Feature Selection (Embedded Methods)

✅ **Calculates feature importance during model building itself.**

✅More efficient than wrapper methods (do not require retraining multiple models).

✅More accurate than filter methods (because they consider feature importance in the context of the model).

✅Work well with high-dimensional datasets.

> 💡 **if you have `coef_` or `feature_importance_` in your algorithm, you can use it as embedded technique.**

`coef_`:

- LR
- LR
- Ridge
- Lasso
- **ElasticNet**

`feature_importance_` (Tree based algorithms)

- Decision tree
- Random Forest
- Gradient Boosting

Regularized Models

- Regularized linear models are linear models that include a penalty term in the loss function during training.
- The penalty term discourages the learning of a too complex model, which can help **prevent overfitting.**

Summary of Embedded Methods:

- **Lasso** (L1 regularization): Shrinks coefficients and removes irrelevant features. (MOSTLY USED)
  - it keeps the important features and makes `coef_` =0 for other ones
- **Ridge** (L2 regularization): Reduces the impact of correlated features but doesn't drop any features.
- **ElasticNet**: A combination of Lasso and Ridge.
- **Tree-based methods**: Feature importance from decision trees, random forests, and gradient boosting methods.

Ridge And Lasso Regression

- Prevents overfitting
- **Regularization** introduces a penalty term to the loss function during model training. This penalty discourages large coefficients and helps produce simpler, more generalizable models.

Ridge Regression

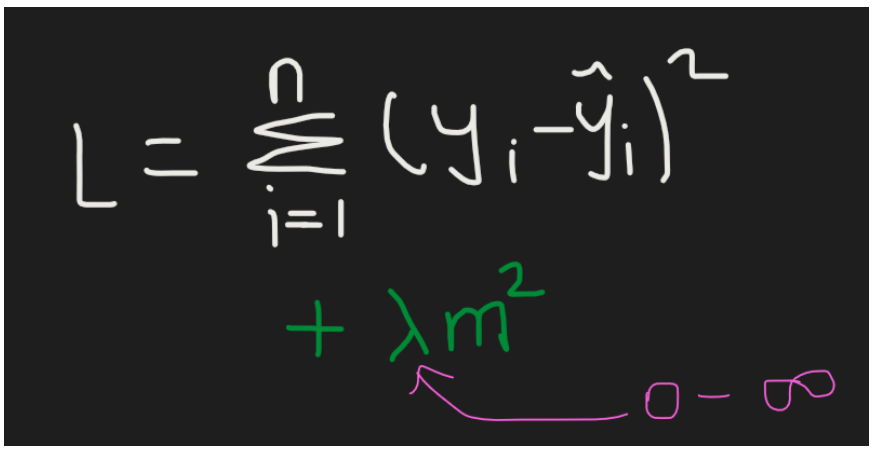
- **Ridge (L2 Regularization)**:
  - Adds a penalty term equal to the **sum of squared coefficients**
  - **No Feature Selection**: Shrinks coefficients but rarely sets them to zero.

Objective Function

$$\text{Cost} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$: Ordinary Least Squares (OLS) loss (sum of squared residuals).
- $\lambda \sum_{j=1}^{p} \beta_j^2$: L2 penalty term (shrinks coefficients toward zero).

- Known as **L2 Regularization because you multiply by square**



- Due to this, the Slope decreases

pass

Feature Selection using LASSO

```
from sklearn import datasets
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/master/diabetes.csv')
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

- We want to find out which features are important in above data set.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:-1],df.iloc[:,-1],test_size=0.2,random_state=1)
```

`df.iloc[:, 0:-1]` / **`df.iloc[:, :-1]`**

- `df.iloc[:, ...]` means "select all rows."
- `0:-1` refers to the column selection.
  - `0` is the index of the first column.
  - `1` is the index of the last column.
  - **The slice `0:-1` means "select all columns from the first column up to, but *excluding*, the last column."**

`df.iloc[:, -1]`

- `df.iloc[:, ...]` means "select all rows."
- `1` refers to the index of the last column.
- **In essence, `df.iloc[:, -1]` selects the last column of the DataFrame.**

```
X_train.shape
```

```
Outcome: (614, 8)
```

Scale the data:

```
from sklearn.preprocessing import StandardScaler

cols = X_train.columns

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = pd.DataFrame(X_train_scaled,columns=cols)
X_test_scaled = pd.DataFrame(X_test_scaled,columns=cols)
```

Apply Lasso:

