Dendrite.ai Data Science Assignment

```
import pandas as pd
import numpy as np
import json
```

```
df=pd.read_csv('/content/iris.csv')
df.head()
```

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

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
from sklearn.preprocessing import OneHotEncoder
ohe=OneHotEncoder()
df1=pd.DataFrame(ohe.fit_transform(df[['species']]).toarray(),columns=df['species'].unique())
df=pd.concat([df,df1],axis=1)
df.drop('species', axis=1,inplace=True)
```

```
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	Iris-setosa	Iris-versicolor	Iris-virginica
0	5.1	3.5	1.4	0.2	1.0	0.0	0.0
1	4.9	3.0	1.4	0.2	1.0	0.0	0.0
2	4.7	3.2	1.3	0.2	1.0	0.0	0.0
3	4.6	3.1	1.5	0.2	1.0	0.0	0.0
4	5.0	3.6	1.4	0.2	1.0	0.0	0.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df_json = pd.read_json('/content/algoparams.json')
df_json
```

	session_name	session_description	design_state_data
session_info	test	test	{'project_id': '1', 'experiment_id': 'kkkk-11'...
target	test	test	{'prediction_type': 'Regression', 'target': 'p...
train	test	test	{'policy': 'Split the dataset', 'time_variable...
metrics	test	test	{'optomize_model_hyperparameters_for': 'AUC', ...
feature_handling	test	test	{'sepal_length': {'feature_name': 'sepal_lengt...
feature_generation	test	test	{'linear_interactions': [['petal_length', 'sep...
feature_reduction	test	test	{'feature_reduction_method': 'Tree-based', 'nu...
hyperparameters	test	test	{'stratergy': 'Grid Search', 'shuffle_grid': T...
weighting_stratergy	test	test	{'weighting_stratergy_method': 'Sample weights...
probability_calibration	test	test	{'probability_calibration_method': 'Sigmoid - ...
algorithms	test	test	{'RandomForestClassifier': {'model_name': 'Ran

Next steps: Generate code with df_json | View recommended plots | New interactive sheet

```
import findspark
findspark.init()
```

```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Dendrite').getOrCreate()
```

```
spark = SparkSession.builder.appName("Dendrite").getOrCreate()
```

```
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
    v3.5.5
Master
    local[*]
AppName
    Dendrite

```
df_json_pyspark=spark.read.option("multiline","true").json("/content/algoparams.json")
```

```
display(df_json_pyspark )
```

```
DataFrame[design_state_data:
struct<algorithms:struct<DecisionTreeClassifier:struct<is_selected:boolean,max_depth:bigint,min_depth:bigint,min_samples_per_leaf:a
session_description: string, session_name: string]
```

```
df_json_pyspark.show()
```

```
+--------------------+-------------------+------------+
|   design_state_data|session_description|session_name|
+--------------------+-------------------+------------+
|{{{false, 7, 4, [...|               test|        test|
+--------------------+-------------------+------------+
```

```
df_json_pyspark.printSchema()
```

```
 |    |    |-- split: string (nullable = true)
 |    |    |-- time_variable: string (nullable = true)
 |    |    |-- train_ratio: long (nullable = true)
 |    |-- weighting_stratergy: struct (nullable = true)
 |    |    |-- weighting_stratergy_method: string (nullable = true)
 |    |    |-- weighting_stratergy_weight_variable: string (nullable = true)
 |-- session_description: string (nullable = true)
 |-- session_name: string (nullable = true)
```

```
from pyspark.sql import DataFrame
from pyspark.sql.functions import col, explode_outer
from pyspark.sql.types import StructType, ArrayType

def flatten(df: DataFrame, verbose: bool = False) -> DataFrame:
    complex_fields = {field.name: field.dataType for field in df.schema.fields
                      if isinstance(field.dataType, (StructType, ArrayType))}

    while complex_fields:
        col_name = list(complex_fields.keys())[0]
        col_type = complex_fields[col_name]

        if verbose:
            print(f"Processing: {col_name} | Type: {type(col_type).__name__}")

        if isinstance(col_type, StructType):
            expanded_cols = [
                col(f"{col_name}.{nested.name}").alias(f"{col_name}_{nested.name}")
                for nested in col_type
            ]
            df = df.select("*", *expanded_cols).drop(col_name)

        elif isinstance(col_type, ArrayType):
            df = df.withColumn(col_name, explode_outer(col_name))

        complex_fields = {field.name: field.dataType for field in df.schema.fields
                          if isinstance(field.dataType, (StructType, ArrayType))}

    return df
```

```
df_flatten = flatten(df_json_pyspark )
df_flatten.show()
```

```
+-------------------+------------+--------------------------------------------------------------------+----------------------------
|session_description|session_name|design_state_data_feature_generation_explicit_pairwise_interactions|design_state_data_feature_gen
+-------------------+------------+--------------------------------------------------------------------+----------------------------
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
|               test|        test|                                               sepal_width/sepal...|
+-------------------+------------+--------------------------------------------------------------------+----------------------------
only showing top 20 rows
```

```
df_flatten.describe()
```

