waves

August 23, 2024

[7]:
```
import pandas as pd
import numpy as np
from typing import Dict
```

[8]:
```
import plotly.io as pio

pio.renderers.default = "notebook"
```

[9]:
```
def generate_columns() -> Dict[str, type]:
    """
    Generates the columns for the DataFrame. The column names are based off
    the variables table at https://archive.ics.uci.edu/dataset/882/
 ↪large-scale+wave+energy+farm

    Returns:
      Dict[str, type]: A dictionary of column names and their respective types
    """

    columns: dict = {}

    for i in range(1, 50):
        columns[f"X{i}"] = np.float64
        columns[f"Y{i}"] = np.float64

    for i in range(1, 50):
        columns[f"Power{i}"] = np.float64

    columns.update({"qW": np.float64, "Total_Power": np.float64})
    return columns
```

0.1 Loading the dataset

[10]:
```
df: pd.DataFrame = pd.read_csv(
    "../data/WEC_Perth_49.csv",
    dtype=generate_columns(),
)
```

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36043 entries, 0 to 36042
Columns: 149 entries, X1 to Total_Power
dtypes: float64(149)
memory usage: 41.0 MB
```

[12]:
```
df.shape
```

[12]:
```
(36043, 149)
```

[13]:
```
df.columns, df.index
```

[13]:
```
(Index(['X1', 'Y1', 'X2', 'Y2', 'X3', 'Y3', 'X4', 'Y4', 'X5', 'Y5',
        …
        'Power42', 'Power43', 'Power44', 'Power45', 'Power46', 'Power47',
        'Power48', 'Power49', 'qW', 'Total_Power'],
       dtype='object', length=149),
 RangeIndex(start=0, stop=36043, step=1))
```

[14]:
```
df.dtypes
```

[14]:
```
X1             float64
Y1             float64
X2             float64
Y2             float64
X3             float64
                …
Power47        float64
Power48        float64
Power49        float64
qW             float64
Total_Power    float64
Length: 149, dtype: object
```

[15]:
```
df.describe()
```

[15]:

	X1	Y1	X2	Y2	X3	\
count	36043.000000	36043.000000	36043.000000	36043.000000	36043.000000	
mean	366.597060	18.709550	426.314033	51.085762	477.295590	
std	307.911246	44.043295	265.781316	90.151852	270.322011	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	65.770000	0.000000	200.000000	0.000000	289.950000	
50%	250.000000	0.000000	346.090000	37.520000	400.000000	
75%	600.000000	0.000000	745.980000	37.900000	689.800000	
max	1000.000000	885.590000	1000.000000	939.260000	1000.000000	

```
                Y3            X4            Y4            X5            Y5  \
count  36043.000000  36043.000000  36043.000000  36043.000000  36043.000000
mean      57.846020    497.150488     73.323178    684.309548     44.012247
std       42.143917    279.631344     51.140816    237.862684     59.242702
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%       50.000000    300.000000     50.000000    600.000000      0.000000
50%       74.820000    500.000000    100.000000    700.000000      0.080000
75%       74.960000    632.750000    112.150000    850.000000     50.000000
max      990.000000   1000.000000    990.000000   1000.000000    919.590000

       …        Power42       Power43       Power44       Power45  \
count  …   36043.000000   36043.00000   36043.000000   36043.000000
mean   …   93678.772248   96530.68484   96666.293181   97007.214249
std    …    7401.226140    6709.53446    7020.690028    4829.877255
min    …   52516.130000   56391.97000   53877.360000   53050.330000
25%    …   88177.210000   94648.08000   96932.520000   97612.350000
50%    …   93694.540000   98729.91000   99269.310000   98857.150000
75%    …  100997.520000  100622.52000  100282.360000   99156.130000
max    …  110945.940000  109400.43000  114194.520000  106702.150000

             Power46       Power47       Power48       Power49  \
count   36043.000000  36043.000000  36043.000000  36043.000000
mean    98466.265281  98106.278501  97462.663041  96134.920454
std      4978.194259   4263.508074   3134.420742   3889.098339
min     55401.380000  63028.260000  61717.310000  47257.430000
25%     97629.940000  97154.630000  96869.740000  96319.550000
50%    100423.930000  99805.920000  98710.730000  96543.090000
75%    101370.970000 100955.350000  99064.495000  97036.300000
max    104751.350000 102892.110000 102275.480000 101876.140000

                qW   Total_Power
count  36043.000000  3.604300e+04
mean       0.833849  3.938246e+06
std        0.026052  1.226171e+05
min        0.720000  3.388944e+06
25%        0.810000  3.847335e+06
50%        0.830000  3.931541e+06
75%        0.860000  4.063623e+06
max        0.880000  4.177659e+06

