**Assignment No: 3**

import pandas as pd

df = pd.read\_csv("/content/sample\_data/Housing.csv") # Replace with your actual dataset file

print(df.head())

**Output:** price area bedrooms bathrooms stories mainroad guestroom basement \

0 13300000 7420 4 2 3 yes no no

1 12250000 8960 4 4 4 yes no no

2 12250000 9960 3 2 2 yes no yes

3 12215000 7500 4 2 2 yes no yes

4 11410000 7420 4 1 2 yes yes yes

hotwaterheating airconditioning parking prefarea furnishingstatus

0 no yes 2 yes furnished

1 no yes 3 no furnished

2 no no 2 yes semi-furnished

3 no yes 3 yes furnished

4 no yes 2 no furnished

import pandas as pd

df = pd.read\_csv("/content/sample\_data/Housing.csv")

print(df.dtypes)

**Output:** price int64

area int64

bedrooms int64

bathrooms int64

stories int64

mainroad object

guestroom object

basement object

hotwaterheating object

airconditioning object

parking int64

prefarea object

furnishingstatus object

dtype: object

import pandas as pd

df = pd.read\_csv("/content/sample\_data/Housing.csv

# Count column types

num\_categorical = df.select\_dtypes(include=['object']).shape[1]

num\_integer = df.select\_dtypes(include=['int64']).shape[1]

num\_float = df.select\_dtypes(include=['float64']).shape[1]

# Display results

print(f"Number of Categorical Columns: {num\_categorical}")

print(f"Number of Integer Columns: {num\_integer}")

print(f"Number of Float Columns: {num\_float}")

**Output:** Number of Categorical Columns: 7

Number of Integer Columns: 6

Number of Float Columns: 0

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.feature\_selection import chi2

from sklearn.preprocessing import LabelEncoder

df = pd.read\_csv("/content/sample\_data/Housing.csv")

categorical\_cols = df.select\_dtypes(include=['object']).columns

encoder = LabelEncoder()

df\_encoded = df.copy()

for col in categorical\_cols:

df\_encoded[col] = encoder.fit\_transform(df[col])

X = df\_encoded[categorical\_cols] # Features

y = df\_encoded.iloc[:, -1] # Target variable

chi\_scores, p\_values = chi2(X, y)

plt.figure(figsize=(10, 5))

plt.bar(categorical\_cols, chi\_scores, color='skyblue')

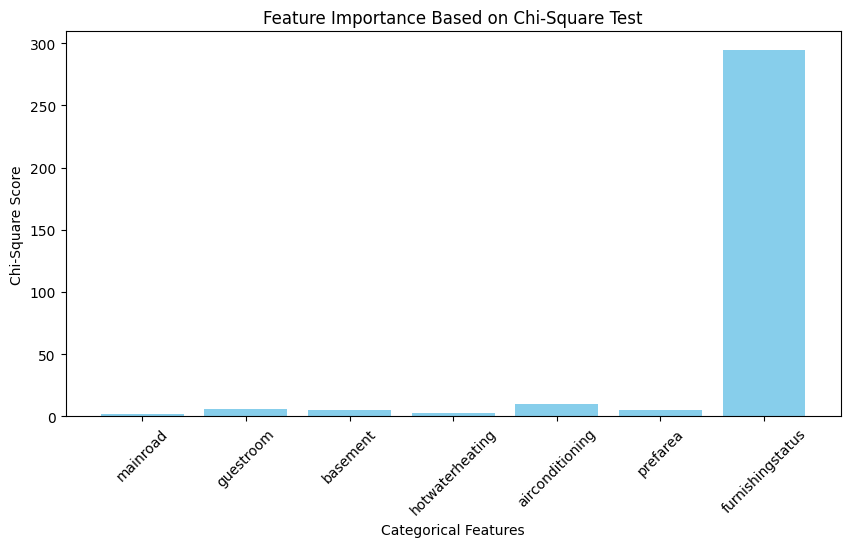
plt.xlabel("Categorical Features")

plt.ylabel("Chi-Square Score")

plt.title("Feature Importance Based on Chi-Square Test")

plt.xticks(rotation=45)

plt.show()



new\_df=df.dropna()

new\_df.isnull().sum()

**Output:** 0

price 0

area 0

bedrooms 0

bathrooms 0

stories 0

mainroad 0

guestroom 0

basement 0

hotwaterheating 0

airconditioning 0

parking 0

prefarea 0

furnishingstatus 0

dtype: int64

import pandas as pd

categorical\_cols = df.select\_dtypes(include=['object']).columns

df[categorical\_cols] = df[categorical\_cols].replace({'Yes': 1, 'No': 0})

df\_encoded = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

print(df\_encoded.head().to\_string(index=False))

import pandas as pd

from sklearn.model\_selection import train\_test\_split

X = df.drop(columns=['price']) # Features (all columns except 'Price')

y = df['price'] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print("Training Features Shape:", X\_train.shape)

print("Testing Features Shape:", X\_test.shape)

print("Training Target Shape:", y\_train.shape)

print("Testing Target Shape:", y\_test.shape)

**Output:** Training Features Shape: (436, 12)

Testing Features Shape: (109, 12)

Training Target Shape: (436,)

Testing Target Shape: (109,)

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVR

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

# Preprocess data

X = pd.get\_dummies(df.drop(columns=['price']), drop\_first=True) # One-Hot Encoding

y = df['price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize features

scaler = StandardScaler()

X\_train, X\_test = scaler.fit\_transform(X\_train), scaler.transform(X\_test)

# Train & predict with SVM

svm = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)

svm.fit(X\_train, y\_train)

y\_pred = svm.predict(X\_test)

# Evaluate model

print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred):.2f}, RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred)):.2f}, R²: {r2\_score(y\_test, y\_pred):.2f}")

**Output:** MAE: 1762754.37, RMSE: 2358789.58, R²: -0.10

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

X = pd.get\_dummies(df.drop(columns=['price']), drop\_first=True) # One-Hot Encoding for categorical data

y = df['price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize features (optional for tree-based models, but good practice)

scaler = StandardScaler()

X\_train, X\_test = scaler.fit\_transform(X\_train), scaler.transform(X\_test)

# Train & predict with Random Forest

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

y\_pred = rf.predict(X\_test)

# Evaluate model

print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred):.2f}, RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred)):.2f}, R²: {r2\_score(y\_test, y\_pred):.2f}")

**Output:** MAE: 1021151.08, RMSE: 1399758.20, R²: 0.61

from sklearn.metrics import r2\_score

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

r2 = r2\_score(y\_test, rf.predict(X\_test))

print(f"R² Score: {r2:.2f}")

**Output:** R² Score: 0.61

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit(X\_train, y\_train)

r2 = r2\_score(y\_test, lr.predict(X\_test))

print(f"R² Score: {r2:.2f}")

**Output:** R² Score: 0.65

from sklearn.linear\_model import Lasso

lasso = Lasso(alpha=0.1) # Adjust alpha for regularization strength

lasso.fit(X\_train, y\_train)

r2 = r2\_score(y\_test, lasso.predict(X\_test))

print(f"R² Score: {r2:.2f}")

R² Score: 0.65