string, design_state_data_algorithms_neural_network_alpha_value: string, design_state_data_algorithms_neural_network_beta_1: string, design_state_data_algorithms_neural_network_beta_2: string, design_state_data_algorithms_neural_network_convergence_tolerance: string, design_state_data_algorithms_neural_network_epsilon: string, design_state_data_algorithms_neural_network_hidden_layer_sizes: string, design_state_data_algorithms_neural_network_initial_learning_rate: string, design_state_data_algorithms_neural_network_max_iterations: string, design_state_data_algorithms_neural_network_model_name: string, design_state_data_algorithms_neural_network_momentum: string, design_state_data_algorithms_neural_network_power_t: string, design_state_data_algorithms_neural_network_solver: string, design_state_data_algorithms_xg_boost_col_sample_by_tree: string, design_state_data_algorithms_xg_boost_early_stopping_rounds: string, design_state_data_algorithms_xg_boost_gamma: string, design_state_data_algorithms_xg_boost_l1_regularization: string, design_state_data_algorithms_xg_boost_l2_regularization: string, design_state_data_algorithms_xg_boost_learningRate: string, design_state_data_algorithms_xg_boost_max_depth_of_tree: string, design_state_data_algorithms_xg_boost_max_num_of_trees: string, design_state_data_algorithms_xg_boost_min_child_weight: string, design_state_data_algorithms_xg_boost_model_name: string, design_state_data_algorithms_xg_boost_parallelism: string, design_state_data_algorithms_xg_boost_random_state: string, design_state_data_algorithms_xg_boost_sub_sample: string, design_state_data_algorithms_xg_boost_tree_method: string, design_state_data_feature_handling_petal_length_feature_name: string, design_state_data_feature_handling_petal_length_feature_variable_type: string, design_state_data_feature_handling_petal_width_feature_name: string, design_state_data_feature_handling_petal_width_feature_variable_type: string, design_state_data_feature_handling_sepal_length_feature_name: string, design_state_data_feature_handling_sepal_length_feature_variable_type: string, design_state_data_feature_handling_sepal_width_feature_name: string, design_state_data_feature_handling_sepal_width_feature_variable_type: string, design_state_data_feature_handling_species_feature_name: string, design_state_data_feature_handling_species_feature_variable_type: string, design_state_data_feature_handling_petal_length_feature_details_impute_value: string, design_state_data_feature_handling_petal_length_feature_details_impute_with: string, design_state_data_feature_handling_petal_length_feature_details_missing_values: string, design_state_data_feature_handling_petal_length_feature_details_numerical_handling: string, design_state_data_feature_handling_petal_length_feature_details_rescaling: string, design_state_data_feature_handling_petal_width_feature_details_impute_value: string, design_state_data_feature_handling_petal_width_feature_details_impute_with: string, design_state_data_feature_handling_petal_width_feature_details_missing_values: string, design_state_data_feature_handling_petal_width_feature_details_numerical_handling: string, design_state_data_feature_handling_petal_width_feature_details_rescaling: string, design_state_data_feature_handling_sepal_length_feature_details_impute_value: string, design_state_data_feature_handling_sepal_length_feature_details_impute_with: string, design_state_data_feature_handling_sepal_length_feature_details_missing_values: string, design_state_data_feature_handling_sepal_length_feature_details_numerical_handling: string, design_state_data_feature_handling_sepal_length_feature_details_rescaling: string, design_state_data_feature_handling_sepal_width_feature_details_impute_value: string, design_state_data_feature_handling_sepal_width_feature_details_impute_with: string, design_state_data_feature_handling_sepal_width_feature_details_missing_values: string, design_state_data_feature_handling_sepal_width_feature_details_numerical_handling: string, design_state_data_feature_handling_sepal_width_feature_details_rescaling: string, design_state_data_feature_handling_species_feature_details_hash_columns: string, design_state_data_feature_handling_species_feature_details_text_handling: string]