[8 rows x 149 columns]
```

[16]:
```
X = df.drop(["Total_Power"], axis=1)
y = df["Total_Power"]
```

0.2 Principal Component Analysis (PCA)

[17]:
```
# drop all XY columns
X = X.drop(X.columns[X.columns.str.contains("X")], axis=1)
X = X.drop(X.columns[X.columns.str.contains("Y")], axis=1)

X.head()
```

[17]:

	Power1	Power2	Power3	Power4	Power5	Power6	Power7	\
0	71265.25	77995.25	72872.99	69061.17	70271.92	70133.17	70275.04	
1	72871.68	76893.17	72604.11	68857.32	74134.45	70271.21	70233.26	
2	72724.29	76995.80	72612.33	68855.75	72698.52	71859.26	70298.29	
3	72759.25	77036.33	72717.21	68656.01	72735.03	71842.15	70158.08	
4	44620.44	45945.24	47067.08	48278.17	56778.37	55045.52	54072.63	

	Power8	Power9	Power10	…	Power41	Power42	Power43	Power44	\
0	72878.46	75002.15	77099.61	…	77097.57	88867.92	98844.30	101283.59	
1	72970.56	74838.49	77055.08	…	77044.77	88896.55	98759.79	101346.07	
2	72987.39	74842.03	77062.81	…	77038.66	88919.83	98746.68	101346.15	
3	73220.73	75117.22	76983.41	…	77057.86	88855.14	98760.96	101338.59	
4	53794.42	73417.62	79860.68	…	76869.43	88005.30	98630.24	100432.73	

	Power45	Power46	Power47	Power48	Power49	qW
0	98934.63	101624.58	100915.03	99625.68	96704.34	0.87
1	98873.59	101629.01	100934.53	99606.13	96718.39	0.87
2	98875.57	101618.32	100941.00	99611.35	96719.14	0.87
3	98971.58	101632.28	100943.59	99589.25	96735.04	0.87
4	98803.01	101064.48	100948.38	99028.87	96286.71	0.79

[5 rows x 50 columns]

[18]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

scaler = StandardScaler()
features_scaled: np.ndarray = scaler.fit_transform(X)

pca: PCA = PCA(random_state=42)
pca_result: np.ndarray = pca.fit_transform(features_scaled)

cumulative_variance_ratio: np.ndarray = np.cumsum(pca.explained_variance_ratio_)
n_components_95: int = np.argmax(cumulative_variance_ratio >= 0.95) + 1

df_pca: pd.DataFrame = pd.DataFrame(
    pca_result[:, :n_components_95],
    columns=[f"PC{i+1}" for i in range(n_components_95)],
)
```

```
df_pca["Total_Power"] = y
```

[19]:
```
import plotly.express as px
import plotly.graph_objects as go

fig_variance = px.line(
    x=range(1, len(cumulative_variance_ratio) + 1),
    y=cumulative_variance_ratio,
    labels={
        "x": "Number of Principal Components",
        "y": "Cumulative Explained Variance Ratio",
    },
    title="Cumulative Explained Variance Ratio",
)

fig_variance.add_hline(
    y=0.95,
    line_dash="dash",
    annotation_text="95% Explained Variance",
    annotation_position="bottom right",
)

fig_variance.add_vline(
    x=n_components_95,
    line_dash="dash",
    line_color="green",
    annotation_text=f"{n_components_95} components",
)

fig_variance.show()
```

[20]:
```
fig_pca = px.scatter(
    df_pca,
    x="PC1",
    y="PC2",
    color="Total_Power",
    labels={"color": "Total Power"},
    title="Principal Components Analysis",
)

fig_pca.show()
```

[21]:
```
correlation_matrix: pd.DataFrame = pd.DataFrame(
    pca.components_[:n_components_95, :].T,
    columns=[f"PC{i+1}" for i in range(n_components_95)],
    index=X.columns,
```

```
)

correlation_matrix = correlation_matrix.abs()
correlation_matrix
```

