = 0.9473$

In [90]:

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat = confusion_matrix(y_test, predict)

sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value');
```





When we look at the confusion matrix diagonally, the sum of 64 + 44 actually shows us the number of patients that the model predicted correctly in total.

TP + TN / (TP + TN + FP + FN)

Precision: Again, when looking at the precision value for a value of 1 that is labeled as a cancer patient in the report, it is seen as 0.94.

This value indicates what percentage of people the model labels as cancer patients actually are cancer?

(TP / (TP + FP))

At confusion matrix:

64 / 64+4 = 0.94

Precision value for a value of 0 that is not labeled as a cancer patient is 0.96.   (44 / 46 = 0.96)

Recall: What percentage of people with cancer actually labeled the model correctly? What percentage did it fail to catch? gives the answer to the question.

(TP / TP + FN))

For label 1:   64 / (64 + 2) = 0.97

The model correctly labeled 64 out of 66 patients in total. 97% predicted it right. 3 percent could not predict correctly. Although they have cancer, they have been given a label, not cancer.

For label 0 : 44 / (44+4) = 0.92 . The model correctly predicted 44 of 48 patients who did not really have cancer, saying that it is not cancer. But he made the wrong predict for 4 patients.

F1 – Score: This score generates a balance score by averaging the precision and recall scores to avoid confusion. It can be thought of as the harmonic mean of precision and recall metrics. This value is always between the values of recall and precision.

F1-score = 2 * Precision * Recall / (Precision + Recall)

For label 0: f1-score is 0.94.

2 * 0.96 * 0.92 / (0.96 + 0.92) = 0.94.

Task-4C:

Using the Grid Search method in the Digits dataset, I created two SVM models with different parameters.

First model

```
In [125]: from sklearn.datasets import load_digits
          digits = load_digits()
```

```
In [126]: X = digits.data
          X.shape
```

```
Out[126]: (1797, 64)
```

```
In [127]: y = digits.target
          y.shape
```

```
Out[127]: (1797,)
```

```
In [165]: from sklearn.model_selection import train_test_split
          Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [166]: from sklearn.svm import SVC
          from sklearn.pipeline import make_pipeline

          svc = SVC(kernel='rbf')
          model = make_pipeline(svc)
```

```
In [167]: from sklearn.model_selection import GridSearchCV

          param_grid= {'svc__C': [1, 5, 10, 50],
                       'svc__gamma':[0.0001, 0.0005, 0.001, 0.005]}
          grid = GridSearchCV(model, param_grid, cv=5) #CV=5

          grid.fit(Xtrain, ytrain)
          print(grid.best_params_)

          {'svc__C': 10, 'svc__gamma': 0.0005}
```

```
In [168]: model = grid.best_estimator_
          y_predict = model.predict(Xtest)
```

```
In [169]: from sklearn.metrics import accuracy_score
          score = accuracy_score(ytest, y_predict)
          print("{} {:0.2f}%".format("Accuracy Score : ", score*100))

          Accuracy Score :  98.89%
```

For the first model c value is 10, gamma 0.00005 and CV = 5.

the accuracy of the model is 98.89%.

Second model:

```
In [170]: param_grid= {'svc__C': [0.001, 0.1, 100, 10e5],
                        'svc__gamma':[10,1,0.1,0.01]}
          grid = GridSearchCV(model, param_grid, cv=5) #CV=5

          grid.fit(Xtrain, ytrain)
          print(grid.best_params_)

          {'svc__C': 100, 'svc__gamma': 0.01}

In [171]: model = grid.best_estimator_
          y_predict = model.predict(Xtest)

In [172]: from sklearn.metrics import accuracy_score
          score = accuracy_score(ytest, y_predict)
          print("{} {:0.2f}%".format("Accuracy Score : ", score*100))

          Accuracy Score :   81.39%
```

For the second model c value is 100 gamma 0.01 and CV = 5.

the accuracy of the model is 81.39%.

One of the most critical parameters in models created using support vector algorithms is C, the penalty coefficient. There is a penalty point for every wrong predictions.

Higher C value indicates narrower margins. This actually prevents them from entering values in those ranges. In the second model c value is 100 and the prediction success rate of the model is 81.39%.

When the value of C parameter is low, it behaves more flexible and the margins are wider. this shows that he actually gives more tolerance to punishment. Because the larger the margins, the more value can fall in that range. In the first model, c value is 10 and the accuracy value is 98.89%.

In fact, in this model, c is lower and the error limit is higher because it shows tolerance. Therefore, we can say as if he memorized the model dataset, it was trained very well, and its accuracy score was very high.

Task-4D:

Decision tree model was created using Iris data set. Accuracy value is 0.92.

Task 4D: Please use the same dataset in Task 4C and this time test Decision Tree algorithms again with the grid search algorithm to test different values.

In [237]:
```
from  sklearn import  datasets
iris=datasets.load_iris()
```

In [238]:
```
X=iris.data
y=iris.target
```

In [239]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=.5)
```

In [240]:
```
from sklearn import tree
classifier = tree.DecisionTreeClassifier()
```

In [241]:
```
from sklearn import neighbors
classifier=neighbors.KNeighborsClassifier()
```

In [242]:
```
print(classifier.get_params())
```

```
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, 'p': 2,
'weights': 'uniform'}
```

In [243]:
```
classifier.fit(X_train,y_train)
```

Out[243]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [244]:
```
predictions=classifier.predict(X_test)
```

In [245]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,predictions))
```

```
0.92
```

In [246]:
```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=300, centers=4,
                  random_state=0, cluster_std=1.0)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='rainbow');
```