1. **INTERPRETING ONE-HOT ENCODED COEFFICIENTS**

That is a much more manageable number of coefficients. Let's go through and interpret these:

\* The **\*\*reference category\*\*** for `origin` is `1` (US) and for `make` is `amc` (American Motor Company)

\* `const`, `weight`, and `model year` are all still statistically significant

  \* When all other predictors are 0, the MPG would be about -18.3

  \* For each increase of 1 lb in weight, we see an associated decrease of about 0.006 in MPG

  \* For each year newer the vehicle is, we see an associated increase of about 0.75 in MPG

\* `origin\_2` and `origin\_3` are not statistically significant any more

  \* While this might seem surprising, our data understanding can explain it. The `origin` feature and the `make` feature are really providing the same information, except that `make` is more granular. Every `make` category (except for `other`) corresponds to exactly one `origin` category. Therefore it probably does not make sense to include both `origin` and `make` in the same model

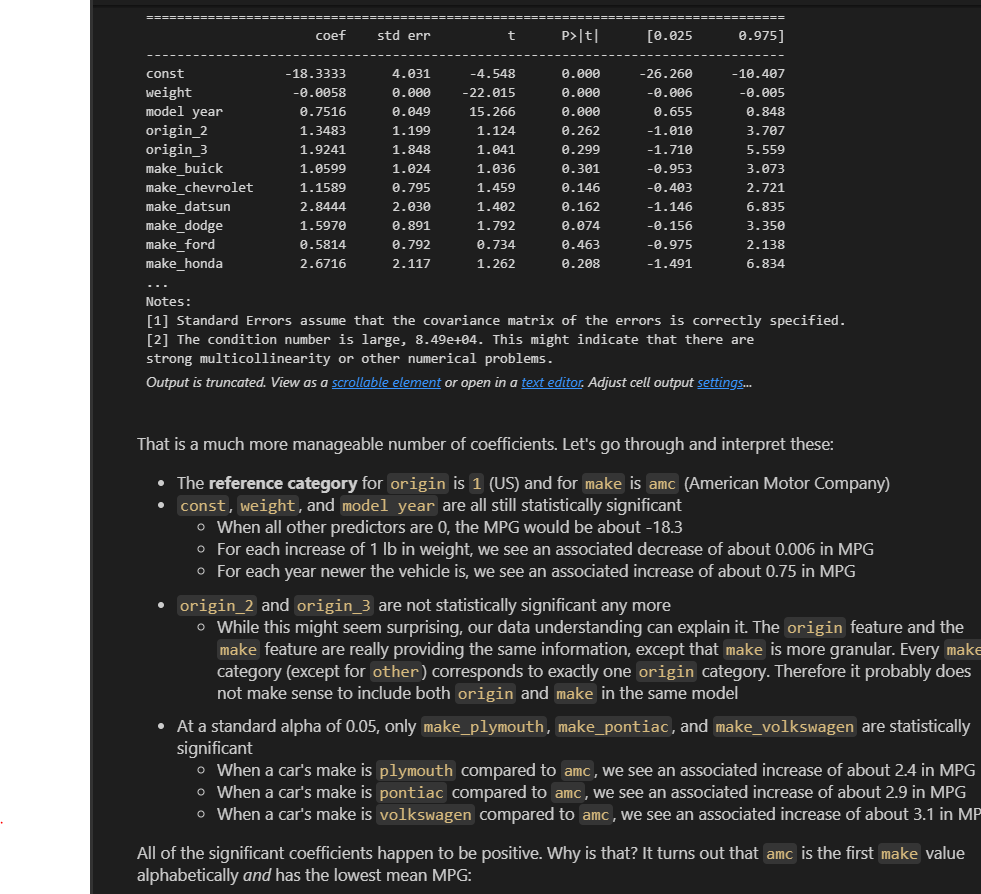
\* At a standard alpha of 0.05, only `make\_plymouth`, `make\_pontiac`, and `make\_volkswagen` are statistically significant

  \* When a car's make is `plymouth` compared to `amc`, we see an associated increase of about 2.4 in MPG

  \* When a car's make is `pontiac` compared to `amc`, we see an associated increase of about 2.9 in MPG

  \* When a car's make is `volkswagen` compared to `amc`, we see an associated increase of about 3.1 in MPG

All of the significant coefficients happen to be positive. Why is that? It turns out that `amc` is the first `make` value alphabetically *\_and\_* has the lowest mean MPG:



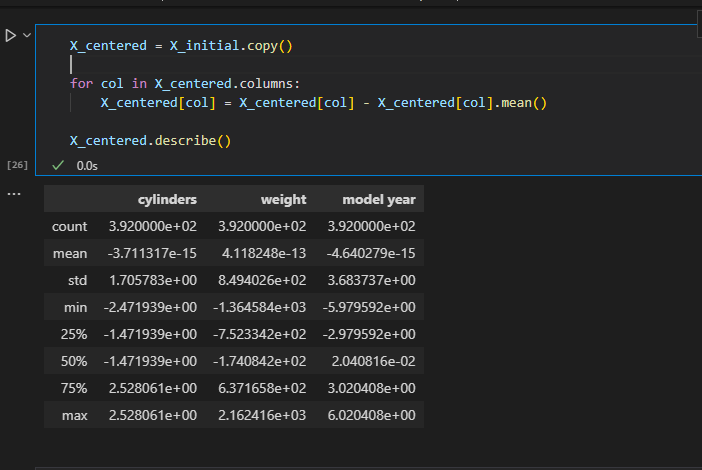
1. **CENTERING**

X\_centered = X\_initial.copy()

for col in X\_centered.columns:

    X\_centered[col] = X\_centered[col] - X\_centered[col].mean()

X\_centered.describe()



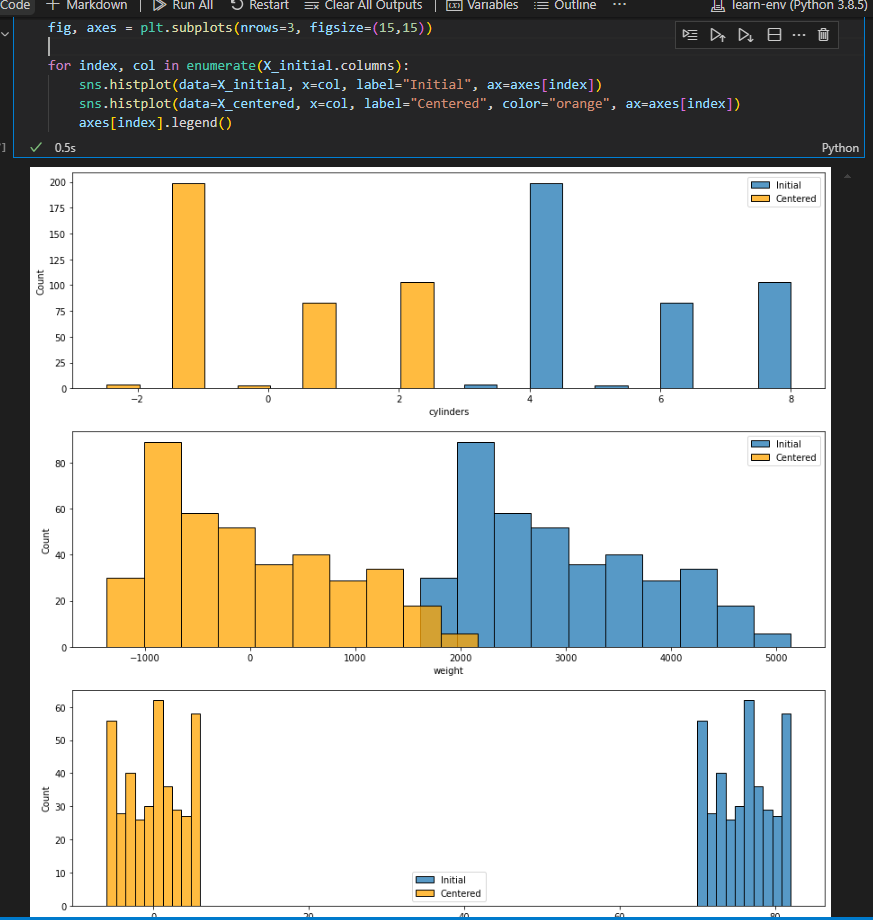
fig, axes = plt.subplots(nrows=3, figsize=(15,15))

for index, col in enumerate(X\_initial.columns):

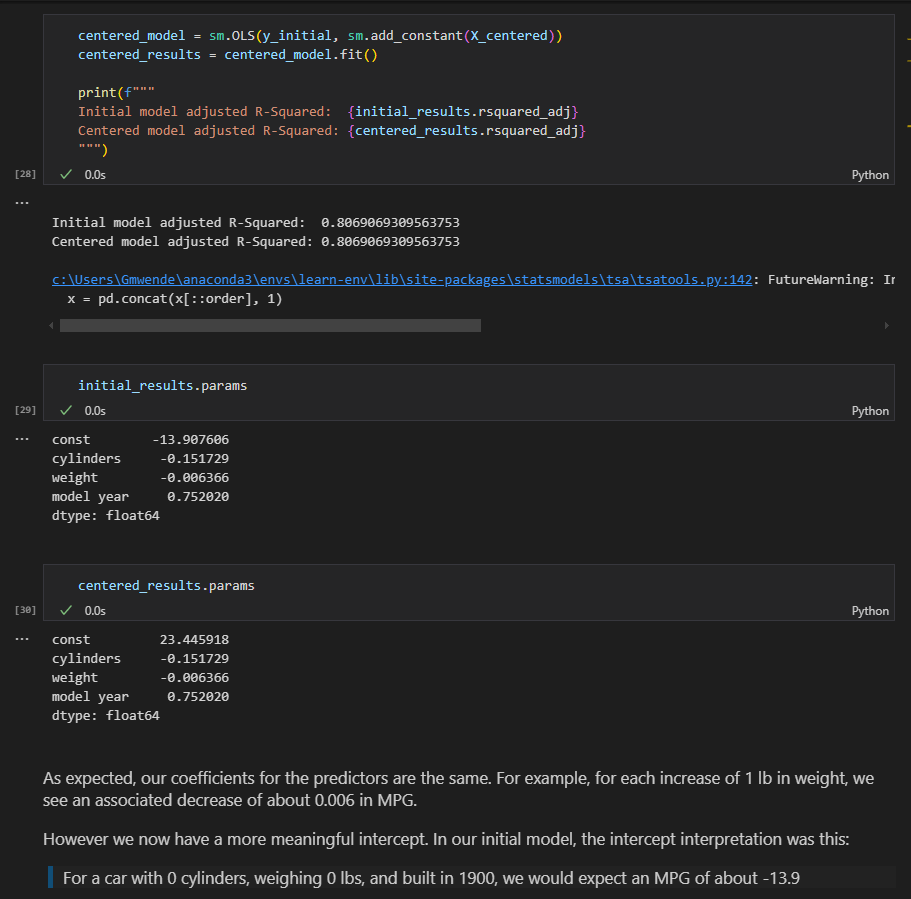
    sns.histplot(data=X\_initial, x=col, label="Initial", ax=axes[index])

