



# Maximizing revenue using recipe recommendation

**Team B members:**

**Radek Holik**

**Yujie Wu**

**Haider Ali**



# Problem statement

Using Machine learning to accurately predict which recipe has the highest protein content given a particular recipe features to maximize the profit.

# Approach



## Data pre-processing

Applied SMOTE to oversample the undersampled class in response variable

Min-Max normalization values before modeling



## Model building

Linear regression using with custom thresholding

Logistic Regression

K Nearest Neighbor

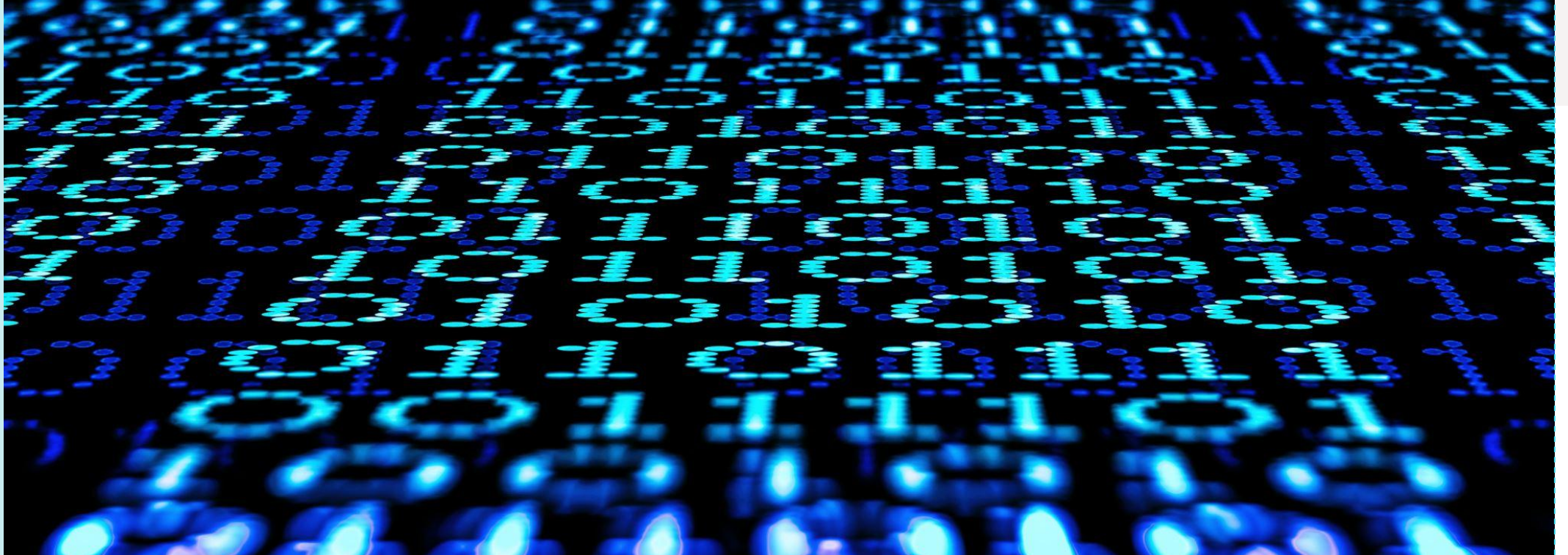
Support Vector machine

Decision tree

Random Forest

Combine Model

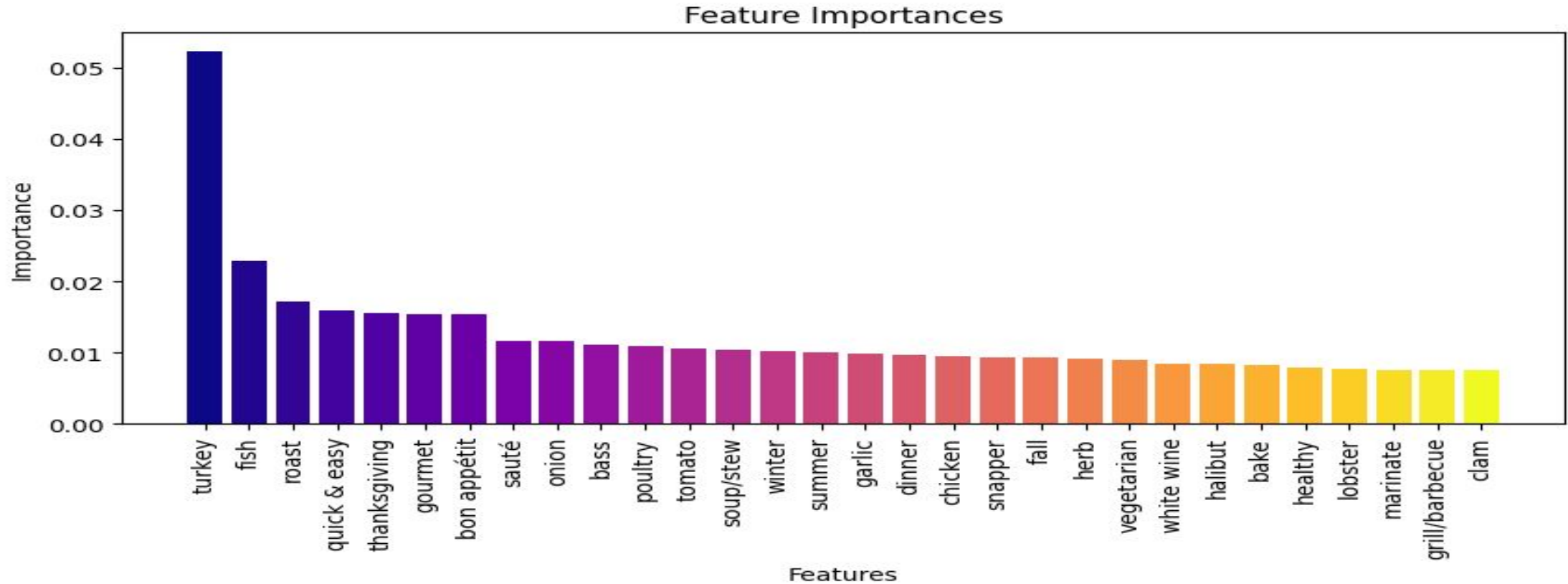




# Data description

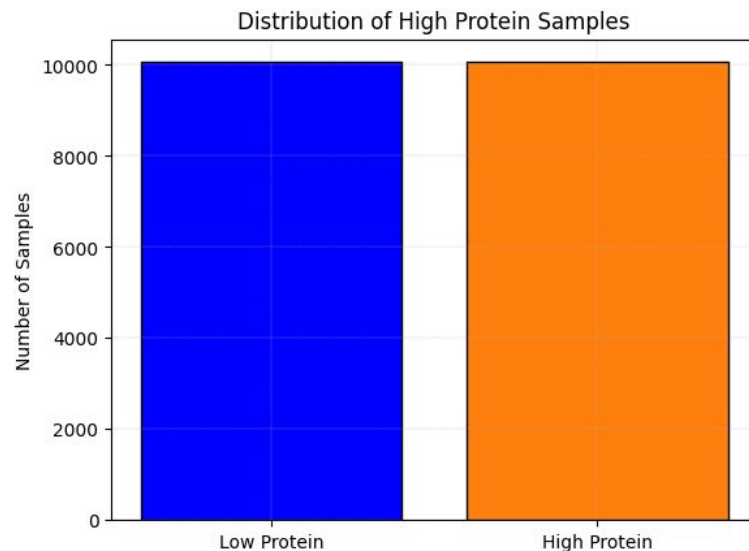
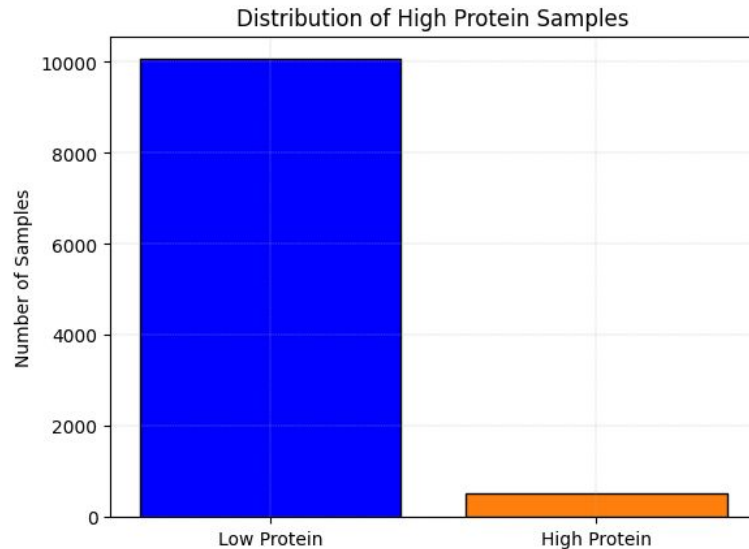
- ◆ Dataset contains 10593 rows and 672 columns
- ◆ We created a binary feature representing high or low protein.