[21]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	\
Power1	0.227347	0.017336	0.027763	0.132492	0.162870	0.079611	0.180954	
Power2	0.204388	0.011084	0.015684	0.127541	0.176283	0.095099	0.240230	
Power3	0.081953	0.075318	0.129046	0.102494	0.231381	0.076801	0.311038	
Power4	0.063604	0.061057	0.072869	0.038584	0.227772	0.052106	0.272914	
Power5	0.072763	0.187117	0.113411	0.002981	0.121768	0.004665	0.026071	
Power6	0.027034	0.249355	0.144408	0.017091	0.062619	0.055337	0.104640	
Power7	0.122589	0.266072	0.237591	0.058025	0.001324	0.087219	0.197753	
Power8	0.218565	0.085514	0.040763	0.076742	0.212975	0.015843	0.049776	
Power9	0.193103	0.178380	0.136795	0.073304	0.200188	0.100830	0.015405	
Power10	0.213358	0.144259	0.141029	0.047198	0.171727	0.127271	0.013608	
Power11	0.174958	0.111400	0.160833	0.059581	0.171864	0.111314	0.154073	
Power12	0.112910	0.064098	0.089124	0.011932	0.242440	0.109305	0.154921	
Power13	0.105105	0.097429	0.132868	0.146317	0.159007	0.288660	0.013339	
Power14	0.161798	0.178991	0.205109	0.074438	0.106923	0.168471	0.095831	
Power15	0.122368	0.033217	0.008460	0.191224	0.266333	0.296097	0.069623	
Power16	0.085956	0.024849	0.051975	0.171145	0.255439	0.245354	0.020314	
Power17	0.208955	0.072050	0.018597	0.259538	0.172133	0.057765	0.064898	
Power18	0.207575	0.106606	0.011227	0.249297	0.235276	0.072278	0.027695	
Power19	0.121424	0.058287	0.021081	0.276774	0.244173	0.125560	0.076902	
Power20	0.000494	0.005350	0.094090	0.300635	0.187008	0.168624	0.043160	
Power21	0.134331	0.041182	0.268916	0.116346	0.183448	0.037770	0.151425	
Power22	0.070338	0.264603	0.085422	0.248437	0.048003	0.125139	0.009500	
Power23	0.018541	0.204931	0.152420	0.263291	0.123314	0.051175	0.172285	
Power24	0.006608	0.236156	0.096299	0.263324	0.042634	0.048062	0.098868	
Power25	0.120833	0.226125	0.226345	0.119236	0.020281	0.088149	0.158897	
Power26	0.128454	0.250847	0.247344	0.123603	0.065548	0.071223	0.118318	
Power27	0.061982	0.313207	0.190410	0.087043	0.114948	0.084285	0.012196	
Power28	0.065871	0.332405	0.055181	0.032023	0.150860	0.116336	0.063946	
Power29	0.005035	0.125663	0.001485	0.206794	0.058755	0.273016	0.002828	
Power30	0.057157	0.081367	0.008710	0.279534	0.133632	0.288415	0.099418	
Power31	0.082415	0.050960	0.043440	0.262860	0.165766	0.265160	0.203206	
Power32	0.071173	0.010459	0.058270	0.234732	0.150464	0.189689	0.178630	
Power33	0.101711	0.058259	0.204562	0.017059	0.081189	0.220016	0.057674	
Power34	0.119857	0.055971	0.320018	0.003285	0.065313	0.202528	0.018577	
Power35	0.032575	0.030344	0.363794	0.041787	0.070488	0.194344	0.034494	
Power36	0.142775	0.012176	0.262219	0.002402	0.012392	0.128267	0.166860	
Power37	0.029421	0.163468	0.068373	0.003339	0.169324	0.134297	0.159665	
Power38	0.164504	0.142610	0.007204	0.037872	0.122413	0.166559	0.079870	
Power39	0.207834	0.079884	0.101382	0.065499	0.079965	0.046849	0.050894	
Power40	0.193978	0.059741	0.135845	0.082534	0.051525	0.023875	0.019199	
Power41	0.002780	0.120219	0.062803	0.116568	0.062588	0.062247	0.308255	

Power42	0.112902	0.140905	0.021379	0.094649	0.099571	0.115727	0.134184
Power43	0.174992	0.104493	0.075757	0.052260	0.036401	0.093607	0.172411
Power44	0.112841	0.096934	0.201341	0.036993	0.086144	0.035587	0.078575
Power45	0.154731	0.025343	0.086579	0.008848	0.005101	0.096234	0.032941
Power46	0.202668	0.020142	0.070133	0.025438	0.029321	0.137958	0.259210
Power47	0.225719	0.012479	0.097804	0.012169	0.034236	0.112184	0.141839
Power48	0.131607	0.057061	0.022571	0.004484	0.000791	0.138420	0.246212
Power49	0.074664	0.180446	0.026383	0.042914	0.063491	0.018026	0.069699
qW	0.334687	0.094894	0.136524	0.054207	0.008752	0.051611	0.224010