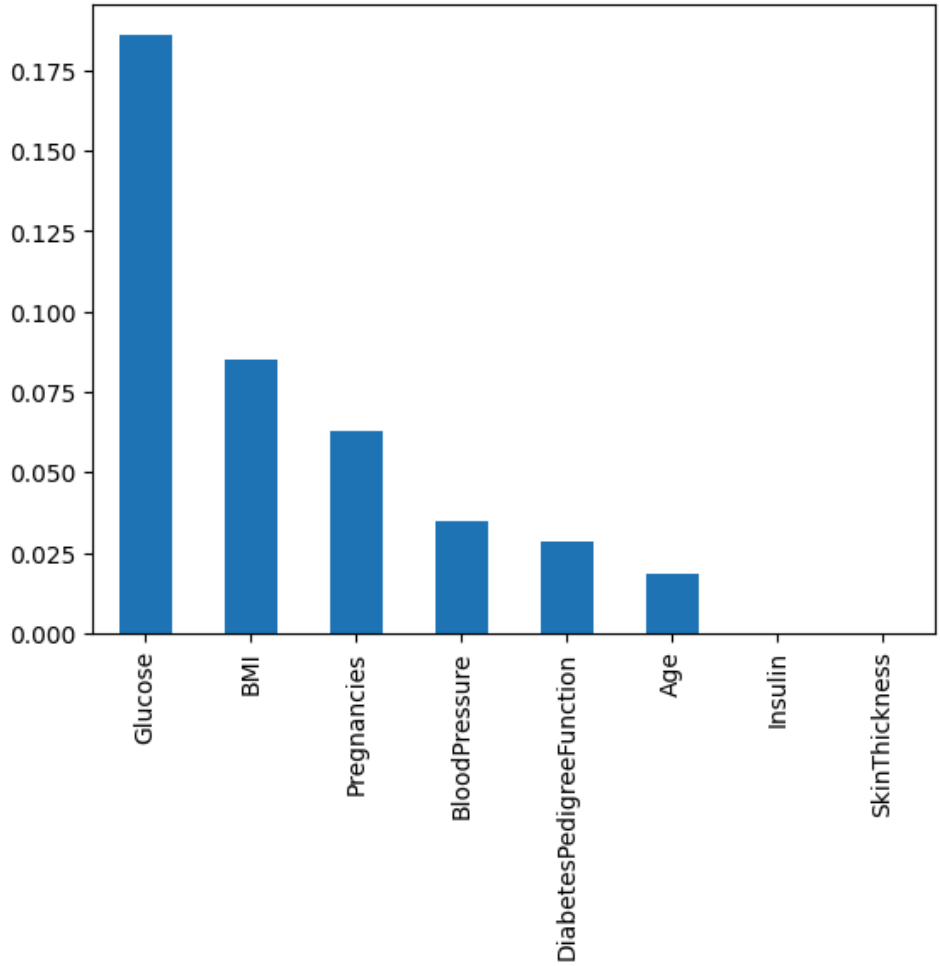
```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.01)

lasso.fit(X_train_scaled, y_train)
```

- Alpha → 0.01 to 0.1

```
lasso.coef_

Output:
array([ 0.        ,  0.1267733 , -0.        ,  0.        ,  0.        ,
        0.00480181,  0.        ,  0.        ])
```

