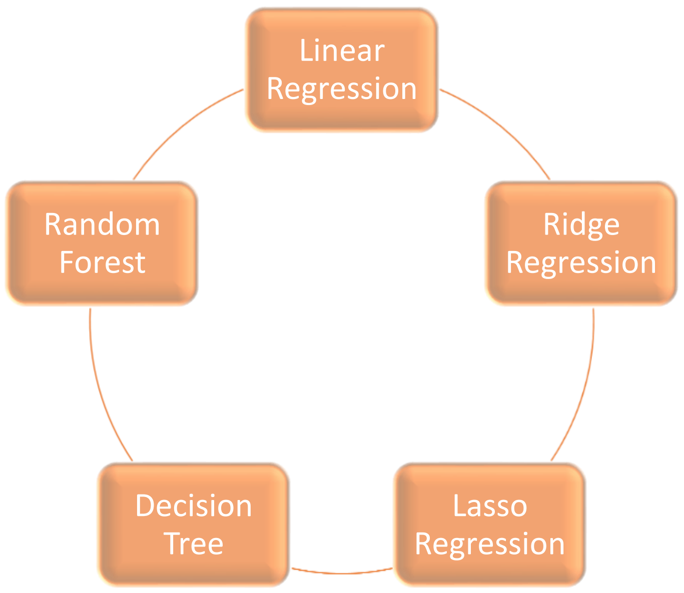
FUTURE SALES PREDICTION

INTRODUCTION :

Predicting future sales using a dataset involves employing various data analysis and machine learning techniques to forecast sales trends, helping businesses make informed decisions. Firstly, dataset should be pre-processed.

We then developed multiple models, including Linear Regression, Ridge Regression, Lasso Regression, Decision Tree, and Random Forest, all of which were evaluated through cross-validation. The model evaluation results revealed that Random Forest outperformed others, yielding an average Mean Squared Error (MSE) of 10.32%, Root Mean Squared Error (RMSE) of 8.09%, Mean Absolute Error (MAE) of 5.99%, and an R-squared value of 94.27%.

Additionally, we conducted classic assumption tests, including tests for linearity, homoscedasticity, normality, multicollinearity, outliers, and independence, to ensure the validity of our model. These results provide in-depth insights into the quality of our prediction model and its relevance in the context of future sales forecasting.



Modelling and Evaluation:

At the modeling stage, we use 5 algorithms for comparison, namely Linear Regression, Ridge Regression, Lasso Regression, Decision Tree, and Random Forest.

And for evaluation using MSE, RMSE, MAE and R-Squared.

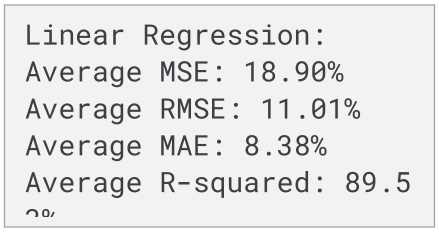
Dataset used here : https://www.kaggle.com/datasets/chakradharmattapalli/future-sales-prediction

CODE :

- ✓ **# Linear Regression**

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso

linear_model = LinearRegression()

linear_mse, linear_rmse, linear_mae, linear_r2 = perform_cross_validation(linear_model, X, y, num_folds)

print("Linear Regression:")

print(f"Average MSE: {np.mean(linear_mse) / np.mean(y) * 100:.2f}%")

print(f"Average RMSE: {np.mean(linear_rmse) / np.mean(y) * 100:.2f}%")

print(f"Average MAE: {np.mean(linear_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(linear_r2) * 100:.2f}%")

print("\n")
```

OUTPUT:



```
Linear Regression:
Average MSE: 18.90%
Average RMSE: 11.01%
Average MAE: 8.38%
Average R-squared: 89.5
2%
```

- ✓ **# Ridge Regression**

```
ridge_model = Ridge(alpha=1.0)  # You can adjust alpha as needed

ridge_mse, ridge_rmse, ridge_mae, ridge_r2 = perform_cross_validation(ridge_model, X, y, num_folds)

print("Ridge Regression:")

print(f"Average MSE: {np.mean(ridge_mse) / np.mean(y) * 100:.2f}%")

print(f"Average RMSE: {np.mean(ridge_rmse) / np.mean(y) * 100:.2f}%")
print(f"Average MAE: {np.mean(ridge_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(ridge_r2) * 100:.2f}%")

print("\n")
```

OUTPUT :

```
Ridge Regression:
Average MSE: 19.67%
Average RMSE: 11.20%
Average MAE: 8.54%
Average R-squared: 89.19
%
```

- ✓ **# Lasso Regression**

```
lasso_model = Lasso(alpha=1.0)  # You can adjust alpha as needed

lasso_mse, lasso_rmse, lasso_mae, lasso_r2 = perform_cross_validation(lasso_model, X, y, num_folds)

print("Lasso Regression:")

print(f"Average MSE: {np.mean(lasso_mse) / np.mean(y) * 100:.2f}%")

print(f"Average RMSE: {np.mean(lasso_rmse) / np.mean(y) * 100:.2f}%")

print(f"Average MAE: {np.mean(lasso_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(lasso_r2) * 100:.2f}%")

print("\n")
```

OUTPUT:

```
Lasso Regression:
Average MSE: 115.55%
Average RMSE: 27.51%
Average MAE: 22.39%
Average R-squared: 35
.98%
```

- ✓ **# Decision Trees**

```
from sklearn.tree import DecisionTreeRegressor

tree_model = DecisionTreeRegressor(max_depth=None, random_state=0)  # You can adjust parameters as needed

tree_mse, tree_rmse, tree_mae, tree_r2 = perform_cross_validation(tree_model, X, y, num_folds)

print("Decision Trees:")

print(f"Average MSE: {np.mean(tree_mse) / np.mean(y) * 100:.2f}%")
```

print(f"Average RMSE: {np.mean(tree_rmse) / np.mean(y) * 100:.2f}%")

print(f"Average MAE: {np.mean(tree_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(tree_r2) * 100:.2f}%")

print("\n")

<u>OUTPUT</u>:

```
Decision Trees:
Average MSE: 16.73%
Average RMSE: 10.40%
Average MAE: 7.56%
Average R-squared: 90.6
5%
```

- ✓ **# Random Forest**

from sklearn.ensemble import RandomForestRegressor

forest_model = RandomForestRegressor(n_estimators=100, random_state=0) # You can adjust parameters as needed
forest_mse, forest_rmse, forest_mae, forest_r2 = perform_cross_validation(forest_model, X, y, n
um_folds)
print("Random Forest:")
print(f"Average MSE: {np.mean(forest_mse) / np.mean(y) * 100:.2f}%")
print(f"Average RMSE: {np.mean(forest_rmse) / np.mean(y) * 100:.2f}%")
print(f"Average MAE: {np.mean(forest_mae) / np.mean(y) * 100:.2f}%")
print(f"Average R-squared: {np.mean(forest_r2) * 100:.2f}%")

<u>OUTPUT</u>:

```
Random Forest:
Average MSE: 10.32%
Average RMSE: 8.09%
Average MAE: 5.99%
Average R-squared: 94
.27%
```

FEATURE ENGINEERING :

All machine learning algorithms use some input data to generate outputs. Input data contains many features which may not be in proper form to be given to the model directly. It needs some kind of processing and here feature engineering helps. Feature engineering fulfills mainly two goals:

- It prepares the input dataset in the form which is required for a specific model or machine learning algorithm.
- Feature engineering helps in improving the performance of machine learning models magically.

GOAL OF FEATURE ENGINEERING:

The goal of feature engineering is simply to make your data better suited to the problem at hand.

Consider "apparent temperature" measures like the heat index and the wind chill. These quantities attempt to measure the *perceived* temperature to humans based on air temperature, humidity, and wind speed, things which we can measure directly. You could think of an apparent temperature as the result of a kind of feature engineering, an attempt to make the observed data more relevant to what we actually care about: how it actually feels outside!

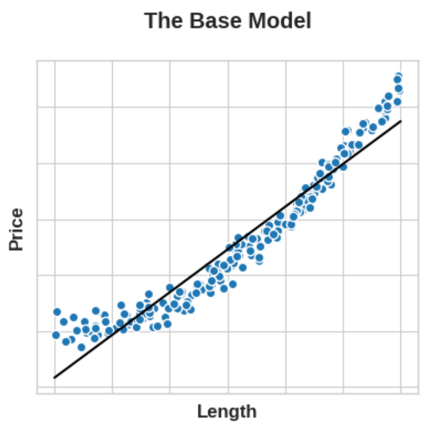
You might perform feature engineering to:

- improve a model's predictive performance
- reduce computational or data needs
- improve interpretability of the results

GUIDING PRINCIPLE OF FEATURE ENGINEERING:

For a feature to be useful, it must have a relationship to the target that your model is able to learn. Linear models, for instance, are only able to learn linear relationships. So, when using a linear model, your goal is to transform the features to make their relationship to the target linear.

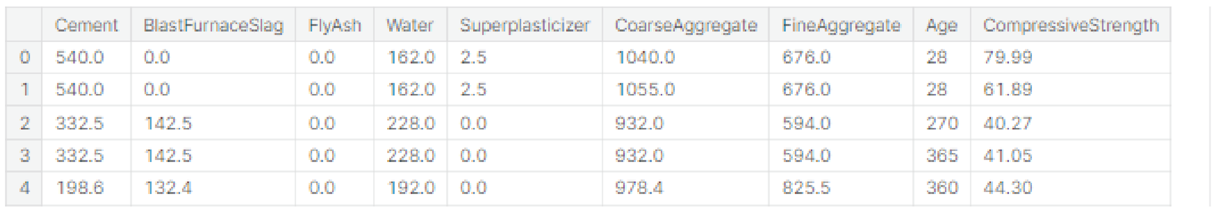
The key idea here is that a transformation you apply to a feature becomes in essence a part of the model itself. Say you were trying to predict the Price of square plots of land from the Length of one side. Fitting a linear model directly to Length gives poor results: the relationship is not linear.



CODE :

```
import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import cross_val_score

df = pd.read_csv("../input/fe-course-data/concrete.csv")

df.head()
```

OUTPUT:



	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	CompressiveStrength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

Conclusion:

Sales prediction is an ongoing process, and it's crucial to adapt to changing circumstances. the more data you collect and the more advanced your modelling techniques, the better your sales predictions are likely to become.Additionally, consider integrating external data sources and market research findings for a more comprehensive analysis.