CANCER TREATMENT COSTS

ESTI STERN

Agenda

- Business Problem
- Data Overview
- Exploratory Data Analysis
- Methodology
- Models
- Final Model
- Conclusion
- Next Steps

Business Problem

- The soaring costs of cancer treatment often place an overwhelming financial strain on patients and families.
- A key factor in managing treatment costs is how insurance companies assess and allocate coverage.
- By predicting treatment costs, insurers can set premiums that balance affordability for the customers with sustainability for the company.
- With a clearer picture of potential costs, insurers could offer personalized recommendations and better financial support for patients, improving the customer experience.

Data Overview

- **Dataset Dimensions:**
  - Rows: 50,000
  - Columns: 15 (original including target variable)
- **Time Span:**
  - Years Covered: 2015 – 2024
- **Data Quality:**
  - No null values
  - No duplicate records
- **Target Variable:**
  - Treatment Cost (USD)
- **Feature Engineering:**
  - One-hot encoding expanded features to 29
  - 18 columns discarded (from encoded Cancer Type and Country)
  - Final Model Input: 11 selected features

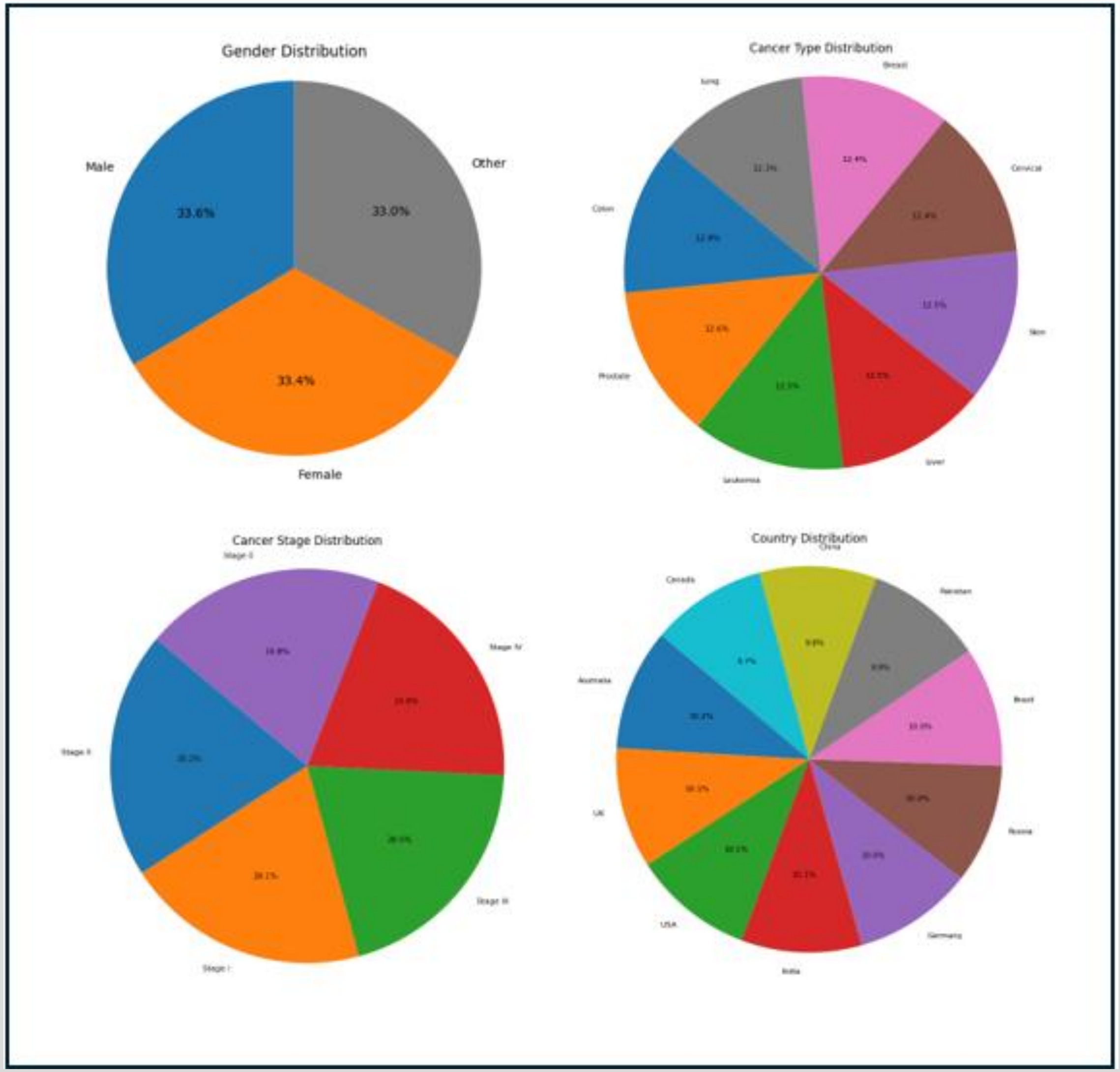
Exploratory Data Analysis – Descriptive Statistics

