K-Markov Sampling for Pascal dataset

```
# Import all dependencies
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import scale
```

```
# mount drive for easy import and export of data
from google.colab import drive
drive.mount('/content/drive')
```

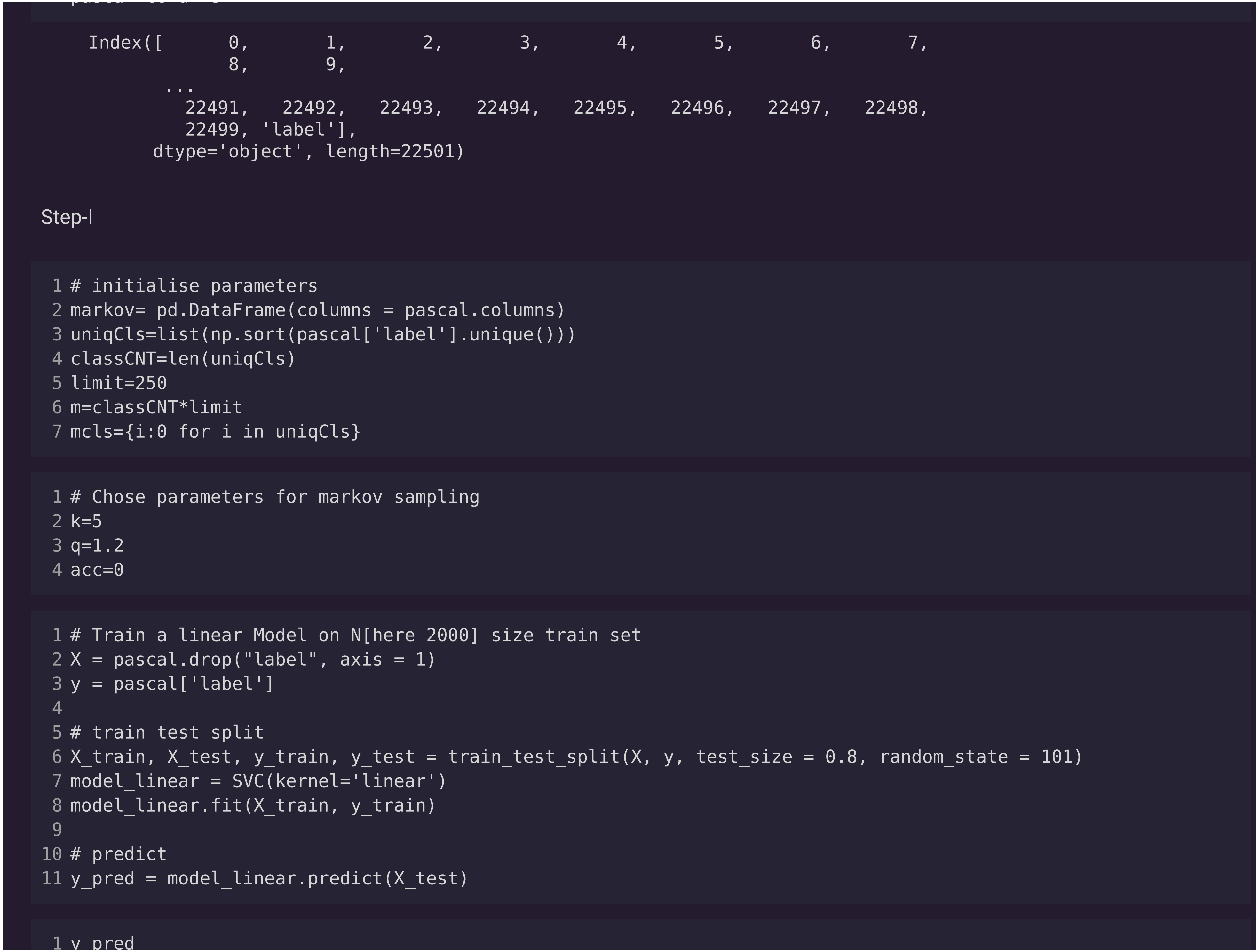
```
Mounted at /content/drive
```

```
# initialise dataframe with letter dataset
pascal = pd.read_csv("/content/drive/MyDrive/DM/Image-pixels.csv")
```

```
pascal.shape
```

```
(4382, 22501)
```

```
col=[i for i in range(22500)]
col.append('label')
pascal.columns=col
pascal.columns
```



```
Index([      0,       1,       2,       3,       4,       5,       6,       7,
             8,       9,
       ...
         22491,   22492,   22493,   22494,   22495,   22496,   22497,   22498,
         22499, 'label'],
      dtype='object', length=22501)
```

