

HONEY PRICE PREDICTION BASED ON PURITY

1.Introduction

1.1Project Overview

Honey, a natural sweetener, is prized for its unique flavor, aroma, and nutritional benefits. The quality and purity of honey significantly impact its market value, making it essential to develop a reliable system for predicting honey prices based on its purity. This overview explores the key factors influencing honey prices, the importance of honey purity, and the various methods for predicting honey prices, highlighting the significance of accurate price prediction in the honey industry.

Honey purity is typically classified into different grades, including:

1. **High-quality honey:** With a high pollen count and low moisture content, this grade commands a premium price.
2. **Standard honey:** Meeting basic quality standards, this grade is widely available and moderately priced.
3. **Low-quality honey:** With high moisture content and low pollen count, this grade is often discounted.

1.2 Project Objective

The primary objective of this project is to develop a comprehensive machine learning model that accurately predicts honey prices based on purity metrics. This model aims to assist beekeepers, honey producers, and market analysts in optimizing resource allocation and maximizing profitability by understanding and anticipating market dynamics influenced by honey purity.

2. Project Initialization and Planning Phase

The "Project Initialization and Planning Phase" marks the project's outset, defining goals, scope, and stakeholders. This crucial phase establishes project parameters, identifies key team members, allocates resources, and outlines a realistic timeline. It also involves risk assessment and mitigation planning. Successful initiation sets the foundation for a well-organized and efficiently executed machine learning project, ensuring clarity, alignment, and proactive measures for potential challenges.

Activity 1: Define Problem Statement

A honey producer aims to accurately predict the market price of honey based on its purity. The challenge lies in understanding and effectively utilizing purity metrics such as moisture content, pollen count, and sugar content, which significantly impact honey prices. Despite the producer's efforts to maintain high purity standards, uncertainty persists regarding the accurate estimation of these factors' influence on market prices.

Ref. template:

Honey price prediction Problem Statement Report: [Click Here](#)

Activity 2: Project Proposal (Proposed Solution)

Our innovative proposed solution harnesses advanced machine learning algorithms and robust data analytics to predict honey prices based on purity, aiming to foster fair market practices, enhance consumer safety, and stimulate industry growth. By leveraging predictive models, we empower beekeepers, honey producers, and consumers with reliable insights into pricing dynamics influenced by honey industry.

Ref. template:

Honey price prediction Project Proposal Report: [Click Here](#)

Activity 3: Initial Project Planning

Initial Project Planning involves outlining key objectives, defining scope, and identifying the price prediction. It encompasses setting timelines, allocating resources, and determining the overall project strategy. During this phase, the team establishes a clear understanding of the dataset, formulates goals for analysis, and plans the workflow for data processing. Effective initial planning lays the foundation for a systematic and well-executed project, ensuring successful outcomes.

Ref. template:

Honey price prediction Initial Project Planning Report: [Click Here](#)

3. Data Collection and Preprocessing Phase

The Data Collection and Preprocessing Phase involves executing a plan to gather relevant Honey price prediction data from Kaggle, ensuring data quality through verification and addressing missing values. Preprocessing tasks include cleaning, encoding, and organizing the dataset for subsequent exploratory analysis and machine learning model development.

Activity 1: Data Collection Plan, Raw Data Sources Identified

The dataset for "Honey price Prediction" is sourced from Kaggle, a reputable platform known for its diverse collection of datasets in agricultural sciences and predictive analytics. This dataset is meticulously curated to encompass a wide array of variables essential for accurate blueberry yield prediction. These variables include climatic factors, Pollinating factors, and historical yield data. This comprehensive dataset provides a robust foundation for developing predictive models.

Ref. template:

Honey price prediction Raw Data Sources Report: [Click Here](#)

Activity 2: Data Quality Report

The dataset for "Honey price Prediction" is sourced from Kaggle. It includes climatic factors, Pollinating factors and historical yield data. Data quality is ensured through verification, addressing missing values and handling Outliers, establishing a reliable foundation for predictive modeling.

Ref. template:

Honey price prediction Data Quality Report: [Click Here](#)

Activity 3: Data Exploration and Preprocessing

Data Exploration involves analyzing the Honey price prediction dataset to understand patterns, distributions, and outliers. Preprocessing includes handling missing values, scaling, and encoding categorical variables. These crucial steps enhance data quality, ensuring the reliability and effectiveness of subsequent analyses in the Honey price prediction project.

Ref. template:

Honey price prediction Data Exploration and Preprocessing Report: [Click Here](#)

4. Model Development Phase

The Model Development Phase entails crafting a predictive model for loan approval. It encompasses strategic feature selection, evaluating and selecting models (Linear Regression, Random Forest, Decision Tree), initiating training with code, and rigorously validating and assessing model performance for informed decision-making in the lending process.

Activity 1: Feature Selection Report

The Feature Selection Report outlines the rationale behind choosing specific features for the honey price prediction model. It evaluates relevance, importance, and impact on predictive accuracy, ensuring the inclusion of key factors influencing the model's ability to predict honey prices accurately.

Ref. template: Honey price prediction Feature Selection Report: [Click Here](#)

Activity 2: Model Selection Report

The Model Selection Report details the rationale behind choosing Linear Regression, Random Forest, Decision Tree, models for Honey price prediction. It considers each model's strengths in handling complex relationships, interpretability, adaptability, and overall predictive performance, ensuring an informed choice aligned with project objectives.