	Age	Year	Genetic_Risk	Air_Pollution	Alcohol_Use	Smoking	Obesity_Level	Treatment_Cost_USD	Survival_Years	Target_Severity_Score
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000
mean	54.421540	2019.480520	5.001698	5.010126	5.010880	4.989826	4.991176	52467.298239	5.006462	4.951207
std	20.224451	2.871485	2.885773	2.888399	2.888769	2.881579	2.894504	27363.229379	2.883335	1.199677
min	20.000000	2015.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5000.050000	0.000000	0.900000
25%	37.000000	2017.000000	2.500000	2.500000	2.500000	2.500000	2.500000	28686.225000	2.500000	4.120000
50%	54.000000	2019.000000	5.000000	5.000000	5.000000	5.000000	5.000000	52474.310000	5.000000	4.950000
75%	72.000000	2022.000000	7.500000	7.500000	7.500000	7.500000	7.500000	76232.720000	7.500000	5.780000
max	89.000000	2024.000000	10.000000	10.000000	10.000000	10.000000	10.000000	99999.840000	10.000000	9.160000

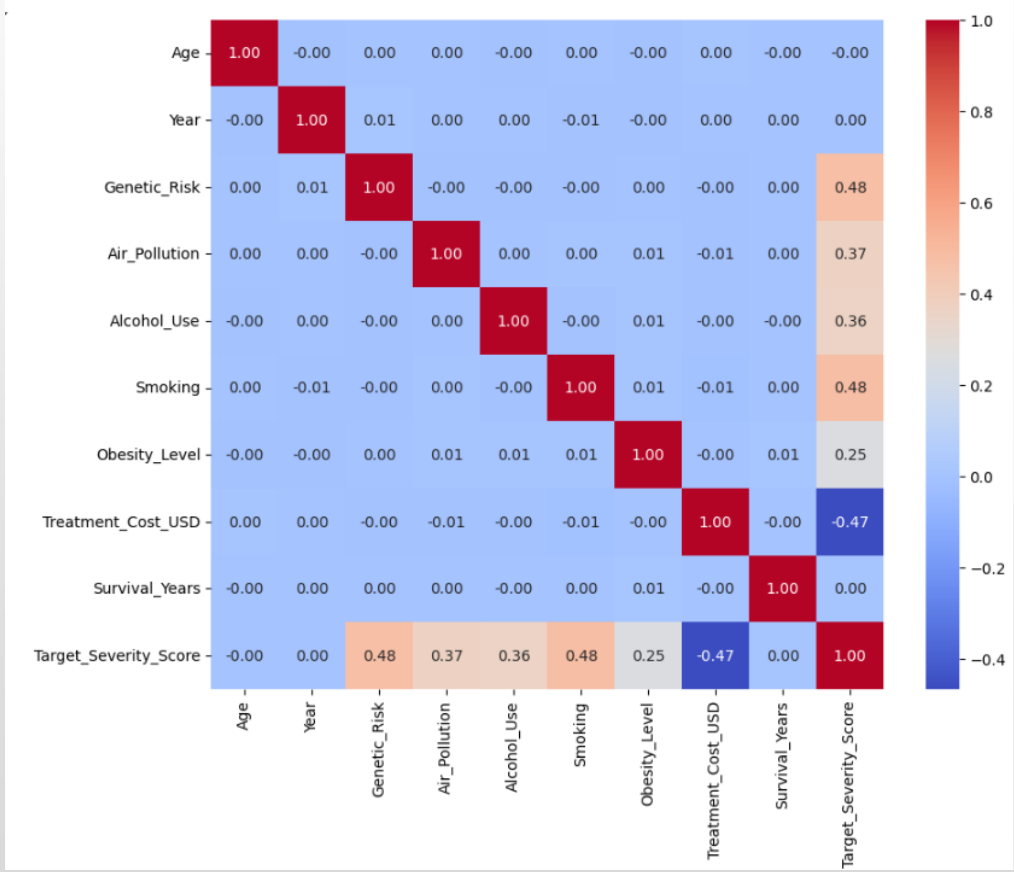
- **Age:** Mean = 54.4, Range = 20 – 89
- **Genetic Risk, Air Pollution, Alcohol Use, Smoking, Obesity Level:** All scaled 0 – 10 with similar distributions (mean ≈ 5.0)
- **Treatment Cost (USD):** Mean = $52,467, Range = $5,000 - $99,999, Std Dev = $27,363
- **Survival Years:** Mean = 5.0, Range = 0 – 10
- **Severity Score:** Mean = 4.95, Range = 0.9 – 9.16
- Most features are symmetrically distributed around their midpoints

