```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
import numpy as np
from prettytable import PrettyTable
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error


url = 'https://raw.githubusercontent.com/rjafari979/Information-Visualization-Data-
Analytics-Dataset-/refs/heads/main/Carseats.csv'
pd.set_option('display.max_columns', None)
df = pd.read_csv(url)
print(df.head())


#%%------------Q1.a-------------------
agg_data = df.groupby(['ShelveLoc','US'])['Sales'].sum().unstack()
max_sales_location = agg_data.sum(axis=1).idxmax()
max_sales_value = agg_data.loc[max_sales_location]
inside_or_outside = 'Yes' if max_sales_value['Yes'] > max_sales_value['No'] else
'No'
print(f"Shelve location with the highest total sales: '{max_sales_location}'")
print(f"Highest sales occurred inside the US: {inside_or_outside}")
if inside_or_outside == 'Yes':
    print(f"Yes    {max_sales_value['Yes']}")
else:
    print(f"No     {max_sales_value['No']}")
agg_data.plot(kind='barh',stacked = False, figsize=(10,6))
plt.xlabel('Sales')
plt.ylabel('Shelve Location')
plt.title('Shelve Location vs Sales')
plt.legend(title='US')
plt.show()

#%%------------Q1.b-------------------
df_one_hot_encode=pd.get_dummies(df,columns=['ShelveLoc','Urban','US'],drop_first=T
rue)
df_one_hot_encode[['ShelveLoc_Medium', 'ShelveLoc_Good', 'Urban_Yes', 'US_Yes']] =
(df_one_hot_encode[['ShelveLoc_Medium', 'ShelveLoc_Good', 'Urban_Yes',
'US_Yes']].astype(int))
converted_features
=df_one_hot_encode[df_one_hot_encode.columns.difference(df.columns)]
print("Printing converted features...\n",converted_features.head())

#%%------------Q1.c-------------------
encoded_features = df_one_hot_encode[['ShelveLoc_Medium', 'ShelveLoc_Good',
'Urban_Yes', 'US_Yes']]
numerical_features = df_one_hot_encode.drop(['ShelveLoc_Medium', 'ShelveLoc_Good',
'Urban_Yes', 'US_Yes'], axis=1)
df_train,df_test =
```

```
train_test_split(df_one_hot_encode,train_size=0.80,test_size=0.20,shuffle=True,rand
om_state=5805)
scaler = StandardScaler()
numerical_train = df_train[numerical_features.columns]
numerical_test = df_test[numerical_features.columns]
df_train[numerical_features.columns] = scaler.fit_transform(numerical_train)
df_test[numerical_features.columns] = scaler.transform(numerical_test)
print("First 5 rows of train set:\n",df_train.head())
print("\nFirst 5 rows of test set:\n",df_test.head())

#%%------------Q2.a,b-------------------
dependent_variable='Sales'
independent_variables= list(filter(lambda col: col != dependent_variable,
df_train.columns))
X_train = df_train[independent_variables]
X_test = df_test[independent_variables]
y_train = df_train[dependent_variable]
y_test=df_test[dependent_variable]
dropped_features=[]
X_model_train=sm.add_constant(X_train)
X_test=sm.add_constant(X_test)
table = PrettyTable()
table.field_names = ["Process",  "AIC", "BIC","Adjusted R-squared", "P-value"]
step = 0
dropped_features = []
while True:
    model = sm.OLS(y_train, X_model_train).fit()
    print(model.summary())
    p_values = model.pvalues
    p_values = p_values.drop('const')
    max_p_value = p_values.max()
    if max_p_value < 0.01:
        break
    feature_to_remove = p_values.idxmax()
    table.add_row([feature_to_remove, round(model.aic, 3), round(model.bic,
3),round(model.rsquared_adj, 3),round(max_p_value, 3)])
    dropped_features.append(feature_to_remove)
    X_model_train = X_model_train.drop(columns=feature_to_remove)
    step += 1
final_model = sm.OLS(y_train, X_model_train).fit()
print(table)
print("\nFinal selected features:", X_model_train.columns.tolist())
print("\nDropped features:", dropped_features)
print("\nFinal Model Summary:")
print(final_model.summary())

#%%------------Q2.c-------------------
intercept = final_model.params['const']
coefficients = final_model.params.drop('const')
equation = f"Sales = {intercept:.3f} "
for feature, coef in coefficients.items():
    equation += f"+ ({coef:.3f} * {feature}) "
print("\nFinal Regression Equation:")
print(equation)
X_stepwise_reg_test = sm.add_constant(X_test[X_model_train.columns.drop('const')],
has_constant='add')
y_stepwise_reg_pred = final_model.predict(X_stepwise_reg_test)
scaler = StandardScaler()
scaler.fit(df_one_hot_encode[['Sales']])
```