	PC8	PC9	PC10	…	PC30	PC31	PC32	\
Power1	0.257010	0.063239	0.048413	…	0.052142	0.019644	0.047825	
Power2	0.212724	0.129802	0.077417	…	0.068552	0.078383	0.038868	
Power3	0.162731	0.137474	0.003015	…	0.191476	0.152822	0.039124	
Power4	0.097365	0.183676	0.095943	…	0.185786	0.064526	0.117848	
Power5	0.206754	0.338698	0.171571	…	0.101821	0.433895	0.047998	
Power6	0.116176	0.313893	0.113713	…	0.069654	0.188579	0.077614	
Power7	0.013045	0.072181	0.014769	…	0.077835	0.090709	0.145118	
Power8	0.195214	0.247799	0.007789	…	0.211751	0.313047	0.187840	
Power9	0.000749	0.110370	0.001297	…	0.063055	0.052854	0.062212	
Power10	0.015620	0.112313	0.006864	…	0.070699	0.061260	0.140616	
Power11	0.007083	0.022212	0.051962	…	0.110425	0.028775	0.222841	
Power12	0.085983	0.014849	0.096927	…	0.021596	0.028461	0.071357	
Power13	0.095498	0.267732	0.054063	…	0.242444	0.187388	0.218503	
Power14	0.132479	0.277573	0.058666	…	0.080835	0.033160	0.092109	
Power15	0.042605	0.133733	0.069832	…	0.133266	0.093730	0.172930	
Power16	0.075570	0.083771	0.095782	…	0.385705	0.120389	0.050647	
Power17	0.262679	0.103659	0.004108	…	0.183567	0.017884	0.135385	
Power18	0.270271	0.091372	0.005984	…	0.299313	0.024145	0.051191	
Power19	0.325115	0.026761	0.039421	…	0.038644	0.053237	0.117467	
Power20	0.128761	0.011280	0.170821	…	0.213990	0.025963	0.082446	
Power21	0.149872	0.094133	0.160659	…	0.211560	0.141535	0.149860	
Power22	0.058197	0.194916	0.127610	…	0.182503	0.165837	0.133196	
Power23	0.198708	0.160825	0.034539	…	0.115091	0.004503	0.240023	
Power24	0.140915	0.209411	0.047145	…	0.183335	0.347926	0.016822	
Power25	0.039859	0.099524	0.022715	…	0.145844	0.012080	0.174113	
Power26	0.087659	0.092509	0.040838	…	0.063427	0.060076	0.059443	
Power27	0.192816	0.106908	0.015546	…	0.075429	0.000875	0.014541	
Power28	0.201008	0.097484	0.030144	…	0.167236	0.005049	0.216827	
Power29	0.052119	0.061716	0.269418	…	0.024971	0.039883	0.026897	
Power30	0.040241	0.139411	0.199842	…	0.117558	0.107732	0.055962	
Power31	0.102728	0.080690	0.057027	…	0.076674	0.161091	0.022489	
Power32	0.181431	0.096091	0.059385	…	0.049315	0.052725	0.023405	
Power33	0.075758	0.123802	0.005041	…	0.026963	0.009781	0.088527	
Power34	0.038318	0.231516	0.031267	…	0.063319	0.025055	0.120017	
Power35	0.158235	0.209331	0.011430	…	0.006972	0.244578	0.097458	
Power36	0.269427	0.133081	0.021152	…	0.087948	0.119969	0.022170	

Power37	0.017319	0.111552	0.135233	…	0.282337	0.055221	0.311740
Power38	0.166147	0.066427	0.190353	…	0.099622	0.203665	0.486301
Power39	0.133530	0.061264	0.283440	…	0.048138	0.129557	0.137288
Power40	0.176381	0.063416	0.346249	…	0.062521	0.091334	0.011199
Power41	0.026275	0.060571	0.403028	…	0.122034	0.066754	0.154223
Power42	0.054972	0.123443	0.394613	…	0.044282	0.009251	0.174760
Power43	0.087901	0.105254	0.202303	…	0.128354	0.021740	0.169108
Power44	0.099904	0.163374	0.031718	…	0.033228	0.195488	0.035145
Power45	0.074482	0.017514	0.049153	…	0.032767	0.106449	0.009731
Power46	0.093195	0.049528	0.208694	…	0.026694	0.107776	0.055078
Power47	0.006642	0.038918	0.032837	…	0.259261	0.363282	0.166295
Power48	0.020770	0.095293	0.194441	…	0.076657	0.026342	0.150667
Power49	0.036029	0.066090	0.077573	…	0.066994	0.073776	0.029687
qW	0.114193	0.056015	0.032020	…	0.049289	0.024877	0.001554