Exploratory Data Analysis – Feature Distributions

- Gender
- Cancer Type
- Cancer Stage
- Country
- All groups are evenly represented



Exploratory Data Analysis – Correlation Heatmap



- Only Severity Score has a significant linear correlation with Treatment Cost
- Genetic Risk, Air Pollution, Alcohol Use, Smoking, and Obesity Level all have some linear correlation with Severity Score

Methodology

Step 1: Data Exploration

- Analyzed feature distributions & correlations
- Encoded categorical variables (One-hot & Ordinal)

Step 2: Modeling & Comparison

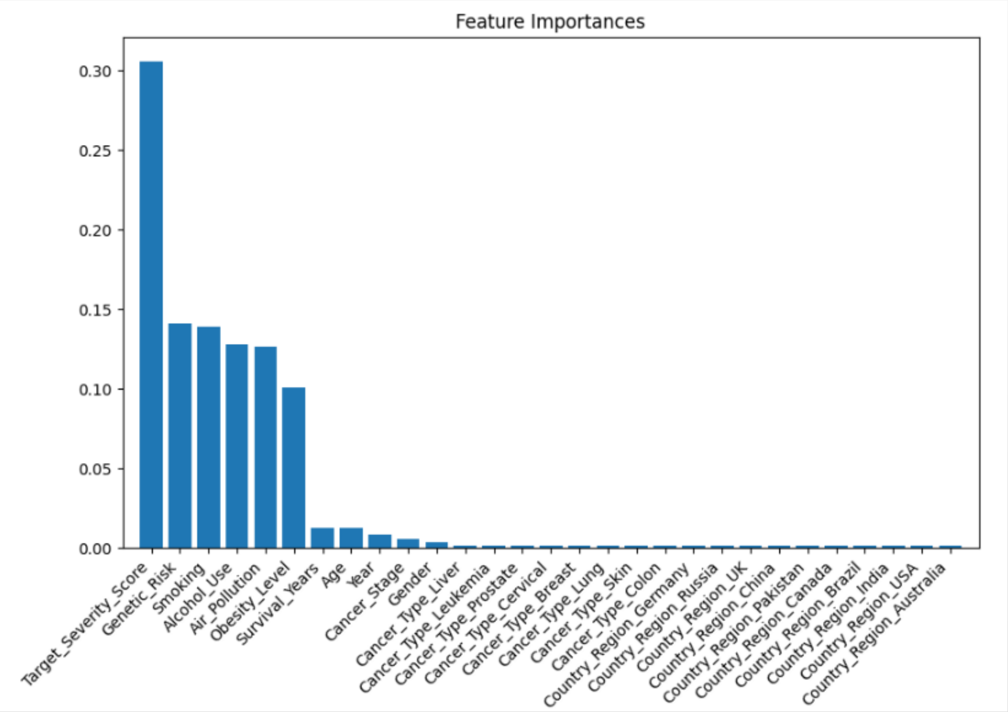
- Trained Random Forest & XGBoost
- Used LazyPredict to explore alternatives, chose Bayesian Ridge