Ref. template:

Honey price prediction Model Selection Report: [Click Here](#)

Activity 3: Initial Model Training Code, Model Validation and Evaluation Report

The Initial Model Training Code employs selected algorithms on the Honey price prediction dataset, setting the foundation for predictive modeling. The subsequent Model Validation and Evaluation Report rigorously assesses model performance, employing metrics like MAE, MSE, RSquared and accuracy to ensure reliability and effectiveness in predicting loan outcomes.

Ref. template:

Honey price prediction Model Development Phase Template: [Click Here](#)

5. Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Activity 1: Hyperparameter Tuning Documentation

The Random forest model was selected for its superior performance, exhibiting high accuracy during hyperparameter tuning. Its ability to handle complex relationships, minimize overfitting, and optimize predictive accuracy aligns with project objectives, justifying its selection as the final model.

Activity 2: Performance Metrics Comparison Report

The Performance Metrics Comparison Report contrasts the baseline and optimized metrics for various models, specifically highlighting the enhanced performance of the Random forest model. This assessment provides a clear understanding of the refined predictive capabilities achieved through hyperparameter tuning.

Activity 3: Final Model Selection Justification

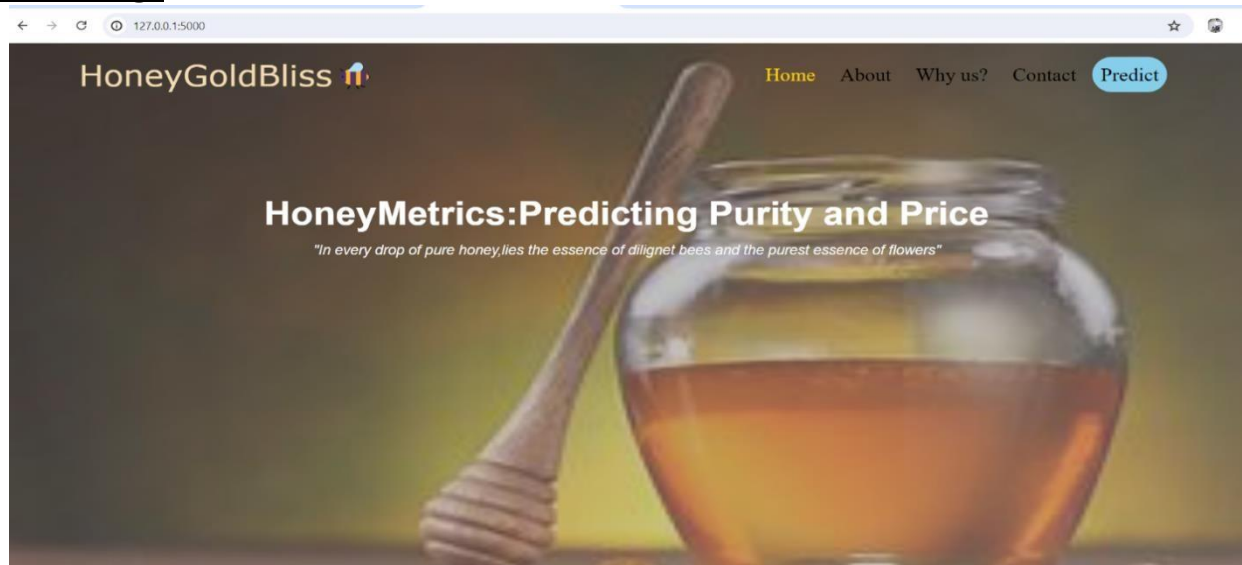
The Final Model Selection Justification articulates the rationale for choosing Random forest as the ultimate model. Its exceptional accuracy, ability to handle complexity, and successful hyperparameter tuning align with project objectives, ensuring optimal Honey price predictions.

Ref. template:

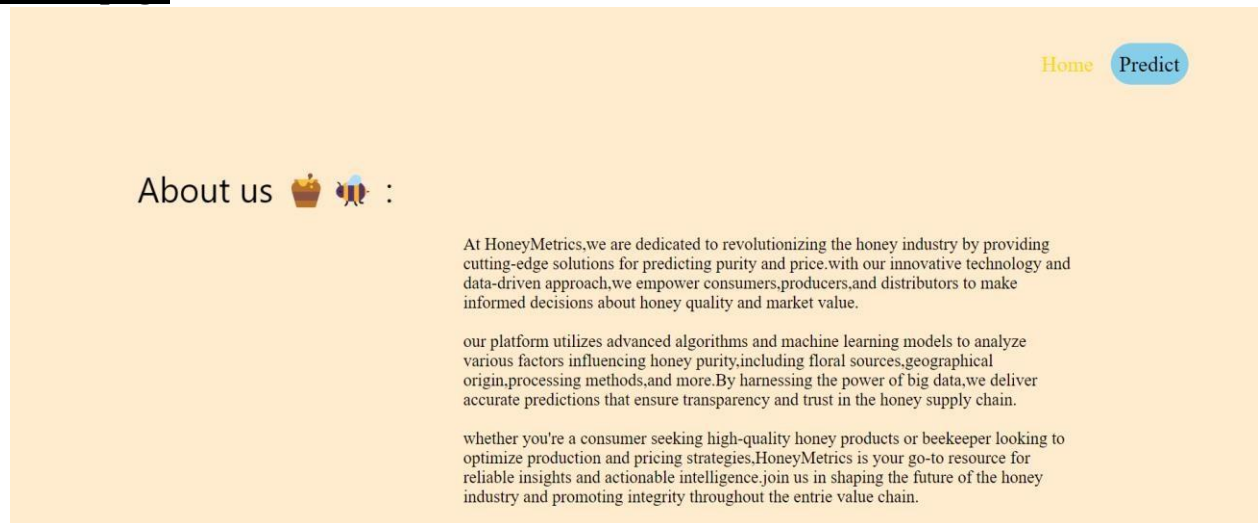
Honey price prediction Optimization and Tuning Phase Report: [Click Here](#)

