

```
import os

os.listdir('/content')

['.config', 'sample_data']
```

```
import pandas as pd

df = pd.read_csv('/content/train.csv')

df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN

5 rows × 15 columns

```
# Check the shape of the dataset
print("Dataset shape:", df.shape)

# Check data types and non-null values
df.info()

# Check missing values
missing_values = df.isnull().sum()

print("\nMissing values:")
print(missing_values[missing_values > 0])
```

```

MasVnrType      872
MasVnrArea       8
BsmtQual        37
BsmtCond        37
BsmtExposure    38
BsmtFinType1    37
BsmtFinType2    38
Electrical       1
FireplaceQu     690
GarageType      81
GarageYrBlt     81
GarageFinish    81
GarageQual      81
GarageCond      81
PoolQC         1453
Fence          1179
MiscFeature     1406
dtype: int64

```

```

# Separate numerical and categorical columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns

# Fill missing numerical values with the median
for col in numerical_cols:
    df[col] = df[col].fillna(df[col].median())

# Fill missing categorical values with "None"
for col in categorical_cols:
    df[col] = df[col].fillna('None')

# Check missing values after treatment
missing_after = df.isnull().sum()

print("Missing values after treatment:")
print(missing_after[missing_after > 0])

```

```

Missing values after treatment:
Series([], dtype: int64)

```

Handle missing values appropriately to prepare the datasets for applying machine learning models.

```

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# Separate features and target
X = df.drop('SalePrice', axis=1)
y = df['SalePrice']

# Identify numerical and categorical features
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='median'), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)

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

print("Training data:", X_train.shape)
print("Testing data:", X_test.shape)

```

```

Training data: (1168, 80)
Testing data: (292, 80)

```

```

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

```

```

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Create Random Forest pipeline
rf_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', RandomForestRegressor(random_state=42))
])

# Define parameter grid
rf_param_grid = {
    'model__n_estimators': [100, 200],
    'model__max_depth': [None, 10, 20],
    'model__min_samples_split': [2, 5]
}

# Grid Search
rf_grid = GridSearchCV(
    rf_pipeline,
    rf_param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

# Train the model
rf_grid.fit(X_train, y_train)

# Best parameters
print("Best Parameters:")
print(rf_grid.best_params_)

# Predictions
rf_pred = rf_grid.predict(X_test)

# Evaluation
rf_mae = mean_absolute_error(y_test, rf_pred)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))
rf_r2 = r2_score(y_test, rf_pred)

print("\nRandom Forest Performance:")
print("MAE:", rf_mae)
print("RMSE:", rf_rmse)
print("R²:", rf_r2)

```

```

Best Parameters:
{'model__max_depth': None, 'model__min_samples_split': 2, 'model__n_estimators': 200}

Random Forest Performance:
MAE: 17502.51866438356
RMSE: 29019.079980160863
R²: 0.8902123612284031

```

```

from sklearn.ensemble import GradientBoostingRegressor

# Create Gradient Boosting pipeline
gb_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', GradientBoostingRegressor(random_state=42))
])

# Define parameter grid
gb_param_grid = {
    'model__n_estimators': [100, 150],
    'model__learning_rate': [0.05, 0.1],
    'model__max_depth': [2, 3]
}

# Grid Search
gb_grid = GridSearchCV(
    gb_pipeline,
    gb_param_grid,
    cv=3,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)

```

```
# Train the model
gb_grid.fit(X_train, y_train)

# Best parameters
print("Best Parameters:")
print(gb_grid.best_params_)

# Predictions
gb_pred = gb_grid.predict(X_test)

# Evaluation
gb_mae = mean_absolute_error(y_test, gb_pred)
gb_rmse = np.sqrt(mean_squared_error(y_test, gb_pred))
gb_r2 = r2_score(y_test, gb_pred)

print("\nGradient Boosting Performance:")
print("MAE:", gb_mae)
print("RMSE:", gb_rmse)
print("R²:", gb_r2)
```

```
Best Parameters:
{'model__learning_rate': 0.1, 'model__max_depth': 2, 'model__n_estimators': 150}

Gradient Boosting Performance:
MAE: 18061.119734365508
RMSE: 29429.483929988328
R²: 0.8870850468427396
```