# Feature Importance



- Here **turkey** has the highest protein content
- High level food categories include **fish**, **onion**, **bass** (fish), **poultry**
- Seasonal/occasional /time-related categories include **Thanksgiving**, **dinner**, **winter**, **fall**, **summer**
- Making types such as **bake**, **roast**, **marinate**, **quick/easy**, **grill/barbeque**, **saute**

# High protein class distribution



- Here, we found that the dependent feature is highly imbalanced, and to overcome this problem, we use **Synthetic Minority Oversampling Technique (SMOTE)** to oversample the high protein variables.

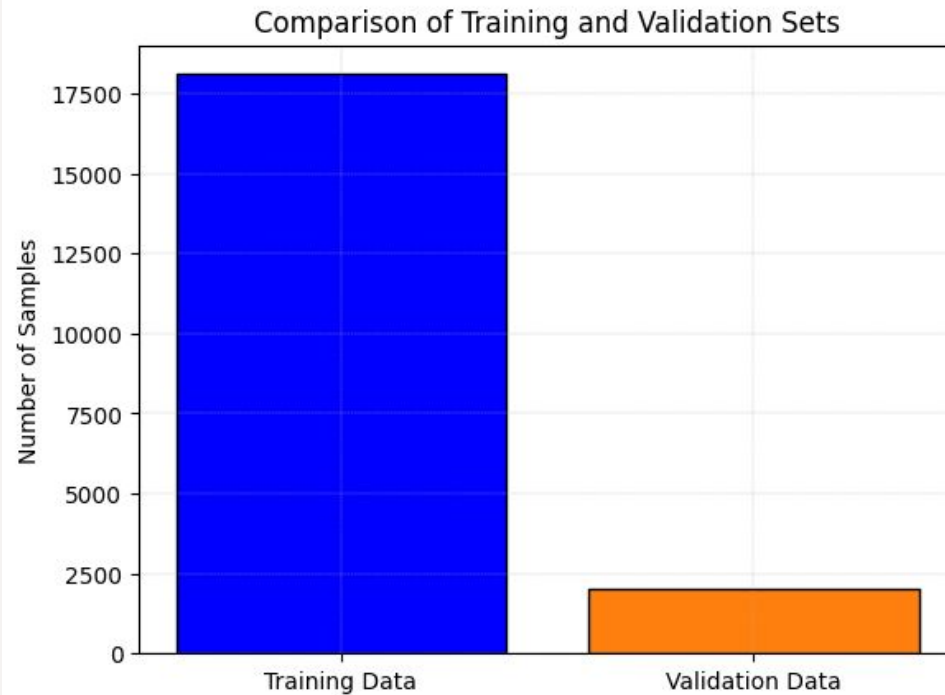
Before SMOTE:

- Number of low protein recipes : 10061
- Number of high protein recipes : 532

After SMOTE:

- Number of low protein recipes : 10061
- Number of high protein recipes : 10061

# Data Preprocessing



## Training and Validation data:

- ◆ 10 percent (2013) for validation and 90 percent (18109) for training
- ◆ Normalized training and validation data using Min-Max normalization

```
train_counts = len(y_train)
val_counts = len(y_val)
print(f"Number of train samples : {train_counts}")
print(f"Number of validation samples : {val_counts}")
```

```
Number of train samples : 18109
Number of validation samples : 2013
```

# Linear Regression

Linear regression is a statistical modeling technique used to establish a relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data.

Parameter combination for GridSearch CV

```
param_grid = {  
    'threshold': [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0], # Different threshold values  
    'fit_intercept': [True, False],  
    'positive': [True, False],  
}
```

## Best parameters

▼	Linear_Regression
Linear_Regression(positive=True, threshold=0.5)	



# Logistic Regression

Logistic regression is a statistical method used for binary classification that models the relationship between a dependent binary variable and one or more independent variables using a logistic function.

Parameter combination for GridSearch CV

```
param_grid = {  
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  
    'penalty': ['l1', 'l2'], # Different regularizers  
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'] # Different optimizers  
}
```

## Best parameters

```
▼ LogisticRegression  
LogisticRegression(C=100, max_iter=1000, penalty='l1', solver='liblinear')
```

# Decision Tree Classifier

Decision Tree Classifier is a non-parametric supervised learning algorithm used for classification and regression tasks, which constructs a decision tree to partition the input space into smaller regions and predicts the class label for a given input based on the majority class of training samples in the corresponding region.

