DSC550-T301 7.2 Exercise by Stephanie Benavidez

Part 1: PCA and Variance Threshold in a Linear Regression

1. Import the housing data as a data frame and ensure that the data is loaded properly.

In [1]:
```
# import libraries for pandas, numpy, and matplotlib
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

In [2]:
```
# load csv file and look at header info
df_housing = pd.read_csv('train.csv')
df_housing.head()
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

2. Drop the "Id" column and any features that are missing more than 40% of their values.

In [3]:
```
# before droping the ID column double check to make sure that colum is there
df_housing.columns
```

Out[3]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [4]:
```
# drop ID column using .drop function
df_housing.drop(['Id'], axis=1, inplace=True)
# remove any features missing 40% of values with .dropna function
df_housing.dropna(axis=1, thresh=len(df_housing)*.4, inplace=True)
```

3. For numerical columns, fill in any missing data with the median value.

4. For categorical columns, fill in any missing data with the most common value (mode).

In [5]:
```
# create a function fill_missing with for loop
def fill_missing(df):
    for col in df_housing.columns:
        if df_housing[col].dtype == 'object':
            df_housing[col].fillna(df_housing[col].mode()[0], inplace=True)
        else:
            df_housing[col].fillna(df_housing[col].median(), inplace=True)
    return df_housing

df_housing = fill_missing(df_housing)
```

In [6]:
```
# show missing fields
df_housing.isna().sum()
```

Out[6]:
```
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
Street           0
                ..
MoSold           0
YrSold           0
SaleType         0
SaleCondition    0
SalePrice        0
Length: 76, dtype: int64
```

5. Convert the categorical columns to dummy variables.

In [7]:
```
# use get_dummies function to convert categorical columns
df_housing = pd.get_dummies(df_housing, drop_first=True)
df_housing.head()
```

Out[7]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	SaleType_ConLI	SaleType_ConLw	SaleType_New	SaleType_Oth	SaleType_WD	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCondition_Family	SaleCond
0	60	65.0	8450	7	5	2003	2003	196.0	706	0	...	0	0	0	0	1	0	0	0	
1	20	80.0	9600	6	8	1976	1976	0.0	978	0	...	0	0	0	0	1	0	0	0	
2	60	68.0	11250	7	5	2001	2002	162.0	486	0	...	0	0	0	0	1	0	0	0	
3	70	60.0	9550	7	5	1915	1970	0.0	216	0	...	0	0	0	0	1	0	0	0	
4	60	84.0	14260	8	5	2000	2000	350.0	655	0	...	0	0	0	0	1	0	0	0	

5 rows × 237 columns

6. Split the data into a training and test set, where the SalePrice column is the target.

In [8]:
```
# split data by SalePrice to be target
X = df_housing.drop(['SalePrice'], axis=1)
y = df_housing['SalePrice']
# use sklearn to split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

7.Run a linear regression and report the R2-value and RMSE on the test set.

In [9]:
```
# create linear regression from sklearn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [10]:
```
# fit model
model_linear_regression = LinearRegression()
model_linear_regression.fit(X_train, y_train)
y_pred = model_linear_regression.predict(X_test)
```

In [11]:
```
# calculate R2 and RMSE on test
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print(f'RMSE: {rmse}')
print(f'R2: {r2}')
```

```
RMSE: 43905.09876139055
R2: 0.7237558331420327
```

8.Fit and transform the training features with a PCA so that 90% of the variance is retained

(see section 9.1 in the Machine Learning with Python Cookbook).

In [12]:
```
# fit model to show PCA 90%
from sklearn.decomposition import PCA
pca = PCA(.9)
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

9. How many features are in the PCA-transformed matrix?

In [13]:
```
# use print function to show transformed matrix
print(f'Number of features in PCA-transformed matrix: {X_train_pca.shape[1]}')
```

