In [1]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import SelectKBest, mutual_info_regression
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# import models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn import svm

import warnings
```

In [2]:
```
df = pd.read_csv('Shelf_life_data.csv')
warnings.filterwarnings('ignore')
```

Data Examination / Preparation

In [3]:
```
# Select necessary columns
df_new = df[['Location','Type','Packaging_Condition','Odor','Color','Surface','Product
df_new.head()
```

Out[3]:

	Location	Type	Packaging_Condition	Odor	Color	Surface	Product_Condition	TPC_Value
0	Plant A	Boneless	Acceptable	Sour	Pale	Slimy	Not Acceptable	3.20
1	Plant A	Bone-In	Leaker	Sour	Pale	Slimy	Not Acceptable	6.72
2	Plant A	Boneless	Leaker	Sour	Pale	Slimy	Not Acceptable	7.36
3	Plant A	Boneless	Leaker	Sour	Pale	Slimy	Not Acceptable	6.61
4	Plant B	Boneless	Acceptable	Off Condition	Good	Slimy	Not Acceptable	3.43

In [4]:
```
# drop Location
df_new.drop(columns=['Location'], inplace=True)
df_new.head()
```

Out[4]:

	Type	Packaging_Condition	Odor	Color	Surface	Product_Condition	TPC_Value	Age
0	Boneless	Acceptable	Sour	Pale	Slimy	Not Acceptable	3.20	41
1	Bone-In	Leaker	Sour	Pale	Slimy	Not Acceptable	6.72	41
2	Boneless	Leaker	Sour	Pale	Slimy	Not Acceptable	7.36	41
3	Boneless	Leaker	Sour	Pale	Slimy	Not Acceptable	6.61	41
4	Boneless	Acceptable	Off Condition	Good	Slimy	Not Acceptable	3.43	40

In [5]:
```
# label Packaging condition: Acceptable as 1 and Leaker as 0
df_new['Packaging_Condition']=df_new['Packaging_Condition'].map({'Acceptable':1, 'Leak