Step 3: Evaluation & Tuning

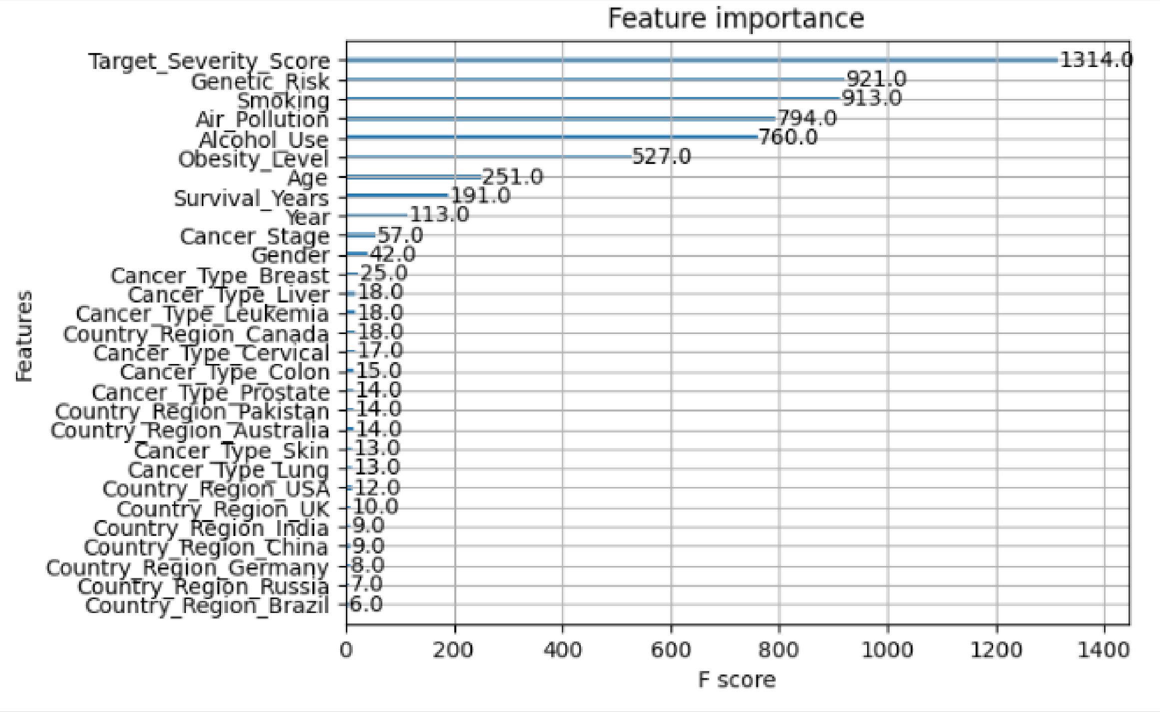
- GridSearchCV (3-fold) for hyperparameter optimization
- Metrics: RMSE & R^2

Models – Feature Selection

Random Forest



XGBoost



Final Model

- Final Model: Optimized XGBoost Regressor
- Achieved RMSE of 3,156 & R^2 of 0.9867
- Selected based on balance of accuracy and generalization

		Random Forest	XGBoost	Bayesian Ridge
Before Optimization	RMSE	9098	5042	144
	R2	0.89	0.966	0.99
After Optimization	RMSE	8862	3156	
	R2	0.89	0.98	

Conclusion

- Although performance of the XGBoost model slightly improved with all features, selected features were removed to simplify the model and reduce overfitting.
- Genetic and environmental factors – though indirect – emerged as key drivers of disease severity and treatment costs.
- Insurance providers can leverage this model to design more personalized policies based on patients' individual risk profiles and anticipated treatment costs.
- Healthcare practitioners can use the model to promote transparency, helping patients better understand potential expenses and make more informed decisions.

Next Steps

- Future improvements include adding treatment type, therapy duration, length of hospital stay, insurance coverage
- These factors may enhance predictive accuracy
- Could deepen insights into cost drivers and care pathways

Appendix

```
[ ]
# Mount google drive
from google.colab import drive
import os

drive.mount('/content/drive')
os.chdir('/content/drive/My Drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ]
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# !pip install xgboost
import xgboost as xg

from xgboost import plot_importance
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
[ ]
# Load the data
cancer_data = pd.read_csv('global_cancer_patients_2015_2024.csv')
```

```
[ ]
# Encode Gender
cancer_data['Gender'] = cancer_data['Gender'].replace({'Other':2, 'Male':1, 'Female':0})
```

```
/tmp/ipython-input-5-2577140762.py:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future versio
  cancer_data['Gender'] = cancer_data['Gender'].replace({'Other':2, 'Male':1, 'Female':0})
```

```
[ ]
# Encode Cancer Stage
cancer_data['Cancer_Stage'] = cancer_data['Cancer_Stage'].replace({'Stage 0':0, 'Stage I':1, 'Stage II':2, 'Stage III':3, 'Stage IV':4}
```

```
/tmp/ipython-input-6-2326103387.py:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future versio
  cancer_data['Cancer_Stage'] = cancer_data['Cancer_Stage'].replace({'Stage 0':0, 'Stage I':1, 'Stage II':2, 'Stage III':3, 'Stage IV':4})
```

```
[ ]
# One Hot encode Country/Region, Cancer Type
cancer_data = pd.get_dummies(cancer_data, columns=['Country_Region', 'Cancer_Type'])
```

```
# Splitting into features and target variable
X = cancer_data.drop(['Treatment_Cost_USD', 'Patient_ID'], axis=1)
y = cancer_data['Treatment_Cost_USD']
```

```
[ ]
# Splitting into training - 75% and testing - 25%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=23)
```

```python
# Random Forest Regressor
# Using all the features
rf1 = RandomForestRegressor(random_state=23)
rf1.fit(X_train, y_train)
```