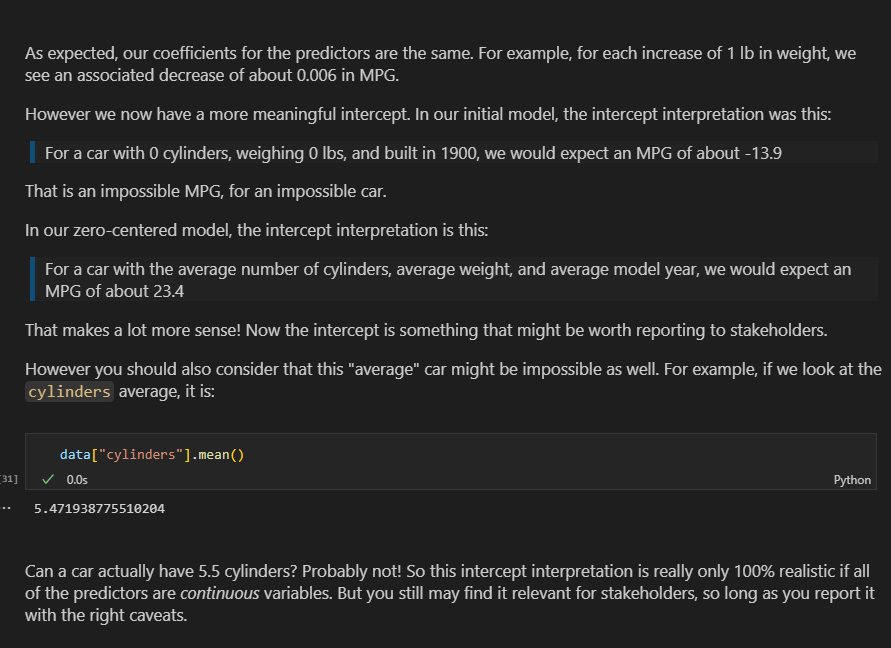
    sns.histplot(data=X\_centered, x=col, label="Centered", color="orange", ax=axes[index])

    axes[index].legend()