# label Bone-In as 1 and Boneless as 0
df_new['Type']=df_new['Type'].map({'Bone-In':1, 'Boneless':0})

# label Product Condition: Acceptable as 1 and Not Acceptable as 0
df_new['Product_Condition'] = df_new['Product_Condition'].map({'Acceptable': 0, 'Not A
df_new.head()
```

Out[5]:

	Type	Packaging_Condition	Odor	Color	Surface	Product_Condition	TPC_Value	Age
0	0	1	Sour	Pale	Slimy	1	3.20	41
1	1	0	Sour	Pale	Slimy	1	6.72	41
2	0	0	Sour	Pale	Slimy	1	7.36	41
3	0	0	Sour	Pale	Slimy	1	6.61	41
4	0	1	Off Condition	Good	Slimy	1	3.43	40

In [6]:
```
# Now arrange columns in appropriate order
data = df_new[['Packaging_Condition','Odor','Color','Surface','Product_Condition','TPC
data.head()
```

Out[6]:

	Packaging_Condition	Odor	Color	Surface	Product_Condition	TPC_Value	Age
0	1	Sour	Pale	Slimy	1	3.20	41
1	0	Sour	Pale	Slimy	1	6.72	41
2	0	Sour	Pale	Slimy	1	7.36	41
3	0	Sour	Pale	Slimy	1	6.61	41
4	1	Off Condition	Good	Slimy	1	3.43	40

In [7]:
```
# convert categorical variables to dummy variables
catCols = ['Odor','Color','Surface']
dummies = pd.get_dummies(df_new[catCols])
data_dummies = df_new.drop(columns=catCols)
data_final = data_dummies.join(dummies)
#create a duplicate column and save for later use in prediction
data_final.head()
```

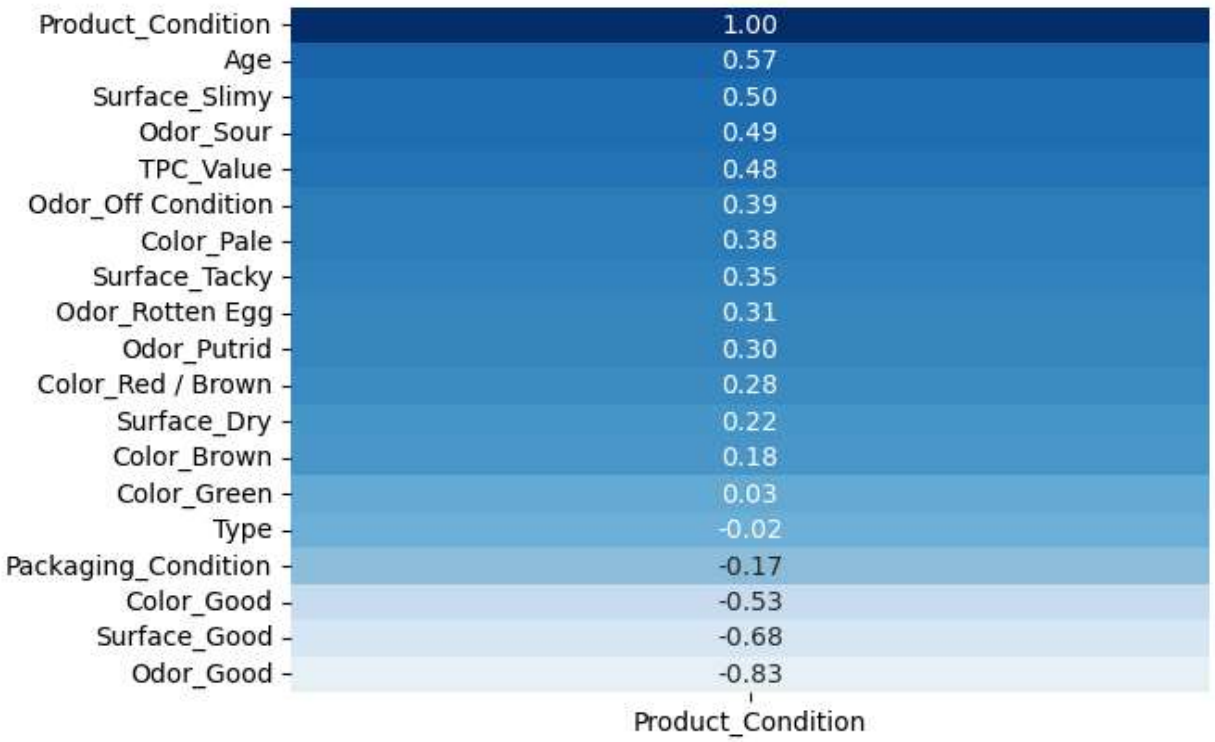
Out[7]:

	Type	Packaging_Condition	Product_Condition	TPC_Value	Age	Odor_Good	Odor_Off Condition	Odor_Putri
0	0	1	1	3.20	41	0	0	
1	1	0	1	6.72	41	0	0	
2	0	0	1	7.36	41	0	0	
3	0	0	1	6.61	41	0	0	
4	0	1	1	3.43	40	0	1	

In [8]:
```
# scale numerical data
scaler = StandardScaler()
data_final[['TPC_Value','Age']] = scaler.fit_transform(data_final[['TPC_Value','Age']]
data_std = data_final
```

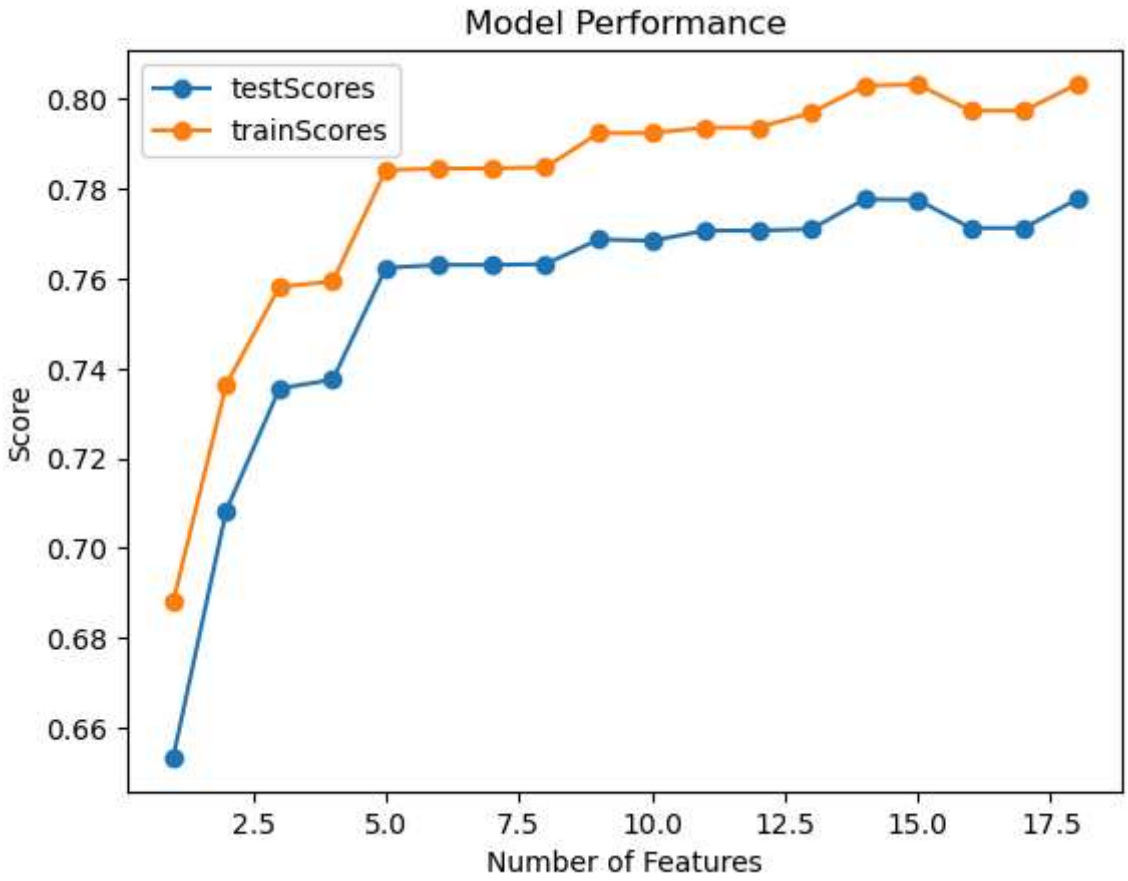
In [9]:
```
# check correlations
data_std.corr()[['Product_Condition']].sort_values('Product_Condition', ascending=Fals
#heatmap of correlation
sns.heatmap(data=data_std.corr()[['Product_Condition']].sort_values('Product_Condition
            annot=True,cmap='Blues',vmin=-1.0, vmax=1.0, cbar=False,fmt='.2f')
```

Out[9]:
```
<Axes: >
```



In [10]:
```
# Check performance of the linear regression model