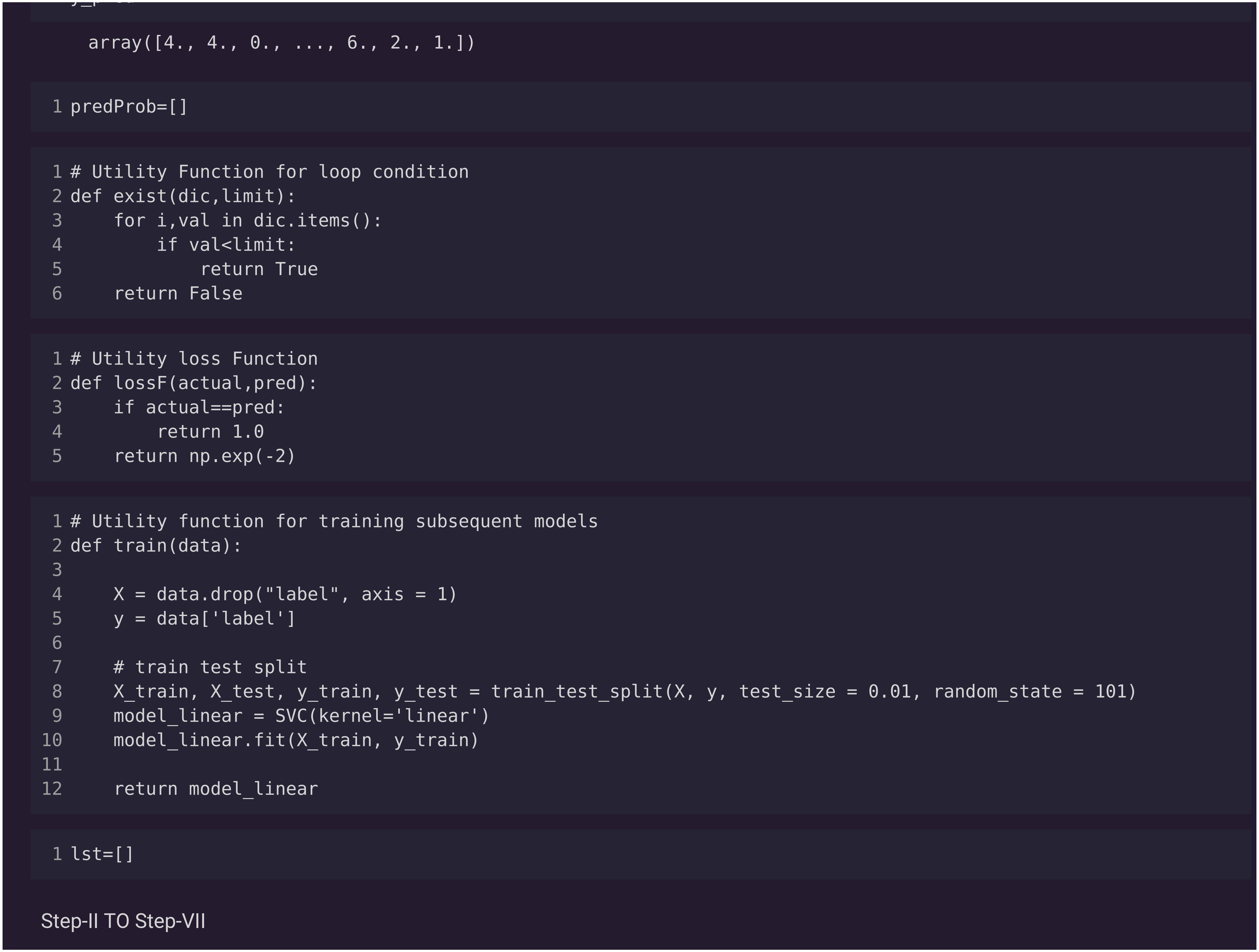
Step-I

```
# initialise parameters
markov= pd.DataFrame(columns = pascal.columns)
uniqCls=list(np.sort(pascal['label'].unique()))
classCNT=len(uniqCls)
limit=250
m=classCNT*limit
mcls={i:0 for i in uniqCls}
```

```
# Chose parameters for markov sampling
k=5
q=1.2
acc=0
```

```
# Train a linear Model on N[here 2000] size train set
X = pascal.drop("label", axis = 1)
y = pascal['label']

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.8, random_state = 101)
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
```

```
y_pred
```



```
y_pred
```

```
array([4., 4., 0., ..., 6., 2., 1.])
```

```
predProb=[]
```

