CAR PRICE PREDICTION - CASE STUDY (REGRESSION)

=================================

1️⃣ Import Libraries

=================================

```
import pandas as pd,import numpy as np import matplotlib.pyplot as plt ,
import seaborn as sns , import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import StandardScaler, LabelEncoder,
OneHotEncoder
from sklearn.model_selection import train_test_split, KFold,
cross_val_score, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.inspection import permutation_importance
import pickle
```

=================================

2️⃣ Load Dataset

=================================

```
df = pd.read_csv("Cars.csv") , df.head() , df.shape , df.info()
df.describe() , df.isnull().sum()
```

=================================

3️⃣ Data Cleaning & Feature Engineering

=================================

```
# Mapping 'owner' column
owner_mapping = {"First Owner": 1, "Second Owner": 2, "Third Owner":3,
                 "Fourth & Above Owner":4, "Test Drive Car": 5}
df['owner'] = df['owner'].map(owner_mapping)

# Filter fuel column to remove LPG and CNG
df = df[~df['fuel'].isin(['LPG','CNG'])]
# Extract numeric part of mileage and convert to float
df['mileage'] = df['mileage'].str.split().str[0].astype(float)
# Rename 'name' to 'brand'
df = df.rename(columns={"name": "brand"})
# Drop unnecessary columns
df.drop(columns=['torque'], inplace=True)

# Remove Test Drive Cars
df = df[df['owner'] != 5]

# Getting categorical and numerical
```

```
cat_col = df.select_dtypes(exclude=['int64','float64'])

num_col = df.select_dtypes(include = ['int64','float64'])
```

==================================

4️⃣ Encode Categorical Columns

==================================

```
label_brand = LabelEncoder()
df['brand'] = label_brand.fit_transform(df['brand'])
```

==================================

5️⃣ Explore & Visualize Data

==================================

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")

# Optional: Boxplots to detect outliers
col_dict = {"brand":1, "year":2,"fuel":3,"mileage":4,"max_power":5}
plt.figure(figsize=(20,30))
for variable, i in col_dict.items():
    plt.subplot(5,11,i)
    plt.boxplot(df[variable])
    plt.title(variable)
```

==================================

6️⃣ Prepare Features and Target

==================================

```
X = df[['year','max_power','brand','mileage','fuel']]
y = np.log(df['selling_price'])  # log-transform target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fill missing numeric values
X_train['mileage'].fillna(X_train['mileage'].mean(), inplace=True)
X_train['max_power'].fillna(X_train['max_power'].median(), inplace=True)
X_test['mileage'].fillna(X_test['mileage'].mean(), inplace=True)
X_test['max_power'].fillna(X_test['max_power'].median(), inplace=True)
# Fill missing values with mode
X_test['category_col'].fillna(X_test['category_col'].mode()[0],
inplace=True)
# Get distribution of existing categories
probs = X_test['category_col'].value_counts(normalize=True)
# Fill missing values randomly based on this distribution
X_test['category_col'] = X_test['category_col'].apply(
```

```
    lambda x: np.random.choice(probs.index, p=probs.values) if
pd.isna(x) else x )
```

================================
7️⃣ **Scale Numerical Columns**
================================

```
num_cols = ['max_power','mileage']
scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

================================
8️⃣ **Train Random Forest Model**
================================

```
rfr = RandomForestRegressor(random_state=1)
rfr.fit(X_train, y_train)

yhat = rfr.predict(X_test)
print("MSE:", mean_squared_error(y_test, yhat))
print("R2:", r2_score(y_test, yhat))

# Feature importance
feature_importances = rfr.feature_importances_
print("Feature Importances:", feature_importances)
```

================================
9️⃣ **Compare Multiple Models with Cross-Validation**
================================

```
algorithms = [LinearRegression(), SVR(), KNeighborsRegressor(),
              DecisionTreeRegressor(random_state=0),
RandomForestRegressor(n_estimators=100, random_state=0)]
algorithm_names = ["Linear Regression", "SVR", "KNeighbors Regressor",
"Decision-Tree Regressor", "Random-Forest Regressor"]

kfold = KFold(n_splits=5, shuffle=True, random_state=1)

for i, model in enumerate(algorithms):
    scores = cross_val_score(model, X_train, y_train, cv=kfold,
scoring='neg_mean_squared_error')
    print(f"{algorithm_names[i]} - Score: {-scores}; Mean:
{-scores.mean()}")
```

================================
🔟 **Grid Search for Random Forest**
================================

```
param_grid = {'max_depth': [5, 10, None],
```

```
              'n_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 20,
30, 100]}

grid = GridSearchCV(estimator=RandomForestRegressor(random_state=1),
                    param_grid=param_grid,
                    cv=kfold,
                    n_jobs=-1,
                    return_train_score=True,
                    refit=True,
                    scoring='neg_mean_squared_error')
grid.fit(X_train, y_train)

best_params = grid.best_params_
best_mse = -grid.best_score_
print("Best Parameters:", best_params)
print("Best MSE:", best_mse)

# Predict on test set with best model
yhat_best = grid.predict(X_test)
print("Test MSE (best):", mean_squared_error(y_test, yhat_best))
print("Test R2 (best):", r2_score(y_test, yhat_best))
```