6.Results

Index Page



About page



Details page

127.0.0.1:5000/#predict

HoneyGoldBliss

Explore the purity Honey Price Prediction Form

Please Fill the Below Details

CS:

Density:

WC:

pH:

EC:

F:

G:

pollen analysis:

viscosity:

purity:

HoneyGoldBliss

Explore the purity Honey Price Prediction Form

Please Fill the Below Details

CS:

Density:

WC:

pH:

EC:

F:

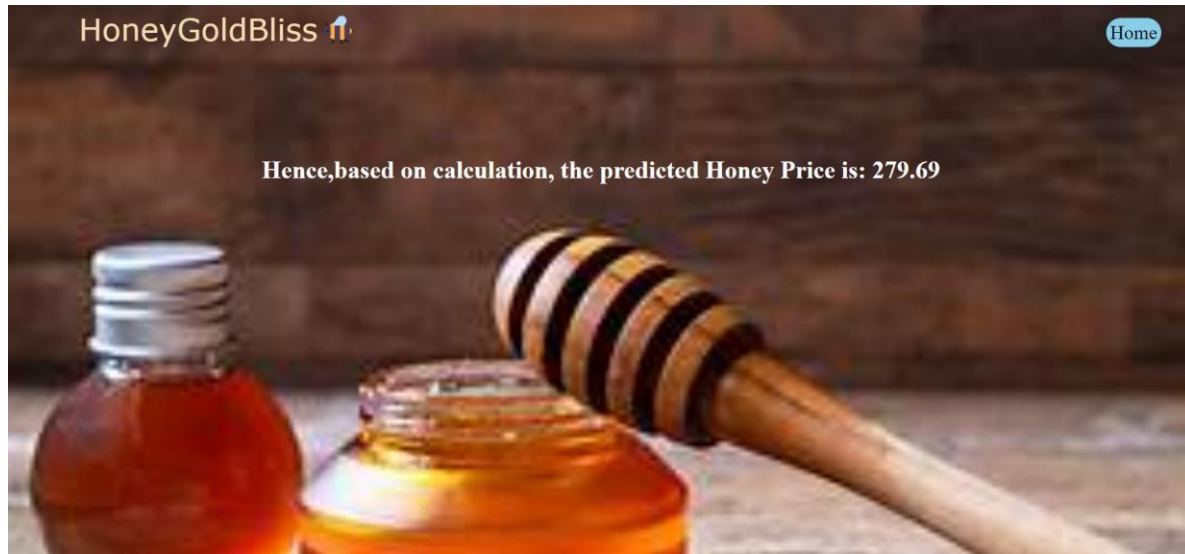
G:

pollen analysis:

viscosity:

purity:

Predict page



7. Advantages & Disadvantages

ADVANTAGES:

- 1. Market Demand:** Consumers often prefer pure honey due to its perceived health benefits and superior taste. As awareness of adulteration grows, the demand for pure honey may increase, influencing prices.
- 2. Supply Chain Transparency:** Brands and suppliers that can guarantee the purity of their honey may command higher prices due to consumer trust and perceived quality.
- 3. Quality Standards and Certifications:** Honey certified by reputable organizations for purity (e.g., USDA Organic, European Union Organic) typically fetches higher prices in the Market
- 4. Consumer Trends:** Trends in health and wellness, as well as environmental consciousness, can influence consumer preferences for pure honey, thereby impacting prices.

DISADVANTAGES:

- 1. Complexity of Purity Assessment:** Determining the purity of honey can be challenging and subjective. Various methods exist to assess purity, such as laboratory testing for contaminants or adulterants (like added sugars), but these methods can be costly and time-consuming.
- 2. Variability in Purity Standards:** Purity standards may vary across regions and countries, making it difficult to establish consistent benchmarks for

comparison. This variability can lead to discrepancies in how purity is defined and assessed in different markets.

3. **Market Manipulation:** Honey prices can be influenced by market speculation, supply chain dynamics, and geopolitical factors. Predicting these influences accurately, especially in volatile markets, can be challenging.

19

19

4. **Economic and Environmental Factors:** Factors such as weather conditions, bee health, and agricultural practices can affect honey production and purity. These external factors add uncertainty to price predictions based on purity alone.
5. **Technological Limitations:** Advanced technology is often required to detect sophisticated adulteration methods, which might not be accessible to all producers or regulatory bodies. Small-scale or local producers may lack the resources to employ high-end testing techniques.
6. **Environmental Impact:** Increased demand for pure honey can lead to intensive beekeeping practices that may not be sustainable. This can have negative impacts on bee populations and biodiversity.
7. **Cultural Differences:** Different cultures may have varying perceptions and traditional uses of honey, complicating the establishment of universal purity standards. What is considered "pure" in one culture might not be the same in another.

8.Conclusion

In this project, we aimed to predict the price of honey based on its purity. To achieve this, we utilized Python to analyze the data and employed three machine learning models: Random Forest, Decision Tree, and Linear Regression.

Methodology and Data

We started by collecting data on honey prices and their corresponding purity levels, which was stored in a CSV file. The dataset was then preprocessed to handle any missing values and to ensure it was suitable for analysis. We used two HTML files for data visualization and interaction, providing an intuitive interface for exploring the data and model predictions.