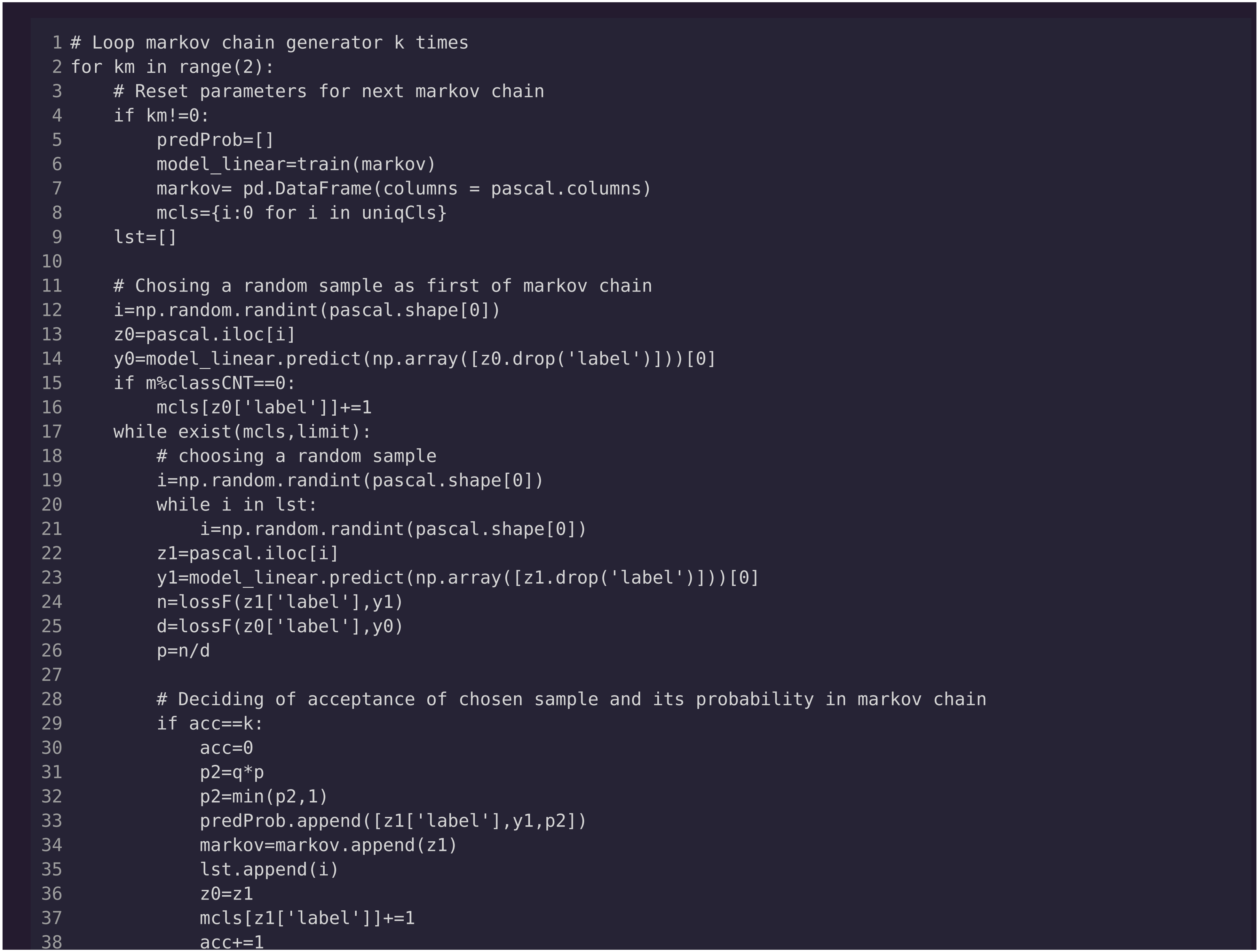
```
# Utility Function for loop condition
def exist(dic,limit):
    for i,val in dic.items():
        if val<limit:
            return True
    return False
```

```
# Utility loss Function
def lossF(actual,pred):
    if actual==pred:
        return 1.0
    return np.exp(-2)
```

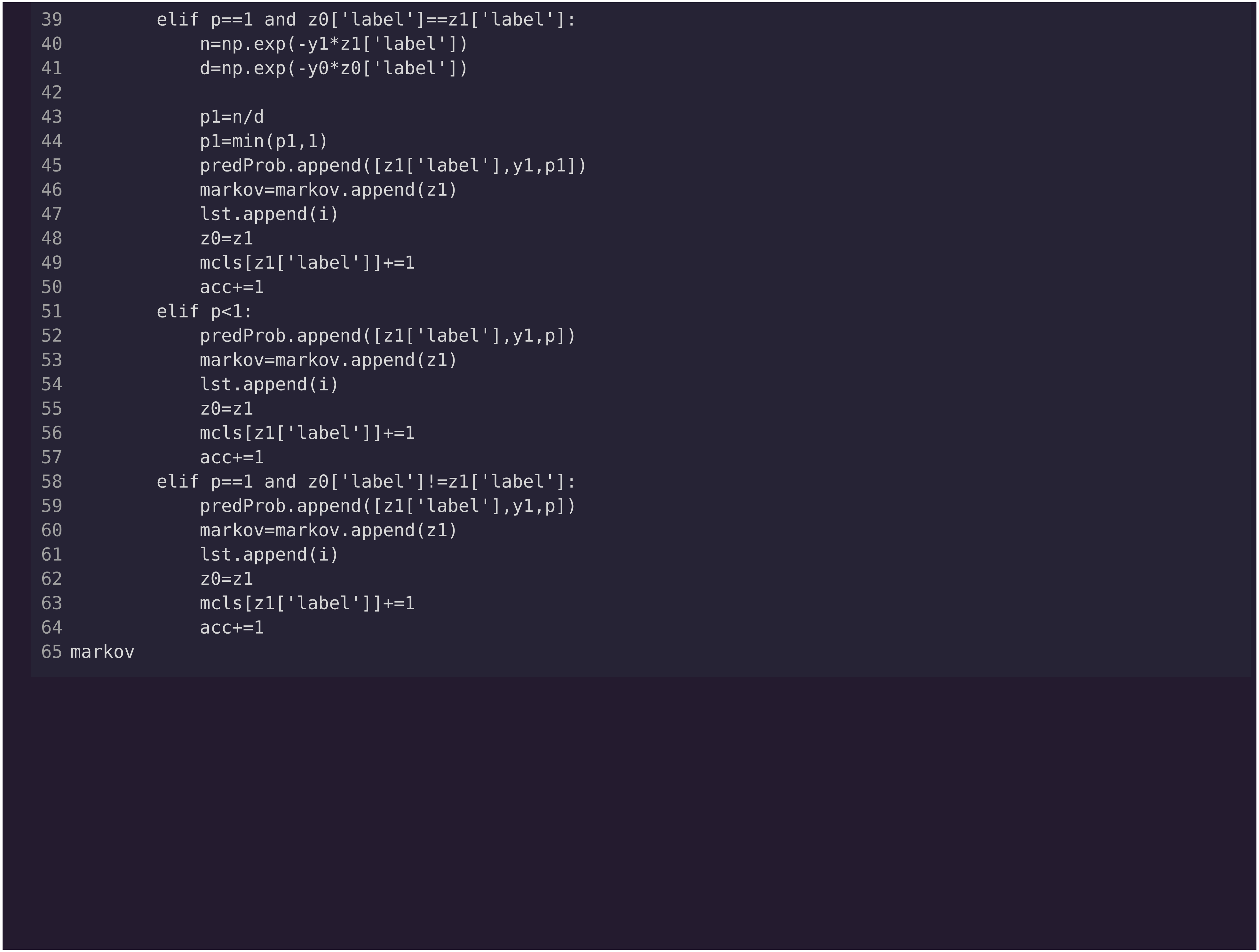
```
# Utility function for training subsequent models
def train(data):

    X = data.drop("label", axis = 1)
    y = data['label']

    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.01, random_state = 101)
    model_linear = SVC(kernel='linear')
    model_linear.fit(X_train, y_train)

    return model_linear
```

```
lst=[]
```