1) Read the target and type of regression to be run.

```
target=df_json.loc['target','design_state_data']['target']
type_of_reg=df_json.loc['target','design_state_data']['type']
```

```
target
```

```
type_of_reg
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
```

```
p = df[["sepal_length", "sepal_width", "petal_length"]]
q = df["petal_width"]
```

```
p_train, p_test, q_train, q_test = train_test_split(p, q, test_size=0.2, random_state=42)
```

```
models={
    'LinearRegression':LinearRegression(),
    'Ridge':Ridge(),
    'Lasso':Lasso(),
    'ElasticNet':ElasticNet(),
    'RandomForestRegressor':RandomForestRegressor(),
    'GradientBoostingRegressor':GradientBoostingRegressor(),
    'DecisionTreeRegressor':DecisionTreeRegressor()
}
```

```
for name, model in models.items():
    model.fit(p_train, q_train)
    score = model.score(p_test, q_test)
    print(f'{name}: {score:.3f}')
```

```
LinearRegression: 0.927
Ridge: 0.928
Lasso: 0.329
ElasticNet: 0.698
RandomForestRegressor: 0.930
GradientBoostingRegressor: 0.924
DecisionTreeRegressor: 0.874
```

2) Read the features (which are column names in the csv) and figure out what missing imputation needs to be applied and apply that to the columns loaded in a dataframe.

```
feature_dict=df_json.loc['feature_handling','design_state_data']
```

```
def feature_handling(feature_handling, column_names,df):
    for col in column_names:
        try:
            if feature_handling[col]['feature_details']['impute_with'] == 'custom':
                df[col] = df[col].fillna(feature_handling[col]['feature_details']['impute_value'])
            elif feature_handling[col]['feature_details']['impute_with'] == 'Average of values':
                df[col] = df[col].fillna(df[col].mean())
        except KeyError:
            print(col)
    return df
```

```
feature_handling(feature_dict, df.columns, df)
```

```
Iris-setosa
Iris-versicolor
Iris-virginica
```

	sepal_length	sepal_width	petal_length	petal_width	Iris-setosa	Iris-versicolor	Iris-virginica
0	5.1	3.5	1.4	0.2	1.0	0.0	0.0
1	4.9	3.0	1.4	0.2	1.0	0.0	0.0
2	4.7	3.2	1.3	0.2	1.0	0.0	0.0
3	4.6	3.1	1.5	0.2	1.0	0.0	0.0
4	5.0	3.6	1.4	0.2	1.0	0.0	0.0
...	...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	0.0	0.0	1.0
146	6.3	2.5	5.0	1.9	0.0	0.0	1.0
147	6.5	3.0	5.2	2.0	0.0	0.0	1.0
148	6.2	3.4	5.4	2.3	0.0	0.0	1.0
149	5.9	3.0	5.1	1.8	0.0	0.0	1.0

150 rows × 7 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df_csv_pyspark = spark.read.csv('/content/iris.csv',header=True,inferSchema=True)
```

```
df_csv_pyspark.printSchema()
```

```
root
 |-- sepal_length: double (nullable = true)
 |-- sepal_width: double (nullable = true)
 |-- petal_length: double (nullable = true)
 |-- petal_width: double (nullable = true)
 |-- species: string (nullable = true)
```

```
df_csv_pyspark.show()
```

```
+------------+-----------+------------+-----------+-----------+
|sepal_length|sepal_width|petal_length|petal_width|    species|
+------------+-----------+------------+-----------+-----------+
```