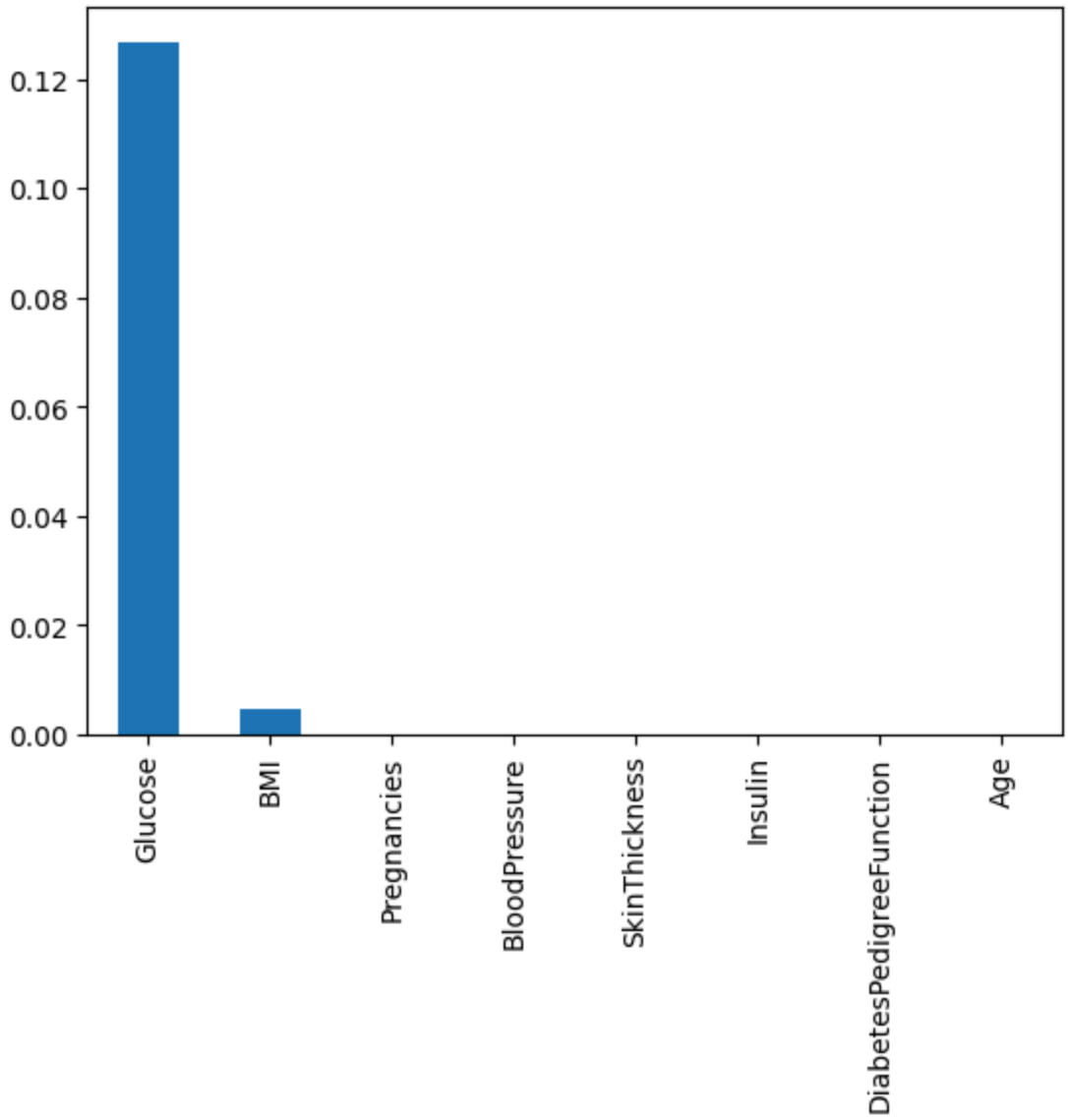
```
import matplotlib.pyplot as plt
import numpy as np

x = pd.Series(np.abs(lasso.coef_),index=cols)

x.sort_values(ascending=False).plot(kind='bar')
```

- glucose is the most important attribute
- 2nd is BMI



- If you increase `alpha` to 0.1

- It makes unimportant features zero



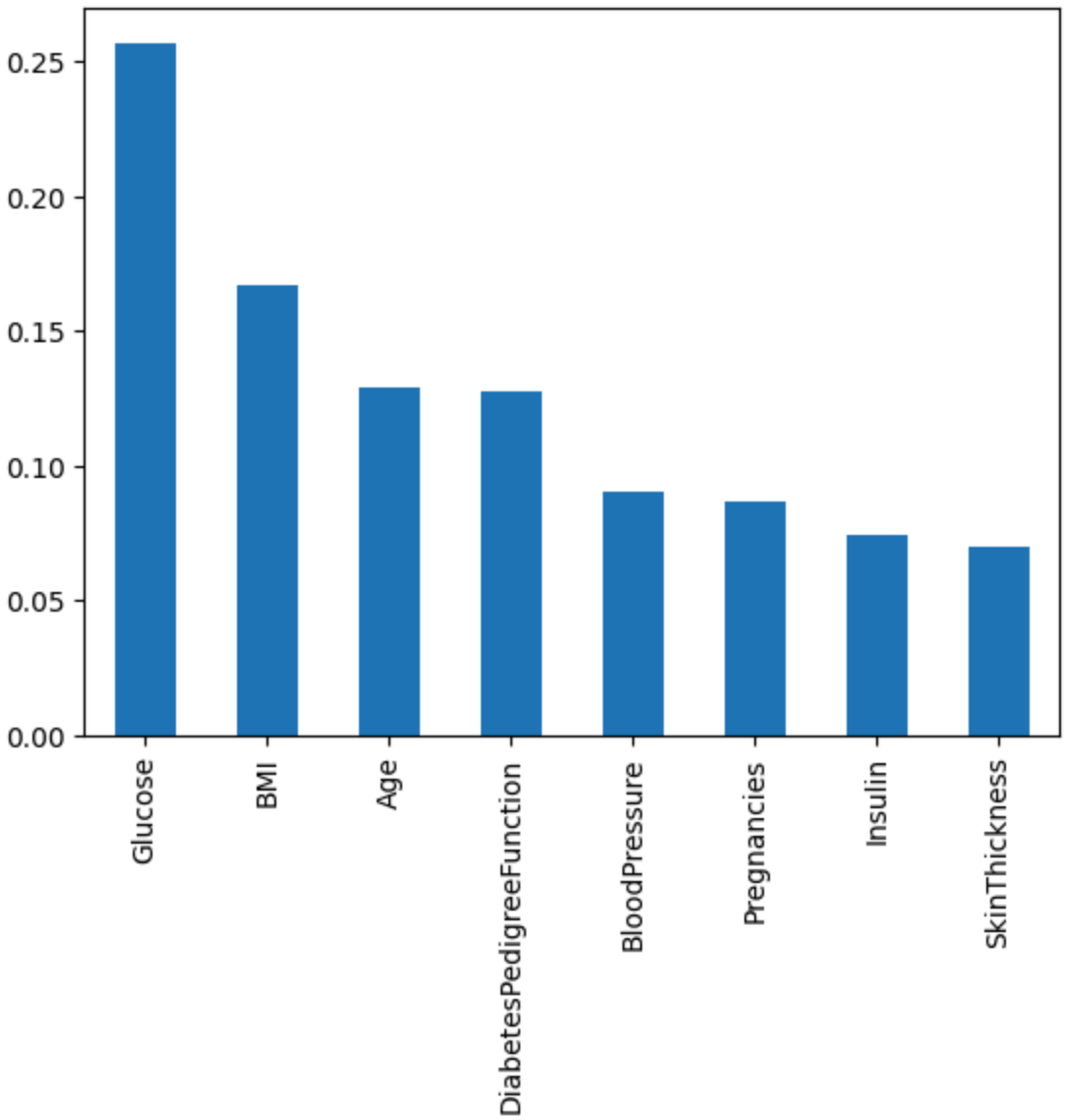
Tree Based Models

```
from pandas.core.common import random_state
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
dt = DecisionTreeClassifier()
rf = RandomForestClassifier()

#dt.fit(X_train,y_train)
rf.fit(X_train,y_train)
```

```
rf.feature_importances_

Output:
array([0.08635461, 0.25644277, 0.09021031, 0.06965868, 0.07414166,
       0.16670238, 0.12745014, 0.12903945])
```

```
x = pd.Series(np.abs(rf.feature_importances_),index=cols)

x.sort_values(ascending=False).plot(kind='bar')
```



