**Feature Engineering – Principal Component Analysis**

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

from sklearn.decomposition import PCA

from sklearn.feature\_selection import mutual\_info\_regression

from sklearn.model\_selection import cross\_val\_score

from xgboost import XGBRegressor

*# Set Matplotlib defaults*

plt.style.use("seaborn-whitegrid")

plt.rc("figure", autolayout=True)

plt.rc(

"axes",

labelweight="bold",

labelsize="large",

titleweight="bold",

titlesize=14,

titlepad=10,

)

def apply\_pca(X, standardize=True):

*# Standardize*

if standardize:

X = (X - X.mean(axis=0)) / X.std(axis=0)

*# Create principal components*

pca = PCA()

X\_pca = pca.fit\_transform(X)

*# Convert to dataframe*

component\_names = [f"PC**{**i+1**}**" for i **in** range(X\_pca.shape[1])]

X\_pca = pd.DataFrame(X\_pca, columns=component\_names)

*# Create loadings*

loadings = pd.DataFrame(

pca.components\_.T, *# transpose the matrix of loadings*

columns=component\_names, *# so the columns are the principal components*

index=X.columns, *# and the rows are the original features*

)

return pca, X\_pca, loadings

def plot\_variance(pca, width=8, dpi=100):

*# Create figure*

fig, axs = plt.subplots(1, 2)

n = pca.n\_components\_

grid = np.arange(1, n + 1)

*# Explained variance*

evr = pca.explained\_variance\_ratio\_

axs[0].bar(grid, evr)

axs[0].set(

xlabel="Component", title="**% E**xplained Variance", ylim=(0.0, 1.0)

)

*# Cumulative Variance*

cv = np.cumsum(evr)

axs[1].plot(np.r\_[0, grid], np.r\_[0, cv], "o-")

axs[1].set(

xlabel="Component", title="% Cumulative Variance", ylim=(0.0, 1.0)

)

*# Set up figure*

fig.set(figwidth=8, dpi=100)

return axs

def make\_mi\_scores(X, y):

X = X.copy()

for colname **in** X.select\_dtypes(["object", "category"]):

X[colname], \_ = X[colname].factorize()

*# All discrete features should now have integer dtypes*

discrete\_features = [pd.api.types.is\_integer\_dtype(t) for t **in** X.dtypes]

mi\_scores = mutual\_info\_regression(X, y, discrete\_features=discrete\_features, random\_state=0)

mi\_scores = pd.Series(mi\_scores, name="MI Scores", index=X.columns)

mi\_scores = mi\_scores.sort\_values(ascending=False)

return mi\_scores

def score\_dataset(X, y, model=XGBRegressor()):

*# Label encoding for categoricals*

for colname **in** X.select\_dtypes(["category", "object"]):

X[colname], \_ = X[colname].factorize()

*# Metric for Housing competition is RMSLE (Root Mean Squared Log Error)*

score = cross\_val\_score(

model, X, y, cv=5, scoring="neg\_mean\_squared\_log\_error",

)

score = -1 \* score.mean()

score = np.sqrt(score)

return score

df = pd.read\_csv("../input/fe-course-data/ames.csv")

Let's choose a few features that are highly correlated with our target, SalePrice.

features = [

"GarageArea",

"YearRemodAdd",

"TotalBsmtSF",

"GrLivArea",

]

print("Correlation with SalePrice:**\n**")

print(df[features].corrwith(df.SalePrice))

Correlation with SalePrice:

GarageArea 0.640138

YearRemodAdd 0.532974

TotalBsmtSF 0.632529

GrLivArea 0.706780

dtype: float64

We'll rely on PCA to untangle the correlational structure of these features and suggest relationships that might be usefully modeled with new features.

Run this cell to apply PCA and extract the loadings.

X = df.copy()

y = X.pop("SalePrice")

X = X.loc[:, features]

*# `apply\_pca`, defined above, reproduces the code from the tutorial*

pca, X\_pca, loadings = apply\_pca(X)

print(loadings)

PC1 PC2 PC3 PC4

GarageArea 0.541229 0.102375 -0.038470 0.833733

YearRemodAdd 0.427077 -0.886612 -0.049062 -0.170639

TotalBsmtSF 0.510076 0.360778 -0.666836 -0.406192

GrLivArea 0.514294 0.270700 0.742592 -0.332837

1) Interpret Component Loadings

Look at the loadings for components PC1 and PC3. Can you think of a description of what kind of contrast each component has captured? After you've thought about it, run the next cell for a solution.

The first component, PC1, seems to be a kind of "size" component, similar to what we saw in the tutorial: all of the features have the same sign (positive), indicating that this component is describing a contrast between houses having large values and houses having small values for these features.

The interpretation of the third component PC3 is a little trickier. The features GarageArea and YearRemodAdd both have near-zero loadings, so let's ignore those. This component is mostly about TotalBsmtSF and GrLivArea. It describes a contrast between houses with a lot of living area but small (or non-existant) basements, and the opposite: small houses with large basements.

