

```
from IPython.display import Image, display
display(Image(filename='Slide1.PNG'))
```



Summary

What is structured data?

- Data organized in a predefined format (rows, columns, and schemas).
- Storage: Relational databases (SQL) and spreadsheets.

Examples:

- Patient records in a hospital database (age, gender, diagnosis).
- Financial transactions (date, amount, account number).

We focus on this type of data to apply classification and regression models.

Classification Models

- Purpose: Predict categories (class labels).
- Output type: Discrete values (e.g., Yes/No, 0/1, or multiple classes).

Examples of tasks:

- Email spam vs. not spam
- Disease present vs. absent

Common algorithms:

- Logistic Regression
- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- Naive Bayes
- KNN

Evaluation Methods for Classification Models

accuracy, precision, recall and F1-score.

Regression Models

- Purpose: Predict continuous numerical values.
- Output type: Continuous numbers (e.g., 2.5, 100.75).

Examples of tasks:

- Predicting house prices
- Forecasting temperature
- Estimating blood glucose levels

Common algorithms:

- Linear Regression
- Ridge/Lasso Regression
- Support Vector Regression (SVR)
- Decision Trees Regression
- Random Forest Regression

Evaluation Methods for Regression Models

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R^2 Score (Coefficient of Determination)
- MAPE (Mean Absolute Percentage Error)

Import necessary libraries

```
import pandas as pd
import numpy as np
```

```

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error
from sklearn.model_selection import GridSearchCV

import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

```

Reading dataset

```

data = pd.read_csv("WineQT.csv")
data.head()

```

	fixed acidity	volatile acidity	citric acid	residual sugar
chlorides \				
0	7.4	0.70	0.00	1.9
0.076				
1	7.8	0.88	0.00	2.6
0.098				
2	7.8	0.76	0.04	2.3
0.092				
3	11.2	0.28	0.56	1.9
0.075				
4	7.4	0.70	0.00	1.9
0.076				

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
\					
0	11.0	34.0	0.9978	3.51	0.56
1	25.0	67.0	0.9968	3.20	0.68
2	15.0	54.0	0.9970	3.26	0.65
3	17.0	60.0	0.9980	3.16	0.58
4	11.0	34.0	0.9978	3.51	0.56

alcohol	quality	Id
---------	---------	----

0	9.4	5	0
1	9.8	5	1
2	9.8	5	2
3	9.8	6	3
4	9.4	5	4

Define features and target variable

```
X = data.drop(['quality', 'Id'], axis=1)
y = data['quality']
```

Feature Scaling (Input Normalization) Methods

These are used to scale input features before training a regression model.

Min-Max Normalization

Scales features to a fixed range, usually [0, 1].

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

```
from sklearn.preprocessing import MinMaxScaler

# Create scaler
scaler = MinMaxScaler(feature_range=(0, 1))

# Fit and transform
X_normalized = scaler.fit_transform(X)

print(X_normalized)

[[0.24778761 0.39726027 0.         ... 0.60629921 0.13772455
 0.15384615]
 [0.28318584 0.52054795 0.         ... 0.36220472 0.20958084
 0.21538462]
 [0.28318584 0.43835616 0.04        ... 0.40944882 0.19161677
 0.21538462]
 ...
 [0.14159292 0.32876712 0.08        ... 0.55905512 0.1497006
 0.32307692]
 [0.11504425 0.29452055 0.1         ... 0.61417323 0.25748503
 0.43076923]
 [0.11504425 0.35958904 0.12        ... 0.65354331 0.22754491
 0.27692308]]
```

Z-score Normalization (Standardization)

Centers data around mean = 0 and std = 1.

$$x_{\text{std}} = \frac{x - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_normalized1 = scaler.fit_transform(X)
print(X_normalized1)
```

```
[[-0.52157961  0.93933222 -1.36502663 ...  1.27069495 -0.57365783
  -0.96338181]
 [-0.29259344  1.94181282 -1.36502663 ... -0.70892755  0.1308811
  -0.59360107]
 [-0.29259344  1.27349242 -1.16156762 ... -0.32577481 -0.04525363
  -0.59360107]
 ...
 [-1.20853813  0.38239855 -0.9581086 ...  0.88754221 -0.45623467
  0.05351522]
 [-1.38027776  0.10393172 -0.8563791 ...  1.33455374  0.60057372
  0.70063152]
 [-1.38027776  0.6330187  -0.75464959 ...  1.65384769  0.30701583
  -0.22382033]]
```

Splitting data

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split( X_normalized, y,
test_size=0.25, random_state=42)
```

Evaluation models

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

def evaluate_model(model_name, y_true, y_pred):
    """
    Evaluate a regression model using MAE, MSE, RMSE, and R^2 score.
    Additionally, plot y_true vs. y_pred with a reference line and
    display model name and R^2 score on the plot.