productTrain, productTest = train_test_split(data_std, test_size=0.2, random_state=20)
productModel = LinearRegression()
testScores = []
trainScores = []

for i in range(1, len(productTrain.columns)):
```

```
    fs = SelectKBest(score_func=mutual_info_regression, k=i)
    fs.fit(productTrain.drop(columns=['Product_Condition']), productTrain['Product_Cor

    x_train_fs = fs.transform(productTrain.drop(columns=['Product_Condition']))
    x_test_fs = fs.transform(productTest.drop(columns=['Product_Condition']))

    productModel.fit(x_train_fs, productTrain['Product_Condition'])

    testScore = productModel.score(x_test_fs, productTest['Product_Condition'])
    trainScore = productModel.score(x_train_fs, productTrain['Product_Condition'])
    testScores.append(testScore)
    trainScores.append(trainScore)


df = pd.DataFrame(data={'testScores': testScores, 'trainScores': trainScores})
df['numFeatures'] = df.index + 1

df.plot(x='numFeatures', y=['testScores', 'trainScores'], marker='o')
plt.xlabel('Number of Features')
plt.ylabel('Score')
plt.title('Model Performance')
plt.show()
```



In [11]:
```
# to improve performance and resize features, we will like to display the importance o
df1 = pd.DataFrame(productTrain.drop(columns=['Product_Condition']).columns, columns=[
df2=pd.DataFrame(fs.scores_, columns=['importance'])
importance = df1.join(df2)
importance.sort_values('importance',ascending=False)

# Based on this results, the location or plant and type are irrelevant so will drop lo
```

Out[11]:

	feature	importance
4	Odor_Good	0.352786
3	Age	0.253635
15	Surface_Good	0.237965
2	TPC_Value	0.153428
10	Color_Good	0.132212
8	Odor_Sour	0.131095
16	Surface_Slimy	0.100226
5	Odor_Off Condition	0.078996
17	Surface_Tacky	0.063024
12	Color_Pale	0.053387
6	Odor_Putrid	0.045717
7	Odor_Rotten Egg	0.034235
13	Color_Red / Brown	0.029874
1	Packaging_Condition	0.025855
9	Color_Brown	0.021981
14	Surface_Dry	0.013629
0	Type	0.000710
11	Color_Green	0.000000

In [12]:
```
# drop irrelevant columns
data_std.drop(columns=['Packaging_Condition','Type'], inplace=True)
data_std.head()
```

Out[12]:

	Product_Condition	TPC_Value	Age	Odor_Good	Odor_Off Condition	Odor_Putrid	Odor_Rotten Egg	Odor_S
0	1	-0.493434	0.746203	0	0	0	0	
1	1	1.119780	0.746203	0	0	0	0	
2	1	1.413092	0.746203	0	0	0	0	
3	1	1.069367	0.746203	0	0	0	0	
4	1	-0.388025	0.706223	0	1	0	0	

List of Models to Compare

In [13]:
```
models = [LogisticRegression(max_iter=1000), SVC(kernel='linear'),KNeighborsClassifier
```

In [14]:
```
# select features and target
x= data_std.drop(columns=['Product_Condition']) # features
```

```
y= data_std['Product_Condition'] # target
x.head()
```

Out[14]:

	TPC_Value	Age	Odor_Good	Odor_Off Condition	Odor_Putrid	Odor_Rotten Egg	Odor_Sour	Color_Brown
0	-0.493434	0.746203	0	0	0	0	1	0
1	1.119780	0.746203	0	0	0	0	1	0
2	1.413092	0.746203	0	0	0	0	1	0
3	1.069367	0.746203	0	0	0	0	1	0
4	-0.388025	0.706223	0	1	0	0	0	0

In [15]:

```
# PCA for dimensionality reduction

import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Perform PCA
pca = PCA()
pca.fit(x)

# Get variance ratios
variance_ratios = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(variance_ratios)

# Print results
print("Variance explained by each component:", variance_ratios)
print("Cumulative variance explained:", cumulative_variance)
```

```
Variance explained by each component: [6.38795111e-01 1.03631172e-01 8.76683585e-02
4.66064671e-02
 2.65675654e-02 2.26828236e-02 2.11844188e-02 1.76974826e-02
 1.30283224e-02 1.06646489e-02 6.27363409e-03 5.08520693e-03
 1.14788747e-04 3.90166545e-32 2.51071291e-32 1.43985827e-32]
Cumulative variance explained: [0.63879511 0.74242628 0.83009464 0.87670111 0.9032686
7 0.9259515
 0.94713592 0.9648334  0.97786172 0.98852637 0.9948     0.99988521
 1.         1.         1.         1.        ]
```

The cumulative variance explained by the principal components shows how much of the total dataset variance is captured as we include more components: With just the first component, 63.88% of the dataset variance is explained. Adding the second component increases this to 74.24%. Including up to the third component reaches 83.01%, and so forth.

Interpretation of Cumulative Proportions: By the time we include the first 12 components, virtually all (99.99%) of the dataset's variance is accounted for. This comprehensive coverage suggests that we can achieve nearly full information retention with just 12 out of the 16 components, though practically, focusing on the first few may be sufficient for most analytical purposes due to diminishing returns in explained variance with each additional component.

In [16]:
```
# function to compare models
def compare_models_cross_val():
    result =[]
    for model in models:
        cv_score = cross_val_score(model,x,y, cv=5)
        mean_accuracy =sum(cv_score)/len(cv_score)
        mean_accuracy = round((mean_accuracy*100),2) # acuracy in percent rounded to 2
        print(f'Cross val accuracies for {model} = {cv_score}')
        print(f'Accurracy score of the {model} = {mean_accuracy} %')

        result.append({"Model": model, "Accuracy": mean_accuracy})