```
from sklearn.feature_selection import SelectKBest, f_regression

# Transform the training and testing data
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

print("Original number of features:", X_train_processed.shape[1])

# Select the 20 most important features
selector = SelectKBest(score_func=f_regression, k=20)

X_train_selected = selector.fit_transform(X_train_processed, y_train)
X_test_selected = selector.transform(X_test_processed)

print("Selected number of features:", X_train_selected.shape[1])
```

```
Original number of features: 302
Selected number of features: 20
```

```
# Get feature names after preprocessing
feature_names = preprocessor.get_feature_names_out()

# Get the selected feature names
selected_features = feature_names[selector.get_support()]

print("Selected Features:")
for feature in selected_features:
    print(feature)
```

```
Selected Features:
num__OverallQual
num__YearBuilt
num__YearRemodAdd
num__MasVnrArea
num__TotalBsmtSF
num__1stFlrSF
num__GrLivArea
num__FullBath
num__TotRmsAbvGrd
num__Fireplaces
num__GarageYrBlt
num__GarageCars
num__GarageArea
cat__ExterQual_TA
cat__Foundation_PConc
cat__BsmtQual_Ex
cat__BsmtQual_TA
cat__KitchenQual_Ex
cat__KitchenQual_TA
cat__FireplaceQu_None
```

```
# Train Random Forest using selected features

rf_selected = RandomForestRegressor(
    n_estimators=200,
    max_depth=None,
    min_samples_split=2,
    random_state=42,
    n_jobs=-1
)

# Train the model
rf_selected.fit(X_train_selected, y_train)

# Make predictions
rf_selected_pred = rf_selected.predict(X_test_selected)

# Evaluate the model
rf_selected_mae = mean_absolute_error(y_test, rf_selected_pred)
rf_selected_rmse = np.sqrt(mean_squared_error(y_test, rf_selected_pred))
rf_selected_r2 = r2_score(y_test, rf_selected_pred)

print("Random Forest with SelectKBest Features:")
print("MAE:", rf_selected_mae)
print("RMSE:", rf_selected_rmse)
print("R²:", rf_selected_r2)
```

```
Random Forest with SelectKBest Features:
MAE: 18836.004345686564
RMSE: 29712.61198685766
R²: 0.8849019862734998
```

```
from sklearn.feature_selection import SelectFromModel

# Use Random Forest for feature importance
rf_importance = RandomForestRegressor(
    n_estimators=200,
    random_state=42,
    n_jobs=-1
)

# Train on all preprocessed features
rf_importance.fit(X_train_processed, y_train)

# Select the most important features
model_selector = SelectFromModel(
    rf_importance,
    max_features=20,
    threshold=-float('inf')
)

X_train_rf_selected = model_selector.fit_transform(
    X_train_processed, y_train
)

X_test_rf_selected = model_selector.transform(
    X_test_processed
)

print("Original number of features:", X_train_processed.shape[1])
print("Selected number of features:", X_train_rf_selected.shape[1])
```

```
Original number of features: 302
Selected number of features: 20
```

```
# Get the selected feature names
rf_selected_features = feature_names[model_selector.get_support()]

print("Selected Features using Random Forest Feature Importance:")

for feature in rf_selected_features:
    print(feature)
```

```
Selected Features using Random Forest Feature Importance:
num__Id
num__LotFrontage
num__LotArea
num__OverallQual
```

```

num__YearBuilt
num__YearRemodAdd
num__BsmtFinSF1
num__BsmtUnfSF
num__TotalBsmtSF
num__1stFlrSF
num__2ndFlrSF
num__GrLivArea
num__FullBath
num__TotRmsAbvGrd
num__GarageYrBlt
num__GarageCars
num__GarageArea
num__WoodDeckSF
num__OpenPorchSF
cat__BsmtQual_Ex

```

```
# Gradient Boosting using SelectKBest features
```

```

gb_selected = GradientBoostingRegressor(
    n_estimators=150,
    learning_rate=0.1,
    max_depth=2,
    random_state=42
)