    Parameters:
        model_name (str): Name of the regression model.
        y_true (array-like): True target values.
        y_pred (array-like): Predicted target values.

    Returns:
        dict: Dictionary containing MAE, MSE, RMSE, and R^2 score.
    """
    # Calculate metrics
```

```

mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_true, y_pred)

# Print the evaluation results
results = {
    "Mean Absolute Error (MAE)": mae,
    "Mean Squared Error (MSE)": mse,
    "Root Mean Squared Error (RMSE)": rmse,
    "R^2 Score": r2
}

return results

```

Ensemble learning

Ensemble learning for regression combines multiple regression models to produce more accurate and stable predictions than a single model.

1. Bagging – Random Forest Regressor

Trains multiple models on different random samples of the training data and averages their predictions.

```

from sklearn.ensemble import RandomForestRegressor, BaggingRegressor

# Create base estimator
base_model = RandomForestRegressor(
    n_estimators=100,
    random_state=42,
    n_jobs=-1
)

# Create Bagging model
model = BaggingRegressor(
    estimator=base_model,    # use 'base_estimator' if sklearn < 1.2
    n_estimators=10,
    random_state=42,
    n_jobs=-1,
    bootstrap=True
)

# Train
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

```

```

# Evaluate
evaluation_results = evaluate_model('Bagging – Random Forest
Regressor', y_test, y_pred)

# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

```

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 0.4332
Mean Squared Error (MSE): 0.3094
Root Mean Squared Error (RMSE): 0.5562
R^2 Score: 0.4586

```

2. Boosting – Gradient Boosting Regressor

```

from sklearn.ensemble import GradientBoostingRegressor

# Create model
model = GradientBoostingRegressor(
    n_estimators=100,
    learning_rate=0.1,
    random_state=42
)

# Train
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
evaluation_results = evaluate_model('Boosting – Gradient Boosting
Regressor', y_test, y_pred)

# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

```

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 0.4639
Mean Squared Error (MSE): 0.3473
Root Mean Squared Error (RMSE): 0.5893
R^2 Score: 0.3923

```

3. Boosting – XGBoost Regressor

```
from xgboost import XGBRegressor

model = XGBRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42
)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
evaluation_results = evaluate_model('Boosting – Gradient Boosting
Regressor', y_test, y_pred)

# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 0.4195
Mean Squared Error (MSE): 0.3335
Root Mean Squared Error (RMSE): 0.5775
R^2 Score: 0.4164
```

3. Boosting- CatBoostRegressor

```
from catboost import CatBoostRegressor

model = CatBoostRegressor(
    iterations=100,
    learning_rate=0.1,
    depth=6,
    verbose=0
)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
evaluation_results = evaluate_model('Boosting- CatBoostRegressor',
y_test, y_pred)
# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

```
Evaluation Results on Test Set:  
Mean Absolute Error (MAE): 0.4449  
Mean Squared Error (MSE): 0.3175  
Root Mean Squared Error (RMSE): 0.5635  
R^2 Score: 0.4444
```

```
#pip install catboost
```

3. Stacking (Stacked Generalization)

Combines predictions from multiple different models using a meta-model.