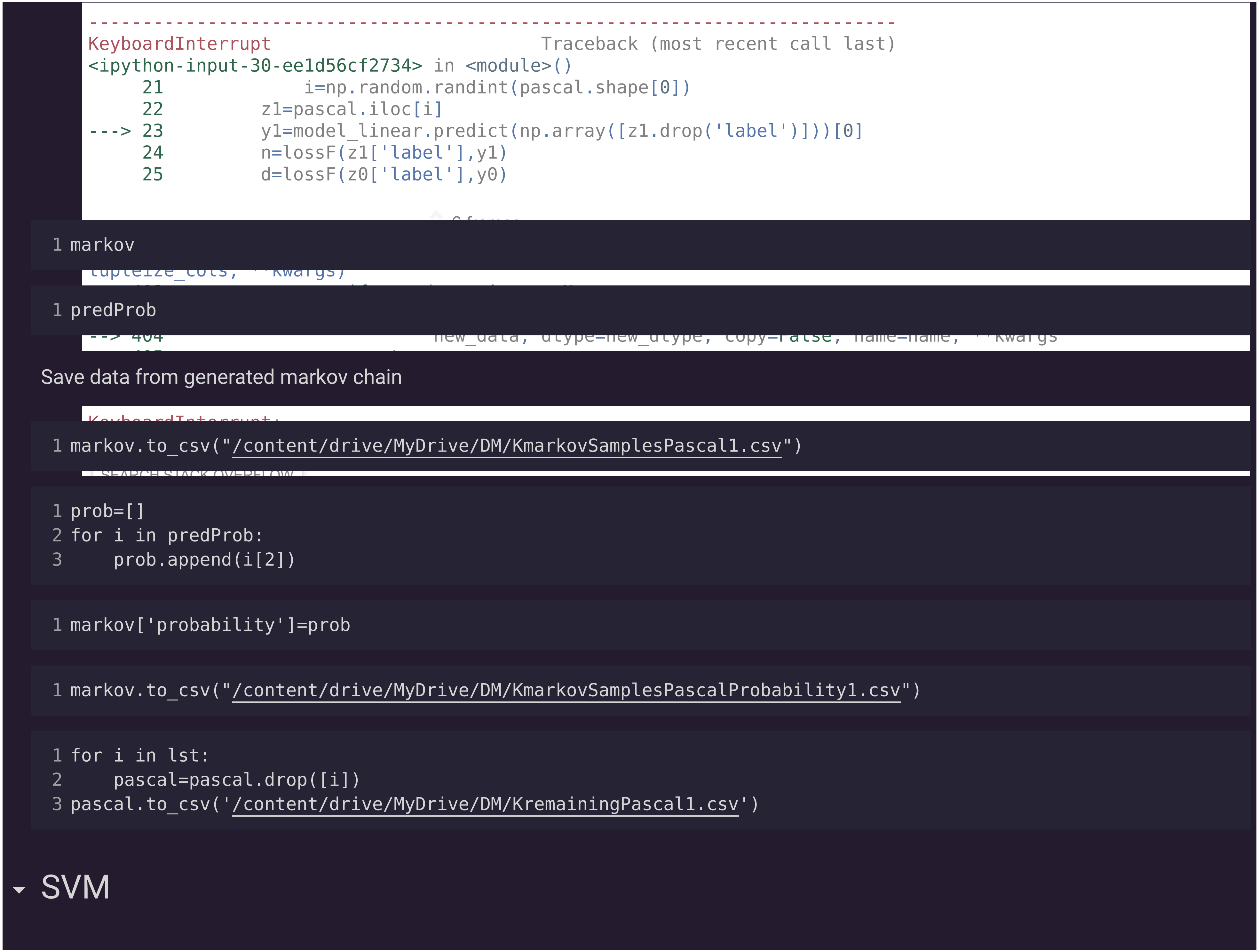
Step-II TO Step-VII



```python
# Loop markov chain generator k times
for km in range(2):
    # Reset parameters for next markov chain
    if km!=0:
        predProb=[]
        model_linear=train(markov)
        markov= pd.DataFrame(columns = pascal.columns)
        mcls={i:0 for i in uniqCls}
    lst=[]

    # Chosing a random sample as first of markov chain
    i=np.random.randint(pascal.shape[0])
    z0=pascal.iloc[i]
    y0=model_linear.predict(np.array([z0.drop('label')]))[0]
    if m%classCNT==0:
        mcls[z0['label']]+=1
    while exist(mcls,limit):
        # choosing a random sample
        i=np.random.randint(pascal.shape[0])
        while i in lst:
            i=np.random.randint(pascal.shape[0])
        z1=pascal.iloc[i]
        y1=model_linear.predict(np.array([z1.drop('label')]))[0]
        n=lossF(z1['label'],y1)
        d=lossF(z0['label'],y0)
        p=n/d

        # Deciding of acceptance of chosen sample and its probability in markov chain
        if acc==k:
            acc=0
            p2=q*p
            p2=min(p2,1)
            predProb.append([z1['label'],y1,p2])
            markov=markov.append(z1)
            lst.append(i)
            z0=z1
            mcls[z1['label']]+=1
            acc+=1
```



```
        elif p==1 and z0['label']==z1['label']:
            n=np.exp(-y1*z1['label'])
            d=np.exp(-y0*z0['label'])

            p1=n/d
            p1=min(p1,1)
            predProb.append([z1['label'],y1,p1])
            markov=markov.append(z1)
            lst.append(i)
            z0=z1
            mcls[z1['label']]+=1
            acc+=1
        elif p<1:
            predProb.append([z1['label'],y1,p])
            markov=markov.append(z1)
            lst.append(i)
            z0=z1
            mcls[z1['label']]+=1
            acc+=1
        elif p==1 and z0['label']!=z1['label']:
            predProb.append([z1['label'],y1,p])
            markov=markov.append(z1)
            lst.append(i)
            z0=z1
            mcls[z1['label']]+=1
            acc+=1
markov
```



```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-30-ee1d56cf2734> in <module>()
     21             i=np.random.randint(pascal.shape[0])
     22         z1=pascal.iloc[i]
---> 23         y1=model_linear.predict(np.array([z1.drop('label')]))[0]
     24         n=lossF(z1['label'],y1)
     25         d=lossF(z0['label'],y0)

                          6 frames
```

```
markov
```

```
tupleize_cols, **kwargs)
```

```
predProb
```

```
--> 404                     new_data, dtype=new_dtype, copy=False, name=name, **kwargs
```