```
Number of features in PCA-transformed matrix: 1
```

10. Transform but DO NOT fit the test features with the same PCA.

11. Repeat step 7 with your PCA transformed data.

In [14]:
```
# create linear regression model with PCA transformed data
model_linear_regression = LinearRegression()
model_linear_regression.fit(X_train_pca, y_train)
y_pred = model_linear_regression.predict(X_test_pca)
```

In [15]:
```
# calculate R2 and RMSE on test
rmse= np.sqrt(mean_squared_error(y_test, y_pred))
r2_pca = r2_score(y_test, y_pred)
print(f'RMSE: {rmse}')
print(f'R2: {r2_pca}')
```

```
RMSE: 80692.38872737018
R2: 0.0668996689358683
```

12. Take your original training features (from step 6) and apply a min-max scaler to them.

13. Find the min-max scaled features in your training set that have a variance above 0.1

(see Section 10.1 in the Machine Learning with Python Cookbook).

In [16]:
```
# import sklearn processing and MinMax Scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

14. Transform but DO NOT fit the test features with the same steps applied in steps 11 and 12.

In [17]:
```
# fit model
model_linear_regression = LinearRegression()
model_linear_regression.fit(X_train_scaled, y_train)
y_pred = model_linear_regression.predict(X_test_scaled)
```

15. Repeat step 7 with the high variance data.

In [18]:
```
# calculate R2 and RMSE on test
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2_scaled = r2_score(y_test, y_pred)
print(f'RMSE: {rmse}')
print(f'R2: {r2_scaled}')
```

```
RMSE: 102854054789167.28
R2: -1.5160238525010898e+18
```

16. Summarize your findings.

Looks like the first set of calculations RMSE: 43905.09876139055; R2: 0.7237558331420327. Then the second set was RMSE: 80692.38872737018; R2: 0.0668996689358683. Then the third set was RMSE: 102854054789167.28; R2: -1.5160238525010898e+18. The results continued to increase for the RMSE with each transformation, and the R2 kept decreasing.

Part 2: Categorial Feature Selection

1. Import the data as a data frame and ensure it is loaded correctly.

In [19]:
```
# import libraries for pandas, numpy, and matplotlib
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

In [20]:
```
# load csv file and look at header info
df_mushrooms= pd.read_csv('mushrooms.csv')
df_mushrooms.head()
```

Out[20]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g

5 rows × 23 columns

2. Convert the categorical features (all of them) to dummy variables.

In [21]:
```
# use get_dummies function to convert categorical columns
df_mushrooms = pd.get_dummies(df_mushrooms, drop_first=True)
df_mushrooms.head()
```

Out[21]:

	class_p	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_g	cap-surface_s	cap-surface_y	cap-color_c	...	population_n	population_s	population_v	population_y	habitat_g	habitat_l	habitat_m	habitat_p	habitat_u	habitat_w
0	1	0	0	0	0	1	0	1	0	0	...	0	1	0	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0	1	0	0	...	1	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	...	1	0	0	0	0	0	1	0	0	0
3	1	0	0	0	0	1	0	0	1	0	...	0	1	0	0	0	0	0	0	1	0
4	0	0	0	0	0	1	0	1	0	0	...	0	0	0	0	1	0	0	0	0	0

5 rows × 96 columns

3. Split the data into a training and test set.

In [22]:
```
# use sklearn to split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

4. Fit a decision tree classifier on the training set.

In [23]:
```
# import Decision Tree Classifier from sklearn
from sklearn.tree import DecisionTreeClassifier
# fit Decision Tree Classifier to training set
clf = DecisionTreeClassifier(max_depth =3, random_state = 42)
clf.fit(X_train, y_train)
```

Out[23]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

5. Report the accuracy and create a confusion matrix for the model prediction on the test set.

In [24]:
```
# report accuracy of training set and print results
y_pred = clf.predict(X_test)
from sklearn.metrics import accuracy_score
print("Model accuracy score for Decision Tree Classifier: {0:0.4f}".format(accuracy_score(y_test,y_pred)))
```