□ Process:

Train multiple base models

Use their predictions as input features

Train a final “meta-regressor”

Example:

Base models: Linear Regression, SVR, Random Forest

Meta-model: Ridge Regression

□ Works well when models are diverse

```
from sklearn.ensemble import StackingRegressor  
  
base_models = [  
    ('rf', RandomForestRegressor()),  
    ('svr', SVR())  
]  
  
model = StackingRegressor(  
    estimators=base_models,  
    final_estimator=SVR()  
)  
  
model.fit(X_train, y_train)  
y_pred = model.predict(X_test)  
  
evaluation_results = evaluate_model('StackingRegressor', y_test,  
y_pred)  
# Print the evaluation results  
print("\nEvaluation Results on Test Set:")  
for key, value in evaluation_results.items():  
    print(f"{key}: {value:.4f}" if isinstance(value, float) else  
f"{key}: \n{value}")
```

```
Evaluation Results on Test Set:
Mean Absolute Error (MAE): 0.3873
Mean Squared Error (MSE): 0.3198
Root Mean Squared Error (RMSE): 0.5655
R^2 Score: 0.4405
```

```
RF- fit- X_train ----- predict---- y_train_prd X_test----prdict--- y_test_prd
```

```
SVM- fit- X_train ----- predict---- y_train_prd X_test----prdict--- y_test_prd
```

```
Stacking training [y_train_prd_RF, y_train_prd_SVM]
```

```
Stacking testing [y_test_prd_RF, y_test_prd_SVM]
```

```
Stacking training -- fit meta model
```

```
Stacking testing prdict meta model
```

Voting / Averaging Regressor

Simple ensemble that averages predictions of multiple models.

Simple Averaging

```
from sklearn.ensemble import VotingRegressor

model = VotingRegressor(
    estimators=[
        ('svr', SVR()),
        ('rf', RandomForestRegressor())
    ]
)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

evaluation_results = evaluate_model('Simple Averaging', y_test,
y_pred)
# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

```
Evaluation Results on Test Set:
Mean Absolute Error (MAE): 0.4468
Mean Squared Error (MSE): 0.3270
```

Root Mean Squared Error (RMSE): 0.5719
R^2 Score: 0.4278

Weighted voting

```
voting_regressor = VotingRegressor(
    estimators=[
        ('lr', LinearRegression()),
        ('rf', RandomForestRegressor()),
        ('svr', SVR())
    ],
    weights=[0.2, 0.5, 0.3] # manually assigned weights
)

voting_regressor.fit(X_train, y_train)
y_pred = voting_regressor.predict(X_test)

evaluation_results = evaluate_model('Simple Averaging', y_test,
y_pred)
# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 0.4373
Mean Squared Error (MSE): 0.3154
Root Mean Squared Error (RMSE): 0.5616
R^2 Score: 0.4481

```
from sklearn.ensemble import VotingRegressor

# Define base models
voting_regressor = VotingRegressor(
    estimators=[
        ('lr', LinearRegression()),
        ('rf', RandomForestRegressor(random_state=42)),
        ('svr', SVR())
    ]
)

# Define hyperparameter grid
param_grid = {
    # RF hyperparameters
    'rf__n_estimators': [100, 200],
    'rf__max_depth': [None, 10],
```

```

# SVR hyperparameters
'svr__C': [0.1, 1, 10],
'svr__kernel': ['rbf', 'linear'],

# Voting weights
'weights': [
    [1, 1, 1],
    [0.2, 0.5, 0.3],
    [0.3, 0.4, 0.3]
]
}

# Grid Search
grid = GridSearchCV(
    estimator=voting_regressor,
    param_grid=param_grid,
    cv=5,
    scoring='r2',    # or 'neg_mean_squared_error'
    n_jobs=-1
)

# Fit
grid.fit(X_train, y_train)

# Best model
print("Best parameters:", grid.best_params_)
best_model = grid.best_estimator_

# Predict
y_pred = best_model.predict(X_test)
# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: \n{value}")

Best parameters: {'rf__max_depth': None, 'rf__n_estimators': 200,
'svr__C': 10, 'svr__kernel': 'linear', 'weights': [0.2, 0.5, 0.3]}

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 0.4373
Mean Squared Error (MSE): 0.3154
Root Mean Squared Error (RMSE): 0.5616
R^2 Score: 0.4481

```

New project

Download dataset

Medical Cost Personal Datasets

<https://www.kaggle.com/datasets/mirichoi0218/insurance>

- age: age of primary beneficiary
- sex: insurance contractor gender, female, male
- bmi: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height,
- objective index of body weight (kg / m^2) using the ratio of height to weight, ideally 18.5 to 24.9
- children: Number of children covered by health insurance / Number of dependents
- smoker: Smoking
- region: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.
- charges: Individual medical costs billed by health insurance

```
df = pd.read_csv('insurance.csv')
df.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