================================

1️⃣1️⃣ Feature Importance (Permutation)

================================

```
perm_importance = permutation_importance(grid.best_estimator_, X_test,
y_test)
sorted_idx = perm_importance.importances_mean.argsort()
plt.barh(X_train.columns[sorted_idx],
perm_importance.importances_mean[sorted_idx])
plt.xlabel("Permutation Feature Importance")
```

================================

1️⃣2️⃣ Save Models and Objects

================================

```
pickle.dump(grid, open('Model/car-prediction.model', 'wb'))
pickle.dump(label_brand, open('Model/brand-label.model', 'wb'))
pickle.dump(scaler, open('Model/car-scaler.model', 'wb'))
pickle.dump(rfr, open('Model/feature_importance.model', 'wb'))

# Load model example
loaded_model = pickle.load(open('Model/car-prediction.model', 'rb'))
predicted_price = loaded_model.predict(X_test[:1])
print("Predicted Price (first sample):", np.exp(predicted_price))
```

BRAIN STROKE PRICE PREDICTION - CLASS ASSIGNMENT (CLASSIFICATION)

Step 1️⃣: Import Libraries ================

Libraries for data handling, preprocessing, modeling, and evaluation

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split, KFold,
cross_val_score, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report
import pickle
```

Step 2️⃣: Load Dataset ===========================

Load Brain Stroke dataset and inspect

```
df = pd.read_csv("brain_stroke.csv")
df.head(), df.info()
df.isnull().sum()
```

Step 3️⃣: Split Features and Target ==============

Separate predictors (X) and target (y)

```
X = df.drop(columns=['stroke'])
y = df['stroke']
```

Step 4️⃣: Handle Missing Values ==================

Fill missing categorical/numeric values

```
X['gender'] = X['gender'].fillna(X['gender'].mode()[0])
X['heart_disease'] =
X['heart_disease'].fillna(X['heart_disease'].mode()[0])
X['avg_glucose_level'] =
X['avg_glucose_level'].fillna(X['avg_glucose_level'].mode()[0])
```

Step 5️⃣: Encode Categorical Variables ==========

Encode categorical columns and apply one-hot encoding

```
le = LabelEncoder()
for col in ['gender','ever_married','Residence_type']:
    X[col] = le.fit_transform(X[col])
X = pd.get_dummies(X, columns=['work_type','smoking_status'])
```

Step 6️⃣: Train-Test Split ======================

Split dataset into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=41)
```

Step 7️⃣: Feature Scaling ===================

Scale numeric columns using MinMaxScaler

```
cols_to_scale = ['age','avg_glucose_level']
scaler = MinMaxScaler()
X_train[cols_to_scale] = scaler.fit_transform(X_train[cols_to_scale])
X_test[cols_to_scale] = scaler.transform(X_test[cols_to_scale])
```

Step 8️⃣: Handle Class Imbalance ===========

Resample majority class to match minority class

```
cond0 = y_train == 0
cond1 = y_train == 1
y_train_0 = y_train[cond0].sample(n=sum(cond1), random_state=999)
y_train_1 = y_train[cond1]
y_train = pd.concat([y_train_0, y_train_1])
X_train = X_train.loc[y_train.index]
```

Step 9️⃣: Model Training & Evaluation ============

Train Logistic Regression, Random Forest, SVC and evaluate using cross-validation

```
models = [LogisticRegression(random_state=999),
RandomForestClassifier(random_state=999), SVC(random_state=999)]
kfold = KFold(n_splits=5, shuffle=True, random_state=999)

for model in models:
    score = cross_val_score(model, X_train, y_train, cv=kfold,
scoring='accuracy')
    print(f"{model.__class__.__name__} Accuracy: {score.mean():.3f}")
```

Step 🔟: Hyperparameter Tuning (Logistic Regression) =========

GridSearchCV for Logistic Regression

```
param_grid = {'solver': ['newton-cg', 'lbfgs', 'liblinear']}
grid = GridSearchCV(LogisticRegression(random_state=999), param_grid,
scoring="accuracy", cv=kfold, refit=True)
grid.fit(X_train, y_train)
print("Best Parameters:", grid.best_params_)
print("Best Cross-Validation Score:", grid.best_score_)
```

Step 1️⃣1️⃣: Model Prediction & Evaluation =================

Predict on test set and generate classification report

```
y_pred = grid.predict(X_test)
print(classification_report(y_test, y_pred))
```

Step 1️⃣2️⃣: Save & Load Model ===========================

Save model using pickle and reload for inference

```
pickle.dump(grid, open('stroke.model', 'wb'))
loaded_model = pickle.load(open('stroke.model', 'rb'))
```

```
Example prediction for a sample
sample = X_test.iloc[:1]
pred = loaded_model.predict(sample)
print("Prediction for first sample:", pred)
```