    # Convert results into a DataFrame
    result_df = pd.DataFrame(result)
    return result_df

# Call the function and print the results DataFrame # for standard scaling
result_df = compare_models_cross_val()
result_df
```

```
Cross val accuracies for LogisticRegression(max_iter=1000) = [0.97331554 0.96064043
0.94596398 0.97264843 0.94796531]
Accurracy score of the LogisticRegression(max_iter=1000) = 96.01 %
Cross val accuracies for SVC(kernel='linear') = [0.96864576 0.96931288 0.9472982  0.9
3929286 0.93328886]
Accurracy score of the SVC(kernel='linear') = 95.16 %
Cross val accuracies for KNeighborsClassifier() = [0.96731154 0.96731154 0.95663776
0.9646431  0.94062708]
Accurracy score of the KNeighborsClassifier() = 95.93 %
Cross val accuracies for RandomForestClassifier(random_state=0) = [0.96664443 0.96264
176 0.95196798 0.9593062  0.95597065]
Accurracy score of the RandomForestClassifier(random_state=0) = 95.93 %
```

Out[16]:

	Model	Accuracy
0	LogisticRegression(max_iter=1000)	96.01
1	SVC(kernel='linear')	95.16
2	KNeighborsClassifier()	95.93
3	RandomForestClassifier(random_state=0)	95.93

Based on these results for the data, Logistic regression has the highest accuracy value with the default hyperparameter values

We will now utilize GridSearchCV to compare the models with different Hyperparameter values

In [17]:
```
Models = [LogisticRegression(max_iter=10000), SVC(),KNeighborsClassifier(),RandomFores

# I will create a dictionery that contains hyperparamter values for the above models
hyper_params = {
    'log_reg_hyper_params': {
        'C': [1, 5, 10, 20]  # Corrected 'c' to 'C' to match scikit-learn's conventio
    },
    'svc_hyper_params': {
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Added a missing comma at th
        'C': [1, 5, 10, 20]  # Corrected 'c' to 'C' to match scikit-learn's convention
```

```
    },
    'knn_hyper_params': {
        'n_neighbors': [2, 3, 5, 10]
    },
    'random_forest_hyper_params': {
        'n_estimators': [10, 20, 50, 100]
    }
}
# create a list of keys for hyperparameters
model_keys = list(hyper_params.keys())
model_keys
```

Out[17]:
```
['log_reg_hyper_params',
 'svc_hyper_params',
 'knn_hyper_params',
 'random_forest_hyper_params']
```

In [18]:
```
#### Now we will apply GridSearchCV
def selectModel(model_list, hyperparam_dict):
    results =[]
    i = 0
    for model in model_list:
        key = model_keys[i]
        params = hyperparam_dict[key]
        i += 1
        print(model)
        print(params)
        print('-------------'*5)
        classifier = GridSearchCV(model, params, cv=5)
        # Fit the data to the classifier
        classifier.fit(x, y)
        results.append({
            'Model used': model.__class__.__name__, # Use model's class name for a cle
            'Highest Score': classifier.best_score_,
            'Best hyperparameters': classifier.best_params_
        })

    results_df = pd.DataFrame(results, columns=['Model used', 'Highest Score', 'Best h
    return results_df
```

In [19]:
```
#apply hypertuning