	PC33	PC34	PC35	PC36	PC37	PC38	PC39
Power1	0.016848	0.215111	0.018091	0.123045	0.038788	0.050289	0.195182
Power2	0.018862	0.137985	0.151256	0.044078	0.118065	0.052743	0.089796
Power3	0.060433	0.481414	0.203428	0.145124	0.056675	0.075012	0.027594
Power4	0.034281	0.105016	0.085659	0.009350	0.008272	0.030375	0.059780
Power5	0.190386	0.111176	0.052621	0.025635	0.110483	0.053868	0.063456
Power6	0.157596	0.101997	0.265855	0.074612	0.259879	0.161803	0.093388
Power7	0.072543	0.101527	0.303669	0.149753	0.079964	0.087608	0.261539
Power8	0.100969	0.240509	0.132339	0.056817	0.397912	0.093769	0.053410
Power9	0.333595	0.011581	0.143807	0.004969	0.136942	0.180471	0.257191
Power10	0.015197	0.000600	0.163574	0.092565	0.069952	0.027406	0.347160
Power11	0.142259	0.119374	0.024481	0.154988	0.252588	0.412203	0.136285
Power12	0.113708	0.087853	0.004803	0.094607	0.054447	0.299941	0.031842
Power13	0.076232	0.085221	0.031071	0.103028	0.019043	0.010133	0.185511
Power14	0.097608	0.022319	0.037375	0.128822	0.197765	0.289130	0.019860
Power15	0.104194	0.218023	0.049406	0.482433	0.018897	0.234707	0.244332
Power16	0.162389	0.167525	0.052222	0.304020	0.154831	0.002759	0.109416
Power17	0.316304	0.024557	0.154105	0.145861	0.194014	0.216447	0.066585
Power18	0.047940	0.070518	0.057448	0.071691	0.046567	0.024443	0.051389
Power19	0.276110	0.105809	0.246504	0.048822	0.309386	0.198309	0.195719
Power20	0.081026	0.041784	0.164385	0.033928	0.209264	0.046890	0.143963
Power21	0.088163	0.138501	0.066596	0.063538	0.111852	0.081151	0.155831
Power22	0.177149	0.112070	0.016154	0.062467	0.107649	0.085902	0.094020
Power23	0.132026	0.058751	0.086818	0.131968	0.043137	0.015655	0.208735
Power24	0.196402	0.011638	0.001710	0.111536	0.022533	0.087943	0.262768
Power25	0.015035	0.023646	0.080263	0.156496	0.226951	0.074284	0.048140
Power26	0.058387	0.055002	0.041314	0.105261	0.027437	0.185836	0.010046
Power27	0.042430	0.046674	0.042748	0.155246	0.009565	0.190257	0.065385
Power28	0.118764	0.034079	0.138387	0.204614	0.209948	0.041111	0.045132
Power29	0.288749	0.137904	0.106097	0.082982	0.276294	0.101672	0.108554
Power30	0.121916	0.077284	0.253424	0.122634	0.344263	0.209706	0.281801
Power31	0.290211	0.032870	0.396219	0.096744	0.053016	0.107405	0.264224

```
Power32  0.198067  0.072717  0.301900  0.075298  0.085309  0.015102  0.048986
Power33  0.035428  0.155474  0.089696  0.180898  0.149353  0.104873  0.130713
Power34  0.018538  0.106640  0.068575  0.197796  0.070676  0.293200  0.277907
Power35  0.150390  0.049267  0.055605  0.017440  0.045161  0.327576  0.141864
Power36  0.223008  0.120181  0.145543  0.013519  0.065452  0.093552  0.070793
Power37  0.053837  0.059307  0.162590  0.051899  0.103161  0.031828  0.078888
Power38  0.065163  0.123511  0.241035  0.072664  0.023316  0.096257  0.118674
Power39  0.141333  0.189348  0.039791  0.206792  0.025734  0.087953  0.050220
Power40  0.029675  0.029943  0.157965  0.138119  0.057777  0.034608  0.132282
Power41  0.157710  0.282256  0.073342  0.229564  0.038932  0.037200  0.033987
Power42  0.227392  0.289217  0.064951  0.230545  0.044474  0.006165  0.068287
Power43  0.066426  0.164391  0.040426  0.147245  0.054256  0.028989  0.017177
Power44  0.001356  0.069968  0.154757  0.028962  0.079282  0.127225  0.007897
Power45  0.046344  0.039122  0.016802  0.164559  0.031194  0.024536  0.037469
Power46  0.007718  0.277700  0.006881  0.154053  0.078827  0.029714  0.029089
Power47  0.102472  0.144255  0.121047  0.035566  0.051884  0.023021  0.002100
Power48  0.008918  0.026019  0.104124  0.075722  0.049169  0.049112  0.013144
Power49  0.048925  0.001827  0.014090  0.009810  0.027272  0.022002  0.019713
qW       0.032842  0.009154  0.014111  0.035810  0.029207  0.008430  0.002990

[50 rows x 39 columns]
```