Parameter combination for GridSearch CV

```
param_grid = {  
    'criterion': ['gini', 'entropy'],  
    'max_depth': [None, 5, 10, 20],  
    'min_samples_split': [2, 5, 10],  
    'min_samples_leaf': [1, 2, 5],  
}
```

## Best parameters

```
▼ DecisionTreeClassifier  
DecisionTreeClassifier(criterion='entropy', min_samples_split=5,  
                      random_state=42)
```

# K Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric machine learning algorithm used for classification and regression tasks, which predicts the class of a new data point based on the majority class of its k-nearest neighbors in the feature space.

Parameter combination for GridSearch CV

```
param_grid = {  
    'n_neighbors': range(1, 31, 2), # K values from 1 to 30  
    'weights': ['uniform', 'distance'], # Different weight options  
    'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski'] # Different distance metrics  
}
```

## Best parameters

```
▼ KNeighborsClassifier  
KNeighborsClassifier(metric='manhattan', n_neighbors=1)
```

# Random Forest Classifier

Random forest is an ensemble learning method that constructs multiple decision trees at training time and outputs the class that is the mode of the classes (classification) of each tree.

Parameter combination for GridSearch CV

```
param_grid = {  
    'n_estimators': [100, 200, 500],  
    'max_depth': [None, 10, 20, 30],  
    'min_samples_split': [2, 5, 10],  
    'min_samples_leaf': [1, 2, 4],  
    'bootstrap': [True, False],  
    'criterion': ['gini', 'entropy']  
}
```

## Best parameters

```
▼ RandomForestClassifier  
RandomForestClassifier(criterion='entropy', n_estimators=200)
```



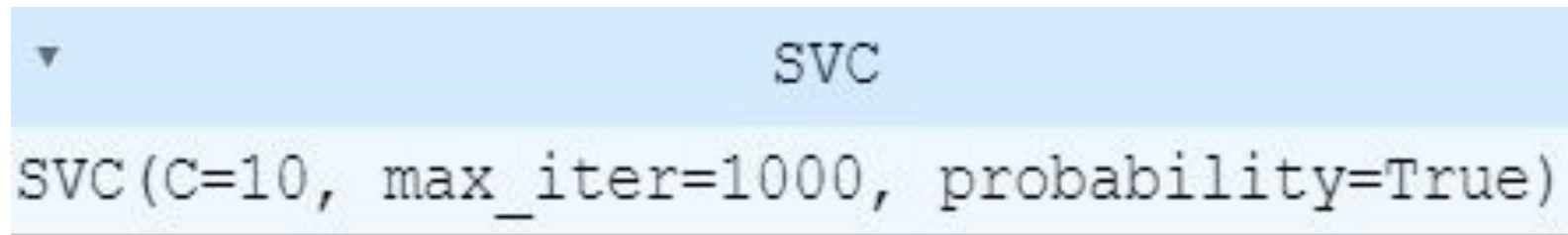
# Support Vector Machine

Support Vector Machine (SVM) is a powerful machine learning algorithm that finds the best boundary to separate different classes of data with maximum margin.

Parameter combination for GridSearch CV

```
param_grid = {  
    'C': [0.1, 1, 10, 100],  
    'kernel': ['linear', 'rbf'], # Different support kernels  
    'gamma': ['scale', 'auto']  
}
```

**Best parameters**



The screenshot shows a Jupyter Notebook cell with a light blue header bar containing a dropdown arrow and the text 'SVC'. Below the header, the code `SVC(C=10, max_iter=1000, probability=True)` is displayed in a monospaced font and is underlined.

```
SVC(C=10, max_iter=1000, probability=True)
```

# Combined model

We've combined our top 3 best models to increase the accuracy.

```
class Combine_Model():  
    def __init__(self, model_1, model_2, model_3):  
        self.model_1 = model_1  
        self.model_2 = model_2  
        self.model_3 = model_3  
  
    def predict(self, X):  
        output_1 = self.model_1.predict(X)  
        output_2 = self.model_2.predict(X)  
        output_3 = self.model_3.predict(X)  
  
        # Average of our three best models  
        pred_labels = np.around((output_1 + output_2 + output_3) / 3)  
  
        return pred_labels  
  
    def predict_proba(self, X):  
        y_pred = self.predict(X)  
        y_pred_proba = np.vstack((1 - y_pred, y_pred)).T  
  
        return y_pred_proba
```