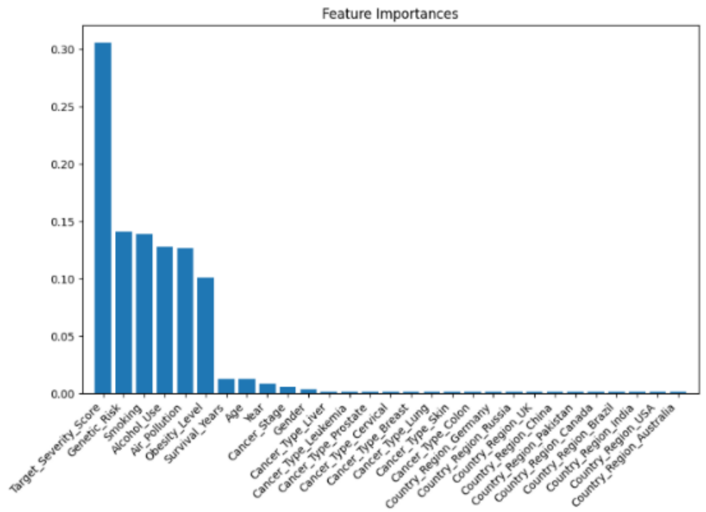
```
RandomForestRegressor
RandomForestRegressor(random_state=23)
```

```python
# Test the model on X_test
y_pred_rf1 = rf1.predict(X_test)
```

```python
# Accuracy metrics
mse_rf1 = mean_squared_error(y_test, y_pred_rf1)
mae_rf1 = mean_absolute_error(y_test, y_pred_rf1)
rmse_rf1 = np.sqrt(mse_rf1)
r2_rf1 = r2_score(y_test, y_pred_rf1)
print('MSE:', mse_rf1)
print('MAE:', mae_rf1)
print('RMSE:', rmse_rf1)
print('R2:', r2_rf1)
```

```
MSE: 82778592.60447659
MAE: 7438.889650951999
RMSE: 9098.27415527124
R2: 0.8900421379072706
```

```python
# Feature Importance
importances = rf1.feature_importances_
indices = np.argsort(importances)[::-1]
features = X.columns

plt.figure(figsize=(10, 6))
plt.title('Feature Importances')
plt.bar(range(len(features)), importances[indices], align='center')
plt.xticks(range(len(features)), [features[i] for i in indices], rotation=45, ha='right')
plt.xlim([-1, len(features)])
plt.show()
```




```python
# Remove columns with very little importance
to_drop = ['Cancer_Type_Colon','Cancer_Type_Leukemia','Cancer_Type_Lung', 'Cancer_Type_Cervical', 'Cancer_Type_Liver', 'Cancer_Type_Breast', 'Cancer_Type_Skin', 'Cancer_Type_Prostate',
           'Country_Region_UK', 'Country_Region_Russia', 'Country_Region_China', 'Country_Region_USA', 'Country_Region_Germany', 'Country_Region_Australia', 'Country_Region_Canada', 'Country_Region_Braz
X2 = cancer_data.drop(['Treatment_Cost_USD', 'Patient_ID'] + to_drop, axis=1)
```

```python
# Split into training and testing
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y, test_size = 0.25, random_state=23)
```

```python
# Random Forest Regressor
# With less features
rf2 = RandomForestRegressor(random_state=23)
rf2.fit(X_train2, y_train2)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=23)
```

```python
# Test the model on X_test
y_pred_rf2 = rf2.predict(X_test2)
```

```python
# Accuracy metrics
mse_rf2 = mean_squared_error(y_test2, y_pred_rf2)
mae_rf2 = mean_absolute_error(y_test2, y_pred_rf2)
rmse_rf2 = np.sqrt(mse_rf2)
r2_rf2 = r2_score(y_test2, y_pred_rf2)
print('MSE:', mse_rf2)
print('MAE:', mae_rf2)
print('RMSE:', rmse_rf2)
print('R2:', r2_rf2)
```

```
MSE: 80033645.19954082
MAE: 7304.561677632
RMSE: 8946.152536120811
R2: 0.8936883529334898
```

```python
# XGBoost
# With all features
xgb1 = xg.XGBRegressor(objective='reg:squarederror', seed=23)
xgb1.fit(X_train, y_train)
```