Your goal in this question is to use the results of PCA to discover one or more new features that improve the performance of your model. One option is to create features inspired by the loadings, like we did in the tutorial. Another option is to use the components themselves as features (that is, add one or more columns of X\_pca to X).

# 2) Create New Features

Add one or more new features to the dataset X. For a correct solution, get a validation score below 0.140 RMSLE

X = df.copy()

y = X.pop("SalePrice")

*# YOUR CODE HERE: Add new features to X.*

X["Feature1"] = X.GrLivArea + X.TotalBsmtSF

X["Feature2"] = X.YearRemodAdd \* X.TotalBsmtSF

score = score\_dataset(X, y)

print(f"Your score: **{**score**:**.5f**}** RMSLE")

Your score: 0.13361 RMSLE

The next question explores a way you can use PCA to detect outliers in the dataset (meaning, data points that are unusually extreme in some way). Outliers can have a detrimental effect on model performance, so it's good to be aware of them in case you need to take corrective action. PCA in particular can show you anomalous variation which might not be apparent from the original features: neither small houses nor houses with large basements are unusual, but it is unusual for small houses to have large basements. That's the kind of thing a principal component can show you.

Run the next cell to show distribution plots for each of the principal components you created above.

sns.catplot(

y="value",

col="variable",

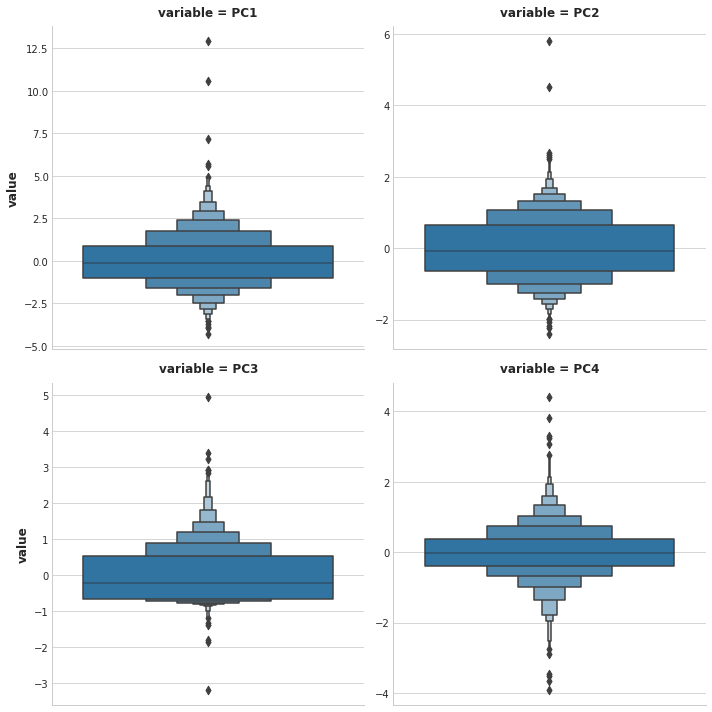
data=X\_pca.melt(),

kind='boxen',

sharey=False,

col\_wrap=2,

);

****

As you can see, in each of the components there are several points lying at the extreme ends of the distributions -- outliers, that is.

Now run the next cell to see those houses that sit at the extremes of a component:

*# You can change PC1 to PC2, PC3, or PC4*

component = "PC1"

idx = X\_pca[component].sort\_values(ascending=False).index

df.loc[idx, ["SalePrice", "Neighborhood", "SaleCondition"] + features]

| SalePrice | Neighborhood | SaleCondition | GarageArea | YearRemodAdd | TotalBsmtSF | GrLivArea |
| --- | --- | --- | --- | --- | --- | --- |
| 1498 | 160000 | Edwards | Partial | 1418.0 | 2008 | 6110.0 | 5642.0 |
| 2180 | 183850 | Edwards | Partial | 1154.0 | 2009 | 5095.0 | 5095.0 |
| 2181 | 184750 | Edwards | Partial | 884.0 | 2008 | 3138.0 | 4676.0 |
| 1760 | 745000 | Northridge | Abnorml | 813.0 | 1996 | 2396.0 | 4476.0 |
| 1767 | 755000 | Northridge | Normal | 832.0 | 1995 | 2444.0 | 4316.0 |

# 3) Outlier Detection

Do you notice any patterns in the extreme values? Does it seem like the outliers are coming from some special subset of the data?

After you've thought about your answer, run the next cell for the solution and some discussion.

Notice that there are several dwellings listed as Partial sales in the Edwards neighborhood that stand out. A partial sale is what occurs when there are multiple owners of a property and one or more of them sell their "partial" ownership of the property.

These kinds of sales are often happen during the settlement of a family estate or the dissolution of a business and aren't advertised publicly. If you were trying to predict the value of a house on the open market, you would probably be justified in removing sales like these from your dataset -- they are truly outliers.