[22]:
```python
top_features: dict = {}
for pc in correlation_matrix.columns:
    top_features[pc] = correlation_matrix[pc].nlargest(10).index.tolist()

print("Top 10 Features for each principal component")
for pc, features in top_features.items():
    print(f"Principal Component {pc}: {features}")
```

```
Top 10 Features for each principal component
Principal Component PC1: ['qW', 'Power1', 'Power47', 'Power8', 'Power10',
'Power17', 'Power39', 'Power18', 'Power2', 'Power46']
Principal Component PC2: ['Power28', 'Power27', 'Power7', 'Power22', 'Power26',
'Power6', 'Power24', 'Power25', 'Power23', 'Power5']
Principal Component PC3: ['Power35', 'Power34', 'Power21', 'Power36', 'Power26',
'Power7', 'Power25', 'Power14', 'Power33', 'Power44']
Principal Component PC4: ['Power20', 'Power30', 'Power19', 'Power24', 'Power23',
'Power31', 'Power17', 'Power18', 'Power22', 'Power32']
Principal Component PC5: ['Power15', 'Power16', 'Power19', 'Power12', 'Power18',
'Power3', 'Power4', 'Power8', 'Power9', 'Power20']
Principal Component PC6: ['Power15', 'Power13', 'Power30', 'Power29', 'Power31',
'Power16', 'Power33', 'Power34', 'Power35', 'Power32']
Principal Component PC7: ['Power3', 'Power41', 'Power4', 'Power46', 'Power48',
'Power2', 'qW', 'Power31', 'Power7', 'Power1']
Principal Component PC8: ['Power19', 'Power18', 'Power36', 'Power17', 'Power1',
'Power2', 'Power5', 'Power28', 'Power23', 'Power8']
```

```
Principal Component PC9: ['Power5', 'Power6', 'Power14', 'Power13', 'Power8',
'Power34', 'Power24', 'Power35', 'Power22', 'Power4']
Principal Component PC10: ['Power41', 'Power42', 'Power40', 'Power39',
'Power29', 'Power46', 'Power43', 'Power30', 'Power48', 'Power38']
Principal Component PC11: ['Power16', 'Power48', 'Power47', 'Power2', 'Power1',
'Power11', 'Power33', 'Power44', 'Power3', 'Power43']
Principal Component PC12: ['Power38', 'Power37', 'Power26', 'Power25',
'Power32', 'Power31', 'Power27', 'Power11', 'Power24', 'Power23']
Principal Component PC13: ['Power12', 'Power33', 'Power11', 'Power34',
'Power37', 'Power5', 'Power10', 'Power9', 'Power14', 'Power13']
Principal Component PC14: ['Power37', 'Power39', 'Power42', 'Power45',
'Power40', 'Power4', 'Power16', 'Power38', 'Power47', 'Power10']
Principal Component PC15: ['Power44', 'Power49', 'Power45', 'Power6', 'Power36',
'Power35', 'Power39', 'Power29', 'Power46', 'Power10']
Principal Component PC16: ['Power45', 'Power32', 'Power29', 'Power41',
'Power49', 'Power30', 'Power33', 'Power31', 'Power37', 'Power42']
Principal Component PC17: ['Power49', 'Power29', 'Power37', 'Power48',
'Power43', 'Power9', 'Power32', 'Power31', 'Power22', 'Power40']
Principal Component PC18: ['Power48', 'Power12', 'Power4', 'Power44', 'Power20',
'Power10', 'Power9', 'Power33', 'Power46', 'Power3']
Principal Component PC19: ['Power4', 'Power43', 'Power49', 'Power48', 'Power39',
'Power45', 'Power16', 'Power44', 'Power47', 'Power20']
Principal Component PC20: ['Power45', 'Power32', 'Power33', 'Power43',
'Power29', 'Power12', 'Power48', 'Power39', 'Power36', 'Power46']
Principal Component PC21: ['Power44', 'Power33', 'Power36', 'Power4', 'Power45',
'Power17', 'Power48', 'Power46', 'Power29', 'Power3']
Principal Component PC22: ['Power4', 'Power47', 'Power9', 'Power6', 'Power11',
'Power13', 'Power30', 'Power16', 'Power43', 'Power21']
Principal Component PC23: ['Power12', 'Power48', 'Power49', 'Power41', 'Power2',
'Power5', 'Power4', 'Power9', 'Power15', 'Power19']
Principal Component PC24: ['Power12', 'Power40', 'Power22', 'Power41',
'Power44', 'Power29', 'Power49', 'Power37', 'Power11', 'Power43']
Principal Component PC25: ['Power3', 'Power25', 'Power2', 'Power42', 'Power47',
'Power13', 'Power21', 'Power37', 'Power1', 'Power15']
Principal Component PC26: ['Power20', 'Power43', 'Power16', 'Power25',
'Power46', 'Power28', 'Power44', 'Power7', 'Power24', 'Power37']
Principal Component PC27: ['Power43', 'Power46', 'Power47', 'Power20', 'Power5',
'Power40', 'Power37', 'Power4', 'Power25', 'Power38']
Principal Component PC28: ['Power40', 'Power42', 'Power44', 'Power39',
'Power38', 'Power4', 'Power43', 'Power3', 'Power25', 'Power16']
Principal Component PC29: ['Power46', 'Power21', 'Power5', 'Power6', 'Power48',
'Power23', 'Power30', 'Power44', 'Power38', 'Power40']
Principal Component PC30: ['Power16', 'Power18', 'Power37', 'Power47',
'Power13', 'Power20', 'Power8', 'Power21', 'Power3', 'Power4']
Principal Component PC31: ['Power5', 'Power47', 'Power24', 'Power8', 'Power35',
'Power38', 'Power44', 'Power6', 'Power13', 'Power22']
Principal Component PC32: ['Power38', 'Power37', 'Power23', 'Power11',
'Power13', 'Power28', 'Power8', 'Power42', 'Power25', 'Power15']
```

```
Principal Component PC33: ['Power9', 'Power17', 'Power31', 'Power29', 'Power19',
'Power42', 'Power36', 'Power32', 'Power24', 'Power5']
Principal Component PC34: ['Power3', 'Power42', 'Power41', 'Power46', 'Power8',
'Power15', 'Power1', 'Power39', 'Power16', 'Power43']
Principal Component PC35: ['Power31', 'Power7', 'Power32', 'Power6', 'Power30',
'Power19', 'Power38', 'Power3', 'Power20', 'Power10']
Principal Component PC36: ['Power15', 'Power16', 'Power42', 'Power41',
'Power39', 'Power28', 'Power34', 'Power33', 'Power45', 'Power25']
Principal Component PC37: ['Power8', 'Power30', 'Power19', 'Power29', 'Power6',
'Power11', 'Power25', 'Power28', 'Power20', 'Power14']
Principal Component PC38: ['Power11', 'Power35', 'Power12', 'Power34',
'Power14', 'Power15', 'Power17', 'Power30', 'Power19', 'Power27']
Principal Component PC39: ['Power10', 'Power30', 'Power34', 'Power31',
'Power24', 'Power7', 'Power9', 'Power15', 'Power23', 'Power19']
```

