2)
```
import pandas as pd

# Load CSV file into a DataFrame (this is a comment)
df = pd.read_csv('path/to/file.csv')

# Print the first few rows of the DataFrame (this is a comment)
print(df.head())
```

In this script, we first import the pandas library with the **import pandas as pd** statement. Then we use the **pd.read_csv()** function to load the CSV file located at 'path/to/file.csv' and read it into a DataFrame. Finally, we use the **df.head()** function to print the first few rows of the DataFrame.
Note: Make sure to replace **'path/to/file.csv'** with the actual file path to your CSV file.

(BASCALLY IN THE ABOVE CODE WHERE IT IS MENTIONED AS PATH TO CSV FILE, DOWNLOAD THE DATASET INTO YOUR LOCAL FILE AND MENTION THE DATA SET FILE PATH IN THE CODE)

3)
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy
import stats from sklearn
import model_selection, feature_selection, linear_model from sklearn.linear_model import
LinearRegression from sklearn.preprocessing
import StandardScaler,MinMaxScaler,OneHotEncoder,OrdinalEncoder from matplotlib
import pyplot from sklearn.linear_model
import Ridge , Lasso,RidgeCV from sklearn.metrics
import mean_squared_error, r2_score from scipy.stats
import shapirofrom statsmodels.stats.diagnostic
import normal_ad from statsmodels.stats.outliers_influence
import variance_inflation_factor
import statsmodels.api as sm from statsmodels.stats.diagnostic
import het_goldfeldquandt
import warningswarnings.filterwarnings('ignore') from sklearn.compose
import ColumnTransformer , make_column_transformer,make_column_selector from
sklearn.pipeline
import Pipeline from sklearn.feature_selection
import mutual_info_regression, SelectKBest,f_regression,RFE
df["normalized-losses"].fillna(df["normalized-losses"].astype("float").mean(), inplace=True)
df["bore"].fillna(df["bore"].astype("float").mean(), inplace=True)
df["stroke"].fillna(df["stroke"].astype("float").mean(), inplace = True)
df["peak-rpm"].fillna(df["peak-rpm"].astype("float").mean(), inplace = True)
df['horsepower'].fillna(df['horsepower'].astype("float").mean(), inplace=True)
```

```
category = ["make","fuel-type","aspiration","num-of-doors","body-style",
        "drive-wheels","engine-location","engine-type","num-of-cylinders","fuel-system"]fig,
axes = plt.subplots(nrows=5, ncols=2, figsize=(14, 18))axe = axes.ravel()for i, category in
enumerate(df_obj[category]):
    df_obj[category].value_counts().plot(kind="bar",
ax=axe[i]).set_title(category)fig.show()fig.tight_layout()

sns.regplot(x = 'engine-size', y = 'price', data = df)plt.ylim(0,) # y axis starts from
zeroprint("correlation between engine-size and price:")df[["engine-size", "price"]].corr()
sns.regplot(x = 'highway-mpg', y = 'price',data = df)print('correlation between highway-mpg
and price:')df[['highway-mpg', 'price']].corr()
sns.regplot(x = 'peak-rpm', y = 'price', data = df)print('correlation between peak-rpm and
price:')df[['peak-rpm','price']].corr()
print('correlation between stroke and prce :')
sns.regplot(x = 'stroke', y = 'price', data = df)
.df[['stroke','price']].corr()
stdscaler=StandardScaler()numeric_features = ["symboling","normalized-losses","wheel-
base", "length","width",
          "height","curb-weight", "engine-size","bore","stroke","compression-
ratio","horsepower",
         "peak-rpm","city-mpg","highway-
mpg"]num_transformed=stdscaler.fit_transform(X_numeric)num_transformed=pd.DataFram
e(num_transformed, columns=numeric_features)num_transformed.head()
def residual(model,feature,label):
    pred=model.predict(feature)
    df2=pd.DataFrame({"Actual":label,"Predicted":pred})
    df2["Residuals"]=df2["Actual"]-df2["Predicted"]
    return df2
def linear_assumption(model, features, label):
    print('Linearity Check:', '\n')

    print('Checking with a scatter plot of actual vs. predicted.',
          'Predictions should follow the diagonal line.')
     df_results = residual(model, features, label)



    sns.lmplot(x='Actual', y='Predicted', data=df_results, fit_reg=False, height=7)
    # sns.residplot(x="Actual", y="Predicted", data=df_results)
    # line_coords = np.arange(df_results.min().min(), df_results.max().max())
    line_coords = np.arange(df_results.iloc[:,0:2].min().min(),
df_results.iloc[:,0:2].max().max())
    plt.plot(line_coords, line_coords,  # X and y points
             color='darkorange', linestyle='--')
    plt.title('Actual vs. Predicted')
    plt.show()
```