Python

```
our_combine_model = Combine_Model(best_knn, best_svm, best_random_forest)
```

Python

- KNN
- SVM
- Random Forest

# Machine learning

- ◆ Trained 6 ML models from them SVM performed the best.
- ◆ Applied GridsearchCV on all the models to get the parameters for best

Model	Best parameters	Accuracy	Cross-Entropy Loss
(KNN + SVM + RF)	Below best models	0.9994	0.020416
SVM	C:10, gamma:scale, Kernel:rbf	0.9982	0.006187
Random Forest	Bootstrap:True, criterion: entropy, min_est:200 min_samples_leaf:1, min_samples_split:2	0.9966	0.033488
KNN	metric: Manhattan, n_neighbors:1, weights: uniform	0.996	0.142909
Decision Tree	criterion:Entropy, min_samples_leaf:1, min_samples_split:5	0.9889	0.210277
Logistic Regression	C:100, penalty:l1, solver:liblinear	0.9209	0.207550
Linear regression with 0.5 threshold	Threshold:0.5	0.8506	0.870005

# Insights

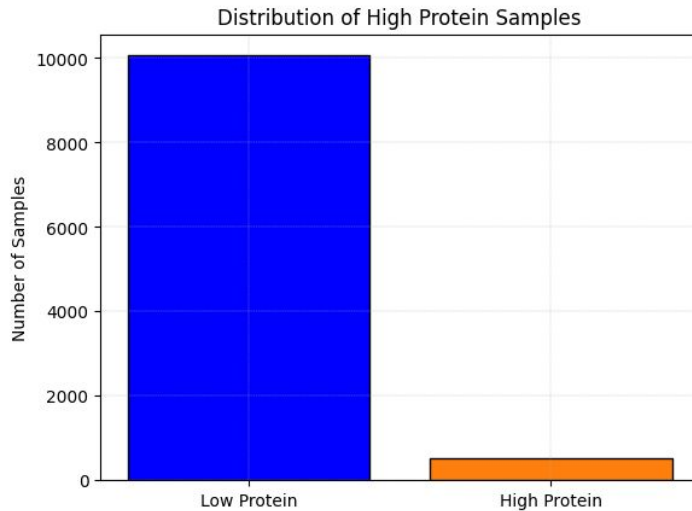
- ◆ From the 672 features, only 35 features are important as they were highly correlated and used by the model as top-level decision nodes.
- ◆ Linear classifiers such as Linear regression and Logistic regression aren't 95 percent accurate.
- ◆ Non-Linear classifiers such as SVC, Random forest, KNN, and Decision Tree all perform better than Linear classifiers
- ◆ Therefore, we can say that the dataset is more non-linearly separable than linear.



# Revenue

◆ Linear Regression with Threshold	: Revenue \$ 90,266.00	SER \$ 29.53
◆ Logistic Regression	: Revenue \$ 90,855.50	SER \$ 29.26
◆ K-Nearest Neighbors	: Revenue \$ 91,823.25	SER \$ 31.36
◆ Support Vector Machine	: Revenue \$ 91,823.25	SER \$ 29.64
◆ Decision Tree	: Revenue \$ 91,816.75	SER \$ 29.28
◆ Random Forest	: Revenue \$ 92,004.50	SER \$ 29.61
◆ Combine Model (Best model)	: Revenue \$ 92,548.25	SER \$ 30.46

# How much better is our model than random guessing?



- ◆ Using the same data set as we received, the ratio between "low protein" and "high protein" labels is 95:5.
- ◆ Then the random guess is 95%.
- ◆ Our combined model has an accuracy of 99.94%.
- ◆ Our best model is better than the random guess.

```
# Calculate the distribution of the "high_protein" column
high_protein_counts = df['high_protein'].value_counts()
print(f"Number of low protein recipes : {high_protein_counts[0]}")
print(f"Number of high protein recipes : {high_protein_counts[1]}")
```

✓ 0.0s

Python

```
Number of low protein recipes : 10061
Number of high protein recipes : 532
```

```
# Random Guess:
print(f"Random Guess : {high_protein_counts[0]/(high_protein_counts[0] + high_protein_counts[1])}")
```

✓ 0.0s

Python

```
Random Guess : 0.949778155385632
```

Thanks for your time

Any Questions?