selectModel(Models, hyper_params)
```

```
LogisticRegression(max_iter=10000)
{'C': [1, 5, 10, 20]}
------------------------------------------------------------
SVC()
{'kernel': ['linear', 'poly', 'rbf', 'sigmoid'], 'C': [1, 5, 10, 20]}
------------------------------------------------------------
KNeighborsClassifier()
{'n_neighbors': [2, 3, 5, 10]}
------------------------------------------------------------
RandomForestClassifier(random_state=0)
{'n_estimators': [10, 20, 50, 100]}
------------------------------------------------------------
```

Out[19]:

	Model used	Highest Score	Best hyperparameters
0	LogisticRegression	0.960907	{'C': 20}
1	SVC	0.970247	{'C': 20, 'kernel': 'poly'}
2	KNeighborsClassifier	0.959306	{'n_neighbors': 5}
3	RandomForestClassifier	0.960507	{'n_estimators': 20}

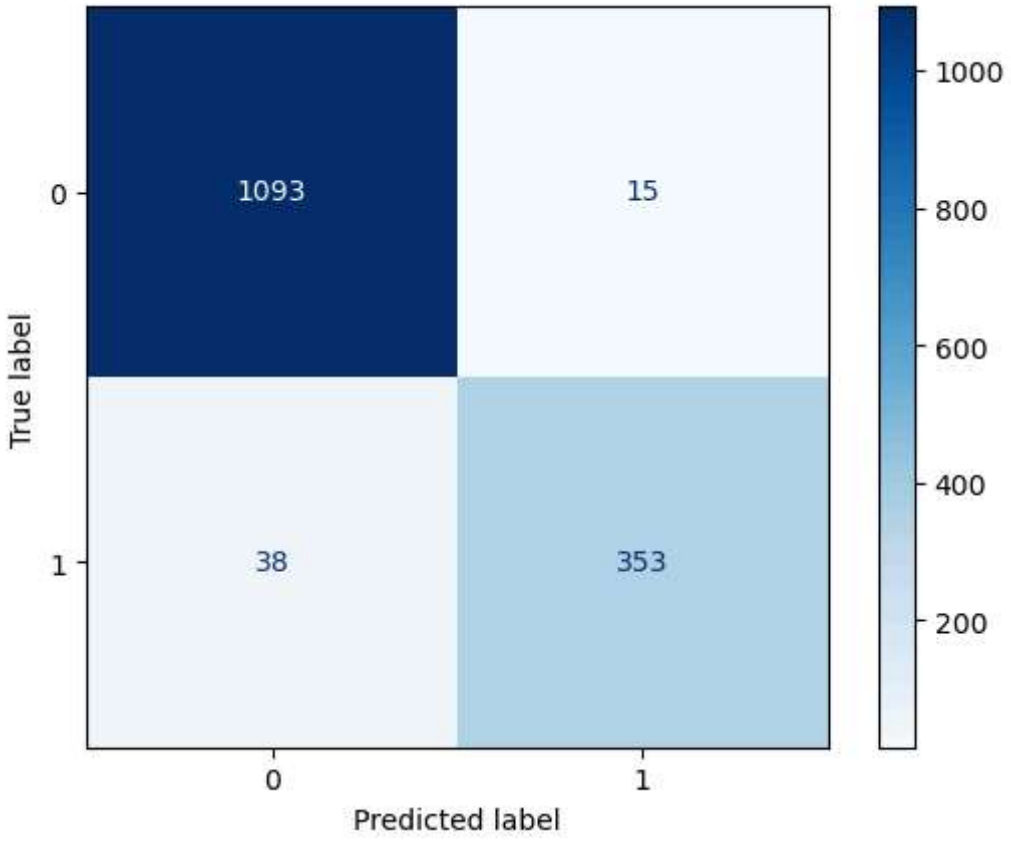
In [20]:
```
# select features and target
x= data_std.drop(columns=['Product_Condition']) # features
y= data_std['Product_Condition'] # target

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=4
```

In [21]:
```
#instantiate the model
log_regression = LogisticRegression(C=20)

#fit the model using the training data
log_regression.fit(x_train,y_train)

#use model to make predictions on test data
y_pred = log_regression.predict(x_test)
```

In [22]:
```
from sklearn import metrics

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[22]:
```
array([[1093,   15],
       [  38,  353]], dtype=int64)
```

In [23]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



In [24]:
```
#From the confusion matrix we can see that:
#True positive predictions: 353
#True negative predictions: 1093
#False positive predictions: 15
#False negative predictions: 38
```

In [25]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

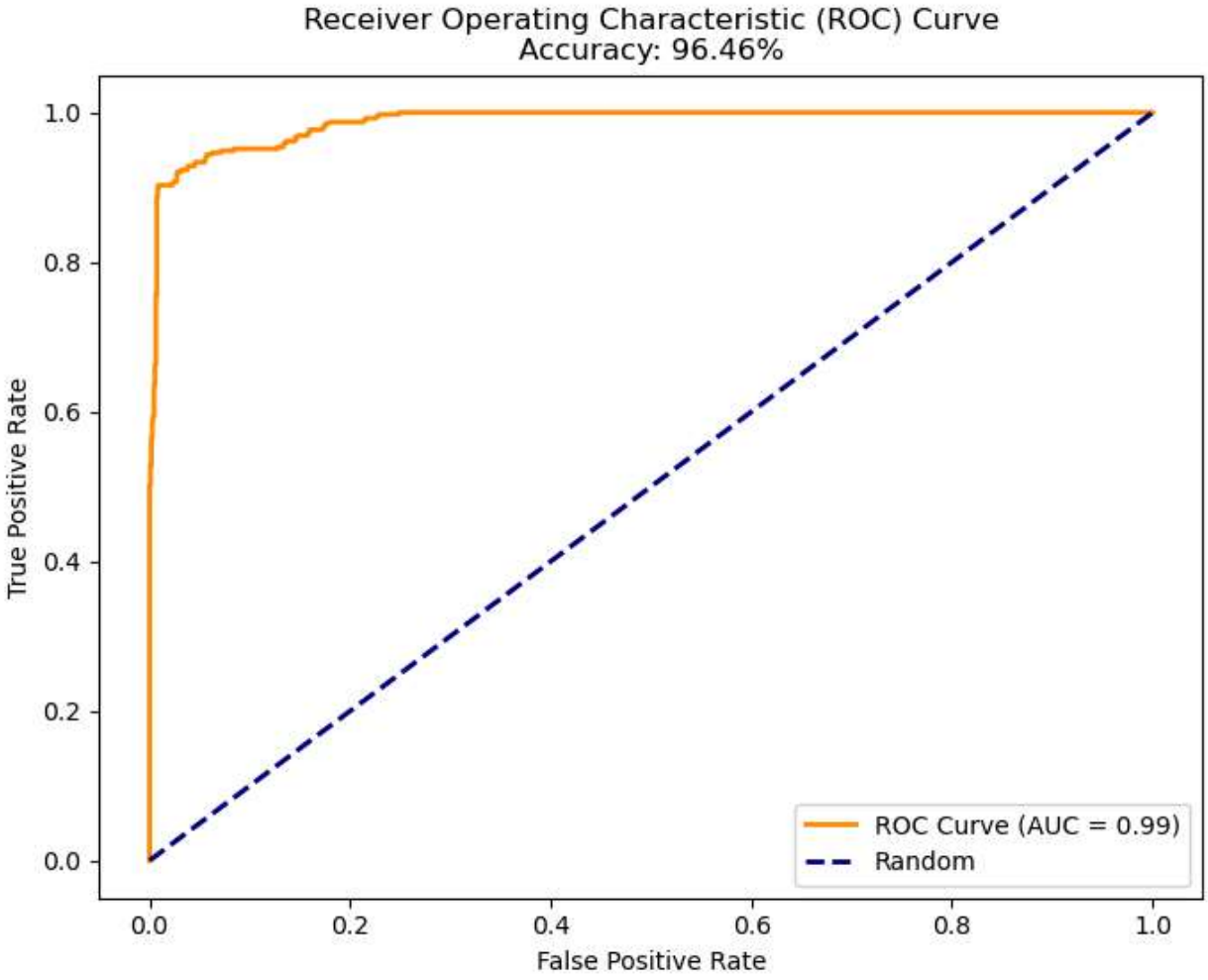
```
Accuracy: 0.9646430953969313

Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98      1108
           1       0.96      0.90      0.93       391

    accuracy                           0.96      1499
   macro avg       0.96      0.94      0.95      1499
weighted avg       0.96      0.96      0.96      1499
```