```
data.isnull().sum()
```

fixed acidity	0
volatile acidity	0
citric acid	0
residual sugar	0
chlorides	0
free sulfur dioxide	0
total sulfur dioxide	0
density	0
pH	0
sulphates	0
alcohol	0
quality	0
Id	0

```
dtype: int64
```

```
df.shape
```

```
(1338, 7)
```

```
df.describe()
```

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

```
from sklearn.preprocessing import LabelEncoder
```

```
#sex
```

```
le = LabelEncoder()
```

```
le.fit(df.sex.drop_duplicates())
```

```
df.sex = le.transform(df.sex)
```

```
# smoker or not
```

```
le.fit(df.smoker.drop_duplicates())
```

```
df.smoker = le.transform(df.smoker)
```

```
#region
```

```
le.fit(df.region.drop_duplicates())
```

```
df.region = le.transform(df.region)
```

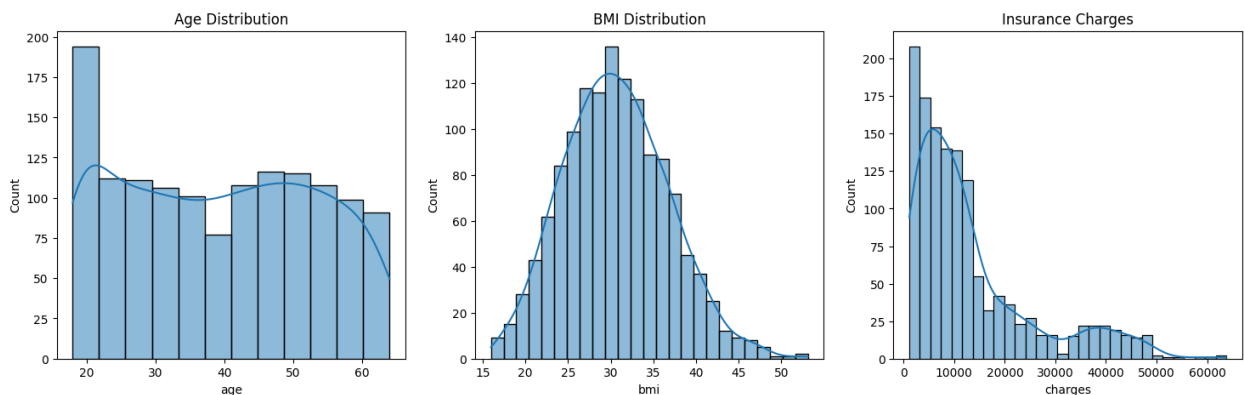
```
fig, axs = plt.subplots(1, 3, figsize=(18,5))
```

```
sns.histplot(df['age'], kde=True, ax=axs[0]).set_title('Age  
Distribution')
```