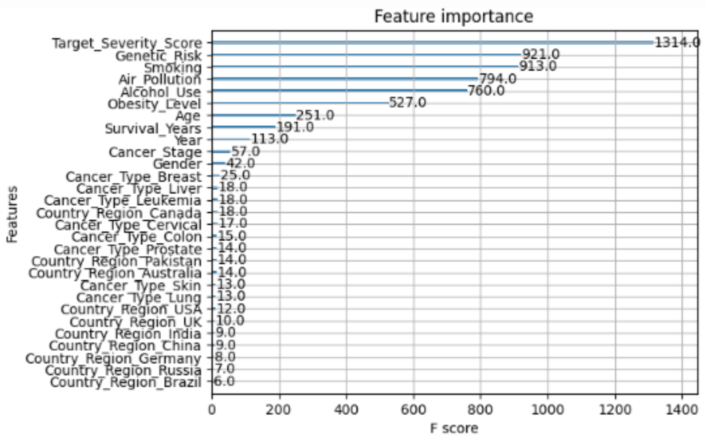
```
XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```python
# Test the model on X_test
y_pred_xgb1 = xgb1.predict(X_test)
```

```python
# Metrics
rmse_xgb1 = np.sqrt(mean_squared_error(y_test, y_pred_xgb1))
r2_xgb1 = r2_score(y_test, y_pred_xgb1)
print('RMSE:', rmse_xgb1)
print('R2:', r2_xgb1)
```

```
RMSE: 5042.690719554216
R2: 0.9662220791051046
```

```python
# Feature Importance
plot_importance(xgb1)
```




```python
# XGBoost
# With less features
xgb2 = xg.XGBRegressor(objective='reg:squarederror', seed=23)
xgb2.fit(X_train2, y_train2)
```

```
XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```python
# Test the model on X_test
y_pred_xgb2 = xgb2.predict(X_test2)
```