Save data from generated markov chain

```
KeyboardInterrupt:
```

```
markov.to_csv("/content/drive/MyDrive/DM/KmarkovSamplesPascal1.csv")
```

```
SEARCH STACK OVERFLOW
```

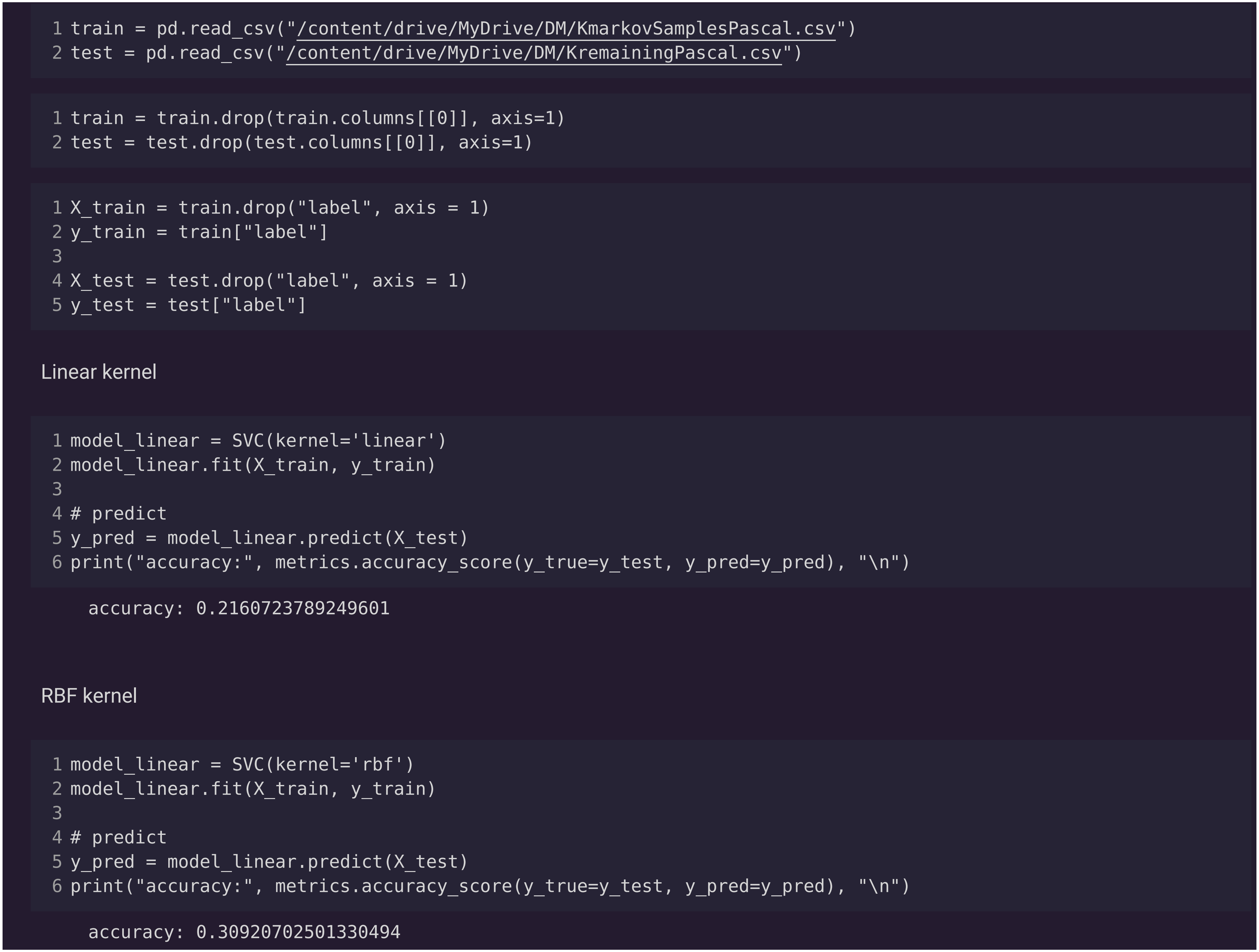
```
prob=[]
for i in predProb:
    prob.append(i[2])
```

```
markov['probability']=prob
```

```
markov.to_csv("/content/drive/MyDrive/DM/KmarkovSamplesPascalProbability1.csv")
```

```
for i in lst:
    pascal=pascal.drop([i])
pascal.to_csv('/content/drive/MyDrive/DM/KremainingPascal1.csv')
```