```
|         5.1|        3.5|         1.4|        0.2|Iris-setosa|
|         4.9|        3.0|         1.4|        0.2|Iris-setosa|
|         4.7|        3.2|         1.3|        0.2|Iris-setosa|
|         4.6|        3.1|         1.5|        0.2|Iris-setosa|
|         5.0|        3.6|         1.4|        0.2|Iris-setosa|
|         5.4|        3.9|         1.7|        0.4|Iris-setosa|
|         4.6|        3.4|         1.4|        0.3|Iris-setosa|
|         5.0|        3.4|         1.5|        0.2|Iris-setosa|
|         4.4|        2.9|         1.4|        0.2|Iris-setosa|
|         4.9|        3.1|         1.5|        0.1|Iris-setosa|
|         5.4|        3.7|         1.5|        0.2|Iris-setosa|
|         4.8|        3.4|         1.6|        0.2|Iris-setosa|
|         4.8|        3.0|         1.4|        0.1|Iris-setosa|
|         4.3|        3.0|         1.1|        0.1|Iris-setosa|
|         5.8|        4.0|         1.2|        0.2|Iris-setosa|
|         5.7|        4.4|         1.5|        0.4|Iris-setosa|
|         5.4|        3.9|         1.3|        0.4|Iris-setosa|
|         5.1|        3.5|         1.4|        0.3|Iris-setosa|
|         5.7|        3.8|         1.7|        0.3|Iris-setosa|
|         5.1|        3.8|         1.5|        0.3|Iris-setosa|
+------------+-----------+------------+-----------+-----------+
only showing top 20 rows
```

```
df_csv_pyspark=df_csv_pyspark.drop('species')
```

```
df_csv_pyspark.show()
```

```
+------------+-----------+------------+-----------+
|sepal_length|sepal_width|petal_length|petal_width|
+------------+-----------+------------+-----------+
|         5.1|        3.5|         1.4|        0.2|
|         4.9|        3.0|         1.4|        0.2|
|         4.7|        3.2|         1.3|        0.2|
|         4.6|        3.1|         1.5|        0.2|
|         5.0|        3.6|         1.4|        0.2|
|         5.4|        3.9|         1.7|        0.4|
|         4.6|        3.4|         1.4|        0.3|
|         5.0|        3.4|         1.5|        0.2|
|         4.4|        2.9|         1.4|        0.2|
|         4.9|        3.1|         1.5|        0.1|
|         5.4|        3.7|         1.5|        0.2|
|         4.8|        3.4|         1.6|        0.2|
|         4.8|        3.0|         1.4|        0.1|
|         4.3|        3.0|         1.1|        0.1|
|         5.8|        4.0|         1.2|        0.2|
|         5.7|        4.4|         1.5|        0.4|
|         5.4|        3.9|         1.3|        0.4|
|         5.1|        3.5|         1.4|        0.3|
|         5.7|        3.8|         1.7|        0.3|
|         5.1|        3.8|         1.5|        0.3|
+------------+-----------+------------+-----------+
only showing top 20 rows
```

```
from pyspark.ml.feature import Imputer

imputer = Imputer(
    inputCols=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
    outputCols=["{}_imputed".format(c) for c in ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
    ).setStrategy("mean")
```

```
imputer.fit(df_csv_pyspark).transform(df_csv_pyspark).show()
```

```
+------------+-----------+------------+-----------+--------------------+-------------------+--------------------+------------------
|sepal_length|sepal_width|petal_length|petal_width|sepal_length_imputed|sepal_width_imputed|petal_length_imputed|petal_width_impute
+------------+-----------+------------+-----------+--------------------+-------------------+--------------------+------------------
|         5.1|        3.5|         1.4|        0.2|                 5.1|                3.5|                 1.4|                0.
|         4.9|        3.0|         1.4|        0.2|                 4.9|                3.0|                 1.4|                0.
|         4.7|        3.2|         1.3|        0.2|                 4.7|                3.2|                 1.3|                0.
|         4.6|        3.1|         1.5|        0.2|                 4.6|                3.1|                 1.5|                0.
|         5.0|        3.6|         1.4|        0.2|                 5.0|                3.6|                 1.4|                0.
|         5.4|        3.9|         1.7|        0.4|                 5.4|                3.9|                 1.7|                0.
|         4.6|        3.4|         1.4|        0.3|                 4.6|                3.4|                 1.4|                0.
|         5.0|        3.4|         1.5|        0.2|                 5.0|                3.4|                 1.5|                0.
|         4.4|        2.9|         1.4|        0.2|                 4.4|                2.9|                 1.4|                0.
|         4.9|        3.1|         1.5|        0.1|                 4.9|                3.1|                 1.5|                0.
|         5.4|        3.7|         1.5|        0.2|                 5.4|                3.7|                 1.5|                0.
|         4.8|        3.4|         1.6|        0.2|                 4.8|                3.4|                 1.6|                0.
|         4.8|        3.0|         1.4|        0.1|                 4.8|                3.0|                 1.4|                0.
|         4.3|        3.0|         1.1|        0.1|                 4.3|                3.0|                 1.1|                0.
|         5.8|        4.0|         1.2|        0.2|                 5.8|                4.0|                 1.2|                0.
|         5.7|        4.4|         1.5|        0.4|                 5.7|                4.4|                 1.5|                0.
|         5.4|        3.9|         1.3|        0.4|                 5.4|                3.9|                 1.3|                0.
|         5.1|        3.5|         1.4|        0.3|                 5.1|                3.5|                 1.4|                0.
|         5.7|        3.8|         1.7|        0.3|                 5.7|                3.8|                 1.7|                0.
```