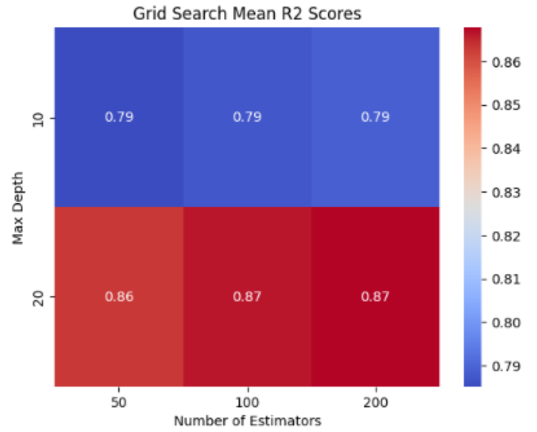
```python
# Metrics
rmse_xgb2 = np.sqrt(mean_squared_error(y_test2, y_pred_xgb2))
r2_xgb2 = r2_score(y_test2, y_pred_xgb2)
print('RMSE:', rmse_xgb2)
print('R2:', r2_xgb2)
```

```
RMSE: 5051.359034992242
R2: 0.9661058517387933
```

```python
# Randomized Search Cross Validation
# Random Forest Regressor
param_distributions = {
    'n_estimators': [i for i in range(5, 101)],
    'max_depth': [1, 3, 5],
    'min_samples_leaf': [1, 3, 5]
}

rscv = RandomizedSearchCV(RandomForestRegressor(), param_distributions)
rscv.random_state = 23
rscv.fit(X_train2, y_train2)

print(f'selected_params: {rscv.best_params_},',
      f'train accuracy: {rscv.score(X_train2, y_train2):.4f},',
      f' test accuracy: {rscv.score(X_test2, y_test2):.4f}')
```

```
selected_params: {'n_estimators': 52, 'min_samples_leaf': 3, 'max_depth': 5}, train accuracy: 0.4258,  test accuracy: 0.4169
```

```python
# Grid Search
# Random Forest Regressor
# Default values: n_estimators=100, max_depth=None, min_samples_split=2, min_samples_leaf=1

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 3, 5]
}

grid_search = GridSearchCV(estimator=RandomForestRegressor(random_state=23), param_grid=param_grid, cv=3, n_jobs=-1, scoring='r2')
grid_search.fit(X_train2, y_train2)

results = pd.DataFrame(grid_search.cv_results_)

pivot_table = results.pivot_table(
    values='mean_test_score',
    index='param_max_depth',
    columns='param_n_estimators'
)

sns.heatmap(pivot_table, annot=True, cmap='coolwarm')
plt.title('Grid Search Mean R2 Scores')
plt.xlabel('Number of Estimators')
plt.ylabel('Max Depth')
plt.show()
```




```python
# Best parameters from grid search
print(grid_search.best_params_)
```

```
{'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}
```

```python
# Random Forest with best parameters from grid search
rf4 = RandomForestRegressor(n_estimators=200, min_samples_split=2, max_depth=None, random_state=23)
rf4.fit(X_train2, y_train2)
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=200, random_state=23)
```

```python
# Test the model on X_test
y_pred_rf4 = rf4.predict(X_test2)
```

```python
# Accuracy metrics
mse_rf4 = mean_squared_error(y_test2, y_pred_rf4)
mae_rf4 = mean_absolute_error(y_test2, y_pred_rf4)
rmse_rf4 = np.sqrt(mse_rf4)
r2_rf4 = r2_score(y_test2, y_pred_rf4)
print('MSE:', mse_rf4)
print('MAE:', mae_rf4)
print('RMSE:', rmse_rf4)
print('R2:', r2_rf4)
```

```
MSE: 78552477.34062521
MAE: 7235.518864732001
RMSE: 8862.983546223315
R2: 0.8956558429093706
```

```python
# Grid Search XGBoost
# Default values: max_depth=6, learning_rate=0.3, n_estimators=100

param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300]
}

xgb_grid_search = GridSearchCV(estimator=xg.XGBRegressor(objective='reg:squarederror', seed=23), param_grid=param_grid, cv=3, scoring='r2', n_jobs=-1)
xgb_grid_search.fit(X_train, y_train)

print(xgb_grid_search.best_params_)print(xgb_grid_search.best_score_)
```

```
GridSearchCV
best_estimator_:
XGBRegressor
XGBRegressor
```

```python
# XGBoost with parameters from grid search
xgb3 = xg.XGBRegressor(objective='reg:squarederror', seed=23, max_depth=3, learning_rate=0.2, n_estimators=300)
xgb3.fit(X_train, y_train)
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.2, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=300, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```python
# Test the model on X_test
y_pred_xgb3 = xgb3.predict(X_test)
```

```python
# Metrics
rmse_xgb3 = np.sqrt(mean_squared_error(y_test, y_pred_xgb3))
r2_xgb3 = r2_score(y_test, y_pred_xgb3)
print('RMSE:', rmse_xgb3)
print('R2:', r2_xgb3)
```

```
RMSE: 3158.3536292263802
R2: 0.9867495765181149
```

```
!pip install lazypredict scikit-learn pandas
from lazypredict.Supervised import LazyRegressor
reg = LazyRegressor(verbose=1, ignore_warnings=True, custom_metric=mean_squared_error)
models, predictions = reg.fit(X_train, X_test, y_train, y_test)
print(models)
```

```
Requirement already satisfied: starlette<0.47.0,>=0.40.0 in /usr/local/lib/python3.11/dist-packages (from fastapi<1->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredict) (0.46
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-packages (from gitpython<4,>=3.1.9->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredict) (4.0.
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.11/dist-packages (from importlib_metadata!=4.7.0,<9,>=3.7.0->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypr
Collecting opentelemetry-semantic-conventions==0.55b1 (from opentelemetry-sdk<3,>=1.9.0->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredict)
  Downloading opentelemetry_semantic_conventions-0.55b1-py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<3,>=1.10.8->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredict
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<3,>=1.10.8->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredict)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<3,>=1.10.8->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredi
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.17.3->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredi
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.17.3->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredict) (3.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.17.3->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredict) (2