This tells us the model made the correct predictions 96.2% of the time

The classification report shows the logistic regression model fits well on this dataset.

In [26]:
```
# Plot ROC Curve

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, r
```

```
accuracy = accuracy_score(y_test, y_pred)

y_prob = log_regression.predict_proba(x_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve\nAccuracy: {:.2f}%'.format(
    accuracy * 100))
plt.legend(loc="lower right")
plt.show()
```



The ROC curve above displays the percent of true positives predicted by the model as the prediction probability cutoff is lowered from 1 to 0. The higher the AUC (area under the curve), the more accurately our model is able to predict outcomes.

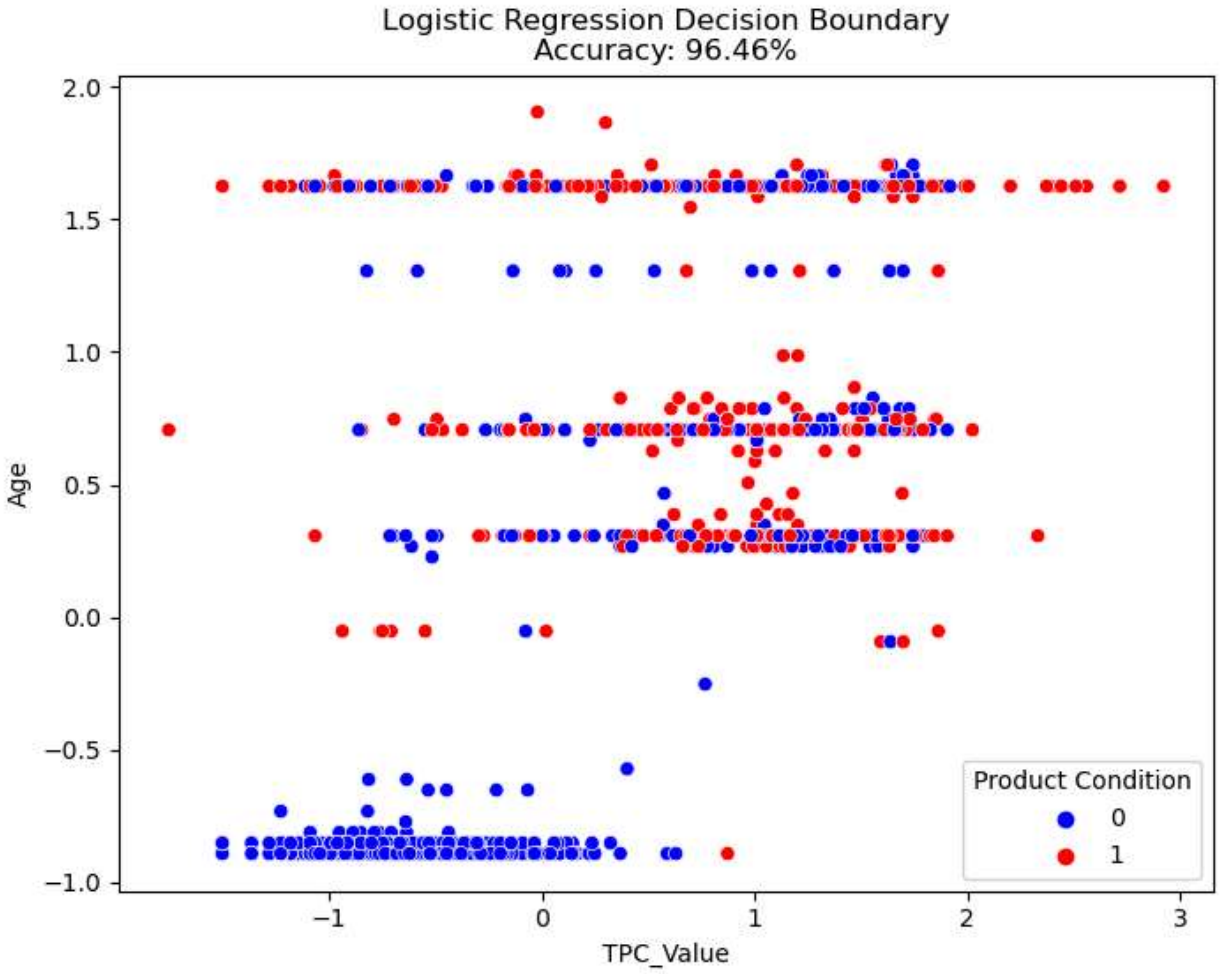
In [27]:
```
x_test.head()
```

Out[27]:

	TPC_Value	Age	Odor_Good	Odor_Off Condition	Odor_Putrid	Odor_Rotten Egg	Odor_Sour	Color_Brov
6416	-1.089224	-0.892962	1	0	0	0	0	
5432	-1.006730	-0.852982	1	0	0	0	0	
6575	1.880557	1.625754	1	0	0	0	0	
5850	-0.566762	-0.892962	1	0	0	0	0	
4776	-0.878406	-0.892962	1	0	0	0	0	

In [28]:

```
# Visualize the decision boundary with accuracy information

plt.figure(figsize=(8, 6))
sns.scatterplot(x=x_test.iloc[:, 0], y=x_test.iloc[:, 1], hue=y_test, palette={
                0: 'blue', 1: 'red'}, marker='o')
plt.xlabel("TPC_Value")
plt.ylabel("Age")
plt.title("Logistic Regression Decision Boundary\nAccuracy: {:.2f}%".format(
    accuracy * 100))
plt.legend(title="Product Condition", loc="lower right")
plt.show()
```



The decision boarder is a scatter plot of Age and TPC_Value color coded by product condition. Product condition of 0 indicates poor product condition and 1 indicates good product

condition.

Predicting Age and TPC when product condition goes from good to bad

In [29]:
```
# convert categorical variables to dummy variables
catCols = ['Odor','Color','Surface']
dummies = pd.get_dummies(df_new[catCols])
data_dummies = df_new.drop(columns=catCols)
df_final = data_dummies.join(dummies)
#create a duplicate column and save for later use in prediction
df_final.head()
```

Out[29]:

	Type	Packaging_Condition	Product_Condition	TPC_Value	Age	Odor_Good	Odor_Off Condition	Odor_Putri
0	0	1	1	3.20	41	0	0	
1	1	0	1	6.72	41	0	0	
2	0	0	1	7.36	41	0	0	
3	0	0	1	6.61	41	0	0	
4	0	1	1	3.43	40	0	1	

In [30]:
```
#we will the saved dataframe from above to use our prediction and
#dropping the two irrelevent columns determined above.

df_final.drop(columns=['Packaging_Condition','Type'], inplace=True)
df_final.head()
```

Out[30]:

	Product_Condition	TPC_Value	Age	Odor_Good	Odor_Off Condition	Odor_Putrid	Odor_Rotten Egg	Odor_Sour
0	1	3.20	41	0	0	0	0	1
1	1	6.72	41	0	0	0	0	1
2	1	7.36	41	0	0	0	0	1
3	1	6.61	41	0	0	0	0	1
4	1	3.43	40	0	1	0	0	0

In [31]:
```
#split the data_final data set into test and train

x = df_final.drop(columns=['Product_Condition']) # features
y = df_final['Product_Condition'] # target

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=4
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[31]:
```
((5996, 16), (1499, 16), (5996,), (1499,))
```

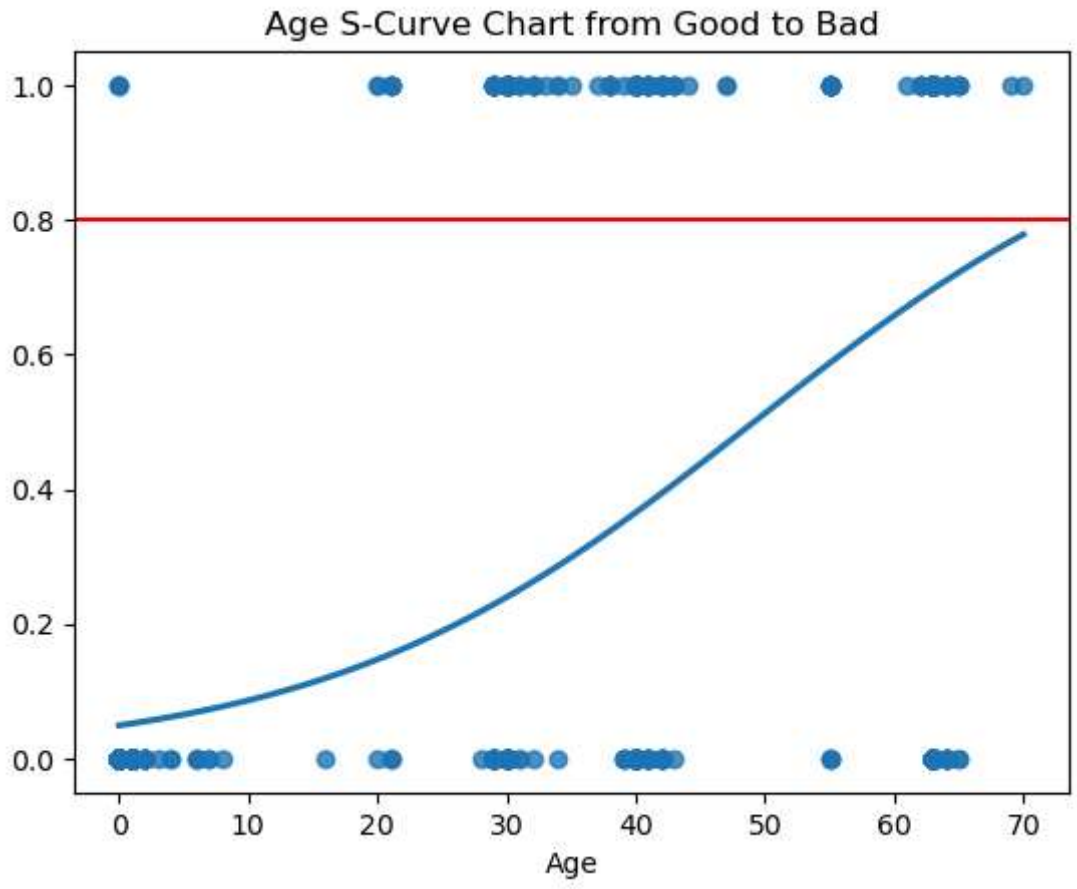
In [32]:
```
#use standard scaler to scale the x_train and x_test dataset.
scaler = StandardScaler()