Transformer- SelectFromModel

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
```

```
from sklearn.feature_selection import SelectFromModel

sfm = SelectFromModel(model, threshold='0.1')
```

`threshold='0.1` → columns with feature importance less than 0.1 will be eliminated

You can use → `'mean'`, `'median'`

`'mean'` = You calculate feature importance of all the columns and keep the ones above mean.

```
sfm.get_support(indices=True)

Output:
array([1, 2, 5, 6, 7], dtype=int64)
```

```
sfm.feature_names_in_[sfm.get_support(indices=True)]

Output:
array(['Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFunction',
       'Age'], dtype=object)
```

Transform:

```
X_train_trans = sfm.transform(X_train)

X_train_trans = pd.DataFrame(X_train_trans,columns=sfm.feature_names_in_
[sfm.get_support(indices=True)])

X_train_trans
```

	Glucose	BloodPressure	BMI	DiabetesPedigreeFunction	Age
0	145.0	80.0	37.9	0.637	40.0
1	129.0	62.0	41.2	0.441	38.0
2	102.0	74.0	37.2	0.204	45.0
3	120.0	78.0	25.0	0.409	64.0
4	120.0	76.0	39.7	0.215	29.0

Recursive Feature Elimination

- You eliminate the feature with least importance
- You repeat this eliminating features one by one

```
df = pd.read_csv('https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/0e7a9b0a5d22642a06d3d5b9bcbad9890c8ee534/iris.csv')
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]
```

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load iris dataset
url = "https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/0e7a9b0a5d22642a06d3d5b9bcbad9890c8ee534/iris.csv"
df = pd.read_csv(url)

# Separate features and target variable
X = df.drop("species", axis=1)
y = df["species"]

# Initialize RandomForestClassifier
model = RandomForestClassifier()

# Initialize RFE
rfe = RFE(estimator=model, n_features_to_select=1)

# Fit RFE
rfe.fit(X, y)

# Print the ranking
ranking = rfe.ranking_
print("Feature ranking:")

for i, feature in enumerate(X.columns):
    print(f"{feature}: {ranking[i]}")
```

Output:

Feature ranking:
sepal_length: 3
sepal_width: 4
petal_length: 1
petal_width: 2

Embedded Methods: Pros

- **Accurate:** Captures feature relationships, improving prediction.
- **Efficient:** Integrates selection into training, saving computation.
- **Regularized:** Reduces overfitting by penalizing less important features.

Embedded Methods: Cons

- **Model-Dependent:** Features selected are specific to the model.
- **Complex:** Interpretation can be challenging, especially with regularization.
- **Hyperparameter-Sensitive:** Performance relies on proper tuning.
- **Unstable:** Feature selection can vary with data changes.