Requirement already satisfied: h11>=0.8 in /usr/local/lib/python3.11/dist-packages (from uvicorn<1->mlflow-skinny==3.1.1->mlflow>=2.0.0->lazypredict) (0.16.0)
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.11/dist-packages (from gitdb<5,>=4.0.1->gitpython<4,>=3.1.9->mlflow-skinny==3.1.1->mlflow>=2.0.0->l
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from google-auth~=2.0->databricks-sdk<1,>=0.20.0->mlflow-skinny==3.1.1->mlf
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.11/dist-packages (from google-auth~=2.0->databricks-sdk<1,>=0.20.0->mlflow-skinny==3.1.1->mlflow>=2.0
Requirement already satisfied: anyio<5,>=3.6.2 in /usr/local/lib/python3.11/dist-packages (from starlette<0.47.0,>=0.40.0->fastapi<1->mlflow-skinny==3.1.1->mlflow>=2.0.0->l
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5,>=3.6.2->starlette<0.47.0,>=0.40.0->fastapi<1->mlflow-skinny==3.1.1->ml
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1->google-auth~=2.0->databricks-sdk<1,>=0.20.0->mlf
Downloading lazypredict-0.2.16-py2.py3-none-any.whl (14 kB)
Downloading mlflow-3.1.1-py3-none-any.whl (24.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.7/24.7 MB 35.7 MB/s eta 0:00:00
Downloading mlflow_skinny-3.1.1-py3-none-any.whl (1.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 52.6 MB/s eta 0:00:00
Downloading pytest_runner-6.0.1-py3-none-any.whl (7.2 kB)
Downloading alembic-1.16.2-py3-none-any.whl (242 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 242.7/242.7 kB 13.3 MB/s eta 0:00:00
Downloading docker-7.1.0-py3-none-any.whl (147 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 147.8/147.8 kB 8.2 MB/s eta 0:00:00
Downloading graphene-3.4.3-py2.py3-none-any.whl (114 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.9/114.9 kB 5.4 MB/s eta 0:00:00
Downloading gunicorn-23.0.0-py3-none-any.whl (85 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 85.0/85.0 kB 5.0 MB/s eta 0:00:00
Downloading databricks_sdk-0.57.0-py3-none-any.whl (733 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 733.8/733.8 kB 28.7 MB/s eta 0:00:00
Downloading graphql_core-3.2.6-py3-none-any.whl (203 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 203.4/203.4 kB 10.3 MB/s eta 0:00:00
Downloading graphql_relay-3.2.0-py3-none-any.whl (16 kB)
Downloading opentelemetry_api-1.34.1-py3-none-any.whl (65 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.8/65.8 kB 3.4 MB/s eta 0:00:00
Downloading opentelemetry_sdk-1.34.1-py3-none-any.whl (118 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 118.5/118.5 kB 8.0 MB/s eta 0:00:00
Downloading opentelemetry_semantic_conventions-0.55b1-py3-none-any.whl (196 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.2/196.2 kB 9.7 MB/s eta 0:00:00
Installing collected packages: pytest-runner, gunicorn, graphql-core, opentelemetry-api, graphql-relay, docker, alembic, opentelemetry-semantic-conventions, graphene, datab
Successfully installed alembic-1.16.2 databricks-sdk-0.57.0 docker-7.1.0 graphene-3.4.3 graphql-core-3.2.6 graphql-relay-3.2.0 gunicorn-23.0.0 lazypredict-0.2.16 mlflow-3.1
 24%  10/42 [00:34<02:46,  5.20s/it]
{'Model': 'AdaBoostRegressor', 'R-Squared': 0.3286490447844169, 'Adjusted R-Squared': 0.3270877634932179, 'RMSE': np.float64(22481.261695240097), 'Time taken': 5.384049415
{'Model': 'BaggingRegressor', 'R-Squared': 0.8650137183745837, 'Adjusted R-Squared': 0.8646997967894083, 'RMSE': np.float64(10080.70005218789), 'Time taken': 3.92496871948
{'Model': 'BayesianRidge', 'R-Squared': 0.9999723891685212, 'Adjusted R-Squared': 0.9999723249572852, 'RMSE': np.float64(144.17355859043957), 'Time taken': 0.125431776046
{'Model': 'DecisionTreeRegressor', 'R-Squared': 0.6332886709465174, 'Adjusted R-Squared': 0.632435853902207, 'RMSE': np.float64(16615.29433416246), 'Time taken': 0.5808100
{'Model': 'DummyRegressor', 'R-Squared': -2.8252380731563775e-05, 'Adjusted R-Squared': -0.0023538994792915435, 'RMSE': np.float64(27437.972111720595), 'Time taken': 0.043
{'Model': 'ElasticNet', 'R-Squared': 0.3667519701627481, 'Adjusted R-Squared': 0.36527930032591727, 'RMSE': np.float64(21833.974519624204), 'Time taken': 0.0637905597686767
{'Model': 'ElasticNetCV', 'R-Squared': 0.03068291003815682, 'Adjusted R-Squared': 0.028428684247547853, 'RMSE': np.float64(27013.372692612338), 'Time taken': 0.463208436965
{'Model': 'ExtraTreeRegressor', 'R-Squared': 0.5630714733012421, 'Adjusted R-Squared': 0.5620553604484542, 'RMSE': np.float64(18136.399584598657), 'Time taken': 0.28869032
{'Model': 'ExtraTreesRegressor', 'R-Squared': 0.906168429528893, 'Adjusted R-Squared': 0.905950216574309, 'RMSE': np.float64(8404.664234821757), 'Time taken': 23.796477794
{'Model': 'GammaRegressor', 'R-Squared': 0.22996286966297297, 'Adjusted R-Squared': 0.22817208563893332, 'RMSE': np.float64(24076.963198336074), 'Time taken': 0.148133754
```

```
# Bayesian Ridge
# Standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
from sklearn.linear_model import BayesianRidge
br1 = BayesianRidge()
br1.fit(X_train_scaled, y_train)
```

```
BayesianRidge
BayesianRidge()
```

```
# Standardize X_test
X_test_scaled = scaler.transform(X_test)
# Test on X_test
y_pred_br1 = br1.predict(X_test_scaled)
```

```
# Metrics
rmse_br1 = np.sqrt(mean_squared_error(y_test, y_pred_br1))
r2_br1 = r2_score(y_test, y_pred_br1)
print('RMSE:', rmse_br1)
print('R2:', r2_br1)
```

```
RMSE: 144.1255113387782
R2: 0.9999724075686152
```