# Train the model
gb_selected.fit(X_train_selected, y_train)

# Make predictions
gb_selected_pred = gb_selected.predict(X_test_selected)

# Evaluate the model
gb_selected_mae = mean_absolute_error(y_test, gb_selected_pred)
gb_selected_rmse = np.sqrt(mean_squared_error(y_test, gb_selected_pred))
gb_selected_r2 = r2_score(y_test, gb_selected_pred)

print("Gradient Boosting with SelectKBest Features:")
print("MAE:", gb_selected_mae)
print("RMSE:", gb_selected_rmse)
print("R²:", gb_selected_r2)

```

```

Gradient Boosting with SelectKBest Features:
MAE: 19971.353984099693
RMSE: 29512.761249464693
R²: 0.8864451063695608

```

```
# Gradient Boosting using Feature Importance selected features
```

```

gb_importance_selected = GradientBoostingRegressor(
    n_estimators=150,
    learning_rate=0.1,
    max_depth=2,
    random_state=42
)

# Train the model
gb_importance_selected.fit(X_train_rf_selected, y_train)

# Make predictions
gb_importance_pred = gb_importance_selected.predict(X_test_rf_selected)

# Evaluate the model
gb_importance_mae = mean_absolute_error(y_test, gb_importance_pred)
gb_importance_rmse = np.sqrt(mean_squared_error(y_test, gb_importance_pred))
gb_importance_r2 = r2_score(y_test, gb_importance_pred)

print("Gradient Boosting with Feature Importance:")
print("MAE:", gb_importance_mae)
print("RMSE:", gb_importance_rmse)
print("R²:", gb_importance_r2)

```

```

Gradient Boosting with Feature Importance:
MAE: 19223.998605030418
RMSE: 29642.138500728553
R²: 0.8854473263375592

```

```
# Train Random Forest using features selected by Feature Importance

rf_importance_selected = RandomForestRegressor(
    n_estimators=200,
    max_depth=None,
    min_samples_split=2,
    random_state=42,
    n_jobs=-1
)

# Train the model
rf_importance_selected.fit(X_train_rf_selected, y_train)

# Make predictions
rf_importance_pred = rf_importance_selected.predict(X_test_rf_selected)

# Evaluate the model
rf_importance_mae = mean_absolute_error(y_test, rf_importance_pred)
rf_importance_rmse = np.sqrt(mean_squared_error(y_test, rf_importance_pred))
rf_importance_r2 = r2_score(y_test, rf_importance_pred)

print("Random Forest with Feature Importance:")
print("MAE:", rf_importance_mae)
print("RMSE:", rf_importance_rmse)
print("R²:", rf_importance_r2)
```

```
Random Forest with Feature Importance:
MAE: 18291.80751712329
RMSE: 28936.252090642647
R²: 0.8908381909144019
```

```
# Final comparison of all models

results = pd.DataFrame({
    'Model': [
        'Random Forest',
        'Gradient Boosting',
        'Random Forest + SelectKBest',
        'Gradient Boosting + SelectKBest',
        'Random Forest + Feature Importance',
        'Gradient Boosting + Feature Importance'
    ],
    'MAE': [
        rf_mae,
        gb_mae,
        rf_selected_mae,
        gb_selected_mae,
        rf_importance_mae,
        gb_importance_mae
    ],
    'RMSE': [
        rf_rmse,
        gb_rmse,
        rf_selected_rmse,
        gb_selected_rmse,
        rf_importance_rmse,
        gb_importance_rmse
    ],
    'R2': [
        rf_r2,
        gb_r2,
        rf_selected_r2,
        gb_selected_r2,
        rf_importance_r2,
        gb_importance_r2
    ]
})

print("Final Model Comparison:")
display(results)
```

Final Model Comparison:

	Model	MAE	RMSE	R2	
0	Random Forest	17502.518664	29019.079980	0.890212	
1	Gradient Boosting	18061.119734	29429.483930	0.887085	
2	Random Forest + SelectKBest	18836.004346	29712.611987	0.884902	
3	Gradient Boosting + SelectKBest	19971.353984	29512.761249	0.886445	
4	Random Forest + Feature Importance	18291.807517	28936.252091	0.890838	
5	Gradient Boosting + Feature Importance	19223.998605	29642.138501	0.885447	

Next steps:

[Generate code with results](#)

[New interactive sheet](#)