SVM



```
train = pd.read_csv("/content/drive/MyDrive/DM/KmarkovSamplesPascal.csv")
test = pd.read_csv("/content/drive/MyDrive/DM/KremainingPascal.csv")
```

```
train = train.drop(train.columns[[0]], axis=1)
test = test.drop(test.columns[[0]], axis=1)
```

```
X_train = train.drop("label", axis = 1)
y_train = train["label"]

X_test = test.drop("label", axis = 1)
y_test = test["label"]
```

Linear kernel

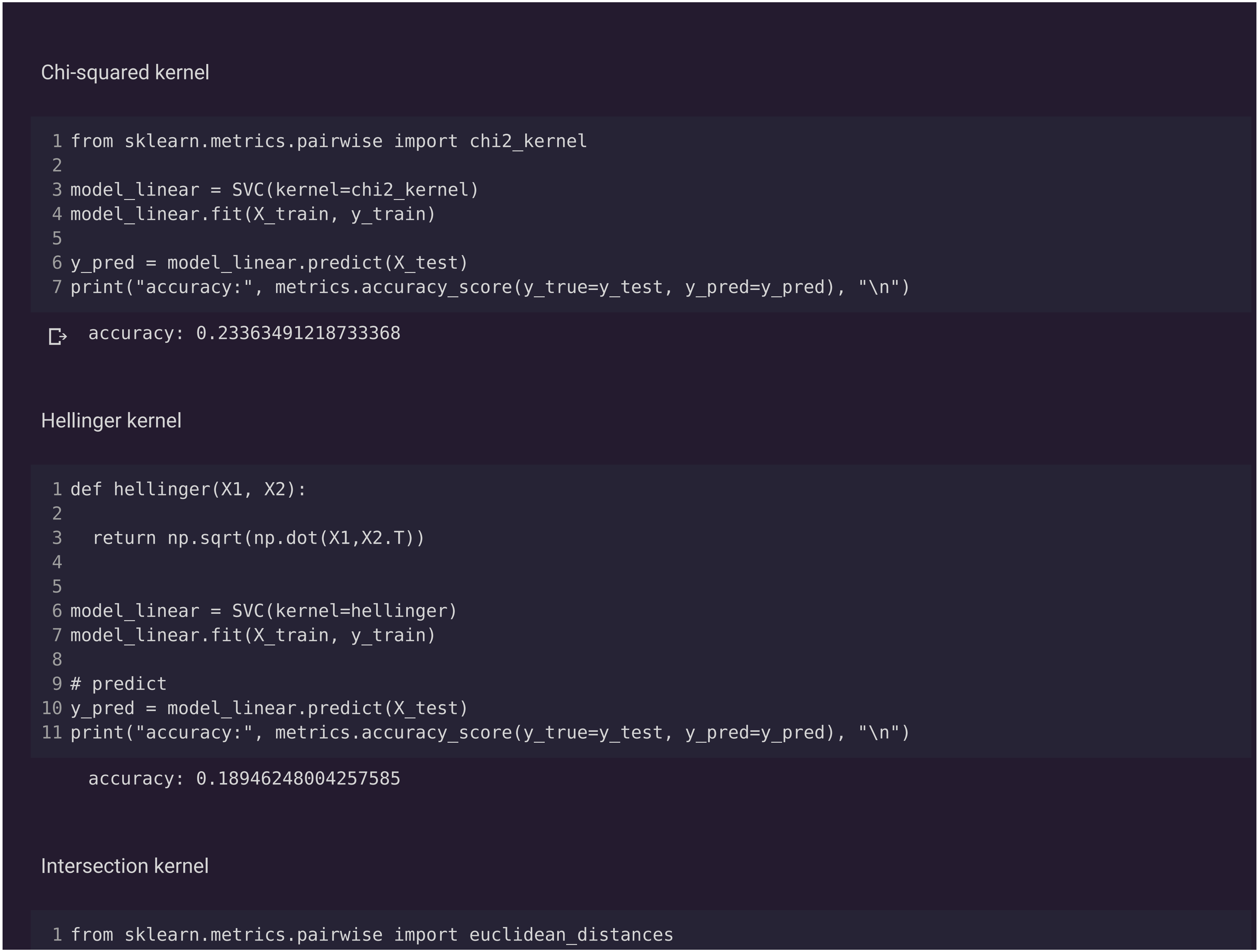
```
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
```

```
    accuracy: 0.2160723789249601
```