```
sns.histplot(df['bmi'], kde=True, ax=axs[1]).set_title('BMI  
Distribution')
```

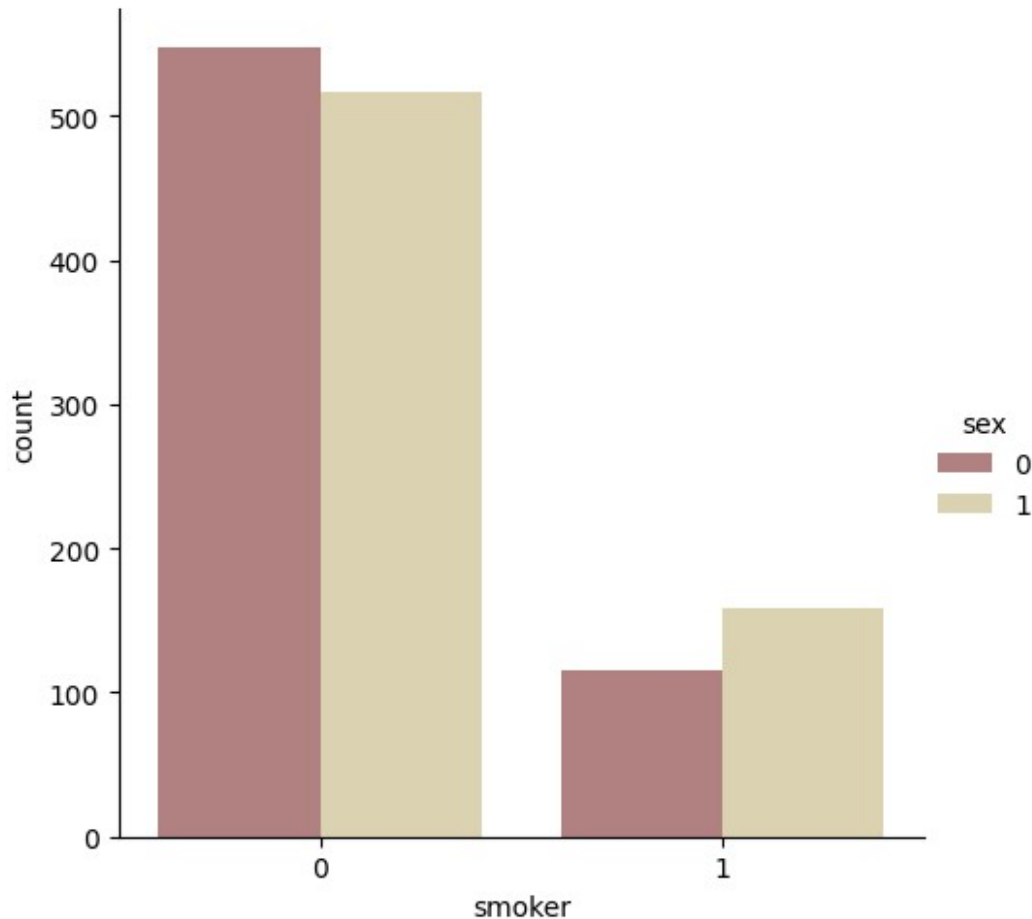
```
sns.histplot(df['charges'], kde=True, ax=axs[2]).set_title('Insurance  
Charges')
```

```
plt.show()
```



```
sns.catplot(x="smoker", kind="count", hue = 'sex', palette="pink", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x746373a92300>
```



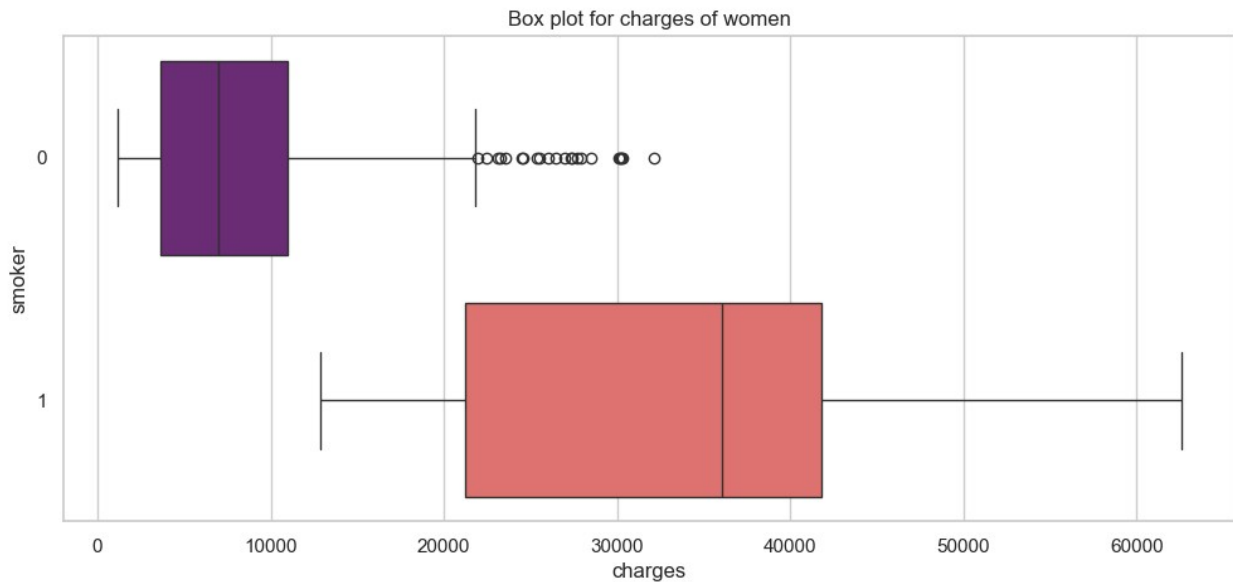
```
pl.figure(figsize=(12,5))
pl.title("Box plot for charges of women")
sns.boxplot(y="smoker", x="charges", data = df[(df.sex == 1)] ,
orient="h", palette = 'magma')
```

C:\Users\Compumarts\AppData\Local\Temp\
ipykernel_26284\2816788822.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(y="smoker", x="charges", data = df[(df.sex == 1)] ,
orient="h", palette = 'magma')
```

```
<Axes: title={'center': 'Box plot for charges of women'},
xlabel='charges', ylabel='smoker'>
```