[23]:
```
feature_importance: pd.DataFrame = pd.DataFrame(
    {
        "feature": X.columns,
        "importance": np.sum(np.abs(pca.components_[:n_components_95, :]),␣
 ↪axis=0),
    }
)

feature_importance = feature_importance.sort_values("importance",␣
 ↪ascending=False)

fig_importance = px.bar(
    feature_importance.head(20),
    x="feature",
    y="importance",
    labels={"importance": "Importance Score"},
    title="Top 20 Features Importance Based on PCA Loadings",
)

fig_importance.show()
```

[24]:
```
print(f"Number of components explaining 95% of variance: {n_components_95}")
print(f"Total number of features: {len(X.columns)}")
```

```
Number of components explaining 95% of variance: 39
Total number of features: 50
```

[25]:
```
fig_3d = go.Figure(
    data=[
        go.Scatter3d(
            x=df_pca["PC1"],
            y=df_pca["PC2"],
```

```
            z=df_pca["Total_Power"],
            mode="markers",
            marker=dict(
                size=3,
                color=df_pca["Total_Power"],
                colorscale="Viridis",
                opacity=0.8,
                colorbar=dict(title="Total Power"),
            ),
            text=df_pca["Total_Power"],
            hoverinfo="text",
        )
    ]
)

fig_3d.update_layout(
    title="3D PCA Scatter Plot",
    scene=dict(xaxis_title="PC1", yaxis_title="PC2", zaxis_title="Total_Power"),
    width=800,
    height=600,
)

fig_3d.show()
```

[26]:
```
X_pca = df_pca.drop(["Total_Power"], axis=1)
y_pca = df_pca["Total_Power"]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y_pca, test_size=0.2, random_state=42
)

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

y_pred = lr_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
print(f"Intercept: {lr_model.intercept_}")
```

```
Mean Squared Error: 17560763.604120795
R^2 Score: 0.9988084940223867
Intercept: 3938248.3609718205
```

[27]:
```
fig_regression = px.scatter(
    x=y_test,
    y=y_pred,
    labels={"x": "Actual Total Power", "y": "Predicted Total Power"},
    title="Actual vs Predicted Total Power",
)

fig_regression.add_trace(
    go.Scatter(
        x=[y.min(), y.max()],
        y=[y.min(), y.max()],
        mode="lines",
        name="Perfect Prediction",
        line=dict(color="red", dash="dash"),
    )
)

fig_regression.show()
```

[28]:
```
feature_importance_lr = pd.DataFrame(
    {"feature": X_pca.columns, "importance": np.abs(lr_model.coef_)}
)

feature_importance_lr = feature_importance_lr.sort_values("importance",␣
 ↪ascending=False)

fig_importance_lr = px.bar(
    feature_importance_lr.head(10),
    x="feature",
    y="importance",
    labels={"importance": "Absolute Coefficient Value"},
    title="Top 10 Important Principal Components in Linear Regression",
)

fig_importance_lr.show()
```

[29]:
```
print("\nTop 5 Important Principal Components in Linear Regression:")
print(feature_importance_lr.head())
```

```
Top 5 Important Principal Components in Linear Regression:
   feature    importance
0      PC1  41267.909115
6      PC7  27144.563317
```

```
2      PC3  16369.408738
12    PC13  16072.631583
11    PC12  15309.717843
```

[30]:
```
residuals = y_test - y_pred
fig_residuals = px.scatter(
    x=y_pred,
    y=residuals,
    labels={"x": "Predicted Total Power", "y": "Residuals"},
    title="Residual Plot",
)
fig_residuals.add_hline(y=0, line_dash="dash", line_color="red")
fig_residuals.show()
```