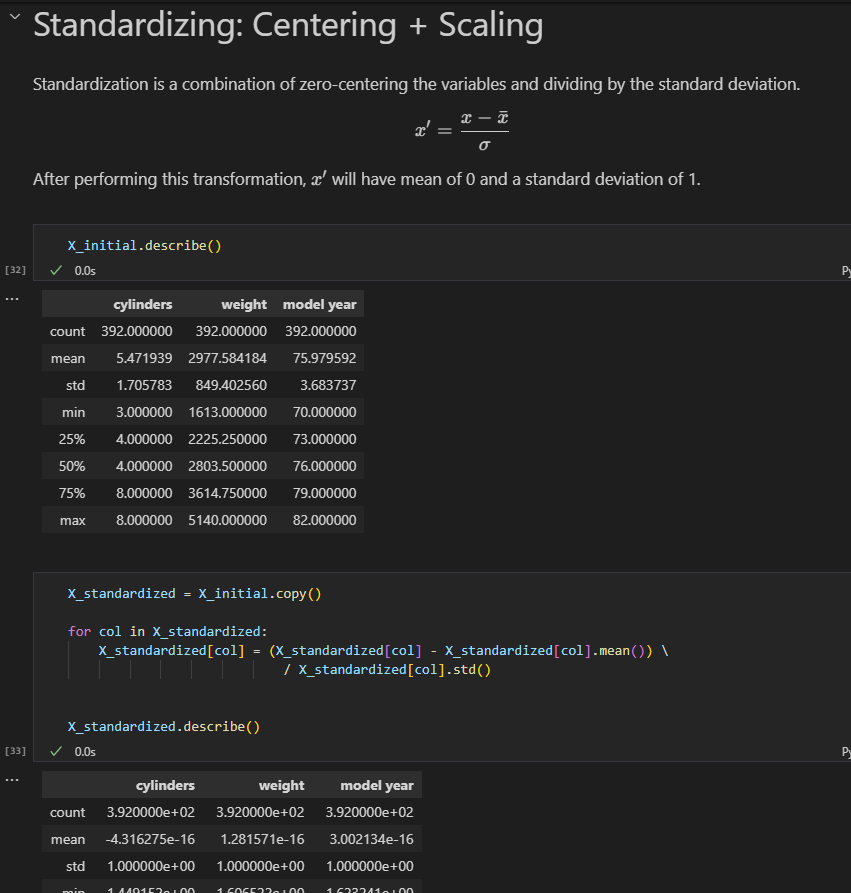


On modelling we can now our coefficients are interpretable

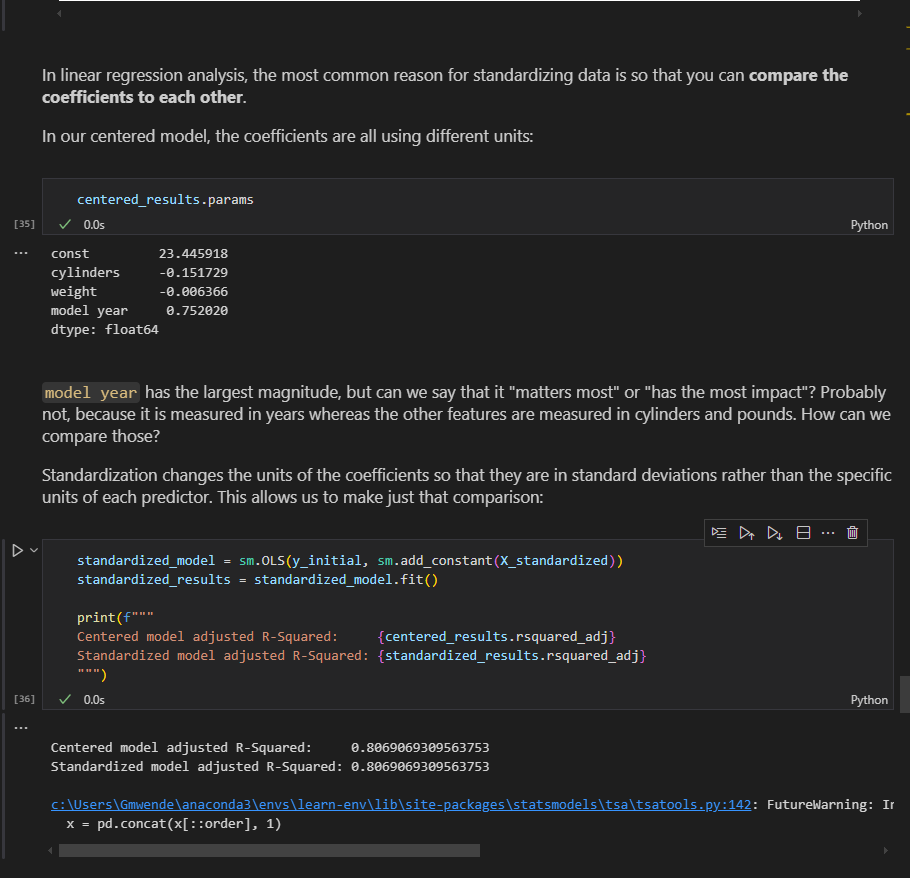


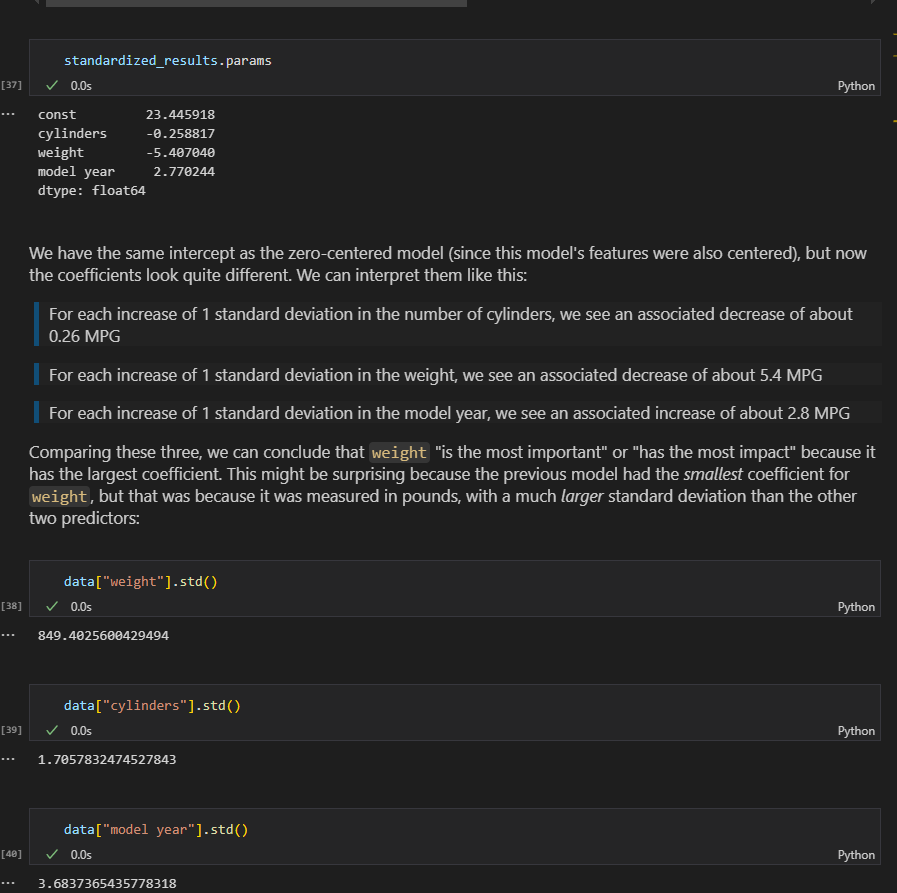


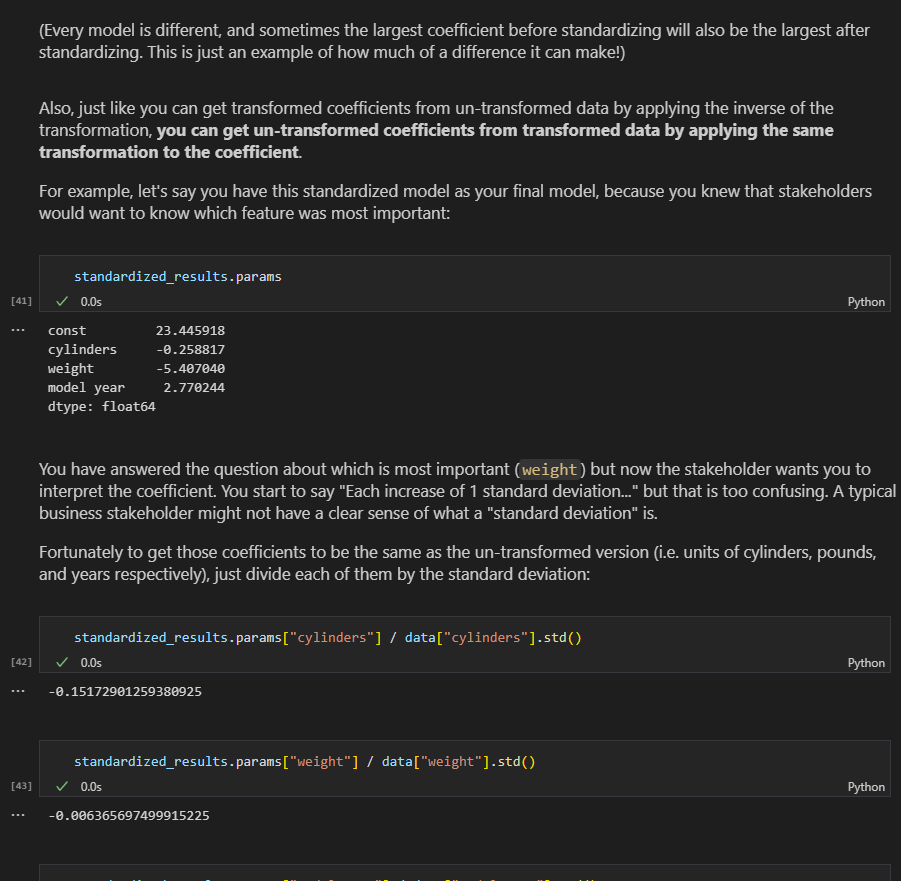
1. Standardization

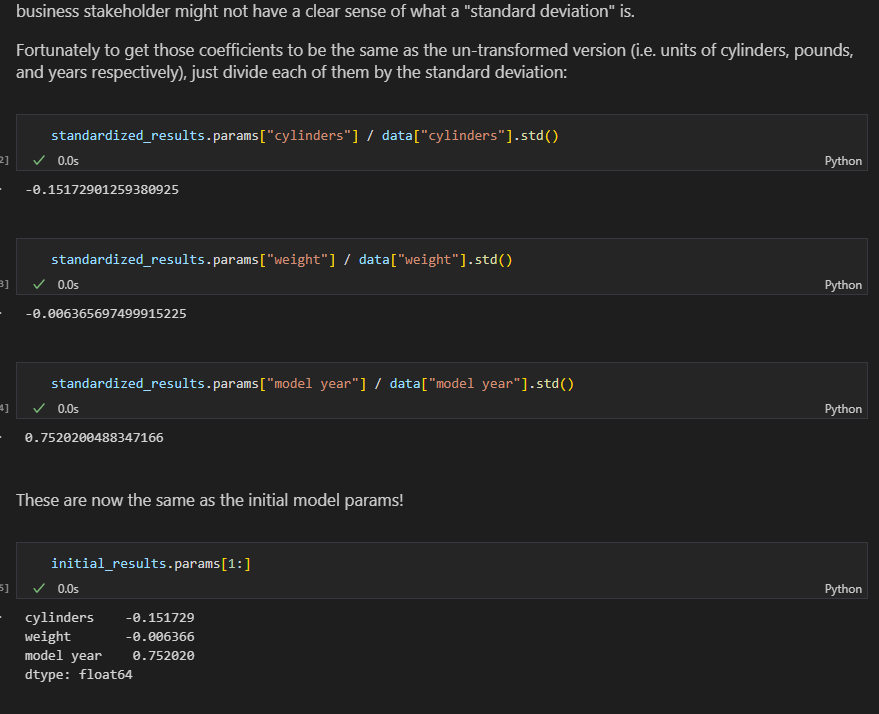












Standardization using sklearn

from sklearn.preprocessing import StandardScaler