Splitting dataset

```
X = df.drop(['charges', 'region'], axis = 1)
y = df.charges
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

Decision Tree Regressor

```
DT = DecisionTreeRegressor(random_state=42)
```

```
# Define the parameter grid
```

```
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

```
# Initialize GridSearchCV
```

```
grid_search = GridSearchCV(
    estimator=DT,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation
    scoring='neg_root_mean_squared_error', # Metric to optimize
    verbose=2,
```

```

    n_jobs=-1 # Use all available cores
)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE:", grid_search.best_score_)

# Train a new model using the best parameters
best_DT = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_DT.predict(X_test)
evaluation_results = evaluate_model('Decision Tree Regressor', y_test,
y_pred)

# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 4,
'min_samples_split': 10}
Best CV RMSE: -5475.090144305333

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 2774.9753
Mean Squared Error (MSE): 23047548.0523
Root Mean Squared Error (RMSE): 4800.7862
R^2 Score: 0.8536

```

Support Vector Regressor (SVR)

```

SVM = SVR()

# Define the parameter grid
param_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=SVM,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation

```

```

        scoring='neg_root_mean_squared_error', # Metric to optimize
        verbose=2,
        n_jobs=-1 # Use all available cores
    )

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE Score:", grid_search.best_score_)

# Train a new model using the best parameters
best_SVM = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_SVM.predict(X_test)
evaluation_results = evaluate_model('Support Vector Regressor',
y_test, y_pred)

# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

Fitting 5 folds for each of 4 candidates, totalling 20 fits
Best Parameters: {'kernel': 'poly'}
Best CV RMSE Score: -12416.632017570571

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 7954.2150
Mean Squared Error (MSE): 168123816.8864
Root Mean Squared Error (RMSE): 12966.2569
R^2 Score: -0.0679

```

Gradient Boosting Regressor

```

GBR = GradientBoostingRegressor(random_state=42)

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],

    'max_depth': [3, 5, 7],

}

# Initialize GridSearchCV
grid_search = GridSearchCV(

```

```

    estimator=GBR,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation
    scoring='neg_root_mean_squared_error', # Metric to optimize
    verbose=2,
    n_jobs=-1 # Use all available cores
)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE Score:", grid_search.best_score_)

# Train a new model using the best parameters
best_GBR = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_GBR.predict(X_test)
evaluation_results = evaluate_model('Gradient Boosting Regressor',
y_test, y_pred)

# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

Fitting 5 folds for each of 9 candidates, totalling 45 fits
Best Parameters: {'max_depth': 3, 'n_estimators': 100}
Best CV RMSE Score: -4855.168838929034

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 2414.6438
Mean Squared Error (MSE): 16420889.4069
Root Mean Squared Error (RMSE): 4052.2697
R^2 Score: 0.8957

```

KNeighborsRegressor

```

# Initialize KNeighborsRegressor
knn = KNeighborsRegressor()

# Define the parameter grid
param_grid = {
    'n_neighbors': [3, 5, 7, 10],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski'],

```

```

}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=knn,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation
    scoring='neg_root_mean_squared_error', # Metric to optimize
    verbose=2,
    n_jobs=-1 # Use all available cores
)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE Score:", grid_search.best_score_)

# Train a new model using the best parameters
best_knn = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_knn.predict(X_test)
evaluation_results = evaluate_model('KNeighborsRegressor', y_test,
y_pred)

# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

```

Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'metric': 'manhattan', 'n_neighbors': 10, 'weights':
'distance'}

```

```

Best CV RMSE Score: -10070.318477865303

```

```

Evaluation Results on Test Set:
Mean Absolute Error (MAE): 6474.5692
Mean Squared Error (MSE): 88800973.7558
Root Mean Squared Error (RMSE): 9423.4269
R^2 Score: 0.4359

```