```
y_test_original_scale = scaler.inverse_transform(y_test.values.reshape(-1, 1))
y_pred_original_scale =
scaler.inverse_transform(y_stepwise_reg_pred.values.reshape(-1, 1))
plt.figure(figsize=(10, 6))
plt.plot(y_test_original_scale, label='Test sales values')
plt.plot(y_pred_original_scale, label='Predicted sales values')
plt.xlabel('Samples')
plt.ylabel('Sales')
plt.title('Test sales vs Predicted sales')
plt.legend()
plt.show()

#%%------------Q2.d-------------------
y_scaler = StandardScaler()
y_scaler.fit_transform(df[['Sales']])
y_test_destand_step2 = y_scaler.inverse_transform(y_test.values.reshape(-1,
1)).flatten()
y_pred_destand_step2 =
y_scaler.inverse_transform(y_stepwise_reg_pred.values.reshape(-1, 1)).flatten()
mse_1 = mean_squared_error(y_test_destand_step2,y_pred_destand_step2)
print(f"Mean Squared Error (MSE): {mse_1:.3f} ")

#%%------------Q3.a-------------------
pca = PCA()
a = pca.fit_transform(df_train.drop(['Sales'], axis=1))
explained_variance = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance)
print(cumulative_explained_variance)
n_components_95 = np.argmax(cumulative_explained_variance > 0.95) + 1
print(f"Number of components needed to explain more than 95% of the variance:
{n_components_95}")

#%%------------Q3.b-------------------
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cumulative_explained_variance) + 1),
cumulative_explained_variance, marker='o', linestyle='-', color='b')
plt.title('Cumulative Explained Variance vs Number of Principal Components')
plt.xticks(np.arange(1, 12, step=1))
plt.yticks(np.arange(0.1, 1.1, step=0.1))
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid(True)
plt.show()

#%%------------Q3.c-------------------
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cumulative_explained_variance) + 1),
cumulative_explained_variance, marker='o', linestyle='-', color='b')
plt.title('Cumulative Explained Variance vs Number of Principal Components')
plt.xticks(np.arange(1, len(cumulative_explained_variance) + 1, step=1))
plt.yticks(np.arange(0.1, 1.1, step=0.1))
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.axhline(y=0.95, color='b', linestyle='--', label='95% Variance Threshold')
plt.axvline(x=n_components_95, color='r', linestyle='-', label=f'{n_components_95}
Components')
plt.scatter(n_components_95, 0.95, color='red')
plt.legend()
plt.grid(True)
```

```
plt.show()

#%%------------Q4.a-------------------
rf_model = RandomForestRegressor(n_estimators=100, random_state=5805)
rf_model.fit(X_train, y_train)
importances = rf_model.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': importances
})
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)
plt.figure(figsize=(12, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'],
color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance from Random Forest')
plt.show()

#%%------------Q4.b-------------------
threshold = 0.02
rf_selected_features = feature_importance_df[feature_importance_df['Importance'] >
threshold]['Feature'].tolist()
rf_eliminated_features = feature_importance_df[feature_importance_df['Importance']
<= threshold]['Feature'].tolist()

print("Selected Features:", rf_selected_features)
print("Eliminated Features:", rf_eliminated_features)

#%%------------Q4.c-------------------

X_rf_train_selected = X_train[rf_selected_features]
X_rf_test_selected = X_test[rf_selected_features]
X_train_rf = sm.add_constant(X_rf_train_selected)
ols_model = sm.OLS(y_train, X_train_rf).fit()
print(ols_model.summary())

#%%------------Q4.d-------------------
X_test_ols = sm.add_constant(X_rf_test_selected)
y_pred = ols_model.predict(X_test_ols)
y_scaler = StandardScaler()
y_scaler.fit(df[['Sales']])
y_test_destand = y_scaler.inverse_transform(y_test.values.reshape(-1, 1)).flatten()
y_pred_destand_step4 = y_scaler.inverse_transform(y_pred.values.reshape(-1,
1)).flatten()
plt.figure(figsize=(10, 6))
plt.plot(y_test_destand, label='Original Test Sales', linestyle='-')
plt.plot(y_pred_destand_step4, label='Predicted Sales', linestyle='-')
plt.xlabel('Samples')
plt.ylabel('Sales')
plt.title('Original Test Sales vs Predicted Sales')
plt.legend()
plt.show()

#%%------------Q4.e-------------------
mse = mean_squared_error(y_test_destand, y_pred_destand_step4)