```
|       5.1|       3.8|        1.5|       0.3|                 5.1|                 3.8|                 1.5|                 0.
+----------+----------+-----------+----------+--------------------+--------------------+--------------------+-----------------
only showing top 20 rows
```

3) Compute feature reduction based on input. See the screenshot below where there can be No Reduction, Corr with Target, Tree-based, PCA. Please make sure you write code so that all options can work. If we rerun your code with a different Json it should work if we switch No Reduction to say PCA.

```
df = pd.read_csv('/content/iris.csv')
df
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df['species'] =df['species'].astype('category').cat.codes
df.corr()
```

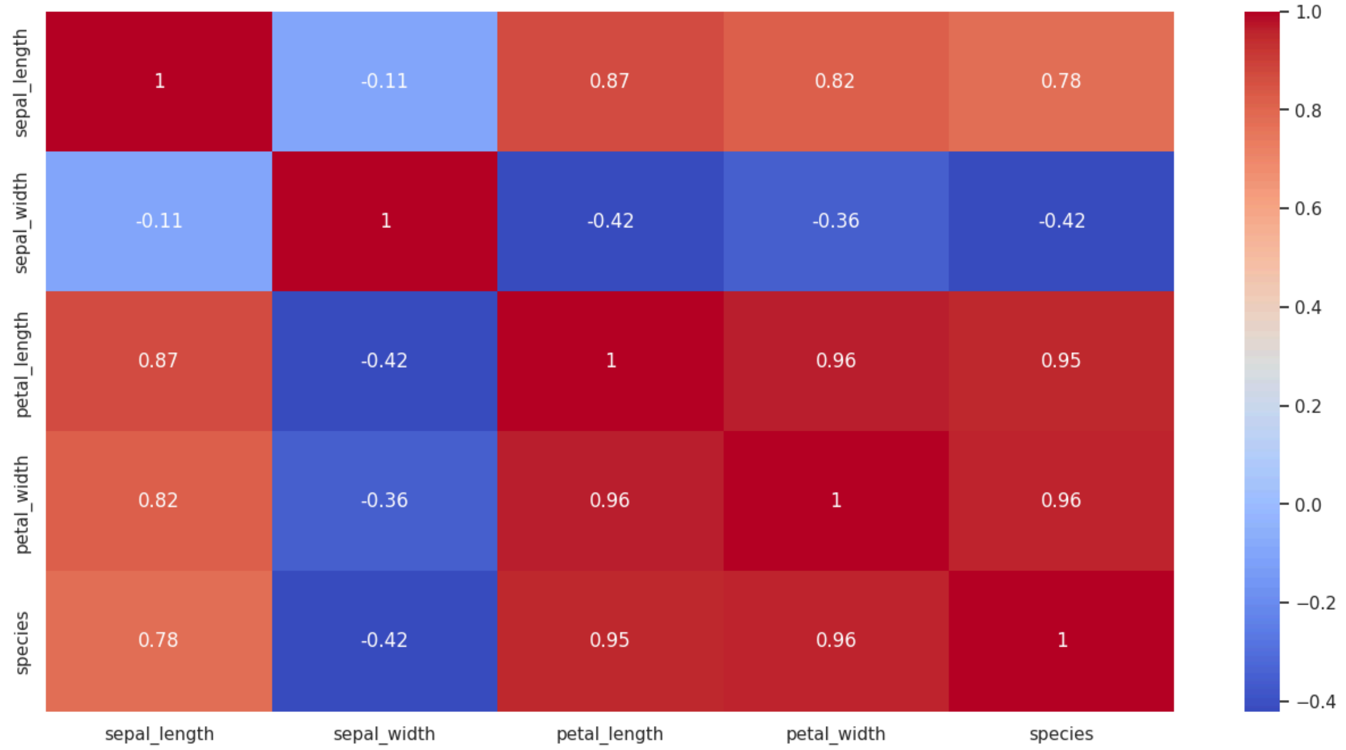
	sepal_length	sepal_width	petal_length	petal_width	species
sepal_length	1.000000	-0.109369	0.871754	0.817954	0.782561
sepal_width	-0.109369	1.000000	-0.420516	-0.356544	-0.419446
petal_length	0.871754	-0.420516	1.000000	0.962757	0.949043
petal_width	0.817954	-0.356544	0.962757	1.000000	0.956464
species	0.782561	-0.419446	0.949043	0.956464	1.000000

Correlation matrix heatmap