Model Implementation

Three different machine learning models were implemented to predict honey prices:

1. **Random Forest:** This ensemble learning method was used for its robustness and ability to handle complex data relationships.
2. **Decision Tree:** This model was chosen for its simplicity and interpretability, making it easy to understand how purity levels impact honey prices.
3. **Linear Regression:** As a fundamental regression technique, this model provided a baseline for understanding the linear relationship between purity and price.

■ Our analysis revealed that purity is a significant factor influencing honey prices. Among the three models, the Random Forest model demonstrated the highest accuracy, effectively capturing the non-linear relationships in the data. The Decision Tree model also performed well, providing clear insights into how different purity levels affect pricing. The Linear Regression model, while less accurate, confirmed a positive correlation between purity and price.

9.Future Scope

Future Scope of the Honey price Prediction and Management System:

1. **Expansion of Data Sources:** Expand data collection methods to include advanced analytical techniques and blockchain technology to enhance transparency in the honey supply chain.
2. **Integration of Machine Learning:** Further integrate machine learning algorithms to improve the accuracy of purity assessment and price prediction models, adapting to evolving market dynamics and consumer preferences.
3. **Global Market Analysis:** Extend predictive models to encompass global honey markets, considering regional variations in production, consumption patterns, and regulatory environments affecting purity standards.
4. **Forecasting Tools for Industry Planning:** Develop forecasting tools tailored for honey producers and distributors to optimize production, inventory management, and pricing strategies based on anticipated shifts in market demand for pure honey.

10.Appendix

10.1 Source Code

Index Page:

```
<!DOCTYPE html>
<html>
  <head>
    <title>HoneyGoldBliss</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='assets/css/index.css') }}">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
  </head>
  <body>
    <div id="lg">
      <div id="ns">
        <header id="name">HoneyGoldBliss&#128029;</header>
        <nav id="nav">
          <ul>
            <li><a href="#" class="active">Home</a></li>
            <li><a href="#about">About</a></li>
            <li><a href="#">Why us?</a></li>
```

```

        <li><a href="#">Contact</a></li>
        <li><a href="#predict" id="pr">Predict</a></li>
    </ul>
</nav>
</div>
<div id="tq">
    <h1 id="title">HoneyMetrics:Predicting Purity and Price</h1>
    <p id="quo">"In every drop of pure honey, lies the essence of diligent bees and the purest
essence of flowers"</p>
</div>
</div>
<!--about-->
<div id="about">
    <nav id="nava">
        <ul>
            <li><a href="#lg" class="active">Home</a></li>
            <li><a href="#predict" id="pr">Predict</a></li>
        </ul>
    </nav>
    <div id="ac">
        <aside id="au">About us &#127855;&#128029; :</aside>
        <main id="cont">
            <p>At HoneyMetrics, we are dedicated to revolutionizing the honey industry by
providing cutting-edge solutions for predicting purity and price. With our innovative technology and
data-driven approach, we empower consumers, producers, and distributors to make informed
decisions about honey quality and market value.
            <br><br>our platform utilizes advanced algorithms and machine learning models to
analyze various factors influencing honey purity, including floral sources, geographical origin, processing
methods, and more. By harnessing the power of big data, we deliver accurate predictions that ensure
transparency and trust in the honey supply chain.
            <br><br>whether you're a consumer seeking high-quality honey products or
beekeeper looking to optimize production and pricing strategies, HoneyMetrics is your go-to
resource for reliable insights and actionable intelligence. Join us in shaping the future of the honey
industry and promoting integrity throughout the entire value chain.
        </p>
    </main>
</div>
</div>
<!--predict-->
<div id="predict">
    <header id="namep">HoneyGoldBliss&#128029;</header>
    <p id="qu">Explore the purity Honey Price Prediction Form</p>
    <div id="formbox">
        <h1><b>Please Fill the Below Details</b></h1>
        <form action="/inner_page" method="POST">
            <label for="cs">CS:</label>
            <input type="number" id="cs" name="cs" step="0.01" required>
            <br>

```

```

<label for="D">Density:</label>
<input type="number" id="D" name="D" step="0.01" required>
<br>
<label for="W">WC:</label>
<input type="number" id="w" name="w" step="0.01" required>
<br>
<label for="P">pH:</label>
<input type="number" id="P" name="p" step="0.01" required>
<br>
<label for="E">EC:</label>
<input type="number" id="E" name="E" step="0.01" required>
<br>
<label for="f">F:</label>
<input type="number" id="f" name="f" step="0.01" required>
<br>
<label for="g">G:</label>
<input type="number" id="g" name="g" step="0.01" required>
<br>
<label for="po">pollen analysis:</label>
<input type="number" id="po" name="po" step="0.01" required>
<br>
<label for="vis">viscosity:</label>
<input type="number" id="vis" name="vis" step="0.01" required>
<br>
<label for="pu">purity:</label>
<input type="number" id="pu" name="pu" step="0.01" required>
<br>
<button id="bu">Predict</button>
</form>
</div>
</div>

```

```

</body>
</html>

```

```

INDEX.CSS body{
    box-sizing: border-box;
    margin: 0; padding: 0;
    background-image: url(img.jpg);
    background-size: 100% 800px;
    object-fit: contain;
    background-repeat: no-repeat;

} #lg{
    background: linear-gradient(to bottom right,rgba(96, 92, 92, 0.5),rgba(168,168,168,0.5));
    width: 100%; height: 750px;
}

```

```

#ns{
    position: sticky;
}
#name{
    font-size: 40px;
margin: 0 5%; padding:
15px;
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    color: wheat;
    position: static;

}

#nav ul{ list-style-
type: none;
    display: flex;
justify-content: flex-end;
gap: 10px; margin:-55px
100px;
    position: sticky;

} #nav li a{ text-
decoration: none;
font-size: 25px;
color: black; padding:
10px; }
#nav li a.active{
color:gold;
} #pr{
    background-color: skyblue;
padding: 2px; border-radius:
25px;
}
@media screen and (min-width:0) and (max-width:768px) {
    #nav ul{
width: auto;
height: auto;
overflow: hidden;
position: relative;
margin-top:25px;
display: block;
    }
    #nav li{
float:left; }
    #nav li a{
padding:10px;
    }
}

```

```

} #tq{
margin: 0;
padding:
2px;
    text-align: center;

} #title{ font-size: 45px; font-
family:Arial, Helvetica, sans-serif;
margin-top: 200px; color: white; }
#quo{
    font-size: 18px; font-family:sans-
serif;
    font-style: italic;
margin-top: -15px; color:
whitesmoke;
}
/ABout/ #about{ width: 100%;
height: 750px; padding: 10px;
background-color: blanchedalmond;
}
#nava ul{ list-style-type:
none; display: flex;
justify-content: flex-end;
gap: 10px; margin:50px
100px;
    position: sticky;

} #nava li a{ text-
decoration: none;
font-size: 25px;
color: black; padding:
10px; } #nava li
a.active{ color:gold;
} #ac{
    display: flex; margin-
left: 70px; margin-top:
10%;
} #au{ font-size:
40px; margin-left:
8%; margin-top: -
50px; font-family:
'Segoe UI', Tahoma,
Geneva, Verdana,
sans-serif;

}
#cont{ width: 50%; height: auto;
padding: 10px; font-size: 20px; margin-

```

```

left: 5%;    font-family:'Times New Roman',
Times, serif;
}
/predict/
#predict{
width: 100%;
height: 780px;
padding: 10px;
    background-color:rgb(240, 222, 115);
}
#namep{
    font-size: 40px;
margin: 0 5%;    padding:
15px;
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    color: black;
} #qu{
    text-align: center;    font-
size: 25px;
}
#formbox{    width:35%;
height: auto;    border:
2px solid black;
border-radius: 15px;
margin-left: 32%;
padding: 10px; }
#formbox h1 {
    text-align: center; }
#formbox label{    font-
size: 20px;
    margin-left: 20%;

}
#formbox input[type=number]{
border: 1px solid beige;
border-radius: 10px;    margin:
10px ;
}
#formbox input[type=number]#cs{
    margin-left: 95px ;
}
#formbox input[type=number]#D{    margin-
left:60px ;
} #formbox input[type=number]#w{    margin-
left: 90px ;
} #formbox input[type=number]#P{    margin-
left: 100px ;
} #formbox input[type=number]#E{

```

```

    margin-left: 100px ;
}#formbox input[type=number]#f{    margin-
left: 115px ;
}#formbox input[type=number]#g{    margin-
left: 115px ;
}#formbox input[type=number]#vis{    margin-
left: 60px ;
}
#formbox input[type=number]#pu{    margin-
left: 80px ;
}
#bu{
    cursor: pointer;
margin: 15px 45%;    font-
size: 20px;    background-
color: white;
    color:black;
padding: 5px 8px;
    border-radius: 12px;

}

```

PREDICT.HTML

```

<!DOCTYPE html>
<html>
    <head>
        <title>HoneyGoldBliss Prediction</title>
        <link rel="stylesheet" href="{{ url_for('static', filename='assets/css/inner_page.css') }}">
    </head>
    <body>
        <header id="name">HoneyGoldBliss&#128029;</header>
        <h1 id="output">{{ prediction_text }}</h1>
    </body>

```

PREDICT.CSS

```

body{
    box-sizing: border-box;
padding: 0;    margin: 0;
    background-image: url("../img/img2.jpeg");
background-size: 100% 400%;    background-repeat:
no-repeat;
    object-fit: contain;
}
#name{
    font-size: 40px;
margin: 0 5%;
padding: 15px;
font-family: Verdana, Geneva, Tahoma, sans-serif;

```



```

    color: wheat;
position: static;
} ul{    list-style-type:
none;
    margin:-50px 90% -10px ;

} li a{    text-
decoration: none;
font-size: 25px;
color: black;    padding:
10px;
} #ho{
    background-color: skyblue;
padding: 5px;    border-radius:
25px;
}
#output{
    text-align: center;
color: white;    margin-
top:10%;
}

```

APP.PY

```

import pickle import
pandas as pd import
os
from flask import Flask,request,render_template

app=Flask(__name__)
model = pickle.load(open('RaRmodel.pkl','rb'))

@app.route('/') def home():    return
render_template('index.html')

@app.route('/inner_page',methods=["POST","GET"]) def
predict():
    input_features = [float(x) for x in request.form.values()]
features_values = [np.array(input_features) ]    print(features_values)

names=['cs','D','w','p','E','f','g','po','Vis','pu']

df = pd.DataFrame(features_values, columns= names)

```


```
prediction = model.predict(df)
print(prediction[0])
rounded_value = round(prediction[0], 2)
text="Hence,based on calculation, the predicted Honey Price is: "

return render_template('inner_page.html', prediction_text=text + str(rounded_value))


if __name__ == '__main__':
    app.run(debug = False, port=5000)
```

Code snippets

Data collection



The screenshot shows a JupyterLab window with a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The code cell contains the following Python code:

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

import warnings
warnings.filterwarnings("ignore")
```

data importing and reading

```
[2]: data=pd.read_csv('honey_purity_dataset.csv')
data
```

[2]:

	CS	Density	WC	pH	EC	F	G	Pollen_analysis	Viscosity	Purity	Price
0	2.81	1.75	23.04	6.29	0.76	39.02	33.63	Blueberry	4844.50	0.68	645.24
1	9.47	1.82	17.50	7.20	0.71	38.15	34.41	Alfalfa	6689.02	0.89	385.85
2	4.61	1.84	23.72	7.31	0.80	27.47	34.36	Chestnut	6883.60	0.66	639.64
3	1.77	1.40	16.61	4.01	0.78	31.52	28.15	Blueberry	7167.56	1.00	946.46
4	6.11	1.25	19.63	4.82	0.90	29.65	42.52	Alfalfa	5125.44	1.00	432.62
...
247898	1.98	1.29	17.90	4.82	0.89	36.10	34.69	Rosemary	8261.63	1.00	754.98
247899	6.18	1.67	19.54	4.91	0.85	31.15	20.82	Acacia	6939.39	1.00	543.41
247900	7.78	1.49	15.78	5.69	0.73	44.60	44.07	Chestnut	4139.79	0.64	615.46
247901	5.78	1.74	14.96	6.81	0.83	47.19	37.79	Avocado	4417.74	0.97	949.32
247902	8.96	1.86	18.62	6.89	0.86	25.94	42.88	Lavender	8119.62	0.64	384.48

247903 rows × 11 columns

FileEditViewRunKernelSettingsHelp

Trusted

+

✂

▶

■

↺

↻

⏮

⏭

Code

▼

JupyterLabPython 3 (ipykernel)

handling missing values

```
[3]: data.shape
```

[3]: (247903, 11)

```
[4]: data.head()
```

[4]:

	CS	Density	WC	pH	EC	F	G	Pollen_analysis	Viscosity	Purity	Price
0	2.81	1.75	23.04	6.29	0.76	39.02	33.63	Blueberry	4844.50	0.68	645.24
1	9.47	1.82	17.50	7.20	0.71	38.15	34.41	Alfalfa	6689.02	0.89	385.85
2	4.61	1.84	23.72	7.31	0.80	27.47	34.36	Chestnut	6883.60	0.66	639.64
3	1.77	1.40	16.61	4.01	0.78	31.52	28.15	Blueberry	7167.56	1.00	946.46
4	6.11	1.25	19.63	4.82	0.90	29.65	42.52	Alfalfa	5125.44	1.00	432.62

```
[5]: data.tail()
```

[5]:

	CS	Density	WC	pH	EC	F	G	Pollen_analysis	Viscosity	Purity	Price
247898	1.98	1.29	17.90	4.82	0.89	36.10	34.69	Rosemary	8261.63	1.00	754.98
247899	6.18	1.67	19.54	4.91	0.85	31.15	20.82	Acacia	6939.39	1.00	543.41
247900	7.78	1.49	15.78	5.69	0.73	44.60	44.07	Chestnut	4139.79	0.64	615.46
247901	5.78	1.74	14.96	6.81	0.83	47.19	37.79	Avocado	4417.74	0.97	949.32
247902	8.96	1.86	18.62	6.89	0.86	25.94	42.88	Lavender	8119.62	0.64	384.48

```
[4]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 247903 entries, 0 to 247902
Data columns (total 11 columns):
 #   Column                Non-Null count  Dtype
---  -
 0   CS                    247903 non-null float64
 1   Density               247903 non-null float64
 2   WC                    247903 non-null float64
 3   pH                    247903 non-null float64
 4   EC                    247903 non-null float64
 5   F                     247903 non-null float64
 6   G                     247903 non-null float64
 7   Pollen_analysis       247903 non-null object
 8   Viscosity              247903 non-null float64
 9   Purity                 247903 non-null float64
10   Price                 247903 non-null float64
dtypes: float64(10), object(1)
memory usage: 20.8+ MB
```

Handling categorical values

FileEditViewRunKernelSettingsHelpTrust

+

✂

▶

■

↺

▶

Code

▼

JupyterLabPython 3 (ipykernel)

Handling categorical values

[5]:

encoder = LabelEncoder()

data['Pollen_analysis'] = encoder.fit_transform(data['Pollen_analysis'])

print(data)

	CS	Density	WC	pH	EC	F	G	Pollen_analysis \
0	2.81	1.75	23.04	6.29	0.76	39.02	33.63	3
1	9.47	1.82	17.50	7.20	0.71	38.15	34.41	1
2	4.61	1.84	23.72	7.31	0.80	27.47	34.36	6
3	1.77	1.40	16.61	4.01	0.78	31.52	28.15	3
4	6.11	1.25	19.63	4.82	0.90	29.65	42.52	1
...
247898	1.98	1.29	17.90	4.82	0.89	36.10	34.69	13
247899	6.18	1.67	19.54	4.91	0.85	31.15	20.82	0
247900	7.78	1.49	15.78	5.69	0.73	44.60	44.07	6
247901	5.78	1.74	14.96	6.81	0.83	47.19	37.79	2
247902	8.96	1.86	18.62	6.89	0.86	25.94	42.88	10

	Viscosity	Purity	Price
0	4844.50	0.68	645.24
1	6689.02	0.89	385.85
2	6883.60	0.66	639.64
3	7167.56	1.00	946.46
4	5125.44	1.00	432.62
...
247898	8261.63	1.00	754.98
247899	6939.39	1.00	543.41
247900	4139.79	0.64	615.46
247901	4417.74	0.97	949.32
247902	8119.62	0.64	384.48

[247903 rows x 11 columns]

checking null values

[8]: data.isnull().sum()

[8]: CS 0
Density 0
WC 0
pH 0
EC 0
F 0
G 0
Pollen_analysis 0
Viscosity 0
Purity 0
Price 0
dtype: int64

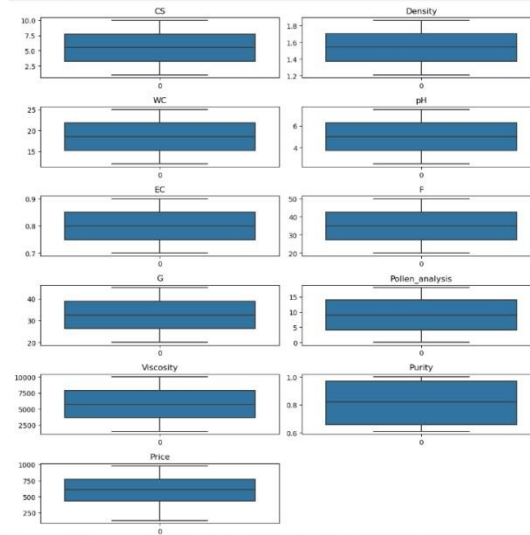
descriptive analysis

[9]: data.describe()

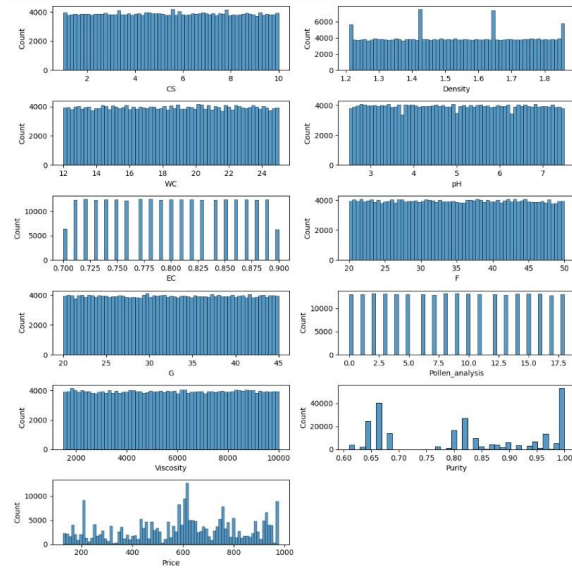
	CS	Density	WC	pH	EC	F	G	Pollen_analysis	Viscosity	Purity
count	247903.000000	247903.000000	247903.000000	247903.000000	247903.000000	247903.000000	247903.000000	247903.000000	247903.000000	247903.000000
mean	5.500259	1.535523	18.502625	4.996047	0.799974	34.970573	32.501006	8.993618	5752.893888	0.824471
std	2.593947	0.187824	3.748635	1.444060	0.057911	8.655898	7.226290	5.473649	2455.739903	0.139417
min	1.000000	0.210000	12.000000	2.500000	0.700000	20.000000	20.000000	0.000000	1500.050000	0.610000
25%	3.260000	1.370000	15.260000	3.750000	0.750000	27.460000	26.230000	4.000000	3627.880000	0.660000
50%	5.500000	1.540000	18.510000	4.990000	0.800000	34.970000	32.490000	9.000000	5753.770000	0.820000
75%	7.740000	1.700000	21.750000	6.250000	0.850000	42.470000	38.760000	14.000000	7886.650000	0.970000
max	10.000000	1.860000	25.000000	7.500000	0.900000	50.000000	45.000000	18.000000	9999.970000	1.000000

visual analysis / finding outliers

```
[10]: plt.figure(figsize=(11,11))
for i,col in enumerate(data.columns):
    plt.subplot(6,2,i+1)
    sns.boxplot(data[col])
    plt.title(col)
    plt.tight_layout()
```



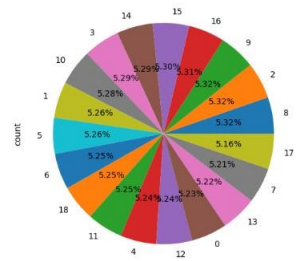
```
[11]: plt.figure(figsize=(11,11))
for i,col in enumerate(data.columns):
    plt.subplot(6,2,i+1)
    sns.histplot(data[col])
    plt.tight_layout()
```



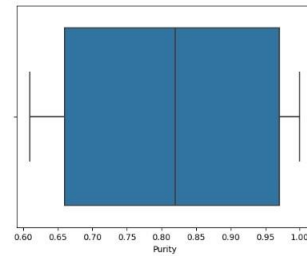
Univariayte analysis:

univariate analysis