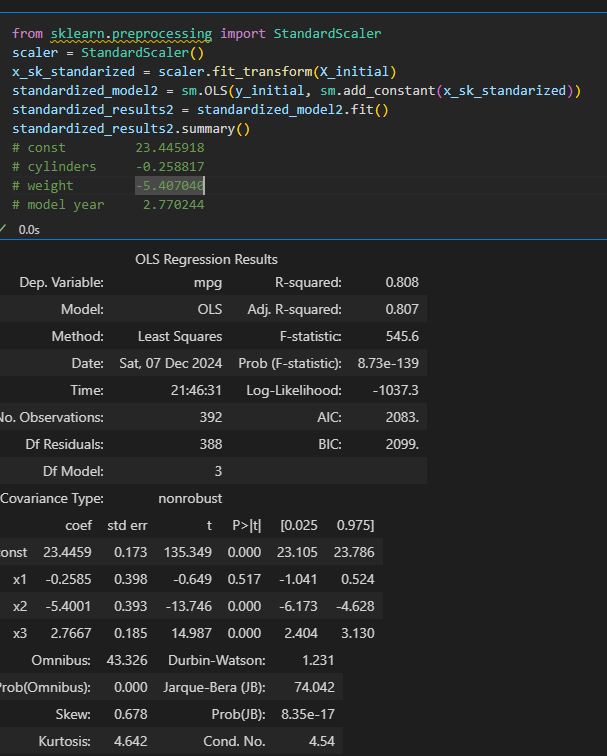
scaler = StandardScaler()

x\_sk\_standarized = scaler.fit\_transform(X\_initial)

standardized\_model2 = sm.OLS(y\_initial, sm.add\_constant(x\_sk\_standarized))

standardized\_results2 = standardized\_model2.fit()

standardized\_results2.summary()



**Draw Graph for multiple columns eg check good candidate for log transformation**

# Run this cell without changes

import matplotlib.pyplot as plt

import numpy as np

y = ames["SalePrice"]

X = ames.drop("SalePrice", axis=1)

fig, axes = plt.subplots(nrows=6, ncols=6, figsize=(15,15), sharey=True)

for i, column in enumerate(X.columns):

    # Locate applicable axes

    row = i // 6

    col = i % 6

    ax = axes[row][col]