```
import seaborn as sns
```

```
sns.set(rc = {'figure.figsize':(16,8)})
sns.heatmap(df.corr(), annot = True, fmt='.2g',cmap= 'coolwarm')
```

<Axes: >




```
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
from scipy.stats import pearsonr
```

```
X = pd.DataFrame(df)
X
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Next steps: Generate code with X | View recommended plots | New interactive sheet

```
y = X.pop('species')
y
```

	species
0	0
1	0
2	0
3	0
4	0
...	...
145	2
146	2
147	2
148	2
149	2

150 rows × 1 columns

```
def no_reduction(X, y):
    return X

def corr_with_target(X, y, threshold=0.5):
    corr_with_target = X.corrwith(y).abs()
    features_to_keep = corr_with_target[corr_with_target >= threshold].index
    return X[features_to_keep]

def tree_based(X, y, n_features=3):
    model = RandomForestRegressor(n_estimators=100, random_state=0)
    model.fit(X, y)
    feature_importances = model.feature_importances_
    features_to_keep = X.columns[np.argsort(feature_importances)[::-1][:n_features]]
    return X[features_to_keep]

def pca_reduction(X, y, n_components=2):
    pca = PCA(n_components=n_components)
    X_reduced = pca.fit_transform(X)
    cols = ['PC'+str(i) for i in range(1, n_components+1)]
    X_reduced_df = pd.DataFrame(X_reduced, columns=cols, index=X.index)
    return X_reduced_df
```

```
reduction_methods = {
    'No Reduction': no_reduction,
    'Corr with Target': corr_with_target,
    'Tree-based': tree_based,
    'PCA': pca_reduction
}
```

```
selected_method = 'Corr with Target'
```

```
X_reduced = reduction_methods[selected_method](X, y)
```

```
print("Original number of features: ", X.shape[1])
print("Selected feature reduction method: ", selected_method)
print("Number of features after feature reduction: ", X_reduced.shape[1])
print("Selected features: ", X_reduced.columns)
```

```
Original number of features:  4
Selected feature reduction method:  Corr with Target
Number of features after feature reduction:  3
Selected features:  Index(['sepal_length', 'petal_length', 'petal_width'], dtype='object')
```

4) Parse the Json and make the model objects (using sklean) that can handle what is required in the “prediction_type” specified in the JSON (See 1 where “prediction_type” is specified). Keep in mind not to pick models that don’t apply for the prediction_type specified.

```
df_json.loc['algorithms']['design_state_data']
```

```
  'use_random': True},
 'SVM': {'model_name': 'Support Vector Machine',
  'is_selected': False,
  'linear_kernel': True,
  'rep_kernel': True,
  'polynomial_kernel': True,
  'sigmoid_kernel': True,
  'c_value': [566, 79],
  'auto': True,
  'scale': True,
  'custom_gamma_values': True,
  'tolerance': 7,
  'max_iterations': 7},
 'SGD': {'model_name': 'Stochastic Gradient Descent',
  'is_selected': False,
  'use_logistics': True,
  'use_modified_hubber_loss': False,
  'max_iterations': False,
  'tolerance': 56,
  'use_l1_regularization': 'on',
  'use_l2_regularization': 'on',
  'use_elastic_net_regularization': True,
  'alpha_value': [79, 56],
  'parallelism': 1},
 'KNN': {'model_name': 'KNN',
  'is_selected': False,
  'k_value': [78],
  'distance_weighting': True,
  'neighbour_finding_algorithm': 'Automatic',
  'random_state': 0,
  'p_value': 0},
 'extra_random_trees': {'model_name': 'Extra Random Trees',
  'is_selected': False,
  'num_of_trees': [45, 489],
  'feature_sampling_statergy': 'Square root and Logarithm',
  'max_depth': [12, 45],
  'min_samples_per_leaf': [78, 56],
  'parallelism': 3},
 'neural_network': {'model_name': 'Neural Network',
  'is_selected': False,
  'hidden_layer_sizes': [67, 89],
  'activation': '',
  'alpha_value': 0,
  'max_iterations': 0,
  'convergence_tolerance': 0,
  'early_stopping': True,
  'solver': 'ADAM',
  'shuffle_data': True,
  'initial_learning_rate': 0,
  'automatic_batching': True,
  'beta_1': 0,
  'beta_2': 0,
  'epsilon': 0,
  'power_t': 0,
  'momentum': 0,
  'use_nesterov_momentum': False}}
```