RBF kernel

```
model_linear = SVC(kernel='rbf')
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
```

```
    accuracy: 0.30920702501330494
```



Chi-squared kernel

```
from sklearn.metrics.pairwise import chi2_kernel

model_linear = SVC(kernel=chi2_kernel)
model_linear.fit(X_train, y_train)

y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
```

```
accuracy: 0.23363491218733368
```

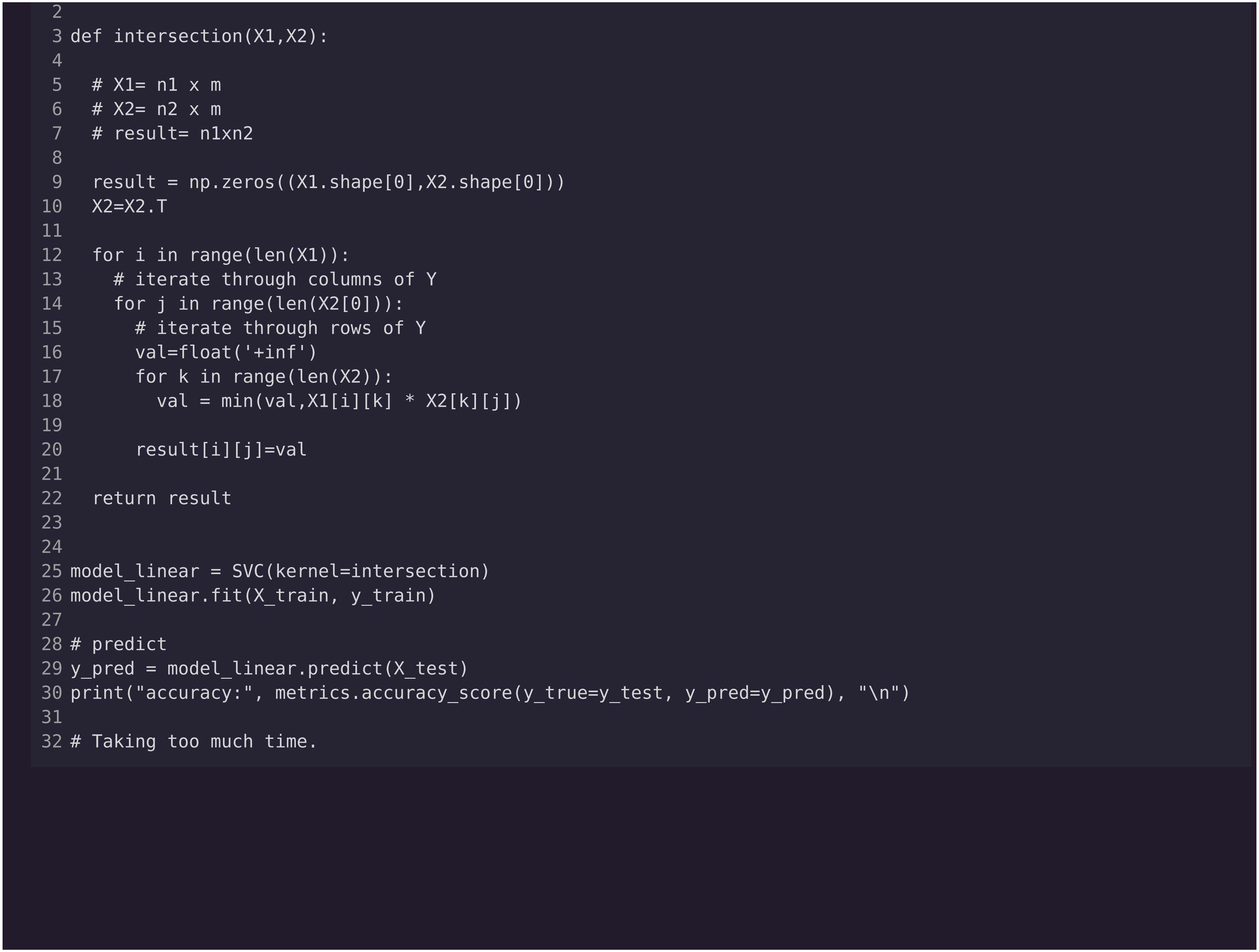
Hellinger kernel

```
def hellinger(X1, X2):

  return np.sqrt(np.dot(X1,X2.T))


model_linear = SVC(kernel=hellinger)
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
```