print(f"Mean Squared Error (MSE): {mse:.3f}")
```

```
#%%------------Q5--------------------
r_squared_step2 = final_model.rsquared
adj_r_squared_step2 = final_model.rsquared_adj
aic_step2 = final_model.aic
bic_step2 = final_model.bic
mse_step2 = mean_squared_error(y_test_destand, y_pred_destand_step2)
r_squared_step4 = ols_model.rsquared
adj_r_squared_step4 = ols_model.rsquared_adj
aic_step4 = ols_model.aic
bic_step4 = ols_model.bic
mse_step4 = mean_squared_error(y_test_destand, y_pred_destand_step4)
table = PrettyTable()
table.field_names = ["Metric", "Stepwise Regression (Step 2)", "Random Forest OLS
(Step 4)"]
table.add_row(["R-squared", round(r_squared_step2, 3), round(r_squared_step4, 3)])
table.add_row(["Adjusted R-squared", round(adj_r_squared_step2, 3),
round(adj_r_squared_step4, 3)])
table.add_row(["AIC", round(aic_step2, 3), round(aic_step4, 3)])
table.add_row(["BIC", round(bic_step2, 3), round(bic_step4, 3)])
table.add_row(["MSE", round(mse_step2, 3), round(mse_step4, 3)])
print(table)

#%%------------Q6--------------------
X_stepwise_reg_test = sm.add_constant(X_test[X_model_train.columns.drop('const')],
has_constant='add')
predictions_with_intervals = final_model.get_prediction(X_stepwise_reg_test)
prediction_summary = predictions_with_intervals.summary_frame(alpha=0.05)
predicted_means = prediction_summary['mean']
lower_bounds = prediction_summary['obs_ci_lower']
upper_bounds = prediction_summary['obs_ci_upper']
scaler = StandardScaler()
scaler.fit(df_one_hot_encode[['Sales']])
y_pred_destand = scaler.inverse_transform(predicted_means.values.reshape(-1,
1)).flatten()
lower_bound_destand = scaler.inverse_transform(lower_bounds.values.reshape(-1,
1)).flatten()
upper_bound_destand = scaler.inverse_transform(upper_bounds.values.reshape(-1,
1)).flatten()
prediction_results = pd.DataFrame({
    'Predicted Sales': y_pred_destand,
    'Lower Bound (95% CI)': lower_bound_destand,
    'Upper Bound (95% CI)': upper_bound_destand
})
print("Trying to print prediction intervals...\n",prediction_results.head())
plt.figure(figsize=(10, 6))
plt.plot(y_pred_destand, label='Predicted Sales', color='red')
plt.fill_between(np.arange(len(y_test_destand)), lower_bound_destand,
upper_bound_destand, color='lightgrey', label='95% Prediction Interval')
plt.xlabel('# of Samples')
plt.ylabel('Sales USD($)')
plt.title('Predicted Sales with Confidence Intervals')
plt.legend()
plt.grid(True)
plt.show()

#%%------------Q7.a,b--------------------
X_pr_price = df['Price']
y_pr = df['Sales']
```

```
X_pr_price = X_pr_price.values.reshape(-1, 1)
y_pr=y_pr.values.reshape(-1,1)
param_grid = {'polynomialfeatures__degree': np.arange(1, 15)}
model = make_pipeline(PolynomialFeatures(), LinearRegression())
grid_search = GridSearchCV(model, param_grid,
scoring='neg_mean_squared_error',cv=5)
grid_search.fit(np.array(X_pr_price).reshape(-1,1),np.array(y_pr).reshape(-1,1))
best_degree = grid_search.best_params_['polynomialfeatures__degree']
print("Optimum Order (n):", best_degree)

#%%------------Q7.c---------------------
rmse_scores = np.sqrt(-grid_search.cv_results_['mean_test_score'])
plt.figure(figsize=(8, 6))
plt.plot(np.arange(1, 15), rmse_scores, marker='o', label='RMSE')
min_rmse_index = np.argmin(rmse_scores)
min_rmse_value = rmse_scores[min_rmse_index]
min_degree = np.arange(1, 15)[min_rmse_index]
plt.scatter(min_degree, min_rmse_value, color='orange', label=f'Minimum RMSE
(n={min_degree})', zorder=5)
plt.xlabel('Polynomial Degree (n)')
plt.ylabel('RMSE')
plt.title('RMSE vs Polynomial Degree')
plt.grid(True)
plt.legend()
plt.show()

#%%------------Q7.d---------------------
X_train, X_test, y_train, y_test = train_test_split(X_pr_price, y_pr,
test_size=0.2, random_state=5805)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plt.figure(figsize=(8, 6))
plt.plot(np.arange(len(y_test)), y_test, label='Test set')
plt.plot(np.arange(len(y_test)), y_pred, label='Linear Regression')
plt.xlabel('Observations')
plt.ylabel('Sales')
plt.title('Regression Model - Carseats dataset')
plt.legend()
plt.grid(True)
plt.show()

#%%------------Q7.e---------------------
mse = mean_squared_error(y_test, y_pred)
print(f'MSE for Polynomial Degree {best_degree}: {round(mse,3)}')
```