You are exploring the wilderness of *Mushroomia*, a land populated by a plethora of diverse fauna and flora. In particular, *Mushroomia* is known for its unparalleled variety in mushrooms. However, not all the mushrooms in *Mushroomia* are edible. As you make your way through *Mushroomia*, you would like to know which mushrooms are edible, in order to forage for supplies for your daily mushroom soup.

You have access to:

- *Shroomster Pro Max* ^{TM} - a state of the art data collection device, developed by *Mushroomia*, that allows you to collect various data points about any mushroom you encounter in the wild
- *The National Archives on Mushrooms* - a dataset collected over the years by the government of *Mushroomia*

To address this problem, you decide to use the skills you learnt in CSM148 and train machine learning models on the *The National Archives on Mushrooms* in order to use your *Shroomster Pro Max* ^{TM} to determine whether the mushrooms you encounter on your adventure can be added to your daily mushroom soup.

This project will be more unstructured than the previous two projects in order to allow you to experience how data science problems are solved in practice. There are two parts to this project: a Jupyter Notebook with your code (where you explore, visualize, process your data and train machine learning models) and a report (where you explain the various choices you make in your implementation and analyze the final performance of your models).

1. Loading and Viewing Data

In [ ]:
```
# Delete this cell if not using google colab

from google.colab import files
files.upload()
```

In [1]:
```
import pandas as pd
mushroomTest = pd.read_csv('mushroom_test.csv', sep=";")
mushroomTrain = pd.read_csv('mushroom_train.csv', sep=";")

# note delimitter for this file is semi-colon (;) not comma
```

In [3]:
```
mushroomTest.head()
```

Out[3]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	...	stem-root
0	p	2.50	b	NaN	k	f	a	NaN	k	8.42	...	NaN
1	p	3.07	b	NaN	k	f	a	NaN	n	7.24	...	NaN
2	p	3.30	b	NaN	n	f	a	NaN	n	10.22	...	NaN
3	p	3.49	b	NaN	k	f	a	NaN	k	11.00	...	NaN
4	p	2.79	b	NaN	n	f	a	NaN	n	6.97	...	NaN

5 rows × 21 columns

In [ ]:
```
mushroomTrain.head()
```

Out[ ]:

	class	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	...	stem-root
0	p	15.26	x	g	o	f	e	NaN	w	16.95	...	s
1	p	16.60	x	g	o	f	e	NaN	w	17.99	...	s
2	p	14.07	x	g	o	f	e	NaN	w	17.80	...	s
3	p	14.17	f	h	e	f	e	NaN	w	15.77	...	s
4	p	14.64	x	h	o	f	e	NaN	w	16.53	...	s

5 rows × 21 columns

In [ ]:
```
mushroomTrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50213 entries, 0 to 50212
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   class                 50213 non-null  object 
 1   cap-diameter          50213 non-null  float64
 2   cap-shape             50213 non-null  object 
 3   cap-surface           37915 non-null  object 
 4   cap-color             50213 non-null  object 
 5   does-bruise-or-bleed  50213 non-null  object 
 6   gill-attachment       42447 non-null  object 
 7   gill-spacing          31064 non-null  object 
 8   gill-color            50213 non-null  object 
 9   stem-height           50213 non-null  float64
 10  stem-width            50213 non-null  float64
 11  stem-root             7413 non-null   object 
 12  stem-surface          19912 non-null  object 
 13  stem-color            50213 non-null  object 
 14  veil-type             3177 non-null   object 
 15  veil-color            6297 non-null   object 
 16  has-ring              50213 non-null  object 
 17  ring-type             48448 non-null  object 
 18  spore-print-color     4532 non-null   object 
 19  habitat               50213 non-null  object 
 20  season                50213 non-null  object 
dtypes: float64(3), object(18)
memory usage: 8.0+ MB
```

In [ ]:
```
mushroomTest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10856 entries, 0 to 10855
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   class                 10856 non-null  object 
 1   cap-diameter          10856 non-null  float64
 2   cap-shape             10856 non-null  object 
 3   cap-surface           9034 non-null   object 
 4   cap-color             10856 non-null  object 
 5   does-bruise-or-bleed  10856 non-null  object 
 6   gill-attachment       8738 non-null   object 
 7   gill-spacing          4942 non-null   object 
 8   gill-color            10856 non-null  object 
 9   stem-height           10856 non-null  float64
 10  stem-width            10856 non-null  float64
 11  stem-root             2118 non-null   object 
 12  stem-surface          3033 non-null   object 
 13  stem-color            10856 non-null  object 
 14  veil-type             0 non-null      float64
 15  veil-color            1116 non-null   object 
 16  has-ring              10856 non-null  object 
 17  ring-type             10150 non-null  object 
 18  spore-print-color     1822 non-null   object 
 19  habitat               10856 non-null  object 
 20  season                10856 non-null  object 
dtypes: float64(4), object(17)
memory usage: 1.7+ MB
```

In [ ]:
```
mushroomTrain["class"].head()
```

Out[ ]:
```
0    p
1    p
2    p
3    p
4    p
Name: class, dtype: object
```

2. Splitting Data into Features and Labels

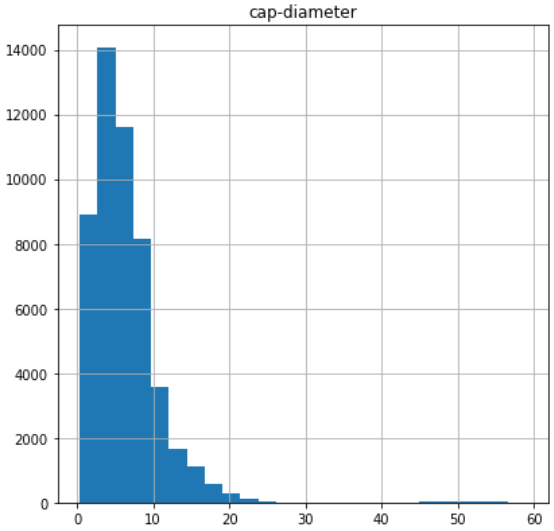
In [2]:
```
# Only keep features that shroomster can detect!

# Since we are trying to tell whether a mushroom is poisonnous or not, we want
to remove the label named "class" that tells whether the mushroom is poisonous
or edible
# If we leave this label in the data, then the model will just train based on
"class" and will perform poorly when actually presented with new data

mushroomTrain_copy = mushroomTrain["class"].copy()
mushroomTrain = mushroomTrain.drop("class", axis=1)

mushroomTest_copy = mushroomTest["class"].copy()
mushroomTest = mushroomTest.drop("class", axis=1)
```