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.fit_transform(x_test)
```

In [33]:
```
#fit the data to a logistic regression model with predetermined parameters

log_reg = LogisticRegression(C=20).fit(x_train_scaled, y_train)
```

In [34]:
```
log_reg_pred = log_reg.predict(x_test_scaled)
```

In [35]:
```
#Plot the S_curve of the age from good product condition (0) to bad product condition
#with 80 percent confidence level.

default = log_reg_pred
age = x_test['Age']

sns.regplot(x=age, y=default, data=df_final, logistic=True, ci=None)
plt.title("Age S-Curve Chart from Good to Bad")
plt.axhline(y=0.8, color='r', linestyle='-')
```

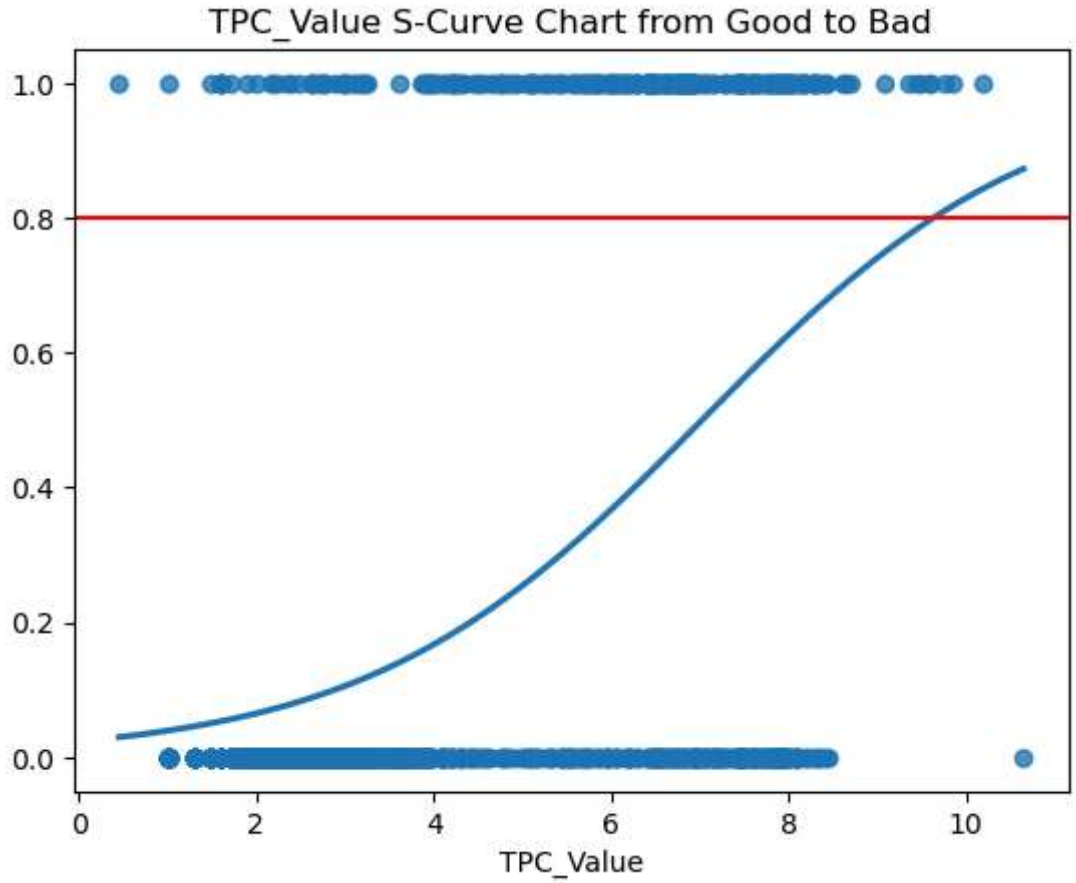
Out[35]:
```
<matplotlib.lines.Line2D at 0x1bbfc16c760>
```



In [36]:
```
#Plot the S_curve of the TPC_Value from good product condition (0) to bad product cond
#with 80 percent confidence level.

default = log_reg_pred
age = x_test['TPC_Value']
```

```
sns.regplot(x=age, y=default, data=df_final, logistic=True, ci=None)
plt.title("TPC_Value S-Curve Chart from Good to Bad")
plt.axhline(y=0.8, color='r', linestyle='-')
```

Out[36]: <matplotlib.lines.Line2D at 0x1bbff89c190>



In [ ]: