

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

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

```
In [4]: mushroomTest.head()
```

Out[4]:

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

Out[5]:

	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 [6]: 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 [7]: `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 [8]: `mushroomTrain["class"].head()`

```
Out[8]: 0    p
1    p
2    p
3    p
4    p
Name: class, dtype: object
```

2. Splitting Data into Features and Labels

```
In [9]: # Only keep features that shroomster can detect!

# Since we are trying to tell whether a mushroom is poisonous 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 [10]: mushroomTrain_copy.head()
# mushroomTrain.head(1000)
```

```
Out[10]: 0    p
1    p
2    p
3    p
4    p
Name: class, dtype: object
```

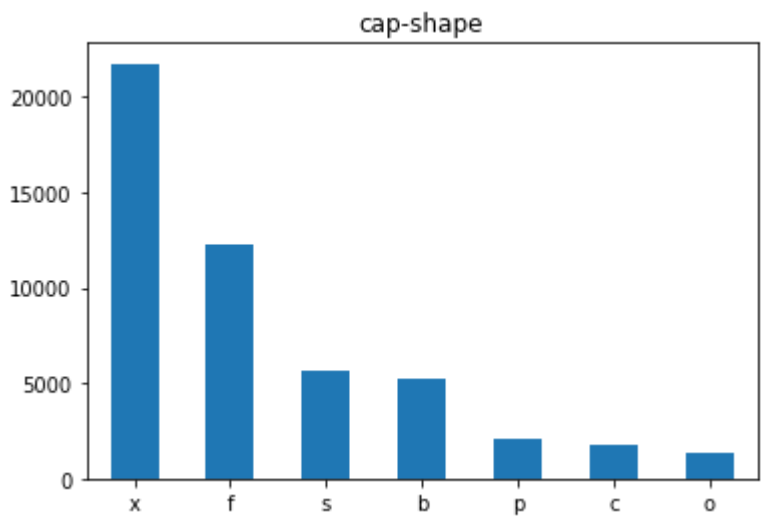
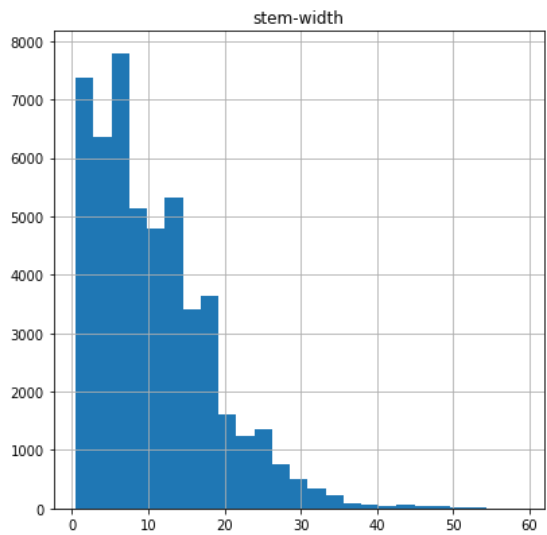
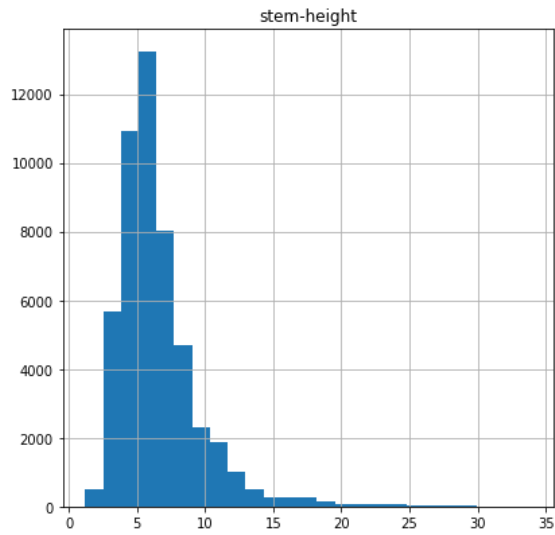
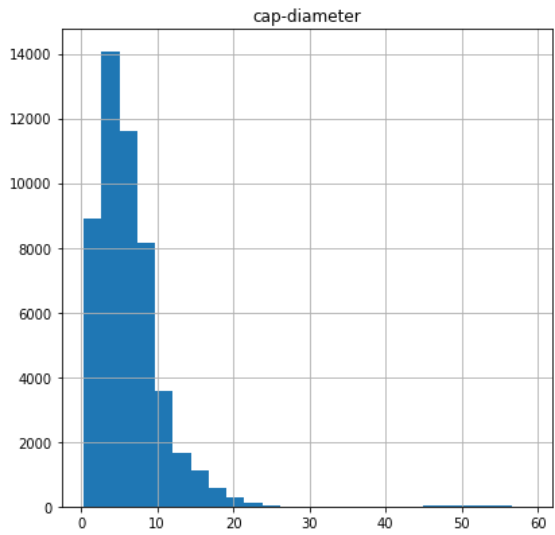
3. Data Exploration and Visualization

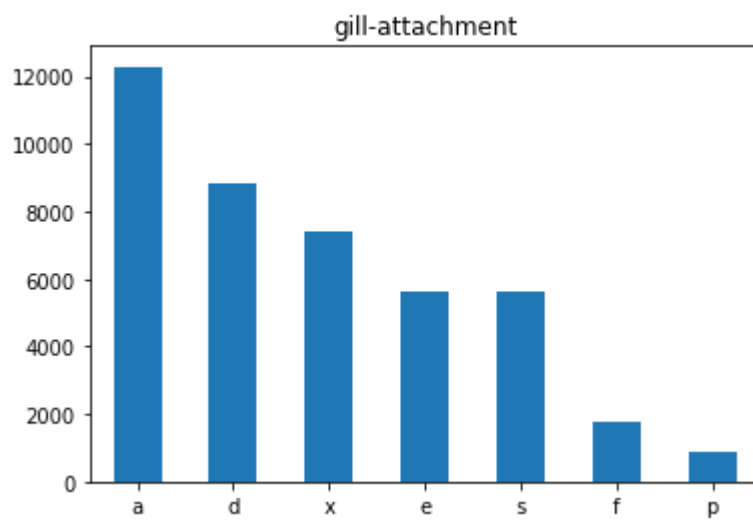
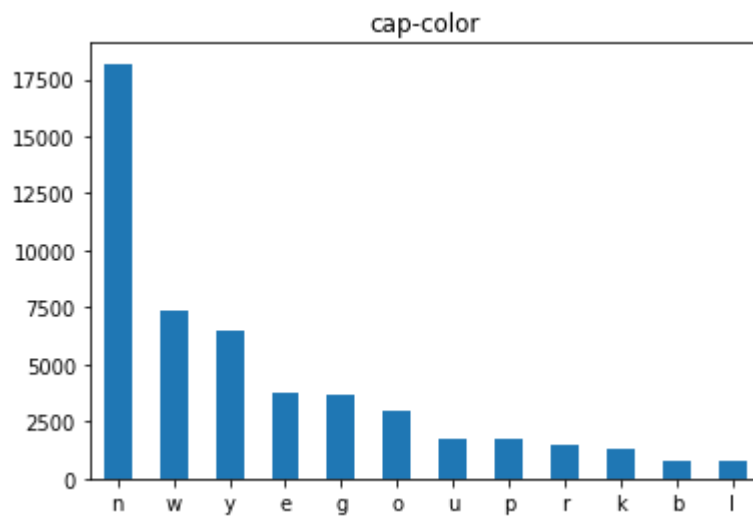
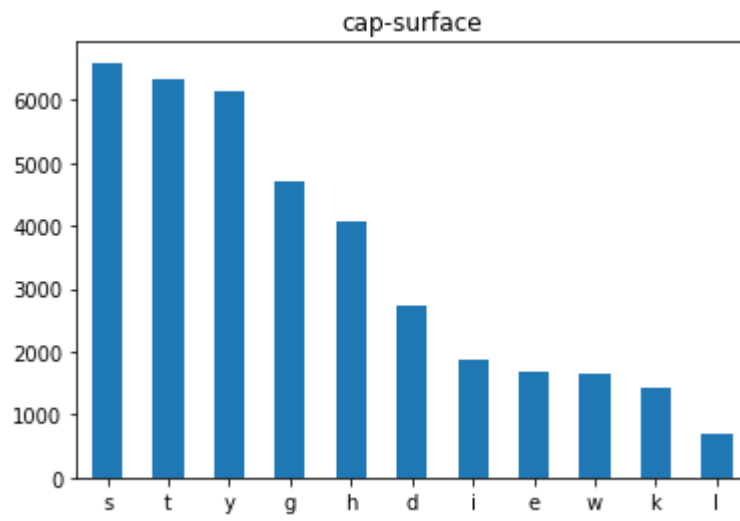
```
In [11]: # Hint: We have done this in both project 1 & 2

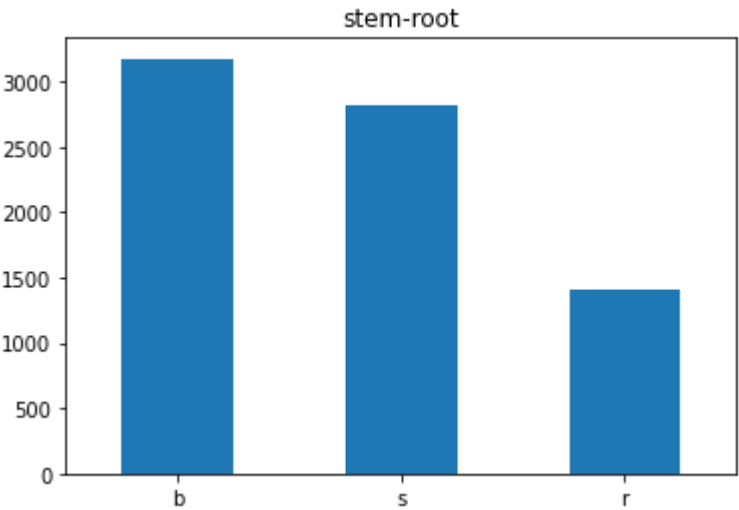
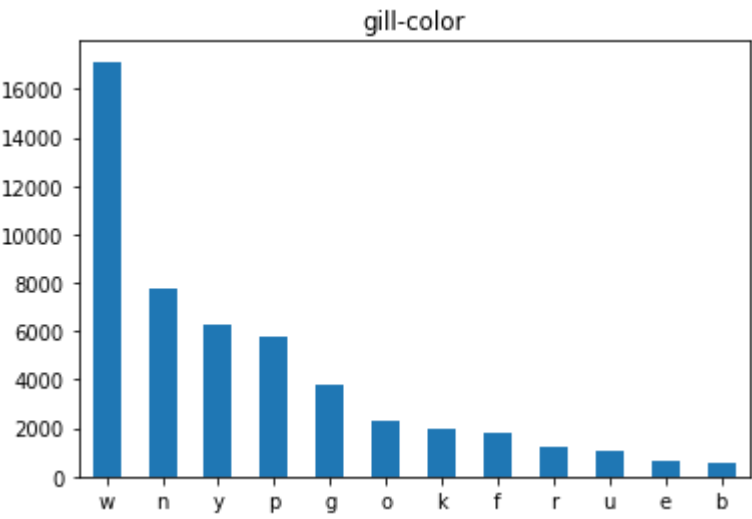
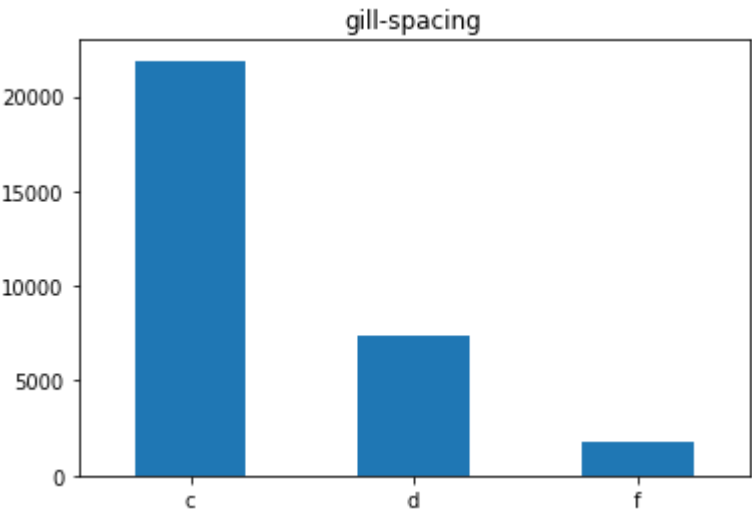
%matplotlib inline
import matplotlib.pyplot as plt #plotting package

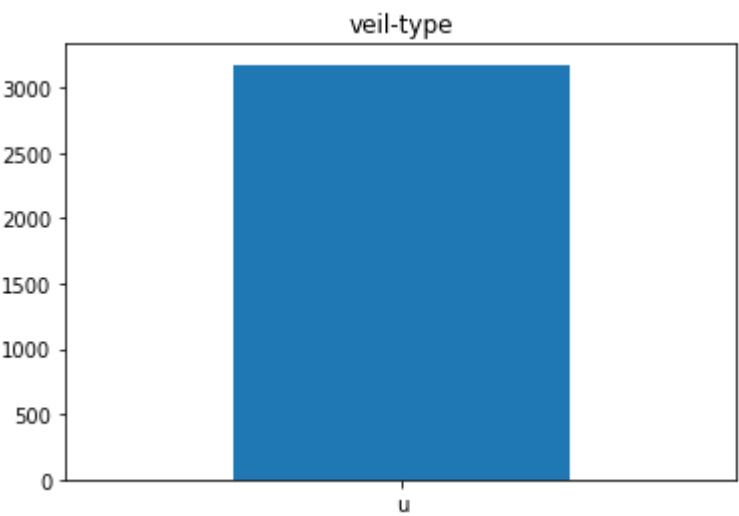
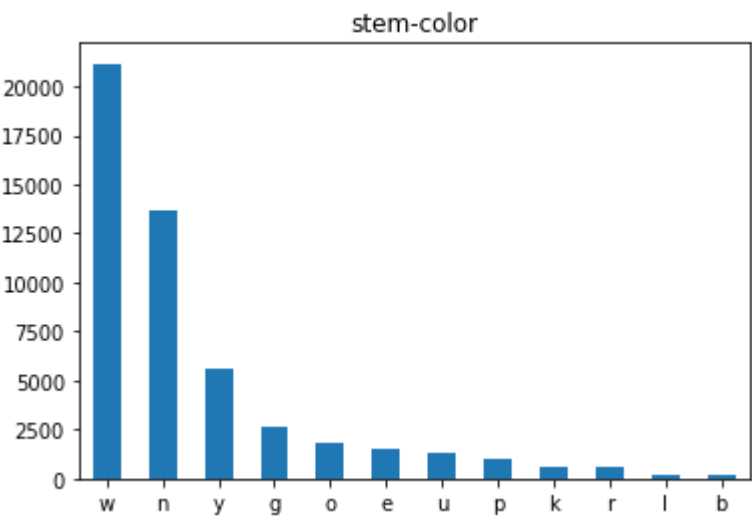
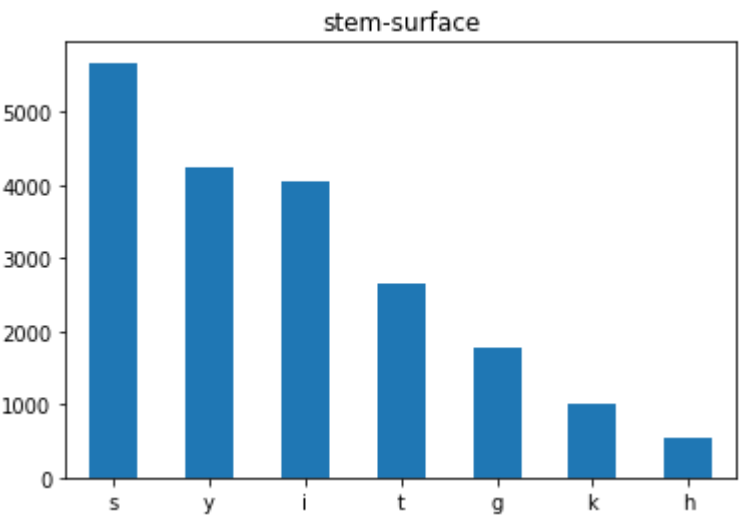
mushroomTrain.hist(bins=25, figsize=(15, 15))
plt.show()

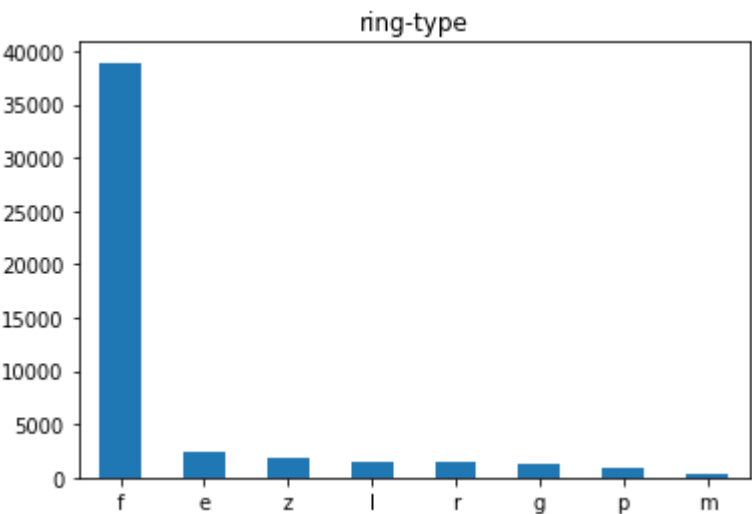
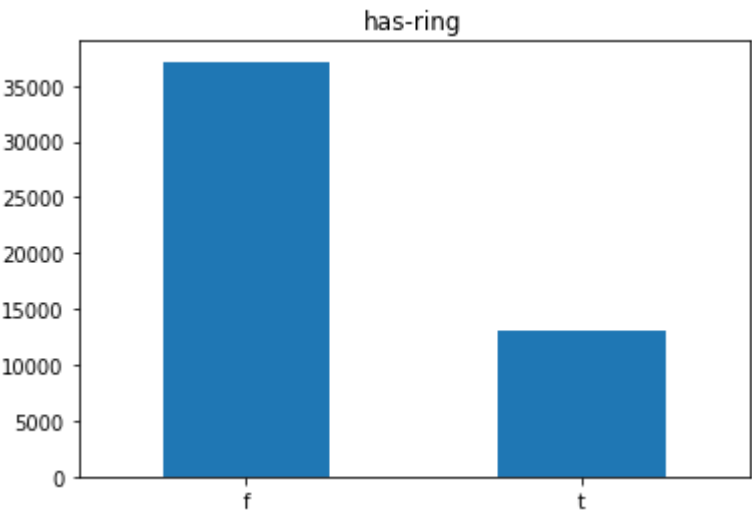
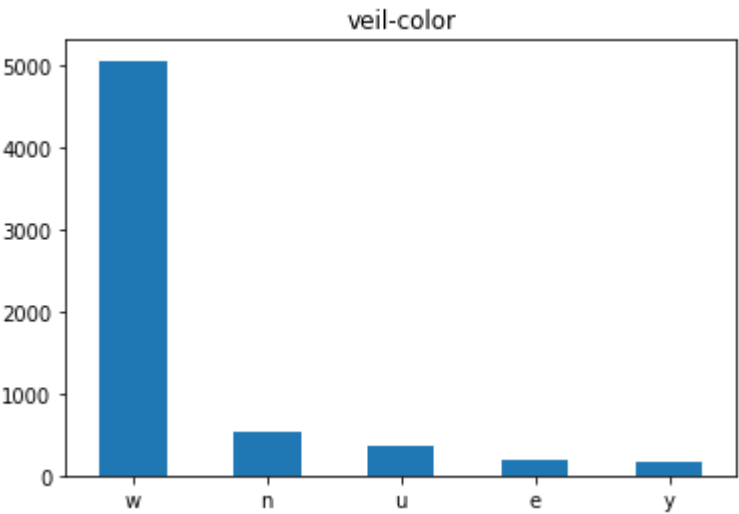
# plots for the categorical data, since they don't get plotted automatically w
ith the hist() function
categorical = ['cap-shape', 'cap-surface', 'cap-color', 'gill-attachment', 'gi
ll-spacing', 'gill-color', 'stem-root', 'stem-surface', 'stem-color', 'veil-ty
pe', 'veil-color', 'has-ring', 'ring-type',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

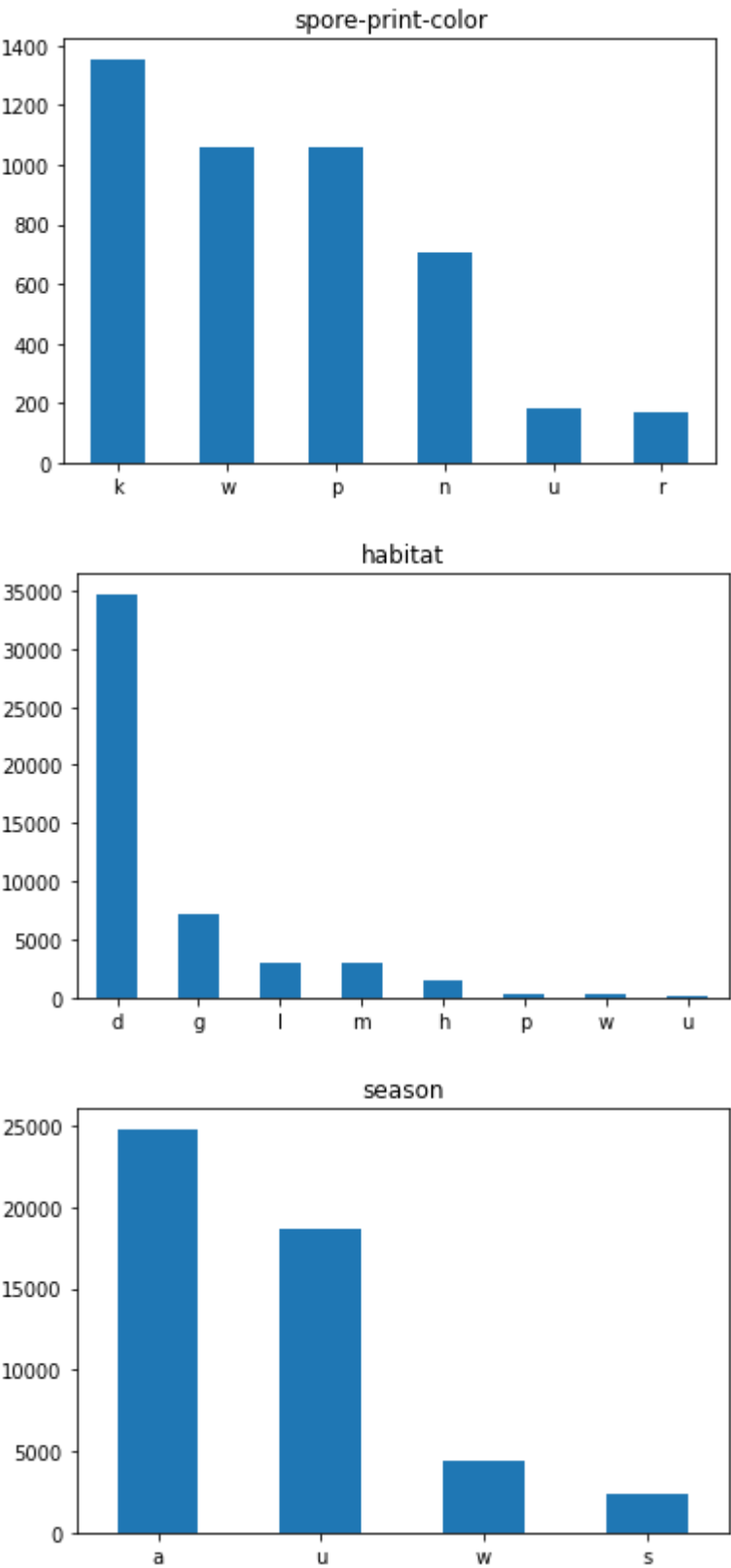












4. Data Processing

```
In [12]: #2
# all the mushrooms fall into the same category for 'veil-type', doesn't really change anything
mushroomTrain = mushroomTrain.drop(["veil-type"], axis=1)
# mushroomTest = mushroomTest.drop(["veil-type"], axis=1)
```

```

In [15]: #3
# Imputing the Data
# Strategy: Replace NaN with the most common value for that column

mushroomTrain['cap-surface'] = mushroomTrain['cap-surface'].fillna(mushroomTrain['cap-surface'].value_counts().index[0])
mushroomTrain['gill-attachment'] = mushroomTrain['gill-attachment'].fillna(mushroomTrain['gill-attachment'].value_counts().index[0])
mushroomTrain['gill-spacing'] = mushroomTrain['gill-spacing'].fillna(mushroomTrain['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(mushroomTrain['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(mushroomTest['gill-attachment'].value_counts().index[0])
mushroomTest['gill-spacing'] = mushroomTest['gill-spacing'].fillna(mushroomTest['gill-spacing'].value_counts().index[0])
mushroomTest['stem-root'] = mushroomTest['stem-root'].fillna(mushroomTest['stem-root'].value_counts().index[0])
mushroomTest['stem-surface'] = mushroomTest['stem-surface'].fillna(mushroomTest['stem-surface'].value_counts().index[0])
mushroomTest['veil-color'] = mushroomTest['veil-color'].fillna(mushroomTest['veil-color'].value_counts().index[0])
mushroomTest['ring-type'] = mushroomTest['ring-type'].fillna(mushroomTest['ring-type'].value_counts().index[0])
mushroomTest['spore-print-color'] = mushroomTest['spore-print-color'].fillna(mushroomTest['spore-print-color'].value_counts().index[0])

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

```

<ipython-input-15-0e89ff351981>:23: UserWarning: Boolean Series key will be reindexed to match DataFrame index.

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

Out[15]:

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

```

In [16]: # Hints:
# 1. Convert the "class" column into labels: 'p' (poisonous) -> 0, 'e' (edible) -> 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.

#1

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)
x_train, x_test, y_train, y_test = train_test_split(data, data_copy, test_size=0.2)
train_df = data.iloc[:50213, :]
test_df = data.iloc[50213:, :]

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

xTrainDrop = x_train.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-type', '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', 'habitat', 'season']
numbers = list(xTrainDrop)
pipeline = ColumnTransformer([("num", StandardScaler(), numbers), ("cat", OneHotEncoder(sparse=False), categories)])

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

```

```

Out[16]: array([[ -0.35793523,  0.18479233,  0.05892075, ...,  0.          ,
                1.          ,  0.          ],
               [ 0.58523253,  0.91328357,  0.11999599, ...,  0.          ,
                0.          ,  0.          ],
               [ 0.05850377, -0.02250193, -0.06423097, ...,  0.          ,
                0.          ,  0.          ],
               ...,
               [ 0.34944059, -0.23571888,  0.28419828, ...,  0.          ,
                1.          ,  0.          ],
               [ 0.71453779,  2.0030591 ,  0.8739248 , ...,  0.          ,
                1.          ,  0.          ],
               [-1.09953891, -1.23961535, -0.99738056, ...,  0.          ,
                0.          ,  0.          ]])

```

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

In [17]: *# Similar to Project 1 and 2.*

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

6. Logistic Regression & Statistical Hypothesis Testing


```
In [18]: 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	27028
1	0.81	0.83	0.82	21827
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	0.87	0.84	0.85	6860
1	0.80	0.83	0.82	5354
accuracy			0.84	12214
macro avg	0.83	0.84	0.83	12214
weighted avg	0.84	0.84	0.84	12214

```
In [19]: stats_data = mushroomTrain[["cap-diameter", "stem-height", "stem-width", "stem
-size", "mushroom-size"]]
stats_data.head()
```

Out[19]:

	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 [20]: stats_label = mushroomTrain_copy  
stats_label.head()
```

```
Out[20]: 0    0  
1    0  
2    0  
3    0  
4    0  
Name: class, dtype: int64
```

```
In [21]: 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:                0.02
7
Model:                  OLS      Adj. R-squared:          0.02
7
Method:                 Least Squares    F-statistic:          353.
8
Date:                   Tue, 28 Feb 2023    Prob (F-statistic):    6.04e-30
1
Time:                   05:10:11    Log-Likelihood:        -3505
6.
No. Observations:       50213    AIC:                   7.012e+0
4
Df Residuals:           50208    BIC:                   7.017e+0
4
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:          0.00
8
Prob(Omnibus):      0.000    Jarque-Bera (JB):       7692.95
1
Skew:               0.330    Prob(JB):               0.0
0
Kurtosis:           1.199    Cond. No.               7.68e+1
5
=====
=

```

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: FutureWarning: In a future version of pandas all arguments of concat except for the 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.decomposition.PCA.html

```

```

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

```

```

In [24]: # 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 transformed data
        mushroomTrainPCA = pca.fit_transform(x_train_pipelined)

```

```

In [25]: logisticReg = LogisticRegression(solver='liblinear')
        logisticReg.fit(mushroomTrainPCA, y_train)

```

```

Out[25]: LogisticRegression(solver='liblinear')

```

```

In [26]: mushroomPCATest = pca.transform(x_test_pipelined)

        mushroomPCATest.shape
        predicted = logisticReg.predict(mushroomPCATest)

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

Accuracy:      0.599476

```

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

```

In [ ]: # Models: https://scikit-learn.org/stable/supervised\_learning.html

```

```
In [42]: # KNN
from sklearn.neighbors import KNeighborsClassifier

# USING PCA VALUES
# Note: may be a bit wonky because there are a lot of features, so try with PCA (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: 0.7941586222564893

```
In [39]: from sklearn import metrics
from sklearn.svm import SVC
from sklearn import svm

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

Out[39]: SVC(probability=True)

```
In [22]: # SVM
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 [28]: from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np
```

```
In [43]: clf = RandomForestRegressor(n_estimators=5)
clf = clf.fit(x_train_pipelined, y_train)
y_pred = clf.predict(x_test_pipelined)

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)

print('r2: ', round(r2,4))
print('MSE: ', round(mse,4))
print('MAE: ', round(mae,4))
print('RMSE: ', round(np.sqrt(mse),4))

r2: 0.7376
MSE: 39015.8964
MAE: 8433911103.7891
RMSE: 91836.3278
```

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

```
In [31]: # Cross-Validation: https://scikit-learn.org/stable/modules/cross\_validation.h
        # tml
        # Hyperparameter Tuning: https://scikit-learn.org/stable/modules/grid\_search.h
        # tml

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

```
In [36]: # Logistic regression
scores = cross_val_score(logisticReg1, x_train_pipelined, y_train, cv=kfold)
print("Logistic Regression Accuracy: %.3f" % scores.mean())

Logistic Regression Accuracy: 0.838
```

```
In [45]: # knn
scores = cross_val_score(knnclf, x_train_pipelined, y_train, cv=kfold, n_jobs=-1)
print("Random Forest Accuracy: %.3f" % scores.mean())

Random Forest Accuracy: 0.794
```

This is supposed to say "KNN Accuracy" - I was copy-pasting the print statements and forgot to change the name. The actual code block for Random Forest is on the next page. Sorry!

```
In [47]: # svm
scores = cross_val_score(svm_classifier, x_train_pipeline, y_train, cv=kfold,
n_jobs=-1)
print("SVM Accuracy: %.3f" % scores.mean())
```

SVM Accuracy: 0.806

```
In [48]: # random forest
scores = cross_val_score(clf, x_train_pipeline, y_train, cv=kfold, n_jobs=-1)
print("Random Forest Accuracy: %.3f" % scores.mean())
```

Random Forest Accuracy: 0.743

```
In [56]: # Logistic regression hyperparameter tuning
parameters = {'solver':['newton-cg','liblinear','sag','saga'], 'max_iter':[100, 300, 500, 750]}

search = GridSearchCV(logisticReg1, parameters, scoring='accuracy', cv=kfold,
n_jobs=-1)
result = search.fit(x_train_pipeline, 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 [58]: # KNN Hyperparameter tuning
parameters = {'n_neighbors':[1, 3, 5, 10, 20, 50, 75, 100]}

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

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

Best Score: 0.7936225459812653

Best Hyperparameters: {'n_neighbors': 10}

```
In [54]: # svm Hyperparameter tuning
parameters = {'C': [0.1, 1, 10], 'gamma': [1, 0.1], 'kernel': ['rbf']}

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

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

Best Score: 0.8061255684615632

Best Hyperparameters: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}


```
In [55]: # random forest Hyperparameter tuning
parameters = {'max_features':['auto','sqrt'], 'max_depth':[10,20,30], 'min_samples_split':[2,5,10], 'min_samples_leaf':[1,2,4]}

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

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

Best Score: 0.7429452987445623

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.
```

Best Accuracy - 0.84, Best Precision - 0.87, Best Recall - 0.84, Best F1 Score - 0.85

These results came from the linear regression model. It seems that linear regression can still work well for this dataset even with the high number of features. The model excels in this case because it is classifying between just two labels - poisonous or edible.