[31]:
```
print(f"\nIntercept: {lr_model.intercept_:.4f}")
```

```
Intercept: 3938248.3610
```

[32]:
```
fig_variance.write_image("cumulative_variance_plot.png")
fig_importance.write_image("feature_importance_plot.png")
fig_pca.write_image("pca_scatter_plot.png")
fig_3d.write_image("pca_3d_plot.png")
```

0.3 Linear Regression for Multiple Variables

[33]:
```
from numba import jit
from typing import Tuple


@jit(nopython=True)
def mse(y_true: np.ndarray, y_pred: np.ndarray) -> float:
    """
    Calculates the mean squared error between the true and predicted values
    of a dataset.

    Args:
        y_true (np.ndarray): The true values of the dataset
        y_pred (np.ndarray): The predicted values of the dataset

    Returns:
        float: The mean squared error between the true and predicted values
    """

    return (1 / len(y_true)) * np.sum((y_true - y_pred) ** 2)
```

```python
@jit(nopython=True)
def r2(y_true: np.ndarray, y_pred: np.ndarray) -> np.float64:
    """
    Calculates the R^2 score between the true and predicted values of a dataset.

    Args:
        y_true (np.ndarray): The true values of the dataset
        y_pred (np.ndarray): The predicted values of the dataset

    Returns:
        float: The R^2 score between the true and predicted values
    """
    ss_res = np.sum((y_true - y_pred) ** 2)
    ss_tot = np.sum((y_true - np.mean(y_true)) ** 2)

    return 1 - (ss_res / ss_tot)


@jit(nopython=True)
def gradient_descent(
    X: np.ndarray, y: np.ndarray, lr: float, epochs: int
) -> Tuple[np.ndarray, np.ndarray]:
    """
    Calculates the optimal weights for a linear regression model using
    gradient descent algorithm.

    Uses the mean squared error as the loss function to minimize.

    Args:
        X (np.ndarray): The input features of the dataset
        y (np.ndarray): The true values of the dataset
        lr (float): The learning rate of the algorithm
        epochs (int): The number of iterations to run the algorithm

    Returns:
        Tuple[np.ndarray, np.ndarray]: The optimal weights of the model and the␣
 ↪loss  at each epoch
    """

    m, n = X.shape
    theta: np.ndarray = np.zeros(n)
    loss: np.ndarray = np.zeros(epochs)

    for i in range(epochs):
        y_pred = X.dot(theta)
        loss[i] = mse(y, y_pred)
```

```
        gradient: np.ndarray = (1 / m) * X.T.dot((y_pred - y))
        theta -= lr * gradient

    return theta, loss
```

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y_pca, test_size=0.2, random_state=42, shuffle=True
)

# bias term
X_train_gd = np.c_[np.ones(len(X_train)), X_train]
y_train_gd = y_train.values

alpha: float = 0.003
epochs: int = 2000

theta, loss = gradient_descent(X_train_gd, y_train_gd, alpha, epochs)
```

[35]:
```
y_pred_gd = X_test.dot(theta[1:]) + theta[0]

mse_gd = mse(y_test.to_numpy(), y_pred_gd.to_numpy())
r2_gd = r2(y_test.to_numpy(), y_pred_gd.to_numpy())

print(f"Mean Squared Error (Gradient Descent): {mse_gd}")
print(f"R^2 Score (Gradient Descent): {r2_gd}")
```

```
Mean Squared Error (Gradient Descent): 121567120.01424004
R^2 Score (Gradient Descent): 0.9917516143691946
```

[36]:
```
# Plot the loss curve
fig_loss = px.line(
    x=range(epochs),
    y=loss,
    labels={"x": "Epochs", "y": "Loss"},
    title="Gradient Descent Loss Curve",
)

fig_loss.show()
```

[37]:
```
fig_regression_gd = px.scatter(
    x=y_test,
    y=y_pred_gd,
    labels={"x": "Actual Total Power", "y": "Predicted Total Power"},
    title="Actual vs Predicted Total Power (Gradient Descent)",
)

fig_regression_gd.add_trace(
```

```
    go.Scatter(
        x=[y.min(), y.max()],
        y=[y.min(), y.max()],
        mode="lines",
        name="Perfect Prediction",
        line=dict(color="red", dash="dash"),
    )
)

fig_regression_gd.show()
```

[38]:
```
# residual plot
residuals_gd = y_test - y_pred_gd

fig_residuals_gd = px.scatter(
    x=y_pred_gd,
    y=residuals_gd,
    labels={"x": "Predicted Total Power", "y": "Residuals"},
    title="Residual Plot (Gradient Descent)",
)

fig_residuals_gd.add_hline(y=0, line_dash="dash", line_color="red")
fig_residuals_gd.show()
```