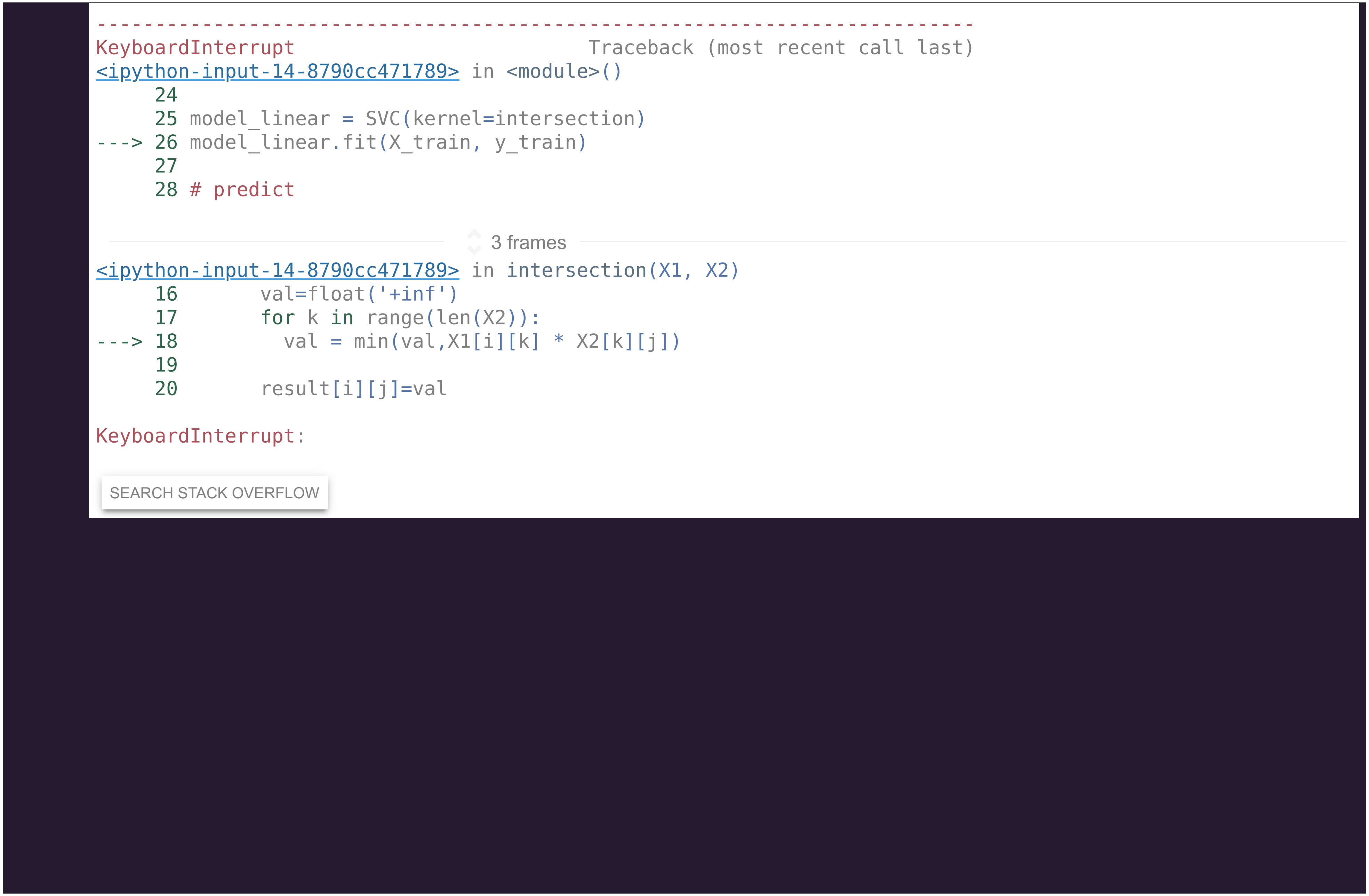
```
accuracy: 0.18946248004257585
```

Intersection kernel

```
from sklearn.metrics.pairwise import euclidean_distances
```



```python

def intersection(X1,X2):

  # X1= n1 x m
  # X2= n2 x m
  # result= n1xn2

  result = np.zeros((X1.shape[0],X2.shape[0]))
  X2=X2.T

  for i in range(len(X1)):
    # iterate through columns of Y
    for j in range(len(X2[0])):
      # iterate through rows of Y
      val=float('+inf')
      for k in range(len(X2)):
        val = min(val,X1[i][k] * X2[k][j])

      result[i][j]=val

  return result


model_linear = SVC(kernel=intersection)
model_linear.fit(X_train, y_train)

# predict
y_pred = model_linear.predict(X_test)
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")

# Taking too much time.
```



```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-14-8790cc471789> in <module>()
     24 
     25 model_linear = SVC(kernel=intersection)
---> 26 model_linear.fit(X_train, y_train)
     27 
     28 # predict

                                3 frames
<ipython-input-14-8790cc471789> in intersection(X1, X2)
     16       val=float('+inf')
     17       for k in range(len(X2)):
---> 18         val = min(val,X1[i][k] * X2[k][j])
     19 
     20       result[i][j]=val

KeyboardInterrupt: 
```