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
p = df[["sepal_length", "sepal_width", "petal_length"]]
q = df["petal_width"]
```

```
p_train, p_test, q_train, q_test = train_test_split(p, q, test_size=0.2, random_state=42)
```

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

models = {
    "Random Forest Regressor": RandomForestRegressor(n_estimators=100, random_state=42),
    "Gradient Boosting Regressor": GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42),
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=0.1),
    "Elastic Net Regression": ElasticNet(alpha=0.1, l1_ratio=0.5),
    "Decision Tree Regressor": DecisionTreeRegressor()
}

for name, model in models.items():
    model.fit(p_train, q_train)
    pred = model.predict(p_test)
    mse = mean_squared_error(q_test, pred)
    print(f"{name}: MSE = {mse:.4f}")
```

```
Random Forest Regressor: MSE = 0.0443
Gradient Boosting Regressor: MSE = 0.0538
Linear Regression: MSE = 0.0464
Ridge Regression: MSE = 0.0455
Lasso Regression: MSE = 0.0516
Elastic Net Regression: MSE = 0.0490
Decision Tree Regressor: MSE = 0.0787
```

5) Run the fit and predict on each model – keep in mind that you need to do hyper parameter tuning i.e., use GridSearchCV.

```
from sklearn.model_selection import GridSearchCV
```

```
p_train, p_test, q_train, q_test = train_test_split(p, q, test_size=0.2, random_state=42)
```

```
models = { "Random Forest Regressor": {"model": RandomForestRegressor(), "params": {"n_estimators": [50, 100, 200], "max_features": ["s
```

```
for name, mp in models.items():
    model = GridSearchCV(mp['model'], mp['params'], cv=3, n_jobs=-1, scoring='neg_mean_squared_error')
    model.fit(p_train, q_train)
    q_pred = model.predict(p_test)
    mse = mean_squared_error(q_test, q_pred)
    r2 = r2_score(q_test, q_pred)

    print(f"--->> {name}:")
    print(f" Best Parameters: {model.best_params_}")
    print(f" Mean Squared Error: {mse:.3f}")
    print(f" R^2 Score: {r2:.3f}")
    print("____________________________________________________________________________________________________________")
```

```
--->> Random Forest Regressor:
 Best Parameters: {'max_features': 'log2', 'n_estimators': 200}
 Mean Squared Error: 0.045
 R^2 Score: 0.929
____________________________________________________________________________________________________________
--->> GBT Regressor:
 Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
 Mean Squared Error: 0.044
 R^2 Score: 0.931
____________________________________________________________________________________________________________
--->> Linear Regression:
 Best Parameters: {}
 Mean Squared Error: 0.046
 R^2 Score: 0.927
____________________________________________________________________________________________________________
--->> Ridge Regression:
 Best Parameters: {'alpha': 0.01}
 Mean Squared Error: 0.046
 R^2 Score: 0.927
____________________________________________________________________________________________________________
--->> Lasso Regression:
 Best Parameters: {'alpha': 0.01}
 Mean Squared Error: 0.045
 R^2 Score: 0.930
____________________________________________________________________________________________________________
--->> Elastic Net Regression:
 Best Parameters: {'alpha': 0.01, 'l1_ratio': 0.25}
```

```
 Mean Squared Error: 0.045
 R^2 Score: 0.929
____________________________________________________________________________________________________
--->> Decision Tree Regressor:
 Best Parameters: {'max_depth': 3}
 Mean Squared Error: 0.052
 R^2 Score: 0.918
____________________________________________________________________________________________________
```

```
RandomForestRegressor_params = {
    'model_name': 'Random Forest Regressor',
    'is_selected': True,
    'min_trees': 10,
    'max_trees': 20,
    'feature_sampling_statergy': 'Default',
    'min_depth': 20,
    'max_depth': 25,
    'min_samples_per_leaf_min_value': 5,
    'min_samples_per_leaf_max_value': 10,
    'parallelism': 0
}