    # Plot feature vs. y and label axes

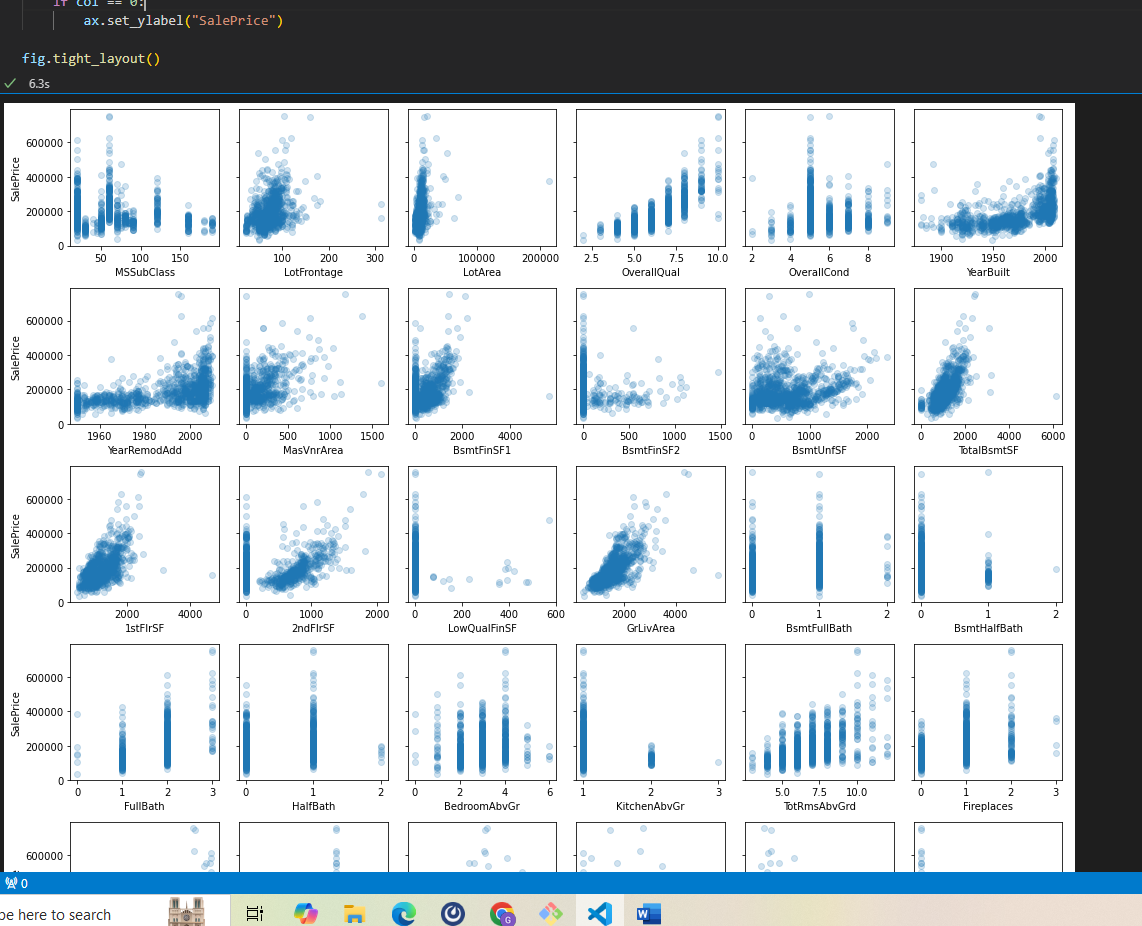
    ax.scatter(X[column], y, alpha=0.2)

    ax.set\_xlabel(column)

    if col == 0:

        ax.set\_ylabel("SalePrice")

fig.tight\_layout()



1. **One Hot Encoding using sklearn**

from sklearn.preprocessing import OneHotEncoder

#encode test data

test\_encoded = ohe.transform(X\_test[columns\_to\_encode])

#Turn into a dataframe

new\_test\_df = pd.DataFrame(

              test\_encoded.todense(),

              columns= ohe.get\_feature\_names\_out(),

              index=X\_test.index

)

new\_test\_df.head()