```
[12]: plt.figure(figsize=(6,6))
      data["pollen_analysis"].value_counts().plot(kind="pie",subplots=True,autopct="%1.2f%%")
[13]: array([dxs: ylabel='count'], dtype=object)
```



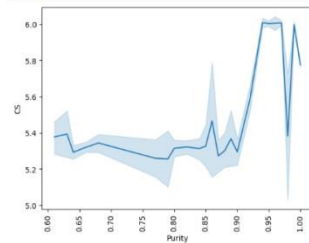
```
[13]: sns.boxplot(x="Purity",data=data)
      plt.show()
```



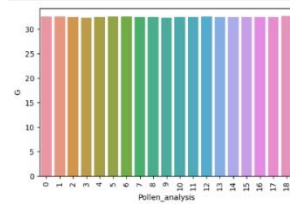
Bivariate analysis:

Bivariate analysis

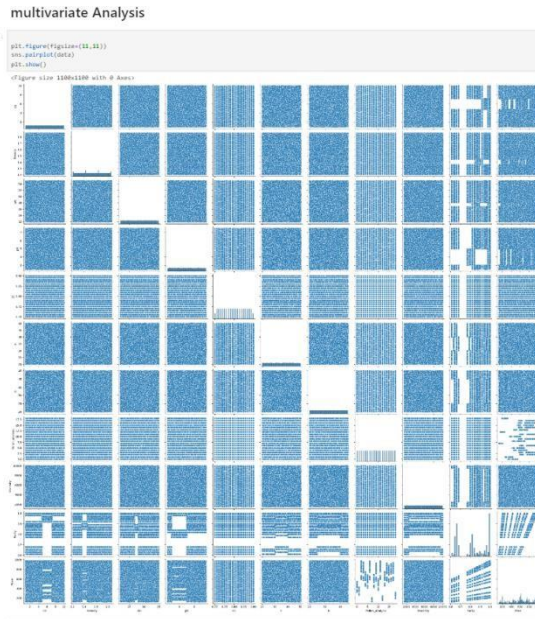
```
[14]: sns.lmplot(y="CS",x="Purity",data=data)
      plt.xticks(rotation=90)
      plt.show()
```



```
[15]: plt.figure(figsize=(6,6))
      sns.barplot(y="U",x="Pollen_analysis",data=data,color="#f08080")
      plt.xticks(rotation=90)
      plt.show()
```



Multivariate analysis:



Model Building:

```
[6]: x=data.drop(["Price"],axis=1)
      y=data["Price"]
      xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.20)

[7]: sts=StandardScaler()
      xtrain=sts.fit_transform(xtrain)
      xtest=sts.transform(xtest)
```

Linear Regression model

```
[23]: lr=LinearRegression()
      lr.fit(xtrain,ytrain)
      ypred=lr.predict(xtest)
      print(ypred)
      print("training accuracy",lr.score(xtrain,ytrain))
      print("testing accuracu",lr.score(xtest,ytest))
      mse=mean_squared_error(ypred,ytest)
      print("mean squared error:",mse)
      r2_lr=r2_score(ypred,ytest)
      print("r2 score",r2_lr)

[704.66490019 461.9722937 731.88227169 ... 460.51045169 513.20263708
 609.95242294]
training accuracy 0.18924656267633866
testing accuracu 0.1951221617711716
mean squared error: 43985.58793110899
r2 score -3.2814933916061033
```

Decision tree model

```
[24]: dt=DecisionTreeRegressor()  
dt.fit(xtrain,ytrain)  
ypred=dt.predict(xtest)  
print(ypred)  
print("training accuracy",dt.score(xtrain,ytrain))  
print("testing accuracu",dt.score(xtest,ytest))  
mse=mean_squared_error(ypred,ytest)  
print("mean squared error:",mse)  
r2_dt=r2_score(ypred,ytest)  
print("r2 score",r2_dt)
```

```
[946.46 621.56 926.3 ... 626.3 219.42 825.17]  
training accuracy 1.0  
testing accuracu 0.9999996378442584  
mean squared error: 0.01979136766100429  
r2 score 0.9999996378446946
```

Random Forest Regressor

```
[25]: rf=RandomForestRegressor()  
rf.fit(xtrain,ytrain)  
ypred=rf.predict(xtest)  
print(ypred)  
print("training accuracy",rf.score(xtrain,ytrain))  
print("testing accuracu",rf.score(xtest,ytest))  
mse=mean_squared_error(ypred,ytest)  
print("mean squared error:",mse)  
r2_rf=r2_score(ypred,ytest)  
print("r2 score",r2_rf)
```

```
[946.46 621.56 926.3 ... 626.3 219.42 825.0917]  
training accuracy 0.99999955348806  
testing accuracu 0.999997275270402  
mean squared error: 0.014890313488231902  
r2 score 0.9999997275267476
```

Comparing All The Models.

```
[26]: Accuracys =pd.DataFrame({"models":["LinearRegression","DecisionTree","RandomForestRegressor"],  
                             "R2score":[r2_lr,r2_dt,r2_rf]})  
Accuracys
```

```
[26]:
```

	models	R2score
0	LinearRegression	-3.281493
1	DecisionTree	1.000000
2	RandomForestRegressor	1.000000

Testing The Model

```
[27]: print(rf.predict(sts.transform([[6.78,1.22,14.84,3.5,0.83,41.63,26.52,0,7691.92,1.0]])))  
[543.41]
```

```
[28]: print(rf.predict(sts.transform([[5.55,4.55,66.5,4,555.2,55.3,666.7,5,888.6,66.5,]])))  
[684.45]
```

```
[29]: print(rf.predict(sts.transform([[6.56,4.54,67.5,4,545.2,54.3,454.7,4,77.4,0.12,]])))  
[454.17]
```

```
[30]: print(rf.predict(sts.transform([[2.81,1.75,23.04,6.29,0.76,39.02,33.63,3,4844.5,0.68]])))  
[645.24]
```