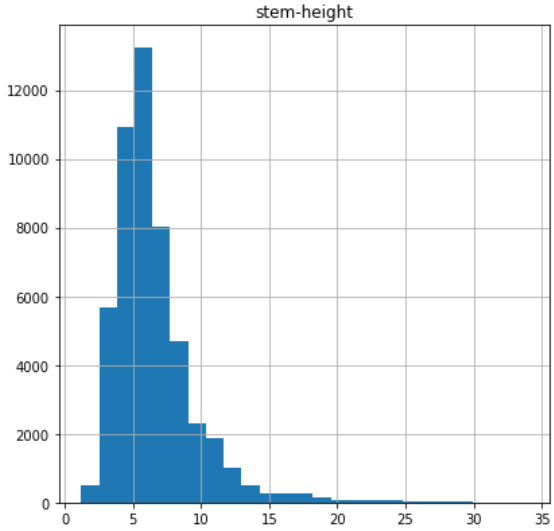
In [5]:
```
mushroomTrain_copy.head()
# mushroomTrain.head(1000)
```

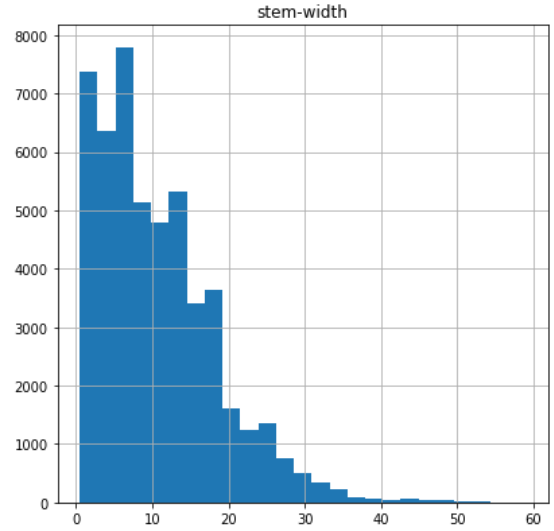
Out[5]:
```
0    p
1    p
2    p
3    p
4    p
Name: class, dtype: object
```

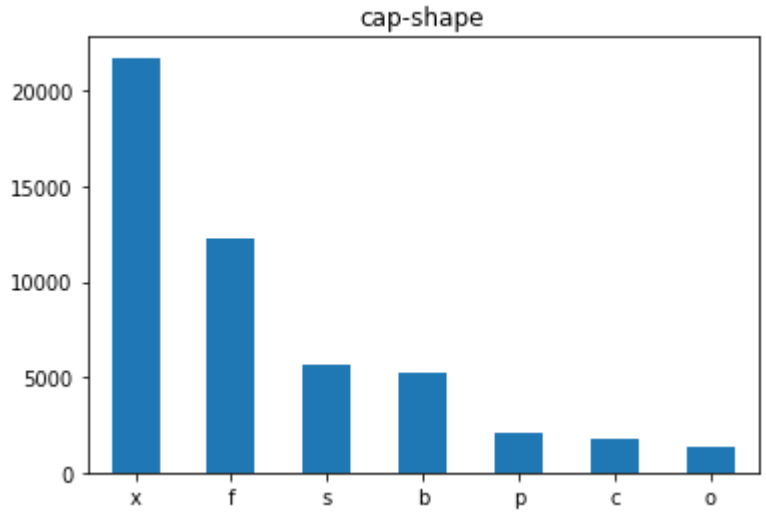
3. Data Exploration and Visualization

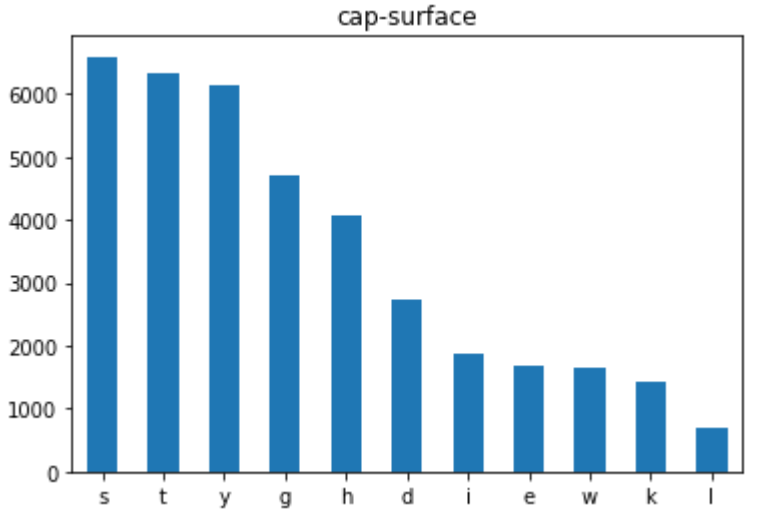
In [3]:
```
# Hint: We have done this in both project 1 & 2

%matplotlib inline
import matplotlib.pyplot as plt #plotting package

mushroomTrain.hist(bins=25, figsize=(15, 15))
plt.show()

# plots for the categorical data, since they don't get plotted automatically w
ith the hist() function
categorical = ['cap-shape', 'cap-surface', 'cap-color', 'gill-attachment', 'gi
ll-spacing', 'gill-color', 'stem-root', 'stem-surface', 'stem-color', 'veil-ty
pe', 'veil-color', 'has-ring', 'ring-type', 'spore-print-color', 'habitat', 's
eason']
for i in categorical:
  mushroomTrain[i].value_counts().plot.bar(rot=0, title=i)
  plt.show()
```

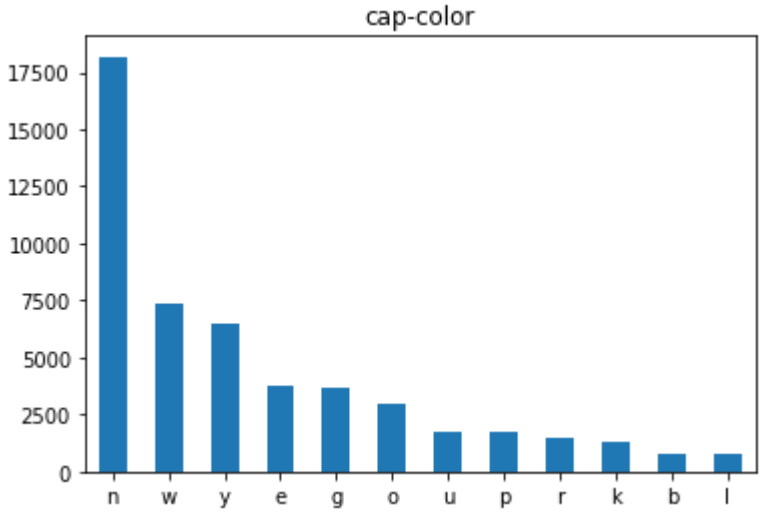

cap-diameter
0
2000
4000
6000
8000
10000
12000
14000
0
10
20
30
40
50
60


stem-height
0
2000
4000
6000
8000
10000
12000
0
5
10
15
20
25
30
35

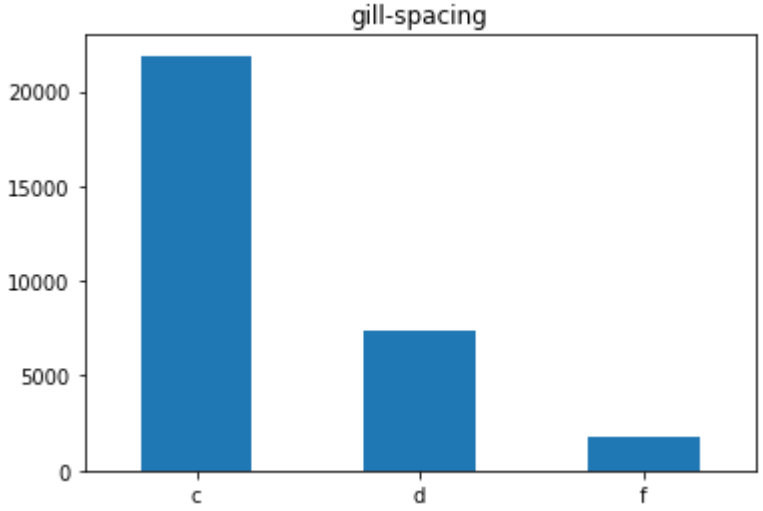

stem-width
0
1000
2000
3000
4000
5000
6000
7000
8000
0
10
20
30
40
50
60

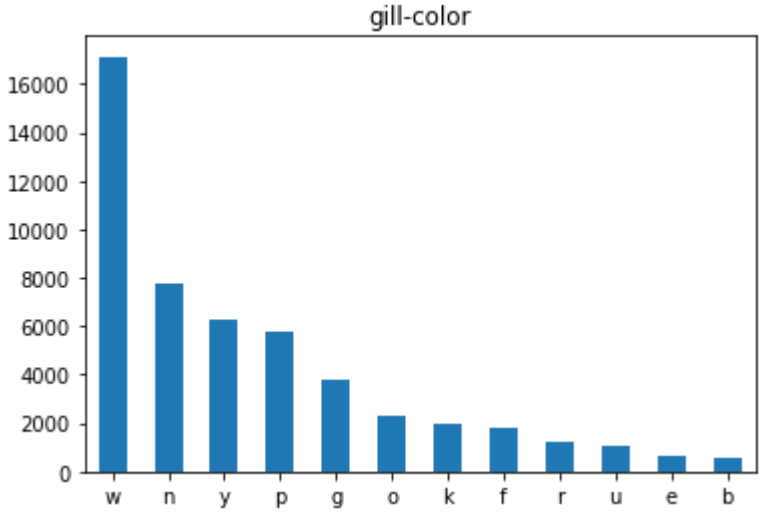

cap-shape
0
5000
10000
15000
20000
x
f
s
b
p
c
o

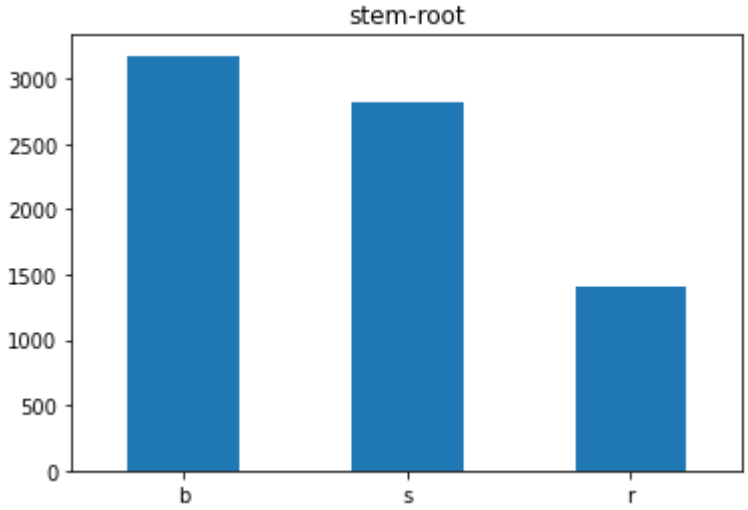

cap-surface
6000
5000
4000
3000
2000
1000
0
s t y g h d i e w k l

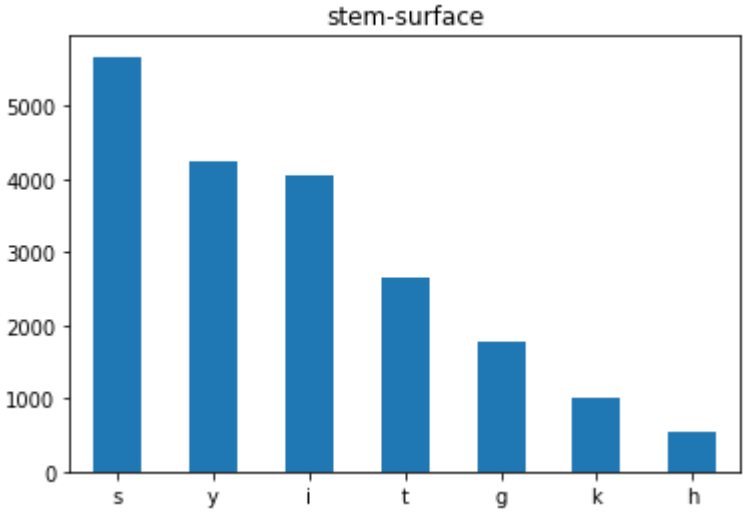

cap-color
17500
15000
12500
10000
7500
5000
2500
0
n w y e g o u p r k b l

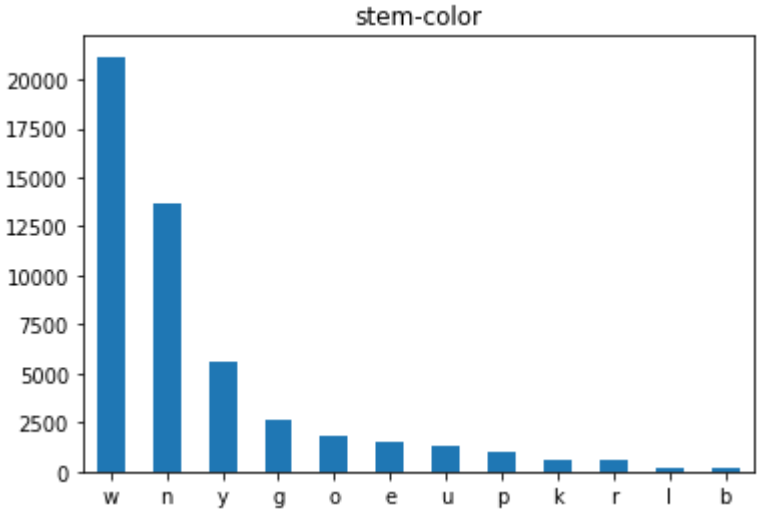

gill-attachment
12000
10000
8000
6000
4000
2000
0
a d x e s f p

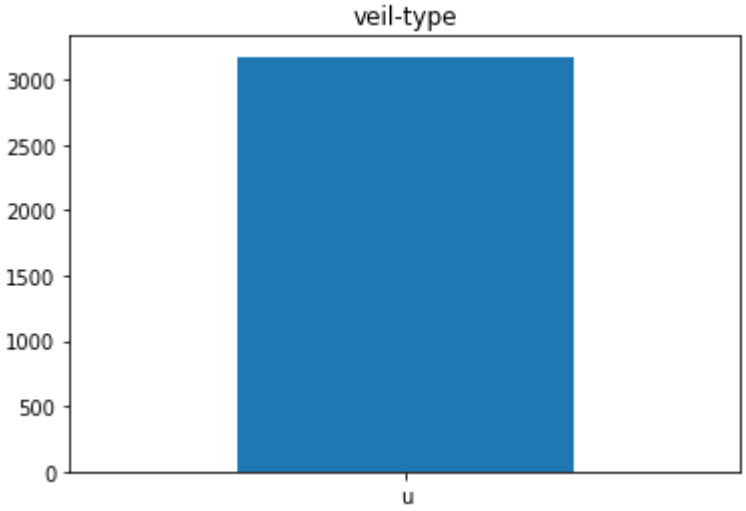

gill-spacing
20000
15000
10000
5000
0
c
d
f

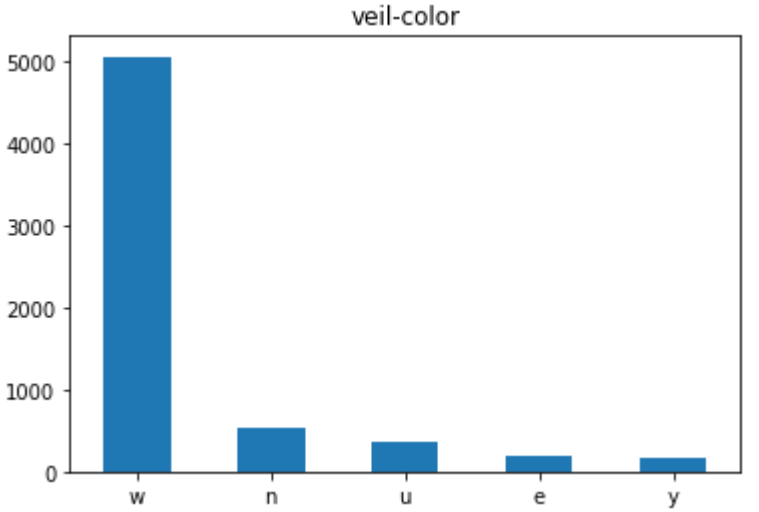

gill-color
16000
14000
12000
10000
8000
6000
4000
2000
0
w
n
y
p
g
o
k
f
r
u
e
b

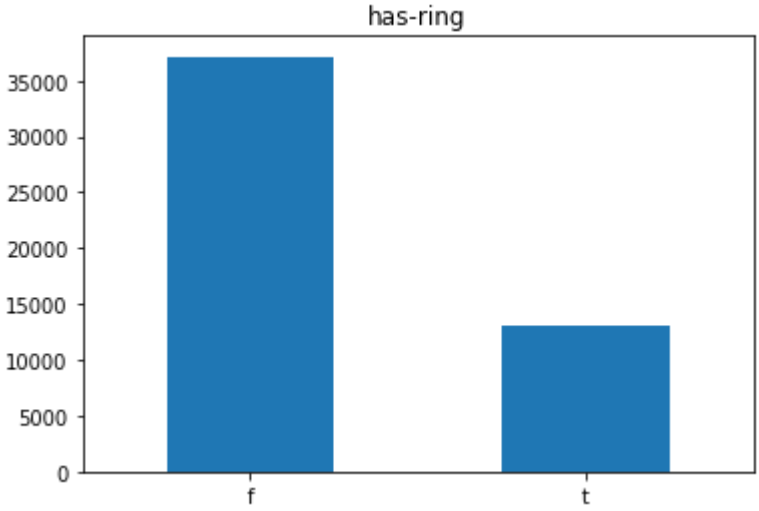

stem-root
3000
2500
2000
1500
1000
500
0
b
s
r

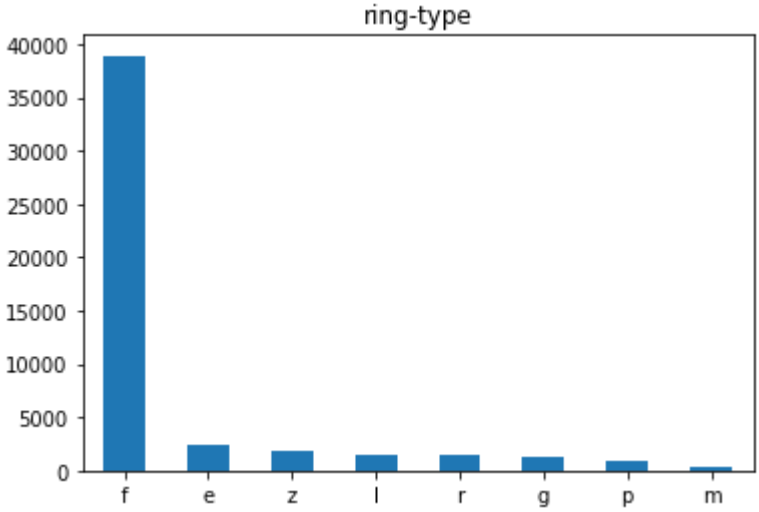

stem-surface
5000
4000
3000
2000
1000
0
s
y
i
t
g
k
h

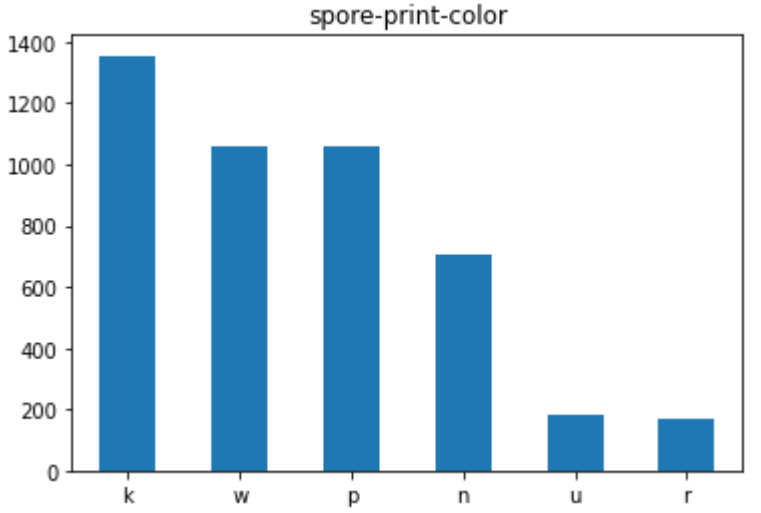

stem-color
20000
17500
15000
12500
10000
7500
5000
2500
0
w
n
y
g
o
e
u
p
k
r
l
b

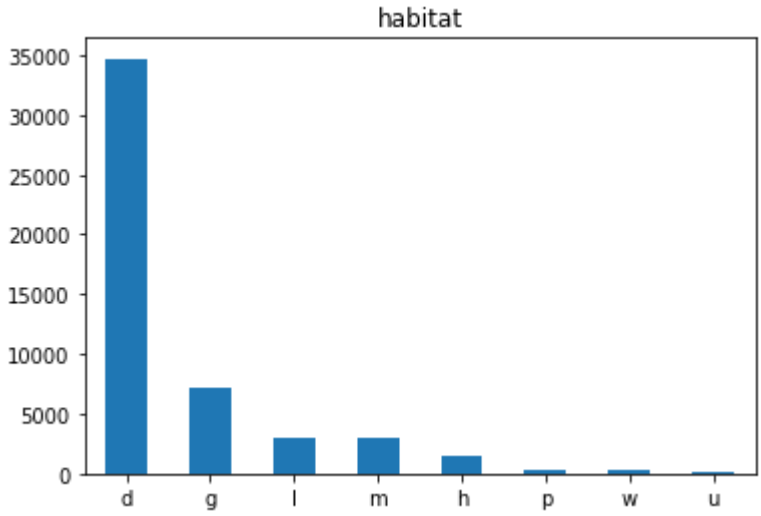

veil-type
3000
2500
2000
1500
1000
500
0
u

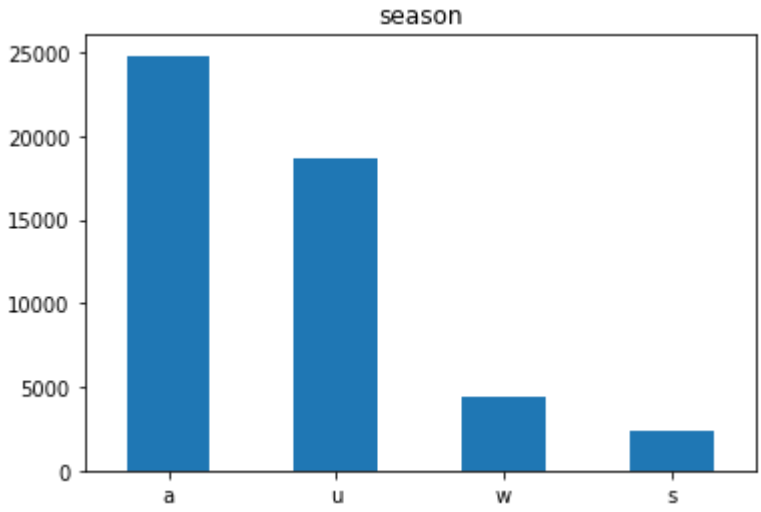

veil-color
5000
4000
3000
2000
1000
0
w
n
u
e
y


has-ring
35000
30000
25000
20000
15000
10000
5000
0
f
t


ring-type
40000
35000
30000
25000
20000
15000
10000
5000
0
f
e
z
l
r
g
p
m







4. Data Processing

In [4]:
```
#2
# all the mushrooms fall into the same category for 'veil-type', doesn't reall
y change anything
mushroomTrain = mushroomTrain.drop(["veil-type"], axis=1)

mushroomTest = mushroomTest.drop(["veil-type"], axis=1)
```

In [5]:
```
#3
# Imputing the Data
# Strategy: Replace NaN with the most common value for that column

mushroomTrain['cap-surface'] = mushroomTrain['cap-surface'].fillna(mushroomTra
in['cap-surface'].value_counts().index[0])
mushroomTrain['gill-attachment'] = mushroomTrain['gill-attachment'].fillna(mus
hroomTrain['gill-attachment'].value_counts().index[0])
mushroomTrain['gill-spacing'] = mushroomTrain['gill-spacing'].fillna(mushroomT
rain['gill-spacing'].value_counts().index[0])
mushroomTrain['stem-root'] = mushroomTrain['stem-root'].fillna(mushroomTrain
['stem-root'].value_counts().index[0])
mushroomTrain['stem-surface'] = mushroomTrain['stem-surface'].fillna(mushroomT
rain['stem-surface'].value_counts().index[0])
mushroomTrain['veil-color'] = mushroomTrain['veil-color'].fillna(mushroomTrain
['veil-color'].value_counts().index[0])
mushroomTrain['ring-type'] = mushroomTrain['ring-type'].fillna(mushroomTrain
['ring-type'].value_counts().index[0])
mushroomTrain['spore-print-color'] = mushroomTrain['spore-print-color'].fillna
(mushroomTrain['spore-print-color'].value_counts().index[0])

mushroomTest['cap-surface'] = mushroomTest['cap-surface'].fillna(mushroomTest
['cap-surface'].value_counts().index[0])
mushroomTest['gill-attachment'] = mushroomTest['gill-attachment'].fillna(mushr
oomTest['gill-attachment'].value_counts().index[0])
mushroomTest['gill-spacing'] = mushroomTest['gill-spacing'].fillna(mushroomTes
t['gill-spacing'].value_counts().index[0])
mushroomTest['stem-root'] = mushroomTest['stem-root'].fillna(mushroomTest['ste
m-root'].value_counts().index[0])
mushroomTest['stem-surface'] = mushroomTest['stem-surface'].fillna(mushroomTes
t['stem-surface'].value_counts().index[0])
mushroomTest['veil-color'] = mushroomTest['veil-color'].fillna(mushroomTest['v
eil-color'].value_counts().index[0])
mushroomTest['ring-type'] = mushroomTest['ring-type'].fillna(mushroomTest['rin
g-type'].value_counts().index[0])
mushroomTest['spore-print-color'] = mushroomTest['spore-print-color'].fillna(m
ushroomTest['spore-print-color'].value_counts().index[0])

sample_incomplete_rows = mushroomTest[mushroomTrain.isnull().any(axis=1)]
sample_incomplete_rows
```

```
<ipython-input-5-0e89ff351981>:23: UserWarning: Boolean Series key will be re
indexed to match DataFrame index.
  sample_incomplete_rows = mushroomTest[mushroomTrain.isnull().any(axis=1)]
```

Out[5]:

cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	stem-width	stem-root	st sur

In [6]:
```
# Hints:
# 1. Convert the "class" column into labels: 'p' (poisonous) -> 0,  'e' (edibl
e) -> 1
# 2. You can drop columns if you see fit
# 3. See any imcomplete data? We learned how to deal with them in project 1.

# pipeline, proj 2

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer

# x_train, x_test, y_train, y_test = train_test_split(mushroomTrain, mushroomT
rain_copy, test_size=0.2)
# x_train = features of the training data
# x_test = labels of the training data


#1
# poisonous = 0   edible = 1
mushroomTrain_copy.replace(['p', 'e'], [0,1], inplace=True)
mushroomTest_copy.replace(['p', 'e'], [0,1], inplace=True)

mushroomTrainDrop = mushroomTrain.drop(['cap-shape', 'does-bruise-or-bleed',
'cap-surface', 'cap-color', 'gill-attachment', 'gill-spacing', 'gill-color',
'stem-root', 'stem-surface', 'stem-color',  'veil-color', 'has-ring', 'ring-ty
pe', 'spore-print-color', 'habitat', 'season'], axis=1)
categories = ['cap-shape', 'cap-surface', 'does-bruise-or-bleed', 'cap-color',
'gill-attachment', 'gill-spacing', 'gill-color', 'stem-root', 'stem-surface',
'stem-color', 'veil-color', 'has-ring', 'ring-type', 'spore-print-color', 'hab
itat', 'season']
numbers = list(mushroomTrainDrop)

mushroomTestDrop = mushroomTest.drop(['cap-shape', 'does-bruise-or-bleed', 'ca
p-surface', 'cap-color', 'gill-attachment', 'gill-spacing', 'gill-color', 'ste
m-root', 'stem-surface', 'stem-color',  'veil-color', 'has-ring', 'ring-type',
'spore-print-color', 'habitat', 'season'], axis=1)
numbers2 = list(mushroomTestDrop)

# if the matrix is too big and affecting performance, can try dropping some fe
atures - or change the binary classifications like how I did with the poisonou
s/edible - onehotencoding can be big
pipeline = ColumnTransformer([("num", StandardScaler(), numbers), ("cat", OneH
otEncoder(), categories)])

pipeline2 = ColumnTransformer([("num", StandardScaler(), numbers2), ("cat", On
eHotEncoder(), categories)])

# x_train - features of the training data
x_train = pipeline.fit_transform(mushroomTrain).toarray()
# x_test - labels of the training data
x_test = pipeline.fit_transform(mushroomTest).toarray()
# y_train - features of the training data
y_train = mushroomTrain_copy
```

```
# y_test - labels of the training data
y_test = mushroomTest_copy
```

In [7]:
```
# Hints:
# 1. Convert the "class" column into labels: 'p' (poisonous) -> 0,  'e' (edibl
e) -> 1
# 2. You can drop columns if you see fit
# 3. See any imcomplete data? We learned how to deal with them in project 1.

# pipeline, proj 2

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

data = mushroomTrain.append(mushroomTest, ignore_index=True)
data_copy = mushroomTrain_copy.append(mushroomTest_copy, ignore_index=True)
# do processing
x_train, x_test, y_train, y_test = train_test_split(data, data_copy, test_size
=0.2)
# note train set size is 50213, so we can use that to split the data again
train_df = data.iloc[:50213, :]
test_df = data.iloc[50213:, :]

#1

# poisonous = 0   edible = 1
mushroomTrain_copy.replace(['p', 'e'], [0,1], inplace=True)
mushroomTest_copy.replace(['p', 'e'], [0,1], inplace=True)

# mushroom = mushroomTrain.append(mushroomTest)
# mushroom_copy = mushroomTrain_copy.append(mushroomTest_copy)
# x_train, x_test, y_train, y_test = train_test_split(mushroom, mushroom_copy,
test_size=0.2)

xTrainDrop = x_train.drop(['cap-shape', 'does-bruise-or-bleed', 'cap-surface',
'cap-color', 'gill-attachment', 'gill-spacing', 'gill-color', 'stem-root', 'st
em-surface', 'stem-color',  'veil-color', 'has-ring', 'ring-type', 'spore-prin
t-color', 'habitat', 'season'], axis=1)
categories = ['cap-shape', 'cap-surface', 'does-bruise-or-bleed', 'cap-color',
'gill-attachment', 'gill-spacing', 'gill-color', 'stem-root', 'stem-surface',
'stem-color', 'veil-color', 'has-ring', 'ring-type', 'spore-print-color', 'hab
itat', 'season']
numbers = list(xTrainDrop)

# if the matrix is too big and affecting performance, can try dropping some fe
atures - or change the binary classifications like how I did with the poisonou
s/edible - onehotencoding can be big
pipeline = ColumnTransformer([("num", StandardScaler(), numbers), ("cat", OneH
otEncoder(sparse=False), categories)])

x_train_pipelined = pipeline.fit_transform(x_train)
x_test_pipelined = pipeline.transform(x_test)
x_train_pipelined
```

Out[7]:
```
array([[-0.08331829,  1.21915959,  0.16849264, ...,  0.        ,
         1.        ,  0.        ],
       [-0.19219895, -0.02985931, -0.43581394, ...,  0.        ,
         1.        ,  0.        ],
       [-1.11290908, -0.99636204, -0.95649324, ...,  0.        ,
         1.        ,  0.        ],
       ...,
       [ 0.00837069, -0.09825797, -0.38006242, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.09623929,  1.99533562, -0.4019648 , ...,  0.        ,
         1.        ,  0.        ],
       [-0.40040933, -0.14881349, -0.79421652, ...,  0.        ,
         0.        ,  1.        ]])
```

5. Data Augmentation (Creating at least 2 New Features)

In [10]:
```
# Similar to Project 1 and 2.

mushroomTrain['stem-size'] = mushroomTrain['stem-height'] * mushroomTrain['ste
m-width']
mushroomTrain['mushroom-size'] = (mushroomTrain['stem-height'] * mushroomTrain
['stem-width']) + mushroomTrain['cap-diameter']
```

6. Logistic Regression & Statistical Hypothesis Testing

In [48]:
```
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test = train_test_split(train_df, mushroomTrain_
copy, test_size=0.2)

logisticReg1 = LogisticRegression(max_iter=800)
logisticReg1.fit(x_train_pipelined, y_train)
y_predict = logisticReg1.predict(x_test_pipelined)
y_train_predict = logisticReg1.predict(x_train_pipelined)

print("Training Data \n")
print(classification_report(y_train, y_train_predict))

print("Test Data \n")
print(classification_report(y_test, y_predict))
```

```
Training Data

              precision    recall  f1-score   support

           0       0.86      0.84      0.85     27206
           1       0.81      0.83      0.82     21649

    accuracy                           0.84     48855
   macro avg       0.84      0.84      0.84     48855
weighted avg       0.84      0.84      0.84     48855

Test Data

              precision    recall  f1-score   support

           0       0.86      0.84      0.85      6682
           1       0.81      0.83      0.82      5532

    accuracy                           0.84     12214
   macro avg       0.83      0.84      0.83     12214
weighted avg       0.84      0.84      0.84     12214
```

In [12]:
```
stats_data = mushroomTrain[["cap-diameter", "stem-height", "stem-width", "stem
-size", "mushroom-size"]]
stats_data.head()
```

Out[12]:

	cap-diameter	stem-height	stem-width	stem-size	mushroom-size
0	15.26	16.95	17.09	289.6755	304.9355
1	16.60	17.99	18.19	327.2381	343.8381
2	14.07	17.80	17.74	315.7720	329.8420
3	14.17	15.77	15.98	252.0046	266.1746
4	14.64	16.53	17.20	284.3160	298.9560

In [13]:
```
stats_label = mushroomTrain_copy
stats_label.head()
```

Out[13]:
```
0    0
1    0
2    0
3    0
4    0
Name: class, dtype: int64
```

In [14]:
```
import statsmodels.api as sm

sm_x = sm.add_constant(stats_data)
mushroom_stats = sm.OLS(stats_label, sm_x)
results_stats = mushroom_stats.fit()
print(results_stats.summary())
```

OLS Regression Results

Dep. Variable:	class	R-squared:	0.027
Model:	OLS	Adj. R-squared:	0.027
Method:	Least Squares	F-statistic:	353.8
Date:	Mon, 27 Feb 2023	Prob (F-statistic):	6.04e-301
Time:	22:00:53	Log-Likelihood:	-35056.
No. Observations:	50213	AIC:	7.012e+04
Df Residuals:	50208	BIC:	7.017e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2725	0.011	25.742	0.000	0.252	0.293
cap-diameter	0.0118	0.001	22.044	0.000	0.011	0.013
stem-height	0.0059	0.002	3.309	0.001	0.002	0.009
stem-width	0.0023	0.001	2.880	0.004	0.001	0.004
stem-size	-0.0061	0.000	-21.799	0.000	-0.007	-0.006
mushroom-size	0.0057	0.000	21.369	0.000	0.005	0.006

Omnibus:	197077.282	Durbin-Watson:	0.008
Prob(Omnibus):	0.000	Jarque-Bera (JB):	7692.951
Skew:	0.330	Prob(JB):	0.00
Kurtosis:	1.199	Cond. No.	7.68e+15

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.35e-23. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

```
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/tsatools.py:142: Futur
eWarning: In a future version of pandas all arguments of concat except for th
e argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

I performed the OLS based on five features: cap-diameter, stem-height, stem-width, (these three were provided in the dataset) stem-size, and mushroom-size (these were two new features that I created). Performing the OLS on these features yields P-values all less than the standard p-value threshold of 0.05. This means that there is a significant relationship and we can reject the null hypothesis that there is no relationship between these features and whether the mushroom is poisonous or edible.

7. Dimensionality Reduction using PCA

In [15]:
```
# PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decompositio
n.PCA.html
```

In [ ]:
```
# mushroomTrainPipelined.shape
# mushroomTrain.shape
x_train.shape
```

Out[ ]:
```
(10856, 87)
```

In [9]:
```
from sklearn import decomposition
from sklearn import metrics
```

In [18]:
```
# First we create a PCA object with the 12 components as a parameter
pca = decomposition.PCA(n_components=5)

# Now we run the fit operation to convert our
# data to a PCA transformmed data
mushroomTrainPCA = pca.fit_transform(x_train_pipelined)

mushroomTrainPCA.shape
```

Out[18]:
```
(48855, 5)
```

In [ ]:
```
logisticReg = LogisticRegression(solver='liblinear')
logisticReg.fit(mushroomTrainPCA, y_train)
```

In [20]:
```
mushroomPCATest = pca.transform(x_test_pipelined)

mushroomPCATest.shape
predicted = logisticReg.predict(mushroomPCATest)

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test, predicted)))
```

```
Accuracy:    0.589651
```

8. Experiment with any 2 other models (Non-Ensemble)

In [ ]:
```
# Models: https://scikit-learn.org/stable/supervised_learning.html
```

In [21]:
```
# KNN -
from sklearn.neighbors import KNeighborsClassifier

# NOT USING PCA VALUES

# values = [1, 2, 3, 5, 7, 9, 10, 20, 50, 75, 100]
# for n in values:
#     knnclf = KNeighborsClassifier(n_neighbors=n)
#     knnclf.fit(x_train_pipelined, y_train)
#     y_predict = knnclf.predict(x_test_pipelined)
#     score = metrics.accuracy_score(y_test, y_predict, normalize=True)
#     print('Accuracy for %d-Nearest-Neighbor: %.3f' %(n, score))

# knnclf = KNeighborsClassifier(n_neighbors=2)
# knnclf.fit(x_train_pipelined, y_train)
# y_predict = knnclf.predict(x_test_pipelined)
# metrics.accuracy_score(y_test, y_predict, normalize=True)


# USING PCA VALUES
# Note: may be a bit wonky because there are a lot of features, so try with PC
A (reduced dimensions works better with KNN)

knnclf = KNeighborsClassifier(n_neighbors=5)
knnclf.fit(mushroomTrainPCA, y_train)
y_predict = knnclf.predict(mushroomPCATest)
metrics.accuracy_score(y_test, y_predict, normalize=True)
```

```
Accuracy for 1-Nearest-Neighbor: 0.986
Accuracy for 2-Nearest-Neighbor: 0.980
Accuracy for 3-Nearest-Neighbor: 0.982
Accuracy for 5-Nearest-Neighbor: 0.980
Accuracy for 7-Nearest-Neighbor: 0.978
Accuracy for 9-Nearest-Neighbor: 0.975
Accuracy for 10-Nearest-Neighbor: 0.974
Accuracy for 20-Nearest-Neighbor: 0.967
Accuracy for 50-Nearest-Neighbor: 0.954
Accuracy for 75-Nearest-Neighbor: 0.944
Accuracy for 100-Nearest-Neighbor: 0.938
```

In [22]:
```
# SVM
from sklearn import metrics
from sklearn.svm import SVC
from sklearn import svm

# svm_classifier = svm.SVC(probability=True)
# svm_classifier.fit(x_train_pipelined, y_train)

# svm_predict = svm_classifier.predict(x_test_pipelined)

# accuracy = metrics.accuracy_score(y_test, svm_predict)
# precision = metrics.precision_score(y_test, svm_predict)
# recall = metrics.recall_score(y_test, svm_predict)
# f1 = metrics.f1_score(y_test, svm_predict)

# print("Accuracy: {}".format(accuracy))
# print("Precision: {}".format(precision))
# print("Recall: {}".format(recall))
# print("F1: {}".format(f1))


svm_classifier = svm.SVC(probability=True)
svm_classifier.fit(mushroomTrainPCA, y_train)

svm_predict = svm_classifier.predict(mushroomPCATest)

accuracy = metrics.accuracy_score(y_test, svm_predict)
precision = metrics.precision_score(y_test, svm_predict)
recall = metrics.recall_score(y_test, svm_predict)
f1 = metrics.f1_score(y_test, svm_predict)

print("Accuracy: {}".format(accuracy))
print("Precision: {}".format(precision))
print("Recall: {}".format(recall))
print("F1: {}".format(f1))
```

```
Accuracy: 0.8186507286720157
Precision: 0.8105223740872496
Recall: 0.7825379609544468
F1: 0.7962843741377724
```

9. Experiment with 1 Ensemble Method

In [ ]:
```
# Ensemble Methods: https://scikit-learn.org/stable/modules/ensemble.html
```

In [23]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error,
explained_variance_score
import numpy as np
```

In [25]:
```
clf = RandomForestRegressor(n_estimators=5)
clf = clf.fit(x_train_pipelined,y_train)
y_pred = clf.predict(x_test_pipelined)

variance = metrics.explained_variance_score(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
mae = metrics.mean_absolute_error(y_test, y_pred)
# accuracy = metrics.accuracy_score(y_test, y_pred)

#also add the accuracy
print('explained_variance: ', round(variance,4))
print('r2: ', round(r2,4))
print('MSE: ', round(mse,4))
print('MAE: ', round(mae,4))
print('RMSE: ', round(np.sqrt(mse),4))
```

```
explained_variance:  0.9972
r2:  0.9972
MSE:  0.0007
MAE:  0.0014
RMSE:  0.0262
```

10. Cross-Validation & Hyperparameter Tuning for All 3 Models

In [43]:
```
# Cross-Validation: https://scikit-learn.org/stable/modules/cross_validation.html
# Hyperparameter Tuning: https://scikit-learn.org/stable/modules/grid_search.html

from sklearn.model_selection import KFold
from sklearn import model_selection
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

In [34]:
```
kfold = KFold(n_splits=10, random_state=42, shuffle=True)
```

In [50]:
```
# logistic regression
scores = cross_val_score(logisticReg1, x_train_pipelined, y_train, cv=kfold)
np.mean(np.absolute(scores))
```

Out[50]: 0.8380104792545368

In [41]:
```
# knn
scores = cross_val_score(knnclf, x_train_pipelined, y_train, cv=kfold, n_jobs=-1)
np.mean(np.absolute(scores))
```

Out[41]: 0.9971343604499895

In [42]:
```
# svm
scores = cross_val_score(svm_classifier, x_train_pipelined, y_train, cv=kfold,
n_jobs=-1)
np.mean(np.absolute(scores))
```

Out[42]:
```
0.9986490551618875
```

In [46]:
```
# random forest
scores = cross_val_score(clf, x_train_pipelined, y_train, cv=kfold, n_jobs=-1)
np.mean(np.absolute(scores))
```

Out[46]:
```
0.9961579155542261
```

In [56]:
```
# logistic regression
parameters = {'solver':['newton-cg','liblinear','sag','saga'], 'max_iter':[10
0, 300, 500, 750]}

search = GridSearchCV(logisticReg1, parameters, scoring='accuracy', cv=kfold,
n_jobs=-1)
result = search.fit(x_train_pipelined, y_train)

print("Best Score: %s" % result.best_score_)
print("Best Hyperparameters: %s" % result.best_params_)
```

```
Best Score: 0.8380309542732961
Best Hyperparameters: {'max_iter': 100, 'solver': 'newton-cg'}
```

In [59]:
```
# KNN Hyperparameter tuning
# I didn't use GridSearch for this since it's just going to choose n=1, so I j
ust manually displayed results with different values of n
values = [1, 2, 3, 5, 7, 9, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100]
for n in values:
  knnclf = KNeighborsClassifier(n_neighbors=n)
  knnclf.fit(mushroomTrainPCA, y_train)
  y_predict = knnclf.predict(mushroomPCATest)
  score = metrics.accuracy_score(y_test, y_predict, normalize=True)
  print('Accuracy for %d-Nearest-Neighbor: %.3f' %(n, score))
```

```
Accuracy for 1-Nearest-Neighbor: 0.986
Accuracy for 2-Nearest-Neighbor: 0.980
Accuracy for 3-Nearest-Neighbor: 0.982
Accuracy for 5-Nearest-Neighbor: 0.980
Accuracy for 7-Nearest-Neighbor: 0.978
Accuracy for 9-Nearest-Neighbor: 0.975
Accuracy for 10-Nearest-Neighbor: 0.974
Accuracy for 15-Nearest-Neighbor: 0.970
Accuracy for 20-Nearest-Neighbor: 0.967
Accuracy for 30-Nearest-Neighbor: 0.962
Accuracy for 40-Nearest-Neighbor: 0.958
Accuracy for 50-Nearest-Neighbor: 0.954
Accuracy for 60-Nearest-Neighbor: 0.950
Accuracy for 70-Nearest-Neighbor: 0.945
Accuracy for 80-Nearest-Neighbor: 0.942
Accuracy for 90-Nearest-Neighbor: 0.940
Accuracy for 100-Nearest-Neighbor: 0.938
```

In [ ]:
```
# svm
parameters = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001],'kernel':
['rbf']}

search = GridSearchCV(svm_classifier, parameters, scoring='accuracy', cv=kfol
d, n_jobs=-1)
result = search.fit(x_train_pipelined, y_train)


print("Best Score: %s" % result.best_score_)
print("Best Hyperparameters: %s" % result.best_params_)
```

In [54]:
```
# random forest
parameters = {'max_features':['auto','sqrt'], 'max_depth':[10,20,30], 'min_sam
ples_split':[2,5,10], 'min_samples_leaf':[1,2,4]}

search = GridSearchCV(clf, parameters, cv=kfold, n_jobs=-1)
result = search.fit(x_train_pipelined, y_train)

print("Best Score: %s" % result.best_score_)
print("Best Hyperparameters: %s" % result.best_params_)
```

```
Best Score: 0.9961976275137614
Best Hyperparameters: {'max_depth': 20, 'max_features': 'auto', 'min_samples_
leaf': 1, 'min_samples_split': 2}
```

11. Report Final Results

In [ ]:
```
# e.g. Accuracy, Precision etc.
```