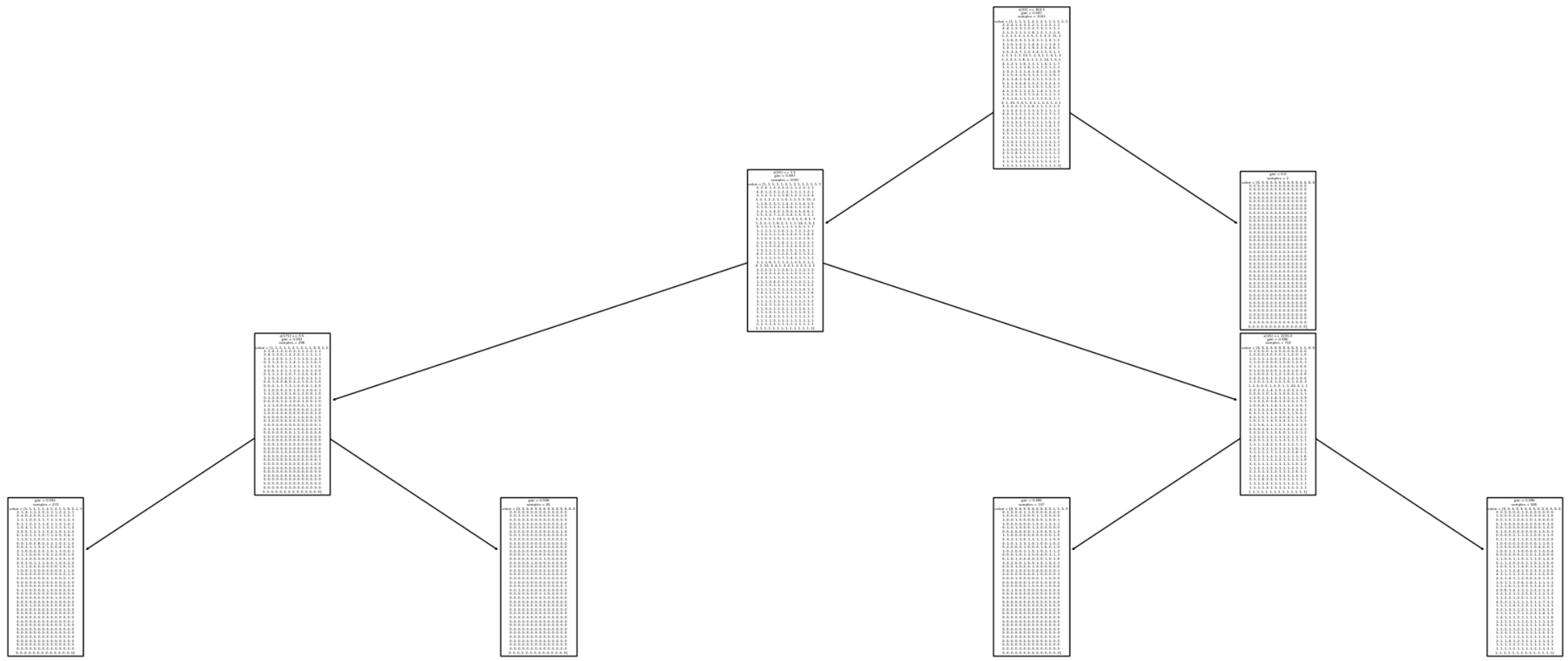
```
Model accuracy score for Decision Tree Classifier: 0.0160
```

In [25]:
```
# create matrix for model prediction
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,y_pred))
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

6. Create a visualization of the decision tree.

In [26]:
```
# import tree visualization
from sklearn import tree
import matplotlib.pyplot as plt
plt.figure(figsize=(30,10))
tree.plot_tree(clf.fit(X_train, y_train))
plt.show()
```



7. Use a χ2-statistic selector to pick the five best features for this data

(see section 10.4 of the Machine Learning with Python Cookbook).

In [27]:
```
# Load libraries from sklearn
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [33]:
```
# k = 5 tells top five features to be selected
# Score function Chi2 tells the feature to be selected using Chi Square
KBest = SelectKBest(chi2, k=5)
fit = KBest.fit_transform(X, y)
fit.shape
```

Out[33]:
```
(1460, 5)
```

8. Which five features were selected in step 7? Hint: Use the get_support function.

In [34]:
```
# five features using get_support
results = KBest.get_support(5)
print(results)
```

```
[ 2  8 13 32 33]
```

9. Repeat steps 4 and 5 with the five best features selected in step 7.

10. Summarize your findings.