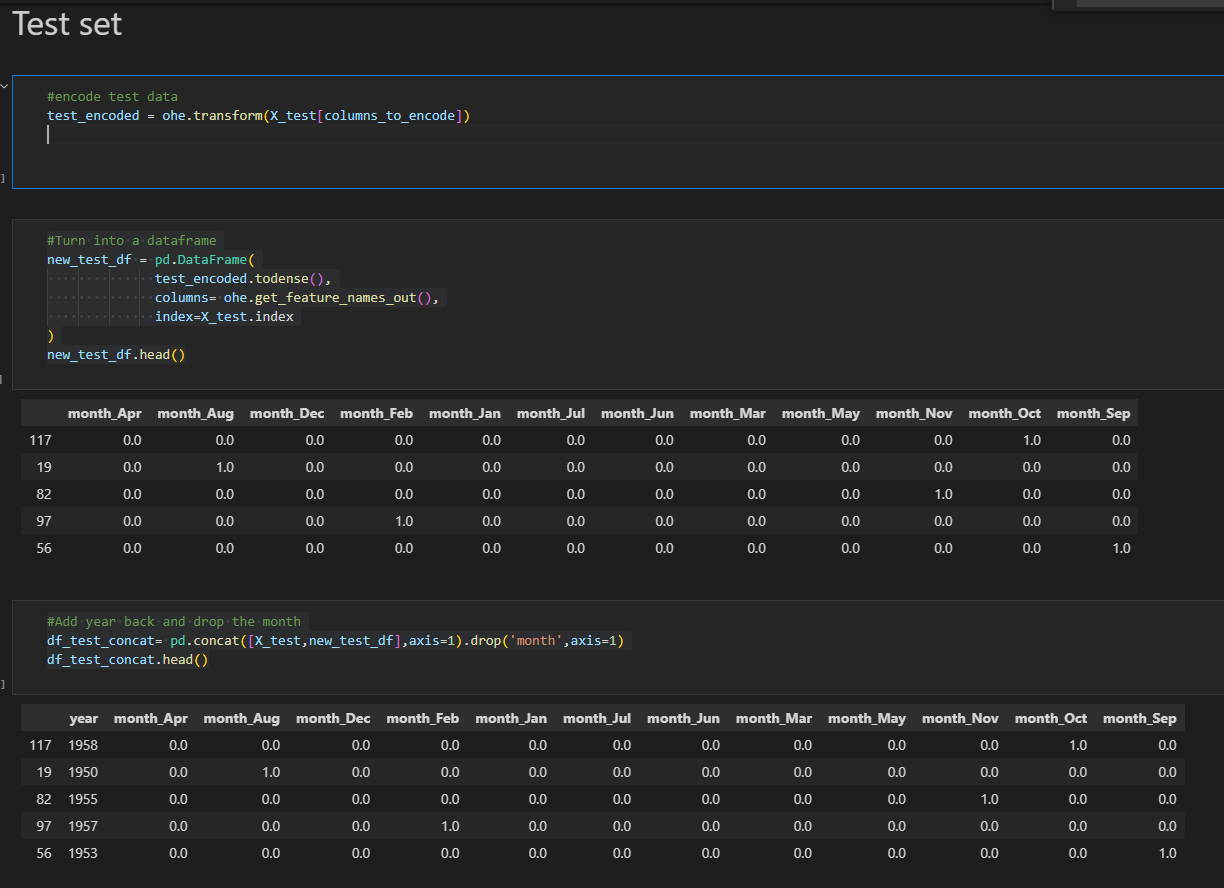
#Add year back and drop the month

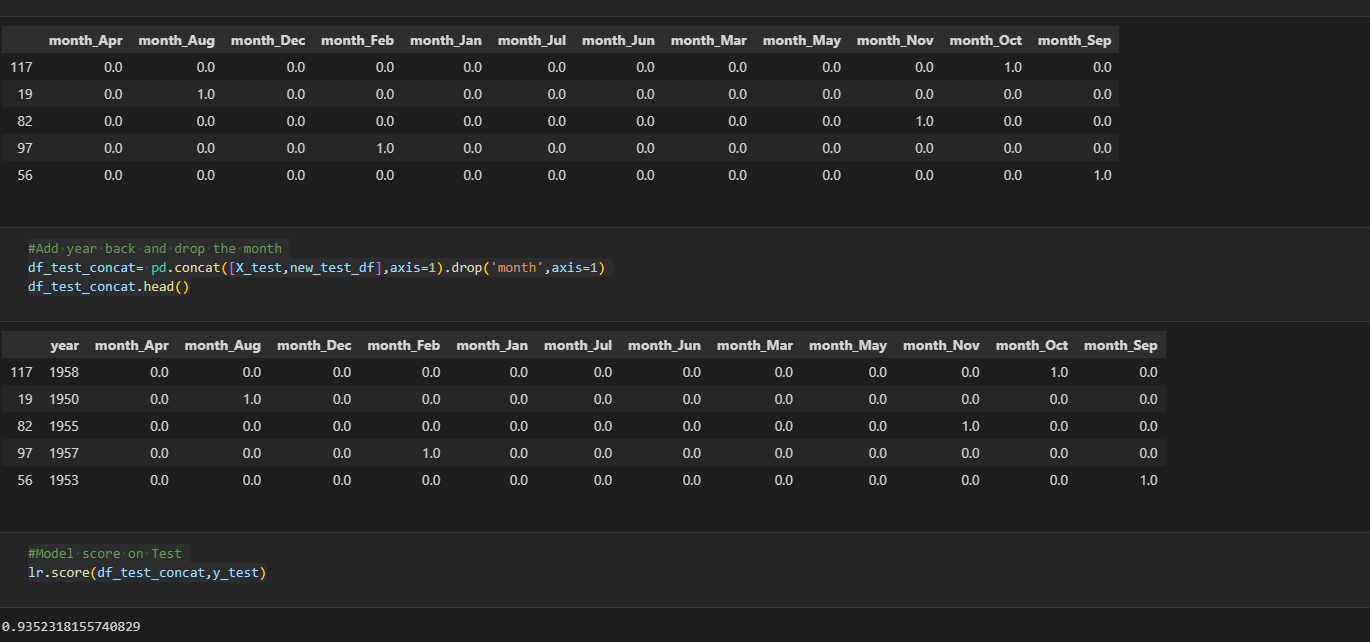
df\_test\_concat= pd.concat([X\_test,new\_test\_df],axis=1).drop('month',axis=1)

df\_test\_concat.head()

#Model score on Test

lr.score(df\_test\_concat,y\_test)





1. **POLYNOMIALS**

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(8)

X\_poly\_high = poly.fit\_transform(x)

X\_poly\_high

x\_poly\_high\_df = pd.DataFrame(X\_poly\_high,columns=poly.get\_feature\_names\_out(x.columns),index=x.index)

x\_poly\_high\_df

x\_poly\_high\_df.drop("1",axis=1,inplace=True)

poly\_results = sm.OLS(y, x\_poly\_high\_df).fit()

poly\_results.summary()

predeictions =  poly\_results.predict(x\_poly\_high\_df)

sns.scatterplot(x=df["Temp"],y=df["Yield"])

sns.lineplot(x=df["Temp"],y=predeictions)

plt.show()

1. 