rf_param_grid = {
    'n_estimators': [10, 15, 20],
    'max_depth': [20, 23, 25],
    'min_samples_leaf': [5, 7, 10]
}

rf_model = RandomForestRegressor(random_state=42)
rf_gs = GridSearchCV(estimator=rf_model, param_grid=rf_param_grid, cv=3, n_jobs=-1, scoring='neg_mean_squared_error')

rf_gs.fit(df.drop(target, axis=1), df[target])

rf_best_model = rf_gs.best_estimator_

rf_preds = rf_best_model.predict(df.drop(target, axis=1))
rf_mse = mean_squared_error(df[target], rf_preds)

print("Model: Random Forest Regressor")
print("Best Parameters: ", rf_gs.best_params_)
print("MSE: ", rf_mse)
print("Predictions: ", rf_preds)
print("=" * 100)

gbt_param_grid = {
    'n_estimators': [67, 89],
    'max_depth': [5, 7],
    'learning_rate': [0.1, 0.3, 0.5],
    'subsample': [1.0, 1.5, 2.0]
}

gbt_model = GradientBoostingRegressor(random_state=42)
gbt_gs = GridSearchCV(estimator=gbt_model, param_grid=gbt_param_grid, cv=3, n_jobs=-1, scoring='neg_mean_squared_error')
gbt_gs.fit(df.drop(target, axis=1), df[target])
gbt_best = gbt_gs.best_estimator_
gbt_preds = gbt_best.predict(df.drop(target, axis=1))
gbt_mse = mean_squared_error(df[target], gbt_preds)

print("Model: Gradient Boosting Regressor")
print("Best Parameters: ", gbt_gs.best_params_)
print("MSE: ", gbt_mse)
print("Predictions: ", gbt_preds)
print("=" * 100)
```

```
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
36 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1382, in wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 436, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.11/dist-packages/sklearn/utils/_param_validation.py", line 98, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'subsample' parameter of GradientBoostingRegressor must be a float in

--------------------------------------------------------------------------------
36 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1382, in wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 436, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.11/dist-packages/sklearn/utils/_param_validation.py", line 98, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'subsample' parameter of GradientBoostingRegressor must be a float in

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are
 -0.52492398         nan         nan -0.52463028         nan         nan
 -0.52352311         nan         nan -0.52345582         nan         nan
 -0.59569559         nan         nan -0.59569548         nan         nan
 -0.57153728         nan         nan -0.57152569         nan         nan
 -0.59001683         nan         nan -0.59001682         nan         nan]
  warnings.warn(
```

6) Log to the console the standard model metrics that apply.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

p_train, p_test, q_train, q_test = train_test_split(p, q, test_size=0.3, random_state=0)

models = [
    LinearRegression(),
    Ridge(alpha=0.1),
    Lasso(alpha=0.1),
    ElasticNet(alpha=0.1),
    RandomForestRegressor(n_estimators=100, random_state=0),
    XGBRegressor(n_estimators=100, objective='reg:squarederror', random_state=0),
    LGBMRegressor(n_estimators=100, random_state=0)
]

rmse_list = []
mae_list = []

for model in models:
    model.fit(p_train, q_train)
    q_pred = model.predict(p_test)
    rmse = np.sqrt(mean_squared_error(q_test, q_pred))
    mae = mean_absolute_error(q_test, q_pred)
    rmse_list.append(rmse)
    mae_list.append(mae)

for i, model in enumerate(models):
    print(f"Model: {model.__class__.__name__}")
    print(f"RMSE: {rmse_list[i]:.3f}")
    print(f"MAE: {mae_list[i]:.3f}")
    print("=" * 30)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
Model: LinearRegression
RMSE: 0.221
MAE: 0.160
==============================
Model: Ridge
RMSE: 0.221
MAE: 0.160
==============================
Model: Lasso
RMSE: 0.233
MAE: 0.173
==============================
Model: ElasticNet
RMSE: 0.232
MAE: 0.170
==============================
Model: RandomForestRegressor
RMSE: 0.